A Replicated Architecture for the Domain Name System

Jussi Kangasharju, Keith W. Ross
Institut Eurécom
2229, route des Crêtes
B.P. 193
F-06904 Sophia Antipolis, France
{kangasha,ross}@eurecom.fr

Abstract—**We propose a new design for the Domain Name System (DNS) that takes advantage of recent advances in disk storage and multicast distribution technology. In essence, our design consists of geographically distributed servers, called replicated servers, each of which having a complete and up-to-date copy of the entire DNS database. To keep the replicated servers up-to-date, they distribute new resource records over a satellite channel or over terrestrial multicast. The design allows Web sites to dynamically wander and replicate themselves without having to change their URLs. The design can also significantly improve the Web surfing experience since it significantly reduces the DNS lookup delay.**

I. Introduction

We propose a new design for DNS that takes advantage of recent advances in disk storage and multicast distribution technology. This design can be implemented incrementally, allowing for a graceful evolution from the current DNS to a new system. Our design does not change the syntax or semantics of the DNS messages; it only affects the way the DNS database is managed. Our proposed design has two principal features:

- First and foremost, it is highly responsive to changes in DNS information. Being able to rapidly change DNS information and propagate the changes is useful in solving the hot-spot problem on the Web. Often, a single web server becomes suddenly popular and receives many more requests than it can handle. In our design, without changing URLs, the web server could replicate its contents on another web server and inform the DNS system of the alternative server. Because the new web server is immediately available in the DNS to every client, many clients would likely choose it over the old server (by using, for example, DNS rotation), thus significantly reducing the load on the server and improving the quality of service for everyone. Modern DNS does not allow this because nameservers are allowed to cache resource records.
- It can significantly reduce the DNS lookup time. The system eliminates the need to query distant authoritative servers. Because DNS round-trip time is often a significant fraction of the delay when accessing a Web page, our design can improve the Web-surfing experience.

Our design is inspired by two recent technological advances. First, disk storage has become abundant and cheap in recent years, and the trend is expected to continue. As we shall argue in the body of this paper, the entire DNS database can be stored in disk of a household PC. Second, emerging terrestrial multicast and satellite broadcast systems can efficiently distribute DNS information. We note here that these two advances abundant disk space and efficient broadcast distribution are currently being exploited by Web cache technology in order to bring the "Web to the edge of the network" [1].

In essence, our design consists of geographically distributed servers, called replicated servers, each of which having a complete an up-to-date copy of the entire DNS database. To keep the replicated servers up-to-date, they distribute new resource records over the satellite channel (or over terrestrial multicast).

In this paper we provide (i) a detailed description of the design, (ii) a detailed feasibility analysis for network traffic, replicated server traffic and disk storage requirements, and (iii) a plan for migrating the existing DNS system to our replicated design. We also discuss the security and fault-tolerance issues for this new design. This paper is organized as follows. In Section II we provide a short overview of the existing DNS system. In Section III we describe the replicated DNS architecture. In this section we also analyze storage and network bandwidth requirements for the new architecture. In Section IV we model the tradeoff between the staleness of the DNS information and the traffic load at the replicated servers. In Section V we use empirical Internet data to study how much our architecture will reduce the time it takes to resolve a hostname. In Section VI we look at how the existing DNS system can migrate to our proposed architecture in a graceful manner. In Section VII we address security and fault tolerance issues for the new architecture. Finally, Section VIII concludes the paper.

II. Overview of DNS

In this section we provide a brief overview of DNS and introduce some terminology that we shall use throughout the paper. The principle task of the DNS is to provide a mapping from the human readable domain names to numerical IP-addresses used to identify hosts on the Internet [2, 3]. It is implemented in a distributed database consisting of a hierarchy of nameservers. The name space is divided into zones and each zone has two or more *authoritative nameservers* that are responsible for keeping information about that zone up-to-date. One of these authoritative servers is the *primary nameserver*, which holds the master file containing all the resource records for that zone. When new hosts are added to a zone, the administrator must edit this file manually to make the new hosts public. The other authoritative servers (*secondary nameservers*) periodically fetch the contents of the master file in order to keep their records up-to-date. These zone transfers are done using the special zone transfer query

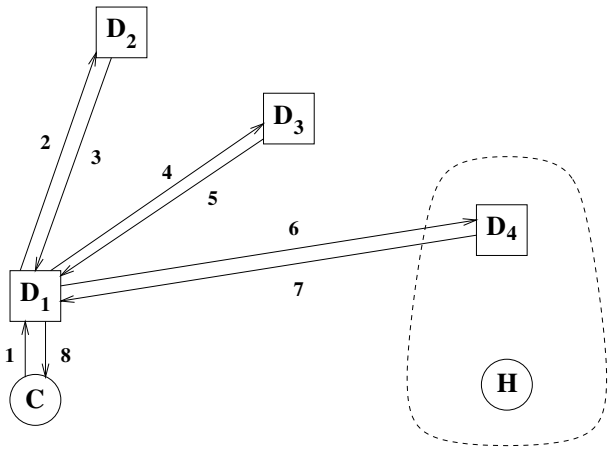


Fig. 1. A DNS query. C is the client, H is the host that the client is trying to resolve, D_1 is the local nameserver, D_2 is a root name server, D_3 is an intermediate nameserver, and D_4 is an authoritative nameserver for the queried host H.

type in DNS (AXFR query type [3]).

When a client needs to obtain an IP-address for a hostname, the query proceeds as shown in Fig. 1. In Fig. 1, C is the client, H is the host that the client is trying to resolve, and D_1–D_4 are DNS servers.

First, the client sends the query to its *local nameserver* D_1. Typically this local nameserver acts as the primary nameserver for the zone where the client resides and has all the DNS information for that zone as well as cached copies of DNS information from other zones that the local clients have recently queried. Assuming that this server does not have a cached copy of the information, it queries one of the *root nameservers*, D_2 (currently there are 13 root nameservers in the world [4]) which returns a referral to a nameserver responsible for the top-level domain of the hostname. The local nameserver then queries this server D_3 and gets a referral to an authoritative server D_4 for the domain in which the host is located. Finally, the local nameserver queries the authoritative nameserver and gets the reply with the IP-address of the host. When the local nameserver receives the reply it sends it to the client and caches a copy. If another client now wants the address of the same host, the local nameserver can immediately return the cached copy, thus avoiding the need to query distant nameservers. (In this example we assumed that all the queries from D_1 are iterative; recursive and combinations of recursive and iterative queries are possible. We also assumed that there is one intermediate nameserver between the root and the authoritative nameserver; in reality there can be more or less.)

III. Replicated DNS Architecture

Our architecture makes use of *replicated nameservers*, with each replicated nameserver storing the *entire* DNS database. Before describing the architecture in detail, we perform a simple feasibility analysis for storing the entire DNS database on a nameserver.

A. Storing the Entire Database on a Server

In 1999 standard PCs are sold with approximately 10 gigabytes of disk storage. If storage capacity continues to grow at current rates, standard PCs will be sold with 20-30 gigabytes of storage in 2000.

Now let us estimate the storage requirements for a replicated nameserver. Since each replicated nameserver replicates all of the DNS information on the Internet, it must have an entry for every host in the DNS. It is possible that a hostname maps to multiple IP-addresses, but this is not very common, so, for simplicity, our analysis assumes that we need one resource record for each existing hostname.

The Internet Domain Survey [5] reports that in July 1999 there were slightly over 56 million hostnames registered in DNS. Using the data from their January 1997 survey we can calculate the average length of a hostname in the DNS which is 20.07 characters. We will conservatively assume that we need 40 bytes of storage per hostname; this includes the actual hostname, IP-address, TTL-information, and possibly other information. In 1 gigabyte of storage we can store entries for 25 million hostnames, thus we would need a bit over 2 GB of storage to store the IP-addresses of all the hostnames in the DNS in 1999.

The above estimate only accounts for IP-addresses and hostnames, i.e., only for DNS A-type resource records. Note that NS-type resource records are not necessary, since the replicated nameservers contain information about all hosts. We will, however, need to account for other types of resource records, most notably PTR, CNAME, MX, and SOA records. In order to get an upper bound estimate, we will assume that there is one resource record of each of these types per host. In reality, only PTR-records exists for each host. Most hosts on the Internet do not have CNAME records, MX records exist usually on a per domain basis, and SOA records exist only for authoritative nameservers. According to our conservative assumptions, we would need 10 GB of storage to store all the information in the DNS system. In reality, this would probably be much less, on the order of a few gigabytes, which is easily stored on the disk of an inexpensive PC.

B. Interaction between Authoritative and Replicated Nameservers

Our proposed architecture consists of the current authoritative primary nameservers and a number of replicated nameservers (something between 10 and 10,000) distributed all over the world. Ideally there should be at least one replicated nameserver per region (e.g., one per country) so that clients can receive replies to queries fast. Replicated nameservers could also be present at the local ISPs and at corporate and university networks.

Our replicated architecture maintains the "local administration, global availability" philosophy behind modern DNS [6]. In our replicated architecture the primary nameservers still keep track of their own zones in the normal way, i.e., they have the master file for the zone and administrators make changes to this file in the usual manner. However, our architecture replaces the secondary authoritative servers with replicated nameservers. A replicated nameserver is responsible for all the primary name-

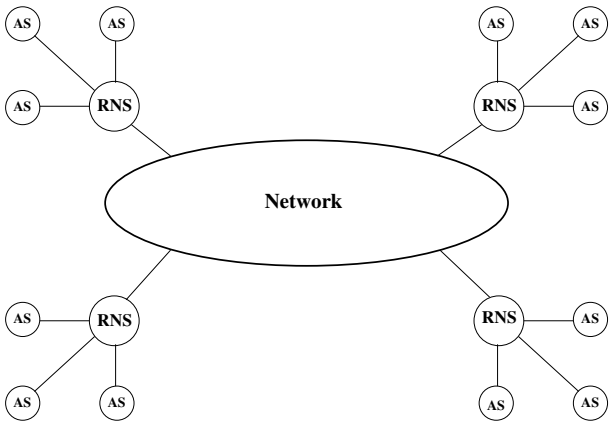


Fig. 2. Connections between authoritative nameservers (AS) and replicated nameservers (RNS)

servers in its region and periodically fetches the zone information from the primary nameservers. Naturally, a replicated nameserver can also replace a primary nameserver, if needed. (For clarity of presentation, we assume in the following that each primary nameserver is associated with a single replicated nameserver; in reality the primary nameserver could interact with any number of replicated nameservers.) In the new architecture, the primary nameserver of a zone also pushes the information to its closest replicated server when the information in a zone changes. This combination of pushing updates and periodically fetching the whole zone guarantees fast updates and guarantees that the information is refreshed periodically in case some of the update messages were lost in the network. (Recall that by default DNS works over UDP which offers no guarantees.) It also provides a mechanism for replicated servers to quickly recover from a crash. For the fetching, the replicated server uses the normal DNS zone transfer query (AXFR query type; zone transfers are done over TCP); for the pushing the primary servers could use the DNS dynamic update method [7].

C. Interaction among Replicated Nameservers

The replicated nameservers contain an up-to-date database of the entire DNS database. As shown in Fig. 2, this is accomplished by having the replicated nameservers communicate with each other in order to share the updates they have received from the primary nameservers that they parent. In this section, we outline two schemes for distributing the update information: multicast and satellite.

IP Multicast

In this scheme the replicated servers all belong to a multicast group which has a single multicast address. When a replicated server receives new information from its child primary nameservers, it sends the information into the multicast address. Eventually all replicated servers will receive the update. The advantage of this solution is that it does not require any special hardware, but it needs all the replicated nameservers to have access to multicast, e.g., to be connected to MBone [8]. This solution works best if the number of replicated servers is not too large and they can be equipped with multicast; also the amount of update traffic may become an issue since the update traffic has to share the bandwidth to the replicated nameserver with the normal query traffic.

Satellites

Another possibility for transmitting information between the replicated servers is to bypass the Internet and send the information over a satellite channel. Because of the broadcast nature of satellite and our need to broadcast the information to all replicated servers, a satellite channel is a very attractive alternative for distributing the information. The main advantages of using a satellite channel for distributing the information are the following: (1) The update traffic does not have to travel long distances over the terrestrial Internet; (2) information is immediately available at all replicated nameservers.

We propose two possible schemes for handling the use of the satellite channel. In the first scheme one of the replicated nameservers is designated as the central server. All the other replicated nameservers collect the updates from the primary nameservers in their own regions, aggregate the updates, and send them over the Internet via unicast to this central server. The central server receives updates from all other replicated servers and broadcasts these updates over a satellite link. This scheme requires that all replicated nameservers be equipped with a satellite receiver. This method has the advantage that there will be no collisions on the satellite channel since only one station is transmitting. This architecture is similar to the SkyCache architecture [1] where Web caches send access reports to a central master server which streams the most popular Web pages over the satellite link.

In the second scheme we have no central replicated server. Instead each replicated nameserver has a satellite transmitter and broadcasts over the satellite channel the updates it has received to all other replicated nameservers. In this scheme we have to use some method of resolving the conflicts that might occur when two different servers want to broadcast information at the same time. According to [9] the best medium access control protocol for bursty traffic is either random access or a simple reservation based protocol, depending whether the messages are short or long, respectively. We expect DNS update traffic to be bursty and the messages to be relatively short, so a simple medium access protocol, such as Aloha or Slotted Aloha, is likely to provide sufficiently good performance.

The choice of the update scheme depends greatly on which kinds of satellites, GEO or LEO, we are using. In order to get global coverage with geostationary satellites, we need to use several satellites in a coordinated fashion. Therefore it is simpler to use the second scheme with GEO satellites. In a LEO constellation, the centralized scheme works as described above since the constellation provides global coverage; the second scheme in a LEO constellation would most likely be analogous to terrestrial IP multicast.

One downside of using satellites, compared with using IP multicast, is that we need to install satellite receivers and transmitters and obtain bandwidth on the satellite channel. The first scheme only requires a satellite receiver on the replicated nameservers; this can be a normal satellite dish, such as the ones used in the DirecPC-service [10]. Such dishes are relatively cheap, around $300–$400, and provide several hundreds of kilobits/second of bandwidth (DirecPC operates at 400 kbps/s). The

TABLE I
NUMBER OF UPDATES PER HOST PER WEEK FOR DIFFERENT SATELLITE LINKS

Bandwidth (kbit/s)	Centralized	Aloha	Slotted Aloha
128	4.83	0.87	1.74
256	9.68	1.74	3.48
512	19.35	3.48	6.97
1024	38.71	6.97	13.93
2048	77.41	13.93	27.87

second scheme is more expensive since we need to install both receivers and transmitters; typically this is done with Very Small Aperture Terminals, or VSATs. An average VSAT costs between $5,000 and $10,000 making the cost of a single replicated nameserver much higher than in the first scheme. Note, however, that in the first scheme we need also the central up-link station which increases the total cost of the installation. These costs are estimates for a system using geostationary satellites.

Reliability

Because having correct information in the replicated nameservers is vital to the DNS system, we must ensure a reliable means of distributing the update messages. This means that some sort or reliable multicast protocol must be employed, whether terrestrial or satellite multicast is used. Protocols using Forward Error Correction (FEC) along with a return channel (which is clearly available for replicated DNS) can efficiently provide reliability [11, 12].

D. Traffic Analysis for Communication among Replicated Servers

Assuming that each resource record is 40 bytes and that the central master replicated server is using a 128 kbit/s satellite channel to broadcast information to other replicated servers, we can send information about 400 modifications every second. Assuming that these modifications are spread uniformly over all hosts, the information about a host could change every 110,000 seconds on the average, or once per 1.3 days. This is less than the default time-to-live period of 2 days in the popular BIND-nameserver. We can therefore safely assume that the rate of change of DNS information is far less than what can be handled over a 128 kbit/s satellite link.

Table I shows the number of updates per host on the average for different satellite link bandwidths. We show the numbers for the centralized architecture and the distributed architecture using both pure Aloha and slotted Aloha. Recall that the throughput for Aloha is 18 % and for slotted Aloha it is 36 % [9]. In [12] it is reported that using the FEC-protocol from [11] we can expect to use about 10 % extra bandwidth due to FEC.

The numbers in Table I assume that the changes are evenly divided over all hosts. In reality this is unlikely, since most hosts do not change their DNS information at all. The hosts that rapidly change their associated DNS information may require high rates of change. Assuming that 90 % of the hosts on the Internet are "stable", i.e., they never change their DNS information, the remaining 10 % of the hosts could change their DNS information 48 times per week using the 128 kbit/s link; this corresponds to roughly 7 changes per day, or one change every 3.5 hours. Using a 1 Mbit/s link, the rapidly changing hosts could change DNS information 55 times per day, or little over twice an hour on the average.

E. Resolving DNS Queries

We discuss two variations for resolving DNS queries. Neither variation requires any changes in client DNS software. In the first variation, a client directly queries its replicated nameserver. (Thus, the client configures the client DNS software to point to its parent replicated nameserver rather than to a local nameserver.) Because the replicated nameserver contains a complete and and up-to-date copy of the entire DNS database, the replicated nameserver will be able to directly return the requested RRs. Furthermore, because the replicated nameservers are "close" to the client, the responses should come back fast.

One problem with the variation just described is that request load on a replicated server can be high if the load is not balanced over a large number of replicated servers. Our second variation for resolving DNS queries makes use of the existing infrastructure of local nameservers. In this variation, a client resolves a hostname as follows. First, the client sends a DNS query to its local nameserver. If the local nameserver is also the primary nameserver for the local zone and the query is for a local hostname, the local nameserver replies with the requested information. If the query is for a remote hostname and the local nameserver does not have the information cached, it sends the query to the nearest replicated nameserver. This phase is similar to a nameserver sending a query to a root nameserver with the exception that in our architecture the closest replicated server is typically well-defined; in normal DNS the nameservers measure round-trip times to other nameservers and make decisions based on past experience. (If the local nameserver has several replicated nameservers to choose from, it can use round-trip time measurements to select the closest one.) The replicated nameserver replies with the requested information, which the local nameserver caches and returns to the client. This scheme has the advantage that if the information is not cached, the query is sent to a replicated nameserver *in the same region as the client* instead of being sent possibly to the other side of the world as can happen in normal DNS. This greatly reduces the latency that a client observes when performing DNS queries.

In this second variation, a replicated nameserver receives less request traffic because many client requests are filtered by the local nameserver. In particular, all queries for a local host (i.e., in the same zone as the requesting host) and all queries for cached RRs are filtered. The downside of the scheme is that RRs in the nameserver caches can be stale (as is the case in the current DNS). Because one of our principle goals is to provide a DNS architecture that is highly responsive to hostname changes, the issue of staleness is important. In Section IV we will study in detail the tradeoff between the amount of traffic at a replicated server and the probability of receiving a stale RR.

F. Arpanet Name Resolution

This architecture is a step towards the old Arpanet `HOSTS.TXT` -solution [6] with some important differences. In

Arpanet, the HOSTS.TXT file, which was maintained on a single computer at Stanford Research Institute (SRI), contained the name-to-address mappings for all hosts on Arpanet. Administrators e-mailed changes to SRI and periodically transferred the HOSTS.TXT file over FTP. As Arpanet grew, several problems with this centralized solution emerged. First, the traffic load on the computer holding the master file became unbearable. Second, maintaining the consistency of the file across the network was difficult. Third, as more hosts were added to the Arpanet, the HOSTS.TXT occupied more and more storage in the server. In summary, the centralized mechanism didn't scale. Instead, a new system was designed to allow local administration of the data yet make the data globally available; this was the modern DNS.

Our replicated architecture takes the old HOSTS.TXT-approach by collecting rather than partitioning the entire DNS database. However, because of its replicated structure, our architecture scales well. When the modern DNS was designed in the early '80s disk storage was expensive; therefore maintaining a replicated database of all hosts on the network was infeasible. Nowadays, disk storage is cheap, and as we previously argued, it is quite feasible to construct a replicated database for all of the information in DNS.

IV. Staleness vs. Traffic at Replicated Nameservers

In this section we suppose that clients direct their DNS queries to local nameservers, and a query is only forwarded to a replicated nameserver when there is a "miss" at the local nameserver (i.e., the second variation as described in Section III-E). Passing queries through local name servers can significantly reduce the query traffic at the replicated servers. However, because the local nameserver caches RRs, there is a risk that the local nameserver will frequently reply with stale RRs.

When a Web site with a particular hostname is moved or copied to a new location, the RRs associated with that hostname change. If the Web site knows its locations will change in exactly t' seconds, then the TTL for the RRs can be set to the remaining lifetime of the RRs. This ensures that local nameservers never deliver to clients stale RRs for the site. However, in order to respond to randomly occurring hotspots, many sites will want to make spontaneous changes to their resource records. If the TTL for the RRs of such a site is set to a large value there is a risk that nameservers caching the RRs will frequently respond with stale RRs. On the other hand, if the TTL is set to a small value, then a large fraction of the queries will be forwarded to the replicated server. In this section we quantify this tradeoff.

For simplicity, we consider only the clients under a single local nameserver accessing one resource record (RR). This scenario is depicted in Fig. 3. The local nameserver either has a cached copy of the resource record, or has to query a replicated nameserver for an up-to-date copy of the resource record. When the local nameserver queries the replicated nameserver, the replicated nameserver indicates a time-to-live value in the reply. We consider the following two criteria:

1. The fraction of queries from the clients to the local nameserver that receive a stale RR because the RR has changed.
2. The number of extra queries to the replicated server which only serve to validate the cached copy at the local nameserver.

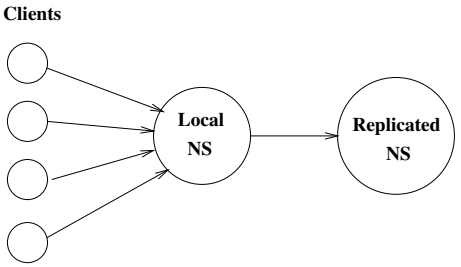


Fig. 3. Clients, Local Nameserver (NS), and Replicated NS

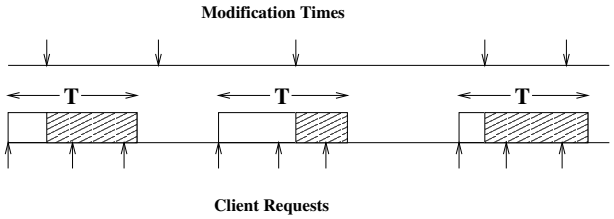


Fig. 4. Client Requests and Modification Times

Denote the rate at which clients request the resource record as λ. When there is a large number of independent and active clients under the nameserver, we can reasonably assume that the inter-request times are exponentially distributed. Denote by T the amount of time for which the local nameserver is allowed to cache the resource record. Finally, let μ denote the rate at which information in the RR changes. We assume that the times between modifications are exponentially distributed.

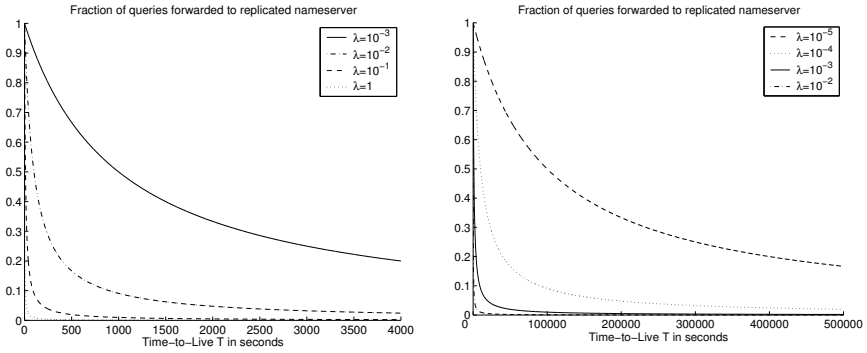
Fig. 4 shows the client queries and modifications to the resource record. The modification times are at the top and client queries are at the bottom. The boxes, each of length T, indicate the period during which the RR is cached at the local nameserver. The shaded boxes indicate the periods during which the clients receive a stale RR from the cache because the RR has changed.

We now calculate what fraction of requests are forwarded to the replicated nameserver. Given the above assumptions, the expected time between successive requests to the replicated nameserver is $T + \frac{1}{\lambda}$. Therefore the rate of requests to the replicated server is $\frac{1}{T+\frac{1}{\lambda}}$. Given that the total rate of requests is λ, the fraction of requests forwarded to the replicated nameserver is

$$\frac{1}{T+\frac{1}{\lambda}}/\lambda = \frac{1}{\lambda T+1}. \quad (1)$$

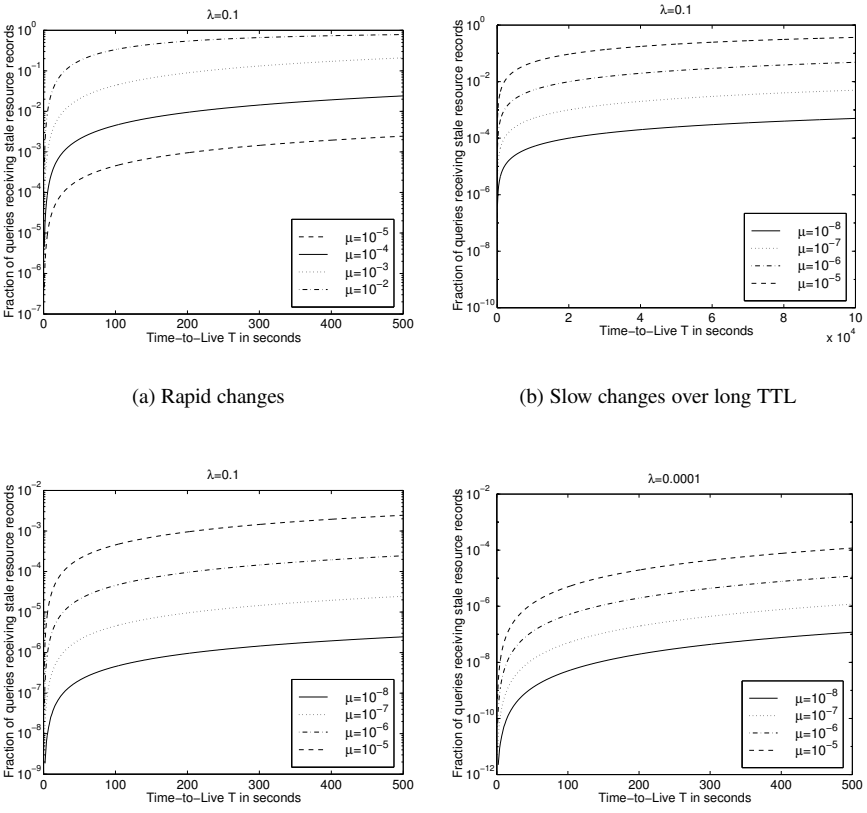
Fig. 5 shows the fraction of queries forwarded to the replicated nameserver for different λ and T. In Fig. 5(a) we show faster request rates and TTLs up to one hour and in Fig. 5(b) we show the slower request rates and TTLs up to 6 days.

From Fig. 5(a) we can see that if the request rate is high enough, on the order of one request every 10 seconds ($\lambda = 10^{-1}$), even a short TTL-value, such as 5 minutes, is sufficient to satisfy most queries from the cache at the local nameserver. Fig. 5(b) shows that if the request rate is very low, the TTL-value has to be extremely high to provide any measurable reduction in query traffic to the replicated nameserver. The situation shown



(a) Short TTL-values (b) Long TTL-values

Fig. 5. Fraction of queries forwarded to replicated nameserver



(a) Rapid changes (b) Slow changes over long TTL

(c) Slow changes, popular RR (d) Slow changes, unpopular RR

Fig. 6. Fraction of stale resource records delivered to clients

in Fig. 5(a) represents a scenario where the resource record is very popular and the product λT is large, i.e., the expected number of requests in a TTL-period is high. Fig. 5(b) corresponds to the case where the resource record is not very popular.

We now calculate the fraction of requests that receive stale RRs. Let Y denote the time between successive modifications. Within each interval of expected length $T + \frac{1}{\lambda}$, $E[N_{(T-Y)^+}]$ requests see stale resource records, where N_t is a Poisson process with rate λ. Thus the fraction of stale resource records within an interval is

$$\frac{E[N_{(T-Y)^+}]}{1+\lambda T} = \frac{\lambda E[(T-Y)^+]}{1+\lambda T}. \tag{2}$$

We can explicitly calculate the expectation:

$$\begin{aligned} E[(T-Y)^+] &= \int_0^T (T-y)\mu e^{-\mu y} dy \\ &= \frac{1}{\mu}(e^{-\mu T} + \mu T - 1) \end{aligned} \tag{3}$$

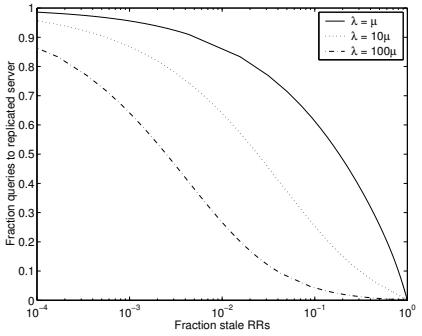


Fig. 7. Comparing fraction stale RRs and fraction queries to replicated server

The fraction of stale resource records is then

$$\frac{\lambda}{1+\lambda T}\left(\frac{e^{-\mu T}+\mu T-1}{\mu}\right)=\frac{1}{\frac{1}{\lambda}+T}\left(\frac{e^{-\mu T}+\mu T-1}{\mu}\right). \quad (4)$$

If $T >> \frac{1}{\lambda}$, i.e., we have a large number of requests in one TTL-period (λT is very large), then λ has very little effect on the overall fraction of stale resource records. In Fig. 6 we show the fraction of stale RRs delivered to the clients in four different scenarios. In Fig. 6(a) and Fig. 6(b) we show rapidly and slowly changing RR, respectively and Fig. 6(c) and Fig. 6(d) compare the effects of λ on slowly changing RRs.

Assuming that we would like to have a fraction of less than 10^{-3} stale RRs, we can see from Fig. 6(a) that this requires us to effectively disable caching at the local nameserver for RRs that change on the average more often than once every 3 hours ($\mu = 10^{-4}$). When the resource records are changing more slowly, as is the case in Fig. 6(b), we can see that even for RRs changing on the average every 115 days ($\mu = 10^{-7}$), the maximum TTL, in order to have the fraction of stale RRs below 10^{-3}, is around 12 hours. Figs 6(c) and 6(d) show that the more popular the RR is, the larger the fraction of stale RRs is (corresponding curves in Fig. 6(c) are higher than in 6(d)). With longer TTL-values, the fraction of stale RRs becomes insensitive to λ, because the increase in T increases the product λT.

In Fig. 7 we compare the fraction of stale RRs and fraction of queries to the replicated nameserver. The fraction of stale RRs is on the x-axis and the fraction of queries to the replicated nameserver is on the y-axis. We fix μ and λ ($\mu = 0.001$ and $\lambda \in \{0.001, 0.01, 0.1\}$; different values for μ and λ give similar results) and vary T to obtain the curves. As we can see from Fig. 7, to get the fraction of stale RRs below 10^{-3}, we will have to forward around 65 % of the queries to the replicated nameserver, even in the case where the RR is requested frequently compared to its rate of change.

From these results we can conclude that replicated nameserver should allow the local nameservers to cache RRs for only short periods of time, on the order of a few minutes. This is because longer TTL-values increase the fraction of stale RRs considerably, and even a short TTL-value provides a sufficient reduction in query traffic to the replicated nameserver for popular RRs.

TABLE II
DNS QUERY RESULTS FOR UNPOPULAR SERVERS

Site	Average latency	Maximum latency	% of lookups exceeding 4 seconds
FR	1.79	47.44	14.4
FI	1.85	42.2	15.5
US	2.10	75.4	6.6
All	1.90	75.4	12.5

V. LATENCY IMPROVEMENT

In this section we study how much our proposed architecture will reduce the time it takes to resolve a hostname. Because resolving some hostnames over the existing DNS requires contacting distant servers, a DNS lookup may introduce a significant delay into Web surfing and other network applications. To evaluate the delay in the existing DNS, we performed DNS lookups using the `host`- and `dnsquery`-commands on several different hostnames all over the Internet. In order to evaluate the effects of DNS lookups in a typical Web surfing context, we divided the hosts into two groups: popular and unpopular hosts. The popular hosts were the 35 most popular Web servers on the Hot100-list [13]. For the unpopular hosts we chose 200 Web servers randomly from 33 different top-level domains from the results of the Netcraft Web Server Survey [14]. We chose to study these two groups separately since the popular hostnames are likely to be found in the local nameserver and therefore the actual DNS architecture in the background has no effect. The less popular hostnames require the local nameserver to go out on the network to find the requested information. We performed the queries from three sites, one in France (FR), one in Finland (FI), and one in the US west coast (US). We discarded queries which either resulted in an error or a timeout (as reported by the command being executed).

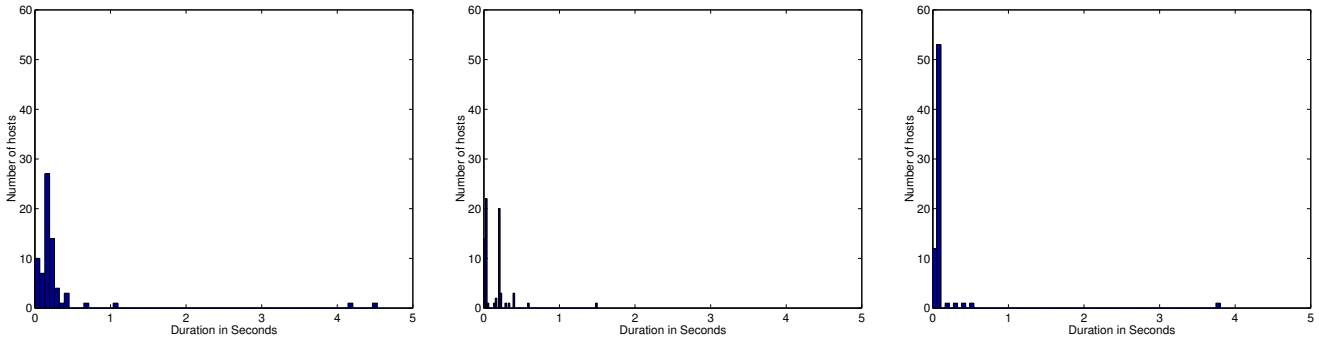
A. Unpopular Servers

Table II shows the results for the unpopular Web servers. We show the average and maximum query latencies and the percentage of requests exceeding *4 seconds*. We can see that the average DNS lookup latency is around 2 seconds and that at the worst it can take over a minute to resolve a hostname. We also see that a significant number of requests takes over 4 seconds to resolve which in the context of Web browsing can induce a significant delay.

In Fig. 8 we show the distributions of the query latencies for the unpopular servers at all three sites.

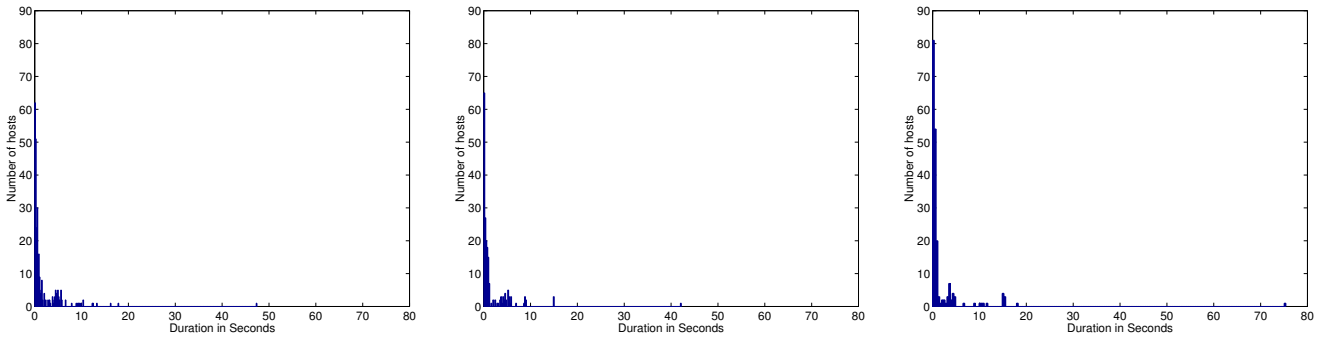
B. Popular Servers

In Table III we show the average and maximum DNS query latencies in seconds from all the three test sites as well as the percentage of queries that took longer than *2 seconds* to resolve. The last line shows the averages over all three sites. We can see that the average latencies are very low which is likely the result of the requested information being present in the local nameserver's cache. We also see that, at worst, the latency can be up to several seconds but that such events are relatively rare.



(a) France (b) Finland (c) US

Fig. 8. DNS query latency histograms for unpopular servers



(a) France (b) Finland (c) US

Fig. 9. DNS query latency histograms for popular servers

TABLE III
DNS QUERY RESULTS FOR POPULAR SERVERS

Site	Average latency	Maximum latency	% of lookups exceeding 2 seconds
FR	0.31	4.53	2.9
FI	0.14	1.5	0
US	0.22	3.8	2.9
All	0.22	4.53	1.7

In Fig. 9 we show the distributions of the query latencies for all the three sites.

C. Replicated Server

We also ran experiments on our local nameserver and the average time to resolve a hostname was 30 ms. The round-trip time to the national Web cache of France from our network is on the order of 20 ms, thus the total time to query a hostname at the replicated nameserver (assuming it was placed at the same level as the national Web cache) would be on the order of 50 ms. This is a reasonable assumption since, even though the database at the replicated nameserver is larger than the one on our local nameserver, the replicated nameserver would be running on a more powerful machine (our local nameserver runs on a SPARCstation 10). If the replicated nameserver was placed closer to the client, the lookup latency would be shorter, since the round-trip time to the replicated nameserver would be much smaller.

We can conclude that the speed-up from using a replicated nameserver would be around one order of magnitude for popular hosts and two orders of magnitude for less popular hosts. This speed-up can significantly improve the Web surfing experience.

VI. Graceful Migration from Distributed to Replicated System

Because it is unreasonable to assume that everybody would be able to upgrade their systems overnight to switch from the "legacy" DNS system to the replicated system, we need to be able to deploy the system incrementally and transparently. In this section we outline one of many possible migration strategies.

In the early stages of deployment, a relatively small number of zones could replace their secondary or primary nameservers with a replicated nameserver. Using satellite distribution or multicast IP, the replicated nameservers would share with each other the up-to-date RRs for which they are responsible. Of course, when only a subset of the local ISPs participate, a replicated

nameserver will not be able to authoritatively answer queries for all hostnames. When a replicated server is unable to answer a query, it will have to resort to the legacy DNS system to obtain the RR, i.e., it will have to query a series of nameservers, as shown in Fig. 1. Whenever a replicated nameserver obtains a RR as a result of a DNS query, it should cache that RR until the TTL of the RR has expired. Additionally, using the satellite or multicast IP infrastructure, we propose that *the replicated nameservers share with each other the RRs that they have recently requested from the legacy DNS system*. In this manner, most of the "popular" hostnames will be cached in the replicated servers, even if a given replicated nameserver handles relatively little DNS traffic. When a RR is about to expire, a replicated server can refresh the RR and then distribute it to all the other replicated servers. Note that this strategy is similar to the Sky Cache model where the participating caches pool their user communities in order to improve the performance. The advantage of this scheme is that it is easy to deploy a few replicated nameservers at the leaves of the network, but the disadvantage is that the replicated database will only contain a small (but widely requested) part of the global DNS database. In addition, only participating zones are able to distribute rapidly changing DNS information; non-participating zones would have to rely on the TTL-values to force the replicated nameservers to update their databases.

Content providers would have an incentive to place their hostnames in zones with participating ISPs (i.e., an ISP with a a replicated server), because a participating ISP will be able to quickly distribute DNS information to all other participating ISPs, even if the hosts are moved or replicated. Furthermore, users will have an incentive to subscribe to participating ISPs, since the databases in the replicated ISPs contain most of the popular DNS RRs. These incentives should provide sufficient motivation for the non-participating ISPs to become participating ISPs. Once all the ISPs participate, the legacy system (including the root servers) can be disabled, and the full benefits of the replicated DNS system can be reaped.

A. Gathering Information

If a replicated nameserver replaces a secondary nameserver for a zone, the replicated nameserver can perform zone transfers from the primary nameserver and thus obtains a copy of the resource records for that zone. Therefore the problem of gathering the DNS information reduces to getting information from zones that are not represented by a replicated nameserver. The simple solution is to contact one of the authoritative nameservers of such zones and try to perform a zone transfer. If this succeeds, the replicated nameserver can periodically perform the zone transfer in order to keep the records up-to-date. In this situation, we lose the ability to track rapidly changing information since the primary nameserver does not inform the replicated nameserver of changes. Some zones, however, do not allow other than the secondary nameservers to perform zone transfers from them. From these zones we cannot get information, except by querying separately for each resource record. In practice this means that every time the replicated nameserver receives a query for a host in such a zone, the replicated nameserver queries one of the authoritative nameservers and caches the reply like a normal local nameserver. The replicated nameserver can then refresh the resource record when it is about to expire. We will thus obtain from uncooperative zones all the resource records that clients actually request; this provides us with enough information.

This method of gathering information presents us with the problem of keeping it up-to-date. All RRs collected from uncooperative zones will have the normal DNS TTL-stamp and the replicated nameservers can store the RR until the TTL expires. If the TTL is very short, the replicated nameservers should not send that RR to the other replicated nameservers. This is in order to avoid sending an almost continuous stream of update messages which would be the result if the replicated nameserver needs to access the same RR again in the near future. Sending the updates would place an unnecessary burden on the other replicated nameservers which would have to store the RR and shortly after remove it because its TTL had expired. As a results, the replicated nameservers will contain all the RRs from the zones they are responsible for as well as the slowly changing RRs from other zones.

VII. Security Issues and Fault Tolerance

Our architecture has one serious security hole, namely, it creates a new possibility for DNS spoofing. DNS spoofing goes as follows.

A user (`pc.client.com`) wants to access the Web servers of two competing companies, `www.company-a.com` and `www.company-b.com`. First, the client issues a DNS lookup for the address of the first server (`www.company-a.com`). The request goes to client's local nameserver, `ns.client.com`, which does not have the answer. The local nameserver contacts the nameserver of Company-A, `ns.company-a.com`, and asks for the IP-address of the Web server. Company-A's nameserver replies with the address and includes in the reply a false A-record for the Web server of Company-B, `www.company-b.com`. The reply, including the false address, is cached at the local nameserver and if the client tries to access `www.company-b.com` while the mapping is still cached, the local nameserver will return the false mapping. The client is thus redirected to whatever server the nameserver at Company-A claimed was the Web server of Company-B. This problem stems from the local nameserver's willingness to accept information that it did not ask (the IP-address of `www.company-b.com`). Modern versions of nameservers (e.g., BIND) block this security hole by not accepting RRs they have not asked.

Our architecture is vulnerable to this type of attack if replicated nameservers accept all information they receive from other nameservers (primary or replicated). In this case, the attack would be very simple. The administrator of a primary nameserver would only have to send a false mapping to the replicated nameserver and the false information would immediately be propagated to everyone on the network. If a client were to request this information, it would receive both the real information (given by the real primary nameserver) and the false information (given by the imposter). The client could easily choose to use the wrong address instead of the correct one.

To counter this problem, the replicated nameservers must ver-

ify that the RRs they receive come from a server that is authorized to provide this information, e.g., the primary nameserver of that zone. We propose the following solution. Each replicated nameserver is responsible for a zone (each of these zones may contain several DNS zones) and can only provide information for that zone. When a replicated nameserver receives updates from a primary nameserver or another replicated nameserver, it must verify that the server sending the original update is authorized to do so. This requires that we extend the DNS SOA-type RR to include the zones handled by replicated nameservers. In addition, to avoid replicated nameservers from sending RRs for non-existing zones, these extended SOA-RRs need to be signed by a trusted certification entity using the DNS security extensions [15].

A. Fault Tolerance

We must have a way of recovering from replicated nameserver crashes. These crashes present us with two problems. First, clients that were using the crashed server must be redirected to another server. Second, when the server comes back on-line, it will have a stale copy of the database and it must get a fresh copy.

To address the first problem, we propose that all local nameservers be configured with the addresses of *several* replicated nameservers. This is similar to configuring a local nameserver with the addresses of all 13 root DNS servers in modern DNS. If the client does not get a reply from a replicated nameserver even after retrying a few times, it can assume that that server has crashed and can switch to another replicated nameserver. In this situation the crash of a replicated nameserver is analogous to the crash of a root DNS server. If the replicated nameserver is acting as the local nameserver for the client, then the local clients will be without DNS service. This situation is identical to the crash of a local nameserver in modern DNS and can be handled by installing several normal local nameservers in addition to the replicated nameserver to be used as backups. These local nameservers would have the addresses of other replicated nameservers and could provide name resolution service to the clients.

The second problem, that of stale information, is more serious and more difficult to handle. When the crashed server is back on-line it must obtain a fresh copy of the database. Because the database is on the order of gigabytes, it is infeasible to download the whole database from another replicated server. We propose the following method for bringing the database up-to-date. First, all the update messages are tagged with a unique identifier (e.g., a counter) in addition to being tagged with the identity of the sender of the update message. This sender is the primary nameserver that owns the resource record, not the replicated nameserver that distributes it to the others; this is required to causally order successive modifications. When a server recovers from a crash, it knows the number and sender of the last update message it has received, and can request another replicated nameserver to send it all the update messages from that primary nameserver since that "time."

The above scheme is sufficient when the replicated server is off-line for a short period of time and we only need to reconstruct the parts that have been updated during that period. In some cases, however, this method may be costly, since the replicated nameserver has to check for updates for all the DNS zones in the world. Also, should a replicated nameserver crash so seriously that the entire database is corrupted and must be rebuilt from scratch, we propose the following strategy. In this strategy, the replicated nameserver first performs zone transfers from the primary nameservers in its zone. When the replicated nameserver receives a query for a RR it does not have, it contacts another replicated nameserver and asks for the RR. It also performs a zone transfer for the zone containing the RR *from the other replicated nameserver*. As the replicated nameserver receives updates from other replicated nameservers, it can also perform zone transfers for the zones concerned. This way, the replicated nameserver will eventually obtain a copy of all the RRs.

VIII. Conclusion

In this paper we have presented a new design for managing the DNS database that takes advantage of recent advances in disk storage and multicast distribution technology. Our design is based on replicating the entire DNS database on geographically distributed servers, called replicated servers. Our design has two main features: (i) It is highly responsive to changes in DNS information and (ii) significantly reduces DNS lookup time. We have closely studied the issues related to storing the DNS database and evaluated the tradeoff between the staleness of DNS information and traffic load on the replicated DNS servers.

References

[1] "SkyCache, Inc.," `http://www.skycache.com`.

[2] P. V. Mockapetris, *RFC 1034: Domain names — concepts and facilities*, Nov. 1987.

[3] P. V. Mockapetris, *RFC 1035: Domain names — implementation and specification*, Nov. 1987.

[4] "DNS root servers," `ftp://rs.internic.net/netinfo/root-servers.txt`.

[5] "Internet domain survey," Jul. 1999, `http://www.isc.org/ds/`.

[6] P. Albitz and C. Liu, *DNS and BIND*, O'Reilly & Associates, Inc., 1994.

[7] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound, *RFC 2136: Dynamic Updates in the Domain Name System (DNS UPDATE)*, Apr. 1997.

[8] H. Eriksson, "MBONE: The multicast backbone," *Communications of the ACM*, vol. 37, no. 8, pp. 54–60, Aug. 1994.

[9] H. Peyravi, "Medium access control protocols performance in satellite communications," *IEEE Communications Magazine*, vol. 37, no. 3, pp. 62–71, Mar. 1999.

[10] "DirecPC home page," `http://www.direcpc.com/`.

[11] J. Nonnenmacher, E. W. Biersack, and D. Towsley, "Parity-based loss recovery for reliable multicast," *IEEE/ACM Transactions on Networking*, vol. 6, no. 4, pp. 349–361, Aug. 1998.

[12] M. Jung, J. Nonnenmacher, and E. W. Biersack, "Reliable multicast via satellite: Uni-directional vs. bi-directional communication," in *Proceedings of KiVS'99*, Darmstadt, Germany, Mar. 1999.

[13] "100hot sites," `http://www.hot100.com`.

[14] "Netcraft web server survey," `http://www.netcraft.com/survey/`.

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