Locating Copies of Objects Using the Domain Name System

Jussi Kangasharju, Keith W. Ross
Institut Eurécom
Sophia Antipolis, France
{kangasha,ross}@eurecom.fr
James W. Roberts
France Télécom – CNET
Issy les Moulineaux, France

Abstract

In order to reduce average delay and bandwidth usage in the Web, geographically dispersed servers often store copies of popular objects. For example, with network caching, the origin server stores a master copy of the object and geographically dispersed cache servers pull and store copies of the object. With site replication, objects stored at master are replicated into secondary sites. In this paper we propose a new network application, Location Data System (LDS), that allows an arbitrary host to obtain the IP addresses of the servers that store a specified URL. Our networking application is an extension to the Domain Name System (DNS), requires only small changes to the domain name servers, and can be deployed incrementally. For the case of network Web caching, we elaborate on our proposal to allow a cache to (i) update a distributed database when it stores or evicts objects, and (ii) push objects to parent caches in order to improve delay and bandwidth usage. For the case of mirrored servers, we show how a client can obtain a list of all servers mirroring all or part of the desired site. LDS applied to partially mirrored sites generates substantially less DNS traffic than LDS applied to caching. Finally, we discuss how a host can use the location data in order to make intelligent decisions about where to retrieve desired objects.

1 Introduction

Network caching of documents has become a standard way of reducing network traffic and latency in the Web. Caches are currently employed in institutional, local, regional and national ISPs. Cache hierarchies, created when caches in lower-level ISPs point to caches in higher-level ISPs, are currently prevalent in the Internet [8, 9]. Today's cache hierarchies use static, manually configured pointers to define the hierarchy tree. Cache hierarchies operate as follows. When a browser requests a document, it sends a request to a leaf cache. This cache then either serves the document (if it is cached) or forwards the document to its parent in the hierarchy. The process is repeated along a static chain of caches until the root of the hierarchy is reached. If there is also a cache miss at the root, the root forwards the request directly to the origin server. A response is returned along the cache chain in the reverse direction. Cooperating caches in cache hierarchies often use ICP (Internet Cache Protocol) to improve and enlarge the scope of the search [15].

Caching hierarchies have several problems. First, requests for less popular documents will experience misses at all caches in the cache chain. For deep hierarchies, these misses lead to poor latency performance [13]; moreover, ICP can further degrade performance, since the cache must wait for a reply from all sibling and parent caches or until a two second timeout before proceeding up the hierarchy.

Second, today's caching hierarchies are static — they do not permit a browser or a cache to choose the subsequent cache according to current topology or traffic conditions. Manual optimizations, such as sending requests for certain top level domains to designated parent caches, are possible, but even with these optimizations, the hierarchy for a given URL remains static. Third, caching hierarchies do not allow the chain of caches to extend beyond the root cache; if there is a miss at the root server, the request is forwarded directly to the origin server, and never to a cache that lies somewhere in between the root server and the origin server. This is a problem because a cache nearby the origin might be able to serve an object much faster than the origin server, especially when the origin server runs on a slow machine or has a low bandwidth connection.

In this paper we propose a new cooperative caching scheme that has the following features. (1) At most two servers (including the origin server) are visited in the request chain; (2) The chain of caches depends on the requested URL and can change dynamically as a function of current network topology and traffic conditions; (3) An arbitrary cache in the Internet can be queried, including a cache that is far from the browser but close to the origin server. (4) The scheme can be incrementally deployed with minor changes to DNS servers. Furthermore, although a thorough performance study is still required, we feel the scheme should lead to a substantial reduction in delay and network traffic as compared to traditional hierarchical caching.

Our caching scheme makes use of a new network application, the Location Data System (LDS), which we also define in this paper. The LDS, defined as an extension of DNS, allows an arbitrary host to obtain the IP addresses of the servers that store a specified URL. The LDS is of independent interest, and can be used for other applications, including choosing the best mirrored site for a given URL. For the case of network Web caching, we specify how a cache updates the LDS distributed database when it stores and evicts objects, and how a cache pushes objects to parent caches in order to improve delay and bandwidth usage.

We recognize that LDS applied to Web caching significantly increases the number of DNS messages. In this paper we also show how LDS can be applied to replicated and partially replicated servers. In this case, the amount of DNS messages does not increase.

This paper is organized as follows. In Section 2 we define the LDS. In particular, we show how DNS can be extended to provide the LDS service. In Section 3 we show how network caches can exploit the LDS; in particular, we discuss how the caches update the LDS distributed database, and how caches at higher levels in the caching hierarchy can be populated. In Section 4 we show how document and site replication can exploit the LDS. In Section 5 we discuss how a host can make routing decisions based on the results of an LDS query. In Section 6 we discuss related research on cooperative caching. Section 7 presents directions for future work and Section 8 concludes the paper.

2 Location Data System (LDS)

Copies of an object, with each object referenced by the same URL, are often available from different servers in the Internet. These servers include **origin servers**, **replicated servers** (also called mirrored sites) and **cache servers** (also called proxy servers). The origin server for an object is the server at which the object originates; it always contains an up-to-date copy of the object. A replicated server contains copies of objects that have been placed into it (typically manually or by pulling); typically, objects in replicated servers are up-to-date. A replicated server may replicate entire Web sites or may replicate only portions of various Web sites. Cache servers obtain copies of objects by pulling them on demand. In particular, when a cache server receives a request for an object, and if the object is not cached, the cache server retrieves the object from another server (which may be the object's origin server, a replicated server, or another cache server), stores a copy of the object, and forwards a copy to the requestor. A cached copy of an object may not be fully up-to-date. In this paper we refer to all three types of servers as **object servers**.

It is highly desirable for a host in the Internet to be able to determine the locations (i.e., the IP addresses) of all the object servers that contain copies of a specified URL. In particular, a host would like to be able to give a network application a URL and receive from the application a list of all the object servers that contain the URL. In addition to receiving the IP addresses of all the servers that contain the object, it is desirable to receive information about the freshness of each copy, e.g., when each copy was last modified or the "age" of the object (as defined in the HTTP/1.1 [2]).

In this section we outline a new networking application that provides the service of mapping a URL to a list of object servers that contain the URL, with each server on the list having associated freshness information. We refer to this system as the **Location Data System (LDS)**. We have taken a rather pragmatic approach in designing the LDS. Our principle goal has been to design a system that can be rapidly deployed with incremental changes to the existing Internet infrastructure. A second goal is scalability, i.e., a system that is decentralized and hierarchical. A third goal is performance, that is, a system that quickly returns the location data while injecting a minimum of overhead traffic into the network.

Figure 1 shows the basic mapping service provided by LDS. In the figure, the client on the left has a URL, it gives it to the LDS system which returns a list of object servers that contain the object. Instead of the whole URL, the client can as well give only a prefix of the URL to the LDS black box which then returns a list of object servers that contain URLs matching the prefix. In the simplest case, the prefix is only the hostname of the URL; in this case, the LDS service is identical to the Domain Name System, operational in the Internet for over 20 years.

Given the similarities in the service provided by both DNS and LDS, we have decided to base the design of LDS on DNS. In fact, LDS can be implemented by making minor extensions to BIND name servers and to the DNS protocol. Recall that DNS provides a mapping of hostnames to IP addresses. (DNS can return more than one IP address for a hostname for mirrored sites.) DNS uses a hierarchy of name servers to implement a distributed database of resource records, whereby a resource record contains a mapping.

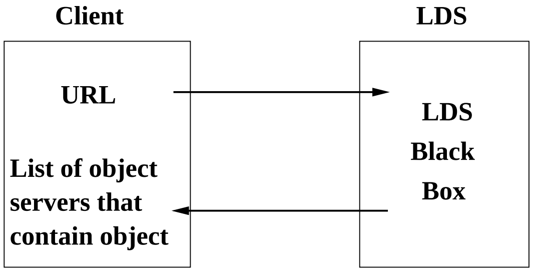

Figure 1: LDS service

We now present an overview of the LDS design; we provide more details in the subsequent subsections. As does DNS, LDS uses a hierarchy of servers to implement a distributed database of resource records. We refer to the LDS servers as **location servers**. Each LDS resource record maps a URL to an object server, with associated freshness information. Each URL has one (or more) authoritative location servers (To convey the main idea, we initially assume that each URL has exactly one authoritative server.) The authoritative location server contains a list of resource records for the URL, that is, a list of object servers that contain the URL (along with the freshness information). Other location servers may contain cached copies of the list of resource records. In our basic design, hosts query location servers in a manner that is fully analogous to the DNS protocol. The sequence of query and reply events is illustrated in Figure 2.

In Figure 2, C is the querying host (e.g., a browser), O_0 is the origin server for the queried object, machines O_1–O_4 are other object servers (e.g., caches), and machines L_1–L_4 are location servers.

The querying host sends a location query to its local location server (L_1). If L_1 does not have the location information cached, it

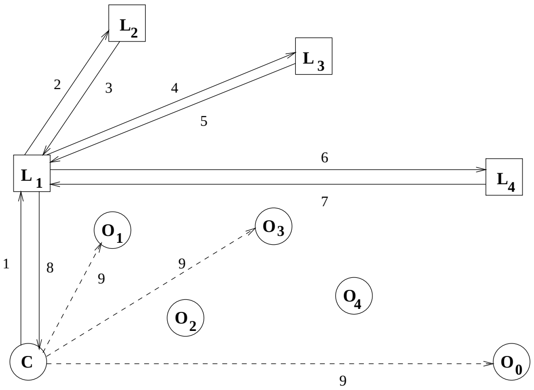

Figure 2: Sequence of Location Queries and Replies

sends a query to a **root location server** (L_2). If L_2 does not have the location information cached, it returns the address of a location server responsible for the domain of the origin server in the query (L_3). L_1 sends a new location query to L_3 and if L_3 does not have the location information, it returns the address of the authoritative location server (L_4). Finally, L_1 queries the authoritative location server and receives the location information. Location server L_1 then sends the information to the querying host and also caches the information. (In this example we assumed that all queries between LDS servers are iterative; recursive and combinations of recursive and iterative can also be used. Like DNS, LDS is not based on either query type being used. We also assumed that there is one intermediate location server between the root and authoritative servers; in practice there can more or less.)

Suppose that the reply indicated that object servers O_0, O_1, and O_3 contain copies of the URL. The querying host then chooses to retrieve the URL from one of these object servers. Ideally, the host chooses the object server that will deliver the object the fastest. (We will discuss how this choice might be made in Section 5.)

Given the similarities between our basic design of the LDS and the DNS, we now address whether the LDS can implemented in the existing DNS, or within a slightly extended DNS. A DNS implementation could lead to rapid deployment of the LDS.

2.1 Resource Records, Query Messages and Reply Messages

For location requests a new DNS resource record type is needed. It is similar to the standard A-type resource record, which maps a hostname to an IP address. We now describe this new DNS resource record type.

Each resource record of this type must contain an object identifier. In order to be compatible with standard DNS queries, object identifiers must be encoded in a standardized way. Using this encoding, the query appears like a normal DNS A-query and can be resolved without changing any parsing routines in DNS servers. Figure 3 shows how the URL `http://www.eurecom.fr/~bob/index.html` would be encoded.

4	h	t	m	l	5	i	n	d	e	x	4	~	b	o	b	4	h	t
t	p	3	w	w	w	7	e	u	r	e	c	o	m	2	f	r	0	

Figure 3: Representation of a URL

This encoding allows for efficient compression of identifiers sharing a common part, just as when querying the addresses of all of the servers in a given domain. Since the identifier can be viewed as a generalized hostname, all the normal facilities of DNS can be used.

We mention that there is a 255 character limit on hostnames in DNS queries which implies that URLs longer than 255 characters will have to be truncated. Also, the length of a single label (e.g., one component of hostname or URL path) is limited to 63 characters. We do not expect these limits to be a problem. We also mention that URLs are case-sensitive but DNS does not guarantee case-sensitive treatment. Nonetheless, we do not expect this to be a major issue since most URLs cannot be confused with other URLs even if case is not preserved.

We studied the access log from one of the NLANR top-level caches and out of the

1.1 million URLs in the file, only 122 exceeded these limits. Even if these 122 URLs were truncated to conform to DNS limitations, we did not observe any collisions between two different URLs. Neither did we find any URLs where case-insensitivity would have presented any problems. Therefore, we do not propose any methods for handling such URLs beyond simple truncation.

The reply will include several resource records, one for each object server holding a copy of the URL. Each of these resource records has a time-to-live (TTL) field which specifies how long that resource record can be cached at a DNS server. This TTL-information can either be specified by the authoritative location server or, for a better estimate, by the object server. If the location server sets this TTL-field, it will be the same for all resource records from that server. If, on the other hand, this field is set by the object server, it allows the object server to communicate information on how long the object is likely to be available at that server.

The actual data section of the resource record (RDATA) contains the IP-address of the object server and some freshness information about the object. This information could be for example the last modification date of the object, which provides an easy way of distinguishing between possible stale copies of the object. In this case, the authoritative location server would have to periodically query the origin server to get an up-to-date modification date for the original object.

Another possible piece of information that could be included in the resource record, instead of the last modification date, is the "age" of the object, as defined in HTTP/1.1. An object server could determine this locally, and a small age would refer to a relatively fresh copy. This method would not, however, offer the same guarantees on object freshness as would the last modification date.

Yet another possibility is to include simple TTL-information, but this would be redundant, since the resource record by definition includes a TTL-field. A TTL-estimate is useful when the querying host decides which object server to use. Object servers with short expected TTLs can be discarded from the decision making process.

Because all location servers are allowed to cache LDS replies, some location servers may have stale location information in their caches. When the status of an object changes at an object server, the object server notifies the authoritative location server. This update is not, however, reflected on the cached copies of the location information. If stale, cached location information is delivered to a querying host, the querying host may decide to forward the request to a Web cache that has evicted the object.

One solution is to use the `only-if-cached` directive of HTTP/1.1 when requesting objects from Web caches. If the distant cache has the object cached, it will return the object from the cache; otherwise it will return an error to the querying host indicating that the object is no longer cached. The querying host will then choose another object server from the list. We are currently looking into ways of reducing the amount of stale location information in LDS.

2.2 Updating the Location Servers

The object servers need to keep the information at the authoritative location server up-to-date. For this we need a new protocol that is used to exchange information about new copies of objects. For example, when a cache caches a new object, it must send a message to the authoritative location server responsible for this object, indicating that the object has been cached. This message should also include the information which will be present in the resource record (e.g., last modification date) as well as a TTL-estimate. Similarly, when the cache removes an object, it must send a message to the location server indicating that the object is no longer cached.

Since the location data is managed over DNS, the dynamic update facility of DNS [14] can be used to dynamically update the location information. With this method, it is possible for a host to add or delete resource records atomically in an authoritative server.

The host can specify update prerequisites if desired.

Using the dynamic update facility of DNS, an object server sends an update message every time the status of the object changes. For a Web cache this change of status could be the caching of a new object, a removal of an object or caching a new version of the object after the object has been modified at the origin server. In order to reduce the amount of update traffic, object servers can send their reports in batches. This is especially beneficial for Web caches that can send the information about an HTML page and all inlined images in one message, thereby reducing the overhead considerably.

2.3 Implementation

Implementing the LDS is relatively straight-forward and can be done incrementally. Since the system is an extension to DNS, no new servers need to be created and introduced.

For a DNS server to be LDS-capable, it needs to be able to interpret a new query type code and the associated resource records. DNS already handles numerous different query and resource record types, so adding one more is simple. Furthermore, the authoritative server has to be able to maintain the location information database and construct resource records from this database to include in replies to queries. Although this URL database contains more data than a normal DNS database, it can be maintained in a similar way. For a Web cache to be LDS-capable, it simply has to be able to send the update messages to the authoritative server.

The architecture allows for incremental deployment, since if a content provider does not operate a location server, no LDS resource records will exist. If no LDS resource records are available, the querying host would forward all requests using a static policy, for example, forwarding them directly to the origin server or a parent in a cache hierarchy. If LDS records exist, then LDS-enabled clients could use them to provide a better service for users. Non-LDS clients would again use the normal, static methods for finding objects. In the early phases of deployment, the traditional hierarchy should be kept as a fallback for clients and servers that do not implement the new protocols.

3 Network Web Caching

In this section we propose a new cooperative caching scheme that exploits the LDS. We assume that browsers forward their requests through a proxy cache in addition to the browser cache. This method is currently widely used by ISPs and other institutions, such as companies and universities, to offer a better service to their clients as well as to reduce the out-going traffic. Also, if a firewall is used, then all requests must go through a proxy at the firewall in any case, so adding a cache there does not present a significant overhead.

We now describe our cooperative caching scheme. A browser first sends an HTTP request for an object to its proxy cache. If the proxy cache does not have the object, the proxy cache invokes LDS to obtain a list of all the object servers (i.e., origin server and some other caches in the network) that contain the object. The proxy cache then chooses the “best” object server from the list and forwards the HTTP request to this object server (see Section 5). The proxy receives the object from the best object server and forwards the object to the browser.

Our caching scheme has several appealing features. First, at most two servers are visited to retrieve an object. Standard hierarchical caching schemes (such as NLANR [8] and Renater [9]) can cause requests to pass through a large number of object servers before a copy of the object is found, which can severely increase object retrieval time [13]. Second, the scheme allows the proxy server to choose from all the object servers that contain the object. Let us look at a couple of scenarios:

- LDS returns a list of object servers, with one cache server in a neighboring ISP of the ISP that contains the proxy cache, and all the other object servers on more distant ISPs. In this case, the proxy server would choose the cache in the neighboring ISP.

- LDS returns a list of object servers, with one cache server on the same continent as the proxy server, and all the other object servers on the other side of transoceanic links. In this case, the proxy server chooses the cache in its continent.
- LDS returns a list of object servers, with all the object servers far from the proxy and near or in the origin server's ISP. In this case, the proxy may still prefer to choose one of the caches over the origin server: The origin server may run on a slow machine or have a slow network connection.

It is important to note that traditional hierarchical caching does not permit the last option. With hierarchical caching, if there is a miss at the root cache, the root cache forwards the request directly to the origin server. Thus our scheme enables the proxy to select from *all* caches that are on the "route" between the proxy and the origin server.

Our caching scheme does have an unusual peculiarity. In the scheme just described, the proxy server will always retrieve an object either from the origin server or from some other proxy server. Because all the proxy servers are near the bottom of the Internet hierarchy (in institutional ISPs or local residential ISPs), with the exception of the origin server, all copies of an object are stored, essentially, in the leaves of the Internet. There are, however, several compelling reasons to also store copies of objects higher up in the Internet hierarchy, in particular in regional and national ISP caches. First, the requesting proxy may be able to retrieve an object more quickly if it is available from a common parent (or grandparent) of some other proxy cache that has the object. Second, hierarchical caching provides an asynchronous multicast infrastructure, which can significantly reduce bandwidth usage in the Internet [10].

Given that cache servers are present at regional and national levels, and peering agreements exist which organize the proxy, regional and national caches into hierarchies, we now modify our caching scheme to exploit the higher-level caches.

3.1 Populating Caches

In order to exploit the higher-level caches, we must make sure that the higher level caches obtain copies of the objects cached at lower levels. We propose that lower level caches push objects to their higher level parents. This way the objects are easily available for siblings under the same parent, and requests from distant hosts can be satisfied at a higher level in the network.

The details of our pushing strategy are as follows. A low level cache periodically sends a list of new objects (with last modification dates) to its parent. The parent decides which of the objects in the list to cache and retrieves these from the child cache. This is done at every level in the hierarchy and, in the end, the caches are filled up as they would have been using traditional hierarchical caching. Caches at the lowest level should send information about every object, but at higher levels a cache might use some locally defined policy (e.g., objects that are cached in multiple child caches) to decide which objects to report to its parent.

3.2 Network Traffic

Since LDS increases the number of messages sent over the network, it may overload some links or servers and in fact degrade performance. We are currently performing an analysis of how much LDS increases current network traffic. The following are the key factors:

1. **LDS queries and replies:** Although these are normal DNS messages, an LDS query is sent every time a proxy needs to retrieve a URL that it is doesn't have cached. Caching LDS records at the location servers will reduce some of this traffic; however, caching is not expected

to have the same impact that it has with ordinary DNS, since a URL reference is more specific than a hostname reference.

2. **Update traffic:** When the status of an object changes in a cache, the cache must send an update message to the object's authoritative location server. For popular, widely cached objects, this may result in too much traffic directed at the authoritative server.

In order to reduce the amount of LDS lookup traffic in the Internet, we propose a variation to our basic LDS scheme. In this variation, once we get the location information for an HTML-page we assume that all the inlined objects on the page are also available from the same set of object servers. One advantage of this scheme is that it heavily cuts down on the amount of LDS traffic compared to that basic scheme since we now only send one LDS query for each HTML-page instead of each URL. The disadvantage of the variation is that there are no guarantees that the inlined objects are available from the same object servers. Origin servers and replicated servers should have the objects but a cache may have already purged some or all of them. If the object server does not have the requested object, we will do an LDS query for that object to obtain up-to-date location information.

In Section 4 we will show how the amount of LDS traffic can be significantly reduced when the LDS system is restricted to replicated servers.

To address the issue of update traffic, we propose that any cache that has a parent refrain from sending update information. Instead, the update information is forwarded to the parent during the pushing phase. The highest parent that does not have a copy of the object sends to the authoritative server an update message that contains update information for itself and its updated children. The benefit is that there are significantly less higher level caches than low level caches, thus much less update traffic.

3.3 Practical Considerations

In the above scheme, the highest parent in the hierarchy sends update messages for all its children which are thus all included in the database at the location server. As a result, institutional caches would also be included in the list returned by the location server, which means that they could receive requests for cached objects from hosts outside their own network. It is desirable to keep the institutional caches off the list of locations for two reasons. First, institutions likely do not want outside hosts using their bandwidth to retrieve objects. Second, objects at institutional caches are likely to be cached in the parent caches and outside hosts can retrieve objects faster from the parent cache.

Cache digests [11] are used in traditional hierarchies to represent cache contents in a compressed form. A cache fetches the digests of its neighbor caches and when a request cannot be satisfied locally, the cache checks the digests of the neighbor caches to see whether any of them have the object cached. If so, the object is retrieved from that cache; otherwise the request is forwarded to the parent cache. In LDS-based caching, having the digest of the parent cache is useful because this way we avoid the LDS-lookup for objects that are cached at the parent and would in any case be retrieved from there.

4 Replicated Servers

The LDS scheme is not limited to caching. It can also be used to disseminate information about replicated or mirrored objects. When an object is replicated at another server, this information is added into the location database at the authoritative location server. When a client requests this replicated object, it does an LDS lookup and gets a list of all servers holding a copy of the object. Because the information is not dynamically replicated as in caching, but rather placed at well-chosen locations, there is rarely need to notify the location servers of new copies. Likewise, there is no need to push objects into other servers since all replication decisions are made offline.

With LDS, a user addresses a replicated object by the object's unique URL, LDS returns the list of servers that have the object. The browser then transparently chooses one of these (the best in some sense). The user would not have to know about the actual physical location of the object.

4.1 Reducing LDS Query Traffic

The scheme just described for replicated servers still requires that clients send an LDS lookup for each URL. In order to reduce the number of LDS messages we propose the following method of replicating servers and storing information about the replicated URLs. In this scheme, the LDS no longer keeps track of cached objects. It only tracks replicated and partially replicated servers. We require that replicated servers replicate either whole sites or complete sub-trees of servers. An example sub-tree of an origin server is all the objects on the origin server below a given URL prefix, e.g., all objects under `http://cnn.com/sports/`.

Suppose that a client wants to retrieve a URL. Normally, a client would send a DNS lookup to obtain the IP address of the origin server and retrieve the object from there. Using LDS the client proceeds as follows. First, the client sends an LDS query for the hostname in the URL. The LDS system returns a list of all servers that mirror either the whole site or a complete sub-tree from the site. We need to extend the resource record defined in Section 2.1 to include a prefix indicating the sub-tree that is mirrored by that particular server. The client then chooses the "best" server among those servers that mirror the desired sub-tree and forwards the actual HTTP-request to this server. A server may mirror multiple sub-trees from a single origin server; in this case LDS would return one resource record for each sub-tree.

Compared to the caching scheme discussed in Sections 2 and 3, this scheme has several advantages. First, and most important, in this mirroring scheme the client sends only one LDS lookup for each server; in the caching scheme, the client must send an LDS lookup for *each* URL. The caching scheme drastically increases DNS traffic in the network while the mirroring scheme keeps DNS traffic at its current level. (Recall that a client would in any case have to do a DNS lookup to get the IP address of the origin server.) Second, objects at mirrored servers tend to stay at the mirrored servers for long periods of time while objects in caches are cached and purged dynamically. Hence, LDS information for a mirrored server can be cached longer at location servers which greatly reduces lookup traffic. When the authoritative LDS server sends information about a mirrored site, it should first send out information about mirrors that mirror the most of the site. This is because DNS replies are by default sent over UDP and the message size is severely limited (512 bytes). By sending information about mirrors that mirror large subtrees, we maximize the likelihood that the client has the necessary information in the reply.

Compared to the standard DNS-way of accessing objects, our mirroring scheme does not increase the number of DNS messages in the network. The reply messages are slightly larger, however, since we need to indicate the prefix in the resource record. We do not expect this increase in size to be significant since URL prefixes can be efficiently encoded using the encoding specified in Section 2.1 and DNS resource record pointers.

5 Routing Decision

When the cache has received a reply to its LDS-lookup containing several object servers (caches and origin servers), the next question is: "Which one of the possibilities to choose?" Determining the best alternative is an important topic of our ongoing research.

Possible solutions include:

- All querying hosts measure connection qualities to object servers and keep a list of servers with known good performance.

- Pass QoS information along in LDS updates and replies.
- Use explicit routing information from BGP.

We will now provide some details on how these methods could be used.

In the first option, all querying hosts (e.g. low level Web caches) measure the time it takes to fetch objects from other servers (Web servers or other caches). This download time gives a crude estimate of the actual bandwidth between the two hosts. The querying host keeps a list of servers with which it has had good connections and prefers those servers to others when both types of servers are present in the LDS reply. Of course, the actual network conditions change all the time, but the results presented in [7] on performance characteristics of mirror servers indicate, that from a large set of mirror servers, only a small number need to be considered as candidates for download. Although the study in [7] uses a fixed number of mirror sites, we believe that the results can be applied to situations where the number of servers changes dynamically.

The second option is to pass some QoS information in the LDS replies along with the age information about the object. For example, a Web server can measure the connection times of incoming connections, estimate the connection bandwidth and take an average of the estimated bandwidths over all connections. This average bandwidth can be seen as the bandwidth that a random client somewhere in the network could expect to get when requesting objects from this server. Likewise, caches can measure the bandwidths of all out-going connections and calculate the average bandwidth.

The third option uses explicit routing information obtained over BGP by talking to local routers. This information gives the host a topological map of the network which can be used to find out how far each of the servers in the LDS reply are. Unfortunately, routing information gives only reachability, not quality, so some quality measures, as in the first two options, would be necessary. Grimm et al. [4] have studied using routing information in request routing and have decided against using BGP information because of configuration and security issues, amount of data and difficulty of obtaining it. Their approach is based on using whois-services.

Regardless of the approach chosen, any querying host can implement any local policies necessary when deciding where to forward a request. For example, a cache operator may want to forward requests to other caches based on the domain of the origin server.

6 Related Research

In [3] Gadde et al. compare traditional hierarchical caching with an architecture where a single centralized server holds information about where each document is cached. When a low level cache wants to find out where an object is cached, it sends a message to this central mapping server which responds by either redirecting the request to a peer server or by forwarding it to the origin server. This scheme results in all of the objects being cached at institutional caches. The authors also perform simulations using a small number of caches and find out that the performance of the centralized solution is on average better than that of the traditional hierarchy. The number of caches in the simulations is low, only 8 and 32 cache configurations are simulated. The authors express their doubts about the scalability of their solution, but present some ideas for replicating and distributing the location information.

In our approach, the querying host gets a list of *all servers* holding a copy of the object instead of being redirected to one of them by a mapping server. This puts the querying host in control over where the request will be forwarded. This decision might greatly depend on local conditions and policies unknown to a central server. Another difference in our approach is that the location information is distributed over DNS which is ubiquitous and provides satisfactory service.

Tewari et al. [13] present an architecture where data is cached near the browser and location hints are passed through a metadata hierarchy. Their architecture arranges the caches in virtual hierarchies, one for each object, for distributing the metadata information. In their architecture, when a cache caches a copy of an object, it sends out a location hint indicating the URL and its address. This hint is propagated in the virtual metadata hierarchy and could eventually reach all caches in the system. Caches keep track of all the hints they have received and use them to forward requests. If a cache receives a request for an object which is not cached locally but the hints indicate another cache holding the object, the request is forwarded to this cache. The virtual metadata hierarchies are constructed using IP-addresses and URLs in a way which tries to minimize the distances between parents and children. Also, the hierarchy guarantees that a cache can receive only one hint for any object. This hint is likely to be from the nearest cache holding the object, but this is not guaranteed.

One difference between their and our approaches is the way caches are placed. In their architecture only institutional caches hold objects and every request forwarded using a hint would be forwarded to another institutional cache. This behavior might be unacceptable to some people who do not want more external traffic on their networks. In our approach, objects are pushed to higher levels and this eliminates the need for going to other institutional caches. Second important difference is that using LDS for locating objects, a querying host gets a list of all possible locations. It can then choose from the list according to a local policy or by adapting to current network conditions. In [13] a cache receives only one location hint. Since the virtual metadata trees are based on IP-addresses and URLs, there are no guarantees that this hint indicates a good server. We are currently performing a comparison of the two schemes.

Amir et al. [1] study three different methods for redirecting requests to mirrored sites – HTTP redirection, DNS round trip time measurements, and shared IP addresses. In their DNS-based solution they use DNS round trip time measurements to determine the best replicated server for a client. They place the replicated servers near an authoritative DNS server which gives the address of the replicated server when queried for the origin server. This results in clients being eventually redirected to their closest replicated server. There are two major differences between their approach and our replicated server approach (Section 4.1). First, in their system replicated servers must always replicate the entire site while in our system a server may replicate only sub-trees of the original site. Second, in their system the replicated servers must be placed in close proximity of the authoritative DNS server. Our architecture places no constraints on the placement of the servers.

7 Future Work

We will continue our work on LDS by performing quantitative comparisons of the different architectures presented in this paper. In particular we will concentrate on evaluating the amount of new traffic in the network caused by the lookup and update messages. We will also compare the traditional caching hierarchy with our different LDS architectures to find out how much object access latency can be reduced through LDS. We will also closely study the request routing issue in order to find out how much information is needed to make good routing decisions.

8 Conclusion

In this paper we have presented the Location Data System, a new network application that can be used to locate where copies of objects are stored on the Web. LDS provides a service similar to DNS and can be implemented as an extension to DNS and deployed incrementally. We have presented two applications of LDS, locating objects in mirrored servers and locating objects in Web caches. We have also discussed how clients can decide the best server to forward requests to based on information on network conditions and topology collected from the dynamically from the network.

Acknowledgments

We would like to thank Neilze Dorta from INRIA for the discussions we had on accessing mirrored sites and her input on the subject. The logfiles used in Section 2.1 were provided by National Science Foundation (grants NCR-9616602 and NCR-9521745), and the National Laboratory for Applied Network Research.

References

[1] Y. Amir, A. Peterson, and D. Shaw, "Seamlessly Selecting the Best Copy from Internet-Wide Replicated Web Servers", in *Proc. 12th International Symposium on Distributed Computing (DISC'98)*, Andros, Greece, September 1998.

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1", RFC 2068, January 1997.

[3] S. Gadde, J. Chase, and M. Rabinovich, "Directory Structures for Scalable Internet Caches", Technical Report CS-1997-18, Department of Computer Science, Duke University, 1997.

[4] C. Grimm, J.-S. Vöckler, H. Pralle, "Request Routing in Cache Meshes", in *Proc. 3rd Web Caching Workshop*, Manchester, UK, June 1998.

[5] P. Mockapetris, "Domain Names – Concepts and Facilities", RFC 1034, November 1987.

[6] P. Mockapetris, "Domain Names – Implementation and Specification", RFC 1035, November 1987.

[7] A. Myers, P. Dinda, and H. Zhang, "Performance Characteristics of Mirror Servers on the Internet", Technical Report CMU-CS-98-157, School of Computer Science, Carnegie Mellon University, 1997.

[8] A Distributed Testbed for National Information Provisioning, <URL:http://ircache.nlanr.net>.

[9] Réseau National de télécommuncations pour la Technologie, l'Enseignement et la recherche, <URL:http://www.renater.fr>.

[10] P. Rodriguez, K. W. Ross, E. W. Biersack, "Improving the WWW: Caching or Multicast?", in *Proc. 3rd Web Caching Workshop*, Manchester, UK, June 1998.

[11] A. Rousskov, D. Wessels, "Cache Digests", in *Proc. 3rd Web Caching Workshop*, Manchester, UK, June 1998.

[12] Squid Internet Object Cache, <URL:http://squid.nlanr.net>.

[13] R. Tewari, M. Dahlin, H. M. Vin, and J. S. Kay, "Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet", Technical Report TR98-04, Department of Computer Sciences, University of Texas at Austin.

[14] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, April 1997.

[15] D. Wessels, K. Claffy, "Internet Cache Protocol (ICP), version 2", RFC 2186, September 1997.