On Expiration-Based Hierarchical Caching Systems

Y. Thomas Hou, *Member, IEEE*, Jianping Pan, *Member, IEEE*, Bo Li, *Senior Member, IEEE*, and Shivendra S. Panwar, *Senior Member, IEEE*

***Abstract*—Caching is an important means to scale up the growth of the Internet. Weak consistency is a major approach used in Web caching and has been deployed in various forms. This paper investigates some fundamental properties and performance issues associated with an expiration-based caching system. We focus on a hierarchical caching system based on the time-to-live expiration mechanism and present a basic model for such system. By analyzing the intrinsic timing behavior of the basic model, we derive important performance metrics from the perspectives of the caching system and end users, respectively. Based on the results for the basic model, we introduce threshold-based and randomization-based techniques to further enhance and generalize the basic model. Our results in this paper offer some important insights in a hierarchical caching system based on the weak consistency paradigm.**

***Index Terms*—Freshness threshold, hierarchy, Internet, randomization, time-to-live (TTL), weak consistency, Web caching.**

I. Introduction

As Van Jacobson once put it a few years ago, "with 25 years of Internet experience, we've learned exactly *one* way to deal with exponential growth: *caching*" [15]. This statement undoubtedly indicates the significance of caching as a primary means to scale up the Internet. There are significant advantages of deploying cache systems over the Internet [20]: 1) for end users, a nearby cache server (CS) can often satisfy requests faster than a faraway origin server (OS); 2) for network providers, the CSs can reduce the amount of traffic over the network; and 3) for service providers, the CSs can distribute the load that the OSs have to handle and increase reliability in offering the Internet services.

An important issue in the design of a caching system is to maintain some level of *consistency* between cached copies of an object and the object maintained at the OS. Every time the original object is updated at the OS, copies of that object cached elsewhere become stale. Caching consistency mechanisms ensure that cached copies of an object are eventually updated to reflect changes to the original object. Depending on how soon the cached copies are updated, cache consistency mechanisms fall into two major categories: *strong consistency* and *weak consistency*.

Manuscript received November 15, 2002; revised June 1, 2003.

Y. T. Hou is with the Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061 USA (e-mail: thou@vt.edu).

J. Pan is with Fujitsu Laboratories of America, Sunnyvale, CA 94086-3922 USA (e-mail: jpan@fla.fujitsu.com).

B. Li is with the Department of Computer Science, Hong Kong University of Science and Technology, Kowloon, Hong Kong (e-mail: bli@cs.ust.hk).

S. S. Panwar is with the Department of Electrical and Computer Engineering, Polytechnic University, Brooklyn, NY 11201 USA (e-mail: panwar@catt.poly.edu).

Digital Object Identifier 10.1109/JSAC.2003.818804

Under strong consistency, upon an update of an object at the OS, the OS immediately notifies all CSs about this update. Example caching applications that require strong consistency include time-sensitive content delivery (e.g., emergency public announcements). The main challenge to strong consistency mechanisms (e.g., invalidation [14]) is that they often involve high overhead and complexity that could be expensive to deploy. Nevertheless, strong consistency is an indispensable approach to deliver mission-critical contents on the Web. On the other hand, under weak consistency, it is acceptable to have a user get a somewhat stale object from a CS. A CS only validates an object's freshness with the OS periodically and may lag behind the actual update at the OS. Weak consistency is particularly useful for those web contents that can tolerate a certain degree of discrepancy between cached content and the content at OS as long as it is understood that such discrepancy in time does not cause any harm. Still, it is important to keep such discrepancy not to exceed a reasonable period of time. Example applications using weak consistency include online newspapers and magazines, personal homepages, and the majority of web sites—although the original content may be further updated at the OS, it is still useful (or at least not harmful) to retrieve a cached copy at a cache or proxy server.[1] It has been shown that weak consistency is a viable and economic approach to deliver content that does not have a strict freshness requirement [14].

To support weak consistency, the concept of time-to-live (TTL) is introduced. TTL is an *a priori* estimation of an object's remaining lifetime and can be used to determine how long a cached object remains useful. Under the TTL approach, each object is initialized with a TTL value and TTL is supposed to decrease with time when the object is cached. An object that has been cached longer than its initial TTL is said to *expire* and the next request for this object will cause the object to be requested (or validated) from the OS or some other CSs that have a copy with an unexpired TTL. In practice, the TTL-based strategy is easy to implement (e.g., by using the "expires" or "last-modified" fields in HTTP header [20]).

There are many alternative approaches to construct a caching infrastructure. However, it has been shown [9] that a *hierarchically* organized caching infrastructure is particularly effective to scale up with the Web growth since the Internet topology also tends to be organized hierarchically. In light of this, in this paper, we focus on a hierarchy-based caching system. We conduct an investigation of its performance and behavior under the weak consistency paradigm, which employs the TTL-based expiration mechanism. Although the current HTTP protocols on Web caching provide a lot of similar features [11], we intend to

[1] A user always has the option to reload the fresh content from the OS if he/she prefers to have the most updated copy of the object.

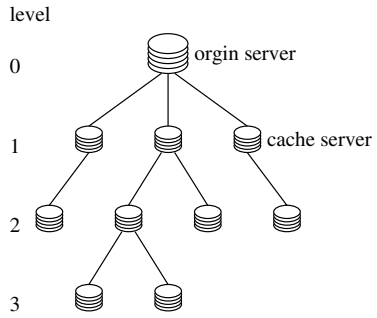


Fig. 1. Example of a hierarchical caching system based on tree topology.

conduct our investigation on weak consistency in a more general setting and will not limit ourselves to the details of the HTTP implementation. Our objectives in this paper are threefold: 1) to provide some fundamental understanding of the behavior and performance of a weak-consistency-based hierarchical caching system, which is yet not well studied and understood; 2) to demonstrate quantitatively the efficacy of such hierarchical caching system; and 3) to provide the necessary guidelines and insights to service providers on various tradeoffs on such hierarchical caching systems.

We start with a basic model, which is a generic hierarchical caching system based on tree topology. Under the basic model, the root node represents the OS whereas all the other nodes in the tree represent CSs (see Fig. 1). We assume each node (or CS) is deployed in a metropolitan region and the user requests[2] within this particular metro region always goes to this regional CS for content service. When the object is not available or its TTL has expired at a CS, the CS will query its immediate parent CS, which may further query its immediate parent CS and so forth, until a "fresh" copy of the object is retrieved or the OS is reached. Here, a "fresh" copy is defined as a copy of the object with an unexpired (i.e., positive) TTL. The OS always maintains an updated copy of the object and will initialize the TTL of an object upon request. The TTL value for an object at any CS decreases with time. Since TTL is a fundamental parameter that determines the intrinsic behavior of the overall hierarchical caching system, we analyze the behavior of TTL at each level of CS under a tree structure. Based on this analysis, we conduct a performance study for the hierarchical caching system from the perspectives of both the caching system and end users by deriving performance metrics such as hit rate, miss rate, response time, and network load. We use simulation results to substantiate the accuracy of our analysis and provide insights on various system design tradeoffs.

Based on our understanding of the basic model, we further explore possible enhancement of the basic model along several directions. One possible direction for enhancement is to introduce TTL threshold from user and CS perspectives. The motivation for introducing threshold is to give further flexibility to users and the cache system: users can specify their content freshness requirement and CSs may put a higher demand on the TTL of objects that they retrieve. A second direction to enhance the basic model is to introduce randomization into the caching system, which is motivated by observing certain undesirable behavior of the basic model (e.g., inherent miss synchronization) during simulation studies. It turns out that both enhancement can also be regarded as the generalization of the basic model. The results presented in this paper is general and can be applied to expiration-based caching systems for other application domains.

[2]Note that a user may also have a browser cache built in its host. Here, we only consider those user requests sent to the *proxy CS* by the user after a miss at its own browser cache. That is, we only consider the "effective" request sent to the proxy CS from the user and not consider those requests that can be served by the user's own browser cache.

TABLE I
NOTATIONS

Symbol	Definition
H	Height of tree for the hierarchical caching system
h	Level number of a cache server within the tree, $0 \le h \le H$
$\mathcal{S}_h$	A level-h cache server
$r_h(t)$	Remaining TTL at a level-h cache server at time t
T_h	Peak value of $r_h(t)$ during a renewal period
τ	Maximum TTL value initialized by the origin server $\mathcal{S}_0$
$E(T_h)$	Average value of T_h
I_h	Idle time during a renewal period at a level-h cache server
Λ_h	Aggregate Poisson request rate at a level-h cache server
λ_h	Poisson request rate at a level-h cache server
Γ_h^s	System miss rate at a level-h cache server
Γ_h^u	User perceived miss rate at a level-h cache server
Θ_h^s	System hit rate at a level-h cache server
Θ_h^u	User perceived hit rate at a level-h cache server
σ_h^s	System response time at a level-h cache server
σ_h^u	User perceived response time at a level-h cache server
C_h	Number of child cache servers for a level-h cache server
M_h	Number of requests a level-h cache server receives from its child cache servers during a renewal period, $0 \le M_h \le C_h$
d_h	Round trip time between a level-h cache server and its immediate parent cache server
ρ_h^s	Average load (request rate) from a level-h cache server to its parent cache server
α	User required content freshness threshold
β	Cache server prefetching threshold
p_i	Probability distribution used for randomization by a level-h cache server, $\sum_{i=0}^{h-1} p_i = 1$
r	Geometric parameter used for probability distribution

The remainder of this paper is organized as follows. In Section II, we present the basic model for the hierarchical caching system based on the weak consistency paradigm. We also analyze the TTL behavior of the basic model and derive its performance metrics. In Section III, we present simulation results to substantiate our analytical results for the basic model. In Section IV, we extend the basic model by introducing threshold and randomization techniques. Section V discusses related work and Section VI concludes this paper.

II. System Modeling and Analysis

In this section, we describe the basic model under our study and analyze its performance behavior. Table I summaries notations that will be used throughout the paper.

A. Basic Model

In a TTL-based caching system, each cached object is associated with a TTL value, which was first initialized to maximum lifetime by the OS where the object was retrieved. The TTL value decreases with time and the object expires when TTL reaches zero. A cached object is considered *fresh* when its TTL is positive; otherwise, it is considered *stale.*

We assume the hierarchical caching system follows a tree structure (see Fig. 1). Typically, each server is deployed within

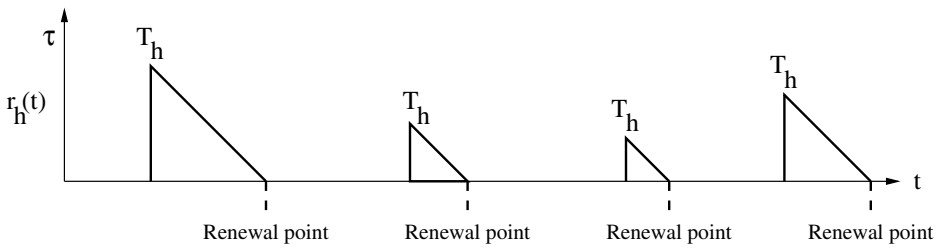


Fig. 2. Sample path of the TTL for a level-h CS.

a metro region. At level 0, we have an OS[3] $\mathcal{S}_0$, which always maintains the latest (updated) copy of an object. The OS is logically connected to some child servers which we refer to as level-1 CSs, each of which are geographically located in different metro regions. A level-1 CS may also connect to some child servers which we call level-2 CSs, and so forth. Finally, a CS that does not have any child CS is called a leaf CS. The maximum number of levels of such hierarchical tree is called the height of the tree. Fig. 1 shows a simple example of our caching system based on a tree structure with a height of three.

We assume that the aggregate user requests to the CS within a metro region follow a Poisson process.[4] When a user request arrives at the CS, if the object already exists at the CS and its TTL is still greater than zero, the CS will deliver the object directly to the user. We consider such event a *user hit.* On the other hand, when the user request arrives at the local CS, if the object does not exist or the TTL timer has expired (i.e., decreased to zero), we consider such event a *user miss.* When a miss happens, the CS will generate a request and query its immediate parent CS to see if it has the object with a valid TTL. If its parent CS does have this object with an unexpired TTL, we call this event a *system hit* since the request is generated by a child CS rather than *directly* from a user. Upon a system hit, the object will be delivered to the CS and will subsequently be delivered to the user. Otherwise, we have a *system miss* and the parent CS will generate a request and further query its own parent CS and so forth, until the query process reaches the OS, in which case we assume that the OS always maintains an updated fresh copy of the object. The OS will deliver the object with a TTL field initialized to maximum lifetime τ, where $\tau > 0$, and the TTL value decreases linearly as time goes on. Thus, the maximum *age* that an object (delivered to a user) can have under such hierarchical caching system is bounded by τ. Under the basic model, upon the event of a system hit, not only the user will be delivered with a copy of the object with an updated TTL, *all* the CSs involved in the query process will also get a copy of this object with an updated TTL.

Note that we distinguish hit rate and miss rate from user and cache system perspectives. Such distinction will help us better understand the details of the system behavior, as we shall soon find out.

[3]Note that the OS may consist of a cluster of servers. But we assume that they all locate at the same site (e.g., an Internet service providers (ISP)'s data center).

[4]An exact traffic model for individual user is still an open research topic. However, it is reasonable to assume that, for a large population in a metro region, the aggregate requests follow a Poisson process.

B. Performance Analysis

In this section, we investigate the performance of the basic model. We conduct performance evaluation along two dimensions: *caching system performance* and *end user quality of experience.* By caching system's performance, we refer to the behavior and properties of the hierarchical caching structure, such as the aggregated behavior of TTL, miss rate, hit rate, average response time, and traffic load at each CS.[5] On the other hand, user's quality of experience refers to user's perceived quality in content delivery, e.g., hit rate, miss rate, average response time, which only counts requests from end users and does not include auxiliary traffic within the hierarchical caching system.

1) Average TTL Behavior Analysis: Suppose we are at a level-h CS, $1 \leq h \leq H$, where H is the height of the tree. Denote the (remaining) TTL at the CS as $r_h(t)$. Then, $r_h(t)$ is a renewal process [21], with the renewal point starting at time $t = t_*$ when $r_h(t)$ just decreases to 0, i.e., $r_h(t_*) = 0$ and $r_h(t_*^-) > 0$ (see Fig. 2). It should be clear that when $r_{h^*}(t) = 0$, then for all $h^* \leq h \leq H$, $r_h(t) = 0$. This is because that under the basic model, an object maintained at a parent CS always has its remaining TTL larger than or equal to the remaining TTL of the same object maintained at its child CSs.

Referring to Fig. 2, denote the *peak* value of $r_h(t)$ during each renewal period as T_h, $1 \leq h \leq H$. Then, we have $T_1 = \tau$ and T_h is a random variable defined over $(0, \tau]$ for $2 \leq h \leq H$. We are interested in the average value of T_h, for $2 \leq h \leq H$, denoted as $E(T_h)$, which is a fundamental system parameter in the basic model.

Due to the nature of the hierarchical tree and TTL-based expiration, there is an important property on TTL that *links* the CSs at all levels. In particular, any TTL renewal point at level h, $2 \leq h \leq H$ (see Fig. 2) coincides (or *synchronizes*) with a renewal point at level 1. However, the converse is not true, i.e., a renewal point at level 1 may not be a renewal point at level h, $2 \leq h \leq H$. This is because that the smaller the h for a CS, the more child servers (and, thus, user population) it will support, which translates into a smaller *idle* period I_h (the time period when $r_h(t)$ remains 0). This observation leads to the fact that the average renewal period at each level increases with h, with the smallest at level 1 and the largest at level H.

Referring to Fig. 3, for each renewal period at level 1, it is clear that the first request that initiates the TTL triangle within the renewal period follows a Poisson process with rate

[5]By *aggregated*, we count both external user requests, as well as internal requests from child CSs.

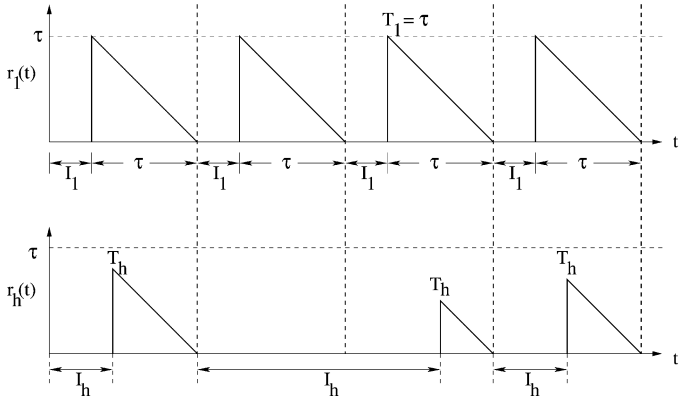


Fig. 3. Sample path of the TTL behavior for a level-1 and level-h CSs.

Λ_1, which is the sum of *all* Poisson arrival rates at all CSs of the subtree with $\mathcal{S}_1$ as the root. This Poisson process (with rate Λ_1) can be considered as an *aggregate* of two Poisson processes: the first with a rate of Λ_h representing the arrivals at the subtree with $\mathcal{S}_h$ as the root and the second (with a rate of $\Lambda_1 - \Lambda_h$) representing arrivals from the rest of the tree within $\mathcal{S}_1$ excluding the subtree $\mathcal{S}_h$. Clearly, the probability that the TTL triangle is initiated by a request from the subtree with root $\mathcal{S}_h$ is Λ_h/Λ_1 and the probability that the TTL is initiated by a request from the rest of tree (i.e., $\mathcal{S}_1 \backslash \mathcal{S}_h$) is $(\Lambda_1 - \Lambda_h)/\Lambda_1$.

We now look at the time interval at level h that corresponds to the same renewal period at level 1. There are three cases and the sum of probabilities of these three cases is 1.

Case 1) With probability Λ_h/Λ_1, the TTL triangle at level 1 is initiated by a request from the subtree with root at $\mathcal{S}_h$. In this case, $T_h = \tau$.

Case 2) With probability $(\Lambda_1 - \Lambda_h)/\Lambda_1 \cdot \int_0^\tau \Lambda_h e^{-\Lambda_h t} dt = (\Lambda_1 - \Lambda_h)/\Lambda_1 \cdot (1 - e^{-\Lambda_h \tau})$, the TTL triangle at level 1 is *not* initiated by a request from the subtree with root at $\mathcal{S}_h$ *and* there is a request arrival in the same renewal period from the subtree with root $\mathcal{S}_h$. The average T_h in this case is given by

$$\frac{\int_0^\tau (\tau - t)\Lambda_h e^{-\Lambda_h t} dt}{\int_0^\tau \Lambda_h e^{-\Lambda_h t} dt} = \frac{\tau - \frac{1}{\Lambda_h} + \frac{1}{\Lambda_h} e^{-\Lambda_h \tau}}{1 - e^{-\Lambda_h \tau}}. \quad (1)$$

It can be shown that the average T_h in this case [i.e., the right side of (1)] is always greater than $\tau/2$ (see proof of Property 1 in the Appendix). This can be intuitively explained by the Poisson property of the arrival process.

Case 3) With probability of $(\Lambda_1 - \Lambda_i)/\Lambda_1 \cdot \int_\tau^\infty \Lambda_h e^{-\Lambda_h t} dt = (\Lambda_1 - \Lambda_h)/\Lambda_1 \cdot e^{-\Lambda_h \tau}$, there is no request arrival in this interval. In this case, there is no TTL triangle at level h in this interval.

To calculate $E(T_h)$, all we need to do is to take the probabilistically weighted average of T_h under Cases 1 and 2. We have

$$E(T_h) = \frac{\frac{\Lambda_h}{\Lambda_1} \cdot \tau + \frac{\Lambda_1 - \Lambda_h}{\Lambda_1} \cdot (1 - e^{-\Lambda_h \tau}) \cdot \frac{\tau - \frac{1}{\Lambda_h} + \frac{1}{\Lambda_h} e^{-\Lambda_h \tau}}{1 - e^{-\Lambda_h \tau}}}{\frac{\Lambda_h}{\Lambda_1} + \frac{\Lambda_1 - \Lambda_h}{\Lambda_1} \cdot (1 - e^{-\Lambda_h \tau})}$$
$$= \frac{\Lambda_h \tau + (\Lambda_1 - \Lambda_h)(\tau - \frac{1}{\Lambda_h} + \frac{1}{\Lambda_h} e^{-\Lambda_h \tau})}{\Lambda_h + (\Lambda_1 - \Lambda_h)(1 - e^{-\Lambda_h \tau})}. \quad (2)$$

Property 1: Under the basic model, the average of T_h at level h, $h = 1, 2, \ldots, H$, has the following property:

$$\tau \geq E(T_h) > \frac{\tau}{2} \quad (3)$$

and

$$E(T_1) > E(T_2) > \cdots > E(T_h) > \cdots > E(T_H). \quad (4)$$

The proof is given in the Appendix.

A Special Case: In the following, we consider the special case when $\tau \gg 1/\lambda_H$. This may correspond to access behavior for most popular contents, as popular objects typically receive much more accesses than other unpopular objects [2].[6] Under this case, we assume that with probability 1, there is at least one request from the user at the leaf level CS during each renewal period at level 1.[7] Under this scenario, the TTLs renewal period at each level h, $2 \leq h \leq H$ would be perfectly synchronized with that at level 1 (see Fig. 4).

Note that within the *same* renewal period, T_h is in a nonincreasing order, i.e., $T_1 \geq T_2 \geq \cdots \geq T_{h-1} \geq T_h \geq \cdots \geq T_H$, where $T_1 = \tau$. Each renewal period at level h, $h = 1, 2, \ldots, H$, starts with a Poisson arrival process with a rate Λ_h, which equals to the sum of all the Poisson rates of the *subtree* with this CS as its root.[8] The idle process ends with a Poisson arrival and the remaining TTL at this CS decreases with rate of 1 with time until it reaches 0 (i.e., the end of the period).

We are interested in finding the properties of T_h, $h = 1, 2, \ldots, H$. At level 1, we have $T_1 = \tau$. To find T_h for $h = 2, 3, \ldots, H$, we observe that $X_h = X_1$ (see Fig. 4), or $E(X_h) = E(X_1)$. But $E(X_h) = (1/\Lambda_h) + E(T_h)$, and $E(X_1) = (1/\Lambda_1) + \tau$. We have the following result for $\tau \gg 1/\lambda_H$, $h = 1, 2, \ldots, H$

$$E(T_h) = \tau - \left(\frac{1}{\Lambda_h} - \frac{1}{\Lambda_1} \right). \quad (5)$$

This result can also be verified directly by using the general form of $E(T_h)$ in (2). That is, given that $\tau \gg 1/\lambda_H$, or $\lambda_H \tau \gg 1$, we have $\Lambda_h \tau \gg 1$ since $\Lambda_h \geq \lambda_H$, $h = 1, 2, \ldots, H$. When $\Lambda_h \tau \gg 1$, $e^{-\Lambda_h \tau}$ becomes negligible and

$$E(T_h) = \frac{\Lambda_h \tau + (\Lambda_1 - \Lambda_h)\left(\tau - \frac{1}{\Lambda_h} + \frac{1}{\Lambda_h} e^{-\Lambda_h \tau}\right)}{\Lambda_h + (\Lambda_1 - \Lambda_h)(1 - e^{-\Lambda_h \tau})}$$
$$\simeq \frac{\Lambda_h \tau + (\Lambda_1 - \Lambda_h)\left(\tau - \frac{1}{\Lambda_h}\right)}{\Lambda_h + (\Lambda_1 - \Lambda_h)}$$
$$= \tau - \left(\frac{1}{\Lambda_h} - \frac{1}{\Lambda_1} \right).$$

2) Performance Metrics: We distinguish the performance metrics along two dimensions: system performance and user

[6] In particular, it has been shown that the object popularity conforms to a *Zipf-like* power law distribution [4], [5].

[7] This constraint also reflects the general design principle for tuning the value τ in a hierarchical caching system to ensure the leaf CS will serve more than one request during a renewal cycle.

[8] For simplicity in our analysis (i.e., to preserve Poisson property), we ignore impact of the round trip time on the Poisson property here. Note that the typical setting of maximum TTL (τ) is much larger than network round trip time (e.g., τ is on the order of 24 h [10], while network round trip time is of the order of 100 ms).

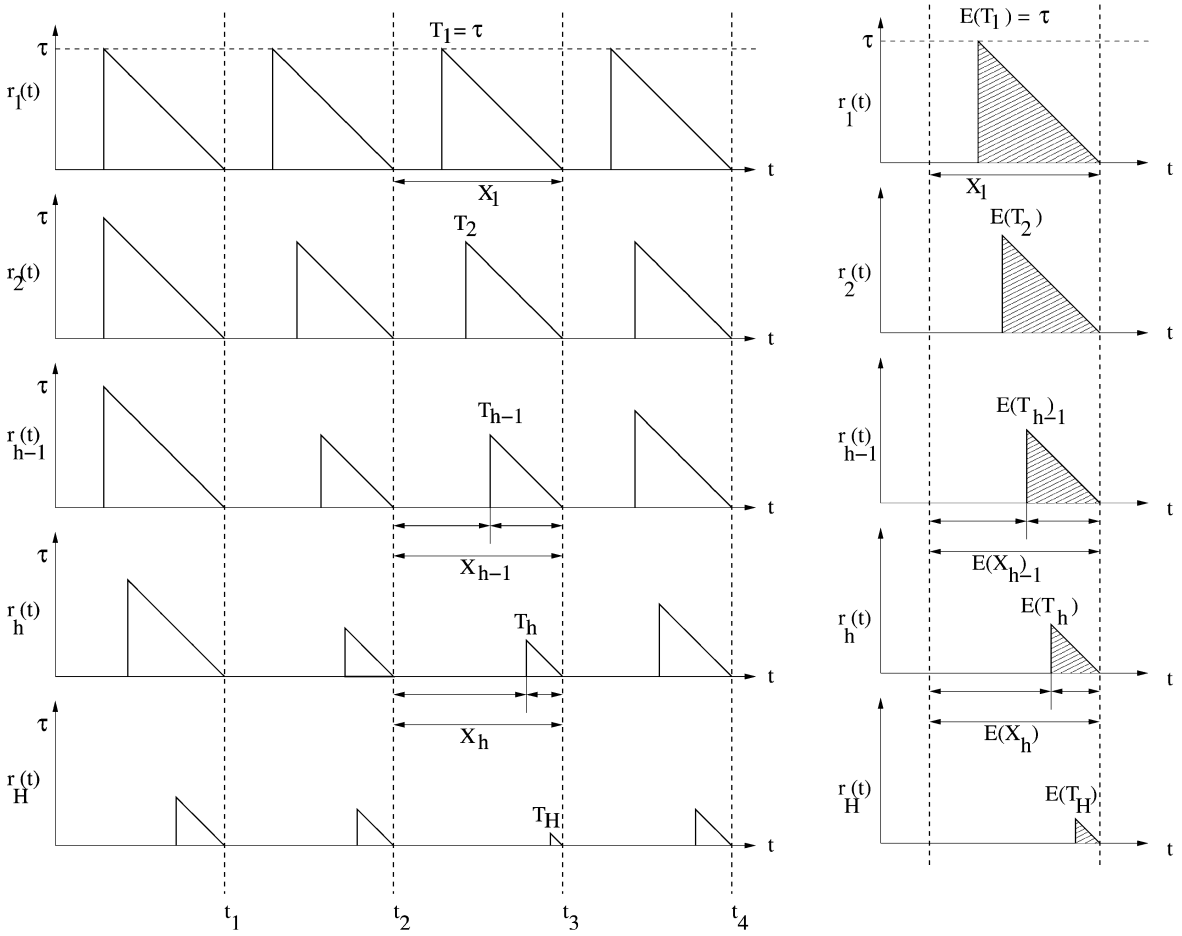


Fig. 4. Timing diagram illustrating properties of remaining TTL at different levels of the tree hierarchy when $\tau \gg 1/\lambda_H$. (a) A sample path demonstrating that the renewal points, t_1, t_2, t_3, and t_4, are synchronized at each level. (b) Averaged behavior.

perceived performance. Denote Γ_h^s, Θ_h^s, and σ_h^s as the system miss rate, hit rate, and response time at a level-h CS, respectively. The system miss rate, hit rate, and response time take into account of all requests, both from the users in the (local) metro region and from child CSs (internal dynamics within the hierarchical caching tree). Similarly, denote Γ_h^u, Θ_h^u, and σ_h^u as the user perceived miss rate, hit rate, and response time at a level-h CS, respectively. The user perceived performance parameters consider only requests generated by the users in the local metro region and do not consider those requests forwarded from any child CSs.

Before we calculate the miss rate Γ_h^s at level h, we make the following observation [see Fig. 4(b)]: each CS can make at most one request to its parent CS during any renewal cycle. Denote M_h the number of request a CS at level h receives from its child CSs during a renewal cycle and c_h the number of child CSs of this server at level h. Then, M_h is a random variable defined over $0, 1, \ldots, c_h$ and the probability distribution of M_h is a combinatorial of exponential distributions since user requests at a CS of any level follows a Poisson distribution. Therefore, $E(M_h)$ can be easily calculated explicitly using combinatorics and $E(M_h) \leq c_h$.

To calculate the miss rate, Γ_h^s, we condition on whether the first miss (i.e., the request that initiates the TTL triangle) is from a user within the CS's metro region or from a child CS. We have

$$\Gamma_h^s = \frac{\lambda_h}{\Lambda_h}\left\{\frac{1}{1+[\lambda_h \cdot E(T_h)+E(M_h)]}\right\} + \frac{\Lambda_h - \lambda_h}{\Lambda_h}\left\{\frac{1}{1+[\lambda_h \cdot E(T_h)+E(M_h)-1]}\right\}. \quad (6)$$

Note that for a constant λ_h and c_h at each level h, the miss rate increases as the level h increases.

The system's hit rate $\Theta_h^s = 1 - \Gamma_h^s$, is then

$$\Theta_h^s = \frac{\lambda_h}{\Lambda_h}\left\{\frac{\lambda_h \cdot E(T_h)+E(M_h)}{1+\lambda_h \cdot E(T_h)+E(M_h)}\right\} + \frac{\Lambda_h - \lambda_h}{\Lambda_h}\left\{\frac{\lambda_h \cdot E(T_h)+[E(M_h)-1]}{1+\lambda_h \cdot E(T_h)+[E(M_h)-1]}\right\}. \quad (7)$$

Denote d_h as the round trip time (including processing delay at the CS) between a child CS at level h and its immediate parent CS, and assume the delay between an end user and its local CS is negligible. The average system response time σ_h^s is

$$\sigma_h^s = \Theta_h^s \cdot 0 + \Gamma_h^s \cdot \pi_h \quad (8)$$

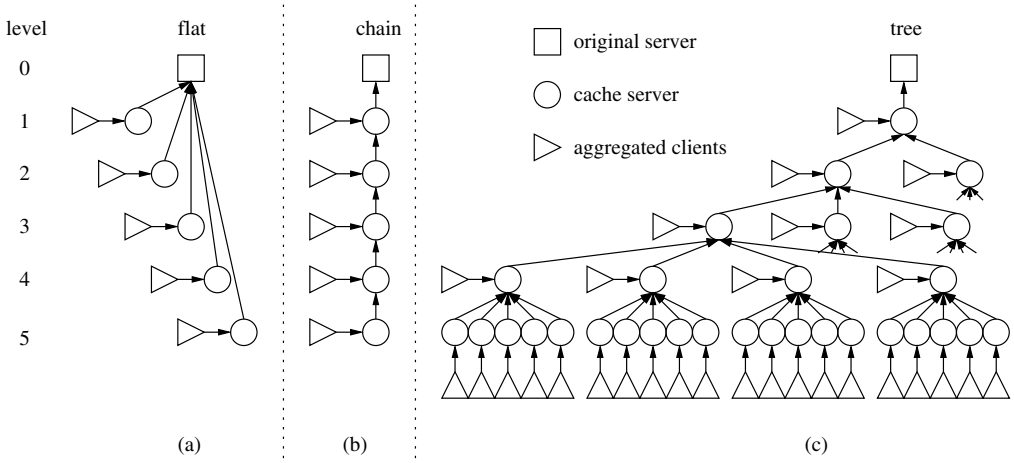


Fig. 5. Topologies of caching systems used in the simulation study: (a) flat structure, (b) chain topology, and (c) tree topology.

where π_h is delay until getting a fresh object given that there is a miss at the local CS. From (8), we have

$$\pi_h = \frac{\sigma_h^s}{\Gamma_h^s}. \tag{9}$$

On the other hand

$$\pi_h = d_n + \left\{ \frac{\Lambda_{h-1} - \Lambda_h}{\Lambda_{h-1}} \cdot 0 + \frac{\Lambda_h}{\Lambda_{h-1}} \cdot \pi_{h-1} \right\}. \tag{10}$$

Combining (8)–(10), we have the following recursive relationship for σ_h^s:

$$\begin{aligned} \sigma_h^s =& \Gamma_h^s \cdot \left(d_h + \frac{\Lambda_h}{\Lambda_{h-1}} \cdot \pi_{h-1} \right) \\ =& \Gamma_h^s \cdot \left(d_h + \frac{\Lambda_h}{\Lambda_{h-1}\Gamma_{h-1}^s} \cdot \sigma_{h-1}^s \right) \end{aligned} \tag{11}$$

with $\sigma_1^s = \Gamma_1^s \cdot d_1$.

We now calculate the user perceived hit rate (Θ_h^u), miss rate (Γ_h^u), and response time (σ_h^u), at a level-h CS. These performance metrics will be slightly different from those corresponding to the system performance. This is because we need to filter out the effect of the requests from child CSs (which represent internal dynamics of the hierarchical caching system). Again, by conditioning on whether the first request comes from local users or child CSs, we have

$$\Theta_h^u = \frac{\lambda_h}{\Lambda_h} \left\{ \frac{\lambda_h \cdot E(T_h)}{1 + \lambda_h \cdot E(T_h)} \right\} + \frac{\Lambda_h - \lambda_h}{\Lambda_h} \cdot 1. \tag{12}$$

As $\Gamma_h^u = 1 - \Theta_h^u$, we have for the users' miss rate

$$\Gamma_h^u = \frac{\lambda_h}{\Lambda_h} \left\{ \frac{1}{1 + \lambda_h \cdot E(T_h)} \right\}. \tag{13}$$

The response time a user experiences is

$$\sigma_h^u = \Theta_h^u \cdot 0 + \Gamma_h^u \cdot \pi_h = \Gamma_h^u \cdot \left(d_h + \frac{\Lambda_h}{\Lambda_{h-1}\Gamma_{h-1}^s} \cdot \sigma_{h-1}^s \right) \tag{14}$$

with $\sigma_1^u = \Gamma_1^u \cdot d_1$.

3) Network Traffic Load: So far, we have calculated the hit rate, miss rate, and response time from both the system's and user's perspectives. There is one more important system performance metric that we want to include. This is the traffic load associated with the hierarchical caching system. As we discussed earlier, one of the major benefits of a caching system is to reduce the overall network traffic load and, thus, to achieve *scalability* as the Internet continues to grow. Here, we calculate the network load associated with the hierarchical caching system and in Section III, we will further demonstrate the scalability property of the hierarchical caching system comparing to a nonhierarchical (or "flat") caching system.

One way to measure network traffic load for the hierarchical caching system is to perform an accounting on how much traffic each CS generates to its immediate parent CS. Note that a CS will initiate a request to its parent CS only when a request (either from local users in the metro region or from child CSs) incurs a miss. Denote the average request rate that a CS $\mathcal{S}_h$ at level h sends to its parent CS as ρ_h^s. By definition, we have

$$\rho_h^s = \frac{1}{E(I_h + T_h)} \tag{15}$$

where I_h is the idle period during a renewal cycle for a CS at level h and $E(I_h) = 1/\Lambda_h$.

III. Simulation Investigation

In this section, we use simulation results to demonstrate the intrinsic properties of a hierarchical caching system. We also compare its performance with a nonhierarchical caching systems (i.e., those with a flat structure). The objectives of this study are to validate our analysis in Section II and to offer further insights on the dynamics of such caching systems.

A. Simulation Settings

Our simulation is built on the network simulator *ns–2* platform [19]. We define three new objects, namely, OS, CS, and aggregated clients (ACs) as follows. An OS is a reply-only object which always returns the requested object with its TTL value initialized to τ. An AC is a request-only object with a rate of λ. For each CS and AC objects at each level of the hierarchy, we attach a log to them in the simulation to record all requests and replies events, which will be used for off-line data processing.

Fig. 5 shows the three topologies of caching systems used in our simulation study. Under the "flat" topology, each CS can only make requests directly to the OS and, thus, the level number

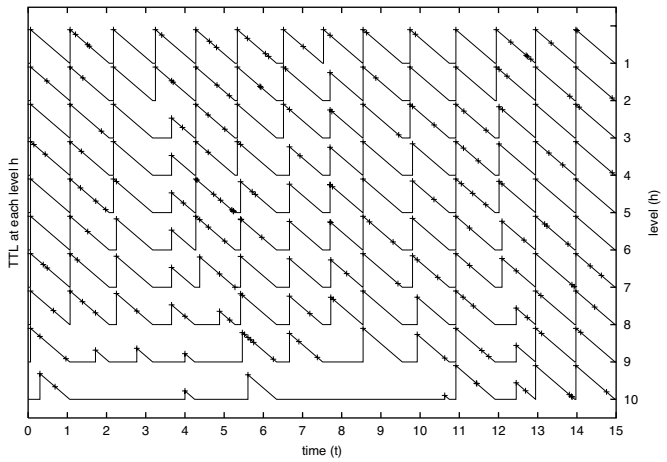


Fig. 6. Sample path of TTL evolution at level h, $1 \leq h \leq 10$ under the chain topology.

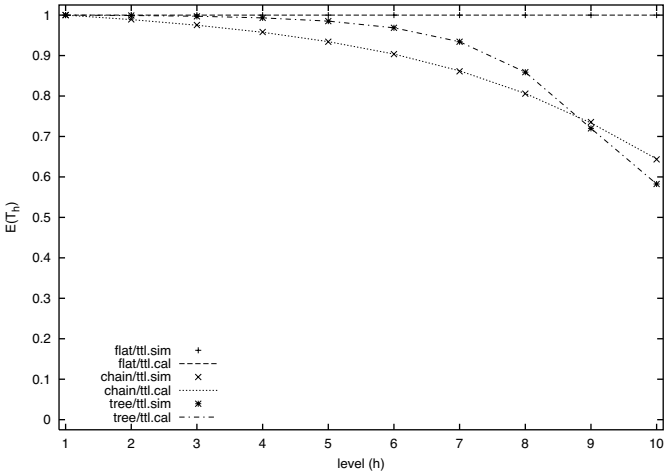


Fig. 7. $E(T_h)$ behavior at each level h, $1 \leq h \leq 10$ for the flat, chain, and tree topologies.

only represents distance between a CS and the OS. On the other hand, the hierarchical "chain" and "tree" topologies represent two scenarios for the basic model. Note that the chain topology is a special case for the tree topology with a span of one. For the tree topology, we consider a binary *complete* tree (with a span of two for all nonleaf CSs). For all three topologies, we set the maximum number of levels $H = 10$. We use the round trip time between two consecutive levels as a measure of distance between two consecutive levels and set the round trip time between two consecutive levels to two units in our simulation. As an example, under the flat topology, a CS corresponding to level 4 CS under the chain or tree topology will have eight units of round trip time between itself and the OS.

In our simulation, we set $\tau = 1$ unit time[9] and $\lambda_h = 1$ per τ for all h (i.e., same user request rate at each level) unless otherwise stated explicitly.

B. Simulation Results and Discussions

We organize our presentation as follows. First, we examine the TTL behavior (i.e., $E(T_h)$) for CS at each level, which is the most important parameter in characterizing the dynamics of the hierarchical caching system. Then, we present simulation results for the performance metrics from both CSs and user's perspectives. This is followed by a study of traffic load generated by the CSs. Finally, we examine how the user request patterns can affect the performance of the caching system.

1) TTL Behavior: To get a clear picture on how TTL behaves, we first present a set of simulation results showing the TTL behavior at different levels under the chain topology. Fig. 6 shows the sample TTL evolution (in connected lines) and requests (cross points) at each level during a time window in the simulation. As expected, each TTL triangle falls within its parent CSs TTL triangle, with each renewal point when TTL decreases to zero being synchronized to a renewal point of its parent CS. The closer a CS to the OS in the hierarchy, the denser the TTL triangle within the time window.

We now examine the TTL behavior quantitatively and compare it with our closed-form result in Section II-B. In Fig. 7, we plot the calculated average TTL, $E(T_h)$ [using (2)] and $E(T_h)$ from simulations at each level CS for the flat, chain, and tree topologies, respectively. The connected lines represent the calculated results from our analysis, while the disconnected points shows the results extracted from the simulations. Clearly, our analysis for $E(T_h)$, $1 \leq h \leq H$ statistically matches the actual simulations in all cases. With the flat structure, each CS always gets the object with the maximum TTL ($\tau = 1$) (at the expense of larger response time and traffic load). Under the chain or tree structure, as expected, the average TTL $E(T_h)$ at a CS is always lower than that in the flat structure. A smaller TTL implies that, when an object is delivered to the a user or child CS, the object is of less freshness but remains valid under the weak consistency paradigm. Also, we observe that the TTL behavior for both the chain and tree topologies follow Property 1, i.e., for $h = 1, 2, \ldots, 10$, $1 \geq E(T_h) > 1/2$ and $1 = E(T_1) > E(T_2) > \cdots > E(T_{10})$.

2) Performance Metrics: We now present results from analysis and simulations for the performance metrics from both CSs and user's perspectives. In particular, we show (in Fig. 8) the hit rate[10] and response time (delay) from a CSs (i.e., system) and user's perspectives.

In Fig. 8(a), we plot the hit rate at a CS from both analysis [(7)] and simulation results for the flat, chain, and tree topologies. Clearly, our analysis matches simulation results very well. For the flat structure, the hit rate is the same for all CSs at all levels (i.e., 0.5) since each CS interacts with the OS directly and independently from other CSs. On the other hand, the hit rate under chain or tree topology exhibits nonincreasing behavior. Furthermore, the CS hit rate at most levels (except the leaf level) for the tree is higher than that for a chain, and the hit rate for a chain is higher than that for the flat structure. This demonstrates the effect of request aggregation and object sharing under a hierarchical caching system. That is, a CS (except the leaf CS) under the tree topology handles a higher volume of requests than a CS at the same level under the chain or flat topology. At the leaf CS,

[9]The scale of time unit for TTL and round trip time (between a child CS and its parent CS) is different quantitatively.

[10]Since miss rate is the complement of the hit rate, we omit to present its simulation results to conserve space in the paper.

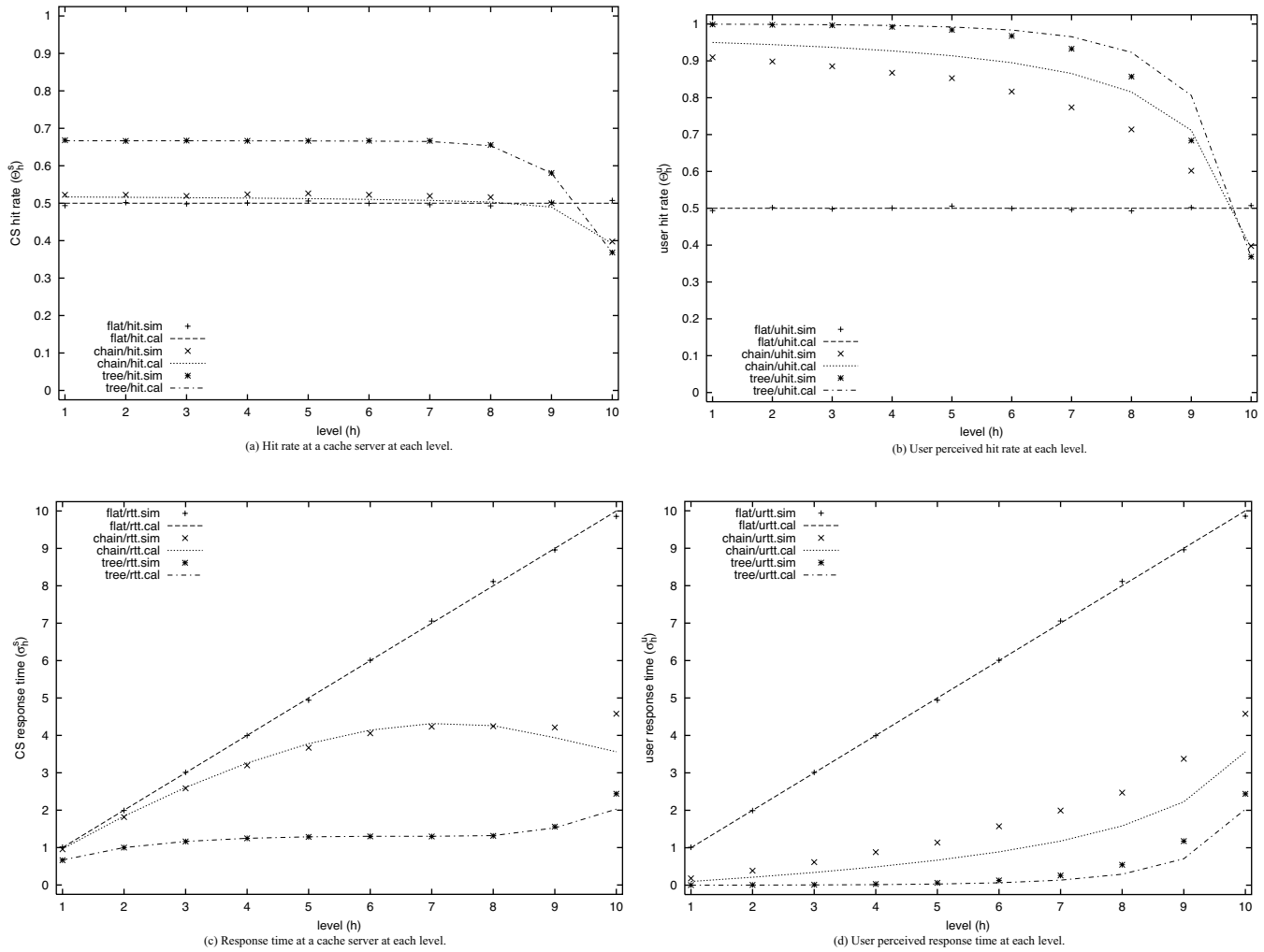


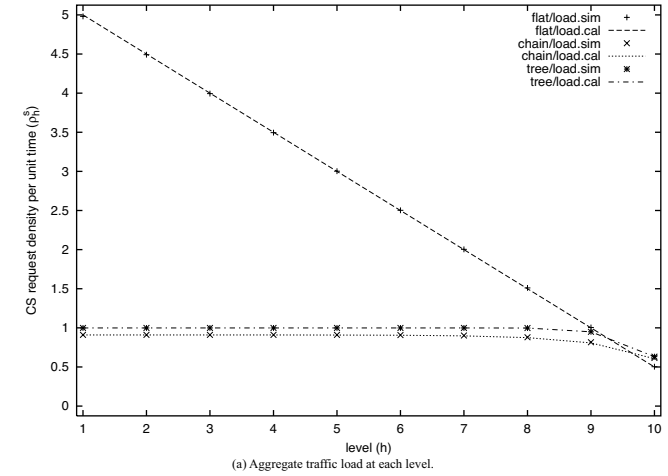
Fig. 8. Hit rate and response time from caching system's and user's perspectives at each level under the flat, chain, and tree topologies.

due to a smaller TTL and less request aggregation and object sharing, the hit rate under chain or tree topology is even lower than that under the flat structure. This shows that the system favors CS closer to the OS and penalizes CS close to leaf CS, which is an intrinsic limitation of a hierarchical caching system. In Section IV-B, we show that this problem can be alleviated by introducing the *randomization* technique.

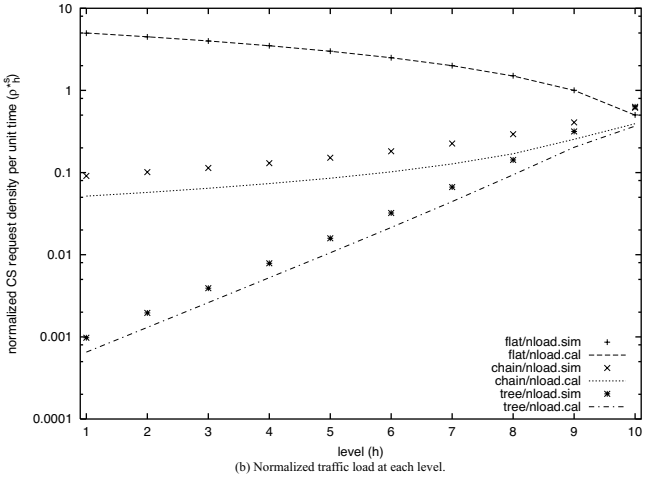
In Fig. 8(b), we plot the hit rate experienced by a user at each level through both analysis [in (12)] and simulation results for the flat, chain, and tree topologies. We observe similar hit rate behavior from user's perspective. Comparing Fig. 8(b) to (a), we find that, under the chain or tree topology, a CSs hit rate is always less than the user's hit rate. This is because once there is a miss at a CS for a particular user request, this request may trigger *multiple* misses at CS(es) along the upstream path toward the OS, which leads to a higher CS miss rate (or lower CS hit rate). In Section IV-B, we will introduce a randomization technique to reduce such multiple miss phenomenon. We also find that, in Fig. 8(b), the hit rate (as well as the response time to be discussed shortly) for the leaf CS is the same as that for the user's under all topologies. This can be easily explained by the fact that a leaf CS can only have requests from users (i.e., no child CS below it).

Fig. 8(c) and (d) shows the response time for a request from a CS and user's perspectives at each level, respectively. The response time shown in the figure is in unit of time with the round trip time between two consecutive levels of CS being 2 units (see Section III-A for our simulation settings). Therefore, the average response time is proportional to (more precisely, twice) the average number of levels [or CS(es)] that a request needs to travel (query) in order to get a hit.

In Fig. 8(c), under the flat structure, the response time increases linearly as the level increases, which is expected. Since the hit rate is 0.5 based on our simulation settings (i.e., $\tau = 1$ and $\lambda_h = 1$ for all levels), the distance (measured in time) between the leaf CS (at level 10) and the OS is 20 units, the average response time for a leaf CS is, therefore, 10 units. The response time under the tree topology is smaller than that under the chain topology and the response time under the chain topology is smaller than that under the flat topology. In particular, under the tree topology, a request only travels less than one hop upward on average to get a hit. This is a substantial improvement than that under the flat structure, where the response time increases linearly with the level number h. This demonstrates that, under a hierarchical caching system, a request can be filled by a nearby CS along its upstream path (toward OS). Fig. 8(d) shows the re-



(a) Aggregate traffic load at each level.



(b) Normalized traffic load at each level.

Fig. 9. Traffic load at each level under the flat, chain, and tree topologies.

sponse time from user's perspective at each level for all three topologies. We have similar observations as in Fig. 8(c). The results in Fig. 8(c) and (d) conclusively demonstrate that a hierarchical caching system (e.g., chain/tree) can significantly reduce the response time than a flat caching system.

3) Traffic Load: Another important performance metric for caching system is the traffic load, in particular, the request and response traffic traveling the network path and processed by the CS(es). From a network provider's perspective, such traffic measure is an important input for network capacity planning and traffic engineering. From a service provider (or content delivery provider)'s perspective, traffic load distribution among CS(es) is directly related to user perceived latency, as well as an indication of whether any load balancing is necessary. In this set of simulation results, we attempt to show traffic load behavior with analysis [see (15)] and simulation results.

Fig. 9(a) shows the aggregated request traffic volume at each level, which is the sum of request traffic traversing the same level for the three topologies. Again due to the request aggregation and object sharing, the hierarchical caching system (i.e., chain or tree) has much lower traffic load at most levels (except the leaf level) than the flat structure. In particular, under the flat topology, the closer to the OS, the higher network traffic load, which poses a potential congestion bottleneck at or near the OS. In contrast, for the hierarchical caching system under the chain or tree topology, network traffic is evenly distributed at all levels, which fulfills the objective of load balancing for both the network providers and service providers.

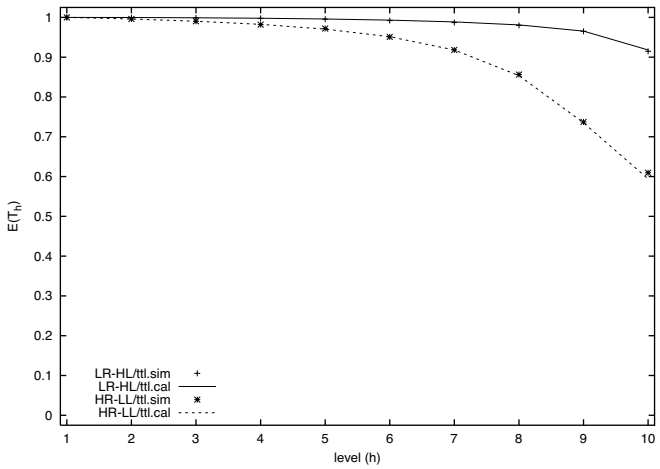


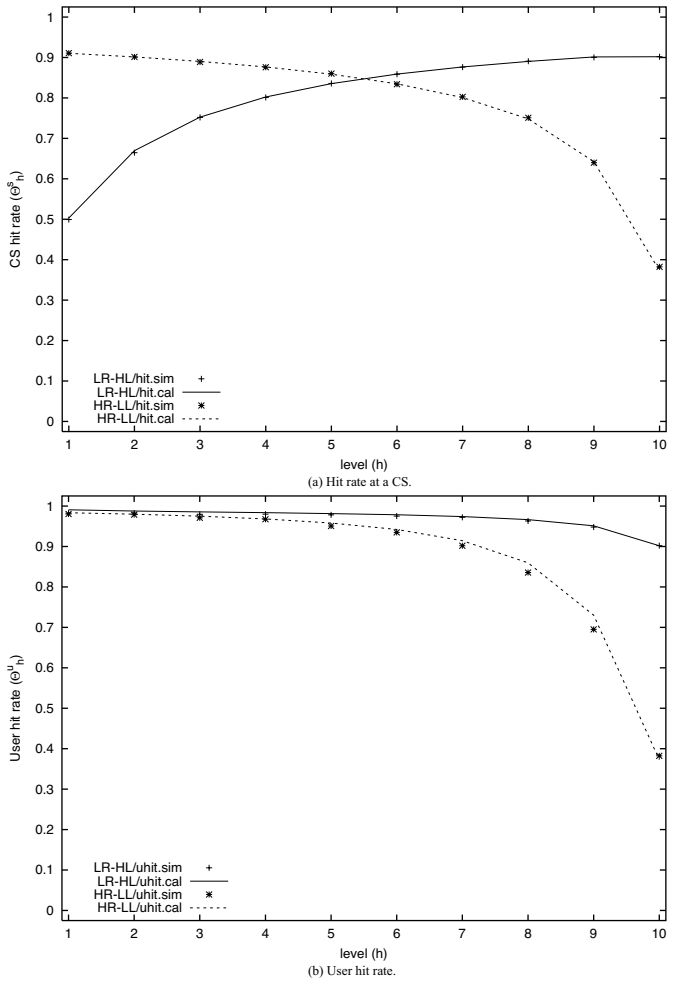
Fig. 10. TTL behavior under different request pattern for the chain topology.

The *lump sum* network traffic at the same level in Fig. 9(a) does not fully demonstrate the superior advantage of a highly aggregated hierarchical caching system such as the tree topology. This is because the user population supported under the tree topology is much larger than that under the chain or flat topology. To illustrate this point, in Fig. 9(b), we plot the *normalized traffic load*, defined as the ratio of request traffic summed over the CS(es) at the same level normalized with respect to the total number of user requests received at the same level. For clarity, we use the $\log_{10}$ scale for the vertical axis in Fig. 9(b) due to the small numerical scale of the result for the tree topology. In Fig. 9(b), we find that, for the tree topology, the normalized traffic load per request at each level is several orders lower than that under the flat topology for upstream CS(es) [i.e., CS(es) close to the OS]. At the leaf level, the advantage of tree topology disappears since there is no more request aggregation and object sharing at leaf CS.

4) Impact of User Request Traffic Pattern: So far our simulation results are based on uniform request pattern at each level (see Section III-A), where user request rate $\lambda_h = 1$ per τ for all $h = 1, 2, \ldots, 10$. In this set of simulation results, we explore how a nonuniform user request pattern at each level can affect the system's and user's performance. In particular, we consider two contrary scenarios.

- *Heavy Root-Light Leaf* (HR-LL) represents the case where the closer it is toward the OS, the more user request rate a CS receives from its local metro region. In our simulation study, we choose $\lambda_h = H - h + 1$, where $1 \le h \le H$ and $H = 10$. That is $\lambda_1 = 10, \lambda_2 = 9, \ldots, \lambda_{10} = 1$.
- *Light Root-Heavy Leaf* (LR-HL) represents the opposite scenario of the HR-LL scenario. Here, the further away it is from the OS, the larger the user request rate a CS receives from its local metro region. In our simulation study, we choose $\lambda_h = h$, where $1 \le h \le H$ and $H = 10$. That is $\lambda_1 = 1, \lambda_2 = 2, \ldots, \lambda_{10} = 10$.

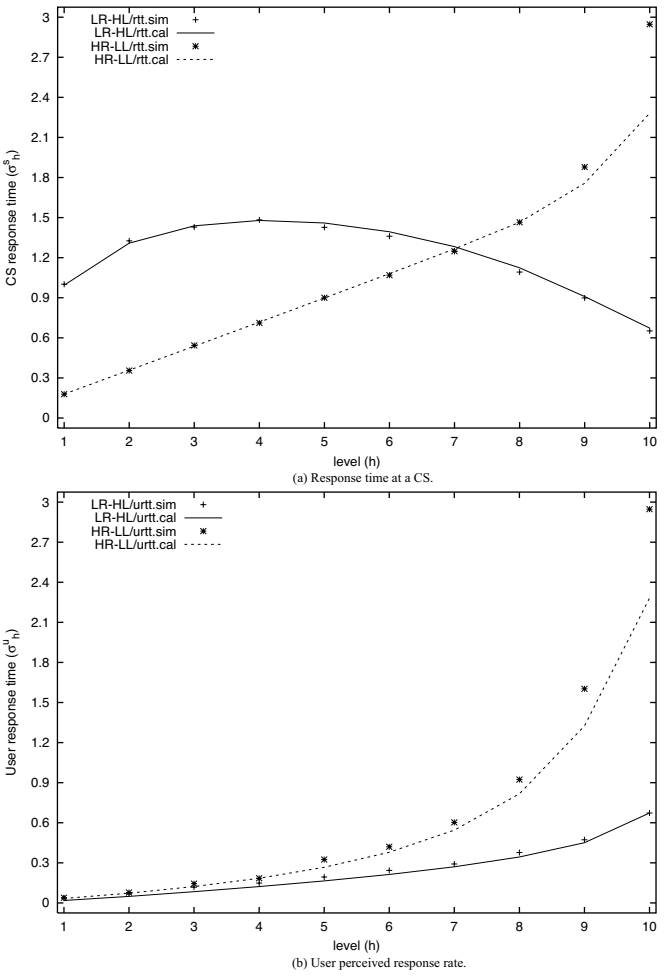
For illustration purpose, we will only present results for the chain topology. Fig. 10 shows the TTL behavior for both the HR-LL and LR-HL user traffic patterns for the chain topology.



(a) Hit rate at a CS.

(b) User hit rate.

Fig. 11. Hit rate for the caching system and user under different request pattern for the chain topology.



(a) Response time at a CS.

(b) User perceived response rate.

Fig. 12. Response time for the caching system and user under different request pattern for the chain topology.

We observe that the TTL behavior under both traffic patterns still follows Property 1, i.e., for $h = 1, 2, \ldots, 10$, $1 \geq E(T_h) > 1/2$ and $1 = E(T_1) > E(T_2) > \cdots > E(T_{10})$. However, due to the difference in user traffic patterns, $E(T_h)$ for the HR-LL case is smaller than that for the LR-HL case when h becomes large. This is intuitive since under the LR-HL case, more user requests at the leaf CS help to compensate the loss of request aggregation and object sharing due to its farther distance away from the OS.

In Fig. 11(a), we plot the CS hit rate at each level for both the HR-LL and LR-HL cases. We find that the hit rate can have drastically different behavior under different user traffic patterns. In particular, we observe that the higher user request rate at a CS, the higher the CS hit rate will be. Fig. 11(b) shows the user hit rate at each level under the HR-LL and LR-HL traffic patterns. Again, we find that hit rate performance favors the LR-HL traffic pattern than that for the HR-LL traffic pattern.

Fig. 12(a) shows the response time at CS at different level under the two traffic patterns (also see Fig. 10). Although the response time for both cases have different performance for CS(es) at different level, the CS response time under the LR-HL traffic pattern is more desirable than that under the HR-LL traffic pattern due to its concave behavior. Fig. 12(b) shows the user perceived response time under the two traffic patterns. Again, we find that the performance under the LR-HL traffic pattern is better than that under the HR-LL traffic pattern.

5) Summary of Simulation Results: From the above results for the TTL behavior, hit rate, response time, traffic load, and user traffic pattern, we observed some important properties and tradeoffs for the flat and hierarchical caching system. For a flat structure, although it can obtain an object with the largest TTL initialized at the OS, it usually has higher miss rate and larger response time, and generates more and uneven traffic load (especially in terms of the per request load) than a hierarchical system (chain or tree). Under the hierarchical caching system, we observe that there is a bias against leaf CS, which is due to the loss of request aggregation and object sharing at leaf CS. Through our simulation results, we have also conclusively demonstrated that, for content distribution employing the weak consistency based caching system, a hierarchical caching system is a scalable solution.

IV. Extensions of the Basic Model

In this section, we explore two extensions (using the concepts of content freshness threshold and randomization) to the basic model. It turns out that these enhancement can be viewed as a generalization of the basic model. Such generalizations provide

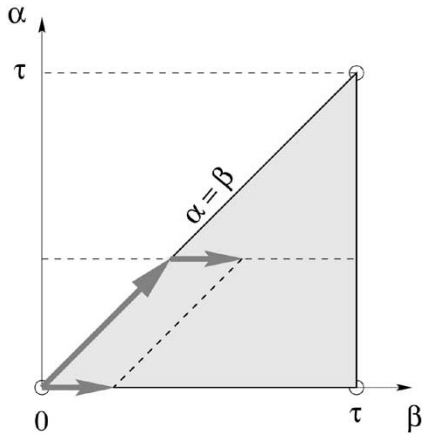


Fig. 13. Feasible region for user required content freshness threshold α and CS prefetching threshold β.

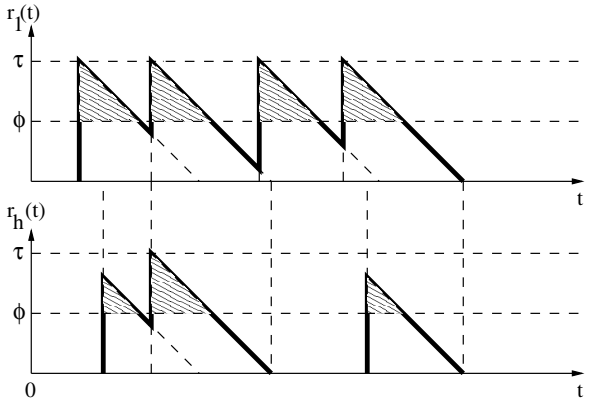


Fig. 14. Sample path of the TTL behavior for a level-1 CS and level-h CS with $\alpha = \beta = \phi$.

us further understanding on possible tradeoffs in the design of a hierarchical caching system.

A. Freshness Threshold

Under the basic model, as long as the user request finds the object with its remaining TTL greater than zero, the request will be filled satisfactorily. Similarly, within the hierarchical caching system, an internal (system) request (from a child CS to its parent CS) will be filled satisfactorily as long as the object's remaining TTL at the parent CS is greater than zero.

We introduce two performance enhancement parameters, namely, *user required content freshness threshold* (α) and *CS prefetching threshold* (β) to enhance the flexibility from both the *user*'s and *CS* (or system)'s perspectives. More specifically, let α, $0 < \alpha \leq \tau$, be the user predefined (required) minimum TTL (or freshness) of the object. That is, if the object's remaining life is greater than α, the object is considered useful to the user. Otherwise, the CS will forward the request to its parent CSs until it finds this object with its remaining TTL greater than α. Then, this object will be used to update the TTL of the initiating child CS, as well as all the intermediate CSs involved in the query process. The parameter α is particular important when a user needs some time to work on the object for a minimum period of time before it expires at the user's local browser.

The second parameter β, which is closely related to α, is defined to be a minimum TTL threshold used by a CS to *prefetch* an object within the hierarchical caching system. That is, upon a request arrives at a CS and finds that the object's remaining TTL is less than β, the CS will initiate a request for this object to its parent CSs along its upstream path until the object with remaining TTL larger than β is found. Clearly, we should have $0 < \alpha \leq \beta \leq \tau$. Note that the basic model discussed in Section II corresponds to the special case when $\alpha = \beta = 0$.

In general, each user may have different freshness requirement for an object, and different α value for different objects. To simplify our discussion, we assume that the α parameter for an object is the same for *all* users within the hierarchical caching system. Under this assumption, we shall show below how the TTL behavior and performance metrics can be derived.

Fig. 13 illustrates the relationship between user required content freshness threshold α and CS prefetching threshold β. Note that by introducing these two parameters, we have generalized the basic model (corresponding to the origin point in Fig. 13) along two dimensions. Furthermore, the *feasible region* of α and β is characterized by the shaded triangle shown in Fig. 13, i.e., $0 < \alpha \leq \beta \leq \tau$. Any point within the triangle represents a parameter pair for (α, β), which determines the behavior and performance of the hierarchical caching system.

Before we discuss the system's behavior and performance for a general setting of (α, β) pair, we first consider some special cases for α and β.

1) Case 1: $0 < \alpha = \beta < \tau$: This corresponds to the diagonal border line of the feasible region in Fig. 13. A sample path of TTL, $r_h(t)$, is illustrated in Fig. 14. Since the object freshness threshold for a user and a CS is the same, the *effective* TTL for user and CS is, thus, the current TTL *minus* (i.e., offset) the α and β threshold. That is, the TTL behavior and CS/user performance can be analyzed by shifting the horizontal axis upward by the threshold value. As an illustration, we calculate the remaining TTL at each level. We follow the same token for the development of (2) and denote $\alpha = \beta = \phi$.

Subcase A. With probability Λ_h/Λ_1, the TTL triangle at level 1 is initiated by a request from the subtree with root at CS $\mathcal{S}_h$. In this case, $T_h = \phi + (\tau - \phi) = \tau$.

Subcase B. With probability $(\Lambda_1 - \Lambda_h/\Lambda_1) \cdot \int_0^{\tau-\phi} \Lambda_h e^{-\Lambda_h t} dt = (\Lambda_1 - \Lambda_h/\Lambda_1) \cdot [1 - e^{-\Lambda_h(\tau-\phi)}]$, the level 1 TTL triangle is triggered by a request outside the subtree rooted at $\mathcal{S}_h$ and there is a request arrival in the same renewal period from the subtree with root $\mathcal{S}_h$. The average T_h in this case is given by

$$\frac{\int_0^{\tau-\phi}(\tau - t)\Lambda_h e^{-\Lambda_h t} dt}{\int_0^{\tau-\phi} \Lambda_h e^{-\Lambda_h t} dt} = \frac{\tau - \phi e^{-\Lambda_h(\tau-\phi)} - \frac{1}{\Lambda_h} + \frac{1}{\Lambda_h} e^{-\Lambda_h(\tau-\phi)}}{1 - e^{-\Lambda_h(\tau-\phi)}}.$$

Denote $E_\phi(T_h)$ the average remaining TTL at level h for $\alpha = \beta = \phi$. To calculate $E_\phi(T_h)$, all we need to do is to take the probabilistically weighted average of T_h under cases A and B. We have (16), shown at the bottom of the next page.

Since $E(T_h)$ in (2) corresponds to the special case when $\phi = 0$ in $E_\phi(T_h)$, we have the following property showing the relationship between $E_\phi(T_h)$ and $E(T_h)$.

Property 2: Let $E_\phi(T_h)$ and $E(T_h)$ be defined in (16) and (2), respectively. Then, we have

$$E_\phi(T_h) = \phi + E(T_h)|_{\tau:=(\tau-\phi)} \tag{17}$$

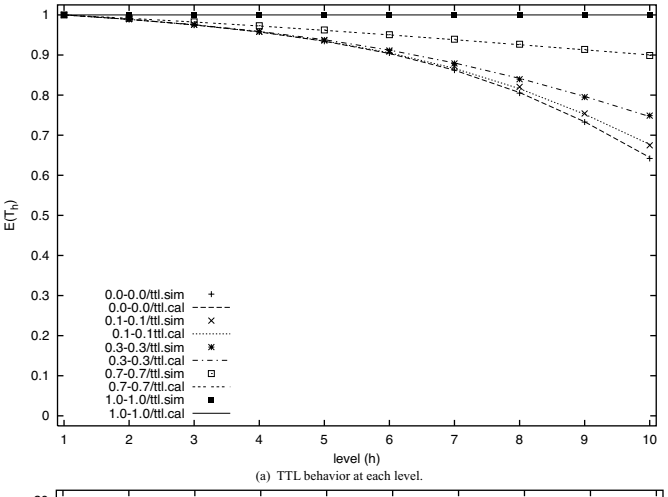
where $E(T_h)|_{\tau:=(\tau-\phi)}$ denotes replacing τ with $(\tau - \phi)$ for $E(T_h)$ in (2).

Property 2 can be easily verified and is also quite intuitive. Since the user required content freshness threshold α and the CS prefetching threshold are the same (ϕ), the "relative" (or "effective") TTL can be considered as the absolute TTL minus the threshold ϕ (see shaded area in Fig. 13).

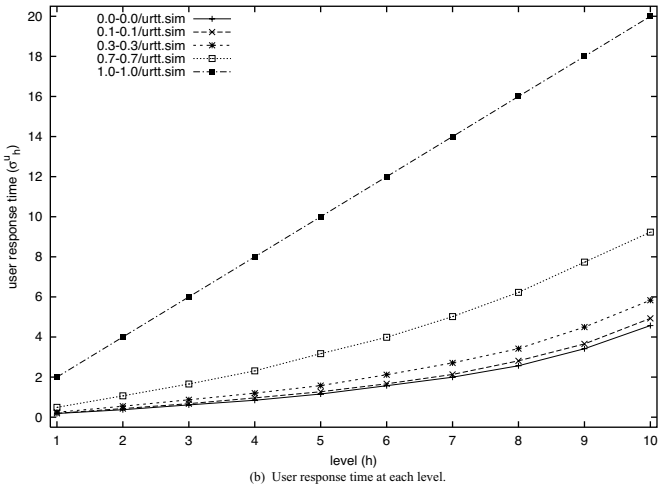
Once we obtain the $E_\phi(T_h)$, we can follow the same token as in Section II to obtain other performance metrics, which are similar to that for the basic model. To conserve space, we omit to elaborate them further.

We use simulation results to substantiate our analysis. We will use the chain topology and the same set of simulation parameters used in Section III, i.e., $\tau = 1$ unit, $\lambda_h = 1$ per τ, and $d_h = 2$ units. Fig. 15(a) shows the $E_\phi(T_h)$ at a CS at each level $h, h = 1, 2, \ldots, 10$, under different (α, β) pair. We observe that the simulation results for $E_\phi(T_h)$ match perfectly to our analytical results (Property 2). In the figure, $\alpha = \beta = 0$ corresponds to the basic model. When $\alpha = \beta = \tau$, it is similar to the flat case (i.e., without hierarchy) where each CS has to contact the OS directly. For (α, β) between 0 and τ, as users increase their freshness requirement, the TTL curve is pushed upward (from the basic model case to the flat case). This corresponds to an increase in user response time, which is depicted in Fig. 15(b). This is intuitive and shows the tradeoff between user content freshness requirement and response time. We also observe that such tradeoff exhibit uneven behavior. For example, after (α, β) have reached 0.7τ, further increase in (α, β) can only offer a marginal increase for T_h but with a significant increase in user response time.

2) Case 2: $\alpha = 0, 0 < \beta \leq \tau$: A sample path of the TTL triangle at a CS at each level is similar to that illustrated in Fig. 14. The difference here is that the number of triangles, or rather, the triangle density at each level is more sparse than that under Case 1, due to the fact that requests from local users (with $\alpha = 0$) will be filled satisfactorily as long as the local CS's remaining TTL is nonzero. That is, only when a user request finds the local CS with a zero TTL will the local CS contact its parent CS. On the other hand, when a CS initiates a request to its upstream parent server, the request must be filled by a CS with the object's remaining TTL greater than β. Such



(a) TTL behavior at each level.



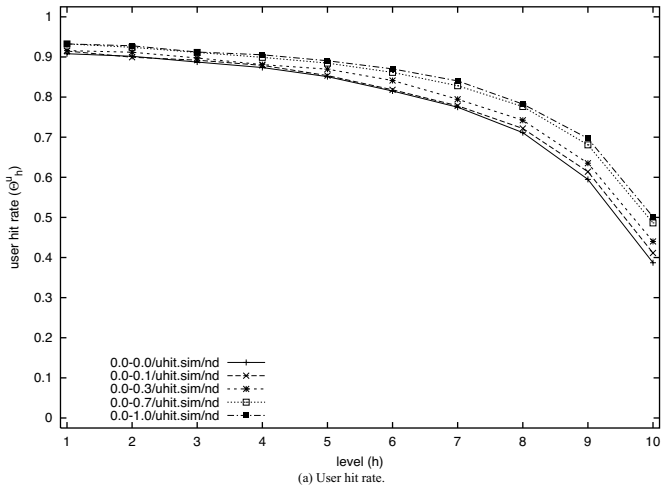
(b) User response time at each level.

Fig. 15. TTL behavior and user response time performance under different (α, β) $(\alpha = \beta)$ parameter pair for the chain topology.

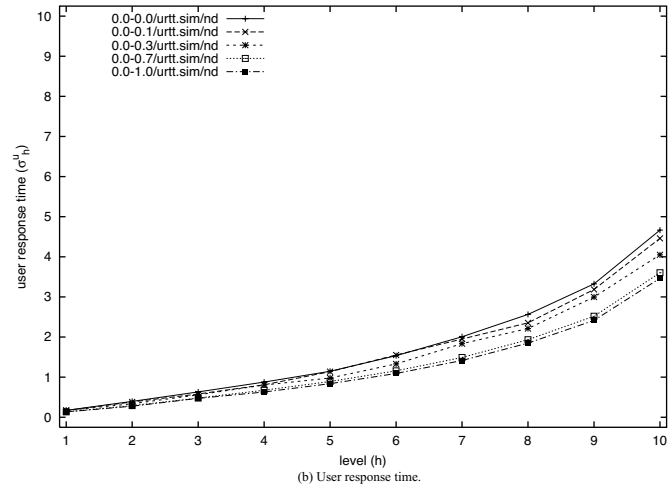
decoupling of α and β clearly weakens the synchronization relationship among the TTLs at different levels than that under the previous case (i.e., $\alpha = \beta$). Although a TTL sample path is still covered under its parent CS, the relationship between them becomes much more complex than the previous case or the basic model. So far, there does not appear to be a clean closed form solution to characterize the nonzero TTL behavior at each level and we leave this as an open problem for future research.

3) General Case: $0 \leq \alpha \leq \beta \leq \tau$: The general case can be considered as a *composition* of the two simpler cases discussed above (see Fig. 13). For the TTL sample path, we can imagine first to make a vertical shift of the horizontal time axis to reflect the nonzero α. Once this is done, we are facing exactly the same problem as in Case 2, i.e., $\alpha = 0$, $0 < \beta < \tau$. Referring to Fig. 13, this corresponds to moving the (α, β) pair first along

$$\begin{aligned} E_\phi(T_h) &= \frac{\frac{\Lambda_h}{\Lambda_1} \cdot \tau + \frac{\Lambda_1 - \Lambda_h}{\Lambda_1} \cdot [1 - e^{\Lambda_h(\tau-\phi)}] \cdot \frac{\tau - \phi e^{-\Lambda_h(\tau-\phi)} - \frac{1}{\Lambda_h} + \frac{1}{\Lambda_h} e^{-\Lambda_h(\tau-\phi)}}{[1 - e^{-\Lambda_h(\tau-\phi)}]}}{\frac{\Lambda_h}{\Lambda_1} + \frac{\Lambda_1 - \Lambda_h}{\Lambda_1} \cdot [1 - e^{-\Lambda_h(\tau-\phi)}]} \\ &= \frac{\Lambda_h \tau + (\Lambda_1 - \Lambda_h)[\tau - \phi e^{-\Lambda_h(\tau-\phi)} - \frac{1}{\Lambda_h} + \frac{1}{\Lambda_h} e^{-\Lambda_h \tau}]}{\Lambda_h + (\Lambda_1 - \Lambda_h)[1 - e^{-\Lambda_h(\tau-\phi)}]} \end{aligned} \tag{16}$$



(a) User hit rate.



(b) User response time.

Fig. 16. Impact of prefetching on user perceived performance under the chain topology.

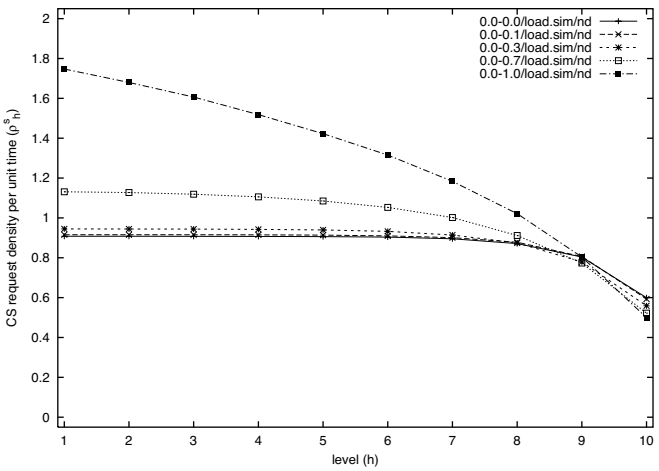


Fig. 17. Impact of prefetching on traffic load under the chain topology.

the $\alpha = \beta$ diagonally. Then, we fix the α value and move β horizontally to its operating point.

Under the general case, when a request (either from local user or a child CS) arrives at a CS, there are three cases.

- *Subcase 1.* The remaining TTL of the object at the CS is less than α. In this case, the CS initiates a request to its upstream CS until it retrieve the object with remaining TTL greater than β.
- *Subcase 2.* The remaining TTL of the object at the CS is greater than β. In this case, the CS will deliver the object and the request will be filled satisfactorily.
- *Subcase 3.* The remaining TTL of the object at the CS is greater than α but less than β. Under this case, the proxy immediately fulfills the request and, at the same time, generate an independent request to its upstream parent CSs until it obtains the object with the remaining TTL greater than β. Here, β serves as a *prefetching* parameter for the CSs.

We use a set of simulation results to demonstrate the impact of prefetching on user perceived performance metrics and network traffic load. For simplicity, in the following discussions, we only show the simulation results for the case when $\alpha = 0$ and $0 < \beta < \tau$. Fig. 16 shows some marginal improvement on user perceived performance metrics (hit rate and response time) when the prefetching parameter β increases from 0 to τ. Fig. 17 shows the network traffic load when prefetching parameter β varies from 0 to τ under the chain topology. We observe that, with prefetching, a CS becomes more "proactive" to generate requests in anticipation of future requests. Although this is some modest improvement in user perceived performance metrics (hit rate, response time), the prefetching parameter β has to be chosen carefully (e.g., less than 0.7τ) in order to avoid introducing excessive request traffic load in the network.

B. Randomization

Under the basic model, the TTL behavior at each level along a path from the root of the tree has strong *synchronization* property (see Figs. 3 and 6). Such synchronization property may adversely affect the performance of the hierarchical caching system. For illustration, in Fig. 18, we plot the probability distribution for average network delay and number of servers involved to process each request under the basic model given that there is a miss. We observe that, except at the leaf CS, it is very likely that a miss at local CS will cause inquiries for *all* the upstream CSs toward the OS.

To minimize the number of upstream CSs involved in processing a request, we introduce *randomization* into the cache query process as follows. For a CS at level h, if there is a miss at this CS, it will make a query to its upstream CSs at levels 0 to $h-1$ with a probability $p_0, p_1, \ldots, p_{h-1}$, where $\sum_{i=0}^{h-1} p_i = 1$. Such randomization enable a CS to *"jump over"* (or *"cut through"*) its immediate parent CSs during the query process to reach a CS farther away along its upstream path. In Fig. 19, we give an example illustrating the randomization process. As shown in the figure, when a user request for an object arrives at CS 5 and experiences a miss, CS 5 will choose a CS 0 to 4 with a probability. Note that CS 0 is the OS and always has a fresh copy of the content. Suppose CS 3 is chosen and a miss happens again. At this time, CS 3 will repeat the same process, i.e., choosing a CS among CSs 0 to 2, each with a certain probability. Now, suppose CS 1 is chosen and again there is a miss. Then, CS 1 will contact CS 0 (with probability 1). When the object is retrieved from CS 0 (OS), the object will update copies at CSs 1, 3, and 5 only along the downstream path, i.e., only those

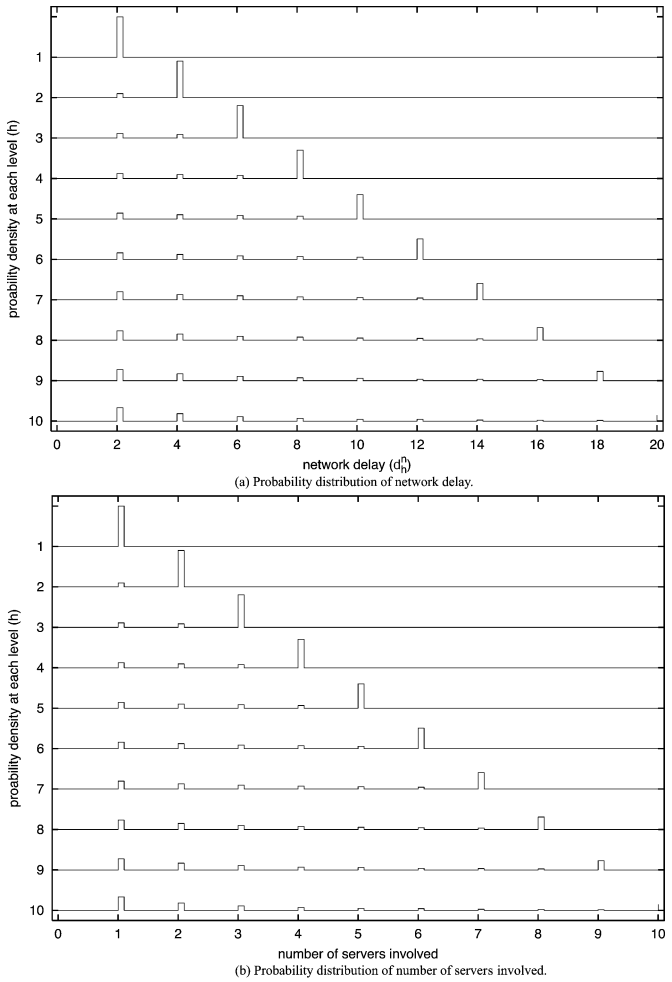


Fig. 18. Probability distribution of network delay and number of servers involved to process a request at each level under given a miss at local CS for the chain topology.

CSs involved in this query process will get an updated copy of the object. Note that due to randomization, CSs 2 and 4 will be bypassed during this query process (i.e., "cut through").

Such randomization process can also be viewed as a *generalization* of the basic model and the flat model, both of which can be regarded as extreme cases under the randomization procedure. In particular, for each CS at level h, $h = 1, 2, \ldots, H$, if we set $p_{h-1} = 1$ (i.e., a child CS can only contact its *immediate* parent CS), then we have the basic model discussed in Section II. On the other hand, for a CS at level h, $h = 1, 2, \ldots, H$, if we set $p_0 = 1$ (i.e., a child CS can only contact the OS), then we have the flat caching system.

We use some simulation results to illustrate the impact on the caching system when randomization is employed. In our simulation, we assume the probability function that a CS chooses one of upstream parent CSs follows a *geometric* distribution. More specifically, when a request experiences misses at a CS of level h, the CS will send the request to the original server (at level 0) with probability $p_0 = p$, or to the CS at level 1 with probability $p_1 = r \cdot p$, or to the CS at level i with probability $p_i = r^i \cdot p$, where $0 \leq i \leq h-1$. Here, $p = 1/\sum_{i=0}^{h-1} r^i$ to ensure $\sum_{i=0}^{h-1} p_i = 1$. It should be clear that the parameter r determines the randomization behavior. For example, when r

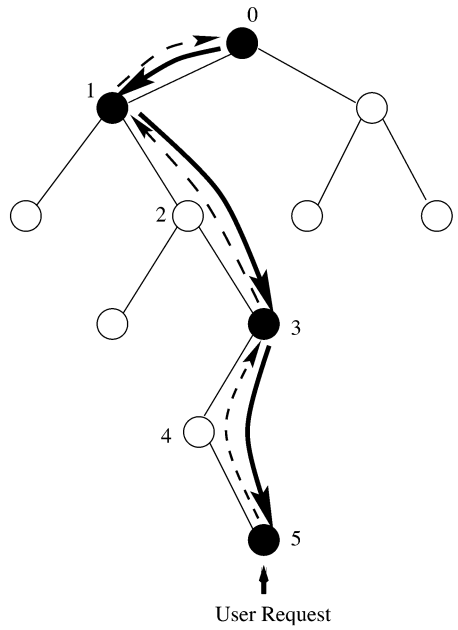


Fig. 19. An example illustrating randomization during the request query process from a CS to an upstream CS.

approaches infinity, it becomes the strict hierarchical case (i.e., the basic model). When $r = 0$ and $p_0 = 1$, it becomes the "flat" case, i.e., each CS will contact the OS upon a miss. When $r = 1$, the probability is uniformly distributed and the CS will contact one of its parent CSs along its upstream path with equal probability. In general, when $r > 1$, the request is more likely directed to the CS which is close to the requesting CS along the upstream path; when $0 < r < 1$, the request is more likely sent to the CS close to the OS.

Again, we use the chain topology for our simulation and use the same set of simulation parameters used in Section III. Fig. 20(a) shows the probability distribution for network delay at each level for $r = 2$. Comparing Fig. 20(a) to Fig. 18(a), we observe that randomization helps shift and balance probability mass toward upstream CS and reduce network delay. A more interesting simulation result showing the number of servers need to be queried for a request is given in Fig. 20(b). Comparing Fig. 20(b) with Fig. 18(b) (which shows the number of servers needed to process a request under the basic model), we find that randomization technique can significantly reduce the number of upstream servers involved in a request query process. This is perhaps the most significant advantage for employing randomization technique.

Currently, we do not have a utility function to obtain user's response time from network delay (i.e., number of hops that a user request needs to travel along its upstream path) with the number of upstream servers involved in processing a request. Since server overload is a critical problem for many popular web sites (or CSs) over current Internet and, in practice, contributes to a large fraction of user perceived latency, it is therefore critical to develop mechanisms to reduce server overhead. The randomizing technique discussed here is a viable approach to achieve this objective.

In Fig. 21, we further explore the impact of randomization parameter r on network delay and server overhead (i.e., number

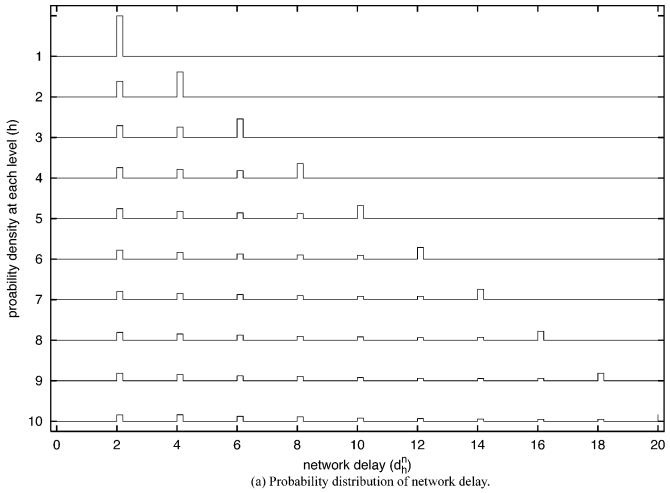


Fig. 20. Probability distribution of network delay and number of servers involved to process a request at each level given there is a miss at local CS. Randomization parameter $r = 2$.

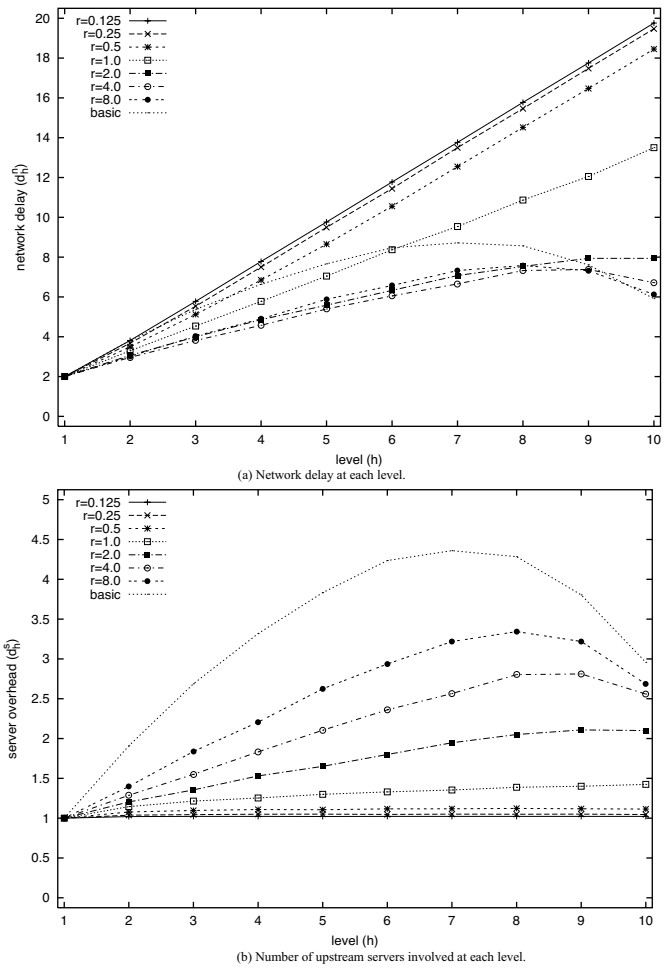


Fig. 21. User response time and number of servers involved to process a request at each level CS given that there is a miss at local CS. Chain topology is used for simulations.

of servers involved to process a request). We also include the results for the basic model as a comparison. In Fig. 21(a), we observe that when r is very small $(r < 1)$, the average user response time increases very quickly along downstream direction. When r increases $(r > 2)$, the increase in user response time slows down as h increases. As discussed earlier, the basic model can be regarded as the case when r approaches infinite. For the basic model, the average network delay is better for CSs close to the leaf CS than other cases of r. Fig. 21(b) shows the average number of servers involved to process a request at each level CS under different settings of randomization parameter r. We observe that the smaller the randomization parameter r is, the less number of upstream servers will be involved in processing a request (and, thus, lower server overhead). Fig. 21(a) and (b) shows that the performance for network delay and CS overhead head to different direction as the randomization parameter increases. This suggests that a "suitable" choice of the randomization parameter r is crucial to make a compromise between network delay and server overhead. In particular, extreme values of r (either too large or too small) should be avoided and a r in the range of 1 to 2 appears desirable since such range gives a good tradeoff between network delay and server overhead.

During our discussion of the randomization technique, we have implicitly assumed that both the user content freshness threshold requirement α and CS prefetching threshold β are 0, i.e., $\alpha = \beta = 0$. In general, randomization technique can work together with freshness threshold in a hierarchical caching system.

V. Related Work

Caching has long been used in many distributed computer and database systems [13]. The tremendous popularity and growth of the web has reiterated the practical significance of caching systems [3], [23]. Most studies focus on measurement and traffic characterization for the user client, cache proxy, and web server in different temporal and spatial scales [1], [18].

Since proxy plays an important role as the intermediate agent in the "client-proxy-server" paradigm, the performance and behaviors of proxy in request aggregation (hit and miss), server selection [7], cache replacement algorithms [25], object prefetching [6], and inter-cache communications [12] have been studied in the literature.

Gwertzman and Seltzer [14] examined several existing cache maintenance schemes experimentally and found that a weak

cache consistency protocol such as the one used in the Alex file system [8] could be tuned to achieve less than 5% stall objects. Chankhunthod *et al.* [9] studied the hierarchical caching and consistency problem in the Harvest caching system. Urgaonkar *et al.* [22] proposed to maintain the cache consistency among proxy servers, in addition to consistency with the OS.

On the other hand, Liu and Cao [17] proposed approaches (adaptive TTL, polling-every-time, and invalidation) to maintain strong cache consistency. Yu *et al.* [26] suggested to convey the invalidation message over the application-level multicast for reduced overhead. Other server-based approaches (e.g., volume lease [24] and validation piggyback [16]) have also been explored in the literature.

Despite the fact that the TTL-based weak consistency hierarchical web caching systems have already been widely adopted and deployed, current understanding and potential enhancement on such system are still largely limited to empirical studies. This paper took an analytical approach on the intrinsic remaining TTL behavior in such system, and derived performance metrics such as hit (or miss) rate and response time from both user and system perspectives. Based on these results and observations, this paper further proposed two schemes to enhance and generalize the the design of a hierarchical caching system, and provided insights on how to make tradeoffs among object freshness, response latency, and traffic load.

The most relevant work to ours is the work by Cohen and Kaplan [10], where the authors focused on the effects of aging on cache miss rate. Among other things, the authors in [10] considered a simple tree structure with a height of two and compared its miss rate with other configurations under different request arrival patterns. Motivated by the work in [10], this paper aimed to obtain a deeper understanding of an expiration-based hierarchical caching system with theoretical underpinning. By casting a hierarchical caching system with a simple model, we were able to obtain better understanding of the time-domain behavior of weak consistency and a comprehensive set of performance metrics from both system's and user's perspectives.

VI. Conclusion

Caching is an important means to scale up with the growth of the Internet and weak consistency is a major approach used in Web caching. This paper presented a fundamental study of hierarchical caching systems based on the weak consistency paradigm. Although the current HTTP implementations of caching provide similar features, our investigation of the weak consistency problem was conducted in a general setting and was not limited to the details of HTTP implementations. The main contribution of this paper are two-folded. First, we presented a basic model for a hierarchical caching system by using the concept of TTL expiration mechanism. We analyzed the intrinsic timing behavior of such system and derived important performance metrics from both the system's and user's perspectives. Second, we introduced the freshness threshold and randomization techniques, both of which generalize the the basic model and enhance its performance. Our simulation results confirmed the efficacy of our analysis and provided further insights on various tradeoffs between performance and cost.

Appendix

Proof of Property 1

By the nature of the TTL update mechanism at each level of the hierarchical caching system, we have $\tau > E(T_1) > E(T_2) > \cdots > E(T_h) > \cdots > E(T_H)$. Therefore, we only need to show that $E(T_h) > \tau/2$.

To start with, we first show that the expression in (1) is greater than $\tau/2$. To show

$$\frac{\tau - \frac{1}{\Lambda_h} + \frac{1}{\Lambda_h}e^{-\Lambda_h\tau}}{1 - e^{-\Lambda_h\tau}} > \frac{\tau}{2}$$

is equivalent to showing that

$$2\tau - \frac{2}{\Lambda_h} + \frac{2}{\Lambda_h}e^{-\Lambda_h\tau} > \tau - \tau e^{-\Lambda_h\tau}.$$

This is equivalent to showing that

$$\Lambda_h\tau - 2 + 2e^{-\Lambda_h\tau} > -\Lambda_h\tau e^{-\Lambda_h\tau}$$

or

$$\Lambda_h\tau e^{\Lambda_h\tau} - 2e^{\Lambda_h\tau} + 2 + \Lambda_h\tau > 0.$$

To simplify notation, we denote $x = \Lambda_h\tau$ and $y = \Lambda_h\tau e^{\Lambda_h\tau} - 2e^{\Lambda_h\tau} + 2 + \Lambda_h\tau = xe^x - 2e^x + 2 + x$. Therefore, all we need to show is that $y > 0$ for $x > 0$. Note that $y'(x) = xe^x - e^x + 1$ and $y''(x) = xe^x$. Clearly, $y''(x) > 0$ for $x > 0$. Since $y'(0) = 0$, it follows that $y'(x) > 0$ for $x > 0$. Similarly, since $y(0) = 0$, we have $y(x) > 0$ for $x > 0$.

With these results for (1), we are ready to prove that $E(T_h) > \tau/2$. From (2), we have

$$E(T_h) = \frac{\frac{\Lambda_h}{\Lambda_1}\cdot\tau + \frac{\Lambda_1-\Lambda_h}{\Lambda_1}\cdot(1 - e^{-\Lambda_h\tau})\cdot y}{\frac{\Lambda_h}{\Lambda_1} + \frac{\Lambda_1-\Lambda_h}{\Lambda_1}\cdot(1 - e^{-\Lambda_h\tau})}.$$

We now show that $E(T_h) > y$, which is a stronger condition than $E(T_h) > \tau/2$ since we have shown that $y > \tau/2$.

To show

$$\frac{\frac{\Lambda_h}{\Lambda_1}\cdot\tau + \frac{\Lambda_1-\Lambda_h}{\Lambda_1}\cdot(1 - e^{-\Lambda_h\tau})\cdot y}{\frac{\Lambda_h}{\Lambda_1} + \frac{\Lambda_1-\Lambda_h}{\Lambda_1}\cdot(1 - e^{-\Lambda_h\tau})} > y$$

is equivalent to showing that $\tau > y$, which is always true by the nature of the hierarchical caching system.

Acknowledgment

The authors wish to thank X. Tang of Hong Kong University of Science and Technology for pointing out a simpler derivation for the average TTL under the basic model.

References

[1] G. Abdulla, "Analysis and Modeling of World Wide Web Traffic," Ph.D. dissertation, Dept. Comput. Sci., Virginia Polytechnic Inst. State Univ., Blacksburg, VA, 1998.

[2] M. Arlitt and C. Williamson, "Web server workload characterization: The search for invariants," in *Proc. ACM SIGMETRICS Conf.*, Philadelphia, PA, May 1996, pp. 126–137.

[3] G. Barish and K. Obraczka, "World wide web caching: Trends and techniques," *IEEE Commun. Mag.*, vol. 38, pp. 178–184, May 2000.

[4] P. Barford, A. Bestavros, A. Bradley, and M. E. Crovella, "Changes in web client access patterns: Characteristics and caching implications," *World Wide Web*, vol. 2, pp. 15–28, 1999.

[5] L. Breslau, P. Cao, L. Fan, G. Philips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *Proc. IEEE INFOCOM'99*, New York, NY, Mar. 1999, pp. 126–134.

[6] P. Cao, E. Felten, A. Karlin, and K. Li, "A study of integrated prefetching and caching strategies," presented at the ACM SIGMETRICS'95, Ottawa, ON, Canada, 1995.
[7] R. Carter and M. Crovella, "Server selection using dynamic path characterization in wide-area networks," in *Proc. IEEE INFOCOM'97*, 1997, pp. 1014–1021.
[8] V. Cate, "Alex—A global filesystem," in *Proc. 1992 USENIX File System Workshop*, Ann Arbor, MI, May 1992, pp. 1–12.
[9] A. Chankhunthod, P. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A hierarchical internet object cache," in *Proc. USENIX 1996 Tech. Conf.*, San Diego, CA, Jan. 1996, pp. 153–163.
[10] E. Cohen and H. Kaplan, "Aging through cascaded caches: Performance issues in the distribution of web content," in *Proc. ACM SIGCOMM Conf.*, San Diego, CA, Aug. 2001, pp. 41–53.
[11] A. Dingle. Cache consistency in the HTTP 1.1 proposed standard. presented at 1st Workshop on Web Caching. [Online]. Available: http://w3cache.icm.edu.pl/workshop/program.html
[12] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Networking*, vol. 8, pp. 281–293, June 2000.
[13] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West, "Scale and performance in a distributed file system," *ACM Trans. Comput. Syst.*, vol. 6, pp. 51–81, 1988.
[14] J. Gwertzman and M. Seltzer, "World-wide web cache consistency," in *Proc. 1996 USENIX Tech. Conf.*, San Diego, CA, Jan. 1996, pp. 141–151.
[15] V. Jacobson, "How to kill the Internet," presented at the ACM SIGCOMM'95 Middleware Workshop, Cambridge, MA, Aug. 28, 1995, [Online]. Available: http://www.root.org/ip-development/.
[16] B. Krishnamurthy and C. Wills, "Study of piggyback cache validation for proxy caches in the world wide web," presented at the USENIX Symp. Internet Technology and Systems (ITS'97), Monterey, CA, 1997.
[17] C. Liu and P. Cao, "Maintaining strong cache consistency for the world-wide web," *IEEE Trans. Comput.*, vol. 47, pp. 445–457, 1998.
[18] A. Mahanti, C. Williamson, and D. Eager, "Traffic analysis of a web proxy caching hierarchy," *IEEE Network*, vol. 14, pp. 16–23, May–June 2000.
[19] Network Simulator—NS-2. [Online]. Available: http://www.isi.edu/nsnam/ns/
[20] M. Rabinovich and O. Spatscheck, *Web Caching and Replication*. Reading, MA: Addison-Wesley, 2002.
[21] S. M. Ross, *Introduction to Probability Models*, 4th ed. New York: Academic, 1989, ch. 7.
[22] B. Urgaonkar, A. G. Ninan, M. S. Raunak, P. Shenoy, and K. Ramamritham, "Maintaining mutual consistency for cached web objects," presented at the IEEE 21st Int. Conf. Distributed Computing Systems, Mesa, AZ, April 2001.
[23] J. Wang, "A survey of web caching schemes for the internet," *ACM Comput. Commun. Rev.*, vol. 25, no. 9, pp. 36–46, 1999.
[24] J. Yin, L. Alvisi, M. Dahlin, and C. Lin, "Volume lease for consistency in large-scale systems," *Knowl. Data Eng.*, vol. 11, no. 4, pp. 563–576, 1999.
[25] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox, "Removal policies in network caches for world-wide web documents," presented at the ACM SIGCOMM, Stanford, CA, Aug. 1996.
[26] H. Yu, L. Breslau, and S. Shenker, "A scalable web cache consistency architecture," presented at the ACM SIGCOMM Conf., Cambridge, MA, Aug./Sept. 31–3, 1999.

Y. Thomas Hou (S'91–M'98) received the B.E. degree (*summa cum laude*) from the City College of New York, New York, NY, in 1991, the M.S. degree from Columbia University, New York, NY, in 1993, and the Ph.D. degree from Polytechnic University, Brooklyn, NY, in 1998, all in electrical engineering.

From 1997 to 2002, he was a Research Scientist and Project Leader in the IP Networking Research Department, Fujitsu Laboratories of America, Sunnyvale, CA (Silicon Valley). He is currently an Assistant Professor in the Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA. His research interests include wireless sensor networks, multimedia delivery over wireless networks, scalable architectures, protocols, and implementations for differentiated services Internet, and service overlay networking.

Dr. Hou has published extensively in the above areas and is a corecipient of the 2002 IEEE International Conference on Network Protocols Best Paper Award and the 2001 IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY Best Paper Award. He is a Member of the Association for Computing Machinery (ACM).

Jianping Pan (M'99) received the B.S. and Ph.D. degrees in computer science from Southeast University, Nanjing, China, in 1994 and 1998, respectively.

From 1999 to 2001, he was a Postdoctoral Fellow and Research Associate with the Center for Wireless Communications at the University of Waterloo, Waterloo, ON, Canada. Since September 2001, he has been a Member of Research Staff at Fujitsu Laboratories of America, Sunnyvale, CA. His research interests include transport protocols and application services for multimedia, high-speed and mobile networks.

Dr. Pan is a Member of the Association for Computing Machinery (ACM).

Bo Li (S'89–M'92–SM'99) received the B.S. and M.S. degrees in computer science from Tsinghua University, Beijing, China, in 1987 and 1989, respectively, and the Ph.D. degree in computer engineering from the University of Massachusetts, Amherst, in 1993.

From 1994 and 1996, he worked on high-performance routers and ATM switches in the IBM Networking System Division, Research Triangle Park, NC. Since January 1996, he has been with the Computer Science Department, Hong Kong University of Science and Technology, Kowloon, where he is now an Associate Professor. He is also an Adjunct Researcher at Microsoft Research Asia (MSRA), Beijing, China. His current research interests include wireless and mobile networks supporting multimedia, video multicast, and all-optical networks with WDM. He has published over 140 technical papers in refereed journals and conference proceedings. He has been an Editor or a Guest Editor for 14 journals and involved in the organization of over 30 conferences.

Dr. Li currently serves as the Chair of the Technical Program Committee of the IEEE INFOCOM 2004. He is a Member of the Association for Computing Machinery (ACM).

Shivendra S. Panwar (S'82–M'85–SM'00) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Kanpur, in 1981, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Massachusetts, Amherst, in 1983 and 1986, respectively.

He is a Professor in the Electrical and Computer Engineering Department, Polytechnic University, Brooklyn, NY. He is currently the Director of the New York State Center for Advanced Technology in Telecommunications (CATT). He spent the summer of 1987 as a Visiting Scientist at the IBM T. J. Watson Research Center, Yorktown Heights, NY, and has been a Consultant to AT&T Bell Laboratories, Holmdel, NJ. His research interests include the performance analysis and design of networks. Current work includes protocol analysis, traffic and call admission control, switch performance, and multimedia transport over wireless networks.

Dr. Panwar has served as the Secretary of the Technical Affairs Council of the IEEE Communications Society (1992–1993) and is a Member of the Technical Committee on Computer Communications. He is a coeditor of *Network Management and Control, Vol. II*, and *Multimedia Communications and Video Coding* (New York: Plenum, 1994 and 1996, respectively).