

DNS-based Internet Client Clustering and Characterization*

Azer Bestavros and Sumit Mehrotra
Computer Science Department
Boston University

{best|sumit}@cs.bu.edu

Abstract

This paper proposes a novel protocol which uses the Internet Domain Name System (DNS) to partition Web clients into disjoint sets, each of which is associated with a single DNS server. We define an L-DNS cluster to be a grouping of Web Clients that use the same Local DNS server to resolve Internet host names. We identify such clusters in real-time using data obtained from a Web Server in conjunction with that server’s Authoritative DNS—both instrumented with an implementation of our clustering algorithm. Using these clusters, we perform measurements from four distinct Internet locations. Our results show that L-DNS clustering enables a better estimation of proximity of a Web Client to a Web Server than previously proposed techniques. Thus, in a Content Distribution Network, a DNS-based scheme that redirects a request from a web client to one of many servers based on the client’s name server coordinates (e.g., hops/latency/loss-rates between the client and servers) would perform better with our algorithm.

1 Introduction

World Wide Web traffic has been increasing at breathtaking rates [5]. Both network congestion and overloaded servers lead to slow response times for Web requests. At times, it is the network latency that is to blame; at other times it is the load on the server which is the culprit. This has been demonstrated in various studies including [11]. Efforts to reduce the impact of overloaded servers use replication and load balancing techniques [25, 7] using methods which vary in complexity (from load balancing using TCP redirection techniques [9] to simplistic DNS-based¹ load balancing [14]). To alleviate network problems, efforts to improve network performance have focused on protocol improvements such as HTTP/1.1 [12], and more recently on the use of overlay networks such as Content Distribution/Delivery Networks(CDNs) [1, 3] to move content to the “edges” of the network—closer to its consumers. Many such CDNs rely on DNS-based [23] redirection schemes for wide area load balancing.

Motivation: The ability of a web site to characterize client access patterns enables the distribution of content to replicas to be done in a “network-aware” fashion, by moving content closer to the web clients and balancing load across replicas. For example, if a Web site (consisting of a farm of webservers) knows that a particular client belongs to a cluster from which requests for small files arrive at times when the network (to that cluster) is highly congested and when most servers are overloaded, then it would make more sense to direct such a request to the least loaded server, because server delay contributes most to the transfer delay for small and medium sized files in times of high server load and high network load [11]. On the other hand, if the client comes in at time of low server load, the request should be handled by the server “closest” to that client, because network delay plays a major part in response time under such a condition [11].

*This work was partially supported by NSF research grants CCR-9706685 and ANIR-9986397.

¹The Internet Domain Name System (DNS) has been used as an effective hierarchical distributed naming service for network hosts and clients. Recently, protocols that leverage the workings of this service have been proposed to redirect web requests from a *Web Client* to a *Web Server* that is “most suitable” with respect to a specific performance metric (e.g., latency optimizations, load balancing across server farms). Case in point is the use of DNS in the implementation of server selection in Content Delivery Networks (CDNs).

Paper Contribution: In this paper we use the term *L-DNS cluster* to denote a set of web clients (IP addresses), which use the same Local Domain Name Server to resolve host names in the Internet. We present a novel technique that enables the identification of such L-DNS clusters in real-time using data obtained from a Web Server in conjunction with that server’s Authoritative DNS—both instrumented with an implementation of our clustering technique. Once derived, L-DNS mappings could be used to characterize the distribution of load that is associated with a particular DNS name server, which in turn can be used to perform a number of unique functions (related to load distribution, engineering, prediction, among others).

Using L-DNS clusters we obtained over a 16-day period (by implementing our algorithm on a “live” Web site), we performed measurements from four distinct Internet locations. For comparison purposes, we performed similar measurements for *random* clusters (referred henceforth as R-DNS clusters) as well as for prefix-matched clusters (referred henceforth as P-DNS clusters). Our results show that L-DNS clustering enables a better estimation of proximity of a Web Client to a Web Server than both of these techniques.

Our findings have significant implications on DNS-based request routing mechanisms employed in Content Distribution Networks (CDNs). A request redirection scheme that redirects a request from a web client to one of many servers based on the client’s name server coordinates (e.g., hops/latency/bandwidth/loss-rates between the client’s name server and the CDN servers) would benefit immensely from using our L-DNS clustering technique. Specifically, our findings suggest that basing request routing decisions on the “proximity” (with respect to some performance metric) of a CDN proxy to the client’s name server is warranted *only* for name servers whose L-DNS cluster exhibits a high-enough level of correlation with respect to the metric of interest.

Paper Overview: The remainder of this paper is organized as follows. In Section 2, we examine related work. We discuss various clustering approaches and overview the use of clustering as a tool for “network-aware” applications. In Section 3, we describe a novel protocol for associating clients of a Web server with Internet Name Servers that act on their behalf for DNS domain name lookup. This association is in the form of a mapping of client (network) IP addresses to Name Server IP addresses. In Section 4, we present our proposed architecture for “network-aware” applications based on our L-DNS clustering algorithm. In Section 5, we present the results of our experimental evaluation of L-DNS as well as other clustering approaches. We conclude in Section 6 with a summary and an overview of current and future work.

2 Related Work

In this section, we review prior work on client clustering approaches and on DNS-based resource management techniques for distributed Web services.

Client Clustering Approaches: The simplest approach to clustering Web Clients is based on prefix length matching of IP addresses of these clients, obtained from the access logs of the web server. Clients which share the first n bits of their IP addresses are said to belong to the same network cluster. For such a clustering to work, the value of n must be known for various classes of IP addresses.² With address aggregation (introduced by Classless Inter-Domain Routing (CIDR) [18]), the applicability of this clustering technique becomes questionable because network address prefixes may have varying lengths (which are not necessarily on octet boundaries). For example, 2 IP addresses 206.205.204.20 and 206.205.204.40 have the same 24-bit prefix but they may lie on two entirely different networks, 206.205.204.0/24 and 206.205.204.32/27, which may be geographically far away from each other.

A more suitable approach is to complement network prefixes with network mask information obtained from Border Gateway Protocol (BGP) [21]. Such a technique was proposed and evaluated in [20]. In this approach, using client IP addresses from web server logs and the collated BGP data from a number of BGP tables, a prefix match is done for the client addresses. Clients having the longest prefix match are clustered together. Though this technique is better (in approach and in accuracy of results) than the aforementioned approach, it depends on frequent transfers and collation of large amounts of BGP information from BGP routers. This would be appropriate to do in an off-line manner. An on-line approach, wherein we use information from communications already taking place in the Internet, using existing Internet infrastructure entities and associated protocols, is more desirable.

²There are different classes of IP address, A (with a network prefix length of 8 bits), B (with a network prefix length of 16 bits) and C (with a network prefix length of 24 bits).

An alternative approach to clustering web clients is based not on the clients IP addresses, but on the domain to which these IP addresses belong. For example, both 128.197.10.4 (`csb.bu.edu`) and 128.197.10.5 (`csd.bu.edu`) belong to the `bu.edu` domain. Using name lookup utilities like `nslookup` and `dig`, a domain name of an IP address can be determined. It is important to note that IP addresses belonging to the same domain may well be on different subnetworks. For example IP address 128.197.12.3 (`csa.bu.edu`) is on a subnetwork different from that of the above two `bu.edu` IP addresses. A network prefix based policy would cluster `csa.bu.edu` and `csb.bu.edu` into different clusters, though in reality they lie in the same administrative domain. However, the methodology described in [20] would cluster these addresses on a higher level (based on BGP data) and would classify them correctly as one belonging to the same administrative domain, but only after some aggregation is performed.

Another clustering technique is based on the distance between clients and servers as measured by `traceroute` [13, 10]. Running `traceroute` is an expensive proposition and is inherently limited by traceable IP addresses.³ These, among other anomalies, limit the extent to which `traceroute distances` can be used effectively for clustering.

The clustering technique proposed in [26] is closest to ours. In that work, a heuristic approach was devised to form `<L-DNS,Client>` clusters based on comparison of time stamps in web server logs and name server logs. Problems resulting from clock skews (between Web server and associated DNS server) and caching by web browsers could have undesirable effects on the accuracy of the clusters generated using this approach. Unlike the work in [26], our approach produces an *authoritative* `<L-DNS,Client>` association of clients to DNS servers. This is due to the architecture we propose and to the deterministic protocol we use to discover such associations (as discussed in Sections 3 and 4).

DNS-based Internet Resource Management: Using DNS is a natural way of balancing (or engineering) load across nodes of a distributed Web serve. DNS-based request distribution could be thought of as a first step in (possibly many) redirection steps that eventually lead a client to one of the many servers that are capable of satisfying the client’s request for service. Early work on using DNS to distribute load across a set of web servers (see [17, 24]) has relied on Round-Robin DNS (RR-DNS) to distribute incoming connections across a cluster of servers. This is done by providing a mapping from a single host name to multiple IP addresses. Recent efforts have extended the purpose of DNS-based request routing beyond simple load balancing. For instance, commercial CDNs leverage DNS by redirecting clients to the CDN proxy that is “closest” to the client’s name server. Due to DNS protocol intricacies (e.g. DNS caching and invalidation), DNS-based request distribution approaches were found to be of limited value. Limitations of DNS-based request distribution approaches are quantified in a number of studies (see [24, 16, 19], for example).

One of the causes of load imbalance when DNS-based schemes are used to distribute load across a set of distributed web servers is that the load resulting from an individual domain-to-IP-address translation varies *significantly*. Later in this paper we show evidence that this variability is quite skewed (heavy-tailed), in the sense that most requests are from name servers that represent clients with a very small percentage of the load, and only very few requests are from “fat” name servers that represent a very large population and thus represent a significant percentage of the load. A RR-DNS request routing scheme does not take information regarding the “load behind” a name server into consideration, and thus would be helpless in mitigating the impact of such load variability (namely a load “imbalance” across the servers). The same holds true for *any* other policy that is incognizant of the load “represented” by a DNS request. The L-DNS clustering protocol proposed in this paper enables the quantification of the “load behind” a name server and thus could be instrumental in optimizing the DNS-based request routing protocols.

3 L-DNS Clustering Algorithm

The basic idea of our protocol is to incrementally discover (and continually maintain) a partition of the population of clients accessing a given (web) server. Clients that belong to the same class in the partition use the same name server to perform the domain-to-IP-address translation (i.e. domain name lookup). The discovery of this partition is done by allowing the name server that serves domain name translation requests for a web server to use a “special” (alias) IP addresses to *color* Internet clients. The protocol works by ensuring that each such special IP addresses is issued for exactly one name server at a time. This enables the association of all clients that use this special IP address with the name server to which this special IP address was advertised.

³Some routers or hosts on the `traceroute` path may disable responses to ICMP messages for security reasons, or may be down at that particular time, or might even reply only intermittently at will [10].

The availability of a mapping of clients to name servers is valuable because it enables a distributed web server to achieve a number of unique functionalities. For example, as motivated in the previous section, it is particularly interesting to estimate the “load quantum” that will result from a particular response to a DNS request—which in turn can be used to improve (possibly significantly) the performance of load balancing/distribution for clustered, as well as geographically-distributed web servers. This functionality is important for load balancers and QoS managers over the wide area. Another particularly valuable use of such mapping is to associate “distance metrics” between multiple servers in a Content Distribution Network (CDN) and a set of clients (represented by a DNS server). This functionality is crucial for distribution schemes employed by CDNs (such as Akamai [1], Digital Island [3], and others). Another important value of this technology is the ability to associate “locales” (say for personalization or localization) to DNS servers.

3.1 Definitions and Terminology

The protocol we present in this section enables a (distributed) Web server and associated name server to incrementally discover (and continually maintain) a partition of its clientele. Clients (or networks) that belong to the same class in the partition use the same name server to perform the domain-to-IP-address translation (i.e. domain name lookup). We call such a class a “*Name-Server-equivalent Cluster*” (NSC). There are four main players in our NSC discovery protocol. We describe these players below, along with other terms we use throughout.

WS (Web Server): This is one of the hosts that the client would eventually contact to access the web site. We assume that WS is configured to have a number of IP aliases. One of these IP addresses is the *Default* IP address (DIP) and the other IP addresses are called the *Special* IP addresses (SIP). One SIP is used to color a NSC. Thus the number of SIP aliases we assign to WS the more concurrent coloring we can perform.

GD (Global Director): This is the Authoritative Name Server for the web site we are managing. The GD is the server that responds to DNS queries for domain-to-IP address translations, and is the entity that controls the issuing of SIPs to color NSC.

IC (Internet Client): This is the client machine. The purpose of this protocol is to assign the IP address of the IC (or its network) to a NSC.

NS (Name Server): This is the (primary) name server that is acting on behalf of the IC. The client will be assigned to the NSC that includes all other ICs that use this NS as their (primary) name server.

3.2 Overview of the NSC Discovery Protocol

We are now ready to overview the NSC discovery protocol. We do so by describing modifications to the protocol that the GD uses (e.g. BIND), which would enable this discovery. For simplicity, we will assume that we want to color only *one* NSC at a time. The discovery process requires—in addition to the Default IP address (DIP)—the use of a Special IP address (SIP) that we will use to color the NSC of interest. As we mentioned before, both the DIP and SIP addresses are valid IP addresses for WS.

The NSC discovery process starts when a NS (the NSC of which we want to identify) queries the GD. Rather than returning DIP to that NS, we return SIP. Since SIP is returned only to that NS, it follows that the first request by an IC to the WS using SIP must be the result of the IC having NS as its name server. Thus for each such incident we can assume that the client belongs to the NSC of NS.

Before detailing the NSC Discovery protocol, we discuss a number of important issues that the protocol must consider.

Length of Time to Enable Coloring of a NSC: The longer we allow the protocol to color a NSC the more clients (or networks) we are likely to classify as belonging to a NSC. As a matter of fact to obtain the full set of client we would need to do this “forever”. However, in practice (and as validated in our experimentation), it is likely that at some point we may reach a point of diminishing return, after which the rate at which we add new clients (or networks) to a NSC decreases.

Canceling the Coloring Process: It is possible that the NSC associated with a name server is “not worth discovering” because it generates so few requests that it warrants ignoring. One way of canceling the coloring process is to check if the load generated from the NSC is below a given threshold and the name server of the NSC fails to come back for a repeat request before some timeout expires. Both of these conditions are necessary.

Black Listing a Name Server: If the load generated from the NSC is large enough (above a given threshold) and the name server of the NSC fails to come back for a repeat request before some timeout expires, then we can assume that the NS is not adhering with the TTL constraint set forth by the GD. Hence, we could “black list” such an NS.

Speed of NSC Mapping versus Number of IP Addresses Used for Coloring: In the discussion above, we have considered what happens when a single name server (representing a single NSC) requests a DNS lookup and how we can use a “special” IP (SIP) to color the NSC. Obviously, we could use multiple SIPs to color multiple NSCs concurrently. The more colors we use the faster the discovery process is likely to be; using n colors would speed up the mapping process n -fold. The disadvantage (of course) is that we have to dedicate many IPs for this process, which may present a problem (from an administration’s point of view). In the pseudo code below, we assume that we have functions that manages the SIP pool (i.e. allocate a SIP, free a SIP, etc.)

NSC Map Compression: An important question, especially for high-volume web sites, is the management of (potentially) very large NSC. Obviously, the question is not the raw space needed to store the client IP addresses for a given NSC, because such space is not really that much, compared to (say) the space needed to store web server logs. The question is how to represent an NSC in a compact data structure that can be easily indexed and searched. To that end, the hierarchical nature of IP addresses suggests a simple tree structure. Traversing this tree structure in a top-down fashion would allow us to identify the various networks and sub-networks belonging to a NSC, along with the cardinality (i.e. number of) clients in each such network.

Synchronization Issues: The coloring of clients using a given SIP starts when that SIP is “issued” to a specific name server and is stopped when that SIP is “withdrawn” from the name server. However, it is possible that a race condition occurs whereby (1) a client issues a DNS lookup and gets back SIP1; (2) the name server of that client later queries the GD and gets a DIP; (3) Another name server queries the GD and gets back SIP1; (4) the client requests the web page using SIP1 and hence gets assigned to the wrong NSC. This race conditions is extremely unlikely. Nevertheless, to minimize the chances of this race condition, the GD should simply wait for a while before recycling a revoked SIP.⁴

3.3 NSC Discovery Protocol

Our NSC discovery starts with a NS requesting a translation. The steps that the GD would take as a result of this request are shown in the function `DNSprocess(query)` shown in Listing 3.3. In addition to these steps, it is necessary to periodically “reclaim” SIPs. The pseudo code for such a process is shown in Listing 3.2. The pseudo code shown in Listings 3.3 and 3.2 make use of a number of timeout parameters. These are defined below.

Time-To-Live (TTL): This sets the expiration time for the translation handed out from the GD to the NS. Typically, to enable more control of load, TTL is set to a small value.⁵

Time-To-Abandon (TTA): This is the length of time we want the GD to wait for a name server before it decides that the NSC associated with that name server is a “deadbeat”. The TTA could be a simple constant threshold value (e.g. 5 minutes) or it could be a dynamically calculated value based on the lack of NSC activity at the WS. In the remainder of this document we assume that the TTA will be a simple constant value. Obviously, TTA will be a multiple of TTL.

Time-To-Color (TTC): This is the length of time we want the GD to keep coloring a particular NSC. The TTC could be a simple constant threshold value (e.g. 10 minutes) or it could be a dynamically calculated value based on the deceleration of the discovery process. In the remainder of this document we assume that the TTC will be a simple constant value. Obviously, TTC will be a multiple of TTA.

⁴Note that an adversarial client can always defeat this approach by waiting for a long time between doing a DNS lookup and sending in the first request to the web server. We do not consider such a scenario to be plausible.

⁵Setting TTL to 0 would force name servers to continuously come back to the GD to obtain new translations (for example to ensure better control over load balancing). Small values (0 in particular) are typically ignored by name servers. One of the advantages of the protocol described in this document is that it enables DNS-based load balancers to tolerate significantly large TTL values.

Time-To-Recycle (TTR): This is the amount of time between reclaiming a SIP from a NS and the time of putting it back in the pool of SIPs for coloring purposes. We need this threshold to avoid race conditions. In particular, this value should be set to be larger than the expected time between when a client request a DNS lookup and when it issues the first request to the web site. This amount of time is typically very small (in the milliseconds).

Listing 3.1 Pseudo Code for function to process a DNS query to the GD

```

Begin DNSprocess(Query){
  If (!Colored(Query.NS)) {
    SIP = GetNextSIP ;
    If (SIP != NULL) {
      SIP->LeasedTo = Query.NS;
      SIP->TimeOfLease = CurrentTime();
      AddToSIPList(SIP);
      AddToColoredList(Query.NS);
      AddToLog("+", CurrentTime, Query.NS, SIP->Address);
    }
  }
  SIP = SIPList ;
  While (SIP != NULL) {
    if (SIP->LeasedTo == Query.NS) {
      If ((CurrentTime()-SIP->TimeOfLease) > TTC) {
        SIP->LeasedTo = NULL;
        Query.Answer = DIP ;
        AddToLog("-", CurrentTime, Query.NS, SIP->Address);
        Sleep(TTR);
        FreeSIP(SIP);
        Return();
      }
      Query.Last = CurrentTime();
      Query.Answer = SIP->Address;
      Return();
    }
  }
  Query.Answer = DIP;
  Return();
}

```

Listing 3.2 Pseudo Code for SIP Reclaimer at the GD (run periodically with a period TTA)

```

Begin SIPreclaimer() {
  SIP = SIPList ;
  While (SIP != NULL) {
    If ((CurrentTime() - SIP->Last) > TTA) {
      If (LoadOnServer(SIP->LeasedTo) < LowLoadThreshold) {
        AddToLog("-", CurrentTime, Query.NS, SIP->Address);
        FreeSIP(SIP);
        Return();
      }
      AddToBlackList(SIP->Address);
    }
    SIP = SIP->Next;
  }
  Return();
}

```

3.4 A Hierarchical Approach to NSC Discovery

The above technique adopts a sequential approach to the discovery of NSC. In other words, it proceeds by coloring (at any point in time) a fixed number of NSC. If the total number of NSC in the Internet is N and we are able to discover n of these NSC every T units of time, then the length of time it takes to color the whole Internet is $(N * T/n)$.⁶ Thus, if $N = 1,000,000$ and $n=100$ and $T=100$ minutes, this results in a time of 1,000,000 minutes ~ 700 days, which is clearly unacceptable.

⁶The values of N , T , and n can vary widely based on the popularity of the web site used to perform the clustering.

The $O(N)$ time complexity of the coloring process can be reduced significantly to $O(\log_n(N))$ by adopting a hierarchical approach to the discovery of NSCs. The basic idea is to recursively partition the name servers in the Internet into n NSC. Here, every T units of time, we are able to subdivide a NSC to n smaller NSCs. Starting with one NSC, it would take $\log_n(N)$ steps to cover all N NSCs. For the above illustrative numbers, it would take only few hours to complete the NSC discovery.

4 An Architecture for NSC Discovery using L-DNS Clustering

We illustrate our proposed architecture to cluster clients around the Local Domain Name Server (NS) they use for name resolution, by using a Web server in conjunction with its Authoritative Domain Name Server.

The typical sequence of events that takes place when the name of a website (e.g., `www.cs.bu.edu`) is looked-up by a browser is: (1) the client queries the local name server (NS) for a name-to-IP translation, (2) if the NS has such a translation in its cache, then it returns it as an answer to the query, otherwise (3) NS queries other name servers, as specified in [22] until it gets an answer, or it reaches an Authoritative Name Server for the domain name.⁷ Once the client gets the answer back, it can go directly to the Web Server, using the IP in the answer.

Architecture Overview: In our architecture, we insert two modules to be invoked as part of the above event sequence: The first module is called COLOR and it interfaces with the GD. The second module is called WSI (Web Server Interface) module and it interfaces with WS. The NSC discovery algorithm presented in Section 3 is implemented within these two modules, along with minimal instrumentation of both a Name Server (NS) and a Web Server (WS). Figure 1 depicts our architecture and the flow of information between the entities in this architecture.

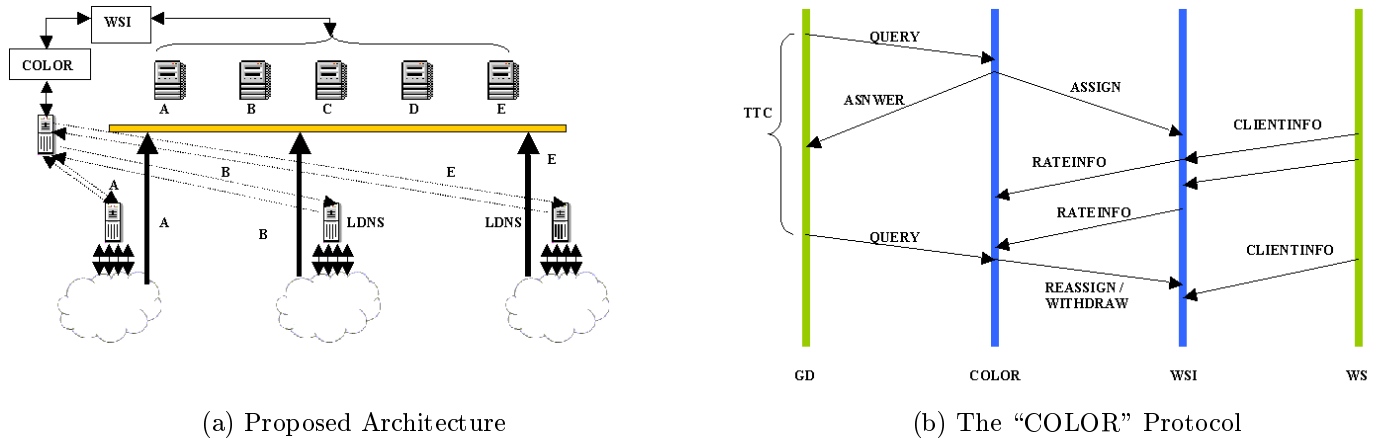


Figure 1: L-DNS Clustering

Web Server Instrumentation: For the basic implementation of our algorithm we used two Web servers hosting a “live” cultural Web site `www.coptics.net`. A *primary* server (whose IP is “widely known”) is used as the main server for the site. Along with it, we use a *replica* server, which mirrors the content on the primary server and is used as the server for coloring/clustering clients. We set up the primary server on a Solaris 5.6 box. As mentioned before, we refer to the IP address of this host as DIP. The replica server is set up on a Linux (Kernel 2.2.14) box using the Apache Web Server (Version 1.3.14) [2]. We set up the network interface on the replica machine to listen to 4 Special IP (SIP) addresses. The Apache Server on this machine is configured so as to listen to HTTP requests on *all* four of these SIP addresses. We instrumented the Apache Web server so that it relays the tuple $\langle \text{WebClientIP}, \text{WebServerIP} \rangle$ to the WSI module. This tuple contains the IP address of the Client that accesses the replica server, and also of the Webserver IP address it uses to access the replica server—in other words, one of the SIP addresses.

⁷An Authoritative Name Server for a domain name is the name server, which by definition has the answer to the query sent by the client.

Authoritative Name Server Instrumentation: We use the Berkeley Internet Name Distribution (BIND) software [4, 8] to set up the name server—i.e., the Global Director (GD)—for the Web site. We registered [6] one of our machines as an Authoritative DNS for the site⁸ `www.coptic.net`. We log all NS accesses to our GD.

We instrumented the GD such that for resolving names of `www.coptic.net` and `coptic.net` (Class IN, Type A queries), instead of picking up IP addresses from the Zone database, it queries the COLOR module. If the COLOR module is running, implying that coloring is in progress, it returns one of the available SIP addresses to the GD, which in turn returns it to the querying NS. If an SIP is not available then the original information from the zone database is returned by the GD.

The COLOR Protocol Implementation: The coloring protocol defines the messages passed on the interfaces between GD and COLOR, between COLOR and WSI and between WSI and the Web Server. Figure 1 shows the messages passed between these various parties. Messages of type Query and Answer are exchanged between GD and COLOR—GD generating the queries and COLOR answering the queries with either SIP (if available) or DIP. Along with the IP address, COLOR also passes along a Time To Abandon (TTA) parameter, which is returned to the NS as the standard Time To Live (TTL) of the Resource Record returned by a Name Server to the NS. During this time all Internet Clients (IC) that access the NS to resolve the name `www.coptic.net` will receive the cached answer. We want ICs to use this answer for TTA units of time. TTA is a configurable parameter of our architecture and we aim to work with different values of TTA and observe the efficacy of these values for our algorithm.

The exchanges between COLOR and WSI form the heart of the L-DNS clustering protocol. COLOR informs WSI whenever it assigns an SIP to an NS as result of a query from GD to COLOR. It also informs WSI whenever it withdraws a SIP given to an NS after its TTA expires. WSI maintains this mapping of NS to SIP, and accordingly clusters clients accessing the webserver with a particular SIP. The protocol facilitates the flow of client access information from the Web Server to WSI. The IP address of every client accessing the replica server is sent to the WSI along with the IP address it accessed the replica with.

As an illustration, assume that a client with IP address 128.197.12.3 requests a lookup for `www.coptic.net` at time t . Assuming that the NS (128.197.27.7) did not have the answer in its cache, it requests a lookup from the GD. Assuming that at that time, COLOR has one SIP (128.197.14.37) unassigned. It returns to the NS (128.197.27.7), this IP address as the answer to the query through the GD and also communicates the association $\langle 128.197.14.37, 128.197.27.7 \rangle$ to WSI. Now, the client accesses 128.197.14.37 over HTTP to get the content from the Web site. The Web server running on 128.197.14.37 records and sends the tuple $\langle 128.197.12.3, 128.197.14.37 \rangle$ to WSI. Using this in conjunction with the DNS-to-SIP mapping, WSI deduces the association of the client $\langle 128.197.12.3 \rangle$ with the NS $\langle 128.197.27.7 \rangle$. During the time $[t, t+TTA]$ all clients accessing 128.197.27.7 for resolution of `www.coptic.net` will get the same SIP address $\langle 128.197.14.37 \rangle$. At the end of this time period WSI can log all the clients that used $\langle 128.197.14.37 \rangle$ as being clustered around the NS $\langle 128.197.27.7 \rangle$.

WSI also sends periodic access statistics to COLOR. Specifically, it reports how many clients accessed the web server using a particular SIP in a given time interval. Based on this information, COLOR may decide to reassign the same SIP to the NS instead of withdrawing it after the TTA period has expired.

5 Internet Client Clustering and Characterization

In this section we present the results of experiments we conducted on the Internet to cluster clients using the L-DNS clustering protocol/architecture described in the previous sections. We present the characteristics of the L-DNS clusters we obtained and compare those to results obtained using other client clustering approaches.

5.1 Characteristics of Name Server Access Patterns

We log access to the the Authoritative Name Server (GD), using different TTLs for the data in our zone (`coptic.net`). We extract the IP addresses of the NSs accessing the GD at three different periods of time and with different values

⁸For the purpose of this paper, we omit the discussion of Secondary Name Servers and Zone transfers, in the context of Authoritative Name Servers.

of TTL. A period of 6 days (from the 6th of December, 2000 to the 11th of December, 2000) which we refer to as **Period 1**. A period of 38 days (from the 12th of December, 2000 to the 18th of January, 2001), referred to as **Period 2**. Finally a period of 16 days (from the 22nd of February, 2001 to the 10th of March, 2001), referred to as **Period 3**.

For **Period 1** we set a TTL for the records handed out by GD, to the NS requests to be one day and for **Period 2** we set the TTL to be one hour. **Period 3** is a special case because in this period we ran our coloring algorithm. We set up the TTL for records returned to NSs that were not being colored to be one minute and the TTA (same as the TTL) for the NSs that were being colored to be five minutes.

	Total NS Hits	Unique NS hits
Period1	1,874	1,221
Period2	19,009	4,517
Period3	10,754	2,939

Table 1: Number of NS accesses logged on the GD for three Time Periods

Table 1 gives the number of NS accesses during these 3 periods. In Figure 5.1(a) we present the number of GD access of these unique NSs in the three time periods. We plot the access frequencies of an NS against the rank of that NS (lower rank = higher frequency of access). In all three graphs we observed a heavy-tailed distribution for the frequency of NS access vs rank.

We extracted the number of client hits we logged during this period, both on the primary web server and on the replica web server. Table 2 shows the total number of client accesses as well as the number of accesses to the replica server. It also shows the number of unique client hits that were observed (for both servers) and those that were clustered during the coloring period. By a client access, we mean a request by a client for a web object and by a unique hit, we refer to the number of unique clients logged, irrespective of the objects they request and the time at which they access the object.

	All NS	NS Colored	Access by All Clients	Access by Clustered Clients
Total Accesses	10,754	6,870	185,256	54,794
Unique Hits	2,939	2,570	30,783	4,249

Table 2: Hits recorded in the logs of the GD and WS during the coloring period (from 02/22/01 to 03/09/01)

We also obtain the distribution of interarrival times of all the NS requests, in the three periods, **Period 1**, **Period 2** and **Period 3**. Figure 5.1(b) shows these distributions. We see that for **Period1** 64% of the requests conformed to the TTL value of one day, returned by GD. For TTL value of one hour we had 92% of the request conforming and for a TTL of one minute we had 89% requests conforming. Of the requests in **Period3**, 84% of the colored NSs conformed to the five minutes TTA. Thus, we conclude that most of the configurations of NSs running on the internet assiduously follow the TTL value returned by the Authoritative servers (as long as that value is not too large).

5.2 L-DNS Clustering Results

We ran our L-DNS clustering algorithm for a period of 16 days and recorded the clusters that were produced. During this period, we performed the coloring for a total of 6,870 times and detected 2,570 unique clusters. The number of clients colored in total was 5,778 of which, 4,249 were unique clients. The maximum number of clients we discovered for a particular NS was 56 and the length of time for which this particular NS was colored equalled 7.5 hours.⁹ This is also the maximum time for which any of the NSs were colored. The minimum number of clients we discovered in a cluster was zero. This can be explained by noting that it is possible for a client to perform a name look-up without

⁹The long time taken for coloring is an indication of the “unpopularity” of the web site we used for our experiments. In particular, we were able to cluster at the rate of one client for every 64 hits (\approx one client every 8 minutes) to the web site. Had the web site used in our experiments been a popular server that commands (say) 100 million hits per day, we would have been able to cluster approximately 1,600,000 clients over a period of one day!

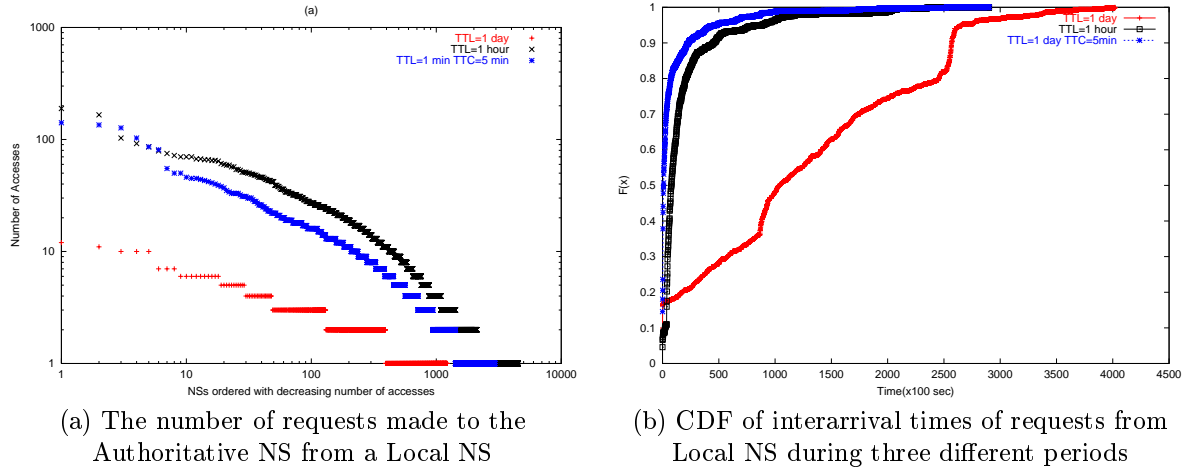


Figure 2: Characterization of requests made to the Authoritative Name Server by Local Name Servers

accessing the web server (e.g., as a result of *nslookup* or *dig* commands). In such a scenario the client IP is not recorded by the web server but the NS is registered with the GD. Of the 2,570 unique clusters, 435 clusters are due to this phenomenon.

In our results, we find a high degree of correlation (0.79) between the time for which an NS is colored and the number of clients discovered in this NS's cluster. We intend to study further the marginal utility of incrementing the coloring period for a given NS.

5.3 P-DNS Clustering Results

To get a feel of how a simple prefix matching would fare as a clustering method, on the 4,249 (2,570 unique L-DNSs) $\langle \text{NS}, \text{Client} \rangle$ pairs that we generated, we clustered clients around the NS whose IP address has the most number of prefix bits in common with a prefix of the client IP address.

For this clustering we calculate the *mean error coefficient* as the mean of the error coefficients, ϵ for each $\langle \text{P-DNS}, \text{Client} \rangle$ due to this prefix based clustering, where ϵ is defined as

$$\epsilon = \begin{cases} 1, & \text{if } L - \text{DNS} \neq P - \text{DNS} \\ 0, & \text{if } L - \text{DNS} = P - \text{DNS} \end{cases}$$

We found the *mean* ϵ to be 0.659, which indicates that if we make use of a simple IP address prefix based scheme to cluster clients around NSs, we would cluster around 66% of the clients incorrectly! This shows the value of our L-DNS clustering approach.

Figure 3(a) gives the distribution of the number of clients clustered using the P-DNS approach as well as the L-DNS approach. This Figure shows that the largest cluster sizes using our L-DNS algorithm (=56) was smaller than that found using P-DNS (=82). The difference between the two results supports the fact that in the internet IP addresses are not the most reliable way to discern information about the administrative domain of a web client. Some other property has to be exploited; our algorithm makes use of such a property.

Figure 3(b) shows the distribution of clusters with the mean number of bits matched in each cluster. For this distribution we take the mean number of bits matched between the NS and its clients, which have been clustered using the prefix-match algorithm. The mean and the median number of bits matched for all the clusters are 11.1 and 7 respectively.¹⁰

¹⁰The peak at 32 bits is attributed to local Name Servers running on the same machine as the origin of a web request by the client (e.g. Web robots), resulting in the NS being associated with itself as a client and hence resulting in a prefix match of 32 bits, using the prefix-match algorithm.

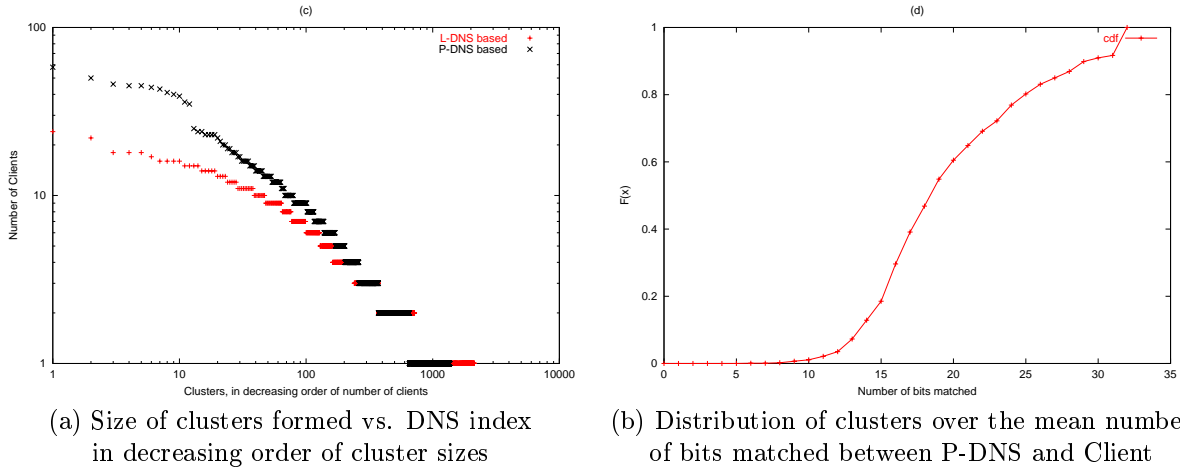


Figure 3: Characteristics of cluster sizes formed using the L-DNS and P-DNS protocols

5.4 Correlation of RTT Metric

Web clients and NSs lying geographically close to each other—for example on a common network (e.g. on a campus network such as Boston University, or on an office network)—would exhibit a high degree of correlation between the round trip times (RTTs) of ping packets from a source to the NS and the RTTs of ping packets from the same source to the web clients. Web clients on a commercial network (e.g. America Online) would show less (or no correlation) with their NSs. We explore quantitatively this correlation using our cluster information.

Using the clusters we obtained using our L-DNS clustering approach, we performed *pings* from a source in Boston University to each NS as well as to all the clients clustered around that NS. During the course of each experiment, we ping each IP address in each cluster in a round-robin PASTA fashion (with the inter-ping times between two ping packets being exponentially distributed, with a mean of one second) for a total of 10 times. Of the 6,819 IP addresses (2,570 NSs and 4,249 clients) that we pinged, we were able to obtain 1,859 $\langle \text{NS}, \text{Client} \rangle$ pairs that responded to our pings. The correlation coefficient (for the ping RTTs to the NS and to the clients) was found to be 0.2835.

To increase the representativeness of our ping RTT correlation measurements, we performed the above experiment (pinging NSs and clients that make up the L-DNS clusters around these NSs) from three additional geographic locations in the US (namely, University of Illinois at Chicago, Purdue University, and Harvard University). Figure 5.4 shows a schematic diagram of our measurement methodology.

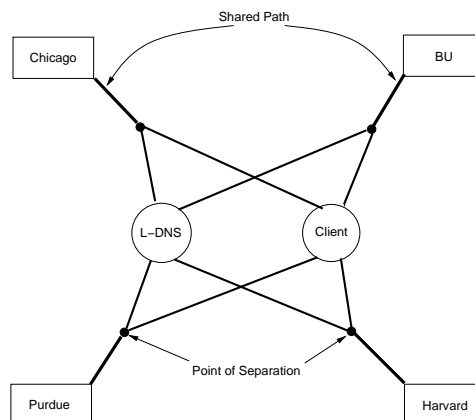


Figure 4: Schematic representation of the paths from source to the NS-Client pairs, used for our measurements

The results of these experiments (from the four geographic locations) are shown in Table 3. These results show that the correlation between ping RTTs to a client and its NS is surprisingly low (average = 0.3280). Thus, inspite of being on the same administrative network, a client and its NS are not necessarily “close” to each other in term of network latencies to a given host. This shows that CDN redirection schemes based on latency measurements between CDN nodes and a client’s NS are unlikely to yield “optimal” server selection decisions, confirming the anomalies described in [19].

Table 3 shows that the correlation coefficient depends on the origin of the experiment (BU versus Purdue, for example). This dependency can be explained by noting that the positive correlation between ping RTTs to a client and its NS is due to the shared portion of the path from a source to a client and its associated NS (as illustrated in Figure 5.4). The “longer” this shared path, the more we expect the ping RTTs to be correlated (i.e. dependent). Notice that all our observation points were hosts within subnets of academic institutions. This implies that the shared portion of the paths to a client and its NS observed from such hosts are likely to be *longer* than those observed from hosts that are located closer to the Internet core—proxies of CDNs at collocation points, for example. Thus, the weak correlation coefficients shown in Table 3 are likely to be even weaker when measured from such CDN proxies.

Ping RTT correlations are even weaker when the accuracy of the clustering approach is lower. To document this, we repeated the above experiments using the P-DNS and R-DNS clustering approaches.¹¹ For experiments conducted from Boston University, the correlation coefficient for P-DNS clustering (0.1520) was found to be lower than that of L-DNS clustering (0.2835). And, as one would expect, the correlation coefficient for R-DNS clustering (0.0257) was found to be negligible.

It is important to note that the RTT correlation coefficient of 0.3280 is an average over all clusters we obtained. The correlation coefficient was not uniformly weak across all cluster. Specifically, we found that for some clusters, the RTT correlation coefficient was quite high, whereas for others it was negligible. This indicates that it is possible to characterize an L-DNS cluster based on whether or not such a cluster exhibits a strong correlation with respect to RTT. This characterization could be quite valuable for CDN request routing. Specifically, if the RTT correlation coefficient for the cluster around a NS is known to be high, then a CDN could reliably assign the server with a minimum RTT to the NS and be assured that such a choice would be (near) optimal for ICs belonging to such an NS. Alternately, if the RTT correlation coefficient for the cluster around a NS is known to be low, then a CDN may base its request routing decision on other metrics (e.g. proxy load).

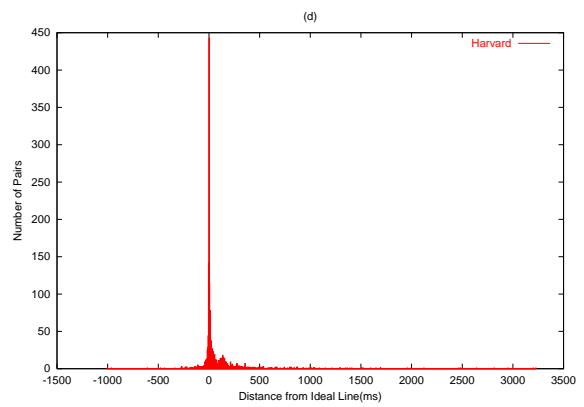
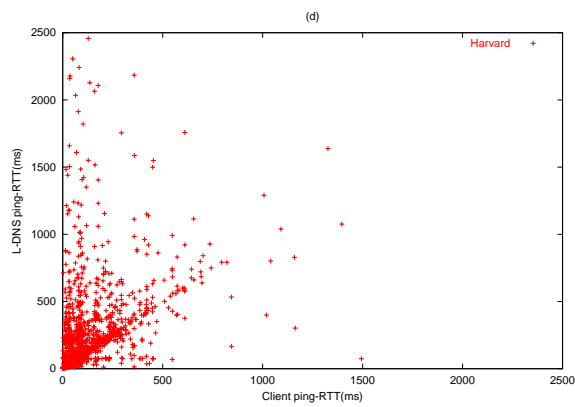
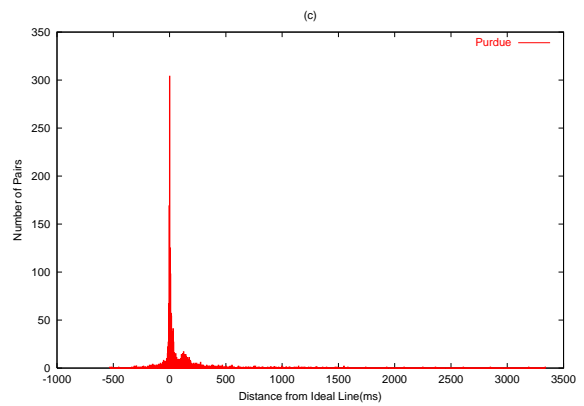
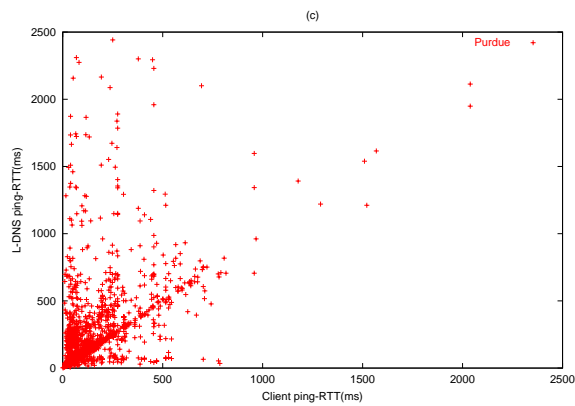
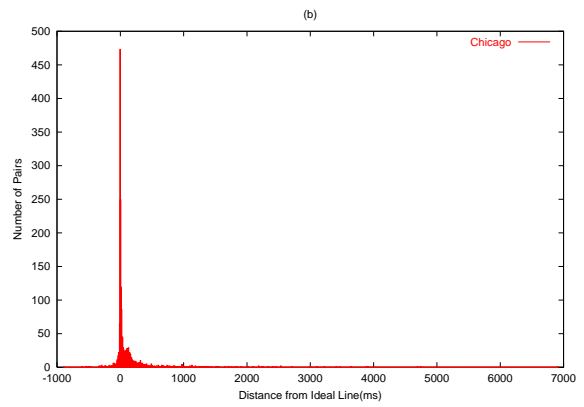
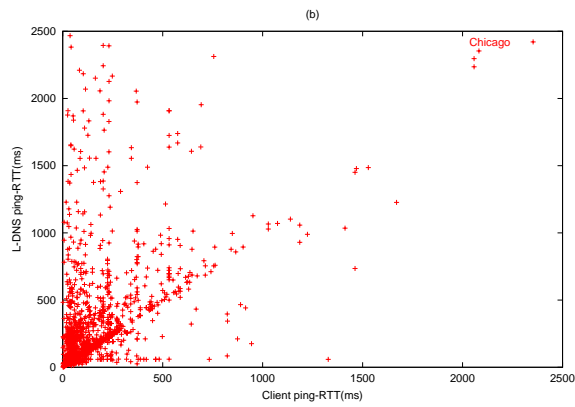
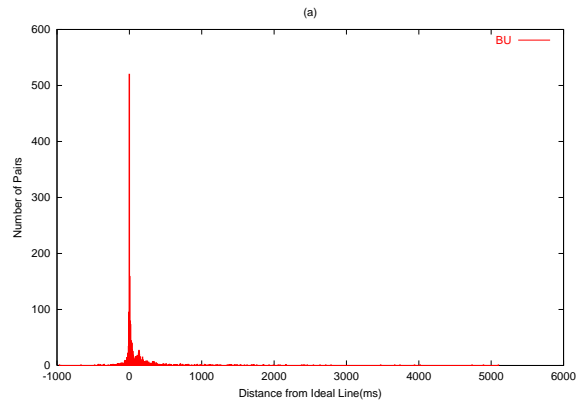
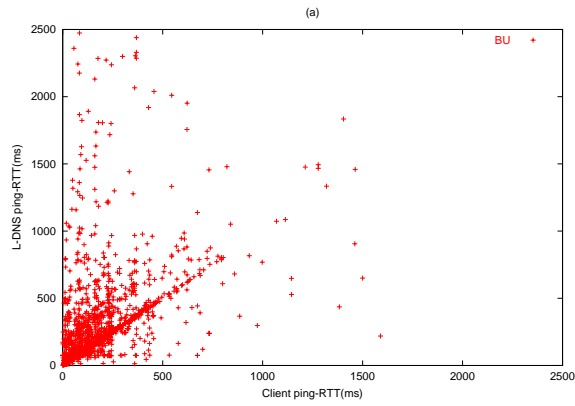
Location	Correlation using L-DNS	Correlation using R-DNS	Correlation using P-DNS	Standard Deviation (msec)
BU	0.2835	0.0257	0.1520	411.19
Chicago	0.2867	0.0239	0.1716	443.92
Purdue	0.4073	0.0297	0.2267	264.54
Harvard	0.3346	0.0249	0.2167	280.74

Table 3: Correlation for ping-RTT measurements to L-DNS-Client, R-DNS-Client and P-LDNS-Client pairs from different locations

In Figure 5(I) we plot the NS ping-RTT against client ping-RTT measured from four locations (BU, Chicago, Purdue and Harvard). We observe that some of the points on the plot lie at extremities, much further than the ideal-case line fit on the dataset. In other words, some of the clients are very far from their L-DNS and vice versa, in terms of network latencies to the source of the measurements. To estimate the deviations of the measured times from those in the “ideal” case, we calculated the standard deviation of the distances of the points from a hypothetical 45-degree line passing through the origin. This ideal case is a scenario, wherein the client and its NS are very close to each other, just like on a campus network, or even on the same host. Table 3 quantifies these deviations.

From Figure 5(II) we can see that although there are large deviations from the “ideal case”, a fair number of clients lie close to their NSs. As we hinted before, identifying clusters for which this is true (or not true) could benefit DNS-based server selection in CDN networks. Specifically, by indicating that for highly-correlated (or highly uncorrelated) clusters a server that is close to the NS is likely (not likely) to be close to the client. This makes it possible to use *different* CDN server selection approaches for different clusters.

¹¹Recall that P-DNS uses IP address prefix matching to assign a client to a NS, whereas R-DNS assigns a client to a random NS.



(I) L-DNS ping-RTT vs. Client ping-RTT

(II) Distribution of distance of the pairs from the ideal line

Figure 5: Comparison of ping RTTs to L-DNS-Client pairs, (a) From Boston (b) From Chicago (c) From Purdue (d) Harvard

5.5 Correlation of Hop-Count Metric

We performed measurements of proximity between a client and its NS using the hop-count metric. We used the *traceroute* tool to obtain routes to each NS and its clients (identified using L-DNS clustering). We did this measurement from three different locations (Boston University, Chicago, Purdue, and Harvard). We calculated the *common-hop-count* metric for $\langle \text{NS}, \text{Client} \rangle$ pairs, which responded to ping measurements. The common-hop-count metric is the number of hops that are common between the path to the client and to NS from the same *traceroute* source.

As with ping measurements, we computed the correlation between the number of hops to the client and the number of hops to its NS. Table 4 presents this correlation. From Boston University, we find that the correlation (0.6574) is higher than that observed for the ping RTT measurements (0.2835). This follows simply from the weak correlation between hop-count and latency metrics (documented in many studies, including [15] for example).

To quantify the difference in “accuracy” between L-DNS clustering and P-DNS clustering (and to compare both to the baseline R-DNS clustering), we measured the hop-count correlation for P-DNS and R-DNS. The results (shown in Table 4) indicate that the correlation is lower for P-DNS, and is much lower for R-DNS (as expected).

Location	Correlation using L-DNS	Correlation using R-DNS	Correlation using P-DNS
BU	0.6574	0.0086	0.4394
Chicago	0.6695	-0.0144	0.4879
Purdue	0.6606	0.0128	0.4763
Harvard	0.5737	0.0273	0.4542

Table 4: Correlation measurements for traceroute to L-DNS-Client and R-DNS-Client pairs from different locations

Figure 6 presents the cumulative distribution of the ratio of common-hops to hops in the path to the client measured from four locations, namely Boston University, Chicago, Purdue and Harvard. Clustering using R-DNS gives the highest probability for the ratio being less than some value, even for low values of the ratio. Comparing L-DNS and P-DNS curves again demonstrates the superiority of our protocol. For the same probability, the L-DNS based approach gives a higher value for the ratio than the P-DNS based approach. For example, ratios of 0.53 for the L-DNS approach and 0.44 for the P-DNS approach have the same probability of 0.53. This indicates that there is an almost 10% higher probability that with our approach we will form pairings in which a client and an NS share longer portions of the path to an arbitrary server.

As we observed for RTT correlation measurement, Table 4 shows that the correlation coefficient for hop-count depends on the origin of the experiment. Again, this dependency can be explained by noting that the positive correlation between hop-count to a client and its NS is due to the shared portion of the path from a source to a client and its associated NS (as illustrated in Figure 5.4). The “longer” this shared path, the more we expect the hop-count to be correlated (i.e. dependent). This leads us to the same conclusion we observed for RTT correlation measurement—namely that the correlation coefficients shown in Table 4 are likely to be weaker when measured from hosts that are closer to to the Internet core—proxies of CDNs at collocation points, for example.

6 Conclusion

Summary: In this paper we proposed a novel approach for clustering Web clients around the local name server they employ. We have demonstrated that this clustering fares better at grouping together NSs and clients which are more correlated, in terms of network latency and network hop-count (compared to clustering heuristics that rely on matching client IP addresses).

We have implemented and deployed our L-DNS clustering protocol. Measurements we have collected suggest that the correlation between an Internet Client and its Name Server—with respect to various distance metrics (e.g. latency, bandwidth, hop-count) measured from an arbitrary host—is weak, but not universally so. In particular, we observed that the level of correlation is cluster-specific, and that it could be efficiently characterized in a near-real-time fashion.

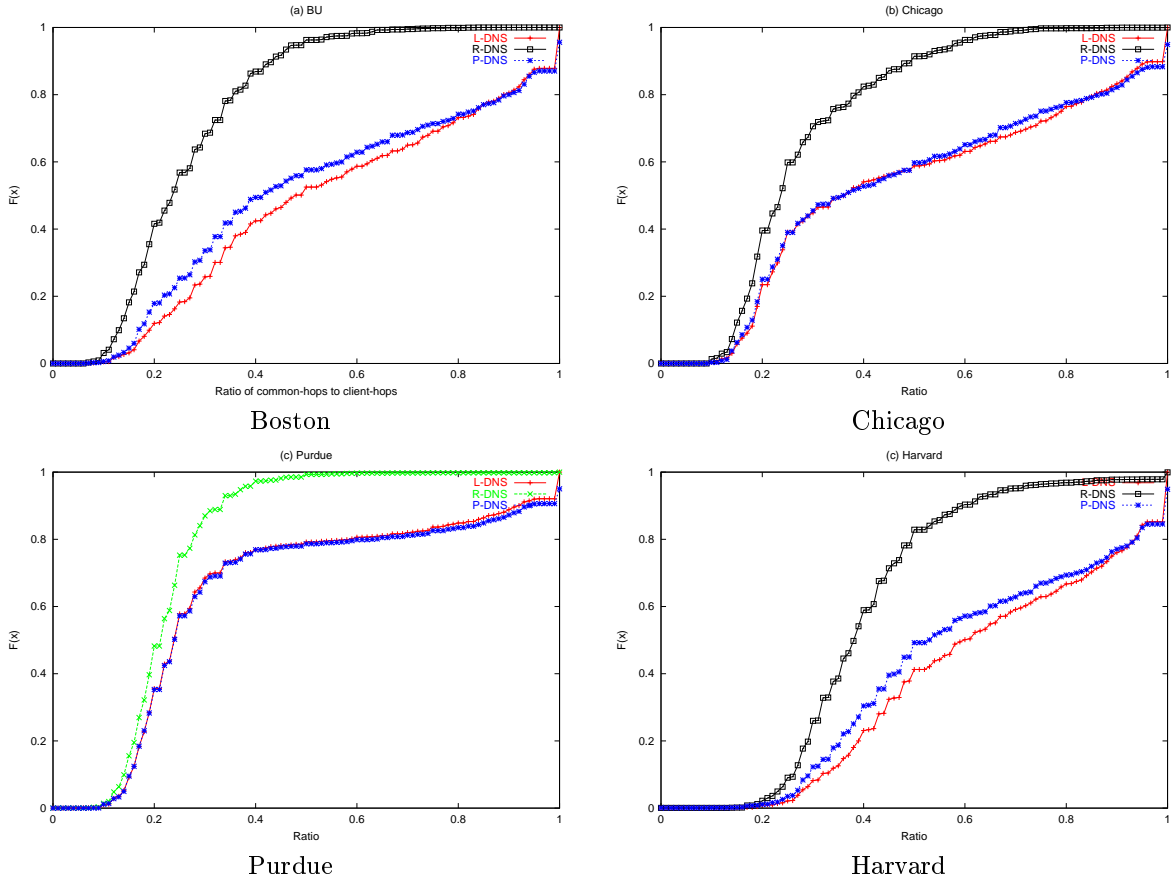


Figure 6: CDF of Ratio of Common-Hops to Client-Hops 3 types of clusterings, measured from Boston, Chicago, Purdue, and Harvard.

On-Going Work: Our findings have significant implications on DNS-based request routing mechanisms employed in Content Distribution Networks (CDNs) because they call into question the wisdom of basing request routing decisions on the “proximity” (with respect to some performance metric) of a CDN proxy to the client’s name server. Doing so is warranted *only* for name servers whose L-DNS cluster exhibits a high-enough correlation with respect to the metric of interest.

Our on-going work aims at extending our results to enable characterization of other metrics of interest, including loss rates, maximum and minimum bandwidth, and jitter. Also, we are working on novel DNS-based request distribution algorithms that take L-DNS cluster characteristics (e.g. size, correlation with respect to various metrics, etc.) into account when making request routing decisions. Initial results suggest that such algorithms are inherently superior to algorithms that do not take such characterization into consideration.

Acknowledgments

We are grateful to the Coptic Network for allowing us to use their domain name and web site to implement our L-DNS clustering. Also, we would like to thank Prof. Kihong Park at Purdue University and Mayank Rawat at the University of Illinois at Chicago for allowing us access to their laboratories for the characterization component of this study. Last but not least, we would like to thank members of the Web and InterNetworking Group (WING) at Boston University for their feedback and support.

References

- [1] <http://www.akamai.com>.
- [2] <http://www.apache.org/>.
- [3] <http://www.digitalisland.com>.
- [4] <http://www.isc.org/products/BIND/>.
- [5] <http://www.mids.org/weather/>.
- [6] <http://www.networksolutions.com>.
- [7] Y. Chawathe A. Fox, S. Gribble and E. A. Brewer. Cluster-based scalable network services. In *SOSP '97*, pp. 78-91, St. Malo, France, 1997.
- [8] Paul Albitz and Cricket Liu. *DNS and BIND*. O’Riely, 1998.
- [9] Luis Aversa and Azer Bestavros. Load Balancing a Cluster of Web Servers Using Distributed Packet Rewriting. In *IPCCC’2000*, 2000.
- [10] Paul Barford, Azer Bestavros, John Byers, and Mark Crovella. On the Marginal Utility of Deploying Measurement Infrastructure. Technical Report BUCS-TR-2000-018, Boston University, Computer Science Department, July 2000.
- [11] Paul Barford and Mark Crovella. Critical Path Analysis of TCP Transactions. In *ACM SIGCOMM*, 2000.
- [12] R. Fielding J. Gettys J. Mogul H. Frystyk T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. In *IETF RFC 2068*, January 1997.
- [13] Azer Bestavros and Carlos Cunha. Server-initiated Document Dissemination for the WWW. In *IEEE Data Engineering Bulletin*, 19:3-11, September 1996.
- [14] T. Brisco. DNS Support for Load Balancing. In *RFC 1794*, 1995.
- [15] Robert L. Carter and Mark E. Crovella. Dynamic Server Selection Using Bandwidth Probing in Wide-Area Networks. In *TR-96-007*, Boston University Computer Science Department, March 1996.
- [16] Daniel M. Dias, William Kish, Rajat Mukherjee, and Renu Tewari. A Scalable and Highly Available Web Server. In *IEEE COMPCON’96*, 1996.
- [17] Michelle Butler Eric Dean Katz and Robert McGrath. A Scalable HTTP Server: The NCSA Prototype. In *Computer Networks and ISDN Systems*, May 1994.
- [18] V. Fuller, J. Yu T. Li, and K. Varadhan. Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy . In *RFC 1519*, September 1993.
- [19] Kirk L. Johnson, John F. Carr, Mark S. Day, and M. Frans Kaashoek. The Measured Performance of Content Distribution Networks. In *5th International Web Caching and Content Delivery Workshop*, May 2000.
- [20] Balachander Krishnamurthy and Jia Wang. On Network-Aware Clustering of Web Clients. In *ACM SIGCOMM*, Stockholm, August 2000.
- [21] K. Lougheed and Y. Rekhter. A Border Gateway Protocol (BGP). In *RFC 1163*, June 1990.
- [22] P. Mockapetris. Domain Names - Implementation and Specification. In *RFC 1035*, November 1987.
- [23] Paul V Mockapetris and Kevin J Dunlap. Development of the Domain Name System. In *ACM Symposium proceedings on Communications architectures and protocols*, 1988.
- [24] Jeffery Mogul. Network Behavior of a Busy Web Server and its Clients. In *Research Report 95/5*, Western Research Laboratory, October 1995.
- [25] Peter Druschel Mohit Aron, Darren Sanders and Willy Zwaenepoel. Scalable Content-aware Request Distribution in Cluster-based Network Servers. In *USENIX 2000 Annual Technical Conference*, June 2000.
- [26] A. Shaikh, R. Tewari, and M. Agrawal. On the Effectiveness of DNS-based Server Selection. In *IEEE INFOCOM*, 2001.