# A Survey of Latency in the Domain Name Service

Huan Keat Toh
Department of Computer Science
Florida State University
huanktoh@cs.fsu.edu

Charles Weddle
Department of Computer Science
Florida State University
weddle@cs.fsu.edu

Abstract: The Domain Name System (DNS) is one of the many critical parts that make the Internet what it is today. DNS serves an important role in translating hostnames to IP addresses. As well as it works, it has some problems. This paper examines the problem of latency in DNS. The paper tries to find out if latency in DNS will ever improve. In this survey, two main approaches to improve latency are examined, proactive caching and cooperative lookup. The ideas researchers have proposed using these approaches are examined and compared.

## 1 Introduction

The Domain Name System (DNS) organizes machines into domains and maps host names onto IP addresses. DNS exists because it is difficult for people to remember IP addresses. Trying to remember diablo.cs.fsu.edu is much easier than trying to remember 128.186.122.12. As well, using DNS allows an IP address to change without having to change a domain name. Imagine if you had to tell everyone your new email address ever time your Independent Service Provider (ISP) changed servers. These two features of DNS eliminate a lot of headaches.

DNS can be thought of as a general—purpose database for managing host information on the Internet. The DNS database implements a hierarchical, domain-based naming scheme. Primarily DNS is used to map host names and email destinations to IP addresses. DNS maps host names and email addresses to IP addresses via a resolver. A resolver is a computer program that simply sends a datagram packet (UDP) to a local DNS server that searches for a mapping and sends back the IP address. A DNS server can search through the hierarchical database either recursively or iteratively looking for the correct address translation for the given host name.

Information is kept in DNS in a data structure called a resource record. Resource records are used to associate values of different types with domain names. For example, a resource record somewhere in the DNS database would have the host name diablo.cs.fsu.edu and the IP address 128.186.122.12 together. The resolver program would find this resource record and report the information back to the requester. The resource records also contain the time to live (TTL) value. This value represents how long a resource record will remain in cache on a nameserver.

The DNS hierarchical database is partitioned into non-overlapping regions called domains. For example, edu is a top level domain. Within the domains are sub-domains that further partition the domains. For example, fsu.edu is a sub-domain of the domain edu. Authoritative nameservers manage all information for hostnames in a particular domain. At the top of the DNS hierarchy are top level domain (TLD) authoritative nameservers for domains such as edu. Above each of the TLD's are the root authoritative nameservers at the very top, which keep track of the TLD authoritative nameservers.

Besides authoritative nameservers there exist non-authoritative nameservers, or just nameservers. An authoritative nameserver must have an authoritative resource record for all the hosts in its domain and every authoritative resource record must be current. While a nameserver neither guarantees the state of the resource record nor guarantees to have all the resource records in a domain. A nameserver has cached resource records that may be out of date. If a nameserver does not have a cached resource record for a hostname, the nameserver will forward the request up the DNS hierarchy.

Searching for resource records starting from the root authoritative nameservers down to the top level domain authoritative nameservers, all the way down to the appropriate sub-domain authoritative nameservers can take a lot of time and introduce long delays for hostname translation. This delay is known as latency. Because of this potential delay, DNS employs caching in order to reduce latency. To do this the resolver program simply saves the resource record it finds in the cache of each nameserver it encountered along its path. Now that the found resource record is available on the local DNS nameserver, each subsequent DNS request for that hostname can be resolved by the local DNS server.

Latency in DNS is the focus of this survey paper. DNS has certainly served a purpose in the initial construction of the Internet. As well as it works, it has some problems and one of the main problems is the delay in name resolution. This survey paper presents solutions that researchers are proposing to improve latency in DNS. Specifically, two approaches are being looked at by different researchers. One approach deals with proactive caching and the other approach deals with cooperative lookups. This survey examines how research in these areas is impacting latency in DNS.

The paper continues with an overview of the current problem with latency in DNS. Next, this survey presents the proposed solutions to the latency problem. Lastly, this survey paper concludes with a discussion on the future of DNS and if latency will ever be improved.

# 2 Latency Problems with Legacy DNS

## 2.1 Low Cache Hit Rate

One mechanism used in DNS to reduce latency is caching. Recently accessed DNS resource records are cached on nameservers so that items that are already validated can be reused for future queries. Many studies have been conducted to evaluate the delay in

accessing web services. Will and Shang revealed DNS lookup contributes 20% of web object retrievals [9]. Huitema et al. observed that 29% of DNS queries take more that 2 second [3]. Also, Jung et al. in another study observed that 10% of queries take more than 2 seconds [4]. The common source of this name resolution latency problem is low cache hit rates, stemming from heavily tailed, Zipf –like query distribution in DNS [1]. It has been documented in studies related to Web caching that heavily tailed query distribution severely limits cache hit rates.

## 2.2 Dynamic Server Selection

Content distribution networks such as Akamai and Digital Island uses DNS to direct clients to closer servers of web content. This is made possible using a method called dynamic server selection. Dynamic nameserver selection, which is common in widespread content distribution networks, limits the effectiveness of caching and imposes an enormous overhead on DNS. This mechanism uses the DNS to direct client queries to closer web content servers. This mechanism imposes a requirement on these servers by restricting them to use shorter TTL values to further supports sudden surge in network loads. A study on the impact of short TTL values on caching shows that cache hit rates decreases significantly for TTL values lower than 15 minutes. Another study regarding the adverse effect of server selection shows that name resolution latency can increase by 2 orders of magnitude.

## 2.3 Improper Configurations

Another cause of latency in resolving DNS queries is manual configuration. The administration of such a large-scale system is not only complex but it also plays a very important role in determining the performance of name resolution. One example of manual configuration and administration is the appropriation of the correct TTL values for DNS records on the authoritative nameservers. In one survey conducted on a pool of nameservers, for 14 % of the domains, the pool of nameservers returned mix responses. Some gave inconsistent responses and a few even reported that the domain does not exist. All of which are due to improperly configured DNS nameservers.

## 2.4 Load Imbalance

Studies show that root nameservers and top-level domain nameservers handle a majority of the DNS traffic. This means that load is not balanced well within the DNS hierarchy, ultimately creating more latency. This also makes the root and top-level domain nameservers popular targets to frequent DoS attacks. As a result, there are now more than 60 root nameservers. Having this many root nameservers means more resource records have to be replicated to maintain a consistent state.

### 2.5 Update Propagation

The effectiveness of caching of DNS resource records relies heavily on finding a suitable TTL value for each resource record. In the presence of dynamic changes, the problem of maintaining consistency of cached DNS records arises. While many researches have examined how to devise a suitable TTL value, no one has been able to correctly anticipate such values. Short TTL values affect the lookup performance and increase network load, and long TTL values interfere with service relocation.

# 3 Solutions to Improve Latency

## 3.1 Proactive Caching

One of the primary reasons for latency in DNS is the delay associated with finding a cached resource record for the hostname being resolved. Researchers have found that by implementing more aggressive caching strategies this delay can be significantly reduced. Two researchers have proposed solutions to improve latency in DNS using proactive caching.

Cohen et al. implemented a proactive caching solution using a set of defined renewal policies [2]. These researchers noted that caching in legacy DNS is passive whereby the DNS record is created as a direct result of a client query and cached until the TTL for the DNS record expires. Proactive DNS caching integrates this passive caching with automatically generated preemptive queries that update the cache.

Ramasubramanian et al. present Cooperative Domain Name Service (CoDoNS) as a backward compatible replacement for conventional DNS [6]. CoDoNS implements proactive caching through a peer-to-peer network and an aggressive caching scheme called Beehive.

### 3.1.1 Renewal Policies

The proactive caching approach introduced by Cohen et al. attempts to balance the number of eliminated cache misses and reduce the overhead of additional queries issued to remote name servers. Their general idea is to implement a set of renewal policies. Renewal policies attempt to refresh the original cached DNS entries upon their expiration time. Several natural policies are evaluated and these policies differ by when a cached entry is renewed.

A renewal of a cached resource record is to perform a new resolution of that resource record when it expires. This would include updating the cached copy and extending the time to live value appropriately. A renewal policy associates a renewal credit with each cached item. A renewal credit is the number of remaining renewals for each resource

record. There credit may be increased when the cached item is used. When the item expires and has a positive credit, it is renewed and the credit is decremented.

There are six different policies introduced by the authors:

1. R-FIFO (r)
   This is analogous to FIFO cache replacement policy in that upon a miss, the item with the earliest entry-time into the cache will be evicted.

2. R-LRU (r)
   This renewal policy evicts item with the least-recent cache hit.

3. R-LRU (e, r)
   This policy is similar to R-LRU(r), but renewal credit is reset to r only as a result of hits that occur after an e-fraction of the current TTL interval had passed.

4. R-LFU (r)
   This policy evicts the item with the smallest number of cache hits.

5. R-Adaptive
   This policy is an enhancement of R-LRU where different credit r(h), is associated with each host h. To find the mapping r(h), R-Adaptive collects per host statistics from "learning data".

6. R-OPT
   This is the optimal policy. The author uses a simple greedy approximation for this policy. A greedy approximation performs a single pass on the input and grants renewals only if the gap between the expiration and the next request is less than r TTLs. Analogous to Belady's cache replacement algorithm.

The authors also introduced simultaneous validation in conjunction with the renewal policies. Simultaneous validation is a solution that tries to overcome the problem with overhead found in the renewal policies. In comparison to renewal policies, simultaneous validation does not impose an overhead of additional DNS queries. Under simultaneous validation, expired DNS records are cached. Whenever a client issues a request to the host, a lookup is made to the expired DNS records that are cached. If an entry is available from this cache, the proxy/ browser issues an HTTP request using this expired address while simultaneously issuing a DNS query to resolve the hostname. To avoid inconsistency, content acquired from the HTTP request are held. The content is delivered to the user upon a successful validation of the expired address entry with the DNS query results.

### 3.1.2 CoDoNS

CoDoNS is a new design to the legacy DNS that combines structured peer-to-peer overlays and analytically informed proactive caching to provide high performance, resilience to attacks, and fast update propagation. CoDoNS achieves high performance through the use of Beehive [7], an analytical-driven proactive caching layer. Beehive automatically replicates the DNS mappings throughout the network to match anticipated demand. The authors envision that CoDoNS will be globally distributed servers that will self-organize to form a flat peer-to-peer network, essentially acting like a large, cooperative, shared cache.

CoDoNS uses Pastry and Beehive as a central part of its design. Pastry is a structured dynamic hash table (DHT) that uses prefix-matching to lookup objects [8]. In Pastry both objects and nodes have randomly assigned identifiers from the same circular space. Each object in Pastry is stored at the nearest node in the identifier space, called the home node. Each node in Pastry routes a request for an object by successively matching prefixes until the home node is reached. Successively matching prefixes means that a request is continually routed to a node that matches one more digit of the object's unique identifier. For example, Figure 1 shows a request for object 2101. Pastry successively matches prefixes by routing the request to a node that matches one more digit with the object until the home node 2100 is reached.

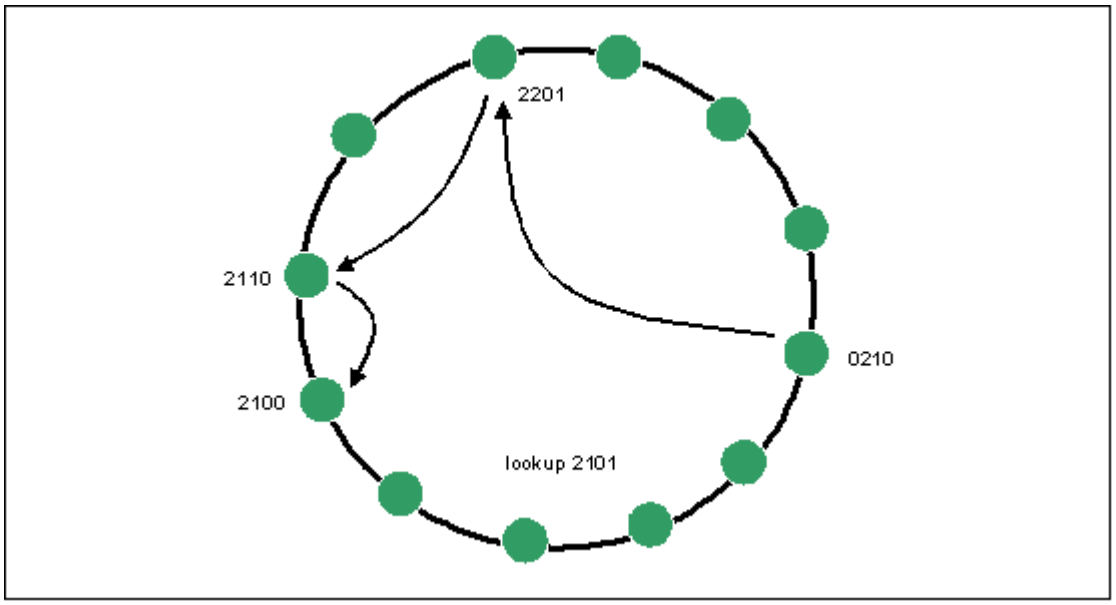


Figure 1: Pastry peer-to-peer network

Beehive improves Pastry through a novel technique based on controlled proactive caching to reduce the average lookup latency of structured DHTs. Beehive is a proactive replication framework that enables prefix-matching distributed hash tables (DHT) to achieve O(1) lookup performance. 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*. Beehive judiciously chooses different levels of replication for different objects allowing the

average lookup performance of the system to be tuned. O(1) lookups can be achieved by replicating every object at every node but this would incur excessive overhead, consume significant bandwidth, and have large update latencies. Instead, Beehive minimizes the total number of replicas subject to the constraint that the aggregate lookup latency is less than a desired constant C. In CoDoNS, Beehive's target C is set to 0.5, meaning that a large percentage of requests are answered immediately without having to take any extra network hops. Figure 2 shows an example of a Beehive of CoDoNS servers built on top of a Pastry peer-to-peer network.

The aggregation and replication protocols of Beehive allow CoDoNS to constantly monitor the access frequency of DNS records and adjust replication accordingly. For example, under flash demand for a particular DNS entry, CoDoNS can increase the number of replicas and spread the load among several nodes.

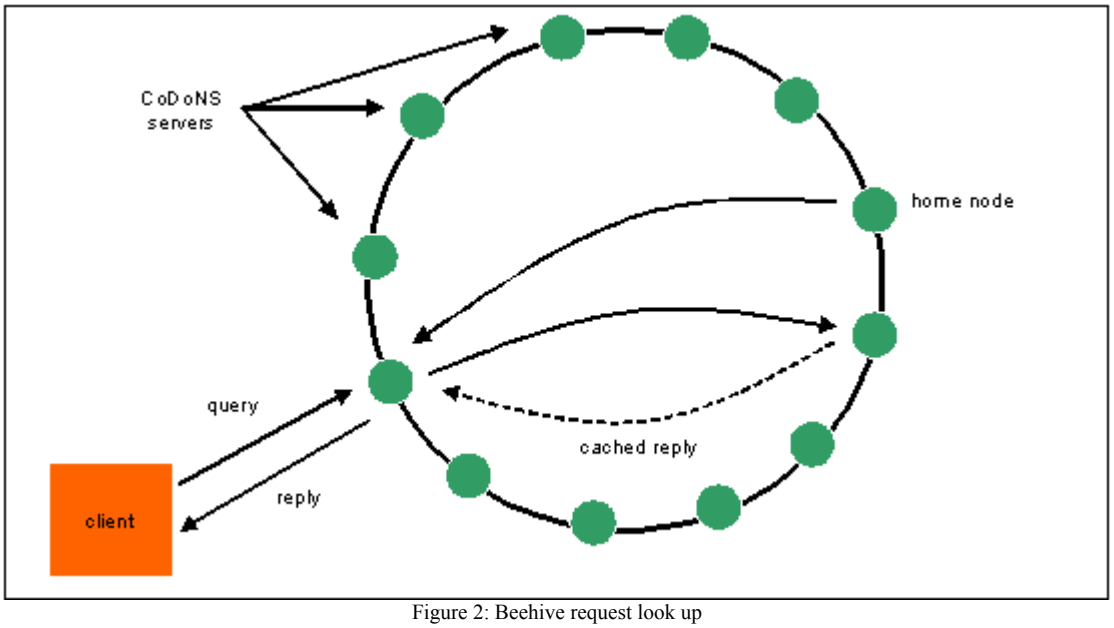

Figure 2: Beehive request look up

### 3.1.3 Proactive Caching Results

Cohen et al. evaluated the effectiveness of various renewal policies by building a simulation based on the proxy logs collected from 4 different sources. The proxy logs contain information regarding user identity, web server, requested resource and cache performance that indicates if the content requested was a hit or miss. DNS data was obtained by recording information on associated IP addresses; aliases (canonical names), immediate nameservers, query times, and TTL values. The authors issued several queries for each host name and varied the time intervals between each query. This made it possible for the evaluation to obtain estimates for the rate-of-change. These TTL values are then used to emulate the local name-server cache and its performance under various renewal policies. The authors did not report any empirical results from their evaluation

but stated that the renewal policies would offer a reduced perceived latency and ultimately improve the experience of web users.

To evaluate CoDoNS, Ramasubramanian et al. deployed the CoDoNS prototype on PlanetLab. PlanetLab is an open platform for developing, deploying, and accessing planetary scale services. By doing this, it allowed the authors to deploy CoDoNS on servers around the world and evaluate it against real world Internet traffic.

The authors wanted to highlight three properties of CoDoNS through experimentation. First, CoDoNS provides a low latency name resolution service. Second, CoDoNS has the ability to resist flash-crowds by spreading load across multiple servers. Lastly, the CoDoNS can support fast update propagation.

CoDoNS showed positive improvement over DNS in terms of latency. In fact, 50% of the queries in CoDoNS were answered immediately by a local CoDoNS server without any network delay. This was the case because of the proactive replication pushes for the most popular domain names to all CoDoNS servers. Most importantly, CoDoNS provided a significant decrease in median latency. CoDoNS had a median latency of 2 milliseconds while DNS has a median latency of 39 milliseconds.

### 3.1.4 Issues with Proactive Caching

One of the issues that the authors identified with the renewal policy approach to proactive caching is that modification to the legacy DNS resolver will be required. In order to maintain the record in the cache, the DNS resolver code would need to be modified in order to accommodate the logic for all the different types of renewal policies.

One of the main issues with CoDoNS that the authors identified was that with any peer-to-peer system, it relies on participants to contribute resources on behalf of others. The authors feel that participants will ultimately not shy away because the load is balanced well in CoDoNS. The authors also point out that the CoDoNS design prohibits dynamic name resolution techniques where mapping is determined as a result of a complex function evaluated at run time. Lastly, the authors mention that malicious participants my also disrupt the system by corrupting the routing tables of peers, leading to misrouting or dropping queries.

## 3.2 Cooperative Lookup

Finding an optimal TTL value will affect the how well updates are propagated to the assigned nameservers. Some servers might receive updates earlier or later than others. To this end, incorporating multiple nameservers in the lookup process can increase the probability of getting a hit from a server with the updates already in place. The researchers at Princeton University have developed a method of masking DNS lookup delay using cooperative lookup [5].

### 3.2.1 CoDNS

Park et al. introduce a solution to reduce latency using cooperative lookup through their prototype called CoDNS. The main idea behind CoDNS is to forward name lookup queries to neighbor nodes when the local name service is experiencing problems. The local DNS lookup is labeled problematic if the local name service cannot resolve the hostname within 200 ms. If this happens, CoDNS will employ a neighbor node to perform a DNS lookup.

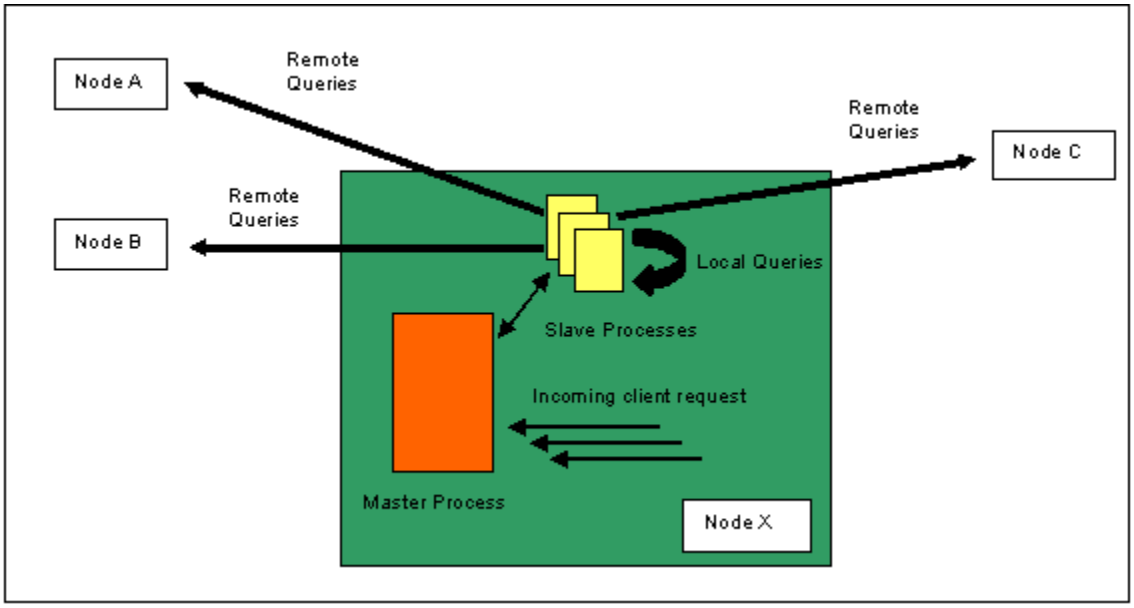


Figure 3: CoDNS master and slave processes

Figure 3 depicts the CoDNS server design. As the figure shows, CoDNS consists of a stand-alone daemon running on each of the nodes. The daemons residing on these nodes are accessible via UDP for remote queries, and loop back TCP for locally originated name lookups. The daemon is event driven, and is implemented as a non-blocking master process with many blocking slave processes. The master process receives a name lookup request from local clients and remote neighbors and hands these requests over to one if its idle slave processes. The slave process then assumes the responsibility to resolve the DNS query by calling gethostbyname(). When complete, the slave process sends the results back to the master process. When the master process receives results from its slave processes, it sends the result to either a local client or a remote neighbor depending on the request origin.

The master process records each request's arrival time from local clients. An idle slave process will then assume the responsibility to resolve the lookup. If this slave process does not return a result to the master process within a pre-defined timeout value, the master process will take charge and send a UDP name lookup to neighbor node.

In the event that neither the local and remote query has responded, CoDNS doubles the delay value before sending the next remote query to another neighbor. Whichever result

arrives first will be delivered as the response for the name lookup to the client. Neighbors may drop remote queries during heavy load and remote queries that fail to resolve are discarded. Slave processes may intentionally add delay if they receive a locally generated query that fails to resolve so that the chance of a successful resolution will increase. The more processes that are involved in the lookup, the higher the probability that a name will be resolved successfully.

To address the issue of choosing the optimal timeout value in CoDNS, the initial timeout value before sending the first remote query is dynamically adjusted based on the recent performance of local name servers and neighbor responses. The intention is that the when the local name servers perform well, that is when responses for each lookup are fast, the CoDNS master process can increase the timeout for the idle processes. This has the affect of reducing the number of remote queries being sent out. However, when the remote queries perform better than the local queries, CoDNS reduces the timeout value for the local processes.

The next problem to tackle is how to select the closest neighbor to perform the remote queries. Sending a remote query to a node that has closer proximity is far better, in terms if latency, than one that is very far away. Each CoDNS node gathers and manages a set of neighbor nodes within a reasonable latency boundary. Among these neighbor nodes, one neighbor is chosen for each remote name lookup. Liveness of neighbor nodes is periodically checked to see if the service is available. When CoDNS starts, it sends a heartbeat to each node in the node list every second. The heartbeat responses contain the round trip time and average response time of the local DNS at the neighbor node. If the sum of these is less than 90 ms, it is chosen as a neighbor. As soon as CoDNS fills up 30 such nodes, it stops scanning the list and only monitors the nodes in the neighbor set. If it could not find enough neighbors, the latency boundary is increased a little bit and scanning is repeated. The default latency boundary and the number of minimum neighbors are configurable according to the distribution of nodes.

Given that neighbors are relatively soft state, CoDNS does not expend excessive effort guaranteeing their availability. After having found enough neighbors, it monitors the liveness of each node by sending a heartbeat every 30 seconds. If the heartbeat does not arrive within short time period, then the node is excluded in the next remote query selection. Periodically, the dead nodes are updated with fresh ones by partial scanning of the node list. By consistently choosing the same node for the same name, a certain amount of request locality can be expected.

### 3.2.2 Cooperative Lookup Results

Park et al. used real traffic to evaluate the effectiveness of CoDNS. The traffic was gathered from an actual CoDNS implementation. The authors found that under real traffic, CoDNS uses the remote neighbors for 18.9% of all lookups. Of all the remote queries sent to remote neighbors, 34.6 % of the remote queries succeed by returning a valid DNS response faster than the local DNS lookup. Ultimately, CoDNS showed a 7

milliseconds average response time, a definite improvement over legacy DNS with a measured 84 milliseconds average response time by the authors.

The authors performed another evaluation to study the effect of querying multiple remote neighbors in the case of CoDNS retransmission. Figure 4 shows the distribution of success, returning a valid DNS response faster than local DNS, by sending out 1 remote query, 2 remote queries and 3+ remote queries. Figure 4 shows that CoDNS will win within 2 remote queries most of the time.

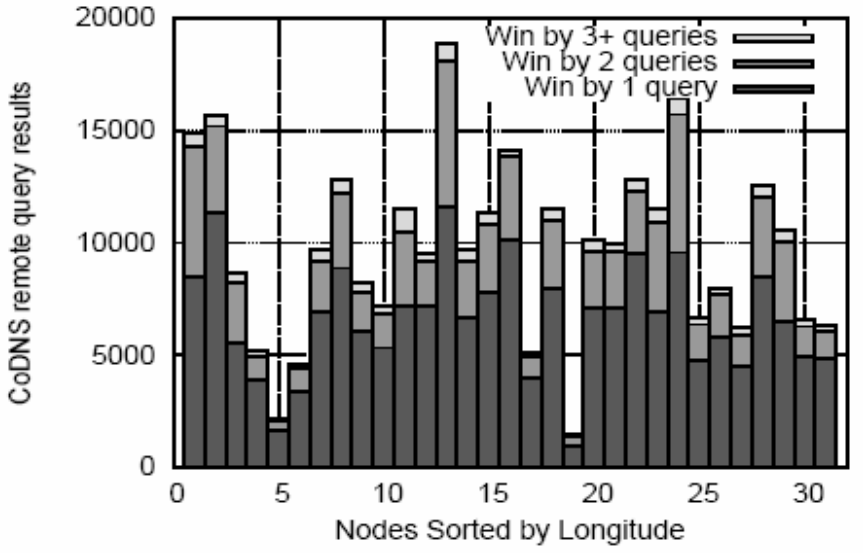

Figure 4: CoDNS results

### 3.2.3 Issues with Cooperative Lookup

According to Park et al. there are two significant issues involved in the CoDNS design. The first issue being the extra DNS lookup per user request and the second being the extra network traffic generated to accommodate not only the extra DNS lookup but also the heartbeat messages between CoDNS nodes.

The authors note that one possible alternative to reduce DNS lookup per user request is to find an optimal initial delay time. As indicated in previous section, this initial delay time is the duration of which the local node will perform the query before requesting assistance from remote node.

## 4 Conclusion

The two areas of research to improve latency in DNS presented in this paper were proactive caching and cooperative lookup. The researchers using proactive caching techniques found an improved latency in their prototype implementations by a factor of 20 over legacy DNS. As well, the researchers using the cooperative lookup technique

found an improved latency in their prototype implementation by a factor of 12 over legacy DNS.

If the papers surveyed are any indication of the future, then latency in DNS should eventually improve. Unfortunately, there remain some problems preventing this from happening quickly. With these new ideas, come new problems. For example, the proactive caching solutions increase the amount of network traffic. The solutions that use peer-to-peer networks introduce new security risks that are inherent with peer-to peer networks.

It is also difficult to adopt a new solution to replace a part of the Internet as significant as DNS. Legacy DNS is everywhere, from the root nameservers all the way down to the local nameservers all across the world. Because DNS is so widespread, it is difficult to replace it. In fact, the authors of the papers surveyed acknowledge this as a problem and typically present a way for their solution to co-exist with legacy DNS for awhile with the hope that it would eventually replace DNS.

# 5 References


[1] 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.

[2] E. Cohen, H. Kaplan. Proactive Caching of DNS Records: Addressing a Performance Bottleneck, p. 85, 2001 Symposium on Applications and the Internet (SAINT'01), 2001.

[3] C. Huitema, S. Weerahandi. Internet Measurements: the Rising Tide and the DNS Snag., ITC Specialist Seminaron Internet Traffic Measurements and Modeling, Monterey CA, Sep 2000

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

[5] K. S. Park, V. S. Pai, L. Peterson, Z. Wang. CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups. In *Proceedings of the Sixth Symposium on Operating Systems Design and Implementation (OSDI '04)*

[6] V. Ramasurbramanian, E. G. Sirer. The Design and Implementation of a Next Generation Name Service for the Internet. SIGCOMM ’04, Portland OR, Aug 2004.

[7] V. Ramasurbramanian, 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.

[8] A. Rowstron, P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *International Conference on Computer Communications*. Anchorage AK, Apr 2001.

[9] C. Wills. The Contribution of DNS lookup Cost to Web Object Retrieval. Worchester Polutechnic Technical Report, Jul 2000.