Maintaining Strong Cache Consistency for the Domain Name System

Xin Chen, Haining Wang, *Member*, *IEEE*, Shansi Ren, *Student Member*, *IEEE*, and Xiaodong Zhang, *Senior Member*, *IEEE*

Abstract—Effective caching in the Domain Name System (DNS) is critical to its performance and scalability. Existing DNS only supports weak cache consistency by using the Time-to-Live (TTL) mechanism, which functions reasonably well in normal situations. However, maintaining strong cache consistency in DNS as an indispensable exceptional handling mechanism has become more and more demanding for three important objectives: 1) to quickly respond and handle exceptions such as sudden and dramatic Internet failures caused by natural and human disasters, 2) to adapt increasingly frequent changes of Internet Protocol (IP) addresses due to the introduction of dynamic DNS techniques for various stationed and mobile devices on the Internet, and 3) to provide fine-grain controls for content delivery services to timely balance server load distributions. With agile adaptation to various exceptional Internet dynamics, strong DNS cache consistency improves the availability and reliability of Internet services. In this paper, we first conduct extensive Internet measurements to quantitatively characterize DNS dynamics. Then, we propose a proactive DNS cache update protocol (*DNScup*), running as middleware in DNS name servers, to provide strong cache consistency for DNS. The core of DNScup is an optimal lease scheme, called dynamic lease, to keep track of the local DNS name servers. We compare dynamic lease with other existing lease schemes through theoretical analysis and trace-driven simulations. Based on the DNS Dynamic Update protocol, we build a DNScup prototype with minor modifications to the current DNS implementation. Our system prototype demonstrates the effectiveness of DNScup and its easy and incremental deployment on the Internet.

Index Terms—Domain name system, cache consistency, middleware, lease.

1 Introduction

The Domain Name System (DNS) is a distributed database that provides a directory service to translate domain names to Internet Protocol (IP) addresses [22], [23]. DNS consists of a hierarchy of name servers, with 13 root name servers at the top. For such a hierarchical system, caching is critical to its performance and scalability. To determine the IP address of a domain name, the DNS resolver residing at a client sends a recursive query to its local DNS name server. If no valid cached mapping exists, then the local DNS name server will resolve the query by iteratively communicating with a root name server, a Top-Level Domain (TLD) name server, and a series of authoritative DNS name servers. All the replied DNS messages including referrals and answers are cached at the local DNS name server so that subsequent queries for the same domain name will be answered directly from the cache. Therefore, DNS caching significantly reduces the workload of root and TLD name servers, lookup latencies, and DNS traffic over the Internet.

With the deployment of caches, cache consistency has become a serious concern. Strong cache consistency is defined as the model in which no stale copy of a modified original will be returned to clients, whereas weak cache consistency is the model in which a stale copy might be returned to clients. Currently, DNS only supports weak cache consistency by using the Time-to-Live (TTL) mechanism. The TTL field of each DNS resource record indicates how long it may be cached. The majority of TTLs of DNS resource records range from 1 hour to 1 day [17]. Although most of the domain-name-to-IP-address (DN2IP) mappings are infrequently changed, the current approach to coping with an expected mapping change is cumbersome. Among numerous DNS-related Request For Comments (RFCs), only RFC 1034 [22] briefly describes how an expected mapping change can be handled: "If a change can be anticipated, the TTL can be reduced prior to the change to minimize inconsistency during the change, and then increased back to its former value following the change." However, the RFC does not specify how much and in what magnitude the TTL value should be reduced. The propagation of the mapping change may take much longer than expected. This pathology is further aggravated by some local DNS name servers that do not follow the TTL expiration rule and violate it by a large amount of time [24].

Therefore, without strong cache consistency among DNS name servers, it is cumbersome to invalidate the out-of-date cache entries. The inefficient and pathological DNS cache update due to weak consistency quite often causes service disruption. More importantly, three recently emerged reasons, in practice, cast serious doubt on

- *X. Chen is with Ask.com/IAC/Search and Media, 343 Thornall Street, Suite 730, Edison, NJ 08837. E-mail: xchen@ask.com.*
- *H. Wang is with the Department of Computer Science, College of William and Mary, McGlothlin-Street Hall, Williamsburg, VA 23187. E-mail: hnw@cs.wm.edu.*
- *S. Ren and X. Zhang are with the Department of Computer Science and Engineering, The Ohio State University, 2015 Neil Avenue, Columbus, OH 43210. E-mail: {sren, zhang}@cse.ohio-state.edu.*

Manuscript received 9 Apr. 2006; revised 25 Dec. 2006; accepted 28 Mar. 2007; published online 18 Apr. 2007.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0165-0406.
Digital Object Identifier no. 10.1109/TKDE.2007.1049.

the efficacy of weak DNS cache consistency provided by the TTL mechanism:

- There are many unpredictable mapping changes due to emergency situations such as terror attacks and natural disasters, in which the loss or failure of network resources (servers, links, and routers) is inevitable [15], and we have to immediately redirect the affected Internet services to alternative or backup sites. Maintaining DNS cache consistency is critical under such an exceptional circumstance, since people need service availability at the crucial moment.
- The dynamic DNS technique, which provides prompt IP mapping for a server at home or a mobile host using a temporary IP assigned by the Dynamic Host Configuration Protocol (DHCP), makes the association between a domain name and its corresponding IP address much less stable.
- The TTL-based DNS redirection service provided by Content Distributed Networks (CDNs) only supports a coarse-grained load balance and is unable to support quick reaction to network failures or flash crowds without sacrificing the scalability and performance of DNS [24].

Thus, *cache inconsistency poses a serious threat to the availability of Internet services*. This is simply because during the cache inconsistency period, the clients served with out-of-date DN2IP mappings cannot reach the appropriate Internet servers or end hosts. Once it happens, the clients have no idea of what the cause of service unavailability is: Is it due to server shutdown, network failure, or something else? An aggressively small TTL (on the order of seconds) can lower the chance of cache inconsistency but at the expense of significant increase of the DNS traffic, name resolution latency, and the workload of domain name servers [32], which seriously degrades the scalability and performance of DNS.

In this paper, we propose a proactive DNS cache update protocol (*DNScup*), working as middleware to maintain strong cache consistency among DNS name servers and improve the responsiveness of DNS-based service redirection. The core of DNScup uses a dynamic lease technique to keep track of the local DNS name servers whose clients are tightly coupled with an Internet server.[1] Upon a DN2IP mapping change of the corresponding Internet server, its authoritative DNS name server proactively notifies these local DNS name servers still holding valid leases. Although the notification messages are carried by the User Datagram Protocol (UDP), dynamic lease also minimizes storage overhead and communication overhead, making DNScup a lightweight and scalable solution. Based on client query rates (or service importance to their clients), it is the local DNS name servers themselves that decide on whether or not to apply for leases (or renewal) for an Internet service. On the other side, the authoritative DNS name server grants and maintains the leases for the DNS resource records of the Internet service. The duration of a lease is dependent on the DN2IP mapping change frequency of the specific DNS resource record.

Although strong cache consistency may be optional for a generic Internet service, DNScup is essential to provide always-on service availability for critical Internet services or some premium clients. In addition to maintaining cache coherence among DNS name servers, DNScup can also be used to improve the responsiveness of DNS-based network control, as suggested in [24]. Also, we can apply the functionality of DNScup to maintain state consistency between a DNS name server of a parent zone[2] and the DNS name servers of its child zones, preventing the lame delegation problem [27].

Based on the DNS Dynamic Update protocol [31], we build a DNScup prototype with minimized modifications to current DNS implementation [14], [23]. Our trace-driven simulation and prototype implementation demonstrate that DNScup achieves strong cache consistency of DNS and significantly improves its performance and scalability. Note that DNScup is backward compatible with the TTL mechanism and can be incrementally deployed over the Internet. Those local DNS name servers without valid leases still rely on the TTL mechanism to maintain weak cache inconsistency.

The remainder of the paper is organized as follows: Section 2 surveys related work. Section 3 presents our DNS dynamics measurements. Section 4 details the proposed DNScup mechanism. Section 5 evaluates the performance of DNScup based on the trace-driven simulations. Section 6 presents the prototype implementation of DNScup. Finally, we conclude the paper in Section 7.

2 Related Work

DNS performance at either root name servers [6], [12] or local DNS name servers and their caching effectiveness [17], [19], [36] have been studied in the past decade. Danzig et al. [12] measured the DNS performance at one root name server and three domain name servers. They identified a number of bugs in DNS implementation, and these bugs and misconfigurations produced the majority of DNS traffic. Brownlee et al. [6] gathered and analyzed DNS traffic at the F root name server. They found that several bugs identified by Danzig et al. still existed in their measurements, and the wide deployment of negative caching would reduce the impact caused by bugs and configuration errors. Observing a large number of abnormal DNS update messages at the top of the DNS hierarchy, Broido et al. [5] discovered that most of them are caused by default configurations in Microsoft DHCP/DNS servers. The load distribution, availability, and deployment patterns in local and authoritative DNS name servers have been characterized in [25]. Based on a half-year measurement, Pappas et al. [27] thoroughly investigated the negative impact of operational errors upon DNS robustness. Furthermore, they presented a distributed troubleshooting tool to identify these DNS configuration errors [26].

1. Either the clients frequently visit the Internet server or the services provided by the Internet server are critical to the clients.

2. Zone is a delegated authority unit that is a manageable domain namespace.

Jung et al. [17] measured the DNS performance at local DNS name servers (Massachusetts Institute of Technology (MIT) and Korea Advanced Institute of Science and Technology (KAIST)) and evaluated the effectiveness of DNS caching. They conducted a detailed analysis of collected DNS traces and measured the client-perceived DNS performance. Based on trace-driven simulations, they found that lowering the TTLs of type-A record to a few hundred seconds has little adverse effect on cache hit rates, and caching of NS records and protecting a single name server from overload are crucial to the scalability of DNS. Instead of collecting data at a few client locations, Liston et al. [19] compared the DNS measurements at many different sites and investigated the degree to which they vary from site to site. They identified the measures that are relatively consistent throughout the study and those that are highly dependent on specific sites. Based on both laboratory tests and live measurements, Wessels et al. [36] found that existing DNS cache implementations employ different approaches in query load balancing at the upper levels. They suggested longer TTLs for popular sites to reduce global DNS query load.

Shaikh et al. [32] demonstrated that aggressively small TTLs (on the order of seconds) are detrimental to DNS performance, resulting in the increases of name resolution latency (by two orders of magnitudes), name server workload, and DNS traffic. Their work further confirmed that DNS caching plays an important role in determining client-perceived latency. Wills and Shang [38] found that only 20 percent of DNS requests are not cached locally, and noncached lookups cost more than 1 sec to resolve. The same authors explored the technique of actively querying DNS caches to infer the relative popularity of Internet applications [37]. Using graphs, Cranor et al. [11] identified local and authoritative DNS name servers from large DNS traces, which is useful for locating the related DNS caches.

Park et al. [28] identified internal failures as a major source of delays in the PlanetLab testbed and proposed a locality and proximity-aware design to resolve the problem. They utilized a cooperative lookup service, in which remote queries are sent out when the local DNS name server experiences problems, to mask the failure-induced local delay. In their design, they considered the importance of cache at the local DNS name server for providing shared information to all local clients and avoided a design that makes the cache useless.

However, none of the previous work focuses on DNS cache consistency. DNS cache inconsistency may induce a loss of service availability, which is much more serious than performance degradation. By contrast, maintaining strong cache consistency in the Web has been well studied. Liu and Cao [20] showed that achieving strong cache consistency with server invalidation is a feasible approach, and its cost is comparable to that of a heuristic approach like adaptive TTL for maintaining weak consistency. To further reduce the cost of server invalidation and its scalability, Yin et al. proposed volume lease [41] and its extension [39], [40] for maintaining Web cache consistency. Instead of keeping a per-client state, Mikhailov and Wills [21] proposed Management of Objects in a Network using Assembly, Relationships, and Change Characteristics (MONARCH) to provide strong cache consistency for Web objects, in which invalidation is driven by client requests. They evaluated MONARCH by using snapshots of collected contents. The weakness of MONARCH is that it does not consider the dynamics of Web page structures.

The adaptive lease algorithm has been proposed in [13] to maintain strong cache consistency for Web contents. A Web server computes the lease duration on the fly based mainly on either the state space overhead or the control message overhead. However, in their analytical models, the space and message overhead are considered separately without gauging the possible trade-offs. Thus, the performance improvement of the adaptive lease algorithm is limited. Cohen and Kaplan [9] proposed proactive caching to refresh stale cached DNS resource records in order to reduce the name resolution latency. However, the client-driven prefetching techniques only reduce the client-perceived latency and cannot maintain strong cache consistency.

Cox et al. [10] considered using the Peer-to-Peer system to replace the hierarchical structure of DNS name servers. For example, for a given Web server, we can search a distributed hash table (DHT) to find its IP address instead of resolving it by DNS. However, compared with conventional DNS, the main drawback of this alternative approach is the significantly increased resolving latency due to P2P routing, although the approach has a stronger support for fault tolerance and load balance.

Based on DHTs [18], Beehive, which is designed for domain name system [30], provides O(1) lookup latency. Differently from widely used passive caching, it uses proactive replication to significantly reduce the lookup latency. In order to facilitate Web object references, *Semantic Free Reference* (SFR) [34], which is also based on DHTs [18], has been proposed to resolve the object locations. SFR relies on the caches at different infrastructure levels to improve the resolving latency. Note that these proposed schemes are heavily dependent on a future and wide deployment of DHTs; thus, the consequent dramatic changes to the Internet directory service will take a large amount of time and effort to become a reality. In contrast, DNScup is an effective enhancement to the current DNS implementation, which can fix the problem in a timely and cost-effective manner.

Although DNS caching does not support strong consistency, the DNS Dynamic Update mechanism [31] maintains a strong consistency between the primary master DNS name server of a zone and its slave DNS name servers within the same zone. The DNS Dynamic Update mechanism [31] and its enhanced secure version [35] have been proposed and implemented to support dynamic addition and deletion of DNS resource records within a zone because of the widespread use of DHCP. According to the DNS Dynamic Update protocol, once the primary master has processed dynamic updates, its slaves will be automatically notified about these changes via zone transfers. Researchers have utilized the DNS Dynamic Update protocol to achieve end-to-end host mobility [33]. In terms of DNS semantics, our proposed DNS cache update mechanism can be viewed as an external

extension to the DNS Dynamic Update protocol, which makes the implementation and deployment of DNScup much easier. The required modifications and additions to the current DNS implementation are minimized.

3 DNS Dynamics Measurement

The purpose of our DNS dynamics measurement is to answer the question of how often a DN2IP mapping changes. In general, a mapping change may cause two different effects. If the original DN2IP mapping is one to one, then the change may lead to the loss of Internet services. We classify these kind of changes as physical changes. However, if the original DN2IP mapping is one to many, then the changes may be anticipated to balance the workload of a Web site as CDN does. We classify these changes as logical changes.

To examine the DN2IP mapping change behaviors, one possible way is to use `dig` to contact remote name servers directly without using a local cache. However, we observe that only about half of authoritative DNS name servers allow direct communication with remote resolvers. Therefore, we set up a local DNS name server using Bind 9.2.3 [3] to generate probing DNS queries for a collection of Web sites (more than 15,000). In order to guarantee that each response comes from an authoritative DNS name server, instead of the local cache, we purge our local cache every time we probe a Web site. The measurement experiments were conducted in two months. In the rest of this section, we describe the DNS resource record classification and the collection of domain names. Also, we present a technique to differentiate the domains using CDN, in which most mapping changes are logical changes, from the domains where most mapping changes are physical changes. According to the affiliated TLD and their popularities, we further categorize the domains into several groups. Then, we measure the TTLs of their DNS resource records and investigate the effect of domain popularity upon DNS TTL behaviors. Based on the measured TTLs, we choose the appropriate sampling resolution to detect the DN2IP mapping changes.

3.1 DNS Resource Record Classification

The various mappings in the DNS namespace are called resource records. The most widely used resource records include `SOA` records (authority indication for a zone), `NS` records (authoritative name server reference lists for a zone), `A` records (DN2IP mappings), `PTR` records (IP address to domain name mappings), `MX` records (mail exchangers for a domain name), and `CNAME` records (alias to canonical name mappings). A type-`A` record provides the standard DN2IP mapping, whereas the other types of records like `NS`, `CNAME`, and `MX` records are used as references. Among these DNS resource records, the type-`A` record is the most popular record being queried, accounting for about 60 percent of DNS lookups on the Internet [17].

Any type of resource records listed above may change for various reasons. For example, the primary master DNS name server within a zone may increase the serial number in `SOA` records to keep the records of the zone's slaves updated, `NS` and `MX` records need to be updated if any authoritative DNS name server or mail exchanger is renamed, `A` and `PTR` records need to be changed if the domain name is either renamed or mapped to a different IP address, and changes on `CNAME` records have already been utilized by CDN providers to redirect a client request to different surrogates. Note that CDN providers and popular Web sites rotate different `A` records with small TTLs for the same domain name to balance the workload of Web servers.

In various DNS resource records, the inconsistency of `A` records may directly lead to service unavailability. In practice, more than one authoritative name server and mail exchanger serves for the same zone to improve reliability. However, the inconsistency of `NS` (or `MX`) records may also cause serious performance degradation and access problems due to lame delegation [27]. In general, our solution is applicable to all kinds of resource records, whereas our DNS measurement is focused on the dynamics of `A` records.

3.2 Domain Name Collection and Grouping

Since Web service is one of the most popular Internet services, our measurements are focused on the dynamics of the mappings between Web domain names and their corresponding IP addresses. We collected the Web domain names from the recent IRCache [4] proxy traces. All Web domain names are classified into three categories: domains using CDN techniques, domains using dynamic DNS techniques, and the rest of collected domains. We refer to them as CDN domains, Dyn domains, and regular domains, respectively. Because most CDN domains and Dyn domains have specific text strings to indicate the names of their providers (for example, `Akamai` for CDN domains and `DynDns.com` for Dyn domains), we can distinguish those domains from the regular ones by the specific strings. In our measurement experiments, we examined 23 major CDN providers [1] and 95 major dynamic DNS providers [2].

Due to the large number of regular domains that we collected, the regular domains are further divided into nine groups with respect to their TLDs. They are ended with `.com`, `.edu`, `.net`, `.org`, `.mil`, `.gov`, `.biz`, `.coop`, and country codes, respectively. The regular domain name distribution with the number of requests in each group is plotted in Fig. 1. As shown in Fig. 1, most regular domain names fall into five major groups: `.com`, `.net`, `.org`, `.edu`, and country domains. Each group consists of three subgroups:

- popular domains (with the number of requests being larger than or equal to 100 in our 1-week trace[3]),
- normal domains (with the number of requests being less than 100 but larger than or equal to 10 in the 1-week trace), and
- unpopular domains (with the number of requests being less than 10 in one trace).

We select 1,000 domain names from each subgroup of the five major groups, except the popular one of `.edu` group, where we only have 514 domain names available. Note that not all domain names in our regular domain groups follow

3. The limited client space and the hidden load factor of caching reduce the number of requests that we have seen.

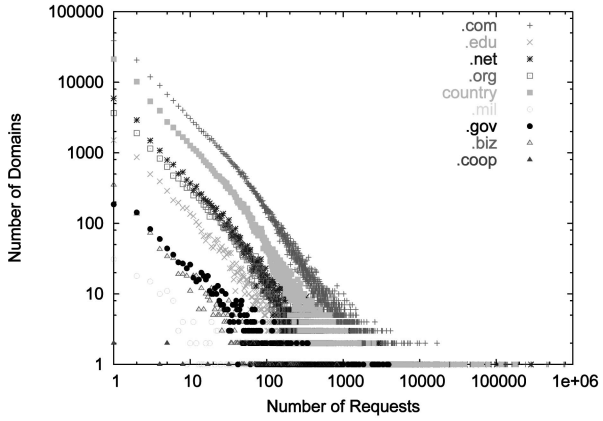

Fig. 1. The regular domain name distribution with the number of requests in each group.

the strict one-to-one mapping between domain names and IP addresses. Some domain names may use CNAME to avoid the direct use of CDN/dynamic DNS providers.

3.3 TTL Distribution

Different domain names have different TTL values for caching their DNS replies. The TTL distribution of all measured domains is shown in Fig. 2a. For CDN domains, the majority of TTLs have the values of 20 or 120 seconds. For Dyn domains, the majority of TTLs have the values of 30, 60, or 90 seconds. For regular domains, the majority of TTLs have the values of 300, 3,600, or 86,400 seconds. The TTL distribution of .com domain names with different popularities is shown in Fig. 2b. The TTL distributions for other kinds of domains with different popularities are similar to that of .com. We observe that the TTL of a domain name is independent of the domain's popularity.

The sampling resolution of detecting a DN2IP mapping change is highly dependent upon the values of TTLs. On one hand, our sampling resolution for a specific Web domain should be at least as small as its TTL in order to capture every possible change that could cause cache inconsistency. On the other hand, to minimize the impact of probing DNS traffic, our sampling resolution should be set as large as possible. Based on the measured TTLs' distribution, we set different sampling resolutions to detect DN2IP mapping changes at different Web sites. The sampling resolutions with respect to the range of TTLs are listed in Table 1.

3.4 Measurement of Mapping Changes

Each domain name in our collection is periodically resolved to check if the mapping has been changed. Depending on the sampling resolution, the duration of a measurement experiment varies from 1 day to 1 month. According to the sampling resolution, the Web domain names being probed in our measurements are divided into five classes, as shown in Table 1. Since all CDN and Dyn domains' TTL values are bounded by 300 sec, they belong to either class 1 or 2. The regular domains of each TLD may fall in all five possible classes because of the wide spectrum of their TTLs.

3.4.1 Dynamics of Mapping Changes

A DN2IP mapping change is detected when the responses of two consecutive DNS probes for the same domain name are different from each other. We define the relative change frequency of a domain name as the ratio between the number of mapping changes that we detected and the total number of DNS probes that we sent for that domain name. The absolute change rate is the product of relative change frequency and the reciprocal of sampling resolution. For ease of presentation, we employ relative change frequency as the metric to study the dynamics of DN2IP mapping changes and simply call it change frequency in the rest of this paper. Note that the sampling resolution varies among different classes. Given the same relative change frequency, the corresponding absolute change rates under different classes are different.

The change frequencies for the five different classes are shown in Figs. 3a, 3b, 3c, 3d, and 3e, respectively.[4] Based on the DNS probing results, we identify three causes that lead to the DN2IP mapping changes: 1) a domain name is relocated to a different IP address, 2) the available IP addresses for a domain name are increased, and 3) the IP address of a domain name rotates around a set of IP addresses. The first cause results in physical changes, whereas the second and third causes result in logical changes. The distributions of the changes due to different causes are shown in Fig. 3f for all five classes.

Physical changes. As shown in Figs. 3c, 3d, and 3e, the domains in classes 3, 4, and 5 rarely change their DN2IP mappings, with about 95 percent of the domains in these classes remaining intact. Moreover, those domains that have changed their DN2IP mappings have very low change frequencies. For instance, in class 5, almost all changed domains have their change frequencies below 10 percent,[5] which means that a change happens every 10 days. On the average, the change frequencies are about 3 percent, 0.1 percent, and 0.2 percent for the domains in classes 3, 4, and 5, respectively. This implies that the average lifetimes of DN2IP mappings are 2.5 hours, 42 days, and 500 days, respectively. However, as shown in Fig. 3f, nearly 40 percent of the mapping changes in class 3 and the majority of mapping changes in classes 4 and 5 are physical changes. Any physical change could cause a cache inconsistency, leading to a loss of service availability. Considering the large number of domain names in classes 3, 4, and 5, the probability of a physical change happening per minute is close to 1. Therefore, maintaining strong cache consistency is essential to avoid connection loss.

Logical changes. The DN2IP mappings in classes 1 and 2 are changed frequently. In class 1, more than 70 percent of the domains changed their IP addresses during a 1-day measurement. Most changed domains have their change frequencies around 0.1.[6] In class 2, only about 20 percent of the domains changed their IP addresses during a 3-day measurement, but most changed domains have relatively high frequencies (for example, 0.8). On average, the change frequencies of classes 1 and 2 are about 10 percent and

4. We also monitored the mapping changes of the corresponding MX and NS records. Our results show that their change frequencies are lower than that of A records.

5. In the 30-day measurement, 214 domains changed one to three times and only seven domains changed four to 19 times among 5,307 domains in class 5.

6. Among all 803 domains in class 1, 442 domains changed every 200 seconds.

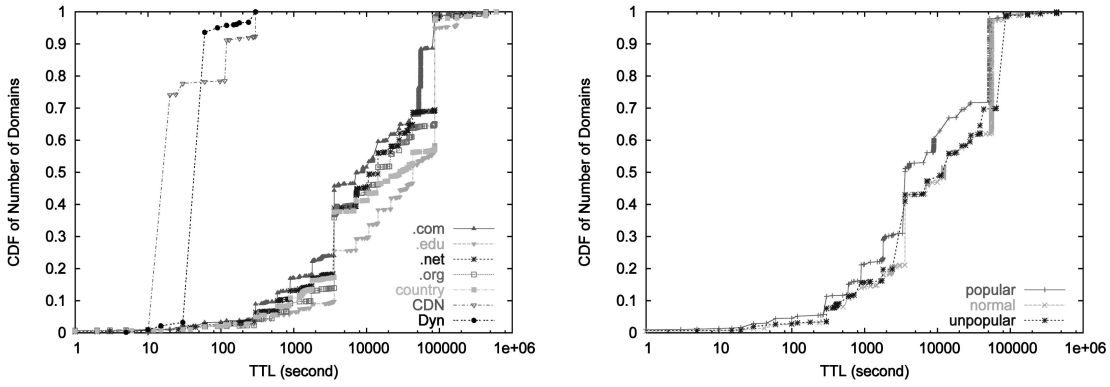

Fig. 2. TTL distributions. (a) All kinds of domain names. (b) `.com` domain names.

8 percent, much higher than the previous classes. The average lifetimes of DN2IP mappings are 200 and 750 seconds in classes 1 and 2, respectively. As shown in Fig. 3f, such frequent changes are mainly due to IP address rotation (for example, CDN's load balancing over multiple hosts), and most of the DN2IP mapping changes are logical ones. The more detailed change frequencies of CDN and Dyn domains are illustrated in Fig. 4.

As shown in Fig. 4, CDN domains have very high change frequencies: 10 percent with TTLs between 0 and 60 seconds and close to 70 percent with TTLs between 60 and 300 seconds. Two major CDN providers dominate the domains of the two ranges: Akamai with TTL 20 seconds and Speedera with TTL 120 seconds. The domain names served by Akamai have change frequencies around 10 percent, whereas those served by Speedera have change frequencies close to 100 percent. In contrast to CDN domains, the Dyn domains have low mapping change frequencies: 0.4 percent with TTL larger than or equal to 300 seconds and close to 0 with TTL less than 300 seconds. Compared with the actual change frequencies of CDN and Dyn domains, the corresponding TTL values are aggressively small, resulting in up to 10 and 25 times more DNS traffic than necessary. This redundant DNS traffic would be significantly reduced if a server-initiated notification service were used.

3.4.2 Change Frequency versus Domain Popularity

Within each TLD domain group, we investigate the relationship between DN2IP mapping change frequencies and domain popularities. The measurement results of `.com` domains are shown in Fig. 5. The results of other TLD domains are similar to those of `.com`. In classes 1 and 2 (most changes are logical changes), we observe that a more popular domain tends to have a higher change frequency than a less popular one. This is because a popular Web site is prone to use CDN or dynamic DNS techniques to improve its scalability and performance. By contrast, in classes 3, 4, and 5 (most changes are physical changes), there is no strong correlation between change frequencies and domain popularities. One explanation for this is that the occurrence of mapping changes in these classes is sporadic (irregular and random) over the entire domain space.

TABLE 1
Measurement Parameters

Class	TTL (s)	Resolution (s)	Duration	Num of Domains
1	[0,60)	20	1 day	803
2	[60,300)	60	3 days	934
3	[300,3600)	300	7 days	2020
4	[3600,86400)	3600	7 days	7217
5	[86400,∞)	86400	1 month	5307

4 DNS Cache Update Protocol (DNScup)

Basically, DNScup consists of three components, including mapping change detection module, state-tracking module, and update notification module. The mapping change detection module is straightforward to implement, since only the authoritative DNS name server has the privilege to change a DNS resource record. There are two ways for an authoritative DNS name server to change a DNS resource record: one is through manual reconfiguration and the other is through the DNS dynamic update command such as `nsupdate`.

The update notification module is in charge of propagating update notifications. To reduce communication overhead and latency, we choose UDP as the primary transport carrier for update propagation. Transmission Control Protocol (TCP) is used only when a firewall is set on the path from the authoritative DNS name server to a DNS cache. Also, we employ timers, retransmissions, and acknowledgment mechanisms to achieve reliable communication for cache updates. When a name server has sent a cache update notification message but has not yet received the corresponding acknowledgment, it retransmits the message three times before aborting cache update. The timer is doubled at each expiration.

The core of DNScup is the state-tracking module, which keeps track of the recent visitors, that is, the other DNS name servers who query and cache a local resource record recently. In the rest of the section, we detail our design on this module, and then, we present the whole working procedure of DNScup.

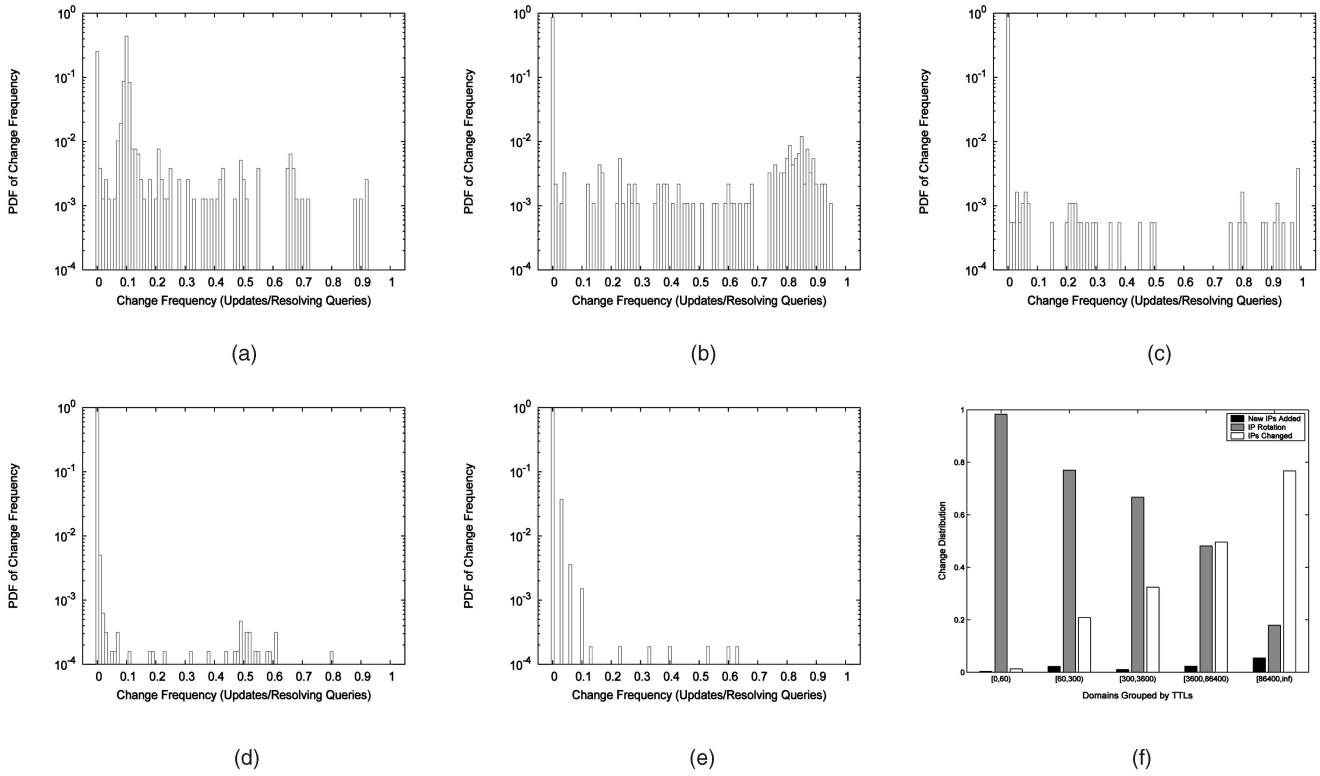

Fig. 3. The DN2IP mapping change for each class with different TTLs. (a) Class 1. (b) Class 2. (c) Class 3. (d) Class 4. (e) Class 5. (f) Change classifications.

4.1 Design Choices

In general, there are three different approaches to maintaining strong cache consistency: adaptive TTL, polling every time, and invalidation. The major challenge of using TTL to maintain cache consistency lies in the difficulty of setting an appropriate TTL value for a record. The adaptive TTL [7] adjusts the values of TTLs based on the prediction of record lifetime, which has been applied in Web caching consistency management [8]. The adaptive TTL may keep the staleness rate very low, but it cannot provide strong cache consistency. The polling-every-time approach is a simple strong consistency mechanism, which validates the freshness of the cached content at the arrival of every query. However, its fatal drawback lies in the poor scalability, as shown in [20], incurring more control messages, higher server workload, and longer response time. The invalidation approach relies on the server to notify the clients when an update happens, which is efficient when objects are rarely updated. Because most DNS resource records are changed at very low rates, server-driven invalidation is an appropriate approach to maintaining strong cache consistency among DNS name servers.

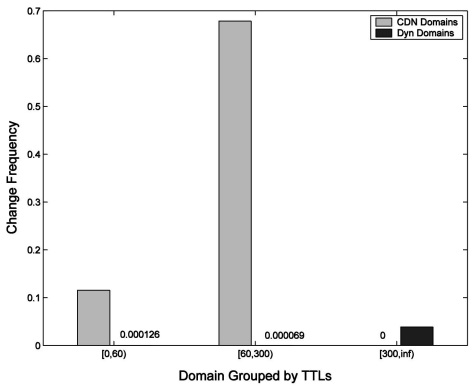

Fig. 4. CDN and Dyn domain change frequencies with different TTLs.

Lease [16] is a variant of the server-driven invalidation mechanism. A lease is a contract between a server and a client.[7] During a leased period, the client is promised to receive an invalidation notification if a leased object is changed. However, if the client does not have a lease, or the lease has already expired, then the client must validate a cached object upon the arrival of a query. The lease mechanism is thus a combination of polling and invalidation approaches. A critical question in applying a lease mechanism is how the appropriate length of a lease can be chosen. A long lease increases server storage and the number of invalidation messages, whereas a short lease increases the number of object requests and lease renewal messages.

A lease contract becomes valid either 1) upon the arrival of a new client request if the current lease expires or 2) by the automatic renewal of an expired-to-be lease. The resultant performance difference lies in the server storage overhead and the client-perceived latency. Because most DNS resource records do not change often, minimizing the consistency maintenance cost is more important than reducing latency. In our study, we always use the first approach to reducing the server storage overhead.

7. In the context of DNS, the client of an authoritative DNS name server is just a local DNS name server or another authoritative DNS name server that queries the authoritative DNS name server.

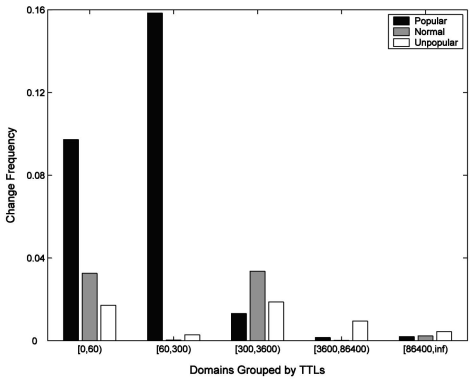

Fig. 5. The change frequencies of .com domains with different popularity and TTLs.

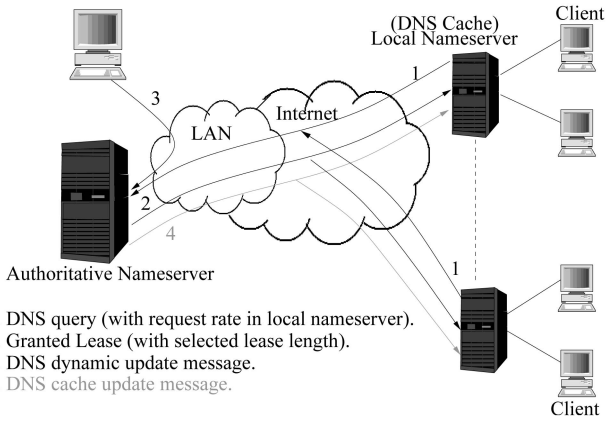

Fig. 6. DNScup update process.

To maintain strong cache consistency, DNScup requires the authoritative DNS name server to keep track of the recent visitors (that is, local DNS name servers) that access and cache a DNS resource record. *Recent* in this context implies that the cached records should have not yet expired in these local DNS name servers' caches. To make the presentation easier to understand, we refer to these local DNS name servers, that is, recent visitors, as DNS caches in the rest of the paper. We design a dynamic lease scheme to balance DNS name server storage requirements and DNS traffic between the authoritative DNS name server and the DNS caches.

Before detailing the design of dynamic lease, we sketch the cache update process as follows. Once the authoritative DNS name server has updated a DNS resource record either manually or via an internal dynamic update message, it retrieves the track file and gets all local DNS name servers that have queried this record whose leases have not yet expired (that is, DNS caches). The authoritative DNS name server then sends cache update messages to these DNS caches through UDP. The notified DNS caches will update their cached DNS resource records and acknowledge the authoritative DNS name server. The cache update process is shown as steps 3 and 4 in Fig. 6, in which steps 1 and 2 are the process of granting a lease to a DNS cache.

4.2 Lease Length Effectiveness

Lease storage overhead on the authoritative DNS name server is represented by the probability of the name server holding a lease for each DNS cache. Its upper bound is 1, indicating that the name server always keeps a lease for a DNS cache. The communication overhead is represented by the query rate between the name server and its DNS caches. If the lease length is much shorter than the lifetime of a resource record, then most messages will be renewal requests from DNS caches, and only very few invalidation and update messages may be observed. In the following analyses, since our practical algorithms always set the maximal lease length much smaller than the resource record lifetime, the communication overhead incurred by invalidation and update messages from the server can be ignored.

We assume that the query arrival rate from DNS caches for a DNS resource record follows a Poisson distribution, with an average arrival rate of λ. The rationale behind this assumption is twofold: 1) a DNS resolution precedes the beginning of a session communication and 2) Floyd and Paxson [29] have shown that the session-level (like FTP and Telnet) arrival rate still follows a Poisson distribution, although the packet arrival rate is non-Poisson.

Since the time interval is exponentially distributed, the time interval between two contiguous leases is equal to the average interval of two contiguous queries $\frac{1}{\lambda}$. Suppose that the authoritative DNS name server grants a fixed-length lease t at the arrival of a query. The expected probability for the name server to maintain the lease P is thus

$$P = t \Big/ \left(t + \frac{1}{\lambda}\right). \tag{1}$$

The lease renewal message rate is defined as lease renewal frequency. Since a lease is renewed at the interval of $t + \frac{1}{\lambda}$, the lease renewal message rate M is

$$M = \frac{1}{t + \frac{1}{\lambda}}. \tag{2}$$

Here, we assume that the arrival of queries for a DNS resource record follows a Poisson distribution. The trace-based validation of this assumption is presented in Section 5.1.

Theorem 1. *For a given resource record with a query rate λ, the ratio between the reduction of message rate and the increase of lease probability is a constant, which is equal to λ.*

Proof. Suppose the lease length is increased from t_1 to t_2. Given the query rate λ, the increase of lease probability on the name server is

$$\triangle P = t_2 \Big/ \left(t_2 + \frac{1}{\lambda}\right) - t_1 \Big/ \left(t_1 + \frac{1}{\lambda}\right) = \frac{\lambda t_2 - \lambda t_1}{(\lambda t_1 + 1)(\lambda t_2 + 1)}.$$

The reduction of message rate is

$$\triangle M = 1 \Big/ \left(t_1 + \frac{1}{\lambda}\right) - 1 \Big/ \left(t_2 + \frac{1}{\lambda}\right) = \lambda * \frac{\lambda t_2 - \lambda t_1}{(\lambda t_1 + 1)(\lambda t_2 + 1)}.$$

Thus, the ratio between the reduction of message rate and the increase of lease probability is equal to λ. □

From Theorem 1, we conclude that leases should be assigned to caches with higher query rates to maximize the message rate reduction. In Theorem 1, we ignore the cache

update messages in the calculation of the communication overhead, and the lease length is fixed. However, we have a similar result if both query messages and update messages are considered.

Theorem 2. *In lease-based consistency schemes, if the request rate λ_q and the update date λ_u of each resource record follow the Poisson distribution, then the ratio between the decrease of message rate and the increase of lease probability is a constant, which is equal to $\lambda_q - \lambda_u$.*

Proof. Suppose a lease is assigned to a cache with length t. The change of the query rate from the cache is

$$\triangle\lambda_q' = \lambda_q - 1 \Big/ \left(t + \frac{1}{\lambda_q}\right) = \lambda_q - \frac{\lambda_q}{t * \lambda_q + 1} = \lambda_q * t \Big/ \left(t + \frac{1}{\lambda_q}\right).$$

Also, the change of the update rate from the server is

$$\triangle\lambda_u' = \lambda_u * t \Big/ \left(t + \frac{1}{\lambda_q}\right).$$

The change of the lease probability in the server is

$$\triangle P = t \Big/ \left(t + \frac{1}{\lambda_q}\right).$$

Then, the ratio between the decrease of message rate and the increase of lease probability is

$$\frac{\triangle M}{\triangle P} = \lambda_q - \lambda_u.$$

□

Varying the lease length cannot have direct influence on the effectiveness of dynamic lease, since it is only decided by the query rate and the update rate. If a lease is assigned to a cache with the highest $\lambda_q - \lambda_u$, then the cost effectiveness is maximized.

4.3 Dynamic Lease Algorithms

Assuming that the overhead allowance (storage or communication) is predefined, we propose two dynamic lease algorithms: one minimizes the communication overhead, given a constraint on storage budget, and the other minimizes the storage overhead, given a constraint on communication traffic. Whether or not a lease is signed between the DNS name server and a DNS cache is based on the DNS cache's query rate, whereas the length of a lease is determined by the DN2IP mapping change rate at the DNS name server.

4.3.1 Storage-Constrained Dynamic Lease

We define the storage overhead allowance as the maximal number of valid leases that a name server can manage. Given the storage overhead allowance P_{max}, the storage-constrained dynamic lease algorithm minimizes the message exchanges for signing and keeping the leases at the name server.

Suppose that a total of n DNS resource records $R_i (i = 1, \ldots, n)$ are maintained on the authoritative DNS name server, each with the maximal lease length $L_i (i = 1, \ldots, n)$. Each record R_i is queried by m DNS caches $C_j (j = 1, \ldots, m)$, with the query rate λ_{ij}. We define M_{ij} and $P_{i,j}$ as the query rate and lease probability of record R_i by cache C_j, respectively. Our objective is to determine the appropriate lease length of every resource record the sum of M_{ij} for each DNS cache l_{ij} in order to minimize the overall communication overhead M_{all}. The decision should be made under the following constraints:

- For the record R_i and DNS cache C_j, the lease length l_{ij} should be within the range of 0 and L_i.
- The total storage consumption P_{all} should be less than the predefined storage overhead allowance P_{max}, the sum of P_{ij}.

Thus, the consistency maintenance problem can be defined as follows:

$$\text{minimize } M_{all} = \sum_{i=1}^{n}\sum_{j=1}^{m} M_{ij},$$

$$\text{subject to, for any } R_i \text{ and } C_{ij}, 0 \le l_{ij} \le L_i,$$

$$P_{all} = \sum_{i=1}^{n}\sum_{j=1}^{m} P_{ij} \le P_{max}.$$

A consistency maintenance scheme that fulfills the above constraint is a feasible solution. We refer to this kind of optimization as the storage-based lease problem (SLP). Since SLP is equivalent to a Knapsack problem, it is NP-complete, but its approximation solution can be found by utilizing the greedy algorithm.

If we have multiple records with different maximal lease lengths, then we need to sort the $\frac{\triangle M_{ij}}{\triangle P_{ij}}$, each of which is equal to λ_{ij} based on Theorems 1 and 2. Then, we grant the lease to the DNS cache with the highest query rate. It is clear that in order to reduce the communication overhead, we should grant the lease to the DNS cache with the highest query rate when the lease probability is close to the storage constraint.

If the name server always grants leases with their maximal lengths to the DNS caches selected as above until reaching the storage constraint, then we can guarantee that the total query rate covered by leases is maximal.

4.3.2 Communication-Constrained Dynamic Lease

Similarly, given the communication overhead allowance, we can design an algorithm that minimizes the storage overhead. It is also an *NP-complete* problem, and we employ the greedy algorithm to find the optimal solution. Different from the storage-constrained dynamic lease, at the beginning of the algorithm, all DNS caches related to each resource record are granted with the maximum-length leases. After that, we select the DNS cache with the smallest query rate and deprive its lease. The selection and deprivation continue until the communication allowance is satisfied. This way, we can guarantee that the number of leases maintained by the name server under the communication constraint is minimal.

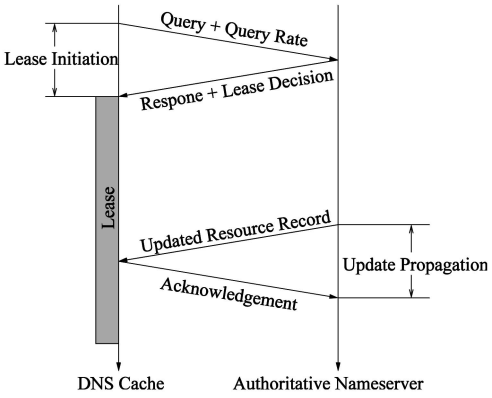

Fig. 7. DNScup procedure.

4.4 Working Procedure of DNScup

Although dynamic lease is an optimal solution in theory, it is not easy to deploy in practice. This is because the parameters of dynamic lease, such as query rates and update rates, are not readily available. For the practical deployment of dynamic lease, we design a simplified dynamic lease in DNScup. Fig. 7 illustrates the working procedure of DNScup. There are two major communication processes in this procedure: lease initiation and update propagation. The lease initiation is prompted by a DNS cache sending a query to the authoritative DNS name server. The query includes the local query rate on the cache, as well as its domain name. The authoritative DNS name server evaluates the query rate by certain metrics (for example, storage or communication constraint) to make a decision on granting a lease to the DNS cache or not. If a lease is granted to the DNS cache, then the authoritative DNS name server records the IP address of the DNS cache and the queried resource record. The decision on granting the lease is piggybacked to the DNS cache with the response of the query.

The authoritative DNS name server initiates the update propagation when one of its resource records has been changed. Notification messages, containing the updated resource record, are sent to the DNS caches with valid leases. All notified DNS caches need to acknowledge the receipt of the update message. Two auxiliary functions are important to DNScup:

- *Monitoring Query Rate at the DNS Cache.* In order to measure the query rate for a cached resource record, the DNS cache uses a reference counter (RC) to record the number of queries during a resource record's lease (or the TTL period if no lease is signed yet). After the cached resource record expires, the DNS cache bookkeeps the RC with the domain name by either writing into a specific file or keeping it at the cache for a certain period. When the resource record is queried again, the number of queries during the previous lease will be retrieved and forwarded to the authoritative DNS name server. Upon the arrival of the new response from the server, the counter will be reset. Fig. 8 illustrates the usage of the RC.
- *Granting Leases in the Authoritative DNS Name Server.* Using dynamic lease, DNScup sets a threshold on the cache query rate to determine whether or not the DNS name server should grant a lease for a DNS cache. The dynamic lease algorithm can be either evoked periodically to recompute the threshold or kept running to adjust it on the fly. In both designs, a query rate monitor maintains the statistics of all related cache query rates as the input for the dynamic lease algorithm. An initial value is set as the threshold, which is adjusted later according to the monitored query rates.

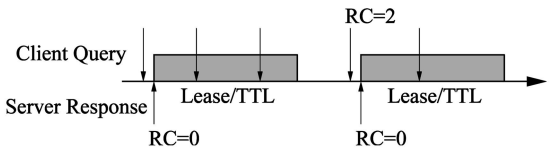

Fig. 8. DNScup cache reference counter.

5 Performance Evaluation

In this section, we evaluate the effectiveness of dynamic lease of DNScup via trace-driven simulation. Our DNS traces were collected in an academic environment, where three local DNS name servers provide DNS services for about 2,000 client machines. The 1-week trace collection is from 2 July 2003 to 9 July 2003. Based on the DNS traces, we simulate a scenario in which a number of clients are using three local DNS name servers. The local DNS name servers decide whether or not to grant a lease for one cached resource record based on its query rate.

Considering the client caching effect on query intervals, we assume that clients cache each resource record for 15 minutes, since this is the default setting in Mozilla. The query rate for each domain name is computed by analyzing the first-day traces. For the three categories of domain names (regular, CDN, and Dyn domains), we set different maximal lease length based on their DN2IP mapping change rates. The maximal length for a regular domain is set to six days, whereas those for DNS and Dyn domains are set to 200 and 6,000 seconds, respectively.

5.1 Poisson Distribution Validation

The DNS query behavior is related to the Web request access pattern. As most Web browsers cache DNS responses, the time interval between two continuous queries for one domain name likely follows the Poisson distribution. We use the mean of Coefficient of Variation (CV) to study the query interval distribution in our DNS traces. Fig. 9 shows the dynamics of the mean of CV with respect to the cache duration at the client side. With the increase of the client cache duration, as the mean of CV is closer to 1, the time intervals are more likely to follow a Poisson distribution. It is also noticeable that the 95 percent confidence interval of the mean is very small in all cases.

5.2 Experimental Results

We introduce two relative system metrics to evaluate the lease algorithms: storage percentage and query rate percentage. The storage percentage is defined as the ratio between the number of leases granted to querying DNS caches and the maximal number of leases that an authoritative DNS name server could grant. There are two extreme cases: 1) If the authoritative DNS name server grants a lease to each query, and all its resource records have valid leases all the time, then the storage percentage is 100 percent. 2) If no lease is granted to any query, then the

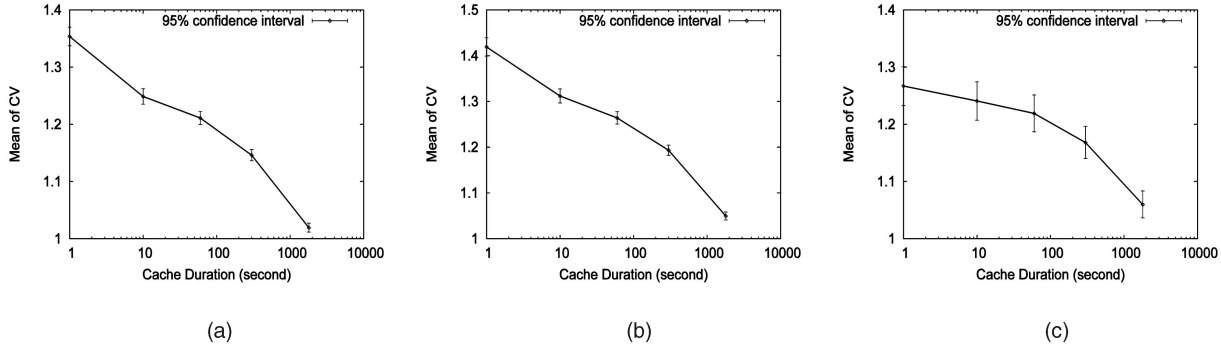

Fig. 9. The mean of CV of query interval in DNS traces. (a) Name server 1. (b) Name server 2. (c) Name server 3.

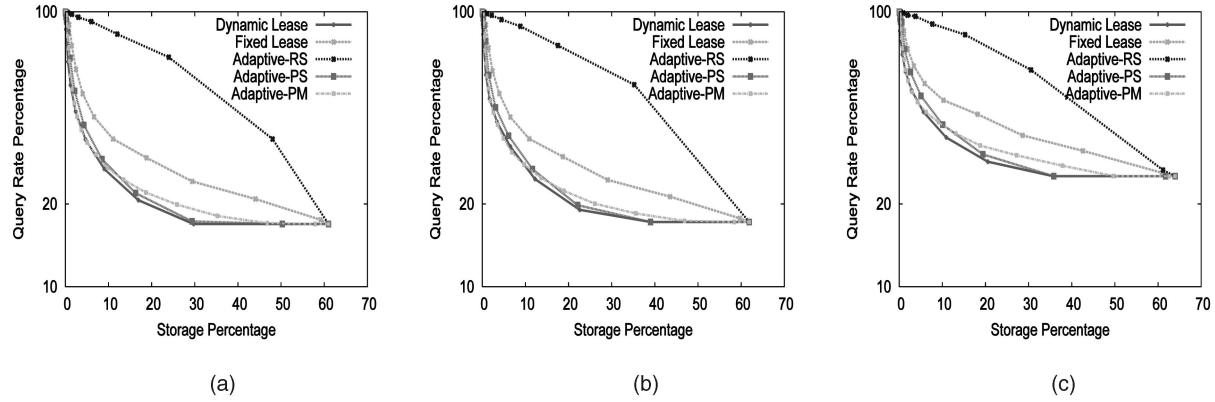

Fig. 10. Storage requirements for given query rates. (a) Name server 1. (b) Name server 2. (c) Name server 3.

storage percentage is 0. The query rate percentage is defined as the ratio between the query rate issued from a DNS cache and the maximal query rate that the DNS cache could generate. If no lease is granted, then the lease algorithm degrades to the polling scheme and generates the maximal query rate. Thus, the query rate percentage becomes 100 percent under this extreme scenario.

We compare the proposed dynamic lease scheme with the simple fixed-length lease scheme, which grants the same length lease to every incoming query, and the three adaptive lease schemes [13], including random-space-based adaptive lease (Adaptive-RS), popularity-space-based adaptive lease (Adaptive-PS), and popularity-message-based adaptive lease (Adaptive-PM). Adaptive-RS equally assigns lease by randomly selecting caches, Adaptive-PS takes the DNS record popularities into consideration and tunes the selection probability of a record proportional to its popularity, and Adaptive-PM adjusts the lease length proportionally to the corresponding DNS record popularity. Our simulation results clearly show that the performance of dynamic lease is superior to those of the Adaptive-RS and the fixed-length lease and is also better than those of Adaptive-PS and Adaptive-PM when the storage percentage is small. Figs. 10 and 11 illustrate the simulation results of regular domains based on the traces at three different DNS name servers. Note that the x-axis in Fig. 11 is in logarithmic scale. For CDN and Dyn domains, we have similar results. Due to space limitation, we do not present them here. In our trace-driven experiments, the storage percentage is bounded at 60 percent, since in practice, only a portion of resource records have valid leases at a time.

Dynamic lease is effective in reducing storage overhead. As shown in Fig. 10a, under the query rate percentage of 20 percent, the storage percentage of dynamic lease is 19 percent, whereas the storage percentages are 58 percent, 47 percent, 28 percent, and 21 percent for Adaptive-RS, fixed lease, Adaptive-PM, and Adaptive-PS, respectively. At the same time, dynamic lease is also effective in reducing communication overhead. As shown in Fig. 11a, under the storage percentage of 0.5 percent, the query rate percentage of dynamic lease is 77 percent, whereas for fixed leases, Adaptive-RS, Adaptive-PS, and Adaptive-PM, they are 100 percent, 99 percent, 91 percent, and 90 percent, respectively.

In another set of experiments, we evaluate the performance of different lease schemes under the given DNS record change rate at the server side. No lease will be granted to a cache if its query rate is lower than the change rate of a DNS record. We only present the results based on the DNS traces collected at name server 1, since we have similar results at other name servers. Fig. 12 shows the storage requirements of lease schemes under the three different record change rates: once per day, 10 times per day, and 20 times per day, respectively. Fig. 13 shows the corresponding query rate reductions. Dynamic lease is better than other schemes in most cases. The differences become more obvious with the increase of the change rates. Since the server-side notification messages increase the query rate, two schemes, the fixed lease and Adaptive-PM, have higher query rates if short lease length is used. It is noticeable that the fixed lease and Adaptive-PM are slightly better when their storage requirements are close to the maximal value, as shown in Figs. 12b and 12c.

In our experiments, due to the limitation of the trace length (seven days), the maximal length for regular domains is relatively small. Since regular domains seldom

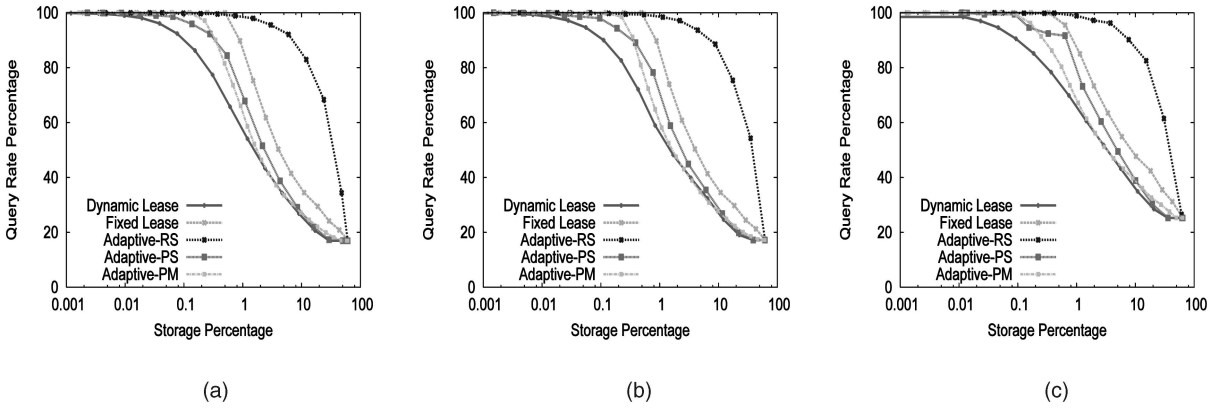

Fig. 11. Query rates for given storage requirements. (a) Name server 1. (b) Name server 2. (c) Name server 3.

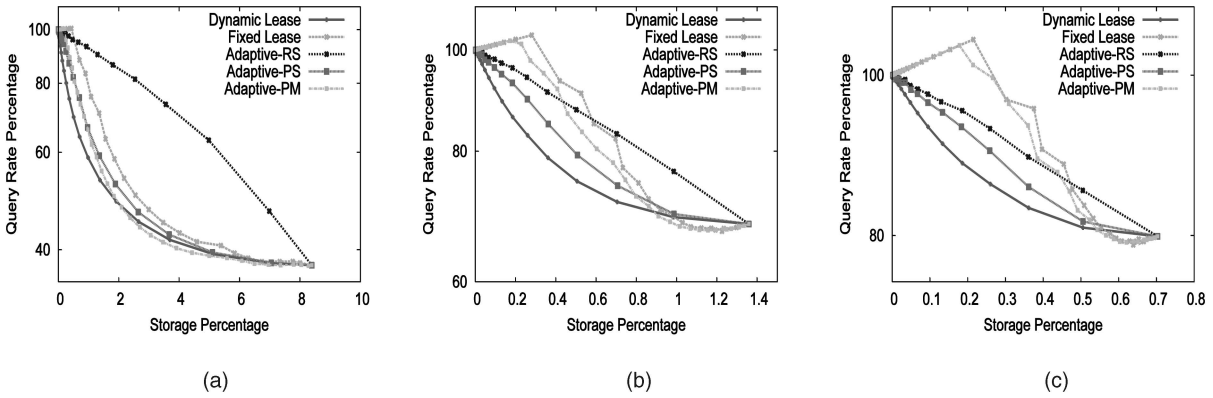

Fig. 12. Storage requirements for given query rates with different change rates. (a) Once per day. (b) Ten times per day. (c) Twenty times per day.

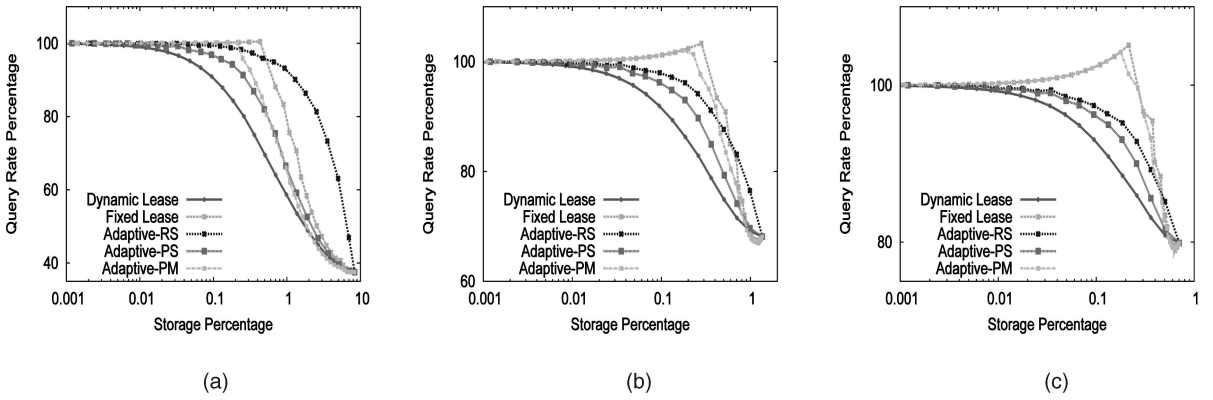

Fig. 13. Query rates for given storage requirements with different change rates. (a) Once per day. (b) Ten times per day. (c) Twenty times per day.

change their DN2IP mappings, we may use a much higher lease length to gain a better performance. Note that the lease selection in our experiment is done offline based on the trace analyses, and the lease length remains constant. In reality, a DNS cache may monitor the rates of cached records in the incoming queries. When it detects a significant change in query rates, the DNS cache will notify the authoritative DNS name server to renegotiate the current leases.

6 Prototype Implementation

We have built our DNScup prototype on top of BIND 9.2.3. In this section, we first present the extension on the DNS message format to support the DNScup mechanism. Then, we describe the structure of the DNScup prototype. Finally, we discuss the security issue related to DNScup. Note that DNScup only keeps cached resource records with the valid leases updated, and the rest of the cached resource records still rely on the TTL mechanism to refresh themselves.

6.1 Message Formats

In the header of DNS messages, a 1-bit field `QR` is used to specify whether it is a query (0) or a response (1). A 4-bit field operation code (`OPCODE`) is used to specify the type of the message. In current implementation of BIND, only types 0, 1, 2, 4, and 5 are used, and the rest are reserved for future use. To support DNScup, a new `OPCODE` 6 in the query/response headers is introduced for lease negotiation. Each DNS query includes the query rate originated from the local clients, and the query rate is expressed in a new 16-bit field

ID:	(new)
op:	CACHE–UPDATE(7)
Zone zcount:	1
Zone zname:	(zone name)
Zone zclass:	(zone class)
Zone ztype:	T_SOA

Fig. 14. Format of a CACHE–UPDATE message header.

recent RC (RRC), with the domain name being queried at the question section. The authoritative DNS name server uses OPCODE 6 in the response header to indicate that the lease information is included. If a lease is granted, then its duration is specified in a new 16-bit field lease length time (LLT) at the answer section.

In the BIND 9.2.3 implementation, a message with the OPCODE of 4 is used for the internal master-slave notification. In order to deal with the wide-area DNS cache update propagation, we define a new type of message called CACHE–UPDATE. This message has the same fields as those in the UPDATE message except for the "op" field in the message header, which is shown in Fig. 14.

6.2 Structure of DNScup Prototype

We have modified the prompt notification of the zone mechanism in the BIND 9.2.3 implementation. According to our design, three core components of DNScup have been added to BIND 9.2.3, including the detection module, the listening module, and the notification module. The detection module detects a DNS record change, the listening module monitors incoming DNS queries and updates the track file when necessary, and the notification module propagates DNS CACHE–UPDATE messages. The normal DNS operations remain intact. The interactions among all components are illustrated in Fig. 15.

For DNS resource records of the authoritative DNS name server, the named daemon creates a database file to keep track of the incoming DNS queries. Each tuple in this file consists of five fields, which are the source IP address, queried zone name, query type, query time, and lease length. When a DNS query comes in, the named first decides if a lease should be granted based on the query rate carried with the query. If yes, then a new tuple is added to the track file, and the corresponding response is sent back.

6.3 Secure DNScup

In our current implementation, we transmit DNS messages in plaintext for simplicity and efficiency. However, to protect DNS caches against poisoned CACHE–UPDATE messages originated from a compromised DNS name server, we need a secure communication channel for cache update. Fortunately, DNS Security Extensions (DNSSEC) [14] and the secure DNS Dynamic Update protocols [38] have been proposed. Coupled with the proposed secure DNS mechanisms, DNScup can achieve a secure cache update without much difficulty.

6.4 Experimental Results

We examine our prototype implementation in a testbed, which is a hierarchy of DNS name servers in a LAN environment. The testbed is shown in Fig. 16. By utilizing multiple virtual IP addresses, we run a master authoritative DNS name server and its two slaves on a machine. The root name server and two DNS caches are mimicked at three different machines, respectively. The machines used in our experiments are 1-GHz Pentium IIIs with 128-Mbyte RAM running RedHat Linux 9.1, connected by an Ethernet of 100 megabits per second (Mbps). From IRcache [4] proxy traces, we select 50 most popular domain names (46 if excluding "localhost" and three individual IP addresses). A total of 40 zones are constructed for the 46 domain names on the authoritative DNS name servers, with their glues recorded on the root server. The zone file data are collected through issuing necessary queries to the Internet.

The average lengths of different messages in DNScup are shown in Table 2. Compared with the existing TTL-based mechanism, the sizes of both query and response messages are increased due to the addition of new fields. However, they are still far below the limitation set by RFC 1035 [23]: A DNS message carried in UDP cannot exceed 512 bytes. Both cache update and its acknowledgment messages are small, having sizes similar to those of messages in the DNS Dynamic Update protocol [31].

In order to measure the processing overhead of DNS queries, we set two timers in Bind 9.2.3: one is right after

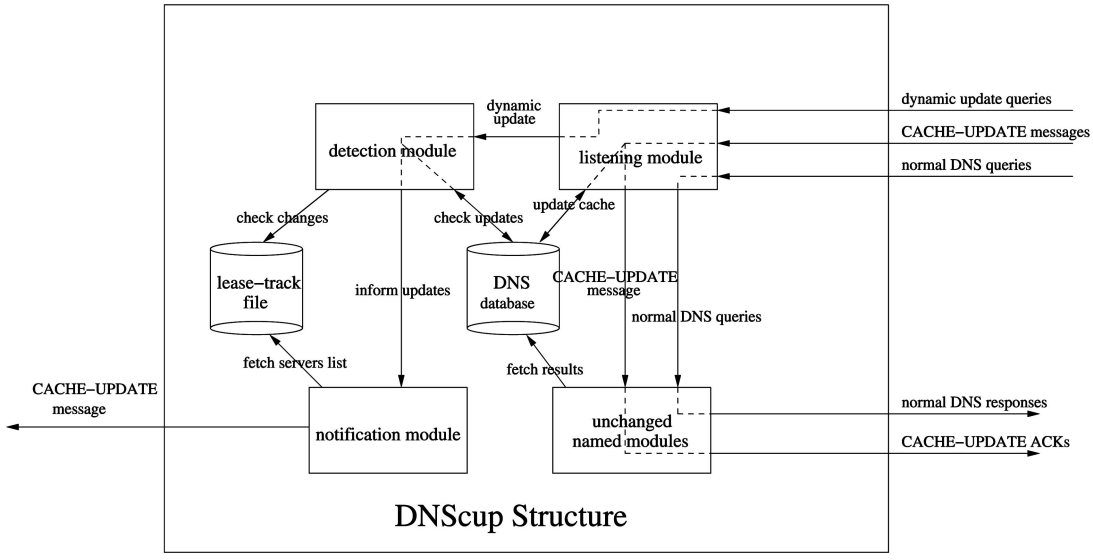

Fig. 15. Structure of DNScup prototype.

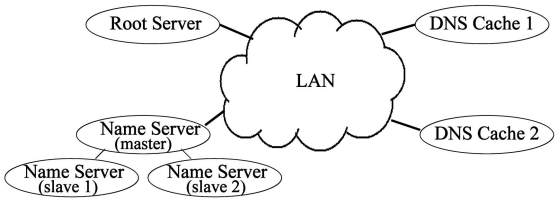

Fig. 16. DNScup implementation testbed.

receiving a query, and the other is right before the corresponding response is sent out. The two DNS caches repeat sending queries to the master authoritative DNS name server for the 46 collected domain names. After each round, we flush out their cached contents so that the authoritative DNS name server can continuously receive and process the queries. Fig. 17 shows the CDF of processing times of 5,000 continuous queries with and without DNScup support, respectively. Although DNScup needs to maintain the query rate statistics, the difference in computational overhead between TTL and DNScup is hardly noticeable.

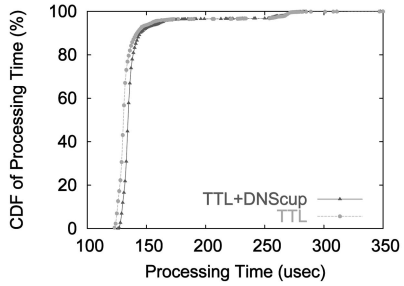

Fig. 17. DNS name server processing overhead: DNScup versus TTL.

7 Conclusion

In this paper, we have proposed *DNScup*, working as middleware to maintain strong consistency in DNS caches. To investigate the dynamics of DN2IP mapping changes, we have conducted a wide range of DNS measurements. Our major findings are summarized as follows:

- Although the physical mapping changes per Web domain name rarely happen, the probability of a physical change per minute within a class is close to 1.
- Compared with the frequencies of logical mapping changes, the values of the corresponding TTLs are much smaller, resulting in a large amount of redundant DNS traffic.
- The TTL value of a Web domain name is independent of its popularity, but its logical mapping change frequency is dependent on the popularity of the Web domain.

Based on our measurements, we conclude that maintaining strong cache consistency is essential to prevent potential losses of service availability. Furthermore, with strong cache consistency support, CDNs and other mechanisms can provide fine-grain load balance, quick responsiveness to network failure or flash crowd, and end-to-end mobility, without degrading the scalability and performance of DNS.

To keep track of the local DNS name servers whose clients need strong cache consistency for always-on Internet services, DNScup uses dynamic lease to reduce the storage overhead and communication overhead. Based on the DNS Dynamic Update protocol, we have built a DNScup prototype with minor modifications to the current DNS implementation. The major components of the DNScup prototype include the detection module, the listening module, the notification module, and the lease-track file. Our trace-driven simulation and prototype implementation demonstrate that DNScup achieves the strong cache consistency in DNS and significantly improves its availability, performance, and scalability.

TABLE 2
Average Message Overhead of DNScup

Type	DNScup (Bytes)	TTL (Bytes)	Increment
DNS query	40.8	36.8	10.9%
DNS response	217.8	203.7	6.9%
cache update	80.3	–	–
cache update ack	25.0	–	–

Acknowledgments

The authors thank Songkuk Kim for providing DNS traces of the Department of Electrical Engineering and Computer Science (EECS) at the University of Michigan, Phil Kearns for supporting the experimental environments, and William Bynum for his valuable comments.

References

[1] Content Delivery and Distribution Networks, http://www.web-caching.com/cdns.html, 2006.
[2] Dynamic DNS Provider List, http://www.technopagan.org/dynamic/, 2006.
[3] Internet Systems Consortium, http://www.isc.org, 2006.
[4] IRcache home page, http://www.ircache.net/, 2006.
[5] A. Broido, E. Nemeth, and K. Claffy, "Spectroscopy of DNS Update Traffic," *Proc. ACM Int'l Conf. Measurement and Modeling of Computer Systems (SIGMETRICS '03)*, pp. 320-321, June 2003.
[6] N. Brownlee, K. Claffy, and E. Nemeth, "DNS Root/gTLD Performance Measurements," *Proc. 15th Usenix Conf. Systems Administration (LISA '01)*, pp. 241-256, Dec. 2001.
[7] V. Cate, "Alex—A Global File System," *Proc. Usenix File System Workshop*, pp. 1-11, May 1992.
[8] A. Chankdunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell, "A Hierarchical Internet Object Cache," *Proc. Usenix Ann. Technical Conf.*, pp. 153-164, Jan. 1996.
[9] E. Cohen and H. Kaplan, "Proactive Caching of DNS Records: Addressing a Performance Bottleneck," *Proc. IEEE Symp. Applications and the Internet (SAINT '01)*, pp. 85-94, Jan. 2001.
[10] R. Cox, A. Muthitacharoen, and R. Morris, "Serving DNS Using a Peer-to-Peer Lookup Service," *Proc. First Int'l Workshop Peer-to-Peer Systems (IPTPS '02)*, pp. 155-165, Mar. 2002.
[11] C. Cranor, E. Gansner, B. Krishnamurthy, and O. Spatscheck, "Characterizing Large DNS Traces Using Graphs," *Proc. First ACM Internet Measurement Workshop (IMW '01)*, pp. 55-67, Nov. 2001.
[12] P. Danzig, K. Obraczka, and A. Kumar, "An Analysis of Wide-Area Name Server Traffic: A Study of the Internet Domain Name System," *Proc. ACM Ann. Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm. (SIGCOMM '92)*, pp. 281-292, Aug. 1992.
[13] V. Duvvuri, P. Shenoy, and R. Tewari, "Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web," *IEEE Trans. Knowledge and Data Eng.*, vol. 15, no. 5, pp. 1266-1276, Sept./Oct. 2003.
[14] D. Eastlake, *Domain Name System Security Extensions*, RFC 2535, Mar. 1999.

[15] J. Eisenberg and C. Partridge, "The Internet under Crisis Conditions: Learning from September 11," *ACM Computer Comm. Rev.*, vol. 33, no. 2, Apr. 2003.
[16] C. Gray and D. Cheriton, "Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency," *Proc. 12th ACM Symp. Operating Systems Principles (SOSP '89)*, pp. 202-210, Dec. 1989.
[17] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, "DNS Performance and the Effectiveness of Caching," *Proc. First ACM Internet Measurement Workshop (IMW '01)*, pp. 153-167, Oct. 2001.
[18] D. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigrahy, "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web," *Proc. 29th Ann. ACM Symp. Theory of Computing (STOC '97)*, pp. 654-663, May 1997.
[19] R. Liston, S. Srinivasan, and E. Zegura, "Diversity in DNS Performance Measures," *Proc. Second ACM Internet Measurement Workshop (IMW '02)*, pp. 19-31, Nov. 2002.
[20] C. Liu and P. Cao, "Maintaining Strong Cache Consistency in the World Wide Web," *IEEE Trans. Computers*, vol. 47, no. 4, pp. 445-457, Apr. 1998.
[21] M. Mikhailov and C. Wills, "Evaluating a New Approach to Strong Web Cache Consistency with Snapshots of Collected Content," *Proc. 12th Int'l World Wide Web Conf. (WWW '03)*, pp. 599-608, May 2003.
[22] P. Mockapetris, *Domain Names—Concepts and Facilities*, RFC 1034, Nov. 1987.
[23] P. Mockapetris, *Domain Names—Implementation and Specification*, RFC 1035, Nov. 1987.
[24] J. Pang, A. Akella, A. Shaikh, B. Krishnamurthy, and S. Seshan, "On the Responsiveness of DNS-Based Network Control," *Proc. ACM Internet Measurement Conf. (IMC '04)*, pp. 21-26, Oct. 2004.
[25] J. Pang, J. Hendricks, A. Akella, R. De Prisco, B. Maggs, and S. Seshan, "Availability, Usage and Deployment Characteristics of the Domain Name System," *Proc. Fourth ACM Conf. Internet Measurement (IMC '04)*, pp. 1-14, Oct. 2004.
[26] V. Pappas, P. Faltstrom, D. Massey, and L. Zhang, "Distributed DNS Troubleshooting," *Proc. ACM Ann. Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm. (SIGCOMM '04) Network Troubleshooting Workshop*, pp. 265-270, Aug. 2004.
[27] V. Pappas, Z. Xu, S. Lu, D. Massey, A. Terzes, and L. Zhang, "Impact of Configuration Errors on DNS Robustness," *Proc. ACM Ann. Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm. (SIGCOMM '04)*, pp. 319-330, Aug. 2004.
[28] K. Park, V.S. Pai, L. Peterson, and Z. Wang, "CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups," *Proc. Sixth Usenix Symp. Operating System Design and Implementation (OSDI '04)*, pp. 199-214, Dec. 2004.
[29] V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Trans. Networking*, vol. 3, no. 3, pp. 226-244, June 1995.
[30] V. Ramasubramanian and E. Sirer, "The Design and Implementation of a Next Generation Name Service for the Internet," *Proc. ACM Ann. Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm. (SIGCOMM '04)*, pp. 331-342, Aug. 2004.
[31] Y. Rekhter, S. Thomson, J. Bound, and P. Vixie, *Dynamic Updates in the Domain Name System*, RFC 2136, Apr. 1997.
[32] A. Shaikh, R. Tewari, and M. Agrawal, "On the Effectiveness of DNS-Based Server Selection," *Proc. IEEE INFOCOM '01*, pp. 1801-1810, Apr. 2001.
[33] A. Snoeren and H. Balakrishnan, "An End-to-End Approach to Host Mobility," *Proc. ACM MobiCom '00*, pp. 155-166, Aug. 2000.
[34] M. Walfish, H. Balakrishnan, and S. Shenker, "Untangling the Web from DNS," *Proc. First Usenix Symp. Networked Systems Design and Implementation (NSDI '04)*, pp. 225-238, Mar. 2004.
[35] B. Wellington, *Secure Domain Name System Dynamic Update*, RFC 3007, Nov. 2000.
[36] D. Wessels, M. Fomenkov, N. Brownlee, and K. Claffy, "Measurement and Laboratory Simulations of the Upper DNS Hierarchy," *Proc. Fifth Int'l Workshop Passive and Active Network Measurement (PAM '04)*, Apr. 2004.
[37] C. Wills, M. Mikhailov, and H. Shang, "Inferring Relative Popularity of Internet Applications by Actively Querying DNS Caches," *Proc. Third ACM Conf. Internet Measurement (IMC '03)*, pp. 78-90, Oct. 2003.
[38] C. Wills and H. Shang, "The Contribution of DNS Lookup Costs to Web Object Retrieval," Technical Report TR-00-12, Worcester Polytechnic Inst., July 2002.
[39] J. Yin, L. Alvisi, M. Dahlin, and A. Iyengar, "Engineering Web Cache Consistency," *ACM Trans. Internet Technologies*, vol. 2, no. 3, pp. 224-259, Aug. 2002.
[40] J. Yin, L. Alvisi, M. Dahlin, and C. Lin, "Hierarchical Cache Consistency in a WAN," *Proc. Second Usenix Symp. Internet Technologies and Systems (USITS '99)*, pp. 13-24, Oct. 1999.
[41] J. Yin, L. Alvisi, M. Dahlin, and C. Lin, "Volume Leases for Consistency in Large-Scale Systems," *IEEE Trans. Knowledge and Data Eng.*, vol. 11, no. 4, pp. 563-576, July/Aug. 1999.

Xin Chen received the BS degree from Xi'an Jiaotong University in 1996, the MS degree from the University of Science and Technology of China in 1999, and the PhD degree from the College of William and Mary in 2005, all in computer science. He is a member of the technical staff at Ask.com of IAC/Search and Media. His research interests include distributed systems, networking, and Internet computing.

Haining Wang (S'97-M'03) received the PhD degree in computer science and engineering from the University of Michigan, Ann Arbor, in 2003. He is an assistant professor of computer science at the College of William and Mary, Williamsburg, Virginia. His research interests lie in the area of networking, security, and distributed computing. He is particularly interested in network security and network quality of service (QoS) to support secure and service-differentiated internetworking. He is a member of the IEEE.

Shansi Ren (S'04) received the BS degree in computer science from the University of Science and Technology of China in 1999 and the MS degree in computer science from Bowling Green State University in 2001. He is currently a PhD student in the Department of Computer Science and Engineering, Ohio State University. His research interests include the Internet, networking, distributed systems, and operating systems. He is a student member of the IEEE.

Xiaodong Zhang (SM'94) received the BS degree in electrical engineering from Beijing Polytechnic University in 1982 and the PhD degree in computer science from the University of Colorado, Boulder, in 1989. He is the Robert M. Critchfield Professor in engineering and the chairman of the Department of Computer Science and Engineering, Ohio State University. He is the associate editor-in-chief of the *IEEE Transactions on Parallel and Distributed Systems*. He is also an associate editor of the *IEEE Transactions on Computers*, *IEEE Micro*, and the *Journal of Parallel and Distributed Computing*. He was the program director of the Advanced Computational Research at the US National Science Foundation from 2001 to 2004. His research interests include high-performance computing and distributed systems. He is a senior member of the IEEE.