The Design and Implementation of a Next Generation Name Service for the Internet

Venugopalan Ramasubramanian Emin Gün Sirer

Dept. of Computer Science, Cornell University,
Ithaca, NY 14853
{ramasv,egs}@cs.cornell.edu

ABSTRACT

Name services are critical for mapping logical resource names to physical resources in large-scale distributed systems. The Domain Name System (DNS) used on the Internet, however, is slow, vulnerable to denial of service attacks, and does not support fast updates. These problems stem fundamentally from the structure of the legacy DNS.

This paper describes the design and implementation of the Cooperative Domain Name System (CoDoNS), a novel name service, which provides high lookup performance through proactive caching, resilience to denial of service attacks through automatic load-balancing, and fast propagation of updates. CoDoNS derives its scalability, decentralization, self-organization, and failure resilience from peer-to-peer overlays, while it achieves high performance using the Beehive replication framework. Cryptographic delegation, instead of host-based physical delegation, limits potential malfeasance by namespace operators and creates a competitive market for namespace management. Backwards compatibility with existing protocols and wire formats enables CoDoNS to serve as a backup for legacy DNS, as well as a complete replacement. Performance measurements from a real-life deployment of the system in PlanetLab shows that CoDoNS provides fast lookups, automatically reconfigures around faults without manual involvement and thwarts distributed denial of service attacks by promptly redistributing load across nodes.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Domain Name System*

1. INTRODUCTION

Translation of names to network addresses is an essential predecessor to communication in networked systems. The Domain Name System (DNS) performs this translation on the Internet and constitutes a critical component of the Internet infrastructure. While the DNS has sustained the growth of the Internet through static, hierarchical partitioning of the namespace and wide-spread caching, recent increases in malicious behavior, explosion in client population, and the need for fast reconfiguration pose difficult problems. The existing DNS architecture is fundamentally unsuitable for addressing these issues.

The foremost problem with DNS is that it is susceptible to denial of service (DoS) attacks. This vulnerability stems from limited redundancy in nameservers, which provide name-address mappings and whose overload, failure or compromise can lead to low performance, failed lookups and misdirected clients. Approximately 80% of the domain names are served by just two nameservers, and a surprising 0.8% by only one. At the network level, all servers for 32% of the domain names are connected to the Internet through a single gateway, and can thus be compromised by a single failure. The top levels of the hierarchy are served by a relatively small number of servers, which serve as easy targets for denial of service attacks [5]. A recent DoS attack [29] on the DNS crippled nine of the thirteen root servers at that time, while another recent DoS attack on Microsoft's DNS servers severely affected the availability of Microsoft's web services for several hours [39]. DNS nameservers are easy targets for malicious agents, partly because approximately 20% of nameserver implementations contain security flaws that can be exploited to take over the nameservers.

Second, name-address translation in the DNS incurs long delays. Recent studies [41, 17, 19] have shown that DNS lookup time contributes more than one second for up to 30% of web object retrievals. The explosive growth of the namespace has decreased the effectiveness of DNS caching. The skewed distribution of names under popular domains, such as *.com*, has flattened the name hierarchy and increased load imbalance. The use of short timeouts for popular mappings, as is commonly employed by content distribution networks, further reduces DNS cache hit rates. Further, manual configuration errors, such as lame delegations [30, 28], can introduce latent performance problems.

Finally, widespread caching of mappings in the DNS prohibits fast propagation of unanticipated changes. Since the DNS does not keep track of cached copies of mappings, it cannot guarantee cache coherency and, instead relies on timeout-based invalidations of stale mappings. Lack of cache coherency in DNS implies that updates may not be visible to clients for extended periods of time, effectively preventing quick service relocation in response to attacks or emergencies.

SIGCOMM'04, Aug. 30–Sept. 3, 20034, Portland, Oregon, USA.

Fresh design of the legacy DNS provides an opportunity to address these shortcomings. A replacement for the DNS should exhibit the following properties.

- **High Performance**: Decouple the performance of DNS from the number of nameservers. Achieve lower latencies than legacy DNS and improve lookup performance in the presence of high loads and unexpected changes in popularity ("the slashdot effect").
- **Resilience to Attacks**: Remove vulnerabilities in the system and provide resistance against denial of service attacks through decentralization and dynamic load balancing. Self-organize automatically in response to host and network failures.
- **Fast Update Propagation**: Enable changes in name-address mappings to quickly propagate to clients. Support secure delegation to preserve integrity of DNS records, and prohibit rogue nodes from corrupting the system.

This paper describes Cooperative Domain Name System (CoDoNS), a backwards-compatible replacement for the legacy DNS that achieves these properties. CoDoNS combines two recent advances, namely, structured peer-to-peer overlays and analytically informed proactive caching. Structured peer-to-peer overlays, which create and maintain a mesh of cooperating nodes, have been used previously to implement wide-area distributed hash tables (DHTs). While their self-organization, scalability, and failure resilience provide a strong foundation for robust large-scale distributed services, their high lookup costs render them inadequate for demanding, latency-sensitive applications such as DNS [19]. CoDoNS achieves high lookup performance on a structured overlay through an analytically-driven proactive caching layer. This layer, called Beehive [33], automatically replicates the DNS mappings throughout the network to match anticipated demand and provides a strong performance guarantee. Specifically, Beehive achieves a targeted average lookup latency with a minimum number of replicas. Overall, the combination of Beehive and structured overlays provides the requisite properties for a large scale name service, suitable for deployment over the Internet.

Our vision is that globally distributed CoDoNS servers self-organize to form a flat peer-to-peer network, essentially behaving like a large, cooperative, shared cache. Clients contact CoDoNS through a local participant in the CoDoNS network, akin to a legacy DNS resolver. Since a complete takeover from DNS is an unrealistic undertaking, we have designed CoDoNS for an incremental deployment path. At the wire protocol level, CoDoNS provides full compatibility with existing DNS clients. No changes to client-side resolver libraries, besides changing the identities of the nameservers in the system configuration (e.g. modifying */etc/resolv.conf* or updating DHCP servers), are required to switch over to CoDoNS. At the back end, CoDoNS transparently builds on the existing DNS namespace. Domain names can be explicitly added to CoDoNS and securely managed by their owners. For names that have not been explicitly added, CoDoNS uses legacy DNS to acquire the mappings. CoDoNS subsequently maintains the consistency of these mappings by proactively checking with legacy DNS for updates. CoDoNS can thus grow as a layer on top of legacy DNS and act as a safety net in case of failures in the legacy DNS.

Measurements from a deployment of the system in Planet Lab [3] using real DNS workloads show that CoDoNS can substantially decrease the lookup latency, handle large flash-crowds, and quickly disseminate updates. CoDoNS can be deployed either as a complete replacement for DNS, where each node operates in a separate administrative and trust domain, or as an infrastructure service within an ISP, where all nodes are in the same administrative and trust domain.

The peer-to-peer architecture of CoDoNS securely decouples namespace management from a server's location in the network and enables a qualitatively different kind of name service. Legacy DNS relies fundamentally on physical delegations, that is, query handoffs from host to host until the query reaches a set of designated servers considered authoritative for a certain portion of the namespace owned by a namespace operator. Since all queries that involve that portion of the namespace are routed to these designated servers, the namespace operator is in a unique position of power. An unscrupulous namespace operator may abuse this monopoly by modifying records on the fly, providing differentiated services, or even creating synthetic responses that redirect clients to their own servers. Nameowners that are bound to that namespace have no other recourse. In contrast, name records in CoDoNS are tamper-proof and self-validating, and delegations are cryptographic. Any peer with a valid response can authoritatively answer any matching query. This decoupling of namespace management from the physical location and ownership of nameservers enables CoDoNS to delegate the same portion of the namespace, say *.com*, to multiple competing namespace operators. These operators, which are each provided with signing authority over the same space, assign names from a shared, coordinated pool, and issue self-validating name bindings into the system. Since CoDoNS eliminates physical delegations and designated nameservers, it breaks the monopoly of namespace operators and creates an even playing field where namespace operators need to compete with each other on service.

The rest of this paper is organized as follows. In the next section, we describe the basic operation of the legacy DNS and highlight its drawbacks. Section 3 describes the design and implementation of CoDoNS in detail. In Section 4, we present performance results from the PlanetLab deployment of CoDoNS. We summarize related work in Section 5, and conclude in Section 6.

2. DNS: OPERATION AND PROBLEMS

The Domain Name System (DNS) is a general-purpose database for managing host information on the Internet. It supports any kind of data, including network address, ownership, and service configuration, to be associated with hierarchically structured names. It is primarily used to translate human-readable names of Internet resources to their corresponding IP addresses. In this section, we provide a brief overview of the structure and operation of the legacy DNS, identify its major drawbacks, and motivate a new design.

2.1 Overview of Legacy DNS

The legacy DNS [26, 27] is organized as a static, distributed tree. The namespace is hierarchically partitioned into non-overlapping regions called *domains*. For example, *cs.cornell.edu* is a sub-domain of the domain *cornell.edu*, which in turn is a sub-domain of the top-level domain *edu*. Top-level domains are sub-domains of a global root domain.

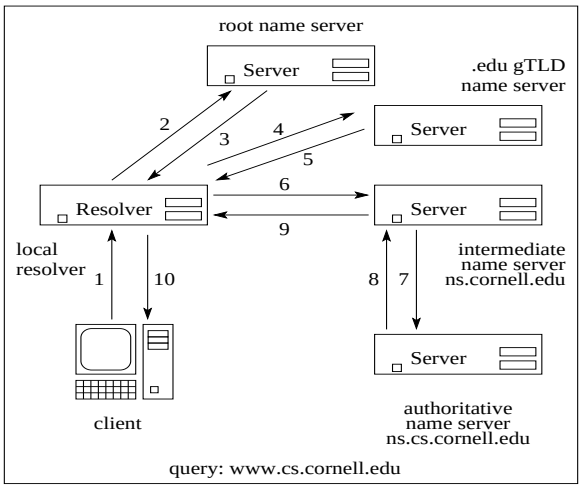

Figure 1: Name Resolution in Legacy DNS: Resolvers translate names to addresses by following a chain of delegations iteratively (2-5) or recursively (6-9).

Domain names, such as *www.cs.cornell.edu*, belong to *name-owners*.

Extensible data structures, called *resource records*, are used to associate values of different types with domain names. These values may include the corresponding IP address, mail host, owner name and the like. The DNS query interface allows these records to be retrieved by a query containing a domain name and a type.

The legacy DNS delegates the responsibility for each domain to a set of replicated *nameservers* called *authoritative nameservers*. The authoritative nameservers of a domain manage all information for names in that domain, keep track of authoritative nameservers of the sub-domains rooted at their domain, and are administered by *namespace operators*. At the top of the legacy DNS hierarchy are *root nameservers*, which keep track of the authoritative nameservers for the *top-level domains* (TLDs). The top-level domain namespace consists of generic TLDs (gTLD), such as *.com, .edu, and .net*, and country-code TLDs (ccTLD), such as *.uk, .tr, and .in*. Nameservers are statically configured with thirteen IP addresses for the root servers. BGP-level anycast is used in parts of the Internet to reroute queries destined for these thirteen IP addresses to a local root server.

Resolvers in the legacy DNS operate on behalf of clients to map queries to matching resource records. Clients typically issue DNS queries to local resolvers within their own administrative domain. Resolvers follow a chain of authoritative nameservers in order to resolve the query. The local resolver contacts a root nameserver to find the top-level domain nameserver. It then issues the query to the TLD nameserver and obtains the authoritative nameserver of the next sub-domain. The authoritative nameserver of the sub-domain replies with the response for the query. This process continues recursively or iteratively until the authoritative nameserver of the queried domain is reached. Figure 1 illustrates the different stages in the resolution of an example domain name *www.cs.cornell.edu*. While this figure provides a simple overview of the communication involved in name resolution, in practice, each query may trigger additional lookups to resolve intermediate nameservers [26, 27].

Bottlenecks	All Domains	Top 500
1	0.82 %	0.80 %
2	78.44 %	62.80 %
3	9.96 %	13.20 %
4	4.64 %	13.00 %
5	1.43 %	6.40 %
13	4.12 %	0 %

Table 1: Delegation Bottlenecks in Name Resolution: A significant number of names are served by two or fewer nameservers, even for the most popular 500 sites.

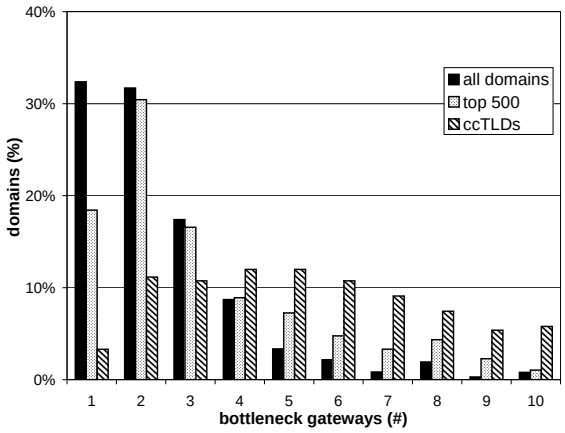

Figure 2: Physical Bottlenecks in Name Resolution: A significant number of domains, including top-level domains, depend on a small number of gateways for their resolution.

Pursuing a chain of delegations to resolve a query naturally incurs significant delay. The legacy DNS incorporates aggressive caching in order to reduce the latency of query resolution. The resolvers cache responses to queries they issue, and use the cached responses to answer future queries. Since records may change dynamically, legacy DNS provides a weak form of cache coherency through a *time-to-live* (TTL) field. Each record carries a TTL assigned by the authoritative nameserver, and may be cached until TTL expires.

2.2 Problems with Legacy DNS

The current use and scale of the Internet has exposed several shortcomings in the functioning of the legacy DNS. We performed a large scale survey to analyze and quantify its vulnerabilities. Our survey explored the delegation chains of 593160 unique domain names collected by crawling the Yahoo! and the DMOZ.ORG web directories. These domain names belong to 535088 unique domains and are served by 164089 different nameservers. We also separately examined the 500 most popular domains as determined by the Alexa ranking service. In this section, we describe the findings of our survey, which highlight the problems in failure resilience, performance, and update propagation in the legacy DNS.

Failure Resilience - Bottlenecks

The legacy DNS is highly vulnerable to network failures, compromise by malicious agents, and denial of service attacks, because domains are typically served by a very small

number of nameservers. We first examine the *delegation bottlenecks* in DNS; a delegation bottleneck is the minimum number of nameservers in the delegation chain of each domain that need to be compromised in order to control that domain. Table 1 shows the percentage of domains that are bottlenecked on different numbers of nameservers. 78.63% of domains are restricted by two nameservers, the minimum recommended by the standard [26]. Surprisingly, 0.82% of domains are served by only one nameserver. Even the highly popular domains are not exempt from severe bottlenecks in their delegation chains. Some domains (0.43%) spoof the minimum requirement by having two nameservers map to the same IP address. Overall, over 90% of domain names are served by three or fewer nameservers and can be disabled by relatively small-scale DoS attacks.

Failure and attack resilience of the legacy DNS is even more limited at the network level. We examined *physical bottlenecks*, that is, the minimum number of network gateways or routers between clients and nameservers that need to be compromised in order to control that domain. We measured the physical bottlenecks by performing traceroutes to 10,000 different nameservers, which serve about 5,000 randomly chosen domain names, from fifty globally distributed sites on PlanetLab [3]. Figure 2 plots the percentage of domains that have different numbers of bottlenecks at the network level, and shows that about 33% of domains are bottlenecked at a single gateway or router. While this number is not surprising - domains are typically served by a few nameservers, all located in the same sub-network - it highlights that a large number of domains are vulnerable to network outages. These problems are significant and affect many top level domains and popular web sites. Recently, Microsoft suffered a DoS attack on its nameservers that rendered its services unavailable. The primary reason for the success of this attack was that all of Microsoft's DNS servers were in the same part of the network [39]. Overall, a large portion of the namespace can be compromised by infiltrating a small number of gateways or routers.

Failure Resilience - Implementation Errors

The previous section showed that legacy DNS suffers from limited redundancy and various bottlenecks. In this section, we examine the feasibility of attacks that target these bottlenecks through known vulnerabilities in commonly deployed nameservers. Early studies [11, 23, 28] identified several implementation errors in legacy DNS servers that can lead to compromise. While many of these have been fixed, a significant percentage of nameservers continue to use buggy implementations. We surveyed 150,000 nameservers to determine if they contain any known vulnerabilities, based on the Berkeley Internet Name Daemon (BIND) exploit list maintained by the Internet Systems Consortium (ISC) [18]. Table 2 summarizes the results of this survey. Approximately 18% of servers do not respond to version queries, and about 14% do not report valid BIND versions. About 2% of nameserves have the *tsig* bug, which permits a buffer overflow that can enable malicious agents to gain access to the system. 19% of nameserves have the *negcache* problem that can be exploited to launch a DoS attack by providing negative responses with large TTL value from a malicious nameserver. Overall, exploiting the bottlenecks identified in the previous section is practical.

problem	severity	affected nameservers	
		all domains	top 500
tsig	critical	2.08 %	0.59 %
nxt	critical	0.09 %	0.15 %
negcache	serious	19.03 %	2.57 %
sigrec	serious	13.56 %	1.32 %
DoS multi	serious	11.11 %	1.32 %
DoS findtype	serious	2.58 %	0.59 %
srv	serious	1.89 %	0.59 %
zxfr	serious	1.81 %	0.44 %
libresolv	serious	1.48 %	0 %
complain	serious	1.33 %	0 %
so-linger	serious	1.15 %	0.15 %
fdmax	serious	1.15 %	0.15 %
sig	serious	0.70 %	0.15 %
infoleak	moderate	4.58 %	0.59 %
sigdiv0	moderate	1.86 %	0.59 %
openssl	medium	1.71 %	0.37 %
naptr	minor	2.58 %	0.15 %
maxdname	minor	2.58 %	0.15 %

Table 2: Vulnerabilities in BIND: A significant percentage of nameservers use BIND versions with known security problems [18].

Performance - Latency

Name resolution latency is a significant component of the time required to access web services. Wills and Shang [41] have found, based on NLANR proxy logs, that DNS lookup time contributes more than one second to 20% of web object retrievals, Huitema et al. [17] report that 29% of queries take longer than two seconds, and Jung et al. [19] show that more than 10% of queries take longer than two seconds. The low performance is due mainly to low cache hit rates, stemming from the heavy-tailed, Zipf-like query distribution in DNS. It is well known from studies on Web caching [7] that heavy-tailed query distributions severely limit cache hit rates.

Wide-spread deployment of content distribution networks, which perform dynamic server selection, have further strained the performance of the legacy DNS. These services, such as Akamai and Digital Island, use the DNS in order to direct clients to closer servers of Web content. They typically use very short TTLs (on the order of 30 seconds) in order to perform fine grain load balancing and respond rapidly to changes in server or network load. But, this mechanism virtually eliminates the effectiveness of caching and imposes enormous overhead on DNS. A study on impact of short TTLs on caching [20] shows that cache hit rates decrease significantly for TTLs lower than fifteen minutes. Another study on the adverse effect of server selection [36] reports that name resolution latency can increase by two orders of magnitude.

Performance - Misconfigurations

DNS performance is further affected by the presence of a large number of broken (lame) or inconsistent delegations. In our survey, address resolution failed for about 1.1% of nameservers due to timeouts or non-existent records, mostly stemming from spelling errors. For 14% of domains, authoritative nameservers returned inconsistent responses; a few authoritative nameservers reported that the domain does

not exist, while others provided valid records. Failures stemming from lame delegations and timeouts can translate into significant delays for the end-user. Since these failures and inconsistencies largely stem from human errors [28], it is clear that manual configuration and administration of such a large scale system is expensive and leads to a fragile structure.

Performance - Load Imbalance

DNS measurements at root and TLD nameservers show that they handle a large load and are frequently subjected to denial of service attacks [5, 6]. A massive distributed DoS attack [29] in November 2002 rendered nine of the thirteen root servers unresponsive. Partly as a result of this attack, the root is now served by more than sixty nameservers and is served through special-case support for BGP-level anycast. While this approach fixes the superficial problem at the topmost level, the static DNS hierarchy fundamentally implies greater load at the higher levels than the leaves. The special-case handling does not provide automatic replication of the hot spots, and sustained growth in client population will require continued future expansions. In addition to creating exploitable vulnerabilities, load imbalance poses performance problems, especially for lookups higher in the name hierarchy.

Update Propagation

Large-scale caching in DNS poses problems for maintaining the consistency of cached records in the presence of dynamic changes. Selection of a suitable value for the TTL is an administrative dilemma; short TTLs adversely affect the lookup performance and increase network load [19, 20], while long TTLs interfere with service relocation. For instance, a popular on line brokerage firm uses a TTL of thirty minutes. Its users do not incur DNS latencies when accessing the brokerage for thirty minutes at a time, but they may experience outages of up to half an hour if the brokerage firm needs to relocate its services in response to an emergency. Nearly 40% of domain names use TTLs of one day or higher, which prohibits fast dissemination of unanticipated changes to records.

3. COOPERATIVE DOMAIN NAME SYSTEM

The use and scale of today's Internet is drastically different from the time of the design of the legacy DNS. Even though the legacy DNS anticipated the explosive growth and handled it by partitioning the namespace, delegating the queries, and widely caching the responses, this architecture contains inherent limitations. In this section, we present an overview of CoDoNS, describe its implementation, and highlight how it addresses the problems of the legacy DNS.

3.1 Overview of Beehive

CoDoNS derives its performance characteristics from a proactive caching layer called Beehive [33]. Beehive is a proactive replication framework that enables prefix-matching DHTs to achieve $O(1)$ lookup performance. Pastry [35], and Tapestry [42] are examples of structured DHTs that use prefix-matching [32, 21] to lookup objects. In these DHTs, both objects and nodes have randomly assigned identifiers from the same circular space, and each object is stored at the nearest node in the identifier space, called the *home node*. Each node routes a request for an object, say 2101, by successively matching prefixes; that is, by routing the request to a node that matches one more digit with the object until the home node, say 2100, is reached. Overlay routing by matching prefixes in this manner incurs $O(\log N)$ hops in the worst case to reach the home node. Figure 3 illustrates the prefix matching routing algorithm in Pastry. A *routing table* of $O(\log N)$ size provides the overlay node with pointers to nodes with matching prefixes. In a large system, $\log N$ translates to several hops across the Internet and is not sufficient to meet the performance requirements of latency critical applications such as DNS.

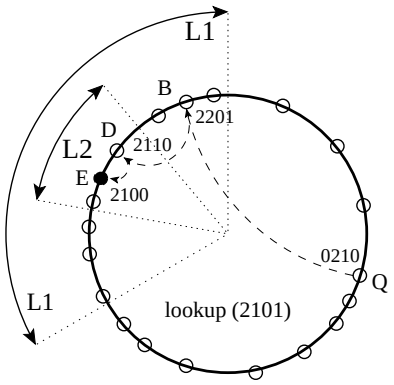

Figure 3: Proactive Caching in Beehive: Caching an object at all nodes with i matching prefix-digits ensures that it can be located in i hops. Beehive achieves $O(1)$ average lookup time with minimal replication of objects.

Beehive proposes a novel technique based on controlled proactive caching to reduce the average lookup latency of structured DHTs. Figure 3 illustrates how Beehive applies proactive caching to decrease lookup latency in Pastry. In the example mentioned above, where a query is issued for the object 2101, Pastry incurs three hops to find a copy of the object. By placing copies of the object at all nodes one hop prior to the home node in the request path, the lookup latency can be reduced by one hop. In this example, the lookup latency can be reduced from three hops to two hops by replicating the object at all nodes that start with 21. Similarly, the lookup latency can be reduced to one hop by replicating the object at all nodes that start with 2. Thus, we can vary the lookup latency of the object between 0 and $\log N$ hops by systematically replicating the object more extensively. In Beehive, an object replicated at all nodes with i matching prefixes incurs i hops for a lookup, and is said to be replicated at *level* i.

The central insight behind Beehive is that by judiciously choosing different levels of replication for different objects, the average lookup performance of the system can be tuned to any desired constant. Naturally, replicating every object at every node would achieve $O(1)$ lookups, but would incur excessive space overhead, consume significant bandwidth and lead to large update latencies. Beehive minimizes bandwidth and space consumption by posing the following optimization problem: minimize the total number of replicas subject to the constraint that the aggregate lookup latency is less than a desired constant C. For power-law (or Zipf-like) query distributions, Beehive analytically derives the optimal

closed-form solution to this problem. The derivation of the analytical solution is provided in [33]; the final expression for the closed-form solution that minimizes the total number of replicas for Zipf-like query distributions with parameter $\alpha < 1$ is the following:

$$x_i = [\frac{d^i(logN - C)}{1 + d + \cdots + d^{logN-1}}]^{\frac{1}{1-\alpha}}, \text{where } d = b^{\frac{1-\alpha}{\alpha}}$$

In this expression, b is the base of the underlying DHT and x_i is the fraction of most popular objects that get replicated at level i. This solution is immediately applicable to DNS, since DNS queries follow a Zipf-like distribution [19].

The analytical result provides several properties suited for latency-sensitive applications such as DNS. First, it succinctly captures the space-time tradeoff and enables applications to achieve any targeted average lookup latency by selecting an appropriate C. In CoDoNS, we set Beehive's target as $C = 0.5$ hops, which means that a large percentage of requests are answered immediately without having to take any extra network hops. Second, it incurs minimal bandwidth and storage overhead by picking the optimal number of replicas required to achieve the target lookup performance. Further, replicating objects across several nodes balances the load on individual Beehive nodes, reduces hotspots and improves resilience against DoS attacks. Finally, the level of replication for each object can be used to quickly determine the location of all object replicas, and to update them when necessary.

Beehive nodes need to know only the Zipf parameter and the relative popularity rank of objects to apply the closed-form solution and determine the extent of replication for each object. Beehive employs a combination of local measurement and limited aggregation to estimate the Zipf parameter and the popularity ranks of objects. Beehive nodes locally measure the access frequency of each object, and periodically aggregate them with other nodes every *aggregation interval*. Each node aggregates values gathered from nodes one level higher in the routing table. Eventually, the aggregates trickle down through different levels of the routing table and reach the home node. The home node computes a final aggregate and propagates it to all replicas in the system. The Zipf parameter is locally derived from the aggregate access frequencies of the object and fed to the analytical model. Using these estimates, each Beehive node invokes the analytical model once every *analysis interval* and obtains the appropriate replication levels for objects it stores. The replication of these objects to the specified levels is then performed by the replication protocol. Replication in Beehive is controlled locally; that is, each node is responsible for replicating an object on nodes at most one hop away from itself. For example, the home node of a popular object replicates it at nodes that share one prefix less. Those nodes then independently decide to replicate that object further to one more level. In the *replication phase*, each node exchanges messages with nodes in its routing table, which are one hop away from them, to push, delete, or update replicas for which they are responsible.

The aggregation and replication protocols enable Beehive to quickly detect changes in the demand for objects. Large changes in the popularity of domain names occur during denial of service attacks and flash crowds. Beehive nodes constantly monitor the access frequency of objects and adjust the extent of replication. In response to DoS attacks, they promptly increase the number of replicas and spread the load among several nodes, curbing the attack.

Proactive replication also enables Beehive to rapidly push updates to all the replicas in the system. In general, proactive propagation of updates demands expensive mechanisms to keep track of all the nodes where the object is cached. Beehive requires just a small integer, the replication level of an object, to determine the range of nodes with replicas of the object. An object at level i is replicated at all nodes with i matching prefix digits. For a level i object, the home node propagates updates to all the nodes at level i in the routing table. Those nodes in turn propagate updates to nodes at level $i + 1$. This recursive process disseminates the updates to nodes with i matching prefix digits. Nodes in the process of joining the DHT may miss the initial update propagation. Such nodes will receive the latest copy of the object in the next replication phase; they may incur a slight performance penalty, but will not serve stale data. Proactive update propagation obviates the need for timeout-based caching.

3.2 CoDoNS: Architecture

CoDoNS consists of globally distributed nodes that self organize to form a peer-to-peer network. We envisage that each institution would contribute one or more servers to CoDoNS, forming a large-scale, cooperative, globally shared DNS cache. CoDoNS provides query resolution services to clients using the same wire format and protocol as legacy DNS, and thus requires no changes to client resolvers.

CoDoNS decouples namespace management from query resolution of the legacy DNS. Nameowners need only to purchase certificates for names from namespace operators and introduce them into CoDoNS; they do not need to provide dedicated hosts for serving those names. CoDoNS places no restrictions on the hierarchical structure of the namespace and is agnostic about the administrative policies of the nameowners. To the nameowners, CoDoNS provides an interface consisting of insert, delete and update.

CoDoNS associates the node whose identifier is closest to the consistent hash [21] of the domain name as the *home node* for that domain name. The home node stores a permanent copy of the resource records owned by that domain name and manages their replication. If the home node fails, the next closest node in the identifier space automatically becomes the new home node. CoDoNS replicates all records on several nodes adjacent to the home node in the identifier space in order to avoid data loss due to node failures.

Replacing the DNS entirely with CoDoNS is an ambitious plan, and we do not expect nameowners to immediately use CoDoNS for propagating their information. In order to allow CoDoNS to gradually grow into a globally recognized system, we have incorporated compatibility to the legacy DNS. CoDoNS uses the legacy DNS to resolve queries for records not explicity inserted by nameowners. The home node retrieves resource records from the legacy DNS upon the first query for those records. The additional redirection latency only affects the first query issued in the entire system for a domain name. CoDoNS decreases the impact of query redirection on lookup performance, by bootstrapping the system with records obtained from legacy DNS nameservers through zone transfers or file transfers.

Overall, query resolution in CoDoNS takes place as follows. Client sends a query in the wire format of the legacy

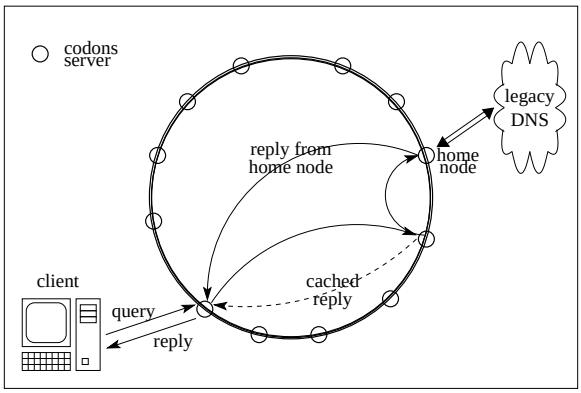

Figure 4: CoDoNS Deployment: CoDoNS servers self-organize to form a peer-to-peer network. Clients send DNS requests to a local CoDoNS server, which obtains the records from the home node or an intermediate node, and responds to the client. In the background, the home nodes interact with the legacy DNS to keep records fresh and propagate updates to cached copies.

DNS to the local CoDoNS server in the same administrative domain. The local CoDoNS server replies immediately if it has a cached copy of the requested records. Otherwise, it routes the query internally in the CoDoNS network using the under-lying DHT. The routing terminates either at an intermediate CoDoNS node that has a cached copy of the records or at the home node of the domain name. The home node retrieves the records from the legacy DNS, if it does not already have it, and sends a response to the first contacted CoDoNS server, which replies to the client. In the background, CoDoNS nodes proactively replicate the records in based on the measured popularity. Figure 4 shows a typical deployment of CoDoNS and illustrates the process of query resolution.

Clients generate a large number of queries for names in their local administrative domain. Since the home node of a name may be located in a different domain, local queries can incur extra latency and impose load on wide-area network links. CoDoNS supports efficient resolution of local names through *direct caching*. Nameowners can directly insert, update, and delete their records at CoDoNS servers in their administrative domain, and configure the local CoDoNS servers to use the direct cache for replying to local queries.

3.3 CoDoNS: Implementation

CoDoNS servers are layered on top of Pastry and Beehive. Each CoDoNS server implements a complete, recursive, caching DNS resolver and supports all requirements described in the specification [26, 27]. CoDoNS also supports inverse queries that map IP addresses to a domain name by inserting reverse address-name records into the DHT when name-address records are introduced.

Domain names in CoDoNS have unique 128 bit identifiers obtained through the SHA-1 hashing algorithm. The home node, the closest node in the identifier space, stores permanent copies of the resource records of the domain name and maintains their consistency in the system. Since CoDoNS does not associate TTLs with the records, the home nodes push the updates to all replicas in the system, which retain them until the replication level of the record is downgraded, or until an update is received. Nameowners insert updated resource records into CoDoNS, and the home nodes proactively propagate the updates.

CoDoNS ensure the consistency of records obtained from the legacy DNS, CoDoNS by proactively refetching them. The home node uses the TTL specified by the legacy DNS as the duration to store the records. It refetches the records from legacy DNS after TTL duration, and propagates the updated records to all the replicas if the records change. Since CoDoNS performs the refetches in the background, its lookup performance is not affected. The TTL values are rounded up to a minimum of thirty seconds; records with lower TTL values are not placed into the system. Low TTL values typically indicate dynamic server-selection in legacy DNS. The home node prompts the server that injected the query to consult the legacy DNS server by issuing a special error-response. This redirection of queries for low-TTL records ensures that services that rely on dynamic server selection will continue to work, and reduces overhead on the CoDoNS home nodes.

The legacy DNS relies on error responses, called *NXDOMAIN* responses, to detect names that do not exist. Since clients reissue a request several times when they do not receive prompt replies, the DNS specification recommends that resolvers cache NXDOMAIN responses. CoDoNS provides complete support for negative caching as described in [1]. However, permanently storing NXDOMAIN responses could exhaust the capacity of the system, since an unlimited number of queries can be generated for non-existent domains. Hence, CoDoNS nodes cache NXDOMAIN responses temporarily and do not refresh them upon expiry.

3.4 Issues and Implications

CoDoNS decouples namespace management from the physical location of nameservers in the network. Instead of relying on physical delegations to trusted hosts and assuming that Internet routing is secure, CoDoNS uses cryptographic delegations and self-verifying records based on the DNSSEC [13] standard.

DNSSEC uses public key cryptography to enable authentication of resource records. Every namespace operator has a public-private key pair; the private key is used to digitally sign DNS records managed by that operator, and the corresponding public key is in turn certified by a signature from a domain higher up in the hierarchy. This process creates a chain of certificates, terminating at a small number of well-known public keys for globally trusted authorities. Since records are signed at the time of issue, the private keys need not be kept online. The signature and the public key are stored in DNS as resource records of type *sig* and *key* respectively. Clients can verify the authenticity of a resource record by fetching the sig record and the key record from the DNS.

The use of cryptographic certificates enables any client to check the verity of a record independently, and keeps peers in the network from forging certificates. To speed up certificate verification, CoDoNS servers cache the certificates along with the resource records and provide them to the clients. Existing clients that are not DNSSEC compliant need to trust only the local CoDoNS servers within their administrative domain, since CoDoNS servers internally verify data fetched from other nodes.

CoDoNS authenticates nameowners directly through certificates provided for every insertion, delete, and update. Insertions simply require a signed resource record with a corresponding well-formed certificate. A version number associated with each record, signed by the owner and checked by every server, ensures that old records cannot be reintroduced into the system. Deletions require a signed request that identifies the record to be expunged, while updates introduce a new signed, self-verifying record that replaces the now-stale version.

Since CoDoNS removes authority from the identity and location of the server providing resource records and vests it with cryptographic keys, it provides a greater degree of freedom over namespace management. CoDoNS enables multiple namespace operators to manage the same part of the name hierarchy. A domain owner can delegate management of the same sub-domain to multiple operators by endorsing their keys as being authoritative for that sub-domain. Ideally, competing operators would preserve a single consistent namespace by issuing names out of a common, shared pool. In the presence of conflicting or inconsistent records, clients simply pick the records signed by an operator they trust, similar to the way they pick between separate sets of root servers today. Essentially, nameowners have the ability to choose the namespace operator that will endorse their records based on price, service and functionality.

Since DNSSEC has not yet been widely deployed in the Internet, CoDoNS cannot rely on the legacy DNS to provide certificates for resource records. Consequently, CoDoNS uses its own centralized authority to sign resource records fetched from the legacy DNS. Queries to the legacy DNS are directed to a small pool of *certifying resolvers*, which fetch authoritative resource records from the legacy DNS, sign them, and append the sig records to the legacy DNS response. This approach requires trust to be placed in the certifying resolvers. Threshold cryptography [43] can be used to limit the impact of adversaries on these resolvers until CoDoNS takes over completely. The certifying name resolvers ensure that CoDoNS participants cannot inject corrupted records into the system.

Malicious participants may also disrupt the system by corrupting the routing tables of peers and misrouting or dropping queries. Castro et al. [8] propose a method to handle routing table corruptions in DHTs. This scheme augments the regular routing table with a secure routing table where the entries need to satisfy strict constraints on node identifiers that limit the impact of corrupt nodes. Since nodes in the secure routing table are not picked based on short network latencies, this scheme may increase the lookup delay. Setting a lower target latency at the Beehive layer can compensate for the increase in lookup latency at the cost of bandwidth and storage.

CoDoNS acts as a large cache for stored, self-verifying records. This design, which separates namespace management from the physical servers, prohibits dynamic name resolution techniques where the mapping is determined as a result of a complex function, evaluated at run time. In the general case, such functions take arbitrary inputs and have confidentiality requirements that may prohibit them from being shipped into the system. For instance, content distribution networks, such as Akamai, use proprietary techniques to direct clients to servers [4, 36]. To nevertheless support such dynamic mapping techniques, CoDoNS enables nameowners to stipulate redirections of queries for certain names using a special *redirection record*. High lookup performance during redirections is ensured through proactive replication and update of the redirection record in the same manner as regular resource records.

	Parameter	Value
Pastry	base	16
	leaf-set size	24
Beehive	target C	0.5 hops
	aggregation interval	6 min
	analysis interval	60 min

Table 3: Parameters for Pastry and Beehive

As with any peer-to-peer system, CoDoNS relies on its participants to contribute resources on behalf of others. While it may seem, at first, that rational actors might be averse to participating in the system for fear of having to serve as home nodes for highly popular records, proactive replication ensures that the load perceived by all nodes is comparable. A highly popular record will be replicated until the load it projects on its home node is comparable to the query load for other records.

4. EVALUATION

We have deployed CoDoNS on PlanetLab [3], an open platform for developing, deploying, and accessing planetary-scale services. PlanetLab enables us to deploy CoDoNS on servers around the world and evaluate it against the background of real Internet with congestion, losses, and unpredictable failures. In this section, we present performance measurements from the PlanetLab deployment for a real DNS workload. Our experiments highlight three important properties of CoDoNS. First, they show that CoDoNS provides a low latency name resolution service. Second, they demonstrate CoDoNS' ability to resist flash-crowds by quickly spreading the load across multiple servers. Finally, they evaluate CoDoNS' support for fast update propagation.

4.1 Setup

We setup a peer-to-peer network of CoDoNS servers on globally distributed PlanetLab nodes. The values used for different parameters of Pastry and Beehive are listed in Table 3. We started the CoDoNS servers with no initial DNS records. After an initial quiescent period to stabilize Pastry, we issue DNS requests from a real workload to the CoDoNS server at each node. During the experiment, we measure the lookup latency of CoDoNS, and periodically record the load handled and overhead incurred by each node. We also apply the same workload to the legacy DNS, and measure its performance.

We obtained the DNS workload from traces collected at MIT between the 4^{th} and the 11^{th} of December 2000 [19]. Our workload comprises of the first 12 hours of this trace, with $281,943$ total queries for $47,230$ unique domain names. The popularity of the domain names in this workload closely follows a Zipf-like distribution with parameter 0.91. We divided this workload uniformly and issued DNS requests to the local CoDoNS server at each node. The measurements reported in this paper were taken from a deployment on 75 geographically distributed PlanetLab nodes.

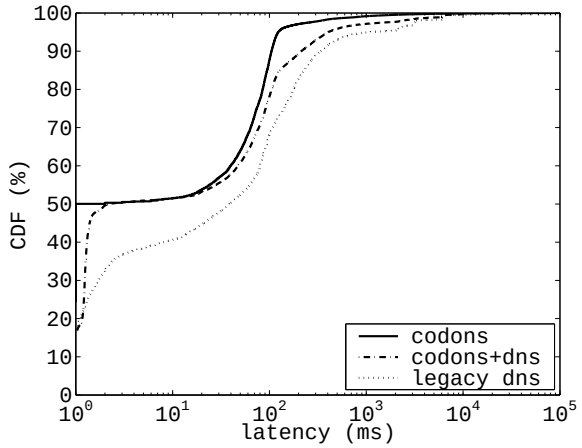

Figure 5: Cumulative Distribution of Latency: CoDoNS achieves low latencies for name resolution. More than 50% of queries incur no network delay as they are answered from the local CoDoNS cache.

Latency	Mean	Median	90^{th} %
CoDoNS	106 ms	1 ms	105 ms
CoDoNS+DNS	199 ms	2 ms	213 ms
Legacy DNS	382 ms	39 ms	337 ms
PlanetLab RTT	121 ms	82 ms	202 ms

Table 4: Query Resolution Latency: CoDoNS provides low latency name resolution through analytically informed proactive caching.

4.2 Lookup Performance

Figure 5 shows the cumulative distribution of lookup latencies incurred by CoDoNS and the legacy DNS. Table 4 summarizes the results of Figure 5 by providing the median, mean, and the 90^{th} percentile of the latency distribution. We aggregate the latency during the second half of the workload, allowing the first half to warm the caches of both CoDoNS and the legacy DNS. Note that the second half of the workload also contains DNS requests for domain names not present in the cache, and CoDoNS incurs the extra latency of redirecting these queries to the legacy DNS. In order to study the impact of legacy DNS redirections on latency, we separately evaluate the lookup performance of CoDoNS without redirections, by inserting the records at their home nodes before applying the work load. This study essentially evaluates the scenario after a complete take over of the legacy DNS by CoDoNS.

50% of the queries in CoDoNS are answered immediately by the local CoDoNS server without incurring network delay, since proactive replication pushes responses for the most popular domain names to all CoDoNS servers. Consequently, CoDoNS provides a significant decrease in median latency to about 2 milliseconds compared to about 39 milliseconds for the legacy DNS. The tail of the latency distribution indicates that cache misses leading to legacy DNS lookups have an impact on the worst-case lookup performance of CoDoNS. However, a complete take over from the legacy DNS would obviate the extra latency overhead. Overall, CoDoNS achieves low latencies in the mean, median, and the 90^{th} percentile, for both deployment scenarios, with and without redirections to the legacy DNS.

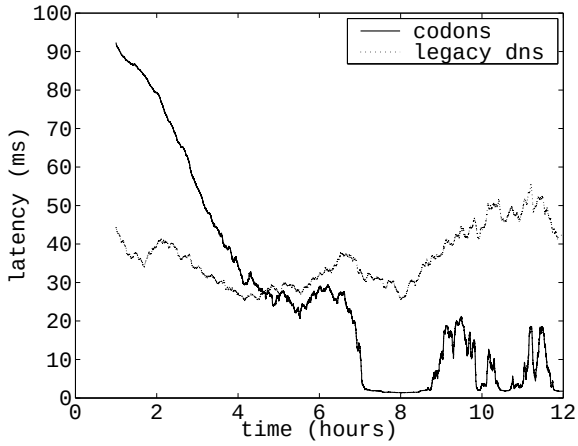

Figure 6: Median Latency vs Time: Lookup latency of CoDoNS decreases significantly as proactive caching takes effect in the background.

Figure 6 shows the median latency of CoDoNS and the legacy DNS over time. The fluctuations in the graph stem from the changing relative popularities of names in the workload over time. CoDoNS reacts to these changes by continuously adjusting the extent of proactive caching. Initially, CoDoNS servers have an empty cache and redirect most of the queries to legacy DNS. Consequently, they incur higher latencies than the legacy DNS. But as resource records are fetched from legacy DNS and replication in the background pushes records to other CoDoNS servers, the latency decreases significantly. The initial surge in latency can be easily avoided by bootstrapping the system with records for well known domain names.

4.3 Flash-crowd Effect

Next, we examine the resilience of CoDoNS to sudden upheavals in the popularity of domain names. To model a flash-crowd effect, we take the DNS workload and modify the second half to reflect large scale changes in the popularity of all domain names. We achieve this by completely reversing the popularities of all the domain names in the workload. That is, the least popular name becomes the most popular name, the second least popular name becomes the second most popular name, and so on. This represents a worst case scenario for CoDoNS because records that are replicated the least suddenly need to be replicated widely, and vice versa, simulating, in essence, a set of flash crowds for the least popular records.

Figure 7 shows the median resolution latencies in CoDoNS during the flash-crowd effect introduced at the six hour mark. There is a temporary increase in the median latency of CoDoNS when flash-crowd effect starts. But, Beehive's proactive replication in the background detects the changes in popularity, adjusts the number of replicas, and decreases the lookup latency. The latency of CoDoNS after popularity reversal quickly reaches the low values in Figure 6, indicating that CoDoNS has recovered completely from the worst-case, large scale changes in popularity.

4.4 Load Balance

We evaluate the automatic load-balancing provided by proactive replication in CoDoNS by quantifying load bal-

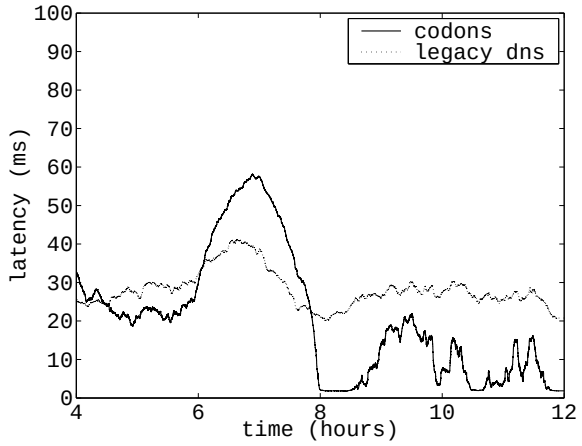

Figure 7: Median Latency vs Time as a flash-crowd is introduced at 6 hours: CoDoNS detects the flash-crowd quickly and adapts the amount of caching to counter it, while continuing to provide high performance.

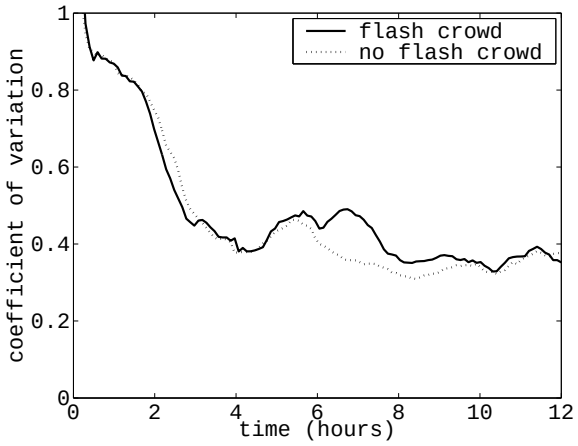

Figure 8: Load Balance vs Time: CoDoNS handles flash-crowds by balancing the query load uniformly across nodes. The graph shows load balance as a ratio of the standard deviation to the mean across all nodes.

ance using the *coefficient of variation*, defined as the ratio of the standard deviation of the load across all the nodes to the mean load. The overall average of query load is about 6.5 per second for the system.

Figure 8 shows the load balance in queries handled by CoDoNS servers, either from their internal cache or by querying the legacy DNS, for the duration of the workload. At the start of the experiment, the query load is highly unbalanced, since home nodes of popular domain names receive far greater number of queries than average. The imbalance is significantly reduced as the records for popular domains get replicated in the system. Even when a flash-crowd is introduced at the six hour mark, dynamic changes in caching keep the load balanced after a temporary increase in load variance. Overall, continuous monitoring and adaptation of proactive caching enable CoDoNS to respond to drastic changes in the popularity of names and handle flash crowds.

The network bandwidth and per-node storage costs incurred by proactive caching are modest. The average bandwidth consumed over the entire experiment was 12.2 KB/s per node (std. dev. 2.26 KB/s) for all network activities. The average number of records per node was 4217 (std. dev. 348), a mere 10% of the total number of records. These records require, on average, 13 MB per node. These measurements indicate that CoDoNS distributes the load evenly across the system and incurs low uniform bandwidth and storage overhead at each node.

4.5 Update Propagation

Next we examine the latencies incurred by CoDoNS for proactive update propagation. Figure 9 shows the delay incurred for disseminating updates to resource records replicated at different levels. 98% of the replicas are updated within one second even for level-0 records, which are replicated at all nodes in the system. It takes a few seconds longer to update some replicas due to high variance in network delays and loads at some hosts. The latency to update 99% of replicas one hop from the home node is about one second. Overall, update propagation latency in CoDoNS depends on the extent of replication of records. In the worst case, it takes $\log N$ hops to update all the nodes in the network. For a million node CoDoNS network, updating 99% of replicas would take far less than a minute for even the most popular domain names replicated throughout. This enables nameowners to relocate their services without noticeable disruptions to their clients.

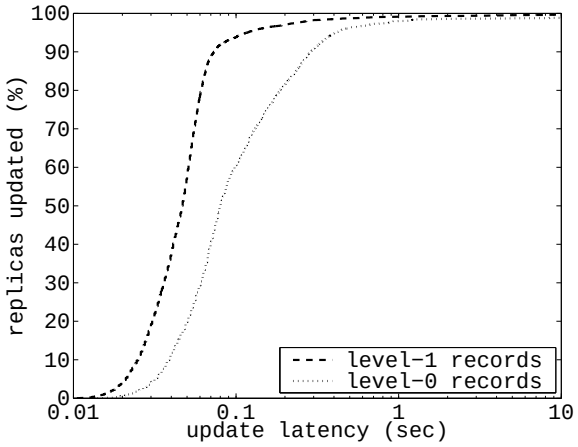

Figure 9: Update Propagation Time: CoDoNS incurs low latencies for propagating updates. 98% of replicas get updated within one second.

4.6 Summary

Performance measurements from a planetary-scale deployment against a real workload indicate that CoDoNS can provide low latencies for query resolution. Massive replication for the most popular records, but a modest number of replicas per server, achieves high performance with low overhead. Eliminating the static query processing hierarchy and shedding load dynamically onto peer nodes greatly decreases the vulnerability of CoDoNS to denial of service attacks. Self-organization and continuous adaptation of replication avoids bottlenecks in the presence of flash crowds. Proactive update propagation ensures that unanticipated changes can be quickly disseminated and cached in the system.

5. RELATED WORK

Several researchers have extensively studied the performance of the legacy DNS, and proposed new mechanisms to improve its performance. In this section, we discuss past approaches to measure and improve the legacy DNS.

Performance Studies: In 1988, Mockapetris and Dunlap published a retrospective study on the development of the legacy DNS identifying its successful features and shortcomings [28]. Several measurement studies since then have provided good insight into the advantages and the drawbacks of the system. A recent survey by Pappas et al. [30] studies the impact of lame delegations, cyclic dependencies and reduced nameserver redundancies, on the performance of the legacy DNS; their findings confirm the results of our survey. Earlier studies by Danzig et al. [11] and Brownlee et al. [5, 6] analyze DNS traffic at root and gTLD nameservers. Huitema et al. [17] and Wills et al. [41] study DNS performance observed from the client side. A more detailed study of the performance experienced by clients and analysis of the effectiveness of caching in the legacy DNS is presented by Jung et al. in [19, 20]. These studies have provided invaluable insight into the operation of the legacy DNS and helped us to motivate and design CoDoNS.

Design Alternatives: Recently, a few schemes have been proposed to improve the failure-resilience and lookup performance of DNS. Cohen and Kaplan [9] propose a proactive caching scheme for DNS records. In their scheme, expired DNS records in the cache are proactively fetched from the authoritative nameservers. They analyze several heuristics-based prefetching algorithms, and evaluate their performance. This scheme entails prefetching at intermediate caches, which generates substantial amount of background traffic to update DNS records. In contrast, CoDoNS fetches each record only once at the home node, significantly reducing the overhead imposed on the legacy DNS.

CoDNS [31] masks delays in the legacy DNS caused by failures in local resolvers by diverting queries to other, healthy resolvers. CoDNS provides resilience against temporary problems with the legacy DNS, but is not intended as a replacement. DDNS [10] and Overlook [38] are peer-to-peer name services designed to enhance fault tolerance and load balancing properties. DDNS implements the legacy DNS functionalities on top of Chord [37], an $O(\log N)$ lookup time structured DHT based on consistent hashing. Overlook is a general purpose name service built on top of Pastry [35]. Both DDNS and Overlook incur high lookup latencies as requests are routed through $O(\log N)$ overlay hops. Beehive provides a replication framework that enables CoDoNS to achieve $O(1)$ lookup performance.

Some researchers have proposed to replace the hierarchical DNS and URL namespace with flat global identifiers [2]. CoDoNS can be used to map such identifiers to physical locations or to their content with high performance.

Structured DHTs: In addition to Chord and Pastry, several structured DHTs have been proposed in recent years. CAN [34] maps both objects and nodes on a d-dimensional torus and provides $O(dN^{\frac{1}{d}})$ latency. Tapestry [42] employs consistent hashing [21] to map objects to nodes, and prefix-matching [32] to route lookups in $O(\log N)$ hops. Kademlia [25] provides $O(\log N)$ lookup performance using a similar search technique, but uses the XOR metric for routing. SkipNet [16] uses skip-lists to provide $O(\log N)$ probabilistic lookup guarantee. Viceroy [24] provides $O(\log N)$ lookup performance with a constant degree routing graph. A few DHTs use De Bruijn graphs [22, 40] to achieve $O(\frac{\log N}{\log \log N})$ lookup performance. The multi-hop lookup performance provided by these DHTs is inadequate to support performance sensitive application like DNS.

A few recent DHTs provide $O(1)$ lookup performance at the cost of increased storage and bandwidth consumption. Kelips [14] limits lookup latency to one or two hops by replicating each object on $O(\sqrt{N})$ nodes and using gossip-based protocols to manage replication. An alternative method to achieve one hop lookups is described in [15], and relies on maintaining full routing state (i.e. a complete description of system membership) at each node. The space and bandwidth costs of this approach scale linearly with the size of the network. Farsite [12] also routes in a constant number of hops using a fixed depth hierarchy, but does not address rapid membership changes. Overall, these DHTs incur a minimum delay of at least one overlay hop, whereas CoDoNS can decrease the average lookup time to less than a single hop.

6. CONCLUSIONS

The Domain Name System is a critical component of the Internet. The growth of the Internet namespace, the explosion in the number of networked hosts, and the recent emergence of large-scale coordinated attacks have strained the hierarchical, static architecture of the legacy Domain Name System. DNS is vulnerable to DoS attacks, incurs high latencies for query resolution and update propagation, suffers from misconfigurations, and is difficult to administer.

In this paper, we propose a novel alternative for DNS, called CoDoNS. CoDoNS retains the most succesful parts of the DNS design; namely, the hierarchical partitioning of the namespace, the independent management of different parts of the hierarchy, and the general-purpose database interface. CoDoNS combines peer-to-peer overlay networks with analytically-informed proactive caching to provide an alternative DNS infrastructure. It resists denial of service attacks, heals around failures, automatically distributes load, supports fast updates and adapts quickly to flash crowds. It decouples nameservice from the physical location of nameservers through cryptographic delegations, and creates a competitive marketplace for name services. Performance measurements from a deployment on PlanetLab using real DNS workloads indicate that CoDoNS can significantly improve the lookup performance of legacy DNS with modest storage and bandwidth overhead.

CoDoNS provides a new platform for nameowners to efficiently publish and manage their data. Our current implementation and deployment provides a simple incremental migration path from legacy DNS towards the performance and functionality offered by CoDoNS. During this process CoDoNS can serve as a safety net alongside legacy DNS.

Acknowledgments

We are grateful to Robert Morris, Hari Balakrishnan, Jaeyon Jung, and Emil Sit for providing their DNS traces.

REFERENCES

[1] M. Andrews. Negative Caching of DNS Queries. *RFC 2308*, Mar 1998.

[2] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, M. Walfish. A Layered Naming Architecture for the Internet. *SIGCOMM*, Portland OR, Aug 2004.

[3] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating System Support for Planetary-Scale Network Services. *Symposium on Networked Systems Design and Implementation*, San Francisco CA, Mar 2004.

[4] T. Brisco. DNS Support for Load Balancing. *RFC 1794*, Apr 1995.

[5] N. Brownlee, kc Claffy, and E. Nemeth. DNS Measurements at a Root Server. *GlobeCom*, San Antonio, TX, Nov 2001.

[6] N. Brownlee, kc Claffy, and E. Nemeth. DNS Root/gTLD Performance Measurements. *Systems Administration Conference*, San Diego CA, Dec 2001.

[7] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. *International Conference on Computer Communications*, New York NY, Mar 1999.

[8] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. *Symposium on Operating Systems Design and Implementation*, Boston MA, Dec 2002.

[9] E. Cohen and H. Kaplan. Proactive Caching of DNS Records: Addressing a Performance Bottleneck. *Symposium on Applications and the Internet*, San Diego-Mission Valley CA, Jan 2001.

[10] R. Cox, A. Muthitacharoen, and R. Morris. Serving DNS using a Peer-to-Peer Lookup Service". *International Workshop on Peer-To-Peer Systems*, Cambridge MA, Mar 2002.

[11] P. Danzig, K. Obraczka, and A. Kumar. An Analysis of Wide-Area Nameserver Traffic: A study of the Internet Domain Name System. *SIGCOMM*, Baltimore MD, 1992.

[12] J. Douceur, A. Adya, W. Bolosky, D. Simon, and M. Theimer. Reclaiming Space from Duplicate Files in a Serverless Distributed File System. *International Conference on Distributed Computing Systems*, Vienna Austria, Jul 2002.

[13] D. Eastlake. Domain Name System Security Extensions. *RFC 2535*, Mar 1999.

[14] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse. Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead. *International Workshop on Peer-To-Peer Systems*, Berkeley CA, Feb 2003.

[15] A. Gupta, B. Liskov, and R. Rodrigues. Efficient Routing for Peer-to-Peer Overlays. *Symposium on Networked Systems Design and Implementation*, San Francisco CA, Mar 2004.

[16] N. Harvey, M. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties., *Symposium on Internet Technologies and Systems*, Seattle WA, Mar 2003.

[17] C. Huitema and S. Weerahandi. Internet Measurements: the Rising Tide and the DNS Snag., *ITC Specialist Seminar on Internet Traffic Measurement and Modeling*, Monterey CA, Sep 2000.

[18] Internet Systems Consortium. BIND Vulnerabilities. *www.isc.org/sw/bind/bind-security.php*, Feb 2004.

[19] J. Jung, E. Sit, H. Balakrishnan, and R. Morris. DNS Performance and Effectiveness of Caching. *SIGCOMM Internet Measurement Workshop*, San Francisco CA, Nov 2001.

[20] J. Jung, A. Berger, and H. Balakrishnan. Modeling TTL-based Internet Caches. *International Conference on Computer Communications*, San Francisco CA, Mar 2003.

[21] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin and R. Panigrahy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot-spots on the World Wide Web. *Symposium on Theory of Computing*, El Paso TX, Apr 1997.

[22] F. Kaashoek and D. Karger. Koorde: A Simple Degree-Optimal Distributed Hash Table. *International Workshop on Peer-To-Peer Systems Workshop*, Berkeley CA, Feb 2003.

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

[24] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A Scalable and Dynamic Emulation of the Butterfly. *Symposium on Principles of Distributed Computing*, Monterey CA, Aug 2002.

[25] P. Maymounkov and D. Maziéres. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. *International Workshop on Peer-To-Peer Systems*, Cambridge MA, Mar 2002.

[26] P. Mockapetris. Domain Names: Concepts and Facilities. *RFC 1034*, Nov 1987.

[27] P. Mockapetris. Domain Names: Implementation and Specification. *RFC 1035*, Nov 1987.

[28] P. Mockapetris and K. Dunlop. Development of the Domain Name System. *SIGCOMM*, Stanford CA, 1988.

[29] R. Naraine. Massive DDoS Attack Hit DNS Root Servers. *www.internetnews.com/dev-news/article.php/1486981*, Oct 2002.

[30] V. Pappas, Z. Xu, S. Lu, D. Massey, A. Terzis, and L. Zhang. Impact of Configuration Errors on DNS Robustness. *SIGCOMM*, Portland OR, Aug 2004.

[31] K. Park, Z. Wang, V. Pai, and L. Peterson. CoDNS : Masking DNS Delays via Cooperative Lookups. *Princeton University Computer Science Technical Report TR-690-04*, Feb 2004.

[32] G. Plaxton, R. Rajaraman, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. *Theory of Computing Systems*, vol 32, pg 241-280, 1999.

[33] V. Ramasubramanian and E. G. Sirer. Beehive: Exploiting Power Law Query Distributions for O(1) Lookup Performance in Peer to Peer Overlays. *Symposium on Networked Systems Design and Implementation*, San Francisco CA, Mar 2004.

[34] S. Ratnasamy, P. Francis, M. Hadley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. *SIGCOMM*, San Diego CA, Aug 2001.

[35] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *International Conference on Distributed Systems Platforms*, Heidelberg, Germany, Nov 2001.

[36] A. Shaikh, R. Tewari, and M. Agarwal. On the Effectiveness of DNS-based Server Selection. *International Conference on Computer Communications*, Anchorage AK, Apr 2001.

[37] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-to-peer Lookup Service for Internet Applications. *SIGCOMM*, San Diego CA, Aug 2001.

[38] M. Theimer and M. Jones. Overlook: Scalable Name Service on an Overlay Network *International Conference on Distributed Computing Systems*, Vienna Austria, Jul 2002.

[39] P. Thurrott. Microsoft Suffers Another DoS Attack. *www.winnetmag.com/WindowsSecurity/Article/ArticleID/19770/WindowsSecurity_19770.html*, Jan 2001.

[40] U. Wieder and M. Naor. A Simple Fault Tolerant Distributed Hash Table. *International Workshop on Peer-To-Peer Systems*, Berkeley CA, Feb 2003.

[41] C. Wills. The Contribution of DNS Lookup Costs to Web Object Retrieval. *Worcester Polytechnic Institute Technical Report TR-00-12*, Jul 2000.

[42] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. *Journal on Selected Areas in Communications*, 2003.

[43] L. Zhou, F. B. Schneider, and R. van Renesse. COCA: A Secure Distributed On-line Certification Authority. *Transactions on Computer Systems* vol 20, Nov 2002.