Estimation of DNS Source and Cache Dynamics under Interval-Censored Age Sampling

Di Xiao, Xiaoyong Li, Daren B.H. Cline and Dmitri Loguinov*
Texas A&M University, College Station, TX, 77843, USA
di@cse.tamu.edu, xiaoyong@cse.tamu.edu, dcline@stat.tamu.edu, dmitri@cs.tamu.edu

***Abstract*—Since inception, DNS has used a TTL-based replication scheme that allows the source (i.e., an authoritative domain server) to control the frequency of record eviction from client caches. Existing studies of DNS predominantly focus on reducing query latency and source bandwidth, both of which are optimized by increasing the cache hit rate. However, this causes less-frequent contacts with the source and results in higher staleness of retrieved records. Given high data-churn rates at certain providers (e.g., dynamic DNS, CDNs) and importance of consistency to their clients, we propose that cache models include the probability of freshness as an integral performance measure. We derive this metric under general update/download processes and present a novel framework for measuring its value using remote observation (i.e., without access to the source or the cache). Besides freshness, our methods can estimate the inter-update distribution of DNS records, cache hit rate, distribution of TTL, and query arrival rate from other clients. Furthermore, these algorithms do not require any changes to the existing infrastructure/protocols.**

I. Introduction

To keep up with the explosive growth of Internet traffic, end-to-end caches continue to be an important part of many distributed systems, including search engines [7], [9], [32], wireless mobile networks [15], [16], P2P structures [34], [37], CDNs [6], [26], DNS [8], [22], [30], data warehouses [36], and various web applications [12], [14], [23], [35]. If ICN (Information-Centric Networking) [1] becomes successful, the Internet may eventually see cache deployment even at the network layer (i.e., at each router). Therefore, modeling cache performance is crucial to our current and future understanding of data churn at origin servers, Internet core, and customer facilities, including such metrics as bandwidth consumption, data consistency (i.e., freshness), and latency.

Depending on the eviction policy, cache operation can be classified into *capacity-based* and *TTL-based*. In the former case, arrival of new items into a full cache causes immediate removal of elements deemed unpopular (e.g., using LRU, LFU, CLOCK, FIFO, Random, and their variations [28]). In the latter case, which is our focus in this paper, items are evicted only when their TTL expires, meaning that cache size is considered infinite. This approach is more suitable in scenarios where object *staleness*, rather than storage limitations, are of primary concern and the duration an item remains cached must be controlled by the source based on the record's churn rate (i.e., frequency of modification).

*Supported by NSF grant CNS-1319984.

One particular area, where TTL-based caching has long been part of the standard, is DNS. With the wide adoption of dynamic DNS services and proliferation of CDN, many authoritative domains now frequently change IP addresses and other records to reflect content availability, load on the servers, and routing/geographic preferences, with more such activity expected in the future [11]. While the cache hit ratio h has been the sole metric of performance for many years [4], [5], [13], [17], [18], [19], [20], [22], the modern Internet requires a different modeling objective that would balance record freshness against cache efficiency. In this context, simply maximizing the hit rate, which essentially means setting the TTL to infinity, is not a meaningful pursuit. Instead, the system involves a tradeoff – higher hit rates h require items to stay longer in the cache, while better freshness f entails the opposite. Unfortunately, the interplay between these metrics has not received much attention in the past.

In order to keep staleness below target levels, one requires a methodology for estimating f within the confines of the current DNS protocol. This process, which we call *remote measurement*, must garner hidden information that is held at both the source and the cache, but without access to either. This makes sampling f a formidable challenge. While existing studies [2], [10], [27], [31], [39] provide a framework for estimating h and user-request rate λ, these methods cannot be easily extended to handle freshness. In fact, just presence of random TTLs renders these techniques impossible to use. We aim to fill this void below.

II. Related Work

The first direction in previous studies focuses on modeling TTL-based caching, where hit rate h has been the most common metric of interest. The majority of literature [2], [10], [31] assumes that clients send background queries using a Poisson process with some rate λ, which in certain cases approximates user requests rather well [11]. General renewal processes are considered in [17], [22]; however, they do not admit closed-form expressions for h and require numerical solutions to the renewal equation. The TTL is usually modeled as a constant [2], [22], [31], although h has been analyzed under a more general distribution in [4], [17], [22]. In all cases except [4], the time to download the object is assumed negligible compared to the TTL.

The second direction aims to remotely sample DNS resolvers and obtain an estimate of λ, assuming the cache

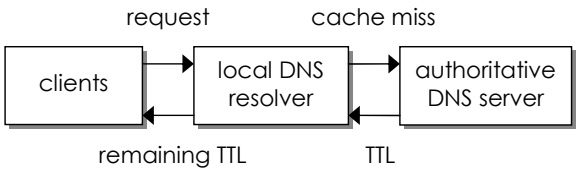


Fig. 1. Operation of DNS caching.

applies constant TTLs known to the observer. This usually requires either Poisson queries [2], [31] or hyper-exponential inter-request delays [27]. For arrivals with longer memory, it currently appears difficult to recover λ with high accuracy.

The last direction studies object staleness under networked replication. The interplay between general update/download processes is covered in [24] and a number of unbiased techniques for estimating the update distribution is proposed in [25]. We review some of their results below. In the context of DNS, freshness is taken into account only by [10], which analyzes the expected number of missed updates between a download and subsequent queries. If updates and user queries are both Poisson, [10] proposes a staleness-measurement technique that requires sources and their caches to exchange real-time information on the observed updates/requests. Because we do not assume cooperation, this is an orthogonal problem to the one studied here.

III. Performance Metrics of DNS Caches

A. System Operation and Notation

Assume a system with a single source, a single replica, and a number of clients that query the replica for a particular data item owned by the source. As shown in Fig. 1, a common networking scenario covered by this model is DNS, where the source is an authoritative server for some domain, the replica is a local DNS resolver, and clients are regular end-hosts. The replica operates based on the TTL provided by the source – each download k is accompanied by a parameter $T_k \geq 0$ that specifies how long the item must remain cached. In all replies, the cache provides to clients the residual delay before they must discard the object. This information is valuable not only during hierarchical caching (e.g., within the OS or browser), but also in remote measurement, as we discuss shortly.

In traditional DNS, the source decides T_k using some internal algorithm (e.g., based on the current load on the available servers, routing preferences, object volatility). This makes the TTL time-varying. Additionally, existing studies [3], [8], [30], [33] show that a large fraction of DNS resolvers violate the source-provided TTL. Reasons for such behavior include attempts to impede cross-site scripting attacks [21]and reduction of cache inconsistency through TTL adaptation [10]. Therefore, the effective TTL may be highly variable not only due to source decisions, but also cache policies. To cover such cases, we model $\{T_k\}$ as iid random variables with some distribution $F_T(x)$.

At the source, suppose the object sustains the i-th update at time u_i and process $N_U(t) = \max(i : u_i \leq t)$ counts

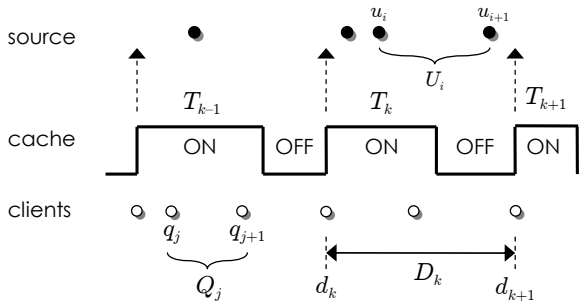


Fig. 2. Process notation.

the number of such events in $[0, t]$. We assume that N_U is *age-measurable*, which is the weakest set of conditions under which various sample-path averages related to staleness are convergent [24]. Age-measurability is a generalization of renewal processes that allows non-stationary dynamics as long as the empirical distribution of cycle lengths converges to a deterministic function. Similarly, denote by q_j the time of client query $j \geq 1$ and assume $N_Q(t) = \max(j : q_j \leq t)$ is a renewal process that has no point at zero, i.e., $q_1 \sim F_Q(x)$.

Following Fig. 2, suppose inter-update durations are $U_i = u_{i+1} - u_i$ and inter-query intervals are $Q_j = q_{j+1} - q_j$. It then follows [24] that the collection of variables $\{U_i\}_{i=1}^{\infty}$ has some distribution $F_U(x)$ and $\{Q_j\}_{j=1}^{\infty}$ has another distribution $F_Q(x)$. A combination of N_Q and $F_T(x)$ uniquely defines the download process N_D between the source and the cache. Assuming its k-th point is d_k, we get that $N_D(t) = \max(k : d_k \leq t)$. In Fig. 2, $D_k = d_{k+1} - d_k$ represents inter-download gaps of N_D, dashed arrows are synchronization instances with the source, and the bold-line ON/OFF process corresponds to object presence in the cache. For all cycles lengths, we assume their mean is positive and finite.

B. Hit Rate

Define $T \sim F_T(x)$ to be a generic variable with the same distribution as the TTL. Then, hit rate [4], [17], [22]

$$h = \frac{E[N_Q(T)]}{1 + E[N_Q(T)]} \tag{1}$$

is the fraction of queries that arrive during ON periods in Fig. 2. The numerator of (1) is the expected number of client queries in $[0, T]$, i.e., after the object has been cached, and the denominator adds 1 to account for the first packet that started the ON cycle.

Notice that $E[N_Q(T)]$ increases for stochastically larger T or stochastically smaller Q, which implies that h in (1) does too. Intuitively, this makes sense – longer ON periods in Fig. 2 or faster query rates yield better hit rates. The opposite holds when the conditions are reversed.

C. Freshness

As the DNS record keeps changing, cache responses may become outdated compared to the current state at the source. For the example in Fig. 2, the item provided to clients at time

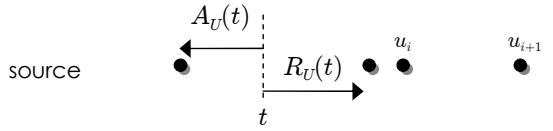


Fig. 3. Age and residual of the update process.

q_j is fresh, while that at q_{j+1} is stale. Let the *update residual* at time t be the distance to the next update point

$$R_U(t) := u_{N_U(t)+1} - t, \tag{2}$$

and the *update age* be the delay to the previous point

$$A_U(t) := t - u_{N_U(t)}, \tag{3}$$

which are illustrated in Fig. 3. If N_U is age-measurable, the distributions of variables (2)-(3) sampled at uniform points t converges to the equilibrium CDF [24]

$$G_U(x) := \frac{1}{E[U]} \int_0^x (1 - F_U(y))dy. \tag{4}$$

Besides variables $A_U, R_U \sim G_U(x)$ needed for various purposes throughout the paper, we often utilize distributions in the form of (4) for other processes. In these cases, a subscript indicates the underlying random variable that governs the cycles of the process, e.g., $G_T(x)$ refers to the equilibrium distribution of $F_T(x)$, while $A_T, R_T \sim G_T(x)$ are the corresponding age and residual random variables.

Suppose *freshness* f is the long-term fraction of queries that return an object that is consistent with that at the source. It is common to use the term *staleness* [29], [40] to refer to $1 - f$. For either quantity to be computable using the properties of the underlying processes (i.e., without access to both source and cache logs), it is necessary that processes (N_U, N_Q) be *age-independent* [24], which means that their cycle lengths not enter a permanent phase-lock as $t \to \infty$. This condition can be satisfied by requiring that one of $F_U(x), F_Q(x)$ be non-lattice (i.e., not defined on rational numbers). For the remainder of the paper, we assume that all defined processes are pairwise age-independent.

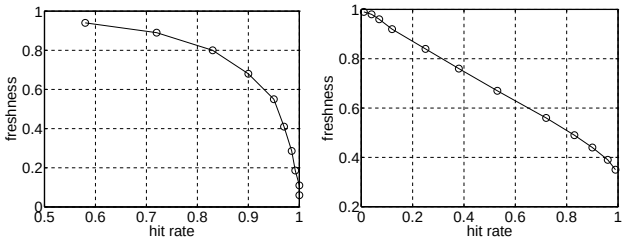
Theorem 1: Cache replies are fresh with probability

$$f = \frac{1 + E[N_Q(\min(R_U, T))]}{1 + E[N_Q(T)]}. \tag{5}$$

D. Freshness-Efficiency Tradeoff

The effect of T and Q on freshness is quite a bit more complex than on the hit rate. Simply making T stochastically larger (or Q smaller) is not enough to predict the change in f in every circumstance. The problem is that (5), unlike (1), has different expressions in the numerator and denominator. In one special case, however, we can establish the asymptotics of how the TTL and query rates affect staleness.

Theorem 2: If $E[T_n] \to \infty$ and $F_Q(x)$ is fixed, freshness under T_n converges to 0. If $Q_n = Q/n$, where $Q \sim F_Q(x)$, and $F_T(x)$ is fixed, freshness under Q_n converges from above to $E[\min(R_U, T)]/E[T]$ as $n \to \infty$.



(a) changing $E[T]$ ($\lambda = 1$) (b) changing λ ($E[T] = 40$)

Fig. 4. Cache tradeoffs (Pareto U, T, Q with $E[U] = 20$ sec).

Intuitively, this result holds because larger TTL yields less-frequent downloads from the source and larger λ produces more stale queries per synchronization event. With the exception of esoteric counter-examples, this generally means that freshness and hit rate are tradeoffs of each other. To illustrate this point, Fig. 4(a) varies $E[T]$ and plots f as a function of h, where the two models come from (1) and (5). As predicted, longer TTLs drive $h \to 1$ and $f \to 0$, reaffirming the tradeoff. A similar picture emerges in Fig. 4(b) as λ changes, except here $f \to E[\min(R_U, T)]/E[T] = 1/3$ as hit rate $h \to 1$.

E. Large User Base

When the local resolver sustains queries from a large population of users, it might be sensible to model N_Q as a Poisson process, which is a common assumption in the field [2], [10], [31]. Under exponential Q, inter-download delay D becomes a convolution of ON/OFF cycle lengths, i.e., $D = T + Q$. Furthermore, letting $\lambda = 1/E[Q]$ be the request-arrival rate, (1) transforms into

$$h = \frac{\lambda E[T]}{1 + \lambda E[T]} \tag{6}$$

and (5) simplifies to

$$f = \frac{1 + \lambda E[\min(R_U, T)]}{1 + \lambda E[T]}. \tag{7}$$

The next result establishes a more useful representation of freshness by making it an explicit function of h.

Theorem 3: Under Poisson queries, freshness is given by

$$f = 1 - h + hp, \tag{8}$$

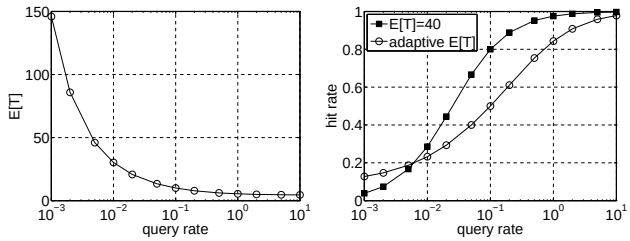
where $p := P(R_T < R_U)$.

Re-writing (8) as $f = 1 - h(1-p)$, it is clear that increasing λ, which also increases h, causes freshness to go down. This is consistent with our earlier conclusions about (5).

IV. Passive Measurement

A. Preliminaries

Assume that the local resolver needs to estimate the long-term freshness f of a given object in its cache. The reasons for this objective could be numerous (e.g., performance monitoring), but consider a more concrete example. Suppose the source provides $T_k = 40$ seconds for all k, but this leads to 10% freshness for the specific user process N_Q at this



(a) adaptive TTL (b) resulting hit rate

Fig. 5. Achieving 90% freshness (Pareto U, T, Q with $E[U] = 20$ sec).

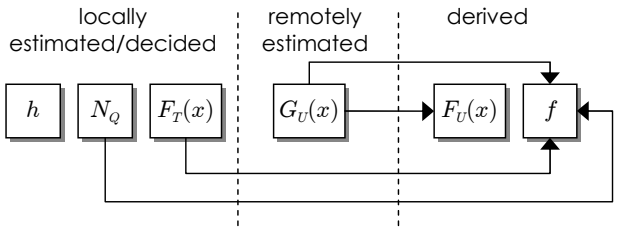


Fig. 6. Model roadmap for passive monitoring.

cache. Assuming that the replica has enough spare bandwidth to sustain more frequent downloads, it may decide to lower T_k (i.e., preemptively evict records that are likely stale) in order to achieve a certain freshness guarantee to its clients, while still maintaining a reasonable level of latency. Since process N_Q is localized to each cache, the source has no ability to optimize T_k simultaneously for all of its replicas.

We call a cache *adaptive* (i.e., staleness-aware) if it selects $G_T(x)$ such that freshness f in (5) is maintained at some desired threshold. For the scenario in Fig. 4, deterministic T, and 90% freshness, Fig. 5(a) shows the relationship between the query rate λ and TTL. Observe that more-frequent client requests mandate a sharp decrease in $E[T]$, while less-loaded conditions do the opposite. The corresponding hit rate is provided in Fig. 5(b) in comparison to the default 40-second eviction delay. If the object is requested more often than once every 2 minutes, the adaptive strategy exhibits lower hit rates and requires more bandwidth, but provides fresher records. The situation is reversed when the record is less popular.

As shown in Fig. 6, parameters h, N_Q, and $F_T(x)$ can be locally determined by the resolver. On the other hand, estimation of f at the replica, which we call *passive sampling*, requires an inference process that obtains the residual update distribution $G_U(x)$. The rest of the section focuses on this.

B. Sampling Update Age

Recall from Fig. 2 that the cache contacts the source at points $\{d_k\}$, which form some download process N_D. From our assumptions, (N_U, N_D) are age-independent, which allows application of the various techniques in blind sampling [25]. There are three variations of methods based on the capabilities of the source. In the first case, the source explicitly provides the update age $A_U(d_k)$ from Fig. 3 during each download k, e.g., using protocol fields such as HTTP

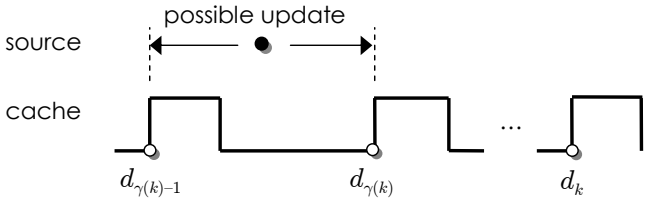


Fig. 7. Bounding the update age in passive measurement.

headers. For this scenario, [25] develops a method called M_2 that is asymptotically unbiased and quickly convergent. Unfortunately, this scenario does not apply to DNS.

As a result, the cache has to infer presence of updates by comparing adjacent versions of downloaded records. Define a binary process Δ_k to be 1 if the object is detected as modified during download k. When inter-download delays D_k are all constant, the second class of methods in [25] uses $\{\Delta_k\}_{k=1}^{\infty}$ to estimate $G_U(x)$ with asymptotically consistency. However, again, this formulation does not work for our problem since D_k is a complex random variable that subsumes T_k and the preceding OFF period in Fig. 2, both of which are random.

This leaves us with the third class of methods in [25], which consists of a single technique M_6 that works with $\{\Delta_k\}_{k=1}^{\infty}$ and random D_k. It has two drawbacks – quadratic computation time in the number of downloads n and lower accuracy compared to the remaining methods in [25]. Therefore, our first goal is to improve this technique in both aspects.

C. Non-Parametric EM

Blind-sampling methods work by estimating the unknown update age $A_U(d_k)$ in download points. Instead of rounding this to some value, as done in [24], our novel approach is to provide the estimator with *interval-censored* values, i.e., upper/lower bounds on $A_U(d_k)$. Define $\gamma(k) = \max(i \le k : \Delta_i = 1)$ to be the last download in $[0, d_k]$ that detected a modification. Consider Fig. 7 and suppose the current time is d_k. Then, the latest update at the source is always confined to the interval $(d_{\gamma(k)-1}, d_{\gamma(k)}]$. This immediately yields

$$d_k - d_{\gamma(k)} \le A_U(d_k) \le d_k - d_{\gamma(k)-1}. \tag{9}$$

Suppose the lower bound in (9) is L_k and the upper is R_k. Then, our technique, which we call Chameleon[1], collects a sequence of pairs $\{(L_k, R_k)\}_{k=1}^{n}$ and feeds them into a non-parametric Expectation Maximization (EM) estimator. Our model draws inspiration from Turnbull's method for interval censoring [38]. First, we quantize the bounds to be a multiple of some bin size, where L_k is rounded *down* and R_k *up*. We then combine upper/lower bounds into a single vector, sort the result ascending, eliminate duplicates, and obtain a new set $(x_1, \ldots, x_m)$, where $m \le 2n$. From this, we can form non-overlapping bins $B_i = [x_{i-1}, x_i)$, where $x_0 = 0$.

Let $p_i(t)$ be the estimated probability that the target random variable $A_U \sim G_U(x)$ belongs to bin B_i during step t of

[1]Chameleons belong to the family of ambush predators, which blend into the environment and passively monitor their surroundings.

the iteration, where $p_i(0) = 1/m$ for all i. Now define a_{ik} to be an indicator variable that B_i is entirely contained in the discretized interval from download k. Using non-quantized bounds, this can be expressed as

$$a_{ik} = \begin{cases} 1 & B_i \cap [L_k, R_k] \neq \emptyset \\ 0 & \text{otherwise} \end{cases}. \tag{10}$$

Next, define $V_k(t)$ to be the probability that $A_U \sim G_U(x)$ belongs to $[L_k, R_k]$ during iteration t

$$V_k(t) = \sum_{i=1}^{m} a_{ik} p_i(t). \tag{11}$$

Each probability is then refined using recurrence

$$p_i(t+1) = \frac{p_i(t)}{n} \sum_{k=1}^{n} \frac{a_{ik}}{V_k(t)} \tag{12}$$

until the stopping criterion is satisfied, i.e., $||p(t+1) - p(t)|| < \epsilon$, where $\epsilon > 0$ is a constant, and $p(t) = (p_1(t), \ldots, p_m(t))$. Note that this process is asymptotically accurate [38], i.e., (12) converges to $G_U(x)$ as $n \to \infty$.

D. Implementation

A naive version of (10)-(12) calculates all mn values a_{ik} and keeps them in RAM, which is highly inefficient. In the worst case (i.e., $m = 2n$), this computation is quadratic in both space/time. Instead, we offer Algorithm 1 whose per-iteration CPU complexity $O(n)$ and storage cost $O(m)$ are optimal. Prior to calling FastEM, assume the program has already determined bin boundaries $(x_1, \ldots, x_m)$ and mapped each pair (L_k, R_k) to the appropriate bin using an array of structs, i.e., `intervals[k].L` and `intervals[k].R`. Note that duplicate tuples (L_k, R_k) are compressed such that `intervals[k].count` is the corresponding frequency.

After initialization, Algorithm 1 computes the CDF of A_U using a prefix sum of the PMF array p (Line 7). Padding with a front zero is needed to properly compute V_k in Line 9. We then use a temporary array Z to accumulate all weights $1/V_k$ that will be distributed into the relevant bins after the loop is over. Specifically, $Z[i]$ stores increments that must be applied to the PMF in position $[i, m]$. This requires adding $1/V_k$ at the left boundary of the interval (Line 10) and subtracting it at the right boundary (Line 11). The second loop in Lines 13-15 computes a prefix sum of Z and stores the result, normalized by $p_i(t)/n$, into the same vector.

With compression of duplicate intervals, the number of unique boundaries supplied to Algorithm 1 is upper-bounded by $\min(m(m-1)/2, n)$. If $m \ll n$ and the runtime is dominated by iteration in Lines 3-4, rather than the initial $n \log n$ sort, Chameleon can exhibit sublinear scaling in a limited range of n. We show such an example below.

E. Concave EM

Note that $G_U(x)$ in (4) has a monotonically decreasing density $g_U(x) = G'_U(x) \sim 1 - F_U(x)$. As a result, $G_U(x)$ is a concave function. This is an important property that must be preserved by the estimator, especially if recovery of $F_U(x) = 1 - g_U(x)/g_U(0)$ is needed from $G_U(x)$. Since the density is commonly estimated by scaling and smoothing the PMF, a proper solution would guarantee $p_i(t) \geq p_{i+1}(t)$. Unfortunately, (10)-(12) fail to do so. Furthermore, none of the previous literature has considered this issue before.

Algorithm 1: Chameleon's implementation of (12).

```
1  Function FastEM(intervals, m)
2      p = (1/m, 1/m, ..., 1/m);                    ◁ initial guess
3      while not converged do
4          p = OneIteration (intervals, p, m)
5  Function OneIteration(intervals, p, m)
6      Z = zeros (1, m+1);                          ◁ temp storage
7      cdf = prefix_sum ([0 p]);                    ◁ CDF padded with 0 at front
8      for k = 1 to intervals.size() do
9          Vk = cdf [intervals [k].R] − cdf [intervals [k].L];
10         Z [intervals [k].L] += intervals [k].count / Vk;
11         Z [intervals [k].R] −= intervals [k].count / Vk;
12     psum = 0;                                    ◁ prefix sum of Z
13     for i = 1 to m do
14         psum += Z [i];                           ◁ total weight from all intervals
15         p [i] *= psum / n ;                      ◁ normalize and save
16     return p;
```

To overcome the setback, we offer a new EM algorithm that ensures proper recovery of residual (i.e., concave) CDFs. Define $\delta_i = x_i - x_{i-1}$ to be the length of the i-th bin and let $g_i(t) = p_i(t)/\delta_i$ be the corresponding estimate of density, where $g_{m+1} = 0$. Suppose

$$q_i(t) = x_i(g_i(t) - g_{i+1}(t)) \tag{13}$$

models the negative derivative of $g_i(t)$, normalized such that $\sum_{i=1}^{m} q_i(t) = 1$. Observe that if the estimated density $g_i(t)$ is a decreasing function of i, then $q_i(t) \geq 0$ for all i. The opposite holds as well since

$$g_i(t) = \sum_{j=i}^{m} \frac{q_j(t)}{x_j}. \tag{14}$$

We next create an EM algorithm for $q_i(t)$ such that $q_i(t) \geq 0$ is preserved during each iteration.

Theorem 4: A concave EM estimator for interval-censored data is given by

$$q_i(t+1) = \frac{q_i(t)}{n x_i} \sum_{j=1}^{i} \delta_j \sum_{k=1}^{n} \frac{a_{jk}}{V_k(t)}. \tag{15}$$

Note that (15) uses variables from (10)-(11), which requires further elaboration. Before the first iteration, we set $p_i(0) = \delta_i / x_m$, which ensures a monotonic initial density, and convert $p(0)$ into vector $q(0)$ using (13). This requires one pass over all m bins. Then, we represent (15) as

$$q_i(t+1) = \frac{q_i(t)}{n x_i} \sum_{j=1}^{i} \delta_j W_j(t), \tag{16}$$

where

$$W_j(t) = \sum_{k=1}^{n} \frac{a_{jk}}{V_k(t)}. \tag{17}$$

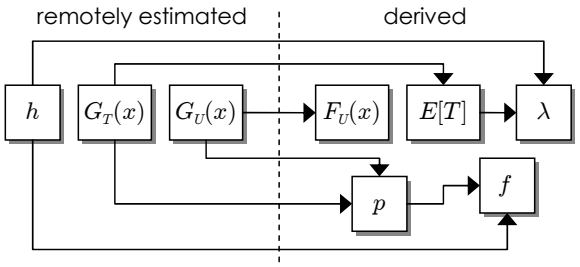


Fig. 8. Model roadmap for active monitoring.

Note that the entire set $\{W_j(t)\}_{j=1}^m$ can be computed by calling a slightly modified function OneIteration in Algorithm 1, where Line 15 does not have the $p_i(t)/n$ multiplier. Once all $\{W_j(t)\}$ are available, an extra prefix sum over m bins produces (16). Finally, $q(t+1)$ is converted back to $p(t+1)$ using (14), which requires another scan over m bins. In the end, per-iteration cost of our concave EM differs from that of Algorithm 1 by $2m$, which is negligible in practice.

V. Active Measurement

A. Preliminaries

We now face the issue of estimating f from a vantage point outside the cache, which we call *active sampling*. This problem may be of interest to sources (e.g., CDN companies), where the goal is to measure what types of freshness their TTL algorithms produce at certain customer networks (e.g., Comcast, Verizon). Additionally, caches may be remotely monitored by campus networks administrators and researchers, who do not have access to the logs, to characterize replication efficiency and diagnose potential problems with bandwidth consumption, staleness, and performance of deployed algorithms.

Due to the lacking cooperation from the cache, computation of f is likely intractable unless N_Q is Poisson, which we assume in the rest of the paper. To calculate (8), we need to estimate $p = P(R_T < R_U)$ and hit rate h, the former of which requires the residual TTL distribution $G_T(x)$ and the residual update distribution $G_U(x)$. This is illustrated in Fig. 8, where $E[T]$, λ, and $F_U(x)$ can be obtained with no extra overhead once the three main parameters are known.

Note that sampling must be performed without intrusion into the ON/OFF process of the cache, which would skew the result. We therefore assume that the resolver accepts *iterative* requests, to which it responds with an error, instead of contacting the source, if the record is not currently cached. For cached objects, the resolver returns their remaining TTL, which equals $R_T(t)$ at time t using our earlier notation.

B. Constant TTL

Previous measurement literature [2], [27], [31] is limited to estimating λ and h under constant T. However, even in our setting with an extra distribution $G_U(x)$, the problem becomes trivial under this condition. Assume T is a fixed value that is known to the observer from a-priori contacts with the source or other means. Checking the cache every T time units ensures that every ON period receives at least one sample point, which allows recovery of the corresponding download instance $d_k = t_k + R_T(t_k) - T$ for all query times t_k. The problem of estimating $G_U(x)$ then becomes identical to that in passive sampling, where Chameleon provides an excellent supporting platform. Furthermore, knowledge of the starting and ending point of each ON interval allows access to all OFF durations, i.e., $d_k - (d_{k-1} + T)$, whose average tends to $1/\lambda$. Finally, the hit rate follows from (6).

C. Random TTL

The issue is significantly more complex when the TTL varies between cycles, which is our assumption from this point forward. The challenge stems from the observer's uncertainty about the location of download point d_k since only the cache knows this information. This precludes direct measurement of the OFF duration or application of Chameleon. We now formulate our framework for solving these issues.

Suppose the observer is a special client that sends only iterative queries with an objective to determine f with asymptotic accuracy as the observation window tends to infinity. Assume that these requests are issued at points $\{s_k\}$ such that inter-sample delays $S_k = s_{k+1} - s_k$ have some distribution $F_S(x)$ and $N_S(t) = \max(k : s_k \le t)$ is an age-measurable process. Given age-independence between (N_D, N_S), the ASTA (Arrivals See Time Averages) property of the constructed system [24] ensures that the fraction of sample points that return a cached object tends to $E[T]/E[T+Q]$ as $n \to \infty$. Since this value directly equals h, i.e., the first unknown parameter in Fig. 8, the observer can measure the hit rate without any additional bandwidth or computational overhead.

Leveraging [24], the observed TTL residuals $\{R_T(s_k)\}_{k=1}^n$ converge in distribution to $R_T \sim G_T(x)$ as $n \to \infty$. With iid $\{S_k\}$ and Poisson N_Q, reconstructing $G_T(x)$ should be possible using arbitrary distributions $F_S(x)$, including lattice cases (e.g., constant S_k). Since TTL density

$$g_T(x) = G_T'(x) = \frac{1 - F_T(x)}{E[T]}, \tag{18}$$

this procedure yields $F_T(x)$ and $E[T] = 1/g_T(0)$ using numerical differentiation of $G_T(x)$. Recalling (6), knowledge of h and $E[T]$ produces λ. Finally, assuming $G_U(x)$ is known, all remaining pieces fall into place, i.e.,

$$p = P(R_T < R_U) = \int_0^\infty (1 - G_U(x)) g_T(x) dx \tag{19}$$

and thus $f = 1 - h(1-p)$. The only still-unknown parameter in Fig. 8 is $G_U(x)$. We focus on its estimation next.

D. Shark

Define s_k' to be the first sample point that hits the k-th ON period seen by the probes and let d_k' be the preceding download time. Then, the observer can lower-bound d_k' by

$$W_k = \max(s_{k-1}' + R_T(s_{k-1}'), \max(s_i : s_i < s_k')), \tag{20}$$

which is the end of the $(k-1)$-st ON period or the preceding sample point, whichever happened later. Similar to

passive sampling, suppose Δ'_k is a binary process that is 1 if a modification is detected during interval $(s'_{k-1}, s'_k]$, i.e., the record is different at download s'_k. We further define $\gamma'(k) = \max(i \leq k : \Delta'_i = 1)$ to be last sample (no later than k) that detected an update. Note that variables Δ_k and $\gamma(k)$ from passive measurement are unavailable in the active case, which is why they are replaced with those that can be computed by the observer, i.e., Δ'_k and $\gamma'(k)$.

Our main idea is to allow the estimator to utilize residuals $R_T(s'_k)$ *together* with the already-recovered $G_T(x)$ to probabilistically determine the location of unknown download points d'_k. Recalling that $A_T(s'_k)$ is the age of the ON period at s'_k, it follows that $d'_k = s'_k - A_T(s'_k)$. The remaining elements of this approach, which we call Shark[2], is to determine the conditional distribution of $A_T(s'_k)$ and change the EM algorithm to work with random bounds $[L_k, R_k]$.

There are two pieces of information known to the observer that affect the distribution of $A_T(s'_k)$. The first is residual $y = R_T(s'_k)$ returned by the cache and the second is upper bound $z = s'_k - W_k$ from preceding samples. Conditioning on $A_T < z$ and $R_T = y$, the tail distribution of A_T is

$$\begin{aligned}\bar{F}_A(x; y, z) &:= P(A_T > x | R_T = y, A_T < z) \\ &= \frac{F_T(y+z) - F_T(x+y)}{F_T(y+z) - F_T(y)},\end{aligned} \tag{21}$$

where $F_T(x)$ comes from (18). Unless T is memoryless (i.e., exponential), parameter y provides useful clues about the possible values of age. For light-tailed distributions (e.g., constant, uniform), the age is generally a decreasing function of y. For heavy-tailed cases (e.g., Pareto), it is the opposite.

Leveraging (9), Shark constrains the update age using *random* upper/lower bounds

$$L_k = s'_k - A_T(s'_k) - s'_{\gamma'(k)} + A_T(s'_{\gamma'(k)}) \tag{22}$$

$$R_k = s'_k - A_T(s'_k) - s'_{\gamma'(k)-1} + A_T(s'_{\gamma'(k)-1}), \tag{23}$$

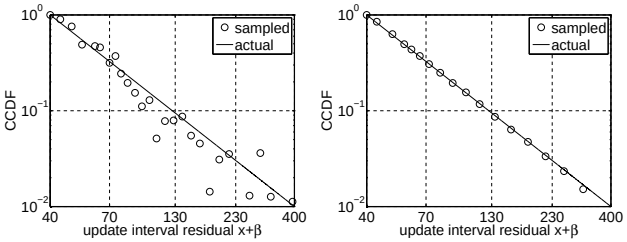
where $A_T(s'_k) \sim F_A(x; R_T(s'_k), s'_k - W_k)$. Note that $\gamma'(k) = k$ implies that $L_k = 0$, which leaves only two random variables on the right side of (22)-(23). Otherwise, there are three of them, all with known parameters needed to construct (21).

For the computation, we discretize interval $[0, s'_k - W_k]$ and replace (21) with a PMF that assigns weights to a number of bins in that range. We then iterate over all possible ways to draw the three (or possibly two) age variables from their respective distributions and compute the probability ν_k for each deterministic bound $[l_k, r_k]$. These are fed into concave EM, which we modify to take weights into account, i.e., use

$$W_j(t) = \sum_{k=1}^{n} \frac{a_{jk}}{V_k(t)} \nu_k. \tag{24}$$

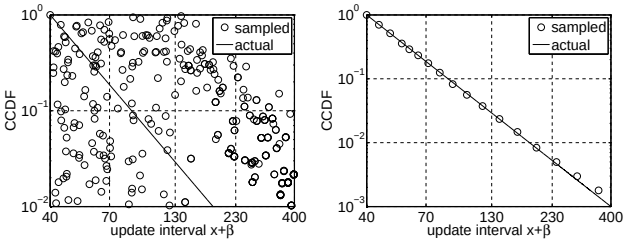
as a replacement for (17).

[2]Ram-ventilation sharks, which must continuously swim to avoid oxygen depletion, are some of the most active animals in the world.



(a) M_6 (b) non-concave EM ($\epsilon = 10^{-5}$)

Fig. 9. Passive estimation of $G_U(x)$ ($n = 10$K samples).



(a) non-concave EM ($\epsilon = 10^{-5}$) (b) concave EM ($\epsilon = 10^{-5}$)

Fig. 10. Passive estimation of $F_U(x)$ in Chameleon ($n = 10$K samples).

VI. Experiments

A. Setup

To investigate the accuracy of the developed sampling techniques, we registered an Internet domain and developed a custom authoritative server $\mathcal{A}$, written in C++, that could answer iterative IPv4 queries from arbitrary Internet hosts. Each DNS record (i.e., a hostname in our domain) that participated in the experiment was equipped with an update process N_U, which changed the returned IP at points $\{u_i\}$. For passive monitoring, we created another C++ solution that ran a local DNS server $\mathcal{L}$, which accepted recursive/iterative queries from IPs in our subnet and resolved them at the authoritative server. Note that $\mathcal{A}$ and $\mathcal{L}$ were placed on different hosts.

For active sampling, we needed a remote server $\mathcal{R}$ that allowed recursive queries, returned non-fake answers, and complied with the source-provided TTL. To discover such options, we performed a port-53 UDP scan of the Internet in March 2017 (over 2.8B probed IPs) and found 8M responding hosts. Out of these, 3.7M (47%) reported no error and 3.2M (41%) supplied correct answers from $\mathcal{A}$. We then selected a subset of IPs from the last list, probed them for several days to ensure longevity, tested for TTL compliance, and screened against load-balancers that did not have a common cache. Out of the surviving options, we picked one at random to be $\mathcal{R}$.

Finally, we added into the mix an observer process and a background-traffic generator whose purpose was to query $\mathcal{L}$ or $\mathcal{R}$ depending on the scenario. The observer asked iterative queries after random delays S_k, while the generator sent packets from the query process N_Q. With the exception of network RTTs and various OS scheduling delays, the system functioned close to that in Fig. 2 and allowed controlled experimentation with a known ground-truth.

TABLE I
RUNTIME IN PASSIVE ESTIMATION OF $G_U(x)$

n	M_6	Naive EM $\epsilon = 10^{-4}$	Algorithm 1 $\epsilon = 10^{-4}$
10^4	0.3 sec	9.4 sec	0.06 sec
10^5	58 sec	2.9 min	0.13 sec
10^6	2.2 hours	43 min	0.19 sec
10^7	–	5.5 hours	0.70 sec
10^8	–	–	5.79 sec
10^9	–	–	27.9 sec

Unless mentioned otherwise, we kept the average delay between updates $E[U] = 20$ seconds. The values of T dispatched from the authoritative server were uniform in $[1, 19]$ seconds, which mimicked Akamai-style churn rates and TTLs. Background clients sent traffic to the cache using a Poisson N_Q with rate $\lambda = 1$ query/sec and the active observer utilized exponential S with mean 4 seconds. When a variable X needed to be Pareto-distributed, we drew it from $F(x) = 1 - (1 + x/\beta)^{-\alpha}$, where $x \geq 0$. We kept $\alpha = 3$ and $\beta = 2E[X]$ throughout all experiments, where $E[X]$ was the desired mean of the variable.

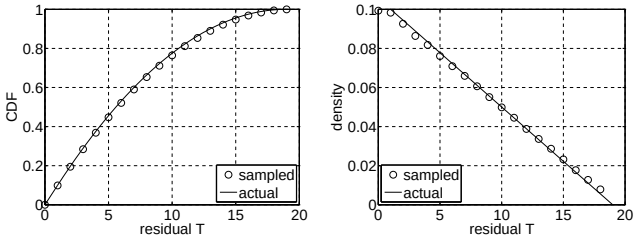
B. Passive Sampling

We start by examining recovery of $G_U(x)$ using our implementation of the local resolver $\mathcal{L}$. As shown in Fig. 9(a) for Pareto U, method M_6 from [25] correctly identifies the trend of the tail, but the produced estimate is rather noisy. This approach is not well suited for such small samples sizes n. Applying the non-concave Chameleon from (12) yields a much better result in Fig. 9(b). Its main drawback, however, is that conversion of the residual CDF into $F_U(x)$ is often impossible in practice. This is illustrated by the mishmash of points in Fig. 10(a). Upgrading to the concave Chameleon from (15) leads to an amazingly better outcome in Fig. 10(b).

Table I compares the runtime of M_6, the naive implementation of EM that directly computes (10)-(12), and our version of Chameleon in Algorithm 1. Observe that quadratic scaling of M_6 quickly makes it infeasible. In fact, in the last row of the table it requires an extrapolated 79 years to finish. The naive EM scales much better, although it still does not offer an appealing framework above 100K samples. On the other hand, Chameleon in the last column delivers blazingly fast results for all input size up to 1B. While collecting this many observations in passive sampling is not likely in practice, recall that Shark generates a huge number of deterministic bounds $[l_k, r_k]$ from (22)-(23). If age $A_T(s'_k)$ is discretized into 50 bins, a workload with 10K random bounds $[L_k, R_k]$ produces 500M intervals for concave EM. Therefore, Algorithm 1 is by far the only feasible way to compute the Shark estimator.

C. Active Sampling

We first analyze accuracy of recovery for $G_T(x)$, h, $E[T]$, and λ from Fig. 8. The result for $G_T(x)$ and its density $g_T(x)$ is given by Fig. 11, which indicates a strong match. Table II displays the remaining three parameters and also compares against Chameleon. As expected, Chameleon's averages converge quicker, although active measurement still produces solid results. Another interesting fact is that λ in the fourth column is highly sensitive to errors in h, which comes from its shape $\lambda = h/(1-h)/E[T]$.



(a) $G_T(x)$ (b) $g_T(x)$

Fig. 11. Distribution of TTL in active sampling ($n = 10$K samples).

TABLE II
RELATIVE ESTIMATION ERROR OF SIMPLE PARAMETERS

n	Shark			Chameleon		
	h	$E[T]$	λ	h	$E[T]$	λ
10^2	2.94%	16.8%	51.6%	1.58%	4.18%	7.93%
10^3	0.88%	5.81%	11.5%	0.42%	1.58%	2.79%
10^4	0.29%	2.40%	3.73%	0.16%	0.41%	0.80%

We next evaluate estimation accuracy of $G_U(x)$. Fig. 12 shows the output of Shark, where it recovers all four classes of distributions with excellent accuracy. A further confirmation of these findings is given by Table III, where Shark comes pretty close to matching the performance of Chameleon, even through the latter operates with substantially more information.

Overall, concave Chameleon emerges as hands-down the best tool for sampling update dynamics and staleness in *single-blind* scenarios (i.e., only the update process is hidden) and Shark does the same in *double-blind* (i.e., both update/download processes are invisible to the observer).

VII. CONCLUSION

We presented a general framework for modeling freshness in TTL-based caching systems, proposed two novel techniques for remotely measuring this metric in the current Internet, and improved the performance of existing update-sampling algorithms. Even under random TTL, our most advanced method can recover all unknown parameters of the system (i.e., hit and request rate, update/TTL distributions, and freshness), without requiring any change to the DNS infrastructure.

REFERENCES

[1] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, , and B. Ohlman, "A Survey of Information-Centric Networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, Jul. 2012.

[2] H. Akcan, T. Suel, and H. Brönnimann, "Geographic Web Usage Estimation by Monitoring DNS Caches," in *Proc. LocWeb*, Apr. 2008.

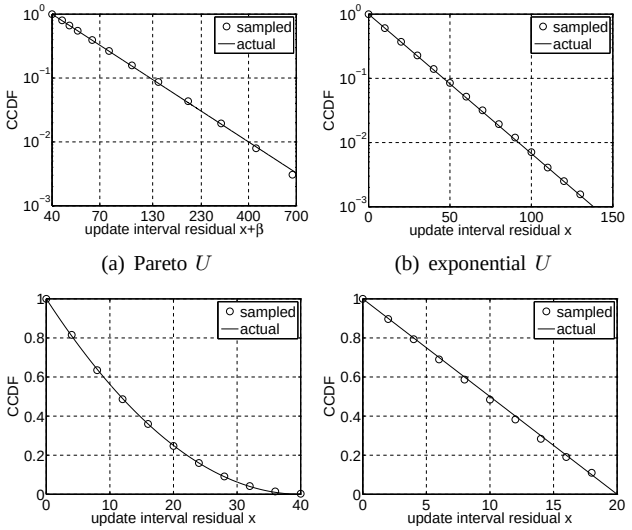
[3] H. A. Alzoubi, M. Rabinovich, and O. Spatscheck, "The Anatomy of LDNS Clusters: Findings and Implications for Web Content Delivery," in *Proc. WWW*, May 2013, pp. 83–94.

[4] O. Bahat and A. M. Makowski, "Measuring Consistency in TTL-based Caches," *Elsevier Perf. Evaluation*, vol. 62, no. 1-4, pp. 439–455, Oct. 2005.

[5] D. S. Berger, P. Gland, S. Singla, and F. Ciucu, "Exact analysis of TTL cache networks," *Performance Evaluation*, vol. 79, pp. 2–23, Apr. 2014.

TABLE III
RELATIVE ESTIMATION ERROR OF COMPLEX PARAMETERS

n	Pareto U				Exponential U				Uniform U				Constant U			
	Shark		Chameleon		Shark		Chameleon		Shark		Chameleon		Shark		Chameleon	
	p	f	p	f	p	f	p	f	p	f	p	f	p	f	p	f
10^2	6.5%	5.8%	6.3%	5.6%	5.6%	5.3%	5.0%	4.4%	5.0%	4.8%	4.0%	3.4%	5.8%	5.2%	3.8%	3.4%
10^3	1.9%	1.8%	1.9%	1.7%	1.8%	1.6%	1.7%	1.5%	1.7%	1.5%	1.2%	1.1%	1.8%	1.6%	1.1%	1.0%
10^4	0.5%	0.4%	0.7%	0.6%	0.6%	0.5%	0.6%	0.5%	0.5%	0.4%	0.5%	0.5%	1.1%	0.8%	0.4%	0.4%



(a) Pareto U (b) exponential U

(c) uniform U (d) constant U

Fig. 12. Tail of $G_U(x)$ in Shark ($n = 10$K samples).

[6] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, "AdaptSize: Orchestrating the Hot Object Memory Cache in a Content Delivery Network." in *Proc. NSDI*, 2017, pp. 483–498.

[7] R. Blanco, E. Bortnikov, F. Junqueira, R. Lempel, L. Telloli, and H. Zaragoza, "Caching Search Engine Results over Incremental Indices," in *Proc. ACM SIGIR*, Jul. 2010, pp. 82–89.

[8] T. Callahan, M. Allman, and M. Rabinovich, "On Modern DNS Behavior and Properties," *ACM CCR*, vol. 43, no. 3, pp. 7–15, Jul. 2013.

[9] B. B. Cambazoglu, F. P. Junqueira, V. Plachouras, S. Banachowski, B. Cui, S. Lim, and B. Bridge, "A Refreshing Perspective of Search Engine Caching," in *Proc. WWW*, Apr. 2010, pp. 181–190.

[10] C. Chen, S. Matsumoto, and A. Perrig, "ECO-DNS: Expected Consistency Optimization for DNS," in *Proc. IEEE ICDCS*, Jun. 2015, pp. 256–267.

[11] X. Chen, H. Wang, S. Ren, and X. Zhang, "DNScup: Strong cache consistency protocol for DNS," in *Proc. IEEE ICDCS*, Jun. 2006.

[12] E. Cohen and H. Kaplan, "The Age Penalty and Its Effect on Cache Performance," in *Proc. USENIX USITS*, Mar. 2001, pp. 73–84.

[13] E. Cohen and H. Kaplan, "Aging Through Cascaded Caches: Performance Issues in the Distribution of Web Content," in *Proc. ACM SIGCOMM*, Aug. 2001, pp. 41–53.

[14] E. Cohen and H. Kaplan, "Refreshment Policies for Web Content Caches," in *Proc. IEEE INFOCOM*, Apr. 2001, pp. 1398–1406.

[15] Y. Fang, Z. J. Haas, B. Liang, and Y.-B. Lin, "TTL Prediction Schemes and the Effects of Inter-update Time Distribution on Wireless Data Access," *Wireless Networks*, vol. 10, no. 5, pp. 607–619, Sep. 2004.

[16] K. Fawaz and H. Artail, "DCIM: Distributed Cache Invalidation Method for Maintaining Cache Consistency in Wireless Mobile Networks," *IEEE Trans. Mobile Computing*, vol. 12, no. 4, pp. 680–693, Apr. 2013.

[17] N. C. Fofack and S. Alouf, "Modeling Modern DNS Caches," in *Proc. Int. Conf. Perf. Evaluation Methodologies and Tools*, Dec. 2013.

[18] N. C. Fofack, P. Nain, G. Neglia, and D. Towsley, "Performance Evaluation of Hierarchical TTL-based Cache Networks," *Elsevier Computer Networks*, vol. 65, pp. 212–231, Jun. 2014.

[19] H. Gomaa, G. G. Messier, and R. Davies, "Hierarchical Cache Performance Analysis Under TTL-Based Consistency," *IEEE/ACM Trans. Networking*, vol. 23, no. 4, pp. 1190–1201, Aug. 2015.

[20] Y. T. Hou, J. Pan, B. Li, and S. S. Panwar, "On Expiration-Based Hierarchical Caching Systems," *IEEE JSAC*, vol. 22, no. 1, pp. 134–150, Jan. 2004.

[21] C. Jackson, A. Barth, A. Bortz, W. Shao, and D. Boneh, "Protecting Browsers from DNS Rebinding Attacks," in *Proc. ACM CCS*, Oct. 2007.

[22] J. Jung, A. W. Berger, and H. Balakrishnan, "Modeling TTL-based Internet Caches," in *Proc. IEEE INFOCOM*, Apr. 2003, pp. 417–426.

[23] J.-J. Lee, K.-Y. Whang, B. S. Lee, and J.-W. Chang, "An Update-Risk Based Approach to TTL Estimation in Web Caching," in *Proc. IEEE WISE*, Dec. 2002, pp. 21–29.

[24] X. Li, D. B. Cline, and D. Loguinov, "On Sample-Path Staleness in Lazy Data Replication," in *Proc. IEEE INFOCOM*, Apr. 2015, pp. 1104–1112.

[25] X. Li, D. B. Cline, and D. Loguinov, "Temporal Update Dynamics under Blind Sampling," in *Proc. IEEE INFOCOM*, Apr. 2015, pp. 1634–1642.

[26] G. Liu, H. Shen, H. Chandler, and J. Li, "Measuring and Evaluating Live Content Consistency in a Large-Scale CDN," in *Proc. IEEE ICDCS*, Jun. 2014, pp. 268–277.

[27] X. Ma, J. Zhang, Z. Li, J. Li, J. Tao, X. Guan, J. C. Lui, and D. Towsley, "Accurate DNS query characteristics estimation via active probing," *J. Network and Computer Applications*, vol. 47, pp. 72–84, Jan. 2015.

[28] V. Martina, M. Garetto, and E. Leonardi, "A Unified Approach to the Performance Analysis of Caching Systems," in *Proc. IEEE INFOCOM*, Jun. 2014, pp. 2040–2048.

[29] C. Olston and J. Widom, "Best-Effort Cache Synchronization With Source Cooperation," in *Proc. ACM SIGMOD*, May 2002, pp. 73–84.

[30] J. Pang, A. Akella, A. Shaikh, B. Krishnamurthy, and S. Seshan, "On the Responsiveness of DNS-based Network Control," in *Proc. ACM IMC*, Oct. 2004, pp. 21–26.

[31] M. A. Rajab, F. Monrose, A. Terzis, and N. Provos, "Peeking Through the Cloud: DNS-based Estimation and Its Applications," in *Proc. ACNS*, Jun. 2008.

[32] F. B. Sazoglu, B. B. Cambazoglu, R. Ozcan, I. S. Altingovde, and O. Ulusoy, "Strategies for Setting Time-to-live Values in Result Caches," in *Proc. ACM CIKM*, Oct. 2013, pp. 1881–1884.

[33] K. Schomp, T. Callahan, M. Rabinovich, and M. Allman, "On Measuring the Client-Side DNS Infrastructure," in *Proc. ACM IMC*, Oct. 2013, pp. 77–90.

[34] H. Shen, "IRM: Integrated File Replication and Consistency Maintenance in P2P Systems," *IEEE Trans. Parallel Distrib. Systems*, vol. 21, no. 1, pp. 100–113, Jan. 2010.

[35] Y. F. Sit, F. C. Lau, and C.-L. Wang, "On The Cooperation of Web Clients and Proxy Caches," in *Proc. IEEE ICPADS*, Dec. 2005, pp. 264–270.

[36] R. Srinivasan, C. Liang, and K. Ramamritham, "Maintaining Temporal Coherency of Virtual Data Warehouses," in *Proc. IEEE RTSS*, Dec. 1998.

[37] X. Tang, J. Xu, and W.-C. Lee, "Analysis of TTL-Based Consistency in Unstructured Peer-to-Peer Networks," *IEEE Trans. Parallel Distrib. Systems*, vol. 19, no. 12, pp. 1683–1694, Dec. 2008.

[38] B. W. Turnbull, "The Empirical Distribution Function with Arbitrarily Grouped, Censored and Truncated Data," *Journal of the Royal Statistical Society*, vol. 38, no. 3, pp. 290–295, 1976.

[39] C. E. Wills, M. Mikhailov, and H. Shang, "Inferring Relative Popularity of Internet Applications by Actively Querying DNS Caches," in *Proc. ACM IMC*, May 2003, pp. 78–90.

[40] J. L. Wolf, M. S. Squillante, P. S. Yu, J. Sethuraman, and L. Ozsen, "Optimal Crawling Strategies for Web Search Engines," in *Proc. WWW*, May 2002, pp. 136–147.