A Framework for Evaluating Caching Policies in A Hierarchical Network of Caches

Eman Ramadan, Pariya Babaie, Zhi-Li Zhang
Department of Computer Science and Engineering
University of Minnesota, Twin Cities
Email: eman, babai008, zhzhang@cs.umn.edu

Abstract—**Much attention of the research community has focused on performance analysis of cache networks under various caching policies. However, the issue of how to evaluate and compare caching policies for cache networks has not been adequately addressed. In this paper, we propose a novel and general framework for evaluating caching policies in a hierarchical network of caches. We introduce the notion of a hit probability/rate matrix, and employ a generalized notion of majorization as the basic tool for evaluating caching policies for various performance metrics. We discuss how the framework can be applied to existing caching policies, and conduct extensive simulation-based evaluation to demonstrate the utility and accuracy of our framework.**

I. Introduction

The emergence of information-centric network (ICN) architectures [1], [2], [3], [4], [5] (see [6] for a survey of ICN architectures) has attracted a flurry of renewed research interest in caching policies and their performance analysis. Classical caching policies such as FIFO, LRU, or LFU – which specify what object should be evicted when the cache is full – have been widely used in computer systems. It is notoriously difficult to exactly analyze the performance of these caching policies, instead one has to resort to approximation methods with various assumptions [7], [8], [9]. These policies can be viewed as organizing the cached objects in an ordered list for replacement. More recently, timer-based caching policies have gained particular attention (see, *e.g.*, [10], [11], [12], [13]) – where each object is associated with a time-to-live (TTL) timer, and is evicted when the timer expires. The interest in timer-based caching policies is due to the fact that objects within a cache can be viewed independently, and their hitting probabilities can be analyzed separately. As an analytical aid, TTL-cache can provide a good avenue to approximate classical list-based caching policies [14], *e.g.*, in terms of characteristic times [7], [15], [16].

One important feature ICNs offer is a distributed network of caches, namely, a *cache network*, which poses additional challenges both in terms of practical cache management issues and performance analysis. For example, when an object is evicted from one cache, should it simply be discarded, or should it be inserted into the next cache (*i.e.*, as a *new arrival* to this next cache) along the path to the origin server? When an object is returned from an upstream cache (or the origin server) back down along the request path, should it be cached along the way (*e.g.*, as advocated by [1], [2]), selectively or probablistically at some caches (*e.g.*, leave-copy-down or leave-copy-probabilistically [7]), or only at the edge (*e.g.*, as advocated by [4])? Should caches in a network be operated independently, in a cooperative fashion [17], or managed *globally* with a coherent view [18]?

Moreover, no single caching policy is likely to perform well for *all* user request patterns. For example, under the ***I**ndependence **R**eference **M**odel (IRM)*, static caching is shown to be optimal [19] if the object popularity distribution is known *a priori*. However, it is shown in [20] that static caching is no longer optimal when the interarrival distribution of object requests has a decreasing hazard rate (*e.g.*, when the interarrivals of object requests are Pareto-distributed). In this case, instead of always caching the most popular objects as in the case of static caching, the optimal policy is to cache each object with a probability less than 1. This means that objects do not have to be always in the cache, leaving space for more objects to be cached. In addition, static caching cannot cope with changes in user request patterns, *e.g.*, a flash-crowd.

From a theoretical standpoint, performance analysis of a network of caches is significantly more difficult: consider the simple case of a line of caches where each cache employs its own cache replacement policy (*e.g.*, LRU) independently; assuming that object request streams at the first cache (the *edge* cache) are independent, the request arrival streams at the upstream caches are no longer independent – they are generated by cache misses from the downstream caches. Such coupling of the caches in tandem network is what makes the analysis of cache networks a challenging task. Approximation results for cache networks of specific topologies (*e.g.*, a line or star network) have been obtained for LRU caches (see, *e.g.*, [7], [15], [16]); exact and approximation results for a general network of TTL-caches have been developed recently under either the renewal arrival processes or Markov arrival processes (MAP) [7], [12], [13]. In our previous work [18], we have shown that when caches in a network are operated independently (with their own cache replacement policy such as LRU), the utilization of intermediate caches can be extremely poor, due to the "thrashing" problem caused by the (non-independent) *filtered* (cache miss) arrival processes at intermediate servers. To address this problem, we proposed the innovative notion of *"BIG" cache abstraction* [18] by viewing a line of caches from an edge server along the path to the origin

ISBN 978-3-903176-08-9 © 2018 IFIP

server as a single "BIG" cache, and argued that it affords the added benefits of simplifying the analysis of a line of caches.

Unlike a single cache where the performance of various caching policies can be directly compared, evaluating and comparing the (relative) performance of caching policies for a network of caches are no longer straightforward. From a user's perspective, hit probabilities at individual caches are immaterial; what matters is the latency he/she experiences. On the other hand, a cache network provider is more concerned with the efficient utilization of *all* cache capacities in the network; whereas from the standpoint of a content provider, the utility of a cache network is its ability to decrease the overall load on its origin server(s), and reduce the network bandwidth cost (it also cares about improved content access latencies to its users). Despite much focus on performance analysis of cache networks, this important problem has not received much attention in the research community.

In this paper, we propose a novel and general framework to evaluate and compare caching policies for a network of caches. Consider a collection of content objects served by a network of caches with a fixed set of ingress points (or edge servers) where user requests for content are routed. We assume caches are organized in a hierarchy, from the edge servers to the origin server(s). Given the request streams for the collection of objects at each ingress point (edge server), we introduce the notion of a *hit probability matrix*, which characterizes the hit probability of content objects that are served at different layers of the cache hierarchy. We employ (and define an extended version of) the *majorization* notion as the basic tool to evaluate and compare caching policies for various performance metrics of cache networks in Section II.

Section III provides an overview of the existing works to estimate the hit probability matrix, including their limitations. Then, we provide and evaluate a general simplified approach to estimate the hit probability matrix based on the "BIG" cache abstraction in Section IV. As we have shown in [18], implementing caching policies as a single caching strategy for the virtual "BIG" cache outperforms their implementation at each layer independently. In this paper, we show that our approach to estimate the hit probability matrix achieves the exact result as the "BIG" cache simulation for LRU and q-LRU caching strategies as examples. Thus, this estimation approach can be used to generate the hit matrix for the comparison framework without wasting any time on simulations. Section V shows the accuracy of applying our proposed framework to compare caching policies for different user request patterns. Finally, the paper is concluded in Section VI.

II. Caching Policies Comparison Framework

In this section, we present a general framework to evaluate and compare two caching policies, P and Q, for network-wide performance analysis, where P, Q represent any caching policy such as LRU, static caching, k-LRU, ... etc. We first describe the network model, basic assumptions, and the key notion of hit probability matrix associated with a caching policy (here we assume it is given). Then, we identify the conditions for policy P to *dominate* policy Q by generalizing the notion of majorization defined for vectors to matrices.

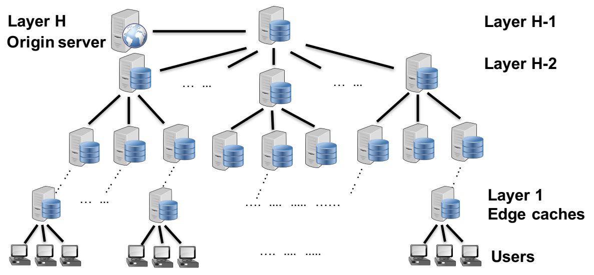


Figure 1. Network Model

A. Network Model and Assumptions

For ease of exposition, we make a simplifying assumption and model the cache network as a hierarchy of cache servers organized in an $(H-1)$ level k-ary tree, as shown in Figure 1, which is commonly used in today's content distribution networks [21], [22]. Cache servers at leaf nodes represent edge servers, which are the closest to users located at layer 1. The root of the tree is connected to the origin server (at layer H, which has a permanent copy of each object). The content population is a collection of N unique objects of unit size, denoted by $\mathcal{O} = \{O_1, O_2, \ldots O_N\}$. The popularity of objects follows a Zipf distribution with parameter α. The access probability of each object is denoted by $a_i = \lambda_i/\lambda$, where λ_i is the request rate of object i, and λ is the aggregate request rate, $\lambda = \sum_i \lambda_i$. The access probability a_i is proportional to $\frac{1}{i^\alpha}$ for $\alpha > 0$, and $\sum_{i=1}^{N} a_i = 1$. Without loss of generality, we assume that $a_1 \geq a_2 \geq \ldots \geq a_N$, namely, O_1 is the most popular object, O_2 is the second most popular object, and so forth. Object requests arrive randomly at one of the edge servers. When a request is received at a server, which does not have the object, it forwards the request to its parent. This process continues till the request reaches the root, and the origin server eventually if no other server on the path has a copy. Each request experiences a latency depending on the layer it is served from. First, we consider comparing the caching policies P and Q for a tandem cache network (a line of caches) starting from one edge cache C_e at layer 1, till the origin server at layer H. The concepts and notations introduced below can be generalized to a cache network of any arbitrary topology, where we construct a request routing tree/graph formed by the request forwarding paths from each edge server towards an origin server as illustrated at the end of this section.

B. Tandem Cache Network

We define L_j to denote the latency experienced by an object served from a cache server at layer j. We assume $L_1 < L_2 < \ldots < L_H$ and $L_j = L_{j-1} + \Delta L_{j-1}$, $2 \leq j \leq H$. For a caching policy P, $a_i p_{ij}$ represents the hit probability of object O_i at layer j cache, where $1 \leq j \leq H$, p_{ij} is the percentage of requests for O_i served from layer j cache, and a_i is the access probability of object O_i.

The set of values $\{a_i p_{ij}\}, 1 \le i \le N, 1 \le j \le H$ can be compactly represented using an $N \times H$ matrix, and by abuse of notation, we denote it as "P".

$$P_{N \times H} = \begin{bmatrix} a_1p_{11} & a_1p_{12} & a_1p_{13} & \dots & a_1p_{1H} \\ a_2p_{21} & a_2p_{22} & a_2p_{23} & \dots & a_2p_{2H} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_Np_{N1} & a_Np_{N2} & a_Np_{N3} & \dots & a_Np_{NH} \end{bmatrix}$$

Majorization of Hit Probability Matrices. We employ the notion of *majorization* as the basic tool to evaluate and compare caching policies for a cache network. The standard notion of majorization is defined for vectors. We generalize it to a (hit probability) matrix as follows. Given two caching policies P and Q, with hit probability matrices represented by $P = [a_i p_{ij}]$ and $Q = [a_i q_{ij}]$ respectively. Without loss of generality, we assume both policies are "sensible" at layer 1 cache – namely, the first column is decreasing in value. As caching policies react based on the received requests, thus popular objects are cached more often than others, which is also confirmed by the simulation results shown in [18] for different caching policies. Therefore, we assume the following holds:

$$\begin{aligned} p_{11} \ge p_{21} \ge p_{31} \ge \dots \ge p_{N1} \\ q_{11} \ge q_{21} \ge q_{31} \ge \dots \ge q_{N1} \end{aligned} \tag{1}$$

Also, the summation of the hit probabilities of object O_i for all H layers equals 1 under both policies, *i.e.*, a request for object O_i has to be served from one of the H layers.

$$\sum_{j=1}^{H} p_{ij} = \sum_{j=1}^{H} q_{ij} = 1, \quad \forall \ 1 \le i \le N \tag{2}$$

Given the above conditions, we say P *majorizes* Q, denoted by $P \succ Q$, if the following criterion holds:

The summation of the hit probabilities in the top-left (k, h) sub-matrix of P is equal or larger than that of Q for all values of k, h, where $1 \le k \le N, 1 \le h \le H$, (*i.e.*, policy P utilizes the first h cache layers to serve the top k objects better than policy Q).

$$\sum_{i=1}^{k}\sum_{j=1}^{h} a_i p_{ij} \ge \sum_{i=1}^{k}\sum_{j=1}^{h} a_i q_{ij} \tag{3}$$

Thus, we say caching policy P **dominates** Q if and only if $P \succ Q$ (*i.e.*, P majorizes Q).

Comparing Overall Performance. Having two general caching policies P, Q, where P dominates Q (from the perspective of the edge server), we show that P outperforms Q in terms of both the overall latency (as seen by the end user), and the load of the origin server.

*1) **Expected Overall Latency**:*

Theorem 1: For a hierarchy of caches of H layers, if caching policy P dominates caching policy Q, the overall expected latency for P is less than or equal to that of Q.

Proof: Expected latency under caching policy P (OL_P):

$$OL_P = \sum_{i=1}^{N}(a_i \sum_{j=1}^{H}(p_{ij} L_j)) \tag{4}$$

Since, $L_j = L_H - \sum\limits_{h=j}^{H-1} \Delta L_h, \ 1 \le j \le H-1$

By substituting for L_j in "(4)", and rearranging the summation indices:

$$OL_P = \sum_{i=1}^{N}(a_i \sum_{j=1}^{H}(p_{ij} L_H)) - \sum_{h=1}^{H-1}(\Delta L_h \sum_{i=1}^{N}(a_i \sum_{j=1}^{h} p_{ij}))$$

From "(2)" and $\sum_{i=1}^{N} a_i = 1$:

$$OL_P = L_H - \sum_{h=1}^{H-1}(\Delta L_h \sum_{i=1}^{N}\sum_{j=1}^{h} a_i p_{ij})$$

From "(3)":

$$OL_P \le L_H - \sum_{h=1}^{H-1}(\Delta L_h \sum_{i=1}^{N}\sum_{j=1}^{h} a_i q_{ij})$$

$$OL_P \le OL_Q$$

Intuitively, caching policy P dominates Q means that under policy P, the top k most popular objects are likely to be placed in the first h layer caches than under policy Q, for $1 \le k \le N$, $1 \le h \le H$. Given that $L_1 < L_2 < \dots < L_H$, we would expect that P outperforms Q in terms of expected latency. ■

*2) **Origin Server Load**:*

Theorem 2: For a hierarchy of caches of H layers, if caching policy P dominates caching policy Q, the origin server load under policy P is less than or equal to that of policy Q.

Proof: Origin server load under caching policy P (S_P):

$$S_P = \sum_{i=1}^{N} a_i p_{iH} \tag{5}$$

From "(2)" and $\sum_{i=1}^{N} a_i = 1$:

$$S_P = 1 - \sum_{i=1}^{N}\sum_{j=1}^{H-1} a_i p_{ij}$$

From "(3)":

$$S_P \le 1 - \sum_{i=1}^{N}\sum_{j=1}^{H-1} a_i q_{ij}$$

$$S_P \le S_Q$$

Intuitively, caching policy P dominates Q means that under policy P, the top k most popular objects are likely to be placed in the first h layer caches than under policy Q, for $1 \le k \le N$, $1 \le h \le H$. Since, the number of requests is directly proportional to the object popularity, we expect more requests to be satisfied form the first h layer caches under policy P. Thus, the origin server load under policy P would be less than or equal to that of Q. ■

C. General Cache Networks

We now discuss how this framework can be utilized for any general cache network. Consider a cache network as shown in Figure 1. Each edge server $C_e \in \mathcal{E}$ (set of all edge servers) is deployed to service content requests from one user populace located close to C_e. The content population of edge C_e is denoted by $\mathcal{O}_e = \{O_1^e, O_2^e, \dots O_{N^e}^e\}$, and the object popularity follows a Zipf distribution with parameter α_e. The access probability of each object is denoted by $a_i^e = \lambda_i^e / \lambda_e$, where λ_i^e is the request rate of object O_i^e, λ^e is the aggregate request rate for the edge server C_e, and $\lambda_e = \sum_i \lambda_i^e$.

A user's request for an object O_i^e is first routed to the edge server C_e closest to her. The request is serviced directly by

C_e if it has O_i^e cached; otherwise C_e routes the request along a request path $\mathcal{L}^e$ – consisting of a sequence of intermediate cache servers, $C_h^e \in \mathcal{L}^e$ – towards the origin server. If one of the intermediate servers, C_h^e, has O_i^e cached, the request is serviced by C_h^e, and the cached copy is returned along the reverse path back to C_e, which then delivers it to the user. Otherwise, the request is serviced by the origin server. Thus, we consider the hierarchical topology as multiple tandem cache networks, each corresponding to an edge server C_e. The request paths $\mathcal{L}^e$ from multiple edge servers traverse and share the cache resources at the intermediate cache servers C_h.

Given two caching policies P and Q, and their hit probability matrices $P^{(e)}$ and $Q^{(e)}$ respectively from the prospective of each edge cache server C_e. We can then apply the comparison framework detailed earlier to compare the two caching policies P and Q for each edge server C_e. Clearly, if $P^{(e)} \succ Q^{(e)}$ for each edge server C_e, then $P = \sum_e P^{(e)} \succ \sum_e Q^{(e)} = Q$, where $P = \sum_e P^{(e)}$ is the aggregate hit probability matrix over all edge servers, as shown in Section V. The aggregate hit probability matrix can be determined using approaches such as [7], in which the authors extend their proposed method (discussed in Section III) for a general cache network, and use the miss rate of lower layer caches as an estimate of the request arrival rate for the current cache node. Finally, these concepts and notations can be generalized to a cache network of any arbitrary topology, where we construct a request routing tree/graph formed by the request forwarding paths from each edge server towards an origin server. Moreover, the cache network does not have to be symmetric, the tandem cache network from each edge server till an origin server could have a different height, and the missing layers could be replaced by dummy cache nodes with zero capacity. Hence, their hit probability is zero, and our proposed comparison framework would still be applicable.

III. Hit Probability Matrix Estimation using TTL Caches

As mentioned before, the analysis of cache networks is a complex and challenging task. This is because the request arrival streams at the upstream caches are not independent – they are generated by cache misses from the downstream caches. Thus, for each cache in the network, the miss rate of content objects should be calculated, along with splitting the miss streams to the other upstream caches. Moreover, the superposition of the miss streams which form the requests arriving at the intermediate caches needs to be calculated. Therefore, several approaches (*e.g.*, [7], [12], [15], [23]) have been proposed to estimate the requests arrival rate at intermediate caches. Thus, the object hit probability for every cache in the network can be calculated.

The complexity of the analysis of capacity-based caching policies is due to the dynamics of the content objects in the cache. Hence, Che *et al.* [15] proposed the relationship between a capacity-based caching policy (LRU) and a timer-based caching, by defining the characteristic time. Timer-based caching also simplifies the performance analysis of caching systems, as it decouples the dynamics of content objects within a cache, as they can be analyzed independently. In this section, we categorize some of the existing TTL-based approaches according to the type of the approximations they proposed to estimate the request arrival rates at caches, and briefly discuss their limitations.

A Hierarchical Network of LRU Caches with LCE, LCP, and LCD Replication Strategies. The authors in [7] provided a unified approach to analyze the performance of caching policies such as LRU, FIFO, RANDOM, q-LRU, k-LRU, ... etc. for single cache, by extending the decoupling technique introduced in [15]. The authors also analyzed LRU for a two-layered cache network with respect to the following object replication strategies, which define how the object is cached when it is returned from an upstream cache (or the origin server) back down along the request path. *1) Leave-Copy-Everywhere* (LCE): the object is cached at all the downstream caches along the request path. *2) Leave-Copy-Probabilistically* (LCP): the object is cached at the downstream caches along the request path with a probability q. *3) Leave-Copy-Down* (LCD): the object is cached only at the cache preceding to the one where it is currently cached.

For a single cache, the authors considered a temporal locality relation between requests by considering renewal process for request arrival process. However, the key assumption for a hierarchical network of caches is that the request arrival process at any cache in the network is Poisson (IRM request traffic). The existing spatial-correlation and temporal-correlation among requests are ignored as the requests interarrival process is considered Poisson at the intermediate caches. The authors justified this assumption by mentioning that the error gets smaller as the network grows (in terms of the number of branches), hence, the proposed model becomes valid. Finally, the authors proposed another extension for their model to work for a network of caches with any topology.

Equations "6" & "7" define the hit probability at each layer for object m in an LRU-cache with LCE & LCP object replication strategies respectively. The hit probability of object m at cache i is denoted by $p_{hit}(i,m)$. T_C^i is the timer assigned to all objects at cache i which is related to the cache size, and q is the probability to cache objects for LCP. The *average arrival rate* for object m at cache i is calculated by: $\overline{\lambda}_m(i) = \sum_j \overline{\lambda}_m(j)(1 - p_{hit}(j,m))r_{j,i}$, where $r_{j,i}$ is the probability that cache j forwards its miss stream to cache i. We use these equations in Section V to calculate the hit probability matrix, and compare them using our proposed framework.

LCE

$$\begin{aligned} p_{hit}(1,m) &= 1 - e^{-\overline{\lambda}_m(1)T_C^1} \\ p_{hit}(2,m) &\approx 1 - e^{-\overline{\lambda}_m(2)(T_C^2 - T_C^1)} \end{aligned} \tag{6}$$

LCP

$$\begin{aligned} p_{hit}(1,m) &= \frac{q(1 - e^{-\overline{\lambda}_m(1)T_C^1})}{e^{-\overline{\lambda}_m(1)T_C^1} + q(1 - e^{-\overline{\lambda}_m(1)T_C^1})} \\ p_{hit}(2,m) &\approx [p_{hit}(2,m) + q(1 - p_{hit}(2,m))] \\ &\qquad (1 - e^{-\overline{\lambda}_m(2)(T_C^2 - T_C^1)} e^{-\overline{\lambda}_m(2)(1-q)T_C^1}) \end{aligned} \tag{7}$$

Approximate Analysis of Hierarchical and General TTL-Cache Networks. Fofack *et al.* [10], [12], [24] focus on analyzing the performance of LRU, FIFO, and RANDOM caching policies using the characteristic time. The timer is set up so that the number of objects in the cache does not exceed the cache size, considering the same size for all objects. The key assumption of this model is considering the requests arriving at any cache in the network to have a renewal process interarrival time. The authors also assume that requests are forwarded in a feed-forward network in contrast with [7]. This approach characterizes three request streams: i) the miss stream of each cache, ii) the splitted miss streams to next layer caches, and iii) the superposition of request streams arriving from different caches along with exogenous request arrival as a renewal process. The main source of error of this approach is considering the superposition of renewal processes as a renewal process, which is a non-renewal in general. Moreover, this approach is computationally very extensive when the size of the cache network grows, thus, it is not scalable. Finally, it has the limitation of assuming the routing of the cache network to be feed-forward.

Exact Analysis of a Hierarchical Network of TTL-Caches. Following the approach in [12], Berger *et al.* [23] propose an exact model for performance evaluation of a hierarchical network of caches. The key contribution of this approach is adopting Markovian Arrival Process (MAP) for request arrival processes, and showing that the miss stream of TTL-based caches is MAP as well. Since the superposition of MAP processes is also MAP, the provided analysis are the exact values for feed-forwarded cache networks with a MAP process as an input for all caches in the network. However, this approach suffers from the computation cost for large networks, as well as [12], and suffers also in case of non feed-forward cache networks.

IV. General Approach for Hit Probability Matrix Estimation

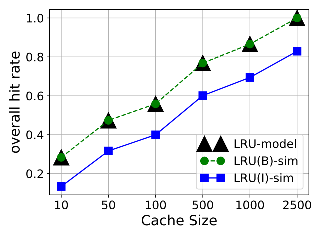
The input to our proposed comparison framework (Section II) is the hit probability matrices of both policies ones wish to compare, which on general is a challenging task to calculate as discussed in Section III. Ether of exact or approximatation calculated values are known for specific network set ups [7], [12], [23], assumptions about the request inter-arrival processes. As mentioned, currently, there exists no *general* methodology for computing, approximating or even bounding the hit probabilities for a *network* of caches within a reasonable computation cost. Even with the simplifying IRM assumption, the request arrival processes to intermediate network caches are *filtered*, and no longer independent. This creates technical difficulty in analyzing a cache network for any arbitrary caching policy under general assumptions of request arrival processes. Thus, in this section for a caching policy P, given the equations to calculate the hit probability for a single cache, we propose a general simplified approach to calculate the hit probability matrix for a tandem cache network, using the notion of "BIG" cache abstraction introduced in [18]. At the end of this section, we extend this approach to be applied for any general cache network.

The main idea of "BIG" cache is to view a group of hierarchical caches as if they are "glued" together to form one virtual "BIG" cache with a storage capacity distributed across multiple layers. Consider a tandem cache network of H layers, where C_H represents the origin server, and C_1 the edge server, where requests are first received. Assume, the cache size of each layer is denoted by C_j, $1 \le j < H$. The size of the virtual "BIG" cache is denoted by $C_B := \sum_{j=1}^{H-1} C_j$. Then, any caching policy can be directly applied to this one (virtual) "BIG" cache as a single consistent strategy. Objects can be cached in any layer of the hierarchy, and moved between cache boundaries of different layers according to the caching policy (see [18] for more details). Using this "BIG" cache abstraction, the cache network can be viewed as a single "giant" (blackbox) cache with storage capability C_B, receiving multiple streams of content requests $\{\lambda_i\}$. The goal of any caching policy is to maximize the overall hit probability of the entire cache network, and minimize the load at the origin server. We now explain in details how "BIG" cache abstraction allows us to estimate the hit probability matrix for a line of caches, given the hit probability $p_i(C, \lambda_i, P)$ of object O_i under caching policy P as a function of the cache size C, and the object request rate λ_i.

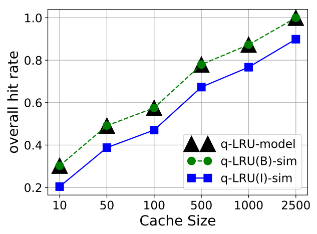
Consider a line of caches with cache sizes C_h, $1 \le h < H$. We can view the caches from layer 1 to layer j as if they represent a single virtual cache of total capacity $C_{[1:j]} := \sum_{h=1}^{j} C_h$. Thus, $p_i(C_{[1:j]}, \lambda_i, P)$ (upper) bounds the probability of serving requests for object O_i from one of the first j caches under P, and $\tilde{p}_{ij} = p_i(C_{[1:j]}, \lambda_i, P) - p_i(C_{[1:j-1]}, \lambda_i, P)$[1] yields an estimate (upper bound) of p_{ij}, the probability that requests for object O_i are served by cache C_j. Knowing the hit probability of object O_i at $L1$, we can find its probability at $L2$. Then, we repeat this process iteratively till layer L_{H-1}. Then, p_{iH} can be calculated as $1 - \sum_{j}^{H-1} p_{ij}$, which represents the percentage of requests served from the origin server. Hence, we can calculate the hit probability matrix. Thus, "BIG" cache abstraction completely avoids the aforementioned technical challenges and interdependency between cache layers, such as the filtered requests at intermediate layers. Since content objects can be stored at any layer of the cache hierarchy, the overall hit probability of each object $p_i = \sum_{h=1}^{H-1} p_{ih}$ is not affected by "BIG" cache since the percentage of requests satisfied by the cache network does not change based on the location of the object in the hierarchy, as the total caching capacity is the same. Hence, the origin server load is not affected either. However, the user's latency depends on which layer the object is served from, which depends on the caching policy. Therefore, our proposed approach can be used to estimate the hit probability matrices for different caching policies. Then, our comparison framework mentioned in Section II can be used to find the appropriate policy for the given request traffic.

Estimation Approach Validation. We evaluate our proposed

[1]This linear relationship is valid under IRM model assumption.



(a) LRU



(b) q-LRU (q = 0.5)

Figure 2. Hit Probability Matrix Estimation

approach to estimate the hit probability matrix by comparing it to the simulation results for some known caching policies, and also to show how implementing these caching policies using "BIG" cache abstraction always enhances their performance when they are implemented at each cache layer independently.

LRU

$$C = \sum_i p_{hit}(i) = 1 - e^{-\lambda_i T_C} \tag{8}$$

q-LRU

$$C = \sum_i p_{hit}(i) = \frac{q(1-e^{-\lambda_i T_C})}{e^{-\lambda_i T_C} + q(1-e^{-\lambda_i T_C})} \tag{9}$$

From [7], and considering Poisson interarrival process for requests, we use "(8)" to calculate the hit probability of each object in a single cache of size C for LRU (always cache a copy of the requested object), and use "(9)" for q-LRU (cache a copy of the object with probability q). These equations calculate the characteristic time T_C for cache using the cache size C and the request rates for each object $\{\lambda_i\}$. Using our proposed approach, we calculate the hit probability matrix by considering the cache size of the first j layers $C_{[1:j]} := \sum_{h=1}^{j} C_h$, which would give us a new value for the characteristic time using the corresponding equation "(8)" or "(9)" according to the caching policy. Using this new characteristic time, we can calculate the probability that the object is being served from the first j layers.

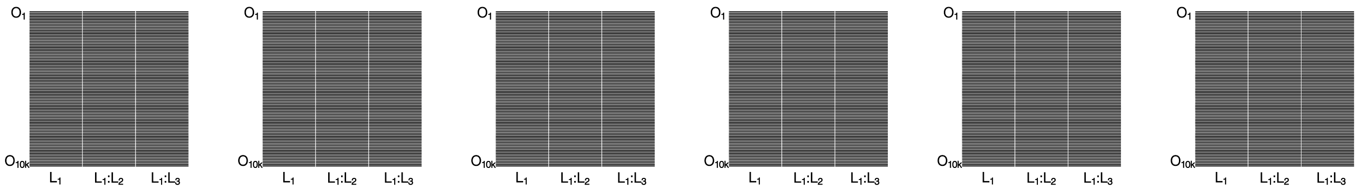
We compare the hit probability matrix calculated by our proposed approach with simulation results. We consider a line of five caches, and a collection of $10K$ unique objects. The edge cache server, and intermediate caches, have the same cache size, ranging from $[10, 50, 100, \ldots, 2500]$. User requests follow Zipf-distribution with $\alpha = 1$. We use LRU as the cache eviction policy at each cache layer, and LCE as the object replication strategy when the object is traversing a request path back to the user. The results are shown in Fig. 2a, in which *LRU(I)-sim* represents the simulation results, and *model* represents our proposed approach results. We also simulate LRU using "BIG" cache abstraction, which maintains one copy at a time at any layer due to applying LRU as a single caching strategy for the "virtual" "BIG" cache, where requested objects are always cached at the first layer $L1$ (edge server), and when $L1$ is full, the evicted objects are cached in $L2$, and so on (denoted by *LRU(B)-sim*).

The result shown in Fig. 2a indicates that *LRU(B)-sim* outperforms LRU-LCE. This is confirmed by the results in [18], which show that applying existing caching policies to a hierarchical network as a single cache, improves the performance, instead of having each cache layer taking decisions independently. In *LRU(B)-sim*, we considered the available storage as a single aggregated cache capacity which leads to caching one copy at most. Whereas other policies like *LRU(I)-sim* (LRU-LCE) results in having more than one copy of the requested object at different cache layers. Having only one copy at the cache, leaves more space for other objects to be cached, and hence improves the overall hit probability, minimizes the latency and origin server load. Our proposed estimation (*model*) consideres available storage as one single "virtual" cache with the aggregate cache capacity allowing one copy of an object in cache. In Fig. 2a it is observed that the values of (*model*) estimations and *LRU(B)-sim* matches. As *LRU(B)-sim* outperforms other policies [18], our proposed method for estimation of hit probability matrix provides an upper bound for other approximations. Similar results are shown for q-LRU in Fig. 2b.

General Cache Network. Finally, this approach can be applied to any general cache network topology using the idea discussed in Section II-C by constructing a path from each edge server to an origin server. However, to apply the "BIG" cache approach to estimate the hit matrix for each line of caches, we need to (*logically*) partition the cache resources at intermediate cache servers through which the request paths $\mathcal{L}^e$ from multiple edge servers traverse, and allot appropriate cache resources to each edge server C_e. Taking into account, the characteristics of the requests of each user populace (*i.e.*, edge server) in terms of object popularity, request interarrival distribution, ... etc, with the goal of optimizing the performance objectives. For an intermediate server C_h, $h \in \mathcal{L}^e$, let C_h^e be the portion of its cache C_h that is (logically) allotted to C_e. In other words, the cache C_h is logically partitioned into multiple pieces, C_h^e's, among the edge servers; $\sum_e C_h^e = C_h$ (here by abuse of notation we also use C_h^e and C_h to denote the cache size). Collectively, C_h^e's, $h \in \mathcal{L}^e$, form a tandem cache network with respect to the edge server C_e; its total size is $C^e := \sum_{h \in \mathcal{L}^e} C_h^e$.

For each tandem cache network corresponding to an edge server C_e, let u_i^e be the object occupancy probability, and let $U^e(\cdot)$ be a generic (*concave*) objective (or utility) function (of object occupancy or allotted cache size). We can formulate the following cache partition/allotment optimization problem. The solution[2] of this optimization problem indicates how to partition the cache resources at each intermediate cache, and allot them to the tandem cache network of each edge server to maximize the hit probability of the entire cache network. Once we have these partitions, we can apply the previously mentioned approach for each edge cache and estimate its hit probability matrix.

[2]Due to space limit, we do not elaborate on the solution of the optimization problem here.



(a) CS = 50 (b) CS = 100 (c) CS = 500 (d) CS = 1000 (e) CS = 2500 (f) CS = 3000

Figure 3. Majorization Conditions "(3)" for the caching policies LCP & LCE in Section III

$$\begin{aligned} \underset{u_i^e\in[0,1],C_h^e}{\text{maximize}} \quad & \sum_{e\in\mathcal{E}}\sum_{i}^{N} U_i^e(u_i^e) \\ \text{s.t.} \quad & \sum_{i=1}^{N} u_i^e \le C^e, \forall e\in\mathcal{E}; \quad \sum_{h\in\mathcal{L}^e} C_h^e = C^e, e\in\mathcal{E}; \\ \text{and} \quad & \sum_{e\in\mathcal{E}} C_h^e \le C_h, h\in H, \end{aligned} \tag{10}$$

V. Evaluation

The goal of this section is to validate our proposed policy comparison framework, and show its evaluation in all the considered cases through simulation. Using common existing caching policies such as static caching, LRU, ... etc, we show that if policy P outperforms policy Q, then P *majorizes* Q according to our definition in Section II, and show that the majorization condition "(3)" is satisfied. First, we start by applying the comparison framework to a tandem line of caches. Then, we show the framework extension (discussed in Section II-C) applied to a hierarchical cache network.

A. Tandem Cache Network

We use a hierarchical network organized as a line of four caches. Edge server lies at $L1$, where user requests are first received. The origin server lies at $L4$, which serves a collection of $10K$ unique objects each of a unit size. The edge server and intermediate caches have the same cache size, which ranges from $[50, 100, \dots, 3000]$. User requests follow Zipf-distribution with $\alpha = 0.8$. Each object has a request rate λ_i, where the aggregate request rate $\lambda = \sum_i \lambda_i = 1$. We simulated two distributions for the request intervarrival times of each object: 1) Poisson & 2) Pareto (w. parameter 2) [20]. We compare the following caching policies: static, q-LRU, LRU with different object replication strategies (LCD & LCE) [7].

Comparison Framework Validation. Using the hit probability matrices of policies P & Q, we calculate new corresponding matrices, $\hat{P}$ & $\hat{Q}$, where $\hat{p}_{kh} = \sum_{i=1}^{k}\sum_{j=1}^{h} a_i p_{ij}$, $1 \le k \le N, 1 \le h \le H$, similarly for $\hat{Q}$. $\hat{P}$ & $\hat{Q}$ represent the summation of all the possible submatrices of P & Q respectively. Finally, we define a comparison matrix X_{P-Q}, where $x_{ij} = 1$ if $\hat{p}_{ij} \ge \hat{q}_{ij}$; and 0 otherwise, where $1 \le i \le N, 1 \le j \le H$. The matrix X_{P-Q} is the representation of the majorization condition "(3)". If all elements of matrix X_{P-Q} are equal to one, then policy P *dominates* policy Q. We visualized X_{P-Q} using a heatmap, representing 1 as black, and 0 as white.

Analytically. We use the equations defined for LRU-LCE and LRU-LCP "6" & "7". The only change to the simulation settings mentioned at the beginning of this section is the number of layers, as these equations are defined for a network of 3 layers. For each policy, we calculate the characteristic time of each cache, then the hit probability matrix. We apply our comparison framework to compare their performance. The heatmap comparing LRU-LCP and LRU-LCE is shown in Fig. 3. As expected, LRU-LCP dominates LRU-LCE for the different cache sizes.

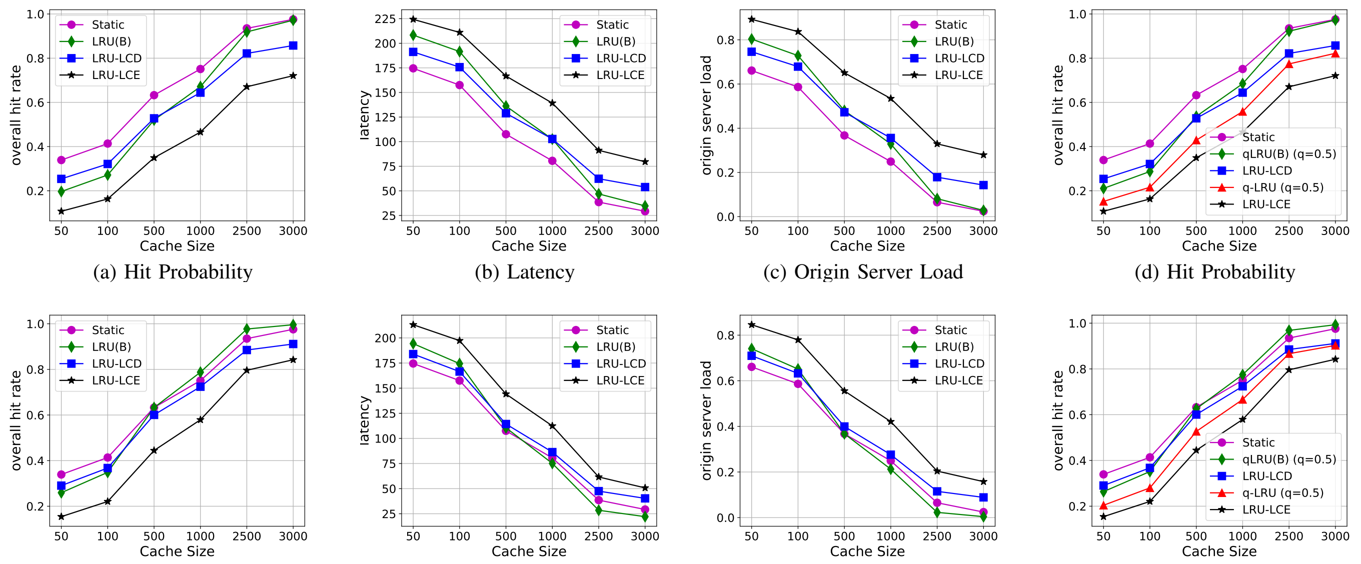
Table I
Poisson

P, Q \ CS	50	100	500	1000	2500	3000
Static, LRU(B)	✓	✓	✓	✓	✓	✓
Static, LRU-LCD	✓	✓	✓	✓	✓	✓
Static, LRU-LCE	✓	✓	✓	✓	✓	✓
LRU-LCD, LRU(B)	✓	✓	✓	✗	✗	✗
LRU-LCD, LRU-LCE	✓	✓	✓	✓	✓	✓
LRU(B), LRU-LCE	✓	✓	✓	✓	✓	✓
Static, qLRU(B)	✓	✓	✓	✓	✓	✓
Static, qLRU	✓	✓	✓	✓	✓	✓
qLRU(B), qLRU	✓	✓	✓	✓	✓	✓
qLRU(B), LRU-LCE	✓	✓	✓	✓	✓	✓
qLRU, LRU-LCE	✓	✓	✓	✓	✓	✓
LRU-LCD, qLRU	✓	✓	✓	✓	✓	✓

Simulation. We implemented several caching policies to compare their performance using our proposed framework. Fig. 4 shows the overall performance of these caching strategies in terms of the average hit probability ($\sum_{i=1}^{N}\sum_{j=1}^{H} a_i p_{ij}$), latency and origin server load for both Poisson (top row) & Pareto (bottom row) request interarrival distributions. As expected in case of Poisson request interarrival distribution, static caching is shown to be optimal [19] if the object popularity distribution is known *a priori*, followed by LRU-LCD, LRU-LCP(q-LRU), and LRU-LCE caching policies (similar to results reported in [7]). The latency and origin server load follows the same behavior of the average hit probability, thus are not included for the other policies due to space limitations. Figs. 4a, 4d show LRU(B) & q-LRU(B), implemented using "BIG" cache abstraction [18].

LRU(B) & q-LRU(B) outperform their corresponding caching policies when implemented independently at each cache. Moreover, their performance is close to static caching performance (similar to results reported in [18]). However, as shown in [20] static caching is no longer optimal when the interarrival distribution of requests has a decreasing hazard rate (*e.g.*, when the interarrivals of requests are Pareto-distributed), we can see that LRU(B) & q-LRU(B) achieve better performance when the cache size is large as shown in Figs. 4e, 4h.

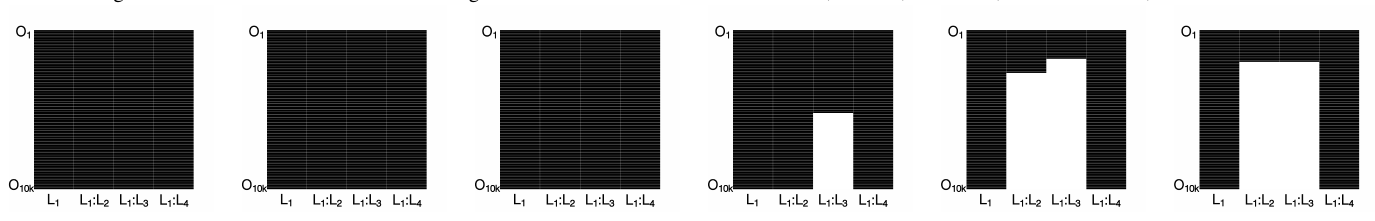
The heatmap of comparing LRU-LCD & LRU(B) in both



(a) Hit Probability (b) Latency (c) Origin Server Load (d) Hit Probability

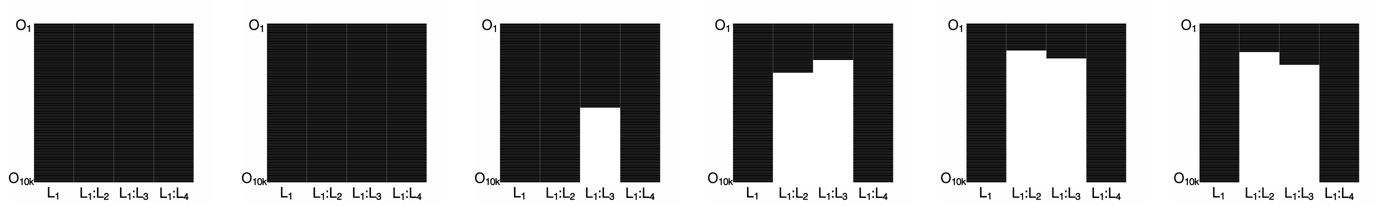
(e) Hit Probability (f) Latency (g) Origin Server Load (h) Hit Probability

Figure 4. Overall Performance for Caching Policies for a network with $N = 10K$, $H = 4$, $R = 5M$, Poisson 1st row, Pareto 2nd row



(a) CS = 50 (b) CS = 100 (c) CS = 500 (d) CS = 1000 (e) CS = 2500 (f) CS = 3000

Figure 5. Majorization Conditions "(3)" for the caching policies LRU-LCD & LRU(B) Poisson in Fig. 4



(a) CS = 50 (b) CS = 100 (c) CS = 500 (d) CS = 1000 (e) CS = 2500 (f) CS = 3000

Figure 6. Majorization Conditions "(3)" for the caching policies LRU-LCD & LRU(B) Pareto in Fig. 4

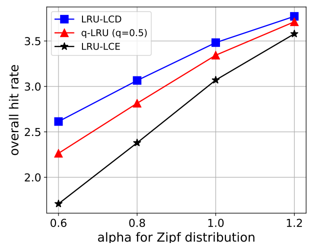
Table II
PARETO

P, Q \ CS	50	100	500	1000	2500	3000
Static, LRU(B)	✓	✓	✓	✗	✗	✗
Static, LRU-LCD	✓	✓	✓	✓	✓	✓
Static, LRU-LCE	✓	✓	✓	✓	✓	✓
LRU-LCD, LRU(B)	✓	✓	✗	✗	✗	✗
LRU-LCD, LRU-LCE	✓	✓	✓	✓	✓	✓
LRU(B), LRU-LCE	✓	✓	✓	✓	✓	✓
Static, qLRU(B)	✓	✓	✓	✗	✗	✗
Static, qLRU	✓	✓	✓	✓	✓	✓
qLRU(B), qLRU	✓	✓	✓	✓	✓	✓
qLRU(B), LRU-LCE	✓	✓	✓	✓	✓	✓
qLRU, LRU-LCE	✓	✓	✓	✓	✓	✓
LRU-LCD, qLRU	✓	✓	✓	✓	✓	✓

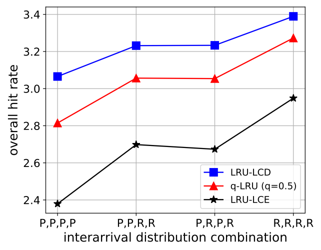
cases (Poisson and Pareto distributions) are shown in Fig. 5 & Fig. 6 respectively, where they accurately reflect the dominance of LRU-LCD over LRU(B) only when the cache size is small as their corresponding average hit probability. Tables I & II summarize the comparison of every two policies P, Q for Poisson & Pareto interarrival distributions respectively for the different cache sizes, where ✓ is used if P dominates Q, and ✗ otherwise. The results in these tables reflect 100% accuracy for our proposed comparison framework in determining the dominance of the policies in comparison with respect to their simulated overall performance.

B. *Tree Topology*

We use a hierarchical network organized as a binary tree of three levels (*i.e.*, four edge servers). Edge servers lie at $L1$, where user requests are first received. The origin server lies at $L4$, which serves a collection of $2K$ unique objects each of a unit size. The size of the cache servers at layers [$L1$, $L2$, $L3$] are $[400, 800, 1600]$. We compare the following caching policies: q-LRU and LRU with different object replication strategies (LCD & LCE) [7].



(a) Changing Alpha



(b) Changing Interarrival Distr.

Figure 7. Hierarchical Tree Cache Network

First Scenario. The distribution of the interarrival time is Poisson. We simulated 4 different values of α for zipf-distribution $[0.6, 0.8, 1.0, 1.2]$ for all the edge servers. The average hit probability is shown in Fig. 7a. We find the relationship between the three policies is still as expected, and when α increases, the average hit probability increases.

Second Scenario. We use $\alpha = 0.8$ for all edge servers, and we changed the distribution of interarrival time for the four edge servers as following: $['P, P, P, P', 'P, P, R, R', 'P, R, P, R', 'R, R, R, R']$, where $'P'$ refers to Poisson and $'R'$ for Pareto for the corresponding edge server. For example, $'P, P, P, P'$ means the distr. is Poisson for all edge servers in this experiment. The average hit probability is shown in Fig. 7b.

For both scenarios, we used our proposed framework to compare each two policies at each edge server. We construct a tandem cache network from this edge server to the origin server, and calculate the hit probability matrix for the objects related to this edge server. If policy P has a better performance than policy Q, policy P majorizes policy Q for each cache server in all the different cases for α.

VI. Conclusion

In this paper, we have discussed the renewed research interest in caching policies and their performance analysis as a result of the ICN architectures. Also, the additional challenges both in terms of practical cache management issues and performance analysis for cache networks. We also have discussed some of the related work done by the research community which focuses on the performance analysis of cache networks under various caching policies, and that the issue of how to evaluate and compare caching policies for a network of caches has not been adequately addressed. In this paper, we propose and develop a novel and general framework for evaluating caching policies in a hierarchical network of caches. We introduced the notion of a hit probability matrix, and employed (a generalized notion of) majorization as the basic tool for evaluating and comparing cache policies in terms of various performance metrics for a network of caches. We discussed how the framework can be applied to various existing caching policies and conduct extensive simulation-based evaluation to demonstrate the utility of our framework.

Acknowledgment: This research was supported in part by NSF grants CNS-1411636, CNS 1618339 and CNS 1617729 and a Huawei gift.

References

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *CoNEXT*, 2009.
[2] "Named data networking," http://named-data.net/.
[3] T. Koponen and et al., "A data-oriented(and beyond)network architecture," in *ACM SIGCOMM Computer Communication Review*, 2007.
[4] S. K. Fayazbakhsh and et al., "Less pain, most of the gain: Incrementally deployable icn," in *SIGCOMM, 2013*.
[5] E. Ramadan, A. Narayanan, and Z.-L. Zhang, "Conia: Content (provider)-oriented, namespace-independent architecture for multimedia information delivery," in *ICMEW*, 2015.
[6] B. Ahlgren and et al., "A survey of information-centric networking," *IEEE Communications Magazine*, July 2012.
[7] M. Garetto, E. Leonardi, and V. Martina, "A unified approach to the performance analysis of caching systems," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 2016.
[8] A. Dan and D. Towsley, "An approximate analysis of the lru and fifo buffer replacement schemes," in *ACM SIGMETRICS*, 1990.
[9] P. R. Jelenkovi, "Asymptotic approximation of the move-to-front search cost distribution and least-recently used caching fault probabilities," *The Annals of Applied Probability*, 1999.
[10] N. C. Fofack, P. Nain, G. Neglia, and D. Towsley, "Analysis of ttl-based cache networks," in *VALUETOOLS, 2012*. IEEE.
[11] M. Dehghan, L. Massoulié, D. Towsley, D. S. Menasché, and Y. C. Tay, "A utility optimization approach to network cache design," *CoRR*, 2016.
[12] N. C. Fofack, M. Dehghan, D. Towsley, M. Badov, and D. L. Goeckel, "On the performance of general cache networks," in *ValueTools*, 2014.
[13] D. S. Berger, S. Henningsen, F. Ciucu, and J. B. Schmitt, "Maximizing cache hit ratios by variance reduction," *SIGMETRICS*, 2015.
[14] N. Gast and B. Van Houdt, "Transient and steady-state regime of a family of list-based cache replacement algorithms," *ACM SIGMETRICS Performance Evaluation Review*, 2015.
[15] H. Che, Z. Wang, and Y. Tung, "Analysis and design of hierarchical web caching systems," in *INFOCOM, 2001*.
[16] C. Fricker, P. Robert, and J. Roberts, "A versatile and accurate approximation for lru cache performance," in *ITC*, 2012.
[17] A. Wolman and et al., "On the scale and performance of cooperative web proxy caching," *ACM SIGOPS Operating Systems Review*, 1999.
[18] E. Ramadan, A. Narayanan, Z.-L. Zhang, R. Li, and G. Zhang, "Big cache abstraction for cache networks," in *ICDCS, 2017*. IEEE.
[19] Z. Liu, P. Nain, N. Niclausse, and D. Towsley, "Static caching of web servers," in *Multimedia Computing and Networking 1998*.
[20] A. Ferragut, I. Rodriguez, and F. Paganini, "Optimizing ttl caches under heavy-tailed demands," in *Proceedings of the 2016 ACM SIGMETRICS*.
[21] V. K. Adhikari, S. Jain, Y. Chen, and Z.-L. Zhang, "Vivisecting youtube: An active measurement study," in *INFOCOM, 2012*.
[22] E. Nygren, R. K. Sitaraman, and J. Sun, "The akamai network: A platform for high-performance internet applications," *SIGOPS*, 2010.
[23] D. S. Berger, P. Gland, S. Singla, and F. Ciucu, "Exact analysis of ttl cache networks," *Performance Evaluation*, 2014.
[24] N. C. Fofack and et al., "Performance evaluation of hierarchical ttl-based cache networks," *Computer Networks*, 2014.