ConfiDNS: Leveraging Scale and History to Detect Compromise

Lindsey Poole, Vivek S. Pai
Princeton University

Abstract

While cooperative DNS resolver systems, such as CoDNS, have demonstrated improved reliability and performance over standard approaches, their security has been weaker, since any corruption or misbehavior of a single resolver can easily propagate throughout the system.

We address this weakness in a new system called ConfiDNS, which augments the cooperative lookup process with configurable policies that utilize multi-site agreement and per-site lookup histories. Not only does ConfiDNS provide better security than cooperative approaches, but for up to 99.8% of unique lookups, ConfiDNS exceeds the security of standard DNS resolvers. ConfiDNS provides these benefits while retaining the other benefits of CoDNS, such as incremental deployability, higher reliability, and improved performance, in some cases faster than CoDNS.

1 Introduction

The use of distributed computing to address performance and reliability problems in the Domain Name System (DNS) [18] has recently received much research attention, and has spawned two widely-deployed distributed systems, CoDNS [19] and CoDoNS [22]. Both of these systems provide clients with improved reliability when performing DNS lookups by distributing the queries across nodes in the system. These systems fetch name-to-IP translations from the existing legacy DNS infrastructure as needed to provide an upgrade path for users.

Due to their interaction with vulnerable legacy DNS infrastructure, these systems can be less secure than traditional local DNS resolvers, even if these DNS replacements are written securely and use secure inter-node communication. If any node performs a DNS resolution and receives an incorrect answer, that answer can be propagated to other nodes. The incorrect answer can occur because of a failure or compromise at a local DNS resolver, or from UDP packet spoofing when the node tries to communicate with an external DNS server. Ironically, the use of aggressive caching and multi-hop request forwarding to improve resilience under flooding attacks can actually cause polluted responses to be kept longer and spread further, magnifying the scope of the problem when compared to traditional DNS configurations.

Rather than being a fundamental trade-off in trying to support legacy DNS while achieving better reliability, we show that worse security is not an unavoidable by-product of cooperative DNS systems. For most DNS deployment scenarios, the greater scale of cooperative DNS systems can be leveraged to provide better security than legacy DNS resolvers for the vast majority of queries. Where scale cannot be used, observing the history of DNS queries can provide some assurance that DNS replies have not been modified. Between these two options, only a small fraction of DNS queries need to trade security for reliability or performance.

In this paper, we present ConfiDNS, a cooperative DNS system that provides improved DNS reliability while allowing customizable security policies. These policies can be tailored by administrators, on a per-domain basis, allowing for very strict security at important sites (e.g. banks and online payment sites), while allowing lower-overhead policies for casual browsing. We also make the following contributions:

- We analyze a running cooperative global DNS system, CoDNS, to understand what kind of traffic these resolvers experience. We find that DNS traffic changed qualitatively in the recent past, and that cooperative DNS systems exhibit traffic patterns unlike that reported for individual DNS resolvers.
- We continuously monitor domains around the world from multiple vantage points, allowing us to reverse-engineer routing decisions for their content distribution networks (CDNs) and data centers. We also observe how the DNS mappings used by these domains change over time.
- We present a range of security policies for ConfiDNS, show what fraction of domains can be handled by each policy, and show the performance and traffic overhead of each policy.

Even in its weakest configuration, ConfiDNS provides better security than local DNS for 99.8% of queries, while stronger security requirements can be met in over 99.2% of queries. On a practical level, ConfiDNS is incrementally deployable, and requires no change to the existing global DNS infrastructure to reap its benefits. At the same time, it also provides benefits even if the DNS infrastructure changes to support authentication, using schemes such as DNSSEC [6, 4, 5].

The rest of this paper is organized as follows: in Section 2, we describe ConfiDNS, describe how it operates, and what kinds of protection it provides. We then describe

the workload observed on an existing cooperative DNS system, CoDNS, in Section 3, and the studies of the DNS hierarchy performed using the CoDNS workload in Section 4. We discuss our implementation in Section 5, and evaluate some sample ConfiDNS policies in Section 6. We then discuss related work in Section 7 and conclude.

2 Overview

Before describing the workings of ConfiDNS, we describe the Domain Name System (DNS) and the terminology we use for its components. DNS maps human-readable machine names to numeric IP addresses using a globally distributed hierarchy of servers, each of which is responsible for a portion of the global namespace. This system, which we call the server-side DNS infrastructure, is operated by the owners of domain names (e.g. example.com) and their surrogates, and by organizations that are responsible for the top-level servers (e.g. com) that point to the per-domain servers. Clients rarely query these machines directly, but instead send DNS lookups to machines within their own organization, called local (client-side) DNS resolvers. These resolvers perform the queries and cache the results, sharing lookup overhead across many clients. The CoDNS system observed that many DNS problems were due to failures at the local DNS resolvers. CoDNS achieves better performance and reliability by brokering queries to peer DNS resolvers at remote sites when the local resolvers are failing, since resolver failures at different sites were largely uncorrelated.

ConfiDNS attempts to increase confidence in DNS lookups by using peer sites at *all* times in order to provide protection against certain attacks and failures. Additionally, ConfiDNS also uses lookup history to detect changes in name-to-IP mappings. The basic idea is simple – users run a ConfiDNS agent which contains configurable DNS lookup policies. This agent is ideally run on the user's own machine, but can be run on a (possibly shared) machine near the user (with some increased risk). This agent receives DNS lookup requests from the user, and sends the request to both the local DNS resolver as well as some number of peer ConfiDNS agents located at remote sites.

Using the response from the local DNS resolver, the peer ConfiDNS agents, and the agent's recorded history of previous lookups for the name, the agent provides the client with a response, or indicates a failure if no response could be provided that met the specified security policy. Response lookups are also logged to determine lookup history. Sample policies for ConfiDNS include the following: (a) the local resolver and at least one peer must agree on the result, (b) at least three sites must agree, (c) if no peers agree with the local resolver, the IP address must not have changed in the past week, (d) if no peers agree within 5 seconds, use any result. Some questions that naturally arise are

1. What attacks or failures do these policies handle?
2. What sites are amenable to various policies, and what kinds of overheads are incurred in using ConfiDNS?
3. How do these policies interact with content distribution networks or load-balancing schemes, which route traffic using active DNS-to-IP mappings?

2.1 Threat Model & Attacks Handled

ConfiDNS is designed to protect against attacks or failures at the client-side DNS infrastructure, including forms of cache poisoning, compromise, non-failstop failure, and spoofing [16]. These failure modes are real – as recently as several months ago, a new spoofing attack was discovered against the most recent BIND. By protecting the client-side DNS infrastructure, ConfiDNS reduces avenues for polluting global lookups in cooperative DNS systems. While ConfiDNS is not designed to detect server-side DNS spoofing, if the spoofing is selective or intermittent, ConfiDNS may still provide some protection.

The decision to focus on client-side problems is pragmatic – we believe that client-side problems are harder to manually detect, and easier to automatically defend. For example, if an attacker compromises a bank's DNS servers and redirects all traffic to a spoofed Web site, the bank's Web site will see a sharp and easily-detectable drop in activity. However, an attacker who wants to draw less attention could compromise an ISP's resolver, and redirect only lookups for one bank – the resulting drop in traffic may go unnoticed. We have seen several client-side resolver behaviors that could pollute a cooperative DNS service. In one scenario, we saw a site administrator pollute CoDNS by configuring a resolver to reply instantly to all requests with the IP address of a local webserver that served a page saying that the resolver was being replaced. Unfortunately, if the browser expected an image and received this error message, the web page displayed broken image icons, causing problems. We also measured three other instances of pollution, which are further described in Section 4.3. In all of these cases, the results were returned quickly, so any peer using the resolvers at these sites could find its own lookups poisoned in the process.

ConfiDNS's protection does not extend to arbitrary collusion among peers in the system. We present different policies that show how many peers need to agree on a result before ConfiDNS accepts it, and these policies are designed to tolerate different numbers of failing DNS resolvers. However, if an attacker controls all of the resolvers, or even the local ConfiDNS agent itself, we cannot determine the validity of IP addresses.

Although some techniques for strengthening DNS security have been proposed or deployed, they do not solve all of the problems mentioned above. For example, DNSSEC, which can prevent DNS spoofing by authentication, does not provide any support for distributing service (and distributing authentication), leaving the

infrastructure with the same reliability problems as the legacy DNS systems. Adding Byzantine fault tolerance to DNSSEC can further strengthen security while helping reliability, but requires much more dramatic changes to the infrastructure, and is unlikely to occur in the near future given the slow adoption of DNSSEC to date.

2.2 Applicability

ConfiDNS's ability to provide protection depends on how DNS is used (and abused). While we take a pragmatic view and try to accommodate the greatest range of DNS use, the extreme ends of the spectrum are worth considering, because they also provide interesting perspectives. DNS was originally designed so that the same name would resolve to the same IP address everywhere, and that this behavior could be exploited to provide distributed caching. In this model, ConfiDNS's agreement mechanisms are trivially satisfiable, since all sites should get the same answers when performing DNS lookups. The number of faults that ConfiDNS can handle is therefore limited by the number of ConfiDNS peers. The only question for ConfiDNS, then, is how stable the name-to-IP mappings are over time, since a site can seamlessly migrate servers by running both machines at different IP addresses and then waiting until old cached translations expire.

However, since the mid-1990's, intelligent DNS redirectors have been used for geographic load balancing, such as redirecting clients to nearby data centers [14, 23]. Content distribution networks such as Akamai [3] use very short DNS response TTLs to more aggressively balance load and locality. The number of possible IPs per domain name can number in the hundreds in these systems, since they will try to place servers at most large ISPs. These systems are potentially more problematic for ConfiDNS, since peers at separate ISPs may not agree on any domain names served by CDNs. In the extreme, if all sites used CDNs with servers at every large ISP, ConfiDNS would become much less useful. Realistically, though, we believe that the Web will continue to have a mix of sites hosted at single locations, a small number of data centers, and commercial large-scale CDNs. Our measurements later in this paper gives some indication of the breakdowns among the different approaches.

In this paper, we take a pragmatic approach to designing and evaluating ConfiDNS and assume that while some sites will adopt one style of delivery over another, the basic options available will remain qualitatively similar going forward. For each approach, we will try to provide the most protection possible, while realizing that some uses of DNS will simply be more amenable to our style of protection than others. Our interest is in determining what is the best we can possibly do given an imperfect situation, rather than trying to fight the same uphill battle of trying to replace the infrastructure that DNSSEC and others have tried. In the process, we hope to also gain insight into how DNS is actually used in practice, since the benefits of ConfiDNS will depend on actual DNS usage patterns, rather than on extreme models of possible usage.

3 Trace Analysis

To study how DNS is actually used and what sorts of workloads need to be considered for ConfiDNS, we perform an analysis of the operating CoDNS system, both in terms of how the DNS hierarchy is being used and what implications it has for caching strategies.

3.1 Building a Global DNS Trace

The CoDNS cooperative DNS system has been operational on PlanetLab [20] since October 2003, and has grown from handling 2 million requests per day on 95 nodes to now handling 5-7 million requests per day across PlanetLab (generally 430-450 live nodes). Its most heavy use comes from the CoDeeN [26] content distribution network, which handles over 25 million requests per day from over 50,000 daily users, but is also used by other PlanetLab researchers as well as members of the public who have installed the CoDNS agent on their personal machines.

To produce a global CoDNS trace, we combine log files from all CoDNS nodes over a 34 day period, producing a log with 1.05 million unique successfully queried names. To mimic a global cache, we reduce this log file by combining multiple lookups of the same domain name on each day. From this global cache, we then create a representative daily log by associating each name with a probability determined by how many days it appears in the global log. So, a name that appears every day in the global log will always be included in the daily log, whereas a name that only appears 17 out of 34 days in the global log would only have a 50% chance of appearing in the daily log. Using this methodology, the daily log is reduced to 132K entries, and is used in place of any particular day's log in the remainder of this paper. Creating the daily log in this manner allows us to build a representative day's traffic for a combined global DNS resolver. It also allows us to understand the limits of how much agreement is possible in ConfiDNS without being bound by *a priori* constraints on how many nodes can resolve each name.

This process intentionally creates a daily log where domain names appear only once. Although a local DNS resolver would see the same requests from multiple clients, we are interested in understanding the cacheability of names, and monitoring them over long periods, so we generate traffic patterns within the day as we desire. This behavior is also more like second-level caches rather than local DNS resolvers, since first-level on-box DNS caches are already commonly used for services like CDNs, mail transfer agents, etc. The on-box caches reduce the load

on the site's DNS resolvers, and isolate any performance impact from these high-demand services.

3.2 Domain Frequency Analysis

We can analyze the frequency and usage of domain names to understand how DNS domains and sub-domains are being used by their owners. Aggregating information across CoDNS is useful for several purposes – (1) it provides a snapshot of the state of the DNS system, (2) it gives some insight into global DNS behavior, whereas most previous studies examined only one or two client sites, and (3) it provides some guidance for designing caching strategies.

Note, however, that these statistics are only for the names themselves, and not for the amount of traffic sent to the site. CoDeeN's traffic is generally similar to the Alexa Top 100 list, but the name lookups and their statistics are more important for DNS caches that are interested in only the names. To give an example, google.com is a very popular site by traffic, but it requires very little space in a DNS cache since is has relatively few subdomains.

From these logs, we can see that domain name popularity, measured by how many days a name appears in the trace, is very bimodal. Figure 1 shows counts of how many names appear for a specified number of days in the 34-day trace, while Figure 2 gives the same counts only for the subset of names that appear in the daily trace. In the 34-day trace, most names (53.5%) appear exactly once, while a small set (12.9%) appear on at least 75% of the days, and a very small fraction (1.4%) appear every day. The trend is similar for the daily trace – 14334 names from the one-day trace appear every day in the 34-day trace, and 16559 names in the one-day trace appear only once each in the 34-day trace. The strong bimodality from the 34-day trace is diluted, since pruning reduces the relative weight of the singletons by a factor of 34.

The implications of this distribution are that aggressive caching of DNS names, particularly for global DNS resolvers, may be wasteful. As we see later, most name translations have relatively short TTL values, so caching infrequently-used names provides no benefit. The traffic to cache and find them, only to discover the mapping is stale, may exceed the traffic to simply re-fetch them from the server-side DNS infrastructure.

To understand why relatively recent studies of DNS traffic did not reveal this kind of bimodal distribution, and actually indicated a Zipf-like behavior [12], we examine the most-requested and most-populated domains, shown in Table 1. The counts indicate how many unique names appear under each domain, and the list shows many vanity sites (web logs) and hosting sites, often used to upload images and videos. Most of these sites are younger than the previous DNS studies (performed in 2001), and use DNS very differently, placing the user's name in the DNS name itself rather than as a subcomponent of the URL. This change causes a change in DNS name usage as millions of people create vanity sites. Since CoDNS is globally deployed, these logs also capture vanity sites in France (free.fr), Russia (narod.ru), and the Arab world (jeeran.com), something not captured in previous studies that only study traffic from one or two sites. The appearance of portals such as Yahoo and QQ (a Chinese portal) on these lists is because they expose cluster names via DNS – these names may map to actual machines, or may just be used for load balancing, but in either case, this partitioning is DNS-visible.

3.3 Caching Implications

We mine this data for implications on caching, since the two existing global DNS systems, CoDNS and CoDoNS, take very different approaches to caching. CoDNS relies on the caches of local DNS resolvers, and performs no caching itself, since it found that paging delays in the virtual memory system were one of BIND's [11] biggest performance and reliability problems. [19] CoDoNS, in contrast, aggressively caches lookup data on each node, in order to reduce the amount of communication needed to return lookup responses.

We query the origin DNS servers to obtain the TTL value for each domain name in the one-day trace, and find that the most common values cluster around 5 minutes, 30 minutes, 1 hour, 4 hours, 8 hours, 12 hours, and 24 hours. We show the breakdown of names in the one-day trace by their TTL groupings in Figure 3. We further select the 2.6% of names that appear every day in the 34-day trace and show their breakdown in Figure 4. In both cases, we add an extra category for those names that had a TTL greater than 24 hours. We observe three important features in this data:

1. Very few translations are cacheable for more than one day, so one day's storage will avoid almost all capacity misses – even if each record requires 64 bytes, all 132K entries in the one-day trace require less than 9MB per node. In practice, nodes will require even less memory, since the one-day trace represents a *combined* trace from all nodes.

2. Most TTLs are four hours or less, so the most popular names will be re-fetched at least six times per day. Predictively refreshing these entries may avoid lookup delays.

3. Many names have TTLs near five minutes, suggesting the use of commercial content distribution networks. The TTL breakdown for the names that appear daily show a greater fraction having a 5-minute TTL than names in the one-day trace. This shift is not surprising, since the most popular sites are prime customers for CDNs.

4 Continuous Monitoring

To obtain a more comprehensive view of how the global DNS hierarchy is used, we perform lookups from multiple vantage points. From this data, we can then analyze

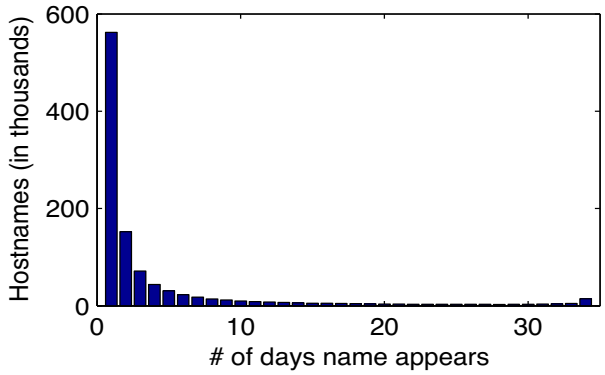


Figure 1: The number of days that each hostname appears in our 34 day survey of CoDNS.

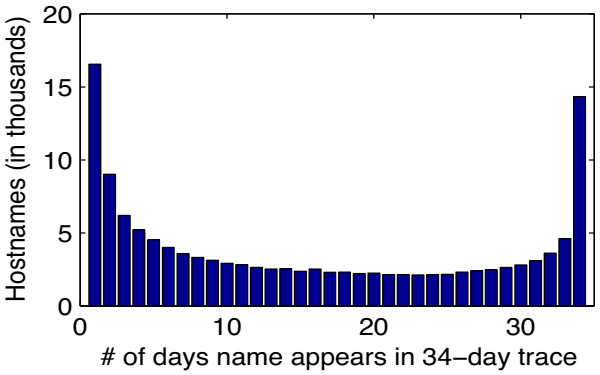


Figure 2: How often hostnames in our one-day trace appear in the monthly trace

34-day (1.05M names)				Daily (132K names)			
top-level		domain		top-level		domain	
592032	com	6642	blogspot.com (V)	82588	com	880	llnwd.net (H)
97948	net	4225	free.fr (H)	12909	net	854	yahoo.com (P)
49343	de	3898	camarades.com (V)	4810	de	494	imageshack.us (H)
44367	org	3775	jeeran.com (V)	4265	org	418	camarades.com (V)
40271	ru	3623	spylog.com (T)	4109	ru	390	blogspot.com (V)
19273	uk	3116	yahoo.com (P)	1848	uk	329	spylog.com (T)
17758	cn	2621	narod.ru (H)	1808	jp	296	free.fr (H)
16750	info	2351	infoseek.co.jp (P/V)	1780	cn	223	jeeran.com (V)
14847	jp	2255	tripod.com (H)	1458	info	220	2o7.net (T)
12054	nl	1413	fastturning.com (T)	1194	nl	200	qq.com (P)

Table 1: Statistics on the most-requested (from the daily trace) and most-populated (from the 34-day trace) top-level domains and regular domains. The count near each name indicates how often it occurs in the set. The indicators near each name indicate the type of site, with V for vanity/weblog, H for hosting, T for tracking, and P for portal. The unique names in the 34-day trace are not always related to aggregated request frequency, as in the case of fastturning.com, which now appears to be non-operational. Also, since this data was gathered, camarades.com has been purchased by another site.

how different usages of DNS affect potential ConfiDNS policies. This approach can give us more insight into the DNS operations of CDNs, as well as the rate of change of name-to-IP mappings.

4.1 Monitoring Setup

To obtain a more comprehensive view, we perform lookups in many distinct locations by using every site in PlanetLab that has its own local DNS resolver. A PlanetLab site consists of two or more co-located nodes; we refer to sites instead of nodes because our choice of node is made dynamically, dependent on which nodes are alive at each site during the trace initialization. After removing dead sites and sites that share DNS resolvers, we are left with approximately 180 accessible sites on most days. Over a period of 30 days, we have each site perform lookups from the one-day trace we described earlier.

To reduce the chances of causing problems for PlanetLab sites, we take a number of steps to make this process more manageable. To reduce local DNS resolver load, we query only one new hostname per second, and allow only 10 outstanding requests. At that rate, resolving 132K names would require over 36 hours, so we randomly select a 40K name subset, but ensure that all of the Alexa Top 100 are included. With this reduced list, we expect to perform all lookups in less than 12 hours at most sites, and no more than 24 hours at the slowest sites. We take two steps to avoid tripping overly-sensitive intrusion detection systems and overloading DNS servers with synchronized lookups. First, we randomize the list to reduce the chances of querying long bursts of names to the same domain. Second, we have each Planetlab site pick a random starting position in the list. This same starting position is used each day so that the same name is resolved at approximately the same time on a per-site basis.

We performed the monitoring using all available sites for 30 days, and collected information such as the elapsed time per lookup, the IP addresses returned, and the canon-

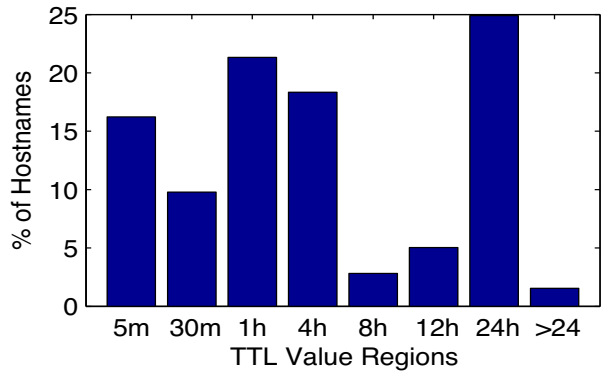


Figure 3: TTL values of hostnames that appear in the one-day trace

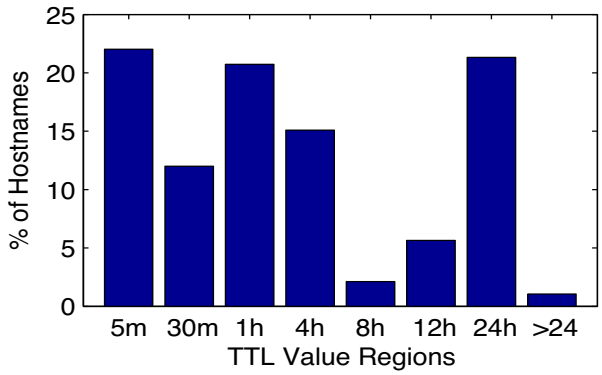


Figure 4: The TTL values of hostnames that appear daily in the 34-day trace

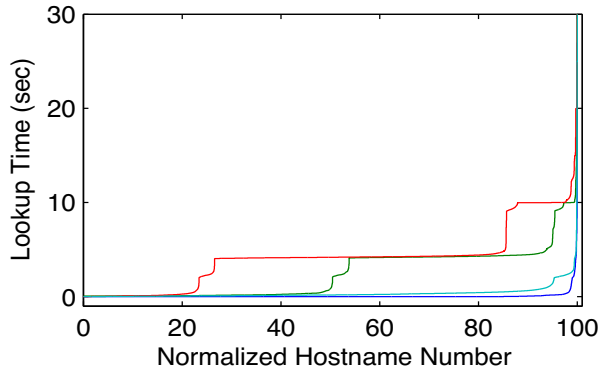


Figure 5: Comparison of Local DNS response times at four sites.

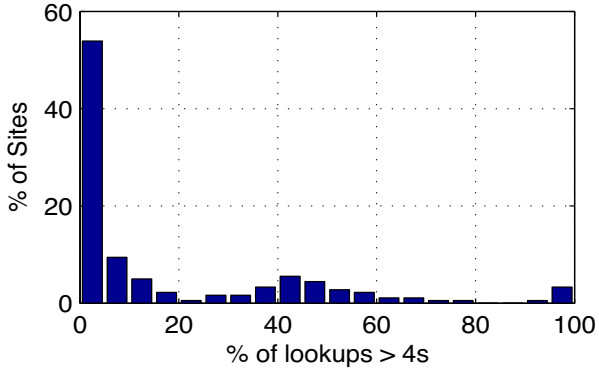


Figure 6: Breakdown of node retry percentages, where DNS lookup times over 4 seconds implies retry.

ical name returned. We also had each node ping its neighbors, and collected round-trip ping times between the PlanetLab nodes. This information is used as the input for the analysis in the rest of this section.

4.2 Query Response Time Breakdowns

These tests give some insight into what range of DNS response times most clients experience when using their local DNS resolver. This measure is important because it provides a baseline comparison for the policies implemented in ConfiDNS. Figure 5 shows the query response times for our set of names at just a few sites, with the values sorted by time to illustrate some common query response behaviors.

We observe three plateaus near zero, five, and ten seconds, and a sharp rise near the end of each line. The plateaus are the timeout values used by the resolver libraries to reissue DNS requests. The plateau shape is due to the fact that most lookups require less than one second when successful, so the response time can be dominated by the retry delay. A larger plateau near zero seconds is desirable, because it indicates that the resolver is either caching very well, and/or is having no problems in the paths between it and the server-side DNS infrastructure.

To obtain a broader perspective of DNS query performance, we can reduce the results from each site to a single statistic – what fraction of sites require more than a given amount of time to complete their query. We choose 4 seconds as the threshold because it conservatively separates retries from non-retries. We calculate the breakdown for each node, and show a histogram of node failure rates in Figure 6. While most sites (almost 100 out of 180) perform reasonably well, a surprisingly large number (approximately 30) have problems with nearly half of their queries. An additional 7 sites appear to have primary resolvers that are almost entirely non-responsive, where more than 90% of queries take longer then 4 seconds.

Notice that this breakdown is worse than the local DNS resolver health measured in the earlier CoDNS work. The explanation for this difference is that CoDNS monitoring measured the failure rate of cached queries only, to separate resolver failures from cache capacity and replacement policies. Once these extra factors are included in

the end-to-end measurements, many local DNS resolvers perform much worse. However, some servers still perform quite well, with failure rates in the low single digit percentages. Given that a sizable fraction of the ConfiDNS queries are for unpopular names, and we can expect these to be evicted from cache regularly, we can see that the DNS server-side infrastructure is not the major source of problems in the system. If it were, few nodes would be able to achieve low retry rates given the fact that our querying process takes 12 hours and that it queries a number of names unlikely to be cached.

4.3 CDNs and Data Centers

The amount of protection-by-agreement that ConfiDNS can provide is a function of how many clients receive the same IP address for each name. For a content provider that has only one server and one connection to the Internet, all clients should receive the same information when resolving its DNS name. However, providers will often use some form of geographic replication, with multiple data centers or a content distribution network. In these cases, the total number of IP addresses that map to the same name can be much larger, especially if the CDN attempts to have a point-of-presence at every major ISP.

The precise number of IP addresses returned per hostname is less important than the pattern of how they are returned. A domain may have two data centers with addresses IP_1 and IP_2. If it selectively returns one IP address based on load or locality, we say that the hostname has two *regions*, but if it returns both addresses for every query, we say that it has a single region because all queries see the same set of addresses. At this level of analysis, a single data center with two connections to the Internet (commonly called a multi-homed site) is equivalent to a domain with two data centers but which returns both IPs to every query. Of the roughly 40K domains we query, we find that roughly 90.8% return the same single IP address to all queries. Another 4.2% return multiple IP addresses, but return the same set of IP addresses to all queries.

These statistics are very positive for ConfiDNS because 95% of domain names in the one-day trace can have agreement bounded only by the total number of nodes participating in ConfiDNS. The remaining 5% of domain names (2002 out of 40154) are not automatically out of reach for ConfiDNS – for small numbers of regions, they should be satisfiable for a number of policies.

The exact breakdown of the 2002 hostnames with two or more regions in our one-day trace is shown in Figure 7(a), with hostnames further differentiated into names served by Akamai (354 hostnames) and names either operating on their own or served by other CDNs (1648 hostnames). Akamai-served hostnames are determined using the canonical name returned in the DNS query – any names on akamai.net, akamaiedge.net, speedera.net, akastream.net, akareal.net, or yimg.com (Yahoo's images served by Akamai) are grouped together as Akamai.

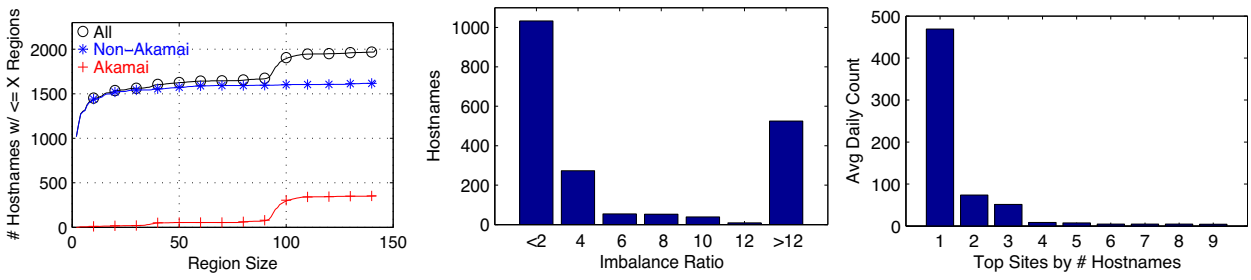
IP Prefix	# sites	Region	Sample Sites
74.125.47.x	2	Southeast US	Georgia Tech
74.125.45.x	3	Kentucky	UKY (Lexington)
74.125.19.x	9	Southwest US	UCSD (San Diego) & ASU (Arizona)
72.14.253.x	2	Wash, Oregon	U Oregon, WSU
72.14.215.x	1	Switzerland	U Zurich
72.14.205.x	4	NYC, Conn	U Conn
66.249.93.x	6	Europe	Vrije U
66.249.91.x	7	Northern Europe	U Helsinki
66.249.89.x	8	Japan,TW	JAIST, Osaka U
66.102.9.x	4	Portugal	U Lisboa
64.233.189.x	5	HK, Korea & China	CUHK (Hong Kong) & Southeast U (Nanjing)
64.233.183.x	9	Western Europe	Cambridge, & UPM (Madrid)
64.233.169.x	25	Eastern US	CMU, Duke
64.233.167.x	35	Great Lakes & Midwest	U Toronto, & Indiana
64.233.161.x	9	Northeast US	NEC Labs (Princeton), & MIT
209.85.193.x	1	Brazil	RNP (Brazil)
209.85.175.x	2	Asia	NTU, SNU
209.85.173.x	15	Western US & Canada	U Washington Intel Res. Berkeley
209.85.135.x	8	Europe	Fraunhofer, LIP6
209.85.137.x	35	Germany, & Austria	U Austria, & TU Darmstadt

Table 2: Breakdown of regions observed for www.google.com with representative sites belonging to each region set

Of the non-Akamai hostnames, 1019 have only two regions, 1429 have ten or fewer regions, and 219 have more than 10 regions. In contrast, most Akamai-served hostnames have 80-90 regions, with Akamai's servers for America Online showing the largest number of regions. Akamai also offers a DNS redirector that resolves to the customers own data centers. These hostnames are handled by akadns.net, and we count these as non-Akamai CDNs since the customer is ultimately handling the actual data centers.

In Table 2 we show a reasonably popular domain - www.google.com, where we find 21 distinct regions visible. The wide range of region sizes implies that different policies will be effective in different parts of the world. Multi-site agreement may work well in the Western United States where there are a sufficient number of available peer sites, but is unlikely to be effective along, for instance, Brazil where there are relatively few available peers.

By comparing the number of sites in each region versus how many would appear in perfectly-balanced regions, we calculate a region imbalance factor for each multi-regioned name in our trace. Given the set of regions with



(a) Region Sizes (b) Region Imbalances (c) Site Poisoning

Figure 7: Region information for names mapping to multiple disjoint IP addresses

the number of nodes per region, we calculate a geometric mean of a series of terms, where each term is either the ratio of actual region size to average region size, or its inverse, whichever is larger. For example, if a hostname has three regions with 15, 55, and 80 sites, its imbalance ratio is $\sqrt[3]{\frac{50}{15} \times \frac{55}{50} \times \frac{80}{50}}$. This calculation is designed to identify names that have gross imbalances in the sizes of their regions. While most names have region sizes that are within a factor of 2-4 of being fully balanced, we see a spike where the imbalance ratio exceeds 12 – in these cases, only one site disagrees over the IP address, and all of the other sites form a second region.

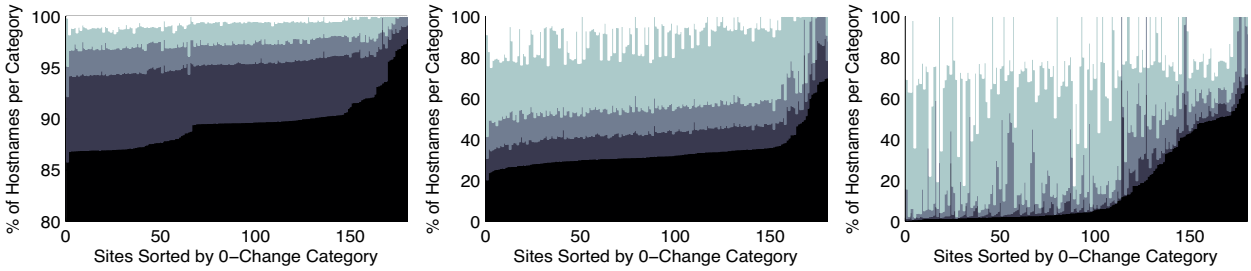
To get a sense of the origins of these heavily-imbalanced regions, we counted how often a site disagreed with all others, and show the daily average for the top ten sites in Figure 7(c). The worst site has an average of 469 hostnames per day whose lookups differ from all others. This set of names is fairly stable, and an examination of their contents suggests that it is policy-driven censorship, since they are resolved to IP addresses that provide no responses. Users will be able to seemingly resolve the name, but will be unable to contact any machine at the address, and may conclude the server does not exist. The second-worst site appears to have a traffic-sniffing virus checker working in conjunction with the local DNS resolver. When it activates, all lookups from the client are directed to a local Webserver with a message warning that your client is infected. Unfortunately, the virus sniffer returns false positives, and indicated that our Linux-based boxes were infected with Windows viruses. The third-worst site appeared to be having sporadic failures, and was randomly returning the IP address of the school's main Web server for queries, with no discernible pattern to its behavior. The remaining sites show no strong patterns of poisoning, with most of the imbalances stemming from slowly-deployed changes in name-to-IP mappings. In all of these cases, any multi-site agreement policy in ConfiDNS would automatically prevent these sites from poisoning the lookup results.

4.4 IP Address Changes

By monitoring for 30 days, we gain some insight into how often name-to-IP mappings actually change, rather than just relying on the advertised TTLs as an estimate. While it might be inadvisable to use past history to serve stale mappings, the fact that a mapping has been the same for an extended period of time may give users more confidence in it. Conversely, a mapping that suddenly changes when it had previously been stable may be cause for concern – it may be as simple as a server being replaced or migrated, or it may be that an attacker has spoofed a response and is trying to divert traffic.

From the monitoring data, we calculate the observed rate of change of name-to-IP mappings during our test period. For each site, we examine the results of the lookup for each name across 30 days, and count the number of times the returned IP differs from the previous day's value. For names that map to multiple IPs, we conservatively consider the result to have changed if any member of the set of IPs change. However, we do not consider set ordering important. Since we monitor the mappings only once per day, our approach should be considered a reasonable estimate of the rate of change rather than the precise answer. A domain with a short TTL could presumably change and then revert its mappings between our measurements, and we would miss the change. However, since the probe order differs at each of our measuring sites, such a change would likely trigger our system to detect a change in the number of regions for the domain. If both changes occurred during the 12 hours we do not monitor, the likelihood of our observing it decreases. Given the patterns we observe and our intended uses for the data, we do not believe these corner cases are much of a concern.

The rate of change for all names across all sites is shown in Figure 8(a). For each site, we group names by the number of changes observed, and then report the size of these groups. We see that at every site, more than 85% of names do not change over the 30 day period, despite having TTL values less than 30 days. The remaining bars



(a) All names

(b) Names with 2-10 regions

(c) Names with 10+ regions

Figure 8: Rate of change for name-to-IP mappings. The bars, from bottom to top, are for zero changes, 1 change, 2-3 changes, 4-14 changes, and 15+ changes (in white). **Note: Y axis truncated to improve detail in first graph.**

group the number of changes and are intended to show that while some names change on virtually every lookup, most change much more slowly. If these rates of change are typical, then we see that most names are stable for a month at a time, and more than half of the names that do change are in fact, stable for two weeks. The set of names that change more frequently than once per week ranges from 1-3% in this study.

The data in Figure 8(b) breaks out the names with a small number of regions (2-10), so that we can determine how often most multi-regioned sites send clients to different regions. We see that even here, a large number of names have long periods of stability – decisions to send clients to nearby data centers are likely to be stable over time, unless the closest data center becomes unavailable for maintenance, link outages, etc. The names that show high rates of change in these measurements may indicate that in addition to (or instead of) geographic proximity, the DNS server is also being used to load balance traffic across multiple data centers.

Finally, the data in Figure 8(c) shows the same statistics, but only for those names that map to more than 10 regions. Included in this set are most of the Akamai-served domains, some domains served by LimeLight Networks (another CDN) as well others that seem to be using a fairly large number of their own data centers (or hosting centers). The increase in the size of the zero change category beginning near node 100 is largely a function of the size and deployment of Akamai clusters – these do not appear to use hardware load balancers, so the larger the cluster, the more IP addresses that are exposed and rotated, causing high rates of IP address changes. In contrast, Google clusters, despite having thousands of nodes, advertise only a small number of IP addresses as entry points.

As can be seen from this data, IP result history can be used profitably to provide an indication of stability, and this pattern holds true across domains served using a variety of different strategies. This observation bodes well particularly for those parts of the world where sparse coverage may preclude certain clients from finding enough peers in agreement to use only agreement-based policies. In these cases, the stability of a name-to-IP mapping can provide some reassurance greater than just the local DNS resolver alone provides.

The stability data we have gathered may also be useful in shaping decisions about when to use stale DNS data, as has been proposed in another system [10]. If the DNS infrastructure and the actual content servers do not share fate, it may be the case that a domain has functioning servers but is not accessible to users because their existing DNS entries have become stale. Note that this fate-sharing requirement is not completely unrealistic – several companies perform outsourced DNS service, so a failure at a third-party DNS provider could have no correlation to the domain's own content servers failing. In these cases, a DNS resolver could potentially be configured to provide stale data if it met some predefined criteria. We do not explore this idea further in this paper, but leave it as a possible avenue for future work.

5 Implementation

ConfiDNS is implemented as a service running on PlanetLab, with an architecture similar to the CoDNS system. Each PlanetLab node runs a ConfiDNS agent, which can also be run on the user's local machine. The agents accept DNS queries using TCP and UDP, and when run on a local machine, can automatically modify the `/etc/resolv.conf` file to automatically handle all locally-generated DNS traffic.

The policy differences between CoDNS and ConfiDNS focus on when to contact peers. In CoDNS, requests are first forwarded to the local DNS infrastructure, and only sent remotely if the local resolver does not respond within an adaptive time-out period. The exception to this policy is if the local resolver is deemed dead, in which case all requests are immediately forwarded to a peer. In ei-

ther case, CoDNS contacts successive peers only when the previous peers fail to respond in an exponentially-increasing timeout period. Peer selection in CoDNS is performed from a set of nearby nodes using the Highest Random Weight (HRW) hashing scheme [25][19], in order to preserve cache locality.

In ConfiDNS, all locally-generated queries are immediately forwarded to a specified number of peers, chosen purely based on proximity using a heartbeat ping. The decision to use proximity alone was motivated by the hope that peers that are closer are more likely to be in the same CDN region. Additional queries are sent only when the agreement policy has not been met within a give timeout. As with CoDNS, queries that are received from remote nodes are not re-forwarded, both to prevent forwarding loops, and to limit the damage that any compromised peer can cause.

In keeping with the desire to prevent any pollution from spreading within ConfiDNS, only locally-resolved lookup results are stored for keeping history and satisfying query requests. Both locally-generated and remote queries can be satisfied from the cache, but only locally-generated results are entered into the cache to avoid pollution.

ConfiDNS uses a configuration file that specifies domain name suffixes and policies, so that policies can be customized as needed. Possible policies include the first response, agreement of N out of M peers, and historical agreement. Multiple policy lines can be provided with different start times, so that one can opt for different decisions if a previous policy is taking too long to satisfy. Canonical names and IP addresses can also be specified, allowing the whitelisting of any Akamai-served name, or just Akamai-served names from a given set of IP addresses. ConfiDNS is responsible for determining when the specified lookup policy has been satisfied. If no agreement is reached between the set of remote peers, the ConfiDNS agent sends a failure response to the client, but does not cache the result. We have thought of having the failure response direct the client to a locally-configured Web server that can explain why the lookup failed (using out-of-band information), but have not implemented this approach. The benefit of this scheme is that the user would then be able to see why a given policy could not be satisfied, and could then choose a different action, including possibly choosing a new policy or reporting any anomaly to a system administrator.

6 Evaluation

In this section, we evaluate a number of ConfiDNS policies, first examining policies that relate only to agreement, and then combining agreement and history. Our primary focus in this evaluation will be coverage (applicability) and latency – we choose policies that are designed to have reasonable network overheads, so our initial analysis will focus on how many domains and how many sites benefit from the various types of security the different policies provide. Given the observations in CoDNS about trading latency for network overhead, we believe that all of these policies can be tuned as required.

We evaluate four agreement policies for ConfiDNS, requiring agreement among varying numbers of peers from various maximum peer-set sizes. In one sample policy, we require that the set of agreeing peers include the local DNS resolver. In the rest, the local DNS is just one of several peers that may participate in the agreement process. Each peer consists of a single node at any particular PlanetLab site that possesses a locally managed (non-shared) DNS resolver. In each case, we place a restriction on the number of peers that can be queried in order to reach agreement. Peers are ranked using their round-trip times, and we choose the set of peer nodes with the lowest RTTs. We use this ranking method to choose our peer sites both for the obvious reason, to minimize query response time, but also to reduce spurious results being returned for multi-region address mappings. For example, if we require five peer agreement on a response to a DNS query without a locality restriction on peering, nearly every domain name can meet the agreement requirement so long as we choose a reasonably well localized set of peers. These peers are not necessarily within our own region, indeed they may be anywhere in the globe. Directing all traffic to that potentially far flung region is unlikely to be a desirable property, both from the point of view of the end-user, who may see a degradation in service performance at a particular host, and also from the point of view of a service operator, who will find DNS-based load-balancing to be less effective with potentially many users circumventing the redirection.

All policies are evaluated at every site, and per-site average latencies are reported in Figures 9– 14. The baseline policy, using only the local DNS resolver, suffers from the problem of retries that we described in Section 4.2. Likewise, the policy of requiring that at least one other site (out of the five closest peers) agree with the local resolver shows similar performance because the local DNS response time is the bottleneck. A simplified form of the CoDNS policy is shown in Figure 11, and takes the first response of the local resolver and the three nearest peers. The dispatch of queries to the peer sites are staggered using the same delay values that CoDNS uses in deployment. This policy (as does CoDNS) shows a significant response time improvement over the local DNS resolver, precisely because we do not have to wait for the local resolver's response when it is slow.

The more aggressive agreement policies for ConfiDNS require 3 of the 10 closest sites agreeing (Figure 12), 5 sites out of 20 agreeing (Figure 13), or 7 sites out of 30 agreeing (Figure 14). Though we are primarily concerned with the level of agreement possible, we stagger the dis-

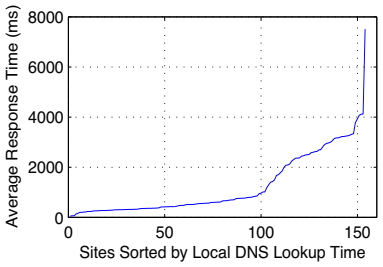


Figure 9: Local DNS resolver only. **Note 8000ms Y axis**

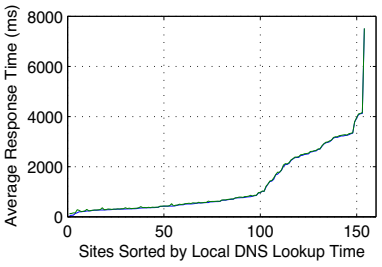


Figure 10: Local + 1 site from 5 peers. **Note 8000ms Y axis**

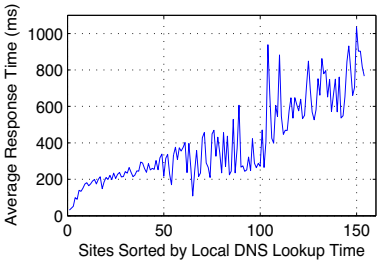


Figure 11: CoDNS

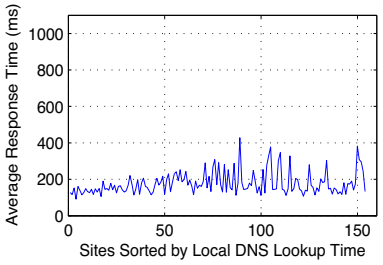


Figure 12: 3 sites from 10 peers

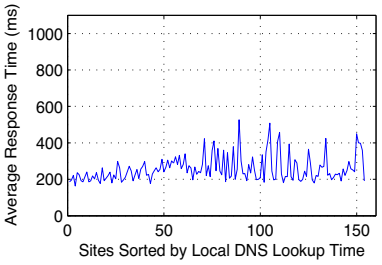


Figure 13: 5 sites from 20 peers

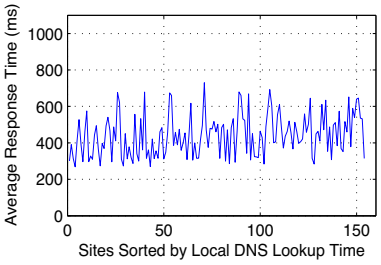


Figure 14: 7 sites from 30 peers

patch of queries in these policies at the rate of 1000ms every 10 lookups for fairness. A practical, low-overhead policy of taking either the local DNS resolver and one peer, or 3 other peers, is not shown because its latency characteristics are identical to Figure 12.

To evaluate our sample implementation of ConfiDNS we once again sampled traffic from CoDNS, and replayed a previous day of CoDNS lookups using the same request timing as in the original trace, but with the previous days names reduced to a single unique lookup per name. This resulted in approximately 20,000 unique names resolved at a frequency determined by the original pattern of traffic. Figure 15 shows the per-site average response time, compared to the same trace resolved at the same sites using CoDNS. We can see that the actual performance is similar to that predicted from our trace-based analysis.

The most important latency observation for these more secure policies is that they perform much better than local DNS resolvers, and are in fact generally better than CoDNS. The 3-agreement policy performs surprisingly well compared to CoDNS, with an average latency almost half of CoDNS's. This is partly due to the fact that CoDNS initially waits 200ms before dispatching peer queries (on nodes with healthy local resolvers) while ConfiDNS dispatches these queries immediately. The 5-agreement policy performs roughly 25% worse than 3-agreement across all sites, but is still better than CoDNS. The 7-agreement policy performs another 60% worse in general, but is otherwise comparable in latency to CoDNS. These results show that the ConfiDNS policies can produce good latencies, even for relatively high levels of agreement. If the accompanying network overheads are unacceptable, then the queries to the peers can be staggered even more so that network overhead is reduced at the cost of some latency.

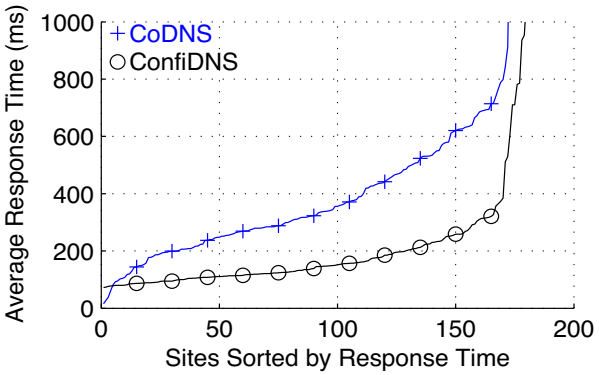


Figure 15: ConfiDNS implementation 3 sites from 10 peers vs. CoDNS

Since ConfiDNS query latency is clearly improved over non-cooperative schemes, the other concern is what fraction of domain names can be satisfied using each of the policies, where satisfiability refers to the ability of a particular client to resolve a name with the given policy setting. Since the agreement policies are tied to how multi-region domains behave, the satisfiability concerns are related to the location of the site, the distance to nearby peers, and the granularity of CDN redirection processes

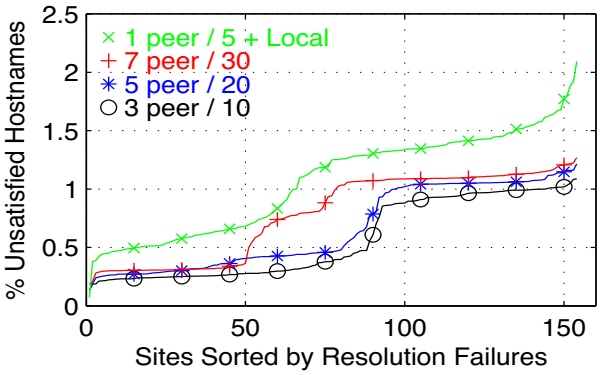


Figure 16: Percentage of names that cannot be satisfied at each site given the particular agreement policy. Data is sorted on a per-policy basis for ease of viewing.

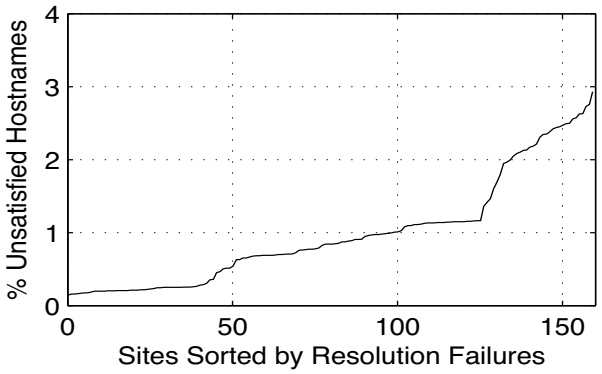


Figure 17: Percentage of names that cannot be satisfied at each site by our ConfiDNS implementation given a 3 peer agreement policy.

in the local area. Rather than reducing these numbers to averages, we present the percentage of domain names that each policy fails to satisfy at each site. This data is shown in Figure 16 for the trace-based analysis, and in Figure 17 for the live implementation on a 3-agreement policy using 10 peers. Again, the live implementation agrees closely with our trace-based analysis.

In general, we see that only about 1% of the domains fail to be satisfied under the various agreement policies, and at one-third of the sites, the rate is even lower, in the range of 0.3%. These results are in-line with what we had observed about the number of regions used by names that had more than one region. Despite multi-regioned names accounting for 5% of the trace, the fact that most of these names have fewer than 10 regions makes them amenable to our agreement policies. The names that are not amenable are the cause of the unsatisfiability rates observed. The plateaus in these graphs are worth mentioning – the lower plateau occurs because of CDN nodes outside of the Unites States and Europe. In these areas, the smaller ISPs may not have enough bandwidth for CDN companies to place nodes inside their networks. As a result, it appears that the CDN nodes are in regional networks, and are used by many ISPs, leading to higher agreement rates than within the US and Europe. The other interesting result worth discussing is why the weaker policy of "local DNS + 1 peer" performs poorly – it dispatches queries to only 5 peers, whereas the 3-agreement tries 10 peers, thus creating greater potential for successful agreement. Additionally, forcing the peers to agree with the local DNS is more restrictive.

	# Days IP is Stable				
	none	2-3	7	15	30
local DNS only	1	2	3	3	4
3 peers	2	3	3	4	4
local DNS + 1	3	3	4	4	5
5 peers	3	4	4	5	5
7 peers	4	4	5	5	6

Table 3: For ease of analysis, we linearize and collapse the range of protection policies in to single value labels, as shown above. Higher numbers indicate better protection than lower numbers.

6.1 Putting It All Together

We can now combine the policy agreement data and the rate-of-change monitoring data to determine the spectrum of protection policies that are possible, and how many hostnames can be satisfied with each. Rather than try to combine the aggregated data we have gathered, we perform the analysis for each hostname on each domain. We have two dimensions, agreement and history, so to simplify the analysis, we linearize the range of possibilities, as shown in Table 3. The process is assigning values to policies is subjective, but our main goal was to give an idea of the strength of combinations, with higher numbers indicating better protection. To get a sense of how often the policies are satisfied, the per-site breakdown for each label is shown in Figure 18.

The average breakdown of labels across sites is shown in Table 4. We can see that the percentage of hostnames that can only be satisfied by label 1, the weakest security policy, averages 0.18%. As label 1 is equivalent to local resolver lookup with no query history, this result indicates that ConfiDNS improves the security of 99.82% of queries. Even if we pick a stronger security requirement, such as label 4, which indicates seven peers agreeing, or 30 days of stability, or some intermediate combinations, ConfiDNS is able to satisfy 99.64% (the sum of labels 4 - 6) of the queries. Even the strongest policy, with 7 peers agreeing and the hostname resolution being stable for 30

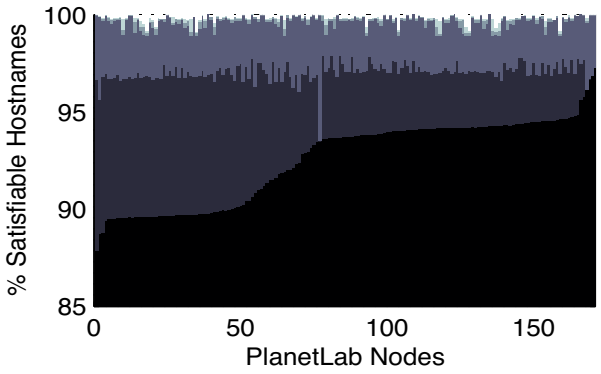


Figure 18: Breakdown of linearized policy labels by site. The bottom bar is linearized label 6, and the top bar (white) is label 1. **Note: Y axis truncated to show detail**

Policy label	Mean %	Std Dev
1	0.18	0.17
2	0.07	0.12
3	0.12	0.13
4	2.59	0.54
5	4.49	2.08
6	92.56	2.16

Table 4: Mean percentage of hostnames satisfied by a particular policy label with corresponding Std Dev. Most lookups can be satisfied by one of the stronger policies (4,5,6) instead of the weaker ones (1,2,3).

days, still works for over 92% of queries. As previously shown, this extra security does not come at a high cost – latency is better than local DNS alone, and is comparable to CoDNS (which has much weaker security than local DNS only).

7 Related Work

The idea of using some form of replication to achieve fault-tolerance is older than the authors of this paper. In the OS community, this approach has recently seen a revival of interest, especially since protecting geographically-distributed systems against Byzantine failures has become a focus of concern. Examples of such include performance-conscious Byzantine fault tolerance for NFS [8], including Byzantine agreement for quorum systems [17], and scaling cooperative services when facing Byzantine and selfish users [2]. These techniques have also been proposed to secure DNS [1, 27], when used in conjunction with DNSSEC.

These approaches provide strong protection, at the cost of relatively high requirements, such as having heterogeneous systems and components. In practice, for widely-deployed services such as DNS, the number of different configurations (including different components) is low enough that Byzantine requirements may be difficult to meet with the current infrastructure [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 [9] using a distributed hash table [24]. This replaces the hierarchical DNS name resolving process with a flat peer-to-peer query style in pursuit of load balancing and robustness. With this design, misconfigurations from administrator mistakes can be eliminated and the traffic bottleneck on the root servers is removed so that load is distributed over the entities joining the system. In CoDoNS, Ramasubramanian *et al.* improve the latency performance of this approach by using proactive replication of DNS records [22]. They exploit the Zipf-like distribution of domain names in web browsing [7] to reduce the replicating overhead while providing O(1) proximity [21]. Our previous work on this subject includes a workshop paper [15] where we sketch the ideas presented here. We expand this work with an in-depth analysis of our global DNS trace including a discussion of name-popularity its caching implications, as well as an investigation of query response-times. Finally, we outline an implementation that is a base for our future work.

8 Conclusion

Cooperative DNS resolvers have proven their utility in improving reliability and performance when compared to local DNS resolvers, but at the cost of weakened security. In this work, we show that by using peer agreement and storing some past history, our new cooperative resolver ConfiDNS, can provide *better* security than both traditional DNS resolvers and previous cooperative approaches, for the majority of domain names. Using a month-long world-wide survey of DNS behavior on a realistic global DNS trace, we are able to determine the applicability of various agreement policies and quantify their effect on latency. This paper also provides some raw data about global DNS behavior that should be useful to the broader research community. In addition to providing some insight into DNS behavior at scale, we also demonstrate that new uses of DNS by the increasingly popular vanity sites is qualitatively changing the patterns of DNS namespace usage. This study also provides us with information on the real usage of DNS mappings at a variety of domains ranging from small, singly-hosted sites to sophisticated, replicated data centers with DNS redirection, and finally to commercial third-party content distribution networks. In all cases, we find that it is possible to leverage scale, history, or both, and provide a much more secure result than local DNS alone.

These benefits are obtained without changing any

server-side DNS infrastructure, and with a tolerable and tunable network overhead. As a result of our design, ConfiDNS is incrementally deployable, requiring only a minimal agent running on either client machines or on client-side resolvers. Performance and network overhead can be improved by adding a small amount of caching to ConfiDNS, and we quantify the cost impact of a reasonable caching scheme. Finally, our approach is compatible with server-side approaches to improving DNS security such as DNSSEC, and together can provide reliability and performance benefits in addition to improved security.

Acknowledgments

We would like to thank our shepherd, David Presotto, and the anonymous reviewers for their useful feedback on the paper. This work was supported in part by NSF Grants ANI-0335214, CNS-0439842, and CNS-0520053.

References

[1] S. Ahmed. A scalable Byzantine fault tolerant secure domain name system, 2001. Massachusetts Institute of Technology Technical Report MIT-LCS-TR-849.

[2] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth. Bar fault tolerance for cooperative services. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, Brighton, United Kingdom, October 2005.

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

[4] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Dns security introduction and requirements. Request for Comments 4034, March 2005.

[5] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Resource records for the dns security extensions. Request for Comments 4035, March 2005.

[6] R. Arends, R. Austein, M.Larson, D. Massey, and S. Rose. Protocol modifications for the dns security extensions. Request for Comments 4033, March 2005.

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

[8] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *In Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, New Orleans, LA, February 1999.

[9] 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.

[10] H. Ballani and P. Francis. A Simple Approach to DNS DoS Mitigation. In *Proceedings of the 5th ACM Workshop on Hot Topics in Networks (HotNets '06)*, Irvine, CA, November 2006.

[11] Internet Systems Consortium (ISC). ISC BIND. http://www.isc.org/.

[12] H. B. Jaeyeon Jung, Emil Sit and R. Morris. DNS Performance and the Effectiveness of Caching. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop '01*, San Francisco, California, November 2001.

[13] B. Knowles. Domain Name Server Comparison: BIND 8 vs. BIND 9 vs. djbdns vs. ??? In *Proceedings of the USENIX LISA Conference 2002 - Technical Program*, Berkeley, CA, November 2002.

[14] B. Krishnamurthy, C. Wills, and Y. Zhang. On the use and performance of content distribution networks. In *Proceedings of SIGCOMM Internet Measurement Workshop*, San Francisco, CA, November 2001.

[15] L. Poole and V. S. Pai. ConfiDNS: Leveraging Scale and History to Improve DNS Security. In *Proceedings of the Third Workshop in Real, Large, Distributed Systems (WORLDS '06)*, Seattle, WA, November 2006.

[16] A. Lioy, F. Maino, M. Marian, and D. Mazzocchi. Dns security. In *Proceedings of the TERENA Networking Conference*, Lisbon, Portugal, May 2000.

[17] D. Malkhi and M. Reiter. Byzantine quorum systems. In *Proceedings of the 29th ACM Symposium on Theory of Computing (STOC)*, El Paso, TX, May 1997.

[18] P. Mockapetris and K. Dunlap. Development of the Domain Name System. In *In Proceedings of the ACM SIGCOMM Conference*, Stanford, CA, August 1988.

[19] K. Park, V. S. Pai, L. Peterson, and Z. Wang. CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups. In *Proceedings of the Sixth Symposium on Operating Systems Design and Implementation (OSDI)*, San Franscisco, CA, December 2004.

[20] PlanetLab. An open testbed for developing, deploying and accessing planetary-scale services, September 2002. http://www.planet-lab.org/.

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

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

[23] A. Shaikh, R. Tewari, and M. Agrawal. On the effectiveness of dns-based server selection. In *Proceedings of INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies*, Anchorage, AK, April 2001.

[24] 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, August 2001.

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

[26] L. Wang, K. Park, R. Pang, V. Pai, and L. Peterson. Reliability and Security in the CoDeeN Content Distribution Network. In *In Proceedings of the USENIX Annual Technical Conference*, Boston, MA, June 2004.

[27] Z. Yang. Using a byzantine fault tolerant algorithm to provide a secure dns, June 1999. Massachusetts Institute of Technology Masters Thesis.