Consistency analysis and evaluation of TTL-based Internet caches

by Omri Bahat, Armand M. Makowski

TR 2005-78

Consistency Analysis and Evaluation of TTL-based Internet Caches[1]

Omri Bahat   Armand M. Makowski

Department of Electrical and Computer Engineering

and the Institute for Systems Research

University of Maryland, College Park, Maryland 20742

Email: obahat@glue.umd.edu, armand@isr.umd.edu

Abstract

Consistency algorithms have been proposed for a wide range of applications that include distributed shared memories (DSM), distributed file systems, and databases. Fundamental definitions and operational constraints that are specific for each system do not necessarily translate well to Internet caches. A Web object is consistent if it is identical to the master document at the origin server, at the time it is served to users, therefore cached objects become stale immediately after the master is modified. Stale cache copies remain served to users until the cache is refreshed, subject to the network transmit delays. However, the performance of Internet consistency algorithms is evaluated through the corresponding cache hit rate and network traffic load that do not inform on the service of stale data, and are therefore inadequate, as outlined in numerous studies. To date, neither an analytical framework nor a suitable measure are available to model the service of stale data to users. In this paper we seek to remedy this state of affairs by formalizing both a framework and the novel hit* rate consistency measure, which captures non-stale downloads from the cache. To demonstrate this new methodology, we analyze and evaluate the consistency performance of a well studied TTL algorithm, under both zero and non-zero download latency. We conclude that data consistency can be significantly degraded even when a high hit rate is achieved, by calculating the incurred hit and hit* rates. The proposed procedure can be used to evaluate additional TTL and other (e.g., polling and invalidation) Web consistency protocols, as well as those retained by other applications (e.g., virtual shared memories).

Key words: Web caching, Cache consistency, TTL, Stochastic modeling

[1] This material is based upon work supported by NASA under award number NCC8235. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Aeronautics and Space Administration.

1 Introduction

Web caching aims to reduce network traffic, server load, and user-perceived retrieval latency by placing document replicas on proxy caches that are strategically placed within the network. One problem that arises with Internet caches is the staleness of stored objects: In order for Web caches to be useful, cache consistency must be maintained with respect to the *master* document at the origin server. In fact, consistency is critical in certain applications, such as stock quotes reported to day traders or online auctions, as user decisions are based on the recency of collected data. Assuring consistency becomes more challenging when server documents are updated rapidly (e.g., weather updates, sports scores, news portals), since frequent server-cache communications are required to properly monitor the freshness at the cache. Since all messages exchanged between the server and the cache are subject to constraints imposed by the network infrastructure, consistency is impeded by download delays that are considerably noticeable in wireless and mobile devices, or when data is transmitted over satellite.

1.1 Enforcing data consistency on the Internet

Consistency algorithms are implemented either at the cache or at the server, for the sole purpose of increasing the likelihood that documents served to users by the cache are identical to those offered at the server. The *consistency model* in use reflects the degree to which cached replicas are kept consistent. On the Internet, algorithms fall in one of three categories, namely Time-To-Live (TTL), client polling, and server invalidation, which are now described. [2]

With TTL algorithms, a document is placed in the cache alongside a server-assigned Time-To-Live, say T. The object is considered *valid* until T time units elapse from the instance of cache placement. Each request for a valid object incurs a *cache-hit* and is served by the cache, and the first request presented after the TTL expiration is a *cache-miss* that is forwarded to the server, which in turn sends a fresh copy to the cache. Algorithms in this class include the fixed TTL [14] where the Time-To-Live is set to a constant T, and the adaptive TTL [11] and Squid's LM-Factor [27] in which the Time-To-Live is determined by timestamps of past updates to the master. The popularity of these algorithms is a direct consequence of their simplicity, sufficiently good performance, and the flexibility to assign a TTL value to a single cached item.

[2] Principles of each category are listed. Practical algorithms may vary based on their implementation: A conditional *If-Modified-Since* can be sent to the server instead of the usual *GET* request, or TTL values can be passed to the cache through one of the *Expires*, *Max-Age*, or *Min/Max-TTL* parameters of the HTTP [12].

A client polling consistency algorithm (e.g., Piggyback cache validation [17]) is invoked according to a schedule dictated by the cache. Each time it runs, the algorithm connects to the server and initiates an *If-Modified-Since* request, accompanied by the identifiers (i.e., *validators* [12]) of the cached item. Commonly used validators include the *Last-Modified-Timestamp*, document version, and the *entity-tag* (ETag). If the server finds that the value of the validators match those of the master (i.e., cached copy is up to date), it sends a *304 Not-Modified* reply; otherwise the latest version is sent to the cache.

A server invalidation protocol, such as Piggyback server invalidation [16] or the Invalidation-report examined in [29] for wireless devices, is launched by the server upon each update to the master (or shortly thereafter). The server informs the cache of the changes, which then marks the copy as invalid. The first request that follows the invalidation incurs a miss and is forwarded to the server, which then loads a new master into the cache. Algorithms in this class are not very popular since each server must maintain a list of all system caches.

1.2 Web consistency models

A given Web consistency algorithm complies with one of two models, namely *weak* or *strong* consistency, which were defined by Cao and Liu [6] for the case of zero download delay.[3] These definitions are revisited below to include the possibility of non-zero network delays, preceded by some new terminology that captures the relationship (i.e., equality) of a cached copy to its master over time.

A cached document is invalidated when the cache detects the expiration of the associated validators, e.g., expiration of the TTL, or the availability of a new version and ETag in polling and invalidation algorithms. If a copy is valid, each request presented at the cache incurs a cache-hit and receives the stored replica; otherwise, requests are forwarded to the server. A *server-fresh* document is defined as the latest serviceable document at the server, whereas a *cache-fresh* document is defined as a valid cached copy, or equivalently as one thought to be server-fresh by the cache. In order to accurately capture the state of a cached copy with respect to the master, define a *cache-fresh** document as one that is both cache-fresh and server-fresh; a document that is not cache-fresh* is termed *cache-stale**. Due to the different schedules used to invoke the various consistency algorithms, as well as the non-zero server-cache transmit delay Δ, it is possible that a cached item is considered valid in spite the availability of a later version at the server, resulting in the undesirable download of cache-stale* (thus inconsistent) documents.

With this new state classification, a *cache-hit** is defined as a cache-hit that prompts a

[3] Comments regarding consistency models in other applications and their inapplicability to Web caching are provided in the next section.

server-fresh download from the cache to the user; its complement is called a *cache-miss**. Each hit* is therefore a hit, but the reverse does not hold. *Strong* consistency algorithms are now characterized by users being served strictly server-fresh documents under zero delays and processing times. A consistency algorithm that is not strong is termed *weak*, in which case there is a possibility that users download inconsistent copies (i.e., ones that are cache-fresh yet not server-fresh), even under zero delays. Causal TTL and polling algorithms are therefore weakly consistent, whereas invalidation protocols are typically strongly consistent.

1.3 Contributions of the proposed work

With a given consistency algorithm, we can associate two basic quantities, namely the hit rate and the hit* rate. These quantities are defined as the long-run average download rate of cache-fresh and cache-fresh* documents from the cache, respectively. Since each hit* is necessarily a hit, the hit* rate is always less than or equal to the hit rate, and for strongly consistent algorithms, equality is achieved if and only if $\Delta = 0$. To the best of the authors' knowledge, the hit* rate appears to be the first performance metric proposed to quantitatively address consistency issues.

Concerns regarding the download of cache-stale* objects are mentioned in numerous studies (e.g., see [6] [7] [11] [13] [14] [24] [26] and references therein), and the need for an appropriate measure was duly noted [15]. The new metric serves exactly that purpose. The hit* rate is most useful when the cache utilizes weak protocols, for then the probability that users retrieve stale objects is potentially large: Consider the fixed TTL algorithm when T is very large, and server documents are updated frequently. A large value for T guarantees a high hit rate and lowered bandwidth utilization [7] [13] [14], but results in poor quality of data (QoD),[4] even when the communication latency is negligible (e.g., broadband connectivity). However, measuring consistency is also important for strong consistency mechanisms deployed in a hierarchy of caches, or when the delay is large (e.g., satellite and wireless [24] [29]), as previously concluded in [26].

In this paper, we address these issues by formulating a framework where user requests and master updates are modelled by mutually independent point processes on $[0, \infty)$. The focus is on a single server-cache pair and a single data object, as we attempt to isolate relevant issues. Then, both the hit* and hit rates of any given consistency algorithm can in principle be evaluated, and various design parameters could be tuned on the basis of the resulting performance.

Here, we carry out such an evaluation for the extremely popular fixed TTL algorithm, by exploiting some of its key properties, for any value of $\Delta \geq 0$. The hit rate of the fixed TTL

[4] We assimilate freshness and object consistency with the quality of data.

was already discussed in [14], but only for $\Delta = 0$. Closed form expressions for the hit and hit* rates are available in some special cases when the requests are generated according to a Poisson process. Computable bounds are derived below when the request process is a renewal process, and are tightened when the inter-request time distribution belongs to certain subclasses of distributions (e.g., IFR, DFR, NBUE and NWUE). On the basis of these results, we can now investigate the degradation of the hit and hit* rates as a function of the key system parameters T and Δ.

While many of the results presented in this paper hold under the assumption that the point processes are stationary and ergodic, most of the discussion given here is carried out under renewal assumptions in the interest of brevity and mathematical simplicity. Moreover, the proposed framework can be easily applied to the study of other algorithms, as was done for the weak LM-Factor protocol in [3]. It can also be used to investigate consistency issues associated with invalidation and polling algorithms, as well as techniques employed by other distributed systems such as virtual shared memories and file sharing systems.

The paper is organized as follows: Consistency issues in various systems and their association to the Web are discussed in Section 2, and the suggested modeling framework is presented in Section 3. In Section 4 we define the performance metrics used, as we describe the fixed TTL algorithm through the nomenclature of the framework. The analysis of the fixed TTL is executed in Section 5 and Section 6, for zero and non-zero download delays, respectively, where expressions are obtained for the hit and hit* rates. Relationships and asymptotics of the hit and hit* rates are presented in Section 7. Results are specialized to Poisson requests in Section 8 where we also discuss the issue of model validation, followed by performance bounds for several Internet applications.

2 Relationship to other distributed systems

A review of the literature quickly reveals that consistency protocols have been extensively studied in computer architectures, distributed shared memories (DSM), network and file sharing systems, and distributed databases. Problems pertaining to each of these systems, and the associated consistency models, are summarized below. Additional details regarding individual system characteristics and (dis)similarities to Internet caches can be found in references [6] [10] and [15].

In multiprocessor computers and DSM, several processors can write to a single memory cell. If read and write operations are not coordinated, then a risk arises that the value read by any processor is other than the one required by the program, so that the execution outcome may not be as expected. Lamport's *sequential consistency* [18] resolves this consistency issue by requiring all memory operations to execute one at a time, and the operations of a single processor to appear in the order described by the program. However, this strict policy inhibits many advantages of distributed computing. For this reason, other relaxed solutions are adopted, among which we find the *processor consistency* whereby writes issued from a processor are only observed in their issuance order, *weak consistency* that demands strict ordering within critical sections of the program, its extension *release consistency*, and others.

On the World Wide Web, a single master document is maintained at the server and is updated by a single writer.[5] A request for a Web object first arrives to the cache. If the latter contains a valid copy it is sent to the user; otherwise the cache launches a request to the server that sends the copy to the cache, and from there to the user. The server is therefore the single source of updatable data, whereas documents are always read from the cache, prompting potential inconsistencies between the master and read objects. These rules do not follow the setup of DSM systems, or that of distributed databases described below.

In databases, consistency is satisfied if database entries are identical to those stored in the cache. Consistency is impeded by transactions that span several databases, by two types of caching techniques: Intra-transaction in which data is cached within the transaction boundaries and removed from the cache upon its completion, and inter-transaction whereby data cached by one transaction is not purged and remains accessible to others. When programs execute concurrently, consistency is achieved through some form of serialization, combined with consistency protocols (see details in [9] and references therein). Solutions proposed in this context typically answer synchronization problems between multiple writers to a single cache, and therefore do not apply to the Web.

Coherency problems in hierarchical computer memories and file sharing systems exhibit most similarities to the consistency issue on the Web. With the first, a local memory is dedicated to each CPU, a main virtual memory (VM) is shared between all processing units, and each processor writes to the VM in turn. Each unit first reads the local memory, and the VM is queried in the absence (or invalidity) of local data. The shared memory, local memory, and each CPU are therefore synonymous to the Web server, cache, and the user, respectively. Invalidation algorithms utilized by the VM are nearly identical to those of the Web [23], with the following key exception: Processors proximity and shared buffers permit a hardware based synchronization and tight control for the VM, that are neither feasible nor desired in most Web applications.

In network and distributed file sharing systems, a client-cache copy is consistent if it is identical to the master at the file server, as on the Internet. If the client does not hold

[5] This is also true when multiple users attempt to update the master, as in online reservations applications, since a single process posts the update at the server.

the requested document, it is sent from the server to the client-cache, and cached files are accessed for read and write purposes. Heterogeneous consistency solutions are adopted by different implementations, as some systems support either sequential or concurrent document accesses over all users, or both. Overlaps with the Internet object consistency are expressed not only as noted above, yet also through the use of similar algorithms: A TTL-based control is exercised in Sun's NFS to send the client's validators to the server, in order to achieve both freshness control and crash recovery. Other systems (e.g., AFS, Sprite, Coda, Zebra, Harp) employ customized invalidation and polling procedures, which are similar to those on the Web.

A general framework for consistency analysis and performance measurement under a given algorithm is not yet available in the literature, for either shared memories or file sharing systems (to our knowledge). The need for such a framework therefore remains [15] [26], in spite of the outlined analogies.

3 A simple framework

We now develop a simple framework to address consistency issues in the context of Web caching, as outlined earlier: The system is made up of a site called the *origin* where the current authoritative version of the data is maintained, and of *requestors*. Each requestor is identified with a cache that is used either by users or by client-caches. Thus, the origin and requestors are synonymous with server and caches, respectively.

Caches are assumed to be of *infinite* size, reflecting the fact that storage is ample. The need to specify a replacement policy is moot, and only the operational rules of the consistency algorithms matter. In particular, once a document has been placed in the cache, a copy is always available at the requesting cache, although said copy may be either fresh or stale at any given time. Under these circumstances, there is no loss of generality in abstracting a caching system into a single cache-server pair, and in considering a single cacheable data item, say D, in isolation, as we do hereunder.

3.1 Modeling requests

User requests for the document D arrive according to a point process $\{T^r_n, n = 0, 1, \ldots\}$ with the understanding that the n^{th} request occurs at time T^r_n. Thus, $T^r_n \leq T^r_{n+1}$ for each $n = 0, 1, \ldots$ with $T^r_0 = 0$. Let $\{R_{n+1},\ n = 0, 1, \ldots\}$ denote the sequence of inter-request times with $R_{n+1} = T^r_{n+1} - T^r_n$ for each $n = 0, 1, \ldots$. The point process $\{T^r_n, n = 0, 1, \ldots\}$ is assumed to be *simple* in that $R_{n+1} > 0$ a.s. for $n = 0, 1, \ldots$, so multiple requests cannot occur simultaneously.

As customary, the counting process $\{R(t),\ t \geq 0\}$ associated with the point process $\{T^r_n, n = 0, 1, \ldots\}$ is given by

$$R(t) = \sup\{n = 0, 1, \ldots :\ T^r_n \leq t\}, \quad t \geq 0 \tag{1}$$

so that $R(t)$ counts the number of requests in the interval $(0, t]$. The corresponding residual lifetime process $\{R_e(t),\ t \geq 0\}$ is defined by

$$R_e(t) = T^r_{R(t)+1} - t, \quad t \geq 0. \tag{2}$$

If $T^r_n \leq t < T^r_{n+1}$ for some $n = 0, 1, \ldots$, then $R(t) = n$ and $R_e(t) = T^r_{n+1} - t$, i.e., $R_e(t)$ represents the amount of time that will elapse until the occurrence of the next request after time t. We assume at minimum that the point process $\{T^r_n, n = 0, 1, \ldots\}$ admits a request rate in the sense that there exists a finite constant $\lambda_R > 0$ given by the limit

$$\lambda_R = \lim_{t\to\infty} \frac{R(t)}{t} \quad a.s. \tag{3}$$

3.2 Modeling document updates

The document D changes over time, and is updated according to the second point process $\{T^u_m, m = 0, 1, \ldots\}$ where T^u_m is the epoch at which the m^{th} update takes place. We denote by $\{U_{m+1},\ m = 0, 1, \ldots\}$ the sequence of inter-update times with $U_{m+1} = T^u_{m+1} - T^u_m$ for each $m = 0, 1, \ldots$. Here as well, $T^u_m \leq T^u_{m+1}$ for each $m = 0, 1, \ldots$ with $T^u_0 = 0$. The point process $\{T^u_m, m = 0, 1, \ldots\}$ is assumed to be simple so that multiple updates are ruled out.

In analogy with (1), the counting process $\{U(t),\ t \geq 0\}$ associated with the point process $\{T^u_m, m = 0, 1, \ldots\}$ is given by

$$U(t) = \sup\{m = 0, 1, \ldots :\ T^u_m \leq t\}, \quad t \geq 0 \tag{4}$$

with $U(t)$ counting the number of updates in the interval $(0, t]$. The corresponding residual lifetime process $\{U_e(t),\ t \geq 0\}$ is defined by

$$U_e(t) = T^u_{U(t)+1} - t, \quad t \geq 0 \tag{5}$$

so that $U_e(t)$ represents the amount of time that will elapse until the next update after t.

As before, the process $\{T^u_m, m = 0, 1, \ldots\}$ admits a rate, referred to as the update rate, in the sense that there exists a finite constant $\lambda_U > 0$ given by

$$\lambda_U = \lim_{t\to\infty} \frac{U(t)}{t} \quad a.s. \tag{6}$$

Throughout, the point processes $\{T_n^r, n = 0, 1, \ldots\}$ and $\{T_m^u, m = 0, 1, \ldots\}$ are assumed to be *mutually independent*. This reflects the lack of correlation between user behavior and the evolution of data content.

For reasons of brevity and mathematical simplicity, these point processes are assumed to be *renewal* processes. For the requests, this means that the inter-request times $\{R_{n+1},\ n = 0, 1, \ldots\}$ form a sequence of i.i.d. rvs distributed according to the common cdf F_R. Let R denote any rv distributed according to F_R. Similarly, when the update process is a renewal process, the inter-update times $\{U_{m+1},\ m = 0, 1, \ldots\}$ form a sequence of i.i.d. rvs distributed according to the cdf F_U. We denote by U any rv distributed according to F_U. The cdfs F_R and F_U are assumed to have finite mean $m(F_R)$ and $m(F_U)$, in which case it is well known [25] that $\lambda_R = m(F_R)^{-1}$ and $\lambda_U = m(F_U)^{-1}$.

Let $\Delta \geq 0$ denote the fixed download delay of D over the network, i.e., if a document is sent from the server at time t, it is received by the cache at time $t + \Delta$, at which point it is ready for access by the users. In the other direction, communication from the cache to the server is deemed *instantaneous* as it entails the transmission of very short control messages.

Before embarking in earnest on the discussion, we stress that many of the developments of the paper could be carried out under the weaker assumptions of stationarity and ergodicity on the point processes of interest.

4 The fixed TTL algorithm and its performance measures

With the nomenclature introduced above, the fixed TTL can be described as follows: Whenever the server receives a request for D, say at time T_n^r for some $n = 0, 1, \ldots$, it returns the current version together with the TTL field $T > 0$. The first miss request for D arrives to the cache at $T_0^r = 0$, and the cache places the returned version at $T_0^r + \Delta$. Any subsequent request for D at the cache in the time interval $(T_0^r + \Delta, T_0^r + \Delta + T)$ is served directly from the cache without contacting the server.

If a request at time T_n^r for some $n = 1, 2, \ldots$ is the first one presented after the TTL has expired, then it is forwarded to the server, in which case requests arriving during $(T_n^r + \Delta, T_n^r + \Delta + T)$ are all hits. The discussion is carried out under the assumption that the requests presented in $(T_n^r, T_n^r + \Delta)$ are sent to the server as well. However, the copies of D received in response to these requests are neither placed nor do they reset the TTL, as assumed in [6] for the empirical evaluation of various TTL algorithms.[6] Results are

[6] We restrict the analysis to the case $T \geq \Delta$, as customary on the Web.

also available when each download resets the TTL [3], yet are omitted here due to space limitations.

With the fixed TTL algorithm, we can associate the two performance measures mentioned earlier, namely the hit and hit* rates. These metrics reflect the quality (or freshness) of D from different viewpoints, namely cache freshness and server freshness. Under the network delay Δ, a validator process $\{L_\Delta(t),\ t \geq 0\}$ is introduced to track the cache freshness of the cached copy of D, in that a hit occurs at request time T^r_n *if and only if* $L_\Delta(T^r_n-) > 0$. The hit rate is then simply defined by

$$H(T, \Delta) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}\left[L_\Delta(T^r_n-) > 0\right]. \tag{7}$$

Similarly, the server freshness of the cached version of D is characterized through another process $\{L^\star_\Delta(t),\ t \geq 0\}$, so that a hit* occurs at request time T^r_n *if and only if* $L^\star_\Delta(T^r_n-) > 0$, and the hit* rate is defined as

$$H^\star(T, \Delta) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}\left[L^\star_\Delta(T^r_n-) > 0\right]. \tag{8}$$

The limits in both (7) and (8) are taken in the a.s. sense and are assumed to exist with $H(T, \Delta)$ and $H^\star(T, \Delta)$ *constants.* In that case, the a.s. limit

$$\lim_{N\to\infty} \frac{\sum_{n=1}^{N} \mathbf{1}\left[L^*_\Delta(T^r_n-) > 0\right]}{\sum_{n=1}^{N} \mathbf{1}\left[L_\Delta(T^r_n-) > 0\right]} = \frac{H^\star(T, \Delta)}{H(T, \Delta)} \tag{9}$$

also exists as a constant. This ratio represents the fraction of server-fresh hits out of all hits, and therefore measures the QoD produced by a given algorithm.

The remainder of the paper is devoted to evaluating these rates for the fixed TTL algorithm, and to understand how they are affected by the values T and Δ, and by the statistics of request and update patterns. The result on the hit rate when $\Delta = 0$ is presented in Section 5, where it is followed by the calculation for the hit* rate for $\Delta = 0$ as well. These results can be easily recovered from the more general findings provided in Section 6, for the case of non-zero delays, and their proofs are therefore omitted. Properties of the hit and hit* rates under the fixed TTL are reported in Section 7.

5 Zero delays

Cache freshness of the object D is completely described[7] by the validator process $\{L(t),\ t \geq 0\}$, which continuously tracks the TTL value at the cache. The process has right-continuous sample paths with left limits, and is defined by

$$L(t) = \left(L(T^r_n) - (t - T^r_n)\right)^+, \quad T^r_n \leq t < T^r_{n+1} \tag{10}$$

[7] In this section, we drop Δ from earlier notation as it is now set to zero.

for each $n = 0, 1, \ldots$, with the update rule

$$L(T_n^r) = \begin{cases} T & \text{if } L(T_n^r-) = 0 \\ L(T_n^r-) & \text{if } L(T_n^r-) > 0 \end{cases}. \tag{11}$$

Operational assumptions made earlier lead to the initial condition $L(0-) = 0$ so that $L(0) = T$ by (11). The timer will have expired at time $t > 0$ if and only if $L(t-) = 0$. Thus, the n^{th} request at time T_n^r produces a hit if $L(T_n^r-) > 0$; otherwise it will be a miss. The hit rate $H(T)$ for the fixed TTL is now as defined in (7). Its evaluation has been carried out already by Jung et al. [14].

Proposition 1 *If the point process $\{T_{n+1}^r,\ n = 0, 1, \ldots\}$ is a renewal process, then it holds that*

$$H(T) = \frac{\mathbf{E}\left[R(T-)\right]}{1 + \mathbf{E}\left[R(T-)\right]}.$$

Note that $\mathbf{E}\left[R(T-)\right] = \mathbf{E}\left[R(T)\right]$ for all $T > 0$ as soon as F_R admits a density, a common occurrence in applications.

As we now turn to the hit* rate, we note that even in the absence of transmission delays between the server and the cache, there is a possibility that a request incurs a hit for a stale copy. The consistency of the cached object with that offered by the server is captured by the process $\{L^*(t),\ t \geq 0\}$ which tracks the time until the expiration of the cache-fresh* copy. This process has right-continuous sample paths with left limits, and is defined by

$$L^\star(t) = \left(L^\star(T_n^r) - (t - T_n^r)\right)^+, \qquad T_n^r \leq t < T_{n+1}^r \tag{12}$$

for each $n = 0, 1, \ldots$, with the update rule

$$L^\star(T_n^r) = \begin{cases} \min(L(T_n^r), U_e(T_n^r)) & \text{if } L^\star(T_n^r-) = 0 \\ L^\star(T_n^r-) & \text{if } L^\star(T_n^r-) > 0 \end{cases}. \tag{13}$$

The initial condition is taken to be $L^\star(0-) = 0$ so that $L^\star(0) = \min(T, U_e(0))$. The hit* rate $H^\star(T)$ is given by (8) with $\{L^*(t),\ t \geq 0\}$ as defined above.

Proposition 2 *Under the assumptions of Proposition 1, it holds that*

$$H^\star(T) = \frac{\mathbf{E}\left[R(\min(T, U_e)-)\right]}{1 + \mathbf{E}\left[R(T-)\right]} \tag{14}$$

provided the point process $\{T_{m+1}^u,\ m = 0, 1, \ldots\}$ is also a renewal process.

In this last expression the stationary forward recurrence time U_e is taken to be independent of the counting process $\{R(t),\ t \geq 0\}$; its distribution is given by

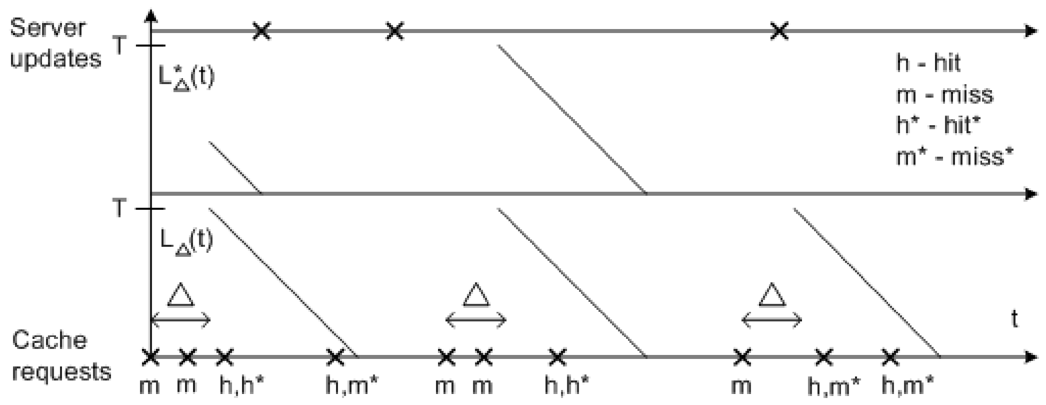

Fig. 1. A time line diagram of requests, updates, and the freshness tracking processes for the fixed TTL with $\Delta > 0$.

$$\mathbf{P}\left[U_e \le t\right] = \lambda_U \int_0^t \mathbf{P}\left[U > u\right] du, \quad t \ge 0.$$

An alternative expression for $H^\star(T)$ follows by specializing (21) with $\Delta = 0$, and is given by

$$H^\star(T) = H(T) - \lambda_U \int_0^T \frac{\mathbf{E}\left[R(T-)\right] - \mathbf{E}\left[R(u)\right]}{\mathbf{E}\left[R(T-)\right] + 1} \mathbf{P}\left[U > u\right] du. \tag{15}$$

6 Non-zero delays

In the presence of a network delay $\Delta > 0$, cache freshness is monitored through the validator process $\{L_\Delta(t),\ t \ge 0\}$, which continuously tracks the TTL value at the cache (see Figure 1). This process has right-continuous sample paths with left limits, and is defined as follows: First, define the $\mathbb{N}$-valued rvs $\{\mu_k,\ k = 0, 1, \ldots\}$ recursively by

$$\mu_{k+1} = \inf\left\{n > \mu_k :\ T^r_{\mu_k} + \Delta + T \le T^r_n\right\} \tag{16}$$

for each $k = 0, 1, \ldots$ with $\mu_0 = 0$. The rv μ_k identifies the k^{th} request forwarded to the server that resets the TTL value to T, as we recall that requests in $(T^r_{\mu_k}, T^r_{\mu_k} + \Delta)$ are forwarded as well, but do not affect the TTL at the cache in that interval. For each $k = 0, 1, \ldots$ we can then write

$$L_\Delta(t) = \left(L_\Delta(T^r_{\mu_k} + \Delta) - (t - (T^r_{\mu_k} + \Delta))\right)^+ \tag{17}$$

on the interval $[T^r_{\mu_k} + \Delta, T^r_{\mu_{k+1}} + \Delta)$, with the update rule $L_\Delta(T^r_{\mu_k} + \Delta) = T$. We initially take $L_\Delta(t) = 0$ for $0 \le t < \Delta$, and the hit rate $H(T, \Delta)$ can be written as in (7) with $\{L_\Delta(t),\ t \ge 0\}$.

Proposition 3 *If the point process $\{T^r_{n+1},\ n = 0, 1, \ldots\}$ is a renewal process, then we have*

$$H(T, \Delta) = \frac{\mathbf{E}\left[R((T + \Delta)-)\right] - \mathbf{E}\left[R(\Delta)\right]}{1 + \mathbf{E}\left[R((T + \Delta)-)\right]} \tag{18}$$

for each $\Delta \geq 0$.

As expected, this result specializes for $\Delta = 0$ to the one stated in Proposition 1 and in [14]. A proof of Proposition 3 is available in Appendix 1.

Next, the evaluation of the hit* rate is made possible through the server-freshness tracking process $\{L^\star_\Delta(t),\ t \geq 0\}$ at the cache. This process has right-continuous sample paths with left limits, and is defined as follows: For each $n = 0, 1, \ldots$ we set

$$L^\star_\Delta(t) = \left(L^\star_\Delta(T^r_n + \Delta) - (t - (T^r_n + \Delta))\right)^+ \tag{19}$$

whenever $T^r_n + \Delta \leq t < T^r_{n+1} + \Delta$ with the reinitialization rule

$$L^\star_\Delta(T^r_n + \Delta) = \begin{cases} \min(L_\Delta(T^r_n + \Delta), (U_e(T^r_n) - \Delta)^+) & \text{if } L^\star_\Delta((T^r_n + \Delta)-) = 0 \\ L^\star_\Delta((T^r_n + \Delta)-) & \text{if } L^\star_\Delta((T^r_n + \Delta)-) > 0 \end{cases}.$$

The initial conditions are taken to be $L^*_\Delta(t) = 0$ for $0 \leq t < \Delta$, as illustrated in Figure 1. The hit* rate $H^\star(T, \Delta)$ is given by (8), this time with $\{L^*_\Delta(t),\ t \geq 0\}$, and is evaluated in the following proposition.

Proposition 4 *Under the assumptions of Proposition 2, we have*

$$H^\star(T, \Delta) = \frac{\mathbf{E}\left[R(\min(\Delta + T, U_e)-)\right] - \mathbf{E}\left[R(\min(\Delta, U_e))\right]}{1 + \mathbf{E}\left[R((T + \Delta)-)\right]} \tag{20}$$

for each $\Delta \geq 0$.

As before, U_e in (20) is taken to be independent of the counting process $\{R(t),\ t \geq 0\}$, which allows us to rewrite the hit* rate as

$$\begin{aligned} H^\star(T, \Delta) &= H(T, \Delta)\mathbf{P}\left[U_e > T + \Delta\right] + \lambda_U \int_\Delta^{T+\Delta} \frac{\mathbf{E}\left[R(u)\right] - \mathbf{E}\left[R(\Delta)\right]}{1 + \mathbf{E}\left[R((T + \Delta)-)\right]} \mathbf{P}\left[U > u\right] du & (21) \\ &= H(T, \Delta)\mathbf{P}\left[U_e > \Delta\right] - \lambda_U \int_\Delta^{T+\Delta} \frac{\mathbf{E}\left[R((T + \Delta)-)\right] - \mathbf{E}\left[R(u)\right]}{1 + \mathbf{E}\left[R((T + \Delta)-)\right]} \mathbf{P}\left[U > u\right] du. \end{aligned}$$

The result (20) reduces to Proposition 2 for the case of $\Delta = 0$, and a proof for it is available in Appendix 2.

7 Bounds and comparisons

Relationships between the derived rates are readily obtained from (21) for any value of $\Delta \geq 0$, under the assumptions of Proposition 2. The ratio (9) that captures the fraction of non-stale hits satisfies the bounds [8]

$$\mathbf{P}\left[U_e > T + \Delta\right] \leq \frac{H^*(T,\Delta)}{H(T,\Delta)} \leq \mathbf{P}\left[U_e > \Delta\right], \tag{22}$$

which shows clearly the interplay between network delays and update statistics. The ability of the fixed TTL to ensure consistency is degraded as the delay increases, as is the case for most algorithms in practice. Better performance can be achieved by lowering the value of T, yet it is possible that frequent updates prevent users from ever being served with server-fresh data. [9]

If documents are rarely updated, i.e., $\lambda_U \simeq 0$, then $H^*(T,\Delta) \simeq H(T,\Delta)$ as expected. On the other hand, it is a simple matter to check that

$$\lim_{T\to\infty} H(T,\Delta) = 1 \quad \text{but} \quad \lim_{T\to\infty} H^\star(T,\Delta) = 0. \tag{23}$$

These asymptotics follow from the expressions for the quantities $H(T,\Delta)$ and $H^\star(T,\Delta)$, and are simple consequences of the Basic Renewal Theorem according to which $\lim_{t\to\infty} \frac{\mathbf{E}[R(t)]}{t} = \lambda_R$ and of the fact $\lim_{u\to\infty} u\mathbf{P}\left[U > u\right] = 0$ (implied by the integrability of F_U). By similar arguments we conclude that

$$\lim_{\Delta\to\infty} H(T,\Delta) = \lim_{\Delta\to\infty} H^\star(T,\Delta) = 0. \tag{24}$$

Additional properties, such as conditions for the monotonicity of the calculated rates as Δ and T vary, can be extracted from the above propositions. These attributes allow us to identify the impact on data consistency under several interesting scenarios, however are omitted from the paper due to limited space.

8 The hit* rate in applications: Bounds and model validation

Interestingly enough, the experimental validation of the suggested model has already been carried out in numerous studies (e.g., see [4] [8] [21] [22] [24] [28] and their references). The selection of F_R and F_U that best fits the model dynamics are specific to each application: Inter-request times in HTTP and FTP caches follow the Weibull and Pareto heavy tailed distributions, as reported in [8] [22] and emphasized by Bestavros et al. in [4]. Locality

[8] In view of a natural probabilsitic impression, we expect that the upper bound at (22) holds for a large class of consistency algorithms.

[9] e.g., when the master is updated every Δ time units in which case $\mathbf{P}\left[U_e > \Delta\right] = 0$.

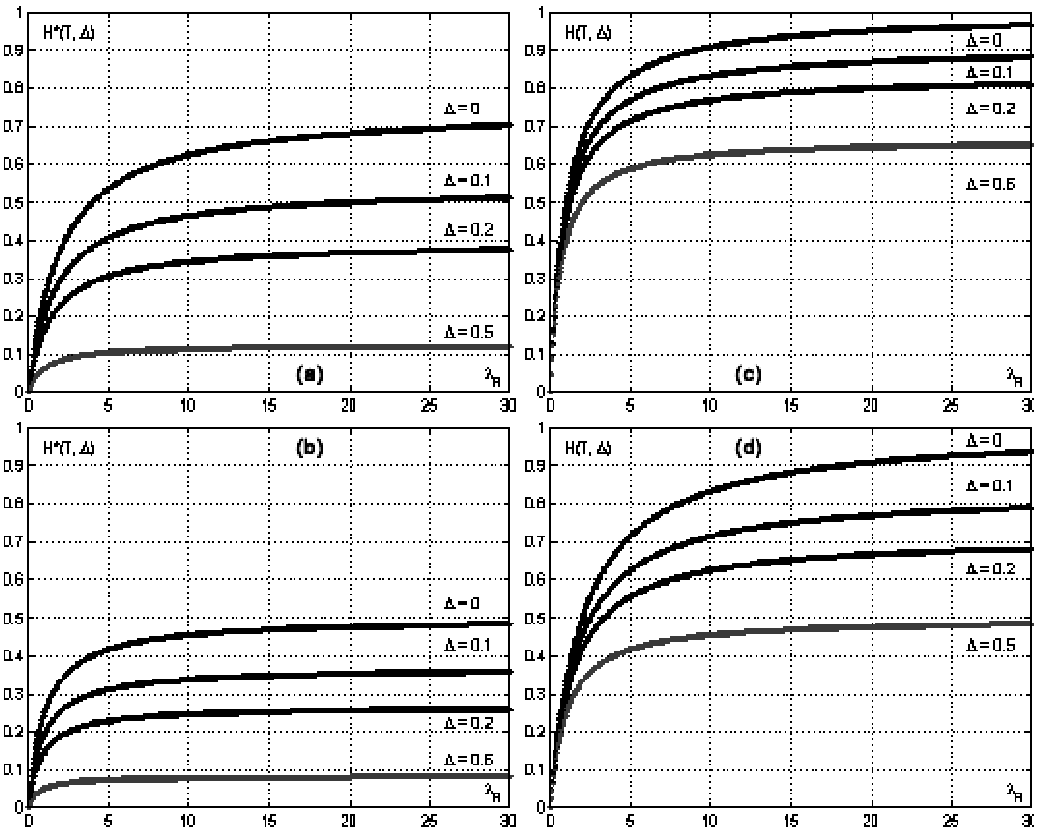

Fig. 2. Hit and hit* rates for Poisson requests and fixed inter-updates $\Delta^u = 1$. (a) Hit* rate, $T = 1$; (b) Hit* rate, $T = 0.5$; (c) Hit rate, $T = 1$; (d) Hit rate, $T = 0.5$.

of reference whereby recently accessed documents are likely to be shortly requested again can be expressed through an appropriate selection of F_R. For the updates, Web pages that contain stocks and weather information on the Yahoo portal are updated periodically every few seconds [24]. Logs collected from news servers [21] and Harvest caches [6] suggest the bimodal inter-update time distribution.

It is rather difficult (if at all possible) to calculate the hit and hit* rates for general distributions. Indeed, the (non-delayed) renewal function $t \to \mathbf{E}[R(t)]$ is not known in closed form, except in some special cases (e.g., when R is lattice or uniformly distributed, or for a class of matrix-exponential distributions [1]). Consequently, in order to apply the results of this paper in general applications, we derive distribution-free bounds on the obtained rates. These bounds can be tightened when F_R belongs to several subclasses of distributions of interest.

8.1 Poisson requests

Poisson requests correspond to the generic inter-request rv R being exponentially distributed, say with rate λ_R, i.e., $F_R(t) = 1 - e^{-\lambda_R t}$ ($t \geq 0$), in which case $\mathbf{E}[R(t)] = \lambda_R t$ for all $t \geq 0$. The hit rate for the fixed TTL algorithm becomes

$$H_{Pois}(T, \Delta) = \frac{\lambda_R T}{\lambda_R(T+\Delta)+1}$$

and the corresponding hit* rate is given by

$$H^{\star}_{Pois}(T, \Delta) = \mathbf{P}\left[U_e > T+\Delta\right] H_{Pois}(T, \Delta) + \frac{\lambda_U \lambda_R}{\lambda_R(T+\Delta)+1} \int_0^T u\mathbf{P}\left[U > u+\Delta\right] du.$$

While a simple closed-form expression is available for the hit rate, the hit* rate can be evaluated in principle once the distribution F_U is specified. For instance, consider the case when updates occur periodically every Δ^u time units. It is plain that $H^{\star}(T, \Delta) = 0$ whenever $\Delta^u \leq \Delta$ (as would be expected). However if $\Delta^u > \Delta$, a simple calculation shows that

$$H^{\star}_{Pois}(T, \Delta) = \frac{\mathbf{1}\left[\Delta^u > T+\Delta\right]\left(\Delta^u - (T+\Delta)\right) \cdot \lambda_R T + \frac{\lambda_R}{2}\min^2(T, \Delta^u - \Delta)}{\Delta^u(\lambda_R(\Delta+T)+1)}.$$

This expression allows the comparison between the hit and hit* rates incurred by the fixed TTL under the practical systems examined in [24], as delineated in Figure 2 for several values of T and Δ.

8.2 Distribution-free results

Bounds are available for the renewal function associated with any distribution F_R. First, recall that the forward recurrence time rv R_e associated with the rv R is distributed according to $\mathbf{P}\left[R_e \leq t\right] = \lambda_R \int_0^t \mathbf{P}\left[R > u\right] du, \ (t \geq 0)$. Lorden [19] has shown the upper bound

$$\mathbf{E}\left[R(t)\right] \leq \lambda_R t + \lambda_R^2 \mathbf{E}\left[R^2\right] - 1, \quad t \geq 0 \tag{25}$$

while Marshall [20] proved the lower bound

$$\mathbf{E}\left[R(t)\right] \geq \lambda_R t - \mathbf{P}\left[R_e \leq t\right], \quad t \geq 0. \tag{26}$$

[10] These bounds can be used to get rough estimates of the hit rate under the fixed TTL, namely

$$\frac{\lambda_R T + \mathbf{P}\left[R_e > T+\Delta\right] - \lambda_R^2 \mathbf{E}\left[R^2\right]}{\lambda_R(T+\Delta) + \lambda_R^2 \mathbf{E}\left[R^2\right]} \leq H(T, \Delta) \leq \frac{\lambda_R T + \lambda_R^2 \mathbf{E}\left[R^2\right] - \mathbf{P}\left[R_e > \Delta\right]}{\lambda_R(T+\Delta) + \mathbf{P}\left[R_e > T+\Delta\right]}.$$

These bounds may not be useful when $\lambda_R^2 \mathbf{E}\left[R^2\right]$ is large for then there is a risk that the upper bound is too loose and the lower bound negative (hence useless). [11] Similar

[10] Additional bounds can be found in [20].

[11] $\lambda_R^2 \mathbf{E}\left[R^2\right] \geq 1$ always since this is equivalent to $\mathrm{var}(R) \geq 0$.

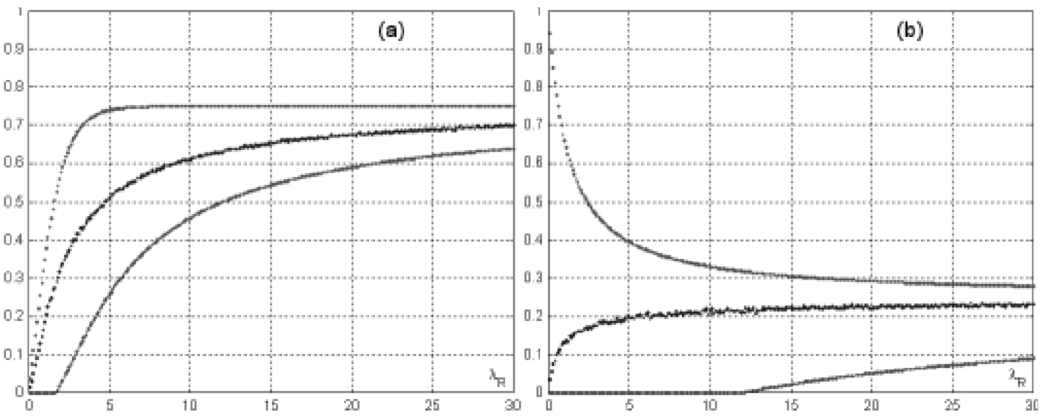

Fig. 3. Upper bound, lower bound, and simulated hit* rate with fixed inter-updates $\Delta^u = 1$ and $T = 0.5$: (a) Weibull inter-request times, $\Delta = 0$, $\alpha = 1.3$; (b) Pareto inter-request times, $\Delta = 1/3$, $\alpha = 2.1$.

bounding arguments can be invoked for the hit* rate $H^\star(\Delta, T)$, in the process yielding the upper bound

$$\frac{\lambda_R(T+\Delta)+1}{\lambda_R(T+\Delta)+\mathbf{P}\left[R_e > T+\Delta\right]} H^\star_{Pois}(\Delta, T) + \frac{\mathbf{P}\left[U_e > \Delta\right](\lambda_R^2 \mathbf{E}\left[R^2\right] - \mathbf{P}\left[R_e > \Delta\right])}{\lambda_R(T+\Delta)+\mathbf{P}\left[R_e > T+\Delta\right]}$$

and the associated lower bound

$$\frac{\lambda_R(T+\Delta)+1}{\lambda_R(T+\Delta)+\lambda_R^2\mathbf{E}\left[R^2\right]} H^\star_{Pois}(\Delta, T) - \frac{\mathbf{P}\left[U_e > \Delta\right]\left(\lambda_R^2 \mathbf{E}\left[R^2\right] - \mathbf{P}\left[R_e > T+\Delta\right]\right)}{\lambda_R(T+\Delta)+\lambda_R^2\mathbf{E}\left[R^2\right]}.$$

8.3 NBUE and NWUE Requests

A distribution F_R on $[0, \infty)$ is said to be Increasing (resp. Decreasing) Failure Rate, denoted IFR (resp. DFR), if the mapping $t \to \frac{\mathbf{P}[R>t+r]}{\mathbf{P}[R>t]}$ is non-increasing (resp. non-decreasing) in t for each $r \geq 0$. [12]

The Weibull distribution often used in request modeling is characterized by $F_R(t) = 1 - e^{-(\beta t)^\alpha}$ $(t \geq 0)$ with $\alpha, \beta > 0$, with the two first moments given by

$$\mathbf{E}\left[R\right] = \frac{1}{\beta}\,\Gamma(\alpha^{-1}+1) \quad \text{and} \quad \mathbf{E}\left[R^2\right] = \frac{1}{\beta}\,\Gamma(2\alpha^{-1}+1).$$

The Weilbull distribution is IFR (DFR) for $\alpha \geq (\leq) 1$.

A second distribution often encountered in network traffic modeling is the Pareto distribution defined as $F_R(t) = 1 - \left(\frac{k}{k+t}\right)^\alpha$ $(t \geq 0)$, with $\alpha, k > 0$. This distribution is DFR, and we restrict the discussion to $\alpha > 2$, in which case the first two moments

$$\mathbf{E}\left[R\right] = \frac{k}{\alpha - 1} \quad \text{and} \quad \mathbf{E}\left[R^2\right] = \frac{2k^2}{(\alpha-2)(\alpha-1)}$$

[12] Much of this material can be found in the monograph by Barlow [5].

are finite.

A distribution F_R on $[0,\infty)$ is said to be New Better (Worse) Than Used in Expectation, in short NBUE (NWUE), if $\mathbf{P}[R_e > t] \leq (\geq)\mathbf{P}[R > t]$ for all $t \geq 0$, respectively. It is well known [5] that if F_R is IFR (resp. DFR), then it is also NBUE (resp. NWUE), in which case the associated renewal function is bounded by

$$\mathbf{E}[R(t)] \leq \text{(resp. } \geq) \ \lambda_R t, \quad t \geq 0. \tag{27}$$

We now demonstrate the use of these bounds, in combination with the Lorden and Marshall bounds, for the hit* rate in (20). For F_R NBUE, we have

$$H^\star(T,\Delta) \leq \frac{(\lambda_R(T+\Delta)+1)H^\star_{Pois}(T,\Delta)}{\lambda_R(T+\Delta)+\mathbf{P}[R_e > T+\Delta]} + \frac{\mathbf{P}[U_e > \Delta]\,\mathbf{P}[R_e \leq \Delta]}{\lambda_R(T+\Delta)+\mathbf{P}[R_e > T+\Delta]}$$

and $$H^\star(T,\Delta) \geq H^\star_{Pois}(T,\Delta) - \frac{\mathbf{P}[U_e > \Delta]\,\mathbf{P}[R_e \leq T+\Delta]}{\lambda_R(T+\Delta)+1}.$$

On the other hand, with F_R NWUE, we have

$$H^\star(T,\Delta) \leq H^\star_{Pois}(T,\Delta) + \frac{\mathbf{P}[U_e > \Delta]\,(\lambda_R^2\mathbf{E}\left[R^2\right]-1)}{\lambda_R(T+\Delta)+1},$$

and $$H^\star(T,\Delta) \geq \frac{(\lambda_R(T+\Delta)+1)H^\star_{Pois}(T,\Delta)}{\lambda_R(T+\Delta)+\lambda_R^2\mathbf{E}[R^2]} - \frac{\mathbf{P}[U_e > \Delta]\,(\lambda_R^2\mathbf{E}\left[R^2\right]-1)}{\lambda_R(T+\Delta)+\lambda_R^2\mathbf{E}[R^2]}.$$

The use of these bounds is exhibited in Figure 3, when inter-request times are modeled by the Weibull and Pareto distributions under fixed inter-update times. Bounds for the obtained hit rate, as well as results for other distribution classes are omitted from the paper but can be found in [3].

9 Appendices

In the next appendices 1 and 2, we make use of the $\mathbb{N}$-valued rvs $\{\mu_k,\ k=0,1,\ldots\}$ defined recursively through (16). Note that $R(T^r_{\mu_k}) = \mu_k$ for all $k=0,1,\ldots$, and that under the renewal assumptions on the request process, the rvs $\{R(T^r_{\mu_{\ell+1}}) - R(T^r_{\mu_\ell}),\ \ell=0,1,\ldots\}$ are i.i.d., each distributed according to $R(T^r_{\mu_1})$. Therefore, by the Strong Law of Large Numbers we find

$$\lim_{k\to\infty}\frac{\mu_k}{k} = \lim_{k\to\infty}\frac{1}{k}\sum_{\ell=0}^{k-1}\Big(R(T^r_{\mu_{\ell+1}}) - R(T^r_{\mu_\ell})\Big) = \mathbf{E}\left[R(T^r_{\mu_1})\right] \quad a.s. \tag{.1}$$

By the very definition of $T^r_{\mu_1}$, we have that

$$\mathbf{E}\left[R(T^r_{\mu_1})\right] = 1 + \mathbf{E}\left[R((T+\Delta)-)\right], \tag{.2}$$

as explained through arguments below.

1 Proof of proposition 3

Since the rvs $\{\mu_k,\ k = 0, 1, \ldots\}$ monotonically exhaust $\mathbb{N}$ a.s., it is plain that

$$H(T, \Delta) = \lim_{k\to\infty} \frac{1}{\mu_k} \sum_{n=1}^{\mu_k} \mathbf{1}\left[L_\Delta(T_n^r-) > 0\right]. \tag{1}$$

In order to make use of this fact, fix $k = 0, 1, \ldots$ and consider the dynamics of the freshness tracking process on the interval $(T^r_{\mu_k}, T^r_{\mu_{k+1}}]$: With $T^r_{\mu_k} < t \leq T^r_{\mu_k} + \Delta$, we have $L_\Delta(t-) = 0$ since $L_\Delta(T^r_{\mu_k}) = 0$. The validator process is reinitialized at time $T^r_{\mu_k} + \Delta$ to the value $L_\Delta(T^r_{\mu_k} + \Delta) = T$, returning to zero with the expiration of the TTL at time $T^r_{\mu_k} + \Delta + T$, i.e., $L_\Delta((T^r_{\mu_k} + \Delta + T)-) = 0$.

Thus, the requests made at the cache in the interval $(T^r_{\mu_k}, T^r_{\mu_k} + \Delta]$ incur misses, while those occurring in $(T^r_{\mu_k} + \Delta, T^r_{\mu_k} + \Delta + T)$ are necessarily hits. Also, there is exactly one request made in the interval $[T^r_{\mu_k} + \Delta + T, T^r_{\mu_{k+1}}]$, and it is necessarily a miss. In summary, we see that there are exactly $1 + \left(R(T^r_{\mu_k} + \Delta) - R(T^r_{\mu_k})\right)$ misses in the interval $(T^r_{\mu_k}, T^r_{\mu_{k+1}}]$. Therefore, for each $k = 1, 2, \ldots$, we get

$$\begin{aligned}\sum_{n=1}^{\mu_k} \mathbf{1}\left[L_\Delta(T_n^r-) = 0\right] &= \sum_{\ell=0}^{k-1} \sum_{\mu_\ell < n \leq \mu_{\ell+1}} \mathbf{1}\left[L_\Delta(T_n^r-) = 0\right] \\ &= \sum_{\ell=0}^{k-1} \left(1 + \left(R(T^r_{\mu_\ell} + \Delta) - R(T^r_{\mu_\ell})\right)\right). \end{aligned} \tag{2}$$

Again, under the renewal assumption on the request process, the rvs $\{R(T^r_{\mu_\ell} + \Delta) - R(T^r_{\mu_\ell}),\ \ell = 0, 1, \ldots\}$ are i.i.d. rvs, each distributed according to $R(\Delta)$, and the Strong Law of Large Numbers now gives

$$\lim_{k\to\infty} \frac{1}{k} \sum_{\ell=0}^{k-1} \left(R(T^r_{\mu_k} + \Delta) - R(T^r_{\mu_k})\right) = \mathbf{E}\left[R(\Delta)\right] \quad a.s. \tag{3}$$

Since

$$1 - H(T, \Delta) = \lim_{k\to\infty} \frac{1}{\mu_k} \sum_{n=1}^{\mu_k} \mathbf{1}\left[L_\Delta(T_n^r-) = 0\right],$$

it is plain from (2) and (3) that

$$1 - H(T, \Delta) = \frac{1 + \mathbf{E}\left[R(\Delta)\right]}{1 + \mathbf{E}\left[R((T + \Delta)-)\right]} \tag{4}$$

by the usual arguments, as we recall (.1) with (.2). The desired expression (18) is finally obtained. $\Box$

2 Proof of Proposition 4

The arguments are similar to those given in the proof of Proposition 3. We

begin by noting that $\quad H^\star(T,\Delta) = \lim_{k\to\infty} \frac{1}{\mu_k} \sum_{n=1}^{\mu_k} \mathbf{1}\left[L^*_\Delta(T^r_n-) > 0\right].$ (1)

In order to use (1), fix $k = 0, 1, \ldots$ and consider the cache-fresh* tracking process on the interval $(T^r_{\mu_k}, T^r_{\mu_{k+1}}]$: A download is requested at time $T^r_{\mu_k}$, resulting in a cache placement at time $T^r_{\mu_k} + \Delta$. If no update occurs before this cache placement, then the validation process is reinitialized at time $T^r_{\mu_k} + \Delta$ to the value $\min(T, U_e(T^r_{\mu_k}) - \Delta)$, and returns to zero with the expiration of this TTL. In other words, $L^*_\Delta((T^r_{\mu_k} + \Delta + \min(T, U_e(T^r_{\mu_k}) - \Delta))-) = 0$. On the other hand, if an update occurs before placement, i.e., $U_e(T^r_{\mu_k}) \le \Delta$, [13] then $L^*_\Delta(t) = 0$ on the entire interval $(T^r_{\mu_k} + \Delta, T^r_{\mu_{k+1}}]$.

As already discussed in the proof of Proposition 3, the hits on the interval $(T^r_{\mu_k}, T^r_{\mu_{k+1}}]$ can occur only in the subinterval $(T^r_{\mu_k} + \Delta, T^r_{\mu_k} + \Delta + T)$. If $U_e(T^r_{\mu_k}) \le \Delta$, none of these requests can be hits*. However, if $\Delta < U_e(T^r_{\mu_k})$, then all the requests made in the interval $(T^r_{\mu_k} + \Delta, T^r_{\mu_k} + \min(\Delta + T, U_e(T^r_{\mu_k})))$ are hits*, while none of those made in the interval $[T^r_{\mu_k} + \min(\Delta + T, U_e(T^r_{\mu_k})), T^r_{\mu_k} + \Delta + T]$ are as they are all hit but miss* requests. Summarizing we conclude that the number H^*_k of hits* in the interval $(T^r_{\mu_k}, T^r_{\mu_{k+1}}]$ is given by the difference

$$H^*_k = R((T^r_{\mu_k} + \min(\Delta + T, U_e(T^r_{\mu_k})))-) - R(T^r_{\mu_k} + \min(\Delta, U_e(T^r_{\mu_k}))). \tag{2}$$

Consequently, for each $k = 0, 1, \ldots$, we can write

$$\sum_{n=1}^{\mu_k} \mathbf{1}\left[L^*_\Delta(T^r_n-) > 0\right] = \sum_{\ell=0}^{k-1} \sum_{\mu_\ell < n \le \mu_{\ell+1}} \mathbf{1}\left[L^*_\Delta(T^r_n-) > 0\right] = \sum_{\ell=0}^{k-1} H^\star_\ell. \tag{3}$$

Under the renewal assumptions on the independent processes $\{T^r_n,\ n = 0, 1, \ldots\}$ and $\{T^u_m,\ m = 0, 1, \ldots\}$, we can easily verify the following (with details available in [3]): First, we have the equalities

$$\lim_{k\to\infty} \frac{1}{k} \sum_{\ell=0}^{k-1} \left(R((T^r_{\mu_\ell} + \min(\Delta + T, U_e(T^r_{\mu_\ell})))-) - R(T^r_{\mu_\ell})\right) = \mathbf{E}\left[R(\min(\Delta + T, U_e)-)\right], \quad a.s.$$

$$\text{and} \quad \lim_{k\to\infty} \frac{1}{k} \sum_{\ell=0}^{k-1} \left(R(T^r_{\mu_\ell} + \min(\Delta, U_e(T^r_{\mu_\ell}))) - R(T^r_{\mu_\ell})\right) = \mathbf{E}\left[R(\min(\Delta, U_e))\right] \quad a.s.$$

In both cases the rv U_e is taken to be independent of the counting process $\{R(t),\ t \ge 0\}$. Combining, we get

$$\lim_{k\to\infty} \frac{1}{k} \sum_{\ell=0}^{k-1} H^\star_\ell = \mathbf{E}\left[R(\min(\Delta + T, U_e)-)\right] - \mathbf{E}\left[R(\min(\Delta, U_e))\right] \quad a.s. \tag{4}$$

[13] Only applies when $\Delta > 0$.

To conclude, we report (3) and (4) into (1), and this yields (20) as we recall (.1) with (.2).
□

References

[1] S. Asmussen and M. Bladt, "Renewal theory and queueing algorithms for matrix-exponential distributions," in: *Matrix-Analytic Methods in Stochastic Models*, S. R. Chakravarthy and A. A. Alfa, Eds., Marcel Dekker, New-York (1996), pp. 313-341.

[2] F. Baccelli and P. Brémaud, *Elements of Queueing Theory*, Applications of Mathematics **26**, Springer-Verlag, Berlin Heidelberg (Germany), 1994.

[3] O. Bahat, *Optimization and Evaluation of Service Speed and Reliability in Modern Caching Applications*, Ph.D. Thesis, Department of Electrical and Computer Engineering, University of Maryland, College Park, August 2005.

[4] P. Barford, A. Bestavros, A. Bradley, and M. Crovella, "Changes in Web client access patterns: Characteristics and caching implications," World Wide Web, Vol. 2., No. 1-2, pp. 15 - 28, 1999.

[5] R. E. Barlow and F. Proschan, *Statistics Theory of Reliability and Life Testing*. 1981.

[6] P. Cao and C. Liu, "Maintaining strong cache consistency in the World-Wide Web," in IEEE Transactions on Computers, Vol. 47, no. 4, pp. 445–457, 1998.

[7] E. Cohen and H. Kaplan, "Aging through cascaded caches: Performance issues in the distribution of web content," in Proceedings of SIGCOMM 2001, *Computer Communication Review*, R. Guérin, Ed., col. 31, 4, pp. 41-54.

[8] A. Feldman, "Characteristics of TCP connections arrivals," Technical Report, AT&T Labs-Research, 1998.

[9] M. J. Franklin, M. J. Carey, and M. Livny, "Transactional client-server cache consistency: Alternatives and performance," in ACM Transactions on Database Systems, Vol. 22, no. 3, pp. 315-363, September 1997.

[10] K. Gharachorloo, D. Lenoski, J. Laudon, and P. Gibbons, A. Gupta and J. Hennessy, "Memory consistency and event ordering in scalable shared-memory multiprocessors," in Proceedings of the 17th Annual International Symposium on Computer Architecture, pp. 15-26, May 1990.

[11] J. Gwertzman and M. Seltzer, "World-Wide Web Cache Consistency," in Proceedings of the 1996 USENIX Technical Conference, San Diego (CA), January 1996.

[12] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1*, The Internet Society, June 1999.

[13] Y. Hou, J. Pan, B. Li, X. Tang, and S. Panwar, "Modeling and analysis of an expiration-based hierarchical caching system," in Proceedings of GLOBECOM 2002, November 2002.

[14] J. Jung, A.W. Berger, and H. Balakrishnan, "Modeling TTL-based internet caches," in Proceedings of INFOCOM 2003, San Francisco (CA), March 2003.

[15] J. Kawash, "Consistency models for internet caching," in Proceedings of the Winter International Symposium on Information and Communication Technology, pp 161-166, January 2004.

[16] B. Krishnamurthy and C. E. Wills, "Piggyback server invalidation for proxy cache coherency," in Proceedings of the 7th WWW Conference, April 1998.

[17] B. Krishnamurthy and C. E. Wills, "Study of piggyback cache validation for proxy caches in the World Wide Web," in Proceedings of USENIX Symposium on Internet Technologies and Systems, Monterey (CA), December 1997.

[18] L. Lamport, "How to make a multiprocessor computer that correctly executes multiprocess programs," in IEEE Transactions on Computers, Vol. C-28, no. 9, September 1979.

[19] G. Lorden, "On excess over the boundary," Ann. Math. Stat., Vol. 41, pp. 521 - 527, 1970.

[20] K.T. Marshall, "Linear bounds on the renewal function," SIAM Journal on Applied Math., 24, pp. 245-250, 1973.

[21] V. N. Padmanabhan and L. Qiu, "The content and access dynamics of a busy Web site: Findings and implications," in Proceedings of the ACM SIGCOMM 2000 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Stockholm, Sweden, August 2000.

[22] V. Paxson and S. Floyd, "Wide area traffic: The failure of Poisson modeling," IEEE/ACM Transactions on Networking, Vol. 3, no. 3, pp. 226-244, 1995.

[23] K. Peterson and K. Li, "An evaluation of multiprocessor cache coherence based on virtual memory support," in Proceedings of the 8th International Parallel Processing Symposium (IPPS'94), pp. 158–164, 1994.

[24] J. Rajendiran, J. Patwardhan, V. Abhijit, R. Lakhotia, and A. Vadhat, "Exploring the benefits of a continuous consistency model for wireless Web portals," in Proceedings of IEEE 2nd Workshop on Internet Applications, 2001.

[25] S. Ross, *Stochastic Processes*, Second Edition, Wiley, 1996.

[26] F. J. Torres-Rojas, M. Ahamad, and M. Raynal, "Timed consistency for shared distributed objects," in Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC'99), pp. 163-172, Atlanta (GA), 1999.

[27] D. Wessels, "Squid Internet object cache," available at http://squid.nlanr.net/Squid/, August 1998.

[28] J. Yin, L. Alvisi, M. Dahlin, and A. Iyengar, "Engineering Web cache consistency," in ACM Transactions on Internet Technology, Vol. 2, no. 3, 2002.

[29] L. Yin, G. Cao, and Y. Cai, "A generalized target driven cache replacement policy for mobile environments*," in Proceedings of the 2003 International Symposium on Applications and the Internet, January 2003.