# Accurate Learning or Fast Mixing? Dynamic Adaptability of Caching Algorithms

Jian Li, *Member, IEEE,*, Srinivas Shakkottai, *Senior Member, IEEE,* John C.S. Lui, *Fellow, IEEE,* and Vijay Subramanian, *Member, IEEE*

***Abstract*—Typical analysis of content caching algorithms using the metric of steady state hit probability under a stationary request process does not account for performance loss under a variable request arrival process. In this work, we instead conceptualize caching algorithms as complexity-limited online distribution learning algorithms, and use this vantage point to study their adaptability from two perspectives: (a) the accuracy of learning a fixed popularity distribution; and (b) the speed of learning items' popularity. In order to attain this goal, we compute the distance between the stationary distributions of several popular algorithms with that of a genie-aided algorithm that has knowledge of the true popularity ranking, which we use as a measure of learning accuracy. We then characterize the mixing time of each algorithm, i.e., the time needed to attain the stationary distribution, which we use as a measure of learning efficiency. We merge both measures above to obtain the "learning error" representing both how quickly and how accurately an algorithm learns the optimal caching distribution, and use this to determine the trade-off between these two objectives of many popular caching algorithms. Informed by the results of our analysis, we propose a novel hybrid algorithm, Adaptive-LRU (A-LRU), that learns both faster and better the changes in the popularity. We show numerically that it also outperforms all other candidate algorithms when confronted with either a dynamically changing synthetic request process or using real world traces.**

***Index Terms*—Caching Algorithms, Online Learning, Dynamic Adaptability, Markov Process, Wasserstein Distance, Mixing Time, Learning Error**

## I. Introduction

The dominant application in today's Internet is streaming of content such as video and music. This content is typically streamed by utilizing the services of a content distribution network (CDN) provider such as Akamai or Amazon. Streaming applications often have stringent conditions on the acceptable latency between the content source and the end-user, and CDNs use caching as a mean to reduce access latency and bandwidth requirements at a central content repository. The fundamental idea behind caching is to improve performance by making information available at a location close to the end-user. Managing a CDN requires policies to route requests from end-users to near-by distributed caches, as well as algorithms to ensure the availability of the requested content in the cache that is polled.

While the request routing policies are optimized over several economic and technical considerations, they end up creating a request arrival process at each cache. Caching algorithms attempt to ensure content availability by trying to learn the distribution of content requests in some manner. Typically, the requested content is searched for in the cache, and if not available, a miss is declared, the content is then retrieved from the central repository (potentially at a high cost in terms of latency and transit requirements), stored in the cache, and served to the requester. Since the cache is of finite size, typically much smaller than the total count of content, some content may need to be evicted in order to cache the new content, and caching algorithms are typically described by the eviction method employed.

Some well known content eviction policies are Least Recently Used (LRU) [1], First In First Out (FIFO), RANDOM [1], CLIMB [1], [2], LRU($m$) [3], k-LRU [4], and Adaptive Replacement Cache (ARC) [5]; these will be described in detail later on. Performance analysis typically consists of determining the hit probability at the cache either at steady-state under a synthetic arrival process (usually with independent draws of content requests following a fixed Zipf popularity distribution, referred to as the Independent Reference Model (IRM)), or using a data trace of requests observed in a real system. It has been noted that performance of an eviction algorithm under synthetic versus real data traces can vary quite widely [4]. For instance, 2-LRU usually does better than LRU when faced with synthetic traffic, but LRU often outperforms it with a real data trace. The reason for this discrepancy is usually attributed to the fact that while the popularity distribution in a synthetic trace is fixed, real content popularity changes with time [6], [7]. Thus, it is not sufficient for a caching algorithm to learn a fixed popularity distribution accurately, it must also learn it *quickly* in order to track the changes on popularity that might happen frequently.

Underlying restrictions on memory usage and computation forces caching algorithms to adopt a low complexity finite-state automata scheme. A key contribution of this paper is to argue that caching algorithm design should be viewed

Manuscript received December 10, 2017; revised April 08, 2018; accepted April 18, 2018. This work was supported in part by NSF grants CNS 1149458, AST 1443891, CNS-Intel 1719384, AST 1343381, AST 1516075, IIS 1538827, ECCS 1608361. The work by John C.S. Lui was partially supported by the GRF Grant 14200117.

J. Li is with the College of Information and Computer Sciences, University of Massachusetts Amherst, Amherst, MA, 01003, USA (e-mail: jianli@cs.umass.edu).

S. Shakkottai is with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX, 77843 USA (e-mail: sshakkot@tamu.edu).

J. C.S. Lui is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong, China (e-mail: cslui@cse.cuhk.edu.hk).

V. Subramanian is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, 48109 USA (email: vgsubram@umich.edu).

as the design of online distribution learning algorithms but using low complexity finite-state automata schemes. Note that the low complexity finite-state automata restriction precludes the use of complex dictionary constructions from universal source coding, the use of (dynamic) index policies from bandit problems or even the use of accurate empirical distribution estimation procedures. Additionally, as the complexity is held fixed (determined by cache size), no causal algorithm can learn perfectly, and hence, it places a distribution on all possible cache configurations. Therefore, the correct measure of absolute performance of an algorithm is the closeness of the distribution it assigns to the distribution of an ideal algorithm.

***Caching Algorithms as Markov Chains:*** Given the finite-state automata structure, each caching algorithm generates a Markov process over the occupancy states of the cache. Suppose there are a total of $n$ content items in a library, and the cache size is $m < n$. Then each state $\boldsymbol{x}$ is a vector of size $m$ indicating the content in each cache spot; we call the state space of all such vectors $\mathcal{S}$. Each cache request generates a state transition based on the caching algorithm used via item entrances and evictions. Hence, for a given request arrival process, a caching algorithm is equivalent to a state transition matrix over the cache states. Since there is one state transition per request, time is discrete and measured in terms of the number of requests seen.

The typical performance analysis approach is then to determine the stationary distribution of the Markov process of occupancy states, and from it derive the hit probability. However, this approach loses all notion of time, and also does not allow us to compare the performance of each algorithm with the best possible (assuming causal knowledge[1] of the request sequence). A major goal of this paper is to define an *error function* that captures the online distributional learning perspective, and hence accounts for both the error due to time lag of learning, as well as the error due to the inaccuracy of learning the popularity distribution. Such an error function would allow us to understand better the performance of existing algorithms, as well as aid in developing new ones.

***Main Contributions:*** Our goal is to design a metric that accounts for both the accuracy of and the lag in learning. To develop a good accuracy metric, we need to characterize the nearness of the stationary distribution of an algorithm to the best-possible cache occupancy distribution. If the statistics of the cache request process are known, the obvious approach to maximizing the hit probability (without knowing the realization of requests) is to simply cache the most popular items as constrained by the cache size, creating a fixed vector of cached content (a point mass). How do we compare the stationary distribution generated by a caching algorithm with this vector? A well known approach to comparing distributions is to determine the Wasserstein distance between them [9]. However, since we are dealing with distributions of permutations of vectors, we need to utilize a notion of a cost that depends on the ordering of elements. Such a notion is provided by a metric called the generalized Kendall's tau [10]. Coupling these two notions together, we define a new metric that we call the "$\tau$-distance", which correctly represents the accuracy of learning the request distribution. Being a distance between cache occupancy distributions, the $\tau$-distance can also be mapped back to hit probability or any other performance measure that depends on learning accuracy. The closest existing work to our approach is [11], in which distributions over permutations of $n$ items over $n$ spots are studied with a cost function resembling Kendall's tau. We also emphasize that the $\tau$-distance formalizes the conceptual remark made earlier on assessing the performance of a caching algorithm by comparing it to an ideal algorithm in terms of the distance between the distributions they engender. Additionally, this is not meant to be calculated for any realistic cache parameters; in fact, in such settings even the stationary hit probability of an algorithm cannot be calculated.

To characterize lag, we need to study the evolution of the Markov chain associated with caching algorithm to understand its rate of convergence to stationarity. The relevant concept here is that of *mixing time*, which is the time (number of requests in our case) required for a Markov process to reach within $\epsilon$ distance (in Total Variation (TV) norm) of its stationary distribution. To the best of our knowledge, except for [12] that studies LRU, there is comparatively little work on analyzing the mixing time of caching algorithms, although there has been some brief commentary on the topic [2], [11]. However, this metric is crucial to understanding algorithm performance, as it characterizes the speed of learning.

Once we have both the $\tau$-distance and the mixing time for a caching algorithm, we can determine algorithm performance as a function of the number of requests. Using the triangle inequality and combining the $\tau$-distance and mixing time (with appropriate normalization), we define a metric that provides an upper bound on the performance at any given time. We call this metric the *learning error,* which effectively combines accuracy and learning lag. Whereas comparisons of learning error may not reflect the true performance differences between two algorithms, nevertheless it correctly determines the trade-off achieved by separately calling out the speed and accuracy of learning achieved.

If we know the time constant of the changes in the requests process by studying the arrival process over time, we can use this knowledge to pick a caching algorithm that has the least learning error, and hence the highest hit probability over a class of caching algorithms. Could we also design an optimal caching algorithm for a dynamic arrival process? While this is a difficult problem to solve optimally, in this paper, we first characterize the performance of an isolated cache through $\tau$-distance and mixing time to study the adaptability of the candidate caching algorithms with simple and meta caches. We use the insights gained in this process to develop an algorithm that operates over the hybrid paradigm. We call the resulting algorithm as Adaptive-LRU (A-LRU). In particular, we focus on a two-level version of A-LRU, and are able to ensure that its learning error at a given time can be made less than either LRU or 2-LRU. We also show that it has the highest hit probability over a class of algorithms that we compare it with using both synthetic requests generated using a Markov-

[1]The provably optimal Bélády's algorithm [8] uses the entire sequence of future requests, and is neither causal nor Markovian.

modulated process, as well as trace-based simulations using traces from YouTube and the IRCache project.

***Related Work:*** Caching algorithms have mostly been analytically studied under the IRM Model. Explicit results for stationary distribution and hit probability for LRU, FIFO, RANDOM, CLIMB [1], [2], [13], [14] have been derived under IRM, however, these results are only useful for small caches due to the computational complexity of solving for the stationary distribution. Several approximations have been proposed to analyze caches of reasonably large sizes [15], and a notable one is the Time-To-Live (TTL) approximation, which was first introduced for LRU under IRM [16], [17]. It has been further generalized to other cache settings [3], [4], [15], [18]. Theoretical support for the accuracy of TTL approximation was presented in [18]. Closest in spirit to our work is [19] that studies TTL caching under non-stationary arrivals, but does not consider mixing times. A rich literature also studies the performance of caching algorithms in terms of hit probability based on real trace simulations, e.g., [4], [5], [7], and we do not attempt to provide an overview here.

***Paper Organization:*** The next section contains some technical preliminaries and caching algorithms. We consider our new notions of learn error, $\tau$-distance and mixing time in Section III. We derive steady state distributions of caching algorithms in Section IV and analyze the mixing time in Section V. We characterize the performance of different algorithms in terms of permutation distance and learning error in Section VII. Finally, we provide trace-based numerical results in Section VIII. We conclude in Section IX. Some additional discussions and proofs are provided in [20].

## II. Preliminaries

***Traffic Model:*** We assume that there is a library of $n$ items. The request processes for distinct content are described by independent Poisson processes with arrival rate $\lambda_i$ for content $i = 1, \cdots, n$. Without loss of generality (w.l.o.g.), we assume that the aggregate arrival rate is 1, then the popularity of content $i$ satisfies $p_i = \lambda_i$. W.l.o.g., we assume that the reference items are numbered so that the probabilities are in a non-increasing order, i.e., $p_1 \geq p_2 \geq \cdots \geq p_n$.

***Popularity Law:*** Whereas our analytical results are not for any specific popularity law, for our numerical investigations we will use a Zipf distribution as this family has been frequently observed in real traffic measurements, and is widely used in performance evaluation studies in the literature [6]. For a Zipf distribution, the probability to request the $i$-th most popular item is $p_i = A/i^\alpha$, where $\alpha$ is the Zipf parameter depending on the application considered, and $A$ is the normalization constant so that $\sum_{i=1}^{n} p_i = 1$ if there are $n$ unique items in total.

***Dimensions of Caching:*** A cache is fundamentally a block of memory that can be used to store data items that are frequently requested. Over the years, different paradigms have evolved on how best to utilize the available memory. Most conventional caching algorithms, such as LRU, RANDOM and FIFO, have been designed and analyzed on a simple (isolated) cache, as shown in Figure 1 (a). New caching algorithms have been proposed that have been shown to have better performance

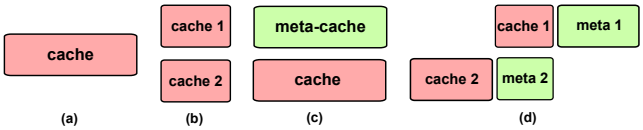


Fig. 1. Dimensions of caching.

than the classical paradigm, often through numerical studies. The different dimensions that have been explored are two fold. On the one hand, the memory block can be divided into two or more levels, with a hierarchical algorithm attempting to ensure that more popular content items get cached in the higher levels. For example, a simple 2-level cache (also called a linear cache network) is shown in Figure 1 (b), and it has been empirically observed that under an appropriate caching algorithm, it could display a higher hit probability than that of a simple cache of the same size. On the other hand, a meta-cache that simply stores content identities can be used to better learn popularity without wasting memory to cache the actual data item. The idea is illustrated in Figure 1 (c) with one level of meta caching. A concept that we will explore further in this paper is to mix both ideas, shown in Figure 1 (d). However, in all cases, it is not clear how the different dimensions enhance the hit probability, and how they impact convergence to stationarity.

***Caching Algorithms:*** There exist a large number of caching algorithms, with the difference being in their choice of insertion or eviction rules. In this paper, we consider the following three dimensions of caching algorithms as illustrated in Figure 1. First, we consider the conventional *signle-level caching algorithms* to manage a single cache shown in Figure 1 (a), including LRU [1], [12], [13], FIFO [1], [13], RANDOM [1] and CLIMB [1], [2]. Second, we consider *meta-cache caching algorithms* to manage caches as shown in Figure 1 (c), including $k$-LRU [3], [4]. Third, we consider *multi-level caching algorithms* to manage caches as shown in Figure 1 (b), including LRU($\boldsymbol{m}$) [3] and ARC. Due to space constraints, a detailed explanation on the operations of these algorithms are available in [20].

Based on the learning error analysis of these algorithms in the following sections, we also propose a novel hybrid algorithm, Adaptive-LRU (A-LRU). Detailed operation of A-LRU is presented in Section VI.

## III. Learning Error

We desire a notion of error that accounts for the tradeoff between accuracy and speed of learning. The accuracy can be characterized by the nearness of the stationary distribution of an algorithm to the best-possible cache occupancy distribution. The speed of learning can be characterized by the mixing time of the caching algorithm. Clearly, a figure of merit of this kind is the distance

$$\begin{aligned}\delta_A(t) &= \sup_{\boldsymbol{x}\in\mathcal{S}} |\pi_A(\boldsymbol{x},t) - \mathbf{c}^*|_\tau \\ &\leq |\pi_A^* - \mathbf{c}^*|_\tau + \sup_{\boldsymbol{x}\in\mathcal{S}} |\pi_A(\boldsymbol{x},t) - \pi_A^*|_\tau, \end{aligned} \tag{1}$$

measured in some metic $\tau$ after $t$ requests, where $\pi_A^*$ is the stationary distribution of algorithm $A$, $\mathbf{c}^*$ is the best-possible

occupancy vector and $\pi_A(\boldsymbol{x}, t)$ is the row corresponding to state $\boldsymbol{x} \in \mathcal{S}$ of the $t$-step transition matrix of algorithm $A$; these will be described in detail later. The first term above is the error in the (eventual) learning of the algorithm (accuracy), and the second term is the error due to time lag of learning (efficiency). We could then argue that if the time-constant of the change in the request distribution is $t$, then the caching algorithm $A$ that has a low value of the RHS would have attained some fraction of optimality by that time. We pause to reemphasize that the utility of the learning error is in revealing the trade-off achieved by an algorithm by disentangling the speed and accuracy of learning. In the rest of this section, we will posit an appropriate metric $\tau$ and characterize both the error terms.

### A. Permutation Distance

We seek a refinement that would allow us to determine "how close" the stationary performance of an algorithm is to the best-possible. If we have full knowledge of the popularity distribution at any time, we could simply cache the most popular items in the available cache spots, placing the most popular element in first cache spot, and then proceeding onwards until the $m$-th spot. This approach would maximize the hit probability, as well as any other metric that yields better performance when more popular items are cached. We denote this ideal occupancy vector as $\mathbf{c}^*$; this ideal distribution is then a point-mass at $\mathbf{c}^*$. As a means to determine closeness of the stationary performance of an algorithm to the ideal scheme, we start by discussing an appropriate distance between the ideal occupancy vector and any possible cache occupancy state.

*1)* **Generalized Kendall's Tau Distance:** Let $[n] = \{1, \cdots, n\}$ be a library of items and $[n]_m$ be the set of $m$ items chosen from $[n]$. Let $S_n^m$ be the set of permutations on $[n]_m$. Consider a permutation $\sigma \in S_n^m$, we interpret $\sigma(i)$ as the position of item $i$ in $\sigma$, and we say that $i$ is ahead of $j$ in $\sigma$ if $\sigma(i) < \sigma(j)$. W.l.o.g, we take $\sigma(i) = 0$ for $i \in [n]/[n]_m$, i.e., all items absent in the cache have position 0.

The classical Kendall's tau distance [21] [2] is given by

$$K(\sigma_1, \sigma_2) = \sum_{(i,j):\sigma_1(i)>\sigma_1(j)} 1_{\{\sigma_2(i)<\sigma_2(j)\}}, \tag{2}$$

where $1_{\mathcal{A}}$ is the indicator function and $1_{\mathcal{A}} = 1$ if the condition $\mathcal{A}$ holds true, otherwise $1_{\mathcal{A}} = 0$.

However, this conventional definition does not take into account the item relevance and positional information, which are crucial to evaluating the distance metric in a permutation. Since we wish to compare with $\mathbf{c}^*$, in which the most popular items are placed in lower positions, the errors in lower positions in the permutation need to be penalized more heavily than those in higher positions. Many alternative distance measures have been proposed to address these shortcomings. In the following, we consider the generalized Kendall's tau distance proposed in [10] that captures the importance of each item as well as the positions of the errors.

[2]We consider $p = 0$ for the definition given in [21], which is an "optimistic approach" that corresponds to the intuition that we assign a nonzero penalty to the pair $\{i, j\}$ only if we have enough information to know that $i$ and $j$ are in the opposite order in the two permutations $\sigma_1$ and $\sigma_2$.

Let $w_i > 0$ be the *element weight* for $i \in [n]$. For simplicity, we assume that $w_i \in \mathbb{Z}^+$; all the subsequent results hold for non-integral weights as well. In addition to the element weight, as discussed earlier, we wish to penalize inversions early in the permutation more than inversions later in the permutations. In order to achieve this, we define *position weights* to differentiate the importances of positions in the permutation. We first consider the cost of swapping between two adjacent positions. Let $\zeta_j \geq 0$ be the cost of swapping an item at position $j-1$ with an item at position $j$, and let $q_0 = 0$, $q_1 = 1$ and $q_j = q_{j-1} + \zeta_j$ for $1 < j \leq m$. Define $\bar{q}^i_{\sigma_1,\sigma_2} = \frac{q_{\sigma_1(i)} - q_{\sigma_2(i)}}{\sigma_1(i) - \sigma_2(i)}$ to be the average cost that item $i$ encountered in moving from position $\sigma_1(i)$ to position $\sigma_2(i)$, with the understanding that $\bar{q}^i_{\sigma_1,\sigma_2} = 1$ if $\sigma_1(i) = \sigma_2(i)$. We set the value of $\bar{q}^j_{\sigma_1,\sigma_2}$ similarly. We then define the generalized Kendall's tau distance as follows:

$$K_{w,\zeta}(\sigma_1, \sigma_2) = \sum_{\sigma_1(i)<\sigma_1(j)} w_i w_j \bar{q}^i_{\sigma_1,\sigma_2} \bar{q}^j_{\sigma_1,\sigma_2} 1_{\{\sigma_2(i)>\sigma_2(j)\}}. \tag{3}$$

**Remark** *1:* Note that if we are interested only in cache hits and misses (eg., if there is no search cost within the cache), the ordering is irrelevant and only content presence or absence matters. The Kendall's tau still applies with weights being just 0 and 1, to indicate presence and absence, respectively. Further, we now need only consider distances between equivalent classes of cache states, where two cache states with identical content are equivalent.

*2)* **Wasserstein Distance:** While the generalized Kendall's tau distance is a way of comparing two permutations, the algorithms that we are interested in do not converge to a single permutation, but yield distributions over permutations with more elements in their support. Hence, we should compare the stationary distribution $\pi^*_A$ of an algorithm $A$, with $\mathbf{c}^*$ using a distance function that accounts for the ordering of content in each state vector. Given a metric on permutations, the Wasserstein distance [9] is a general way of comparing distributions on permutations.

Let $(\mathcal{S}, d)$ be a Polish space, and consider any two probability measures $\mu$ and $\nu$ on $\mathcal{S}$, then the Wasserstein distance[3] between $\mu$ and $\nu$ is defined as

$$W(\mu, \nu) = \inf_{P_{X,Y}(\cdot,\cdot)} \Big\{ \mathbb{E}[d(X, Y)], \quad P_X(\cdot) = \mu, P_Y(\cdot) = \nu \Big\}, \tag{4}$$

which is the minimal cost between $\mu$ and $\nu$ induced by the cost function $d$.

*3)* **$\tau$-distance:** We are now ready to define the specific form of Wasserstein distance between distributions on permutations that is appropriate to our problem. We define the $\tau$-distance as the Wasserstein distance taking the generalized Kendall's distance in (3) as the cost function in (4).

Since the ideal occupancy vector $\mathbf{c}^*$ is unique and fixed, the infimum in (4) over all the couplings is trivially given by the average distance, i.e.,

$$|\pi^*_A - \mathbf{c}^*|_\tau := W(\pi^*_A, \delta_{\mathbf{c}^*}) = \sum_{\boldsymbol{x}} K_{w,\zeta}(\boldsymbol{x}_{k(A)}, \mathbf{c}^*) \pi^*_A(\boldsymbol{x}), \tag{5}$$

[3]W.l.o.g., we are interested in the $L^1$-Wasserstein distance, which is also commonly called the Kantorovich-Rubinstein distance [9]. For convenience, we express Wasserstein distance by means of couplings (joint distributions) between random variables with given marginals.

where $K_{w,\zeta}(\cdot,\cdot)$ is the generalized Kendall's tau distance defined in (3), and $\boldsymbol{x}_{k(A)}$ is the component of state under algorithm $A$ corresponding to real items. Thus, $\boldsymbol{x}_{k(A)} = \boldsymbol{x}$ for all the conventional algorithms considered in Section IV-A, and the multi-level caching algorithms in Section IV-C, but $\boldsymbol{x}_{k(A)} = \boldsymbol{x}_k$ for $k$-LRU as discussed in Section IV-B.

### B. Computation of Mixing Time

While comparing the $\tau$-distance of the stationary distance of an algorithm to the ideal occupancy vector does provide insight into the algorithm's accuracy of learning, it says nothing about how quickly the algorithm can respond to changes in the request distribution—a critical shortcoming in developing and characterizing the ideal algorithm for a given setting. How does one come up with a metric that accounts for both accuracy and speed of learning? It seems clear that one ought to study the evolution of the caching process over time to understand how quickly the distribution evolves. Within the Markovian setting of our algorithms, the metric of relevance in this context is called *mixing time*, which is the time required for a Markov process to reach within $\epsilon$ distance (in Total Variation (TV) norm) of the eventual stationary distribution. If we denote the row corresponding to state $\boldsymbol{x} \in \mathcal{S}$ of the $t$-step transition matrix of algorithm $A$ by $\pi_A(\boldsymbol{x}, t)$, then the mixing time is the smallest value of $t$ such that

$$\sup_{\boldsymbol{x}\in\mathcal{S}} |\pi_A(\boldsymbol{x}, t) - \pi_A^*|_{TV} \leq \epsilon, \tag{6}$$

for a given $\epsilon > 0$ [22]; denote it as $t_{\text{mix}}(\epsilon)$. As mentioned earlier, we will also think of $\pi_A(\boldsymbol{x}, t)$ and $\pi_A^*$ as distributions on permutations of the $n$ objects.

Mixing times can be estimated using many different procedures. Here, we use *conductance* to build bounds on the mixing time through *Cheeger's inequality*. We first introduce these techniques and then characterize the mixing time of various caching algorithms in Section V. These bounds will allow us to compare all the algorithms on an equal footing, and also to determine the first-order dependence on algorithm parameters.

*1)* **Spectral Gap and Mixing Time:** Let $\gamma^*$ be the *spectral gap* of any Markov chain with transition matrix $P$, and denote $\pi_P^*$ as its corresponding stationary distribution. Then defining $\pi_{\min} := \min_{\boldsymbol{x}\in\mathcal{S}} \pi_P^*(\boldsymbol{x})$, an upper bound on the mixing time in terms of spectral gap and the stationary distribution of the chain is given as follows [22], [23]:

$$t_{\text{mix}}(\epsilon) < 1 + \frac{1}{\gamma^*} \ln\left(\frac{1}{\epsilon\pi_{\min}}\right). \tag{7}$$

*2)* **Conductance and Mixing Time:** For a pair of states $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{S}$, we define the transition rate $Q(\boldsymbol{x}, \boldsymbol{y}) = \pi(\boldsymbol{x})P(\boldsymbol{x}, \boldsymbol{y})$. Let $Q(S_1, S_2) = \sum_{\boldsymbol{x}\in S_1} \sum_{\boldsymbol{y}\in S_2} Q(\boldsymbol{x}, \boldsymbol{y})$, for two sets $S_1, S_2 \in \mathcal{S}$. Now, for a given subset $S \in \mathcal{S}$, we define its conductance as $\Phi(S) = \frac{Q(S,\bar{S})}{\min(\pi(S),\pi(\bar{S}))}$, where $\pi(S) = \sum_{i\in S} \pi_i$. and $(S, \bar{S})$ is a cut of the graph. Note that $Q(S, \bar{S})$ represents the "ergodic flow" from $S$ to $\bar{S}$. Finally, the *conductance of the chain P* is the conductance of the "worst" set, i.e.,

$$\Phi = \min_{S\subset\mathcal{S}, \pi(S)\leq\frac{1}{2}} \Phi(S). \tag{8}$$

The relationship between the conductance and the mixing time of a Markov chain (the spectral gap) is given by the *Cheeger inequality* [23], [24]:

$$\frac{\Phi^2}{2} \leq \gamma^* \leq 2\Phi. \tag{9}$$

Combining (9) with the previous result in (7), we can relate the conductance directly to the mixing time as follows:

$$t_{\text{mix}}(\epsilon) \leq \frac{2}{\Phi^2}\left(\ln\frac{1}{\pi_{\min}} + \ln\frac{1}{\epsilon}\right). \tag{10}$$

While the spectral gap and conductance of a Markov chain can provide tight bounds on the mixing time of the chain, these values are often difficult to calculate accurately. If we are interested in proving rapid mixing[4], we can provide a lower bound on the conductance. Canonical paths and congestion are used in this regard as they are easier to compute, and bound the conductance from below. For any pair $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{S}$, we can define a canonical path $\psi_{\boldsymbol{xy}} = (\boldsymbol{x} = \boldsymbol{x}_0, \cdots, \boldsymbol{x}_l = \boldsymbol{y})$ running from $\boldsymbol{x}$ to $\boldsymbol{y}$ through adjacent states in the state space $\mathcal{S}$ of the Markov chain. Let $\Psi = \{\psi_{\boldsymbol{xy}}\}$ be the family of canonical paths between all pairs of states. The *congestion* of the Markov chain is then defined as

$$\rho = \rho(\Psi) = \max_{(\boldsymbol{u},\boldsymbol{v})} \left\{ \frac{1}{\pi(\boldsymbol{u})P_{\boldsymbol{uv}}} \sum_{\substack{\boldsymbol{x},\boldsymbol{y}\in\mathcal{S} \\ \exists\, \psi_{\boldsymbol{xy}} \text{ using } (\boldsymbol{u},\boldsymbol{v})}} \pi(\boldsymbol{x})\pi(\boldsymbol{y}) \right\}, \tag{11}$$

where the maximum runs over all pairs of states in the state space, and the number of canonical path is of the order of $\Gamma^2$, with $\Gamma$ being the number of possible states. Therefore, high congestion corresponds to a lower conductance, and as demonstrated in [26]

$$\Phi \geq \frac{1}{2\rho}. \tag{12}$$

Note that the above result applies to all possible choices of canonical paths, for example, no requirement is made that the shortest path between two states has be chosen.

*3)* **Mixing Time Bound:** Based on the above analysis, we are ready to present an explicit general bound on mixing time as follows.

**Lemma 1:** Suppose the Markov chain associated with caching algorithm $A$ has a reversible transition matrix $P$, and denote the corresponding stationary distribution as $\pi_P^*$. Based on the analysis of the relations between conductance and mixing time in (8) - (12), we can directly characterize the mixing time of a caching algorithm[5]:

$$t_{\text{mix}}(\epsilon) \leq \frac{8(\pi_{P,\max}^*)^4 \cdot \Gamma^4}{(\pi_{P,\min}^* P_{\min})^2}\left(\ln\frac{1}{\pi_{P,\min}^*} + \ln\frac{1}{\epsilon}\right), \tag{13}$$

where $\pi_{P,\max}^* = \max_{\boldsymbol{x}\in\mathcal{S}} \pi_P^*(\boldsymbol{x})$, $\pi_{P,\min}^* = \min_{\boldsymbol{x}\in\mathcal{S}} \pi_P^*(\boldsymbol{x})$, and $P_{\min}$ is the minimal transition probability from one state to another state.

The proof of this lemma is straightforward given (8) - (12), since we only need to find an upper bound on the congestion

[4] A family of ergodic, reversible Markov chain with state space of size $|\mathcal{S}|$ and conductance $\Phi_{|\mathcal{S}|}$ is *rapidly mixing* if and only if $\Phi_{|\mathcal{S}|} \geq \frac{1}{\mathcal{P}(|\mathcal{S}|)}$ for some polynomial $\mathcal{P}$ [25]. This result is commonly used to show rapid mixing of Markov chains.

[5] We omit the superscript $A$ for brevity.

$\rho$ through characterizing $\pi^*_{P,\max}$ and $\pi^*_{P,\min}$ given the steady state distribution. As the analysis for different algorithms varies considerably in terms of the characterization of the spectral gap, we will use the above result for an equal footing comparison.

*4)* **Non-Reversibility and Mixing Time:** Many results on mixing times have been developed in the context of a reversible Markov chain. However, many of the popular caching algorithms such as the LRU family generate non-reversible Markov chains. From [23] it follows that several of results that apply to reversible Markov chains hold even without reversibility but with the modifications described below.

For any non-reversible Markov chain with transition matrix $P$, first determine $P^*$, which is the time-reversed transition matrix:

$$\pi^*_P(\boldsymbol{x})P^*(\boldsymbol{x},\boldsymbol{y}) = \pi^*_P(\boldsymbol{y})P(\boldsymbol{y},\boldsymbol{x}), \tag{14}$$

where $\boldsymbol{x},\boldsymbol{y} \in \mathcal{S}$. Then we can construct a reversible Markov chain with transition matrix $\frac{P+P^*}{2}$, for which the following result holds.

**Lemma 2:** Let $P$ be the transition matrix of a non-reversible Markov chain, and $P^*$ be the time-reversal. Denote $\pi^*_P$ and $\gamma^*_P$ be the corresponding stationary distribution and spectral gap, then we have

$$\pi^*_P(\boldsymbol{x}) = \pi^*_{P^*}(\boldsymbol{x}) = \pi^*_{\frac{P+P^*}{2}}(\boldsymbol{x}), \forall \boldsymbol{x} \in \mathcal{S}, \text{and} \gamma^*_P = \gamma^*_{P^*} = \gamma^*_{\frac{P+P^*}{2}}. \tag{15}$$

This result was originally presented in [27], and the proof in our context is presented in [20] for completeness. Then from (7), we can equivalently use the reversible Markov chain $\frac{P+P^*}{2}$ to bound the mixing time of the non-reversible Markov chain $P$ through applying existing results on reversible Markov chains. This procedure will be utilized in the following subsections.

Given the results in Lemma 1 and Lemma 2, and the fact that $\frac{(P+P^*)}{2}(\boldsymbol{x},\boldsymbol{y}) \geq P(\boldsymbol{x},\boldsymbol{y})/2$ we immediately have the following result:

**Corollary 1:** Suppose the Markov chain associated with caching algorithm $A$ has a non-reversible transition matrix $P$. Then we have

$$t_{\text{mix}}(\epsilon) \leq \frac{8(\pi^*_{P,\max})^4 \cdot \Gamma^4}{(\pi^*_{P,\min}\frac{P_{\min}}{2})^2}\left(\ln\frac{1}{\pi^*_{P,\min}} + \ln\frac{1}{\epsilon}\right), \tag{16}$$

where $\pi^*_{P,\max} = \max_{\boldsymbol{x}\in\mathcal{S}} \pi^*_P(\boldsymbol{x})$, $\pi^*_{P,\min} = \min_{\boldsymbol{x}\in\mathcal{S}} \pi^*_P(\boldsymbol{x})$, $\Gamma$ is the number of possible states and $P_{\min}$ is the minimal transition probability from one state to another state.

**Remark** *2:* From (13) and (16), it is clear that for both reversible and irreversible Markov chains, we need to characterize $\pi^*_{P,\max}$, $\pi^*_{P,\min}$ and $P_{\min}$ in order to obtain mixing time bounds. Thus, identification of the steady state distribution $\pi^*_P$ (discussed in Section IV) in a way that allows us to determine these bounds is crucial in obtaining mixing time bounds.

### C. Learning Error

Since the space of all permutations on $n$ objects is finite, it has a finite diameter in terms of the generalized Kendall's tau distance. Let this diameter be denoted as $\kappa_\tau$. Then (1) can be bounded using the triangle inequality as

$$\delta_A(t) \leq |\pi^*_A - \boldsymbol{c}^*|_\tau + \kappa_\tau \sup_{\boldsymbol{x}\in\mathcal{S}} |\pi_A(\boldsymbol{x},t) - \pi^*_A|_{TV} \triangleq e_A(t). \tag{17}$$

We refer to $e_A(t)$ as the *learning error* of algorithm $A$ at time $t$, which is now only a function of the accuracy and mixing time of the algorithm. In order to compute the learning error $e_A(t)$ in (17), we need the stationary distribution of algorithm $A$. In the following sections, we first characterize the stationary distributions of different caching algorithms in Section IV, then analyze their mixing time in Section V, and finally evaluate their performance in Section VII.

## IV. Steady State Distribution

We consider the question of determining the stationary distribution of the contents of a cache based on the caching algorithm used. Each (known) caching algorithm $A$ under any Markov modulated request arrival process (IRM too) results in a Markov process over the occupancy states of the cache.

### A. Classical Results on Single-Level Caching Algorithms

Suppose there are a total of $n$ content items in a library $\mathcal{L}$, and the cache size is $m < n$. Then each cache state $\boldsymbol{x}$ is a vector of length $m$ that indicates the content in each cache spot. We denote $x_j \in \mathcal{L}$ as the identity of the item at position $j$ in the cache, i.e., $\boldsymbol{x} = (x_1, \cdots, x_m)$. As mentioned earlier, we call the state space of all such vectors $\mathcal{S}$. Our notation for state is consistent with respect to the algorithm descriptions in Section II and represents motion from "left-to-right" under our candidate algorithms. For example, under LRU $x_j$ has been requested more recently than $x_k$ if $j < k$.

One can potentially determine the stationary distribution of the Markov process generated by a particular algorithm $A$, denoted $\pi^*_A$. This procedure has been carried out for several classical caching algorithms in the literature [1], but the results are not available in the desired form (viewed in terms of permutations) so we present them for the Markov chains generated by FIFO, RANDOM, CLIMB and LRU.

**Theorem 1:** Under the IRM, the steady state probabilities $\pi^*_{\text{FIFO}}(\boldsymbol{x})$, $\pi^*_{\text{RANDOM}}(\boldsymbol{x})$, $\pi^*_{\text{CLIMB}}(\boldsymbol{x})$ and $\pi^*_{\text{LRU}}(\boldsymbol{x})$, with $\boldsymbol{x} \in \mathcal{S}$ are as follows:

$$\begin{aligned}
\pi^*_{\text{FIFO}}(\boldsymbol{x}) &= \frac{\Pi_{i=1}^m p_{x_i}}{\sum_{\boldsymbol{y}\in\mathcal{S}}\Pi_{i=1}^m p_{y_i}},\\
\pi^*_{\text{RANDOM}}(\boldsymbol{x}) &= \frac{\Pi_{i=1}^m p_{x_i}}{\sum_{\boldsymbol{y}\in\mathcal{S}'}\Pi_{i=1}^m p_{y_i}},\\
\pi^*_{\text{CLIMB}}(\boldsymbol{x}) &= \frac{\Pi_{i=1}^m p_{x_i}^{m-i+1}}{\sum_{\boldsymbol{y}\in\mathcal{S}}\Pi_{i=1}^m p_{y_i}^{m-i+1}},\\
\pi^*_{\text{LRU}}(\boldsymbol{x}) &= \frac{\Pi_{i=1}^m p_{x_i}}{(1-p_{x_1})(1-p_{x_1}-p_{x_2})\cdots(1-p_{x_1}-\cdots-p_{x_{m-1}})},
\end{aligned} \tag{18}$$

where $\mathcal{S}'$ denotes the set of all combinations of elements of $\{1,\cdots,n\}$ taken $m$ at a time. Note that elements of $\mathcal{S}'$ are subsets of $\{1,\cdots,n\}$, while elements of $\mathcal{S}$ are ordered subset of $\{1,\cdots,n\}$, satisfying $\sum_{\boldsymbol{y}\in\mathcal{S}}\Pi_{j=1}^m p_{y_j} = m!\sum_{\boldsymbol{y}\in\mathcal{S}'}\Pi_{j=1}^m p_{y_j}$. These are the well-known steady state probabilities for FIFO, RANDOM, CLIMB and LRU [1], [2], [13], [14], [28]. The

result for LRU is obtained by a probabilistic argument [29]. We detail this in [20] for completeness, and since we will use the method for other related algorithms.

### B. Meta-cache Caching Algorithms

A closed form result of the stationary distribution of the Markov chain generated by $k$-LRU (see Figure 1 (c) for typical configuration) is not available currently. Motivated by the approach to determining the stationary distribution of LRU [29], we use a probabilistic argument to obtain the general form of the stationary distribution of $k$-LRU. While the expression that we obtain is complex from the perspective of numerical computation, it will provide us with the necessary structure to obtain mixing time bounds in Section III-B.

Consider a cache system with $k-1$ levels of meta-cache, followed by a level of real cache. We denote $x_{(i,j)} \in \mathcal{L}$ as the identity of the item at position $j$ in cache $i$. Here, $i \in \{1, \cdots, k-1\}$ refer to meta caches, while $i = k$ refers to the real cache. We denote the state of level $j$ as $\boldsymbol{x}_j = (x_{(j,1)}, \cdots, x_{(j,m)})$, and the state of the whole cache as $\boldsymbol{x} = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k)$. Note that only $\boldsymbol{x}_k$ caches the real items, while all other levels cache only meta-data. Finally, let $\mathcal{X} \subset \mathcal{L}$ be the set of items present in $\boldsymbol{x}$.

**Definition 1: Sample Path:** A *sample path* $\gamma(\boldsymbol{x})$ for state $\boldsymbol{x}$ is a sequence of requests that leads to the state $\boldsymbol{x}$ under the $k$-LRU algorithm starting from any fixed initial state (such as the empty cache).

For any item $y \in \mathcal{X}$, let $h(y) = \max\{i : x_{(i,j)} = y\}$, i.e., $h(y)$ is highest cache level at which item $y$ is present. Each sample path $\gamma(\boldsymbol{x})$ must contain a set of the final $h(y)$ requests for each $y \in \mathcal{X}$. Call the union of all these requests as $\hat{\mathcal{X}}$. Hence, $\hat{\mathcal{X}}$ will contain exactly $h(y)$ copies of each item in $\mathcal{X}$. Note that $|\hat{\mathcal{X}}|$ is at most $\sum_{j=1}^{k} jm$, which occurs when all the items in $\boldsymbol{x}$ are distinct from each other.

Let $\hat{\boldsymbol{x}}$ represent an arrangement of the items in $\hat{\mathcal{X}}$. Again, note that this arrangement has exactly $h(y)$ copies of each item in $\mathcal{X}$. Then an arbitrary request sequence $\hat{\gamma}(\hat{\boldsymbol{x}})$ following $\hat{\boldsymbol{x}}$ interspersed with any other items drawn from $\mathcal{L}$, which does not violate the condition that $\hat{\boldsymbol{x}}$ is the ordering of the final $h(y)$ requests for each $y \in \mathcal{X}$, is a candidate sample path leading to $\boldsymbol{x}$. However, $\hat{\gamma}(\hat{\boldsymbol{x}})$ might not be consistent with $k$-LRU, i.e., not all arrangements of $\hat{\mathcal{X}}$ can be used to generate sample paths using $k$-LRU. Hence, we denote a sample path consistent with $k-$LRU by $\gamma(\hat{\boldsymbol{x}})$, which gives rise to the following definition.

**Definition 2: Class of Sample Paths:** We define a valid class of sample paths $\Lambda(\hat{\boldsymbol{x}})$ as a set of sample paths $\gamma(\hat{\boldsymbol{x}})$ each of which follows arrangement $\hat{\boldsymbol{x}}$, and is consistent with the operation of $k$-LRU. Let $\Upsilon(\boldsymbol{x})$ be the set of the classes of sample paths associated with state $\boldsymbol{x}$, i.e., $\Upsilon(\boldsymbol{x}) = \{\Lambda(\hat{\boldsymbol{x}})\}$.

**Definition 3: Subclass of Sample Paths:** We define a valid subclass of sample paths $\tilde{\Lambda}(\hat{\boldsymbol{x}})$ as a set of sample paths $\gamma(\hat{\boldsymbol{x}}) \in \Lambda(\hat{\boldsymbol{x}})$ and the set of items that can be requested between any two items on the arrangement $\hat{\boldsymbol{x}}$ are identical. Let $\tilde{\Upsilon}_{\Lambda(\hat{\boldsymbol{x}})}(\boldsymbol{x})$ be the set of the subclasses of sample paths associated with class $\Lambda(\hat{\boldsymbol{x}})$.

Thus, each valid sample path $\gamma(\boldsymbol{x})$ leading to state $\boldsymbol{x}$ under $k$-LRU belongs to a valid subclass $\tilde{\Lambda}(\hat{\boldsymbol{x}}) \in \Lambda(\hat{\boldsymbol{x}})$.

We present one illustrative example to explain the above definitions with a more simple one ($k = 1$) detailed in [20]. We consider $n = 5, m = 2$, and denote the items as $1, 2, 3, 4, 5$, with popularities $p_1 > p_2 > p_3 > p_4 > p_5$ and $\sum_{i=1}^{5} p_i = 1$.

**Example 1:** Consider $k = 2$, i.e., the 2-LRU algorithm. Consider state $\boldsymbol{x} = ((34), (12))$, i.e., (12) are in the second level and (34) are in the first level. So there are totally $2*1+2*2=6$ items that need to be fixed to obtain a sample path, i.e., items $\hat{\mathcal{X}} = \{2, 2, 1, 1, 3, 4\}$. Based on the 2-LRU policy, there are totally 9 valid arrangements (and hence classes) over these 6 items, i.e., $|\Upsilon(\boldsymbol{x})| = 9$. It can be verified that these valid arrangements are: $2 \to 2 \to 1 \to 1 \to 4 \to 3$; $2 \to 2 \to 1 \to 4 \to 1 \to 3$; $2 \to 2 \to 4 \to 1 \to 1 \to 3$; $2 \to 4 \to 2 \to 1 \to 1 \to 3$; $4 \to 2 \to 2 \to 1 \to 1 \to 3$; $2 \to 1 \to 2 \to 1 \to 4 \to 3$; $2 \to 1 \to 2 \to 4 \to 1 \to 3$; $1 \to 2 \to 2 \to 1 \to 4 \to 3$; $1 \to 2 \to 2 \to 4 \to 1 \to 3$. Each arrangement $\hat{\boldsymbol{x}}$ above can be used to generate infinite sample paths that belong to the corresponding class $\Lambda(\hat{\boldsymbol{x}})$. In each class $\Lambda(\hat{\boldsymbol{x}})$, the items can be requested between any two fixed items on arrangement $\hat{\boldsymbol{x}}$ define the subclass $\tilde{\Lambda}(\hat{\boldsymbol{x}})$.

Consider a valid sample path $\gamma(\boldsymbol{x}) \in \tilde{\Lambda}(\hat{\boldsymbol{x}}) \in \Lambda(\hat{\boldsymbol{x}})$. It can be split into the requests following $\hat{\boldsymbol{x}}$ and all other requests. Define $\Xi(\gamma(\boldsymbol{x}))$ as the probability of requesting all these other items. Hence, the probability of any sample path is the product of $\Xi(\gamma(\boldsymbol{x}))$ and the probability of requesting the items in $\hat{\boldsymbol{x}}$. Now, consider an item $x_{(i,j)} \in \boldsymbol{x}$. According to our notation $h(x_{(i,j)})$ is the highest cache level in which the item appears. Define a function

$$\mathcal{J}_{x_{(i,j)}} = \begin{cases} p_{x_{(i,j)}}, & \text{if } i = h(x_{(i,j)}) \\ 1, & \text{otherwise.} \end{cases} \tag{19}$$

We then have a characterization of the steady state distribution of $k$-LRU.

**Theorem 2:** Under the IRM, the steady state probability distribution of $k$-LRU satisfies the following form

$$\pi^*_{k\text{-LRU}}(\boldsymbol{x}) = \prod_{i=1}^{k}\left(\prod_{j=1}^{m}\mathcal{J}_{x_{(i,j)}}\right)^{i} \sum_{\Lambda(\hat{\boldsymbol{x}})\in\Upsilon(\boldsymbol{x})} \sum_{\tilde{\Lambda}(\hat{\boldsymbol{x}})\in\Lambda(\boldsymbol{x})} \sum_{\gamma(\boldsymbol{x})\in\tilde{\Lambda}(\hat{\boldsymbol{x}})} \Xi(\gamma(\boldsymbol{x})), \tag{20}$$

*Proof:* We consider a particular cache state $\boldsymbol{x} = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k)$, where $\boldsymbol{x}_j = (x_{(j,1)}, \cdots, x_{(j,m)})$, and attempt to reconstruct the past history by looking backwards in time. Note that only the $k^{\text{th}}$ level caches real data items (contents are $\boldsymbol{x}_k$), while all other levels cache meta-data.

However, it is not as easy as in the case of LRU to find a particular path to reconstruct the past history. Essentially, we need to deteremine all the possible paths that result in the state $\boldsymbol{x}$ under $k$-LRU. Recall that according to our notation $h(x_{(i,j)})$ is the highest cache level in which the item appears, and that

$$\mathcal{J}_{x_{(i,j)}} = \begin{cases} p_{x_{(i,j)}}, & \text{if } i = h(x_{(i,j)}) \\ 1, & \text{otherwise.} \end{cases} \tag{21}$$

As discussed in Section IV-B, any sample path $\gamma(\hat{\boldsymbol{x}})$ leading to $\boldsymbol{x}$ must contain an arrangement $\hat{\boldsymbol{x}}$ over the set $\hat{\mathcal{X}}$ in which $h(x_{(i,j)})$ copies of $x_{(i,j)}$ appear (these are the final requests for that item in that sample path). This arrangement is common to all paths in subclass $\tilde{\Lambda}(\hat{\boldsymbol{x}})$ and then class $\Lambda(\hat{\boldsymbol{x}})$. Also, every

class must contain some arrangements of the elements in $\hat{\mathcal{X}}$. The probability of occurrence of any of these arrangements, which is common to every sample path leading to $\boldsymbol{x}$ is $\prod_{i=1}^{k}\left(\prod_{j=1}^{m}\mathcal{J}_{x_{(i,j)}}\right)^{i}$.

Next, consider a particular class of sample paths $\Lambda(\hat{\boldsymbol{x}})$. Denote the sequence of items fixed by the arrangement $\hat{\boldsymbol{x}}$ by $\xi_1,\cdots,\xi_\delta\cdots\xi_{\hat{\mathcal{X}}}$, where $\xi_\delta$ stands for the identity of the $\delta$-th item in this arrangement. A sample path can be constructed by adding other items from $\mathcal{L}$ between these fixed items in such a way that the end result is $\hat{\boldsymbol{x}}$ under $k$-LRU (i.e., it is valid). Let the probability of requesting these other items for a particular sample path be $\Xi(\gamma(\boldsymbol{x}))$. Then the probability of a valid sample path $\gamma(\boldsymbol{x})$ is simply $\prod_{i=1}^{k}\left(\prod_{j=1}^{m}\mathcal{J}_{x_{(i,j)}}\right)^{i}\Xi(\gamma(\boldsymbol{x}))$.

Thus, the sum of the probabilities of all the possible sample paths in the subclass $\tilde{\Lambda}(\hat{\boldsymbol{x}})$, is

$$\sum_{\gamma(\boldsymbol{x})\in\tilde{\Lambda}(\hat{\boldsymbol{x}})}\prod_{i=1}^{k}\left(\prod_{j=1}^{m}\mathcal{J}_{x_{(i,j)}}\right)^{i}\Xi(\gamma(\boldsymbol{x})). \tag{22}$$

Following this argument, we consider all the possible subclasses of sample path $\tilde{\Lambda}(\hat{\boldsymbol{x}})\in\Lambda(\hat{\boldsymbol{x}})$, and all the possible classes of sample path $\Lambda(\hat{\boldsymbol{x}})\in\Upsilon(\boldsymbol{x})$, and sum all the corresponding probabilities to obtain (20). ∎

Here, we work through Example 1 to illustrate the above approach for calculating the stationary probabilities; again a more illustrative and simple example ($k=1$) is in [20].

**Example 2:** Consider $k=2$, i.e., 2-LRU algorithm. Consider state $\boldsymbol{x}=((34),(12))$, i.e., (12) are in the second level and (34) are in the first level. From Example 1, $|\Upsilon(\boldsymbol{x})|=9$. Consider a the set of sample paths $\Lambda(2\to2\to1\to1\to4\to3)$. Each sample path $\gamma(\boldsymbol{x})$ in this set must satisfy one of the following conditions: (a) There is no request between any fixed items in $\gamma(\boldsymbol{x})$. In this case, $\Xi(\gamma(\boldsymbol{x}))=1$; (b) In general, between the two fixed instances of $2\to2$, we can request one of the items in $\{1,3,4,5\}$ any number of times. Give these requests, between the two fixed items $2\to1$, we can request item 1 any number of times if it has been requested between $2\to2$, or request one item in $\{1,3,4,5\}$ once that has not been requested between $2\to2$. Given these requests, between the fixed items $1\to1$, we can request one item in $\{3,4,5\}$ once that has not been requested before. Given these requests, between the two fixed items $1\to4$, we can request one item from $\{3,4,5\}$ once that has not been requested before. Given these requests, between the two fixed items $4\to3$, we cannot request any other item. Note that the way of choosing different items between any fixed two items on the arrangement $2\to2\to1\to1\to4\to3$ define different subclass of sample paths $\tilde{\Lambda}(2\to2\to1\to1\to4\to3)$. In this way, we consider all the possible $\gamma(\boldsymbol{x})$ for the subclass of sample path $\tilde{\Lambda}(2\to2\to1\to1\to4\to3)\in\Lambda(2\to2\to1\to1\to4\to3)$. Computing the corresponding probabilities, we can obtain $\Xi(\gamma(\boldsymbol{x}))$ for each $\gamma(\boldsymbol{x})$. Then summing them all up, we get $\sum_{\gamma(\boldsymbol{x})\in\Lambda(\hat{\boldsymbol{x}})}\Xi(\gamma(\boldsymbol{x}))$ for this particular class $\Lambda(2\to2\to1\to1\to4\to3)$.

Similarly, we can consider sample paths corresponding to all the classes in $\Upsilon(\boldsymbol{x})$ to obtain the steady state probability $\pi^{*}_{2\text{-LRU}}(\boldsymbol{x})$ for state $\boldsymbol{x}$.

### C. Multi-level Caching Algorithms

Suppose there are $h$ caches in the linear cache network (see Figure 1 (b)). Each item enters the cache network via cache 1 and moves up one cache upon a cache hit. For simplicity, we denote $x_{(i,j)}$ as the identity of the item at position $j$ in cache $i$, where $i=1,\cdots,h$ and $j=1,\cdots,m_i$. Denote $\pi^{*}_{A}(\boldsymbol{x})$ as the stationary probability of state $\boldsymbol{x}=(\boldsymbol{x}_1,\cdots,\boldsymbol{x}_h)$, with $\boldsymbol{x}_i=(x_{(i,1)},\cdots,x_{(i,m_i)})$. Following a similar sample path argument as discussed for $k$-LRU in Section IV-B, we have

**Theorem 3:** Under the IRM, the steady state probabilities of LRU($\boldsymbol{m}$) satisfies the following form $\pi^{*}_{\text{LRU}(\boldsymbol{m})}(\boldsymbol{x})=$

$$\prod_{i=1}^{h}\left(\prod_{j=1}^{m_i}p_{x_{(i,j)}}\right)^{i}\sum_{\Lambda(\hat{\boldsymbol{x}})\in\Upsilon(\boldsymbol{x})}\sum_{\tilde{\Lambda}(\hat{\boldsymbol{x}})\in\Lambda(\boldsymbol{x})}\sum_{\gamma(\boldsymbol{x})\in\tilde{\Lambda}(\hat{\boldsymbol{x}})}\Xi(\gamma(\boldsymbol{x})). \tag{23}$$

The proof is presented in [20]. This form allows us to analyze the mixing time of LRU($\boldsymbol{m}$)[6] through the conductance argument in Section III-B.

**Remark** *3:* We can directly compute the hit probability of each algorithm once we have the stationary distribution. Due to space constraints, we relegate the details to [20].

## V. Analysis of Mixing time

We characterize the mixing times of LRU, FIFO, RANDOM, CLIMB, $k$-LRU, LRU($\boldsymbol{m}$) using the result in Lemma 1 and Corollary 1. As it is convention to present mixing time results with $\epsilon=\frac{1}{2}$, here onwards we will omit $(\epsilon)$ in $t_{\text{mix}}$. Due to space constraints, we only present the proof for $k$-LRU and relegate all other proofs to [20].

### A. Mixing Time of Single-Level Cache

*1) Mixing Time of LRU:* We consider the IRM arrival process and denote the probability of requesting item $i$ by $p_i$. It is easily shown that the Markov chain associated with the LRU algorithm is non-reversible, e.g., using the Kolmogorov condition [30]. Hence, as discussed in Section III-B4, we first need to construct the time reversal $P^{\text{LRU},*}$, given the transition matrix $P^{\text{LRU}}$ of LRU. Then the Markov chain with transition matrix $\frac{P^{\text{LRU}}+P^{\text{LRU},*}}{2}$ is reversible.

**Theorem 4:** The mixing time of LRU satisfies

$$t^{\text{LRU}}_{\text{mix}}=O\Bigg(\frac{n^{4m}\left(\prod_{j=1}^{m}p_j\right)^{4}\prod_{j=1}^{m-1}\left(1-\sum_{l=1}^{j}p_{n-l+1}\right)^{2}}{p_n^{2}\left(\prod_{j=1}^{m-1}\left(1-\sum_{l=1}^{j}p_l\right)\right)^{4}\prod_{j=n-m+1}^{n}p_j^{2}}\cdot\ln\left(\frac{\prod_{j=1}^{m-1}\left(1-\sum_{l=1}^{j}p_{n-l+1}\right)}{\prod_{j=n-m+1}^{n}p_j}\right)\Bigg). \tag{24}$$

**Corollary 2:** If the popularity distribution is Zipf($\alpha$), then the mixing time of LRU satisfies

$$t^{\text{LRU}}_{\text{mix}}=O(n^{(4\alpha+2)m+2}\ln n). \tag{25}$$

[6]CLIMB is LRU($\boldsymbol{m}$) with $h=m$ and size 1 for all levels.

*2) Mixing Time of RANDOM and FIFO:* Since these algorithms have reversible Markov chains, we can use (13) in Lemma 1 to bound the mixing times.

**Theorem 5:** The mixing times of RANDOM and FIFO satisfy $t_{\text{mix}}^{(\text{RANDOM, FIFO})} =$

$$O\left(\frac{n^{2m}\left(\prod_{i=1}^{m} p_i\right)^6}{p_n^2\left(\prod_{i=n-m+1}^{n} p_i\right)^6}\ln\left(\frac{n^m\prod_{i=1}^{m} p_i}{\prod_{i=n-m+1}^{n} p_i}\right)\right). \tag{26}$$

**Corollary 3:** If the popularity follows a Zipf distribution, then the mixing times of RANDOM and FIFO satisfies

$$t_{\text{mix}}^{(\text{RANDOM, FIFO})} = O(n^{(6\alpha+2)m+2}\ln n). \tag{27}$$

*3) Mixing Time of CLIMB:* We now turn to the CLIMB algorithm. Again we have a reversible Markov chain, and (13) in Lemma 1 is applied.

**Theorem 6:** The mixing time of CLIMB satisfies $t_{\text{mix}}^{\text{CLIMB}} =$

$$O\left(\frac{n^{2m}\left(\prod_{i=1}^{m} p_i^{m-i+1}\right)^6}{p_n^2\left(\prod_{i=n-m+1}^{n} p_i^{i-n+m}\right)^6}\ln\left(\frac{n^m\prod_{i=1}^{m} p_i^{m-i+1}}{\prod_{i=n-m+1}^{n} p_i^{i-n+m}}\right)\right). \tag{28}$$

**Corollary 4:** If the popularity distribution is Zipf($\alpha$), then the mixing time of CLIMB satisfies

$$t_{\text{mix}}^{\text{CLIMB}} = O(n^{3\alpha m(m+1)+2m+2}\ln n). \tag{29}$$

In the worst case, the most likely item takes $m-1$ steps to attain its final position, the next item takes $m-2$ steps, etc., giving the quadratic exponent term in the mixing time bound of CLIMB.

*4) Comparison of Mixing Times:* We are now in a position to compare the bounds on mixing times of our candidate algorithms for the simple cache system under a Zipf($\alpha$) distribution. While the results are all upper bounds on mixing time, they should allow us to make a judgement on the worst case behaviors of each algorithm, and are a conservative estimate on likely performance in practice. From Corollaries 2, 3, and 4, for any $m \geq 2$, the expected ordering in mixing times from smallest to largest is likely to be

$$t_{\text{mix}}^{\text{LRU}} \quad \leq \quad t_{\text{mix}}^{(\text{RANDOM, FIFO})} \quad \leq \quad t_{\text{mix}}^{\text{CLIMB}}. \tag{30}$$

The phenomenon of LRU mixing faster than CLIMB has been observed in earlier in simulation studies [11], [31]. However, analytical characterization and comparison of mixing times of single and multi-level caching algorithms has not been done earlier.

Finally, we note that under the Zipf distribution, the congestion of LRU, FIFO, RANDOM, CLIMB are all polynomial in the size of the state space, i.e., the corresponding associated Markov chains are all rapid mixing. The same phenomenon holds for a uniform distribution. However, if we consider non-heavy-tailed distributions, such as a geometric distribution, mixing time is likely to be severely degraded. This observation for LRU algorithm is discussed in [12].

### B. Mixing Time of Meta-Cache: $k$-LRU

We next characterize the mixing time of the meta-cache algorithm, $k$-LRU. Again the Markov chain associated with the $k$-LRU algorithm is non-reversible (using the Kolmogorov condition [30]). Hence, using Corollary 1 and the general form of the stationary distribution of $k$-LRU presented in Theorem 2, we should obtain mixing time bounds for this algorithm.

**Theorem 7:** The mixing time of $k$-LRU satisfies

$$t_{\text{mix}}^{k\text{-LRU}} = O\Bigg(\frac{n^{4km+4(m-1)[k(k+1)m/2-1]}}{\left(\prod_{i=1}^{k}\left(\prod_{l=1+(i-1)m}^{im} p_{n-l+1}\right)^{k-i+1}\right)^2\cdot p_n^2}$$
$$\left(\left(\prod_{j=1}^{m} p_j\right)^k\left(\frac{1}{1-\max_{\boldsymbol{x}\in\mathcal{S},\delta}\left(\sum_{j\in\chi_{(\delta,\delta+1)}^{\Lambda^{\max}(\boldsymbol{x})}} p_j\right)}\right)^{\frac{k(k+1)m}{2}-1}\right)^4$$
$$\ln\frac{1}{\prod_{i=1}^{k}\left(\prod_{l=1+(i-1)m}^{im} p_{n-l+1}\right)^{k-i+1}}\Bigg). \tag{31}$$

*Proof:* Consider the stationary probability $\pi_{k\text{-LRU}}^*(\boldsymbol{x})$ given in Equation (20). We will first obtain an upper bound $\pi_{\max}^*$ and a lower bound $\pi_{\min}^*$. We omit the subscript $k$-LRU for brevity.

First, we characterize $\pi_{\max}^*$. Recall that we denote the sequence of items fixed by the arrangement $\hat{\boldsymbol{x}}$ by $\xi_1,\cdots,\xi_\delta,\cdots,\xi_{|\hat{\mathcal{X}}|}$, where $\xi_\delta$ stands for the identity of the $\delta$-th item in this arrangement. A sample path contains other items from $\mathcal{L}$ between these fixed items in such a way that the end result is $\hat{\boldsymbol{x}}$ under $k$-LRU, with the probability of requesting these other items being $\Xi(\gamma(\boldsymbol{x}))$.

We obtain an upper bound on $\pi_{\max}^*$ by taking the maximum of each part in (20) as follows: (i) We take the maximum of the product form $\prod_{i=1}^{k}\left(\prod_{j=1}^{m}\mathcal{J}_{x_{(i,j)}}\right)^i$ over all states $\boldsymbol{x}\in\mathcal{S}$; (ii) We consider the maximum number of subclasses of sample paths and classes of sample paths; and (iii) We maximize the sum of $\Xi(\gamma(\boldsymbol{x}))$ over all states, sample paths and classes. Then we have

$$\pi^*(\boldsymbol{x}) \leq \max_{\boldsymbol{x}\in\mathcal{S}}\left(\prod_{i=1}^{k}\left(\prod_{j=1}^{m}\mathcal{J}_{x(i,j)}\right)^i\right)\cdot|\tilde{\Upsilon}_{\Lambda(\hat{\boldsymbol{x}})}(\boldsymbol{x})|\cdot|\Upsilon(\boldsymbol{x})|$$
$$\cdot\max_{\boldsymbol{x}\in\mathcal{S}}\left(\max_{\Lambda(\hat{\boldsymbol{x}})}\left(\sum_{\gamma(\boldsymbol{x})\in\Lambda(\hat{\boldsymbol{x}})}\Xi(\gamma(\boldsymbol{x}))\right)\right). \tag{32}$$

There are three terms in the above expression. We upper bound the first term by using our assumption that $p_1\geq\cdots\geq p_n$.

It is obvious $\max_{\boldsymbol{x}}\left(\prod_{i=1}^{k}\left(\prod_{j=1}^{m}\mathcal{J}_{x(i,j)}\right)^i\right) = \left(\prod_{j=1}^{m} p_j\right)^k$.

The second term is upper bounded by $(\sum_{j=1}^{j} jm)! = (k(k+1)m/2)!$, and the third term is upper bounded by $\left(\binom{n}{m-1}\right)^{k(k+1)m/2-1}$ since at most $m-1$ unique items can be requested between any two fixed items on the arrangement.

We then consider the fourth term in the expression. Now, for any $\boldsymbol{x}$

$$\sum_{\gamma(\boldsymbol{x})\in\Lambda(\hat{\boldsymbol{x}})}\Xi(\gamma(\boldsymbol{x})) = \prod_{\delta=1}^{|\hat{\mathcal{X}}|}\left(\sum_{\zeta=0}^{\infty}\left(\sum_{j\in\chi_{(\delta,\delta+1)}^{\Lambda(\hat{\boldsymbol{x}})}} p_j\right)^\zeta\right), \tag{33}$$

where $\chi_{(\delta,\delta+1)}^{\Lambda(\hat{\boldsymbol{x}})}$ is the set of items that can be requested between the $\delta$-th and $(\delta+1)$-th fixed items on the class of sample path $\Lambda(\hat{\boldsymbol{x}})$.

It is clear that this term is maximized when the number of product terms is in maximum, since each term in the product is greater than or equal to one. The maximum number of items that need to be fixed in a sample path leading to a state $\hat{\boldsymbol{x}}$ is $\sum_{j=1}^{k} jm = k(k+1)m/2$ (which happens when all the items in state $\boldsymbol{x}$ are distinct from each other).

Denote $\Lambda^{\max}(\boldsymbol{x}) = \arg\max_{\Lambda(\hat{\boldsymbol{x}})} \left(\sum_{\gamma(\boldsymbol{x})\in\Lambda(\hat{\boldsymbol{x}})} \Xi(\gamma(\boldsymbol{x}))\right)$ as the class of sample path that achieves the maximum in the third term (33). Let $\chi_{(\delta,\delta+1)}^{\Lambda^{\max}(\boldsymbol{x})}$ be the set of items that can be requested between the $\delta$-th and $(\delta+1)$-th fixed items on this class of sample path.

Then we have

$$\begin{aligned}
&\max_{\boldsymbol{x}\in\mathcal{S}}\left(\max_{\Lambda(\hat{\boldsymbol{x}})}\left(\sum_{\gamma(\boldsymbol{x})\in\Lambda(\hat{\boldsymbol{x}})}\Xi(\gamma(\boldsymbol{x}))\right)\right) = \max_{\boldsymbol{x}\in\mathcal{S}}\left(\sum_{\gamma(\boldsymbol{x})\in\Lambda^{\max}(\boldsymbol{x})}\Xi(\gamma(\boldsymbol{x}))\right)\\
&\overset{(a)}{=} \max_{\boldsymbol{x}\in\mathcal{S}}\left(\prod_{\delta=1}^{k(k+1)m/2-1}\left(\sum_{\zeta=0}^{\infty}\left(\sum_{j\in\chi_{(\delta,\delta+1)}^{\Lambda^{\max}(\boldsymbol{x})}} p_j\right)^{\zeta}\right)\right)\\
&\overset{(b)}{\le} \prod_{\delta=1}^{k(k+1)m/2-1}\left(\sum_{\zeta=0}^{\infty}\left(\max_{\boldsymbol{x}\in\mathcal{S},\delta}\left(\sum_{j\in\chi_{(\delta,\delta+1)}^{\Lambda^{\max}(\boldsymbol{x})}} p_j\right)\right)^{\zeta}\right)\\
&= \left(\frac{1}{1-\max_{\boldsymbol{x}\in\mathcal{S},\delta}\left(\sum_{j\in\chi_{(\delta,\delta+1)}^{\Lambda^{\max}(\boldsymbol{x})}} p_j\right)}\right)^{k(k+1)m/2-1}. \qquad (34)
\end{aligned}$$

Note that in (34), (a) follows from the discussion above, and (b) is true since we take the maximum of the probabilities over all the items that can be requested between any two fixed items over all possible states, sample paths and classes.

Hence, we have

$$\begin{aligned}
\pi^*(\boldsymbol{x}) &\le \left(\prod_{j=1}^{m} p_j\right)^{k}\left(\frac{k(k+1)m}{2}\right)!\left(\binom{n}{m-1}\right)^{\frac{k(k+1)m}{2}-1}\\
&\cdot\left(\frac{1}{1-\max_{\boldsymbol{x}\in\mathcal{S},\delta}\left(\sum_{j\in\chi_{(\delta,\delta+1)}^{\Lambda^{\max}(\boldsymbol{x})}} p_j\right)}\right)^{k(k+1)m/2-1} \triangleq \pi^*_{\max}. \qquad (35)
\end{aligned}$$

Next, we characterize $\pi^*_{\min}$. We obtain $\pi^*_{\min}$ by taking the minimum of each part in Equation (20): (i) We take the minimum over the product form $\prod_{i=1}^{k}\left(\prod_{j=1}^{m}\mathcal{J}_{x(i,j)}\right)^{i}$ over all states $\boldsymbol{x}\in\mathcal{S}$; (ii) We consider the minimum number of subclass of sample path and the minimum number of class of sample path, i.e., $|\tilde{\Upsilon}_{\Lambda(\hat{\boldsymbol{x}})}(\boldsymbol{x})| = 1$ and $|\Upsilon(\boldsymbol{x})| = 1$; and (iii) We minimize the sum of $\Xi(\gamma(\boldsymbol{x}))$ over all states, sample paths and classes.

Then we have

$$\begin{aligned}
\pi^*(\boldsymbol{x}) &\ge \min_{\boldsymbol{x}\in\mathcal{S}}\left(\prod_{i=1}^{k}\left(\prod_{j=1}^{m}\mathcal{J}_{x(i,j)}\right)^{i}\right)\cdot 1\cdot 1\\
&\cdot \min_{\boldsymbol{x}\in\mathcal{S}}\left(\min_{\Lambda(\hat{\boldsymbol{x}})}\left(\sum_{\gamma(\boldsymbol{x})\in\Lambda(\hat{\boldsymbol{x}})}\Xi(\gamma(\boldsymbol{x}))\right)\right). \qquad (36)
\end{aligned}$$

Again, there are three terms in the above expression. We may lower bound the first term by using our assumption that $p_1 \ge \cdots \ge p_n$. It is obvious that this term is lower bounded when the least popular $km$ distinct items are stored in the cache, following the sequence of popularity decreasing with increasing levels. Thus, we have the least popular $m$ items in the $k$-th level, the next least popular items $m+1$ to $2m$ items in the $(k-1)$-level, and so on until the whole cache is filled with the least popular $km$ distinct items, i.e.,

$$\begin{aligned}
\min_{\boldsymbol{x}\in\mathcal{S}}\left(\prod_{i=1}^{k}\left(\prod_{j=1}^{m}\mathcal{J}_{x(i,j)}\right)^{i}\right) &= \left(\prod_{j=1}^{m} p_{n-j+1}\right)^{k}\cdots\left(\prod_{j=(k-1)m+1}^{km} p_{n-j+1}\right)^{1}\\
&= \prod_{i=1}^{k}\left(\prod_{l=1+(i-1)m}^{im} p_{n-l+1}\right)^{k-i+1}. \qquad (37)
\end{aligned}$$

The second and third terms are both already lower bounded by 1.

We then consider the fourth term in the expression. Again, by (33), we know that this term is greater than or equal to one (which happens when all terms equal to one). Since we only consider one class of sample path, we fix the items of the current state (which leads to the first term), and then consider all the possible requests between each two fixed items. Since we want to lower bound the third term, we consider the case where there are no further requests between any two fixed items, i.e., $\min_{\boldsymbol{x}\in\mathcal{S}}\left(\min_{\Lambda(\hat{\boldsymbol{x}})}\left(\sum_{\gamma(\boldsymbol{x})\in\Lambda(\hat{\boldsymbol{x}})}\Xi(\gamma(\boldsymbol{x}))\right)\right) = 1$.

Hence, we have

$$\begin{aligned}
\pi^*(\boldsymbol{x}) &\ge \min_{\boldsymbol{x}\in\mathcal{S}}\left(\prod_{i=1}^{k}\left(\prod_{j=1}^{m}\mathcal{J}_{x(i,j)}\right)^{i}\right)\\
&\ge \prod_{i=1}^{k}\left(\prod_{l=1+(i-1)m}^{im} p_{n-l+1}\right)^{k-i+1} \triangleq \pi^*_{\min}. \qquad (38)
\end{aligned}$$

Therefore, given that $\Gamma = O(n^{km})$, we have

$$\begin{aligned}
\rho &\le \frac{1}{\pi^*_{\min}P_{\min}}\cdot\Gamma^2\cdot(\pi^*_{\max})^2\\
&= O\Bigg(\frac{n^{2km}\cdot n^{2(m-1)[k(k+1)m/2-1]}}{\prod_{i=1}^{k}\left(\prod_{l=1+(i-1)m}^{im} p_{n-l+1}\right)^{k-i+1}\cdot p_n}\left(\left(\prod_{j=1}^{m} p_j\right)^{k}\right.\\
&\left.\cdot\left(\frac{1}{1-\max_{\boldsymbol{x}\in\mathcal{S},\delta}\left(\sum_{j\in\chi_{(\delta,\delta+1)}^{\Lambda^{\max}(\boldsymbol{x})}} p_j\right)}\right)^{\frac{k(k+1)m}{2}-1}\right)^{2}\Bigg). \qquad (39)
\end{aligned}$$

Then follow Corollary 1, we have (31). ■

**Corollary 5:** If the popularity distribution is Zipf($\alpha$), then the mixing time of $k$-LRU satisfies

$$t_{\text{mix}}^{k\text{-LRU}} = O(n^{(k+1)k(2m-1)m+4(k\alpha-1)m+6}\ln n). \qquad (40)$$

**Remark** *4:* From (40), it is clear that increasing the number of levels $k$, the upper bound of mixing time increases. We will see in Figure 4 in Section VII-A that the meta-cache enhances the hit probability. However, as 2-LRU has a larger mixing time upper bound than LRU, this accuracy is at the expense of an increased mixing time.

### C. Mixing time of Multi-level Cache: LRU($\boldsymbol{m}$)

Finally, we characterize the mixing time of the multi-level caching algorithm, LRU($\boldsymbol{m}$). The LRU($\boldsymbol{m}$) algorithm Markov chain is also non-reversible (by using the Kolmogorov condition [30]). Hence, we should obtain mixing time bounds for this algorithm based on Corollary 1, by using the general form of the stationary distribution of LRU($\boldsymbol{m}$) presented in Theorem 3.

**Theorem 8:** The mixing time of LRU($\boldsymbol{m}$) satisfies

$$t_{\text{mix}}^{\text{LRU}(\boldsymbol{m})} = O\Bigg( \frac{n^{4m+4(m-1)(m_1+\cdots+hm_h-1)}}{\prod_{i=1}^{h}\left(\prod_{k=1+\sum_{j=1}^{i-1}m_j}^{\sum_{j=1}^{i}m_j} p_{n+k-m}\right)^{2i}\cdot p_n^2} \left(\prod_{i=1}^{h}\left(\prod_{k=1+\sum_{j=1}^{i-1}m_{h-j+1}}^{\sum_{j=1}^{i}m_{h-j+1}} p_k\right)^{h-i+1}\right)^4 \cdot\left(\left(\frac{1}{1-\max_{\boldsymbol{x}\in\mathcal{S},\delta}\left(\sum_{j\in\chi_{(\delta,\delta+1)}^{\Lambda^{\max}(\boldsymbol{x})}} p_j\right)}\right)^{\sum_{j=1}^{h} jm_j-1}\right)^4 \ln\frac{1}{\prod_{i=1}^{h}\left(\prod_{k=1+\sum_{j=1}^{i-1}m_j}^{\sum_{j=1}^{i}m_j} p_{n+k-m}\right)^{i}}\Bigg). \tag{41}$$

where $m = m_1 + m_2 + \cdots + m_h$ and define $\sum_{j=1}^{i-1} m_{h-j+1} = 0$ for $i = 1$.

**Corollary 6:** If the popularity distribution is Zipf($\alpha$), then the mixing time of LRU($\boldsymbol{m}$) satisfies

$$t_{\text{mix}}^{\text{LRU}(\boldsymbol{m})} = O(n^{(4m+4\alpha-6)(m_1+2m_2+\cdots+hm_h)+6}\ln n), \tag{42}$$

where $m = m_1 + m_2 + \cdots + m_h$.

**Remark** *5:* The mixing time bounds obtained in the above analysis are not as tight as ones that could be obtained directly by the characterization of the spectral gap of a Markov Chain and using the bound presented in (7). However, accurate determination of the spectral gap using the eigenvalues of a Markov chain (if real and positive) is typically difficult. In the special case of LRU, [12], [32], [33] provide all the eigenvalues, which can then be used to tighten the mixing time bound to $O(m \ln n)$. However, such eigenvalue-based results are not generalizable to 2-LRU and above, or indeed to any of the other algorithms considered in this paper. Hence, we use the simpler and uniform bound that, nevertheless, brings out first-order dependence on algorithm parameters: e.g., the exponent of the mixing time upper bound depends quadratically in $k$ and $h$ for $k$-LRU and for $h$-level LRU($\boldsymbol{m}$) paralleling the dramatic increase in mixing time observed in practice/numerically.

## VI. A-LRU

Our analysis thus far shows that different caching algorithms choose a different trade-off between speed and accuracy of learning. LRU has been widely used due to its speed of learning and ease of implementation. FIFO and RANDOM have been used to replace LRU in some scenarios since they are easier to implement with tolerable performance degradation. CLIMB has been numerically shown to have a higher hit ratio than LRU, at the expense of increased time to reach this steady state in comparison to LRU. ARC is an online algorithm with a self-tuning parameter, which has good performance in some real systems but with complex implementation. $k$-LRU has relatively low complexity, which requires just one parameter, i.e., the number of meta caches $k-1$. We will see that these meta caches will provide a significant improvement in hit probability over LRU even for small values of $k$, with most of the gain achieved by $k = 2$. LRU($\boldsymbol{m}$) too has relatively low complexity, and provides much higher hit probability over LRU. Both schemes pay for the higher hit probability in terms of much slower speed of learning.

Based on the previous analysis, we propose a novel hybrid algorithm, Adaptive-LRU (A-LRU), which captures advantages of LRU, 2-LRU and LRU(2), i.e., it learns both faster and better about the changes in the popularity. A $k$-level version of A-LRU will allow for an interpolation between LRU, 2-LRU, $\cdots$, $k$-LRU, while at the same time incorporating multi-levels as in LRU($\boldsymbol{m}$).

**Adaptive-LRU (A-LRU):** We define the quantities $c_1 = \min(1, \lfloor(1-\beta)m\rfloor)$, $c_2 = \lfloor(1-\beta)m\rfloor$, $c_3 = \lfloor(1-\beta)m\rfloor + 1$ and $c_4 = \max(m, \lfloor(1-\beta)m\rfloor + 1)$, where $\beta \in [0,1]$ is a parameter. We partition the cache into two parts with $C2$ defined as the positions from $c_1 \cdots c_2$ and $C1$ as the positions from $c_3 \cdots c_4$. We also define the quantities $m_1 = \min(1, \lfloor\beta m\rfloor)$, $m_2 = \lfloor\beta m\rfloor$, $m_3 = \lfloor\beta m\rfloor + 1$ and $m_4 = \max(m, \lfloor\beta m\rfloor + 1)$. We associate positions $m_1 \cdots m_2$ with meta cache[7] $M2$ and $m_3 \cdots m_4$ with meta cache $M1$. Note that value $m_1 = m_2 = 0$ is an extreme point that yields behavior similar to 2-LRU, while $m_3 = m_4 = m+1$ yields LRU. See Section VII. The cache partitions are shown in Figure 2.

Let us denote the meta-data associated with a generic item $i$ by $M(i)$. If item $i$ is requested, the operation of A-LRU is illustrated in Figure 2. There are two possibilities:

(1) **Cache miss**, then there are three cases to consider:

(1a) $M(i) \notin M1 \cup M2$ : If $c_3 \neq m+1$, $i$ is inserted into cache position $l = c_3$, else (extreme case similar to 2-LRU) $M(i)$ is inserted into meta cache position $l = m_3$. Cache/meta cache items in positions greater than $l$ move back one position, and the last meta-data is evicted;

(1b) $M(i) \in M1$ : Item $i$ is inserted into position $c_1$, all other items in $C2$ move back one position, the meta data of item in cache position $c_2$ is placed in position $m_1$, all other meta-data items move back one position, and the meta data in position $m_2$ moves to position $m_3$;

(1c) $M(i) \in M2$ : If $c_1 = 1$, item $i$ is inserted into position $l = c_1$. All other items in $C2$ move back one position, and the meta data of item in cache position $c_2$ is placed in position $m_1$. Note that this situation cannot occur in the extreme case of LRU, since $M2$ is always empty for LRU;

(2) **Cache hit**, then there are two cases to consider

(2a) $i \in C1$ (suppose in position $j$): If $c_1 = 1$, then item $i$ moves to cache position $l = c_1$, else (extreme case of LRU) item $i$ moves to cache position $l = c_3$. If $l = c_1$, all other items in $C2$ move back one position, the item in cache position $c_2$

[7]Meta cache is also called virtual cache, which only stores meta-data.

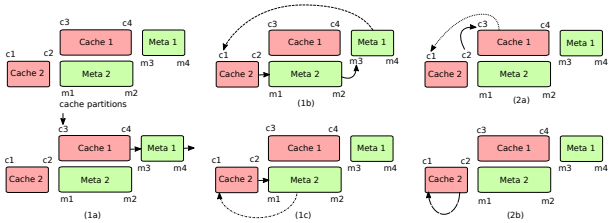


Fig. 2. Operation of the A-LRU algorithm.

is placed in position $c_3$, all other items in $C1$ upto position $j$ move back one position. If $l = c_3$ (extreme case of LRU), all other items in $C1$ upto position $j$ move back one position.

(2b) $i \in C2$ (suppose in position $j$): Item $i$ moves to cache position $c_1$, and all other items in positions $\min(2, c_2)$ to $j-1$ move back one position.

Finally, note that the A-LRU setup can be generalized to as many levels as desired by simply "stacking up" sets of real and meta caches, and following the same caching and eviction policy outlined above (where (1a) would apply to the top level, while (1c) and (2b) would apply to the bottom level). In that case, it would be parameterized by $\beta_1, \cdots, \beta_k$ with $\sum_{i=1}^{k} \beta_i = 1$, for a $k$-level A-LRU. The cache size at level $i$ is $m - \lfloor (1-\beta_i)m \rfloor$, and meta-cache size at level $i$ is $\lfloor (1-\beta_i)m \rfloor$. Note that small corrections have to be made to the above selections for a particular value of $k$ (as we did in the case of $k = 2$ described above) to ensure that the total amount of cache space is limited to $m$.

**Dynamic A-LRU:** Whereas in our description of A-LRU, we use a fixed partitioning parameter $\beta$, the algorithm (and an implementation of it) can easily consider time-varying $\beta$ values. For concreteness, we consider a $k$ levels A-LRU with a sequence $\{\chi_1(t), \chi_2(t), \cdots, \chi_{k-1}(t)\}_{t=1}^{\infty}$, with each term going to 0 as the number of requests go to infinity, satisfying (i) $\sum_t \chi_i(t) \to \infty$; (ii) $\sum_t \chi_i^2(t) < \infty$; and (iii) $\chi_i(t)/\chi_{i+1}(t) \to 0$. Here, $\chi_1$ stands for the proportion of LRU to the rest, $\chi_2$ stands for the proportion between 2-LRU and 3-LRU to the rest, etc. A typical choice of sequences will be $\chi_i(t) = m/(m + \max(0, t - T_i)\frac{i+1}{2i}/c_i)$, where $t$ counts the number of requests, and $T_i, c_i > 0$ are parameters to be varied. Under such setting, the $\beta$s in the previous definition of A-LRU satisfy that (i) at level $i \le k-1$, it is $(1-\chi_1(t))(1-\chi_2(t))\cdots(1-\chi_{i-1}(t))\chi_i(t)$, and (ii) at level $k$, it is $(1-\chi_1(t))(1-\chi_2(t))\cdots(1-\chi_{k-1}(t))$.

In particular, we consider the 2-level A-LRU shown in Figure 2. Here, the $\beta$s are $\beta_1(t) = m/(m + \max\{0, t-T\}/c)$ and $\beta_2(t) = 1 - \beta_1(t)$, where $T$ and $c$ are parameters. With such a sequence of $\beta$s, A-LRU will start at 1 (LRU) and (slowly) decrease to 0 (2-LRU).

**Remark** If the popularity distribution changes with time (in Section VIII), we should only consider constant $\beta$ algorithms. These two distinctions follow from stochastic approximation ideas where while decreasing step-size algorithms can converge to optimal solutions in stationary settings, constant step-size algorithms provide good tracking performance for non-stationary settings. Therefore, we denote A-LRU(f) and A-LRU(d) to distinguish A-LRU with a fixed or dynamic $\beta$, respectively.

## VII. Performance Evaluations

### A. Permutation Distance

Since the $\tau$-distance characterizes how accurately an algorithm learns the popularity distribution, a smaller $\tau$-distance should correspond to a larger hit probability. We illustrate how different algorithms perform using a content library size of $n = 20$, and using caches of size $2, 3, 4, 5$. Figures 3 and 4 compare the $\tau$-distance and hit probabilities of various caching algorithms. The points on each curve correspond to cache size of $2, 3, 4, 5$ from left to right. For example, all the points in the dashed pink circle correspond to cache size of 3. Since the cache size should be an integer, we partition the cache for LRU($\boldsymbol{m}$) and A-LRU such that the size of cache 1 is always 1, and the remaining cache size is allocated to cache 2. Note that this is simply for illustration, and A-LRU's highest hit-probability is when it mirrors 2−LRU. From Figures 3 and 4, we can see that the $\tau$-distance and hit probability follow the same rule, i.e., a smaller $\tau$-distance corresponds to a larger hit probability, which is as expected.

**Remark** *6:* The specific parameters in (3) used to produce Figures 3 and 4 are as follows. We consider a Zipf popularity distribution with $\alpha = 0.8$; for simplicity, we set the element weights as $w_i = n - i + 1$, and the swapping cost $\zeta_i = \log i$ for $i > 1$, and $\zeta_1 = 0.1$. Since different choices of weights result in different values of the $\tau$-distance, the y-axis value in Figures 3 and 4 is only used to show the relative difference between algorithms.

### B. Learning Error

We consider a cache system with $(n, m) = (20, 4)$, where $n$ is the total number of different items in the library and $m$ is the cache size. We consider requests following the IRM model, with a Zipf popularity law with parameter $\alpha = 0.8$. We compare the performance of LRU, FIFO, RANDOM, CLIMB, LRU($\boldsymbol{m}$), 2-LRU, ARC and A-LRU with respect to the stationary hit probability and learning error. In particular, we consider a two-level version of A-LRU which is characterized by a parameter $\beta \in [0, 1]$ that determines the interpolation rate between LRU and 2-LRU; the detailed description is in Section VI. For LRU($\boldsymbol{m}$), we consider the capacity allocation as $(m_1, m_2) = (1, 3)$. The corresponding stationary hit probability of these algorithms are 0.325, 0.308, 0.308, 0.414, 0.407, 0.408, 0.352, 0.408, respectively.

The learning error of these algorithms as a function of the number of requests received is illustrated in Figure 5 for single-level caching algorithms (from Figure 1 (a)), and Figure 6 for multi-level caching algorithms, (from Figure 1 (b), (c), (d)), respectively, where the y-axis is shown in a logarithmic scale. Note that the results for A-LRU presented in Figure 6 used $c = 500$ and $T = 1250$. We see immediately that FIFO and RANDOM have higher learning error than the other algorithms, regardless of the number of requests, which corresponds to the smallest stationary hit probability. This shows why their performance is poor, no matter how long they are trained. The learning error for LRU decreases fast initially and then levels off, whereas 2-LRU and LRU($\boldsymbol{m}$) have a slower decay rate, but the eventual error is lower

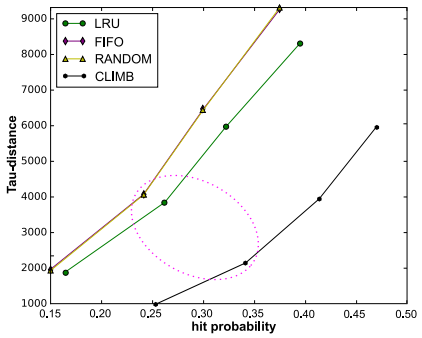


Fig. 3. Stationary $\tau$-distance vs. hit probability for single-level caching with IRM arrivals.

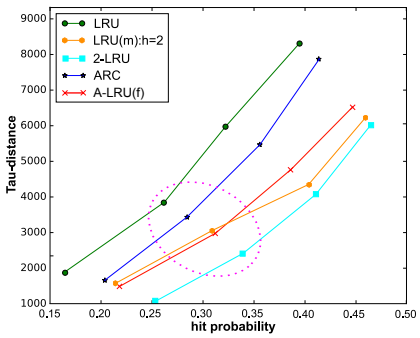


Fig. 4. Stationary $\tau$-distance vs. hit probability for multi-level caching with IRM arrivals.

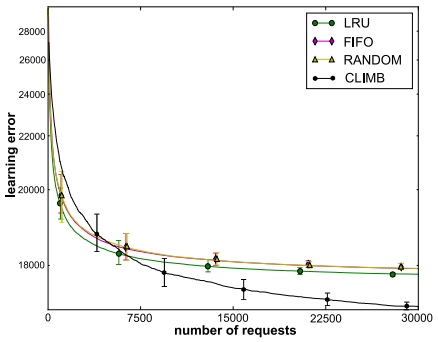


Fig. 5. Dynamic learning error of single-level caching under IRM arrivals.

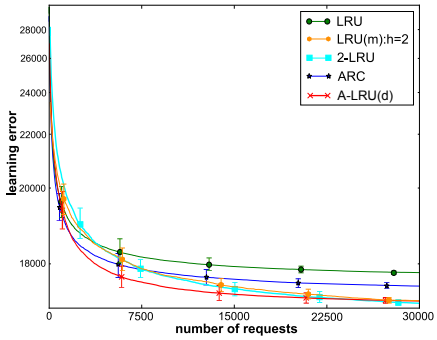


Fig. 6. Dynamic learning error of multi-level caching under IRM arrivals.

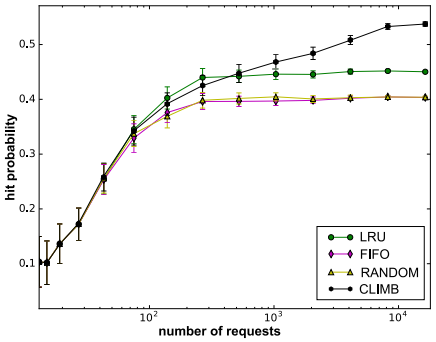


Fig. 7. Hit probability of single-level caching under IRM arrivals.

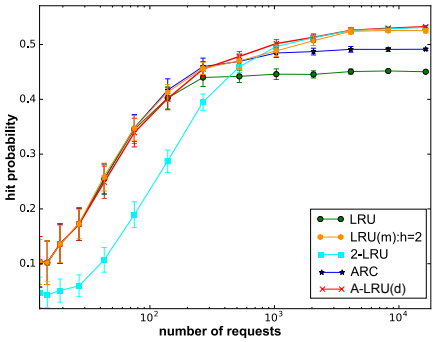


Fig. 8. Hit probability of multi-level caching under IRM arrivals.

than that of LRU. This corresponds to faster mixing of LRU but a poorer eventual accuracy ($\tau$-distance) as compared to 2-LRU and LRU($m$), which are formally characterized in Section III. ARC has a good performance initially, but it too levels off to an error larger than 2-LRU. CLIMB eventually has good performance, but it has a much slower decay rate. This corresponds to the slow mixing of CLIMB when compared to LRU, FIFO and RANDOM, consistent with our analysis in Theorems 5, 6 and 7, Section III-B. A-LRU learns fast initially, and then smoothly transitions to learning accurately, which captures both the merits of LRU and 2-LRU, i.e., accurate learning and fast mixing, and is able to attain a low learning error quickly.

### C. Hit Probabilities

The effects seen in Figures 5 and 6 are also visible in the evolution of hit probabilities shown in Figures 7 and 8, respectively, where the x-axis is on a logarithmic scale. Here, we choose ($n = 150, m = 30$) in order to explore a range of cache partitions for LRU($m$) and A-LRU from $(0, 30)$–$(30, 0)$. We compare the upper envelope of achievable hit probability by LRU($m$) and A-LRU with various other caching algorithms. We find that for any given learning time (requests), there is a cache partition such that A-LRU will attain a higher hit probability after learning for that time. These effects become more pronounced as the partition space (cache size) available for A-LRU increases.

Finally, we characterize the impact of the number of levels on the performance. We compare the hit probability of $k$-LRU for $k = 1, 2, 3, 4$ with $(n, m) = (50, 10)$. We observe that as the number of cache levels $k$ increases, the resulting algorithm achieve a higher hit probability (i.e., higher accuracy) at the expense of much larger number of requests (i.e., larger mixing time), which is consistent with the mixing time analysis of $k$-LRU shown in Section V-B. Due to space constraints, the plot is shown in [20].

## VIII. Trace-based Simulations

The ideas presented thus far have been based on the hypothesis that the request distribution changes dynamically, and hence an optimal caching algorithm should track the changes at a time scale consistent with the time scale of change. We now validate this hypothesis and the benefits of using the A-LRU algorithm using two different trace-based simulations.

### A. YouTube Trace

We use a publicly available data trace [7] that was extracted from a 2-week YouTube request traffic dump between June 2007 and March 2008. The trace contains a total of $611,968$ requests for $303,331$ different videos. About 75% of those videos were requested only once during the trace. There is no information on the video sizes. We therefore assume that the cache size is expressed as the number of videos that can be stored in it. This should have a low impact if the correlation between video popularity and video size is low. We find that $\alpha = 0.605$ is the best fit for a Zipf distribution. However, a detailed inspection shows that this trace exhibits significant non-stationarity, i.e., the popularity distribution is time-varying.

We compare the hit performance of different algorithms by varying cache size. Figures 9 and 10 depicts the hit

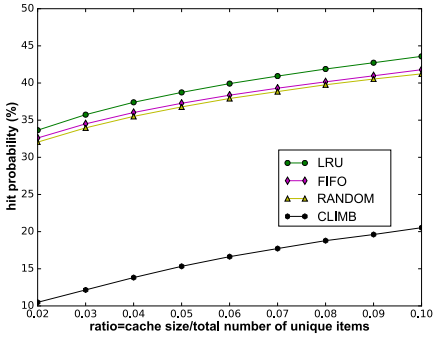


Fig. 9. Hit probability vs. cache size for various single-level caching algorithms with two-week long YouTube trace [7].

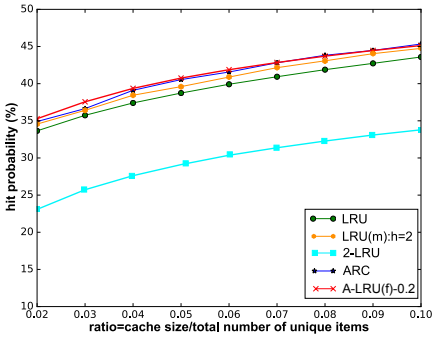


Fig. 10. Hit probability vs. cache size for various multi-level caching algorithms with two-week long YouTube trace [7].

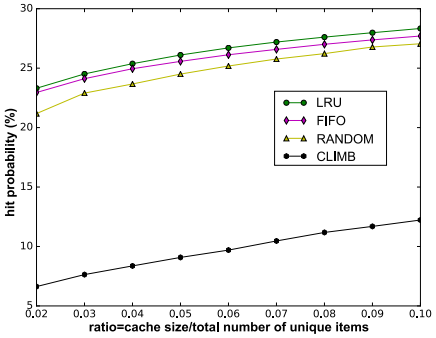


Fig. 11. Hit probability vs. cache size for various single-level caching algorithms with one particular day YouTube trace [7].

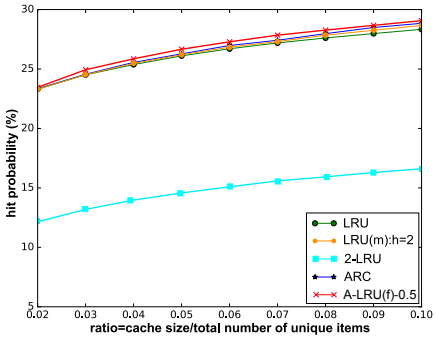


Fig. 12. Hit probability vs. cache size for various multi-level caching algorithms with one particular day YouTube trace [7].

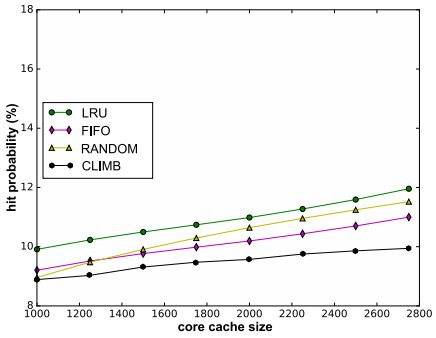


Fig. 13. Hit probability vs. cache size for various single-level caching algorithms with SD Network trace [34] for ICN.

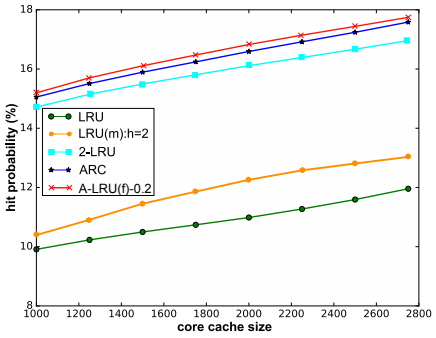


Fig. 14. Hit probability vs. cache size for various multi-level caching algorithms with SD Network trace [34] for ICN.

probability as a function of the cache size when the total number of unique videos is $n = 303,331$, and we use the ratios $m/n = 0.01, \cdots, 0.10$. For ease of visualization, we only depict A-LRU with the optimal $\beta$, which outperforms all the other caching algorithms.

We also conduct experiments on a one-day YouTube trace to illustrate the adaptability of A-LRU. We randomly pick one day from the two-week traces, in which the total number of uniques videos is $3 \times 10^5$ and the Zipf distribution parameter $\alpha = 0.48$ (but popularity varies with time). Figures 11 and 12 depicts the hit probability as a function of the cache ratio. We see that A-LRU again outperforms all other caching algorithms.

### B. ICN Traces

We run similar experiments using the traces from the IRCache project [34], with attention on data gathered from the SD Network Proxy (the most loaded proxy to which end-users can connect) in Feb. 2013. A detailed study shows that such traces capture regional traffic and exhibit significant non-stationaries due to daily traffic fluctuations. We only considered the traces in the 4 hour peak traffic period of each day in order to measure the performance expected in the busy hour. The traces contain 3416817 to 4121865 requests for 811827 to 993711 different data. About 67.61% to 69.27% of those data were requested only once during the trace. We find that $\alpha = 0.814$ to $\alpha = 0.821$ are the best fits for Zipf distributions.

Figures 13 and 14 show the overall cache hit probability versus the core cache size. We again observe that there is a significant improvement using A-LRU, especially with $\beta = 0.2$.

## IX. Conclusion

In this paper, we attempted to characterize the adaptability properties of different caching algorithms when confronted with non-stationary request arrivals. For this we cast caching algorithm design as an online popularity distribution estimation problem with stringent computational and memory restrictions. In this context we proposed the $\tau$-distance metric to measure the accuracy of learning of any given caching algorithm with respect to an ideal genie-aided scheme. To elucidate our online learning perspective, we first considered the stationary distributions of various caching algorithms under a stationary request process, and computed the $\tau$-distance between each one and the optimal content placement in the cache. We then analyzed the mixing time of each algorithm, to determine how long each one takes to attain stationarity. By combining both of these metrics, we constructed the *learning error,* which characterizes the tradeoff between speed and accuracy of learning achieved by each known caching algorithm. The learning error provides insight into the likely performance of each algorithm under non-stationary requests, and using the insights learned we developed a new algorithm, A-LRU, that can adapt to different non-stationary request processes and consequently has a higher hit probability than any of the standard algorithms that we compared against, under both synthetic and trace-based evaluation.

## REFERENCES


[1] E. G. Coffman and P. J. Denning, *Operating Systems Theory*. Prentice-Hall Englewood Cliffs, NJ, 1973.

[2] D. Starobinski and D. Tse, "Probabilistic Methods for Web Caching," *Performance evaluation*, 2001.

[3] N. Gast and B. Van Houdt, "Asymptotically Exact TTL-Approximations of the Cache Replacement Algorithms LRU(m) and h-LRU," in *Proc. of ITC*, 2016.

[4] V. Martina, M. Garetto, and E. Leonardi, "A Unified Approach to the Performance Analysis of Caching Systems," in *Proc. of INFOCOM*, 2014.

[5] N. Megiddo and D. S. Modha, "ARC: A Self-Tuning, Low Overhead Replacement Cache." in *Proc. of FAST*, 2003.

[6] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "Analyzing the Video Popularity Characteristics of Large-Scale User Generated Content Systems," *IEEE/ACM Transactions on Networking (TON)*, 2009.

[7] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Watch Global, Cache Local: YouTube Network Traffic at a Campus Network: Measurements and Implications," in *Electronic Imaging*, 2008.

[8] L. A. Belady, "A Study of Replacement Algorithms for a Virtual-Storage Computer," *IBM Systems journal*, 1966.

[9] C. Villani, *Optimal Transport: Old and New*. Springer Science & Business Media, 2008.

[10] R. Kumar and S. Vassilvitskii, "Generalized Distances between Rankings," in *Proc. of ACM WWW*, 2010.

[11] J. R. Bitner, "Heuristics that Dynamically Organize Data Structures," *SIAM Journal on Computing*, 1979.

[12] J. A. Fill, "An Exact Formula for the Move-to-Front Rule for Self-Organizing Lists," *Journal of Theoretical Probability*, 1996.

[13] W. F. King-III, "Analysis of Demanding Paging Algorithms," in *Proc. of IFIP Congress*, 1971.

[14] E. Gelenbe, "A Unified Approach to the Evaluation of a Class of Replacement Algorithms," *Computers, IEEE Transactions on*, 1973.

[15] E. J. Rosensweig, J. Kurose, and D. Towsley, "Approximate Models for General Cache Networks," in *Proc. of INFOCOM*, 2010.

[16] R. Fagin, "Asymptotic Miss Ratios over Independent References," *Journal of Computer and System Sciences*, vol. 14, no. 2, pp. 222–250, 1977.

[17] H. Che, Y. Tung, and Z. Wang, "Hierarchical Web Caching Systems: Modeling, Design and Experimental Results," *Selected Areas in Communications, IEEE Journal on*, Sep 2002.

[18] D. S. Berger, P. Gland, S. Singla, and F. Ciucu, "Exact Analysis of TTL Cache Networks," *Performance Evaluation*, 2014.

[19] S. Basu, A. Sundarrajan, J. Ghaderi, S. Shakkottai, and R. Sitaraman, "Adaptive TTL-Based Caching for Content Delivery," in *Proc. of Sigmetrics*, 2017.

[20] J. Li, S. Shakkottai, J. C. S. Lui, and V. Subramanian, "Accurate Learning or Fast Mixing? Dynamic Adaptability of Caching Algorithms," *Arxiv preprint arXiv:1701.02214*, 2017.

[21] R. Fagin, R. Kumar, and D. Sivakumar, "Comparing Top k Lists," *SIAM Journal on Discrete Mathematics*, 2003.

[22] D. A. Levin, Y. Peres, and E. L. Wilmer, *Markov Chains and Mixing Times*. American Mathematical Soc., 2009.

[23] R. R. Montenegro and P. Tetali, *Mathematical Aspects of Mixing Times in Markov Chains*. Now Publishers Inc, 2006.

[24] F. Chung, "Laplacians and the Cheeger Inequality for Directed Graphs," *Annals of Combinatorics*, 2005.

[25] M. Mihail, "Conductance and Convergence of Markov Chains-A Combinatorial Treatment of Expanders," in *Proc. of IEEE FOCS*, 1989.

[26] A. Sinclair, "Improved Bounds for Mixing Rates of Markov Chains and Multicommodity Flow," *Combinatorics, probability and Computing*, 1992.

[27] J. A. Fill, "Eigenvalue bounds on convergence to stationarity for nonreversible markov chains, with an application to the exclusion process," *The annals of applied probability*, pp. 62–87, 1991.

[28] O. I. Aven, E. G. Coffman, and I. A. Kogan, *Stochastic Analysis of Computer Storage*. Springer Science & Business Media, 1987.

[29] W. Hendricks, "An Account of Self-organizing Systems," *SIAM Journal on Computing*, pp. 715–723, 1976.

[30] F. P. Kelly, *Reversibility and Stochastic Networks*. Cambridge University Press, 2011.

[31] J. H. Hester and D. S. Hirschberg, "Self-organizing Linear Search," *ACM Computing Surveys (CSUR)*, 1985.

[32] R. Phatarfod, "On the Transition Probabilities of the Move-to-Front Scheme," *Journal of applied probability*, 1994.

[33] D. Tang and V. Subramanian, "Eigenvalues of LRU via a linear algebraic approach," *Operations Research Letters*, 2018.

[34] G. Bianchi, A. Detti, A. Caponi, and N. Blefari Melazzi, "Check before Storing: What is the Performance Price of Content Integrity Verification in LRU Caching?" *ACM SIGCOMM Computer Communication Review*, 2013.


**Jian Li** (S'16-M'17) received the Bachelor of Engineering degree in electrical engineering from Shanghai Jiao Tong University, Shanghai, China, in 2012, and his Ph.D. degree in computer engineering from Texas A&M University, College Station, TX, USA, in 2016. He is currently a postdoctoral research associate with College of Information and Computer Sciences, University of Massachusetts Amherst, MA, USA. His research interests include modeling, analysis and algorithm design of social and computer networks; data science and large-scale data analytics; Internet of Things; optimization in large-scale systems; online learning and online algorithm design; game theory and network economics.

**Srinivas Shakkottai** (S'00-M'08-SM'15) received the Bachelor of Engineering degree in electronics and communication engineering from Bangalore University, Bangalore, India, in 2001, and his M.S. and Ph.D. degrees in electrical engineering from the University of Illinois at Urbana-Champaign, Urbana, IL, USA, in 2003 and 2007, respectively. He was a Postdoctoral Scholar with Stanford University, Stanford, CA, USA in 2007. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX, USA. His research interests include content distribution systems, pricing approaches to network resource allocation, game theory, congestion control, and the measurement and analysis of Internet data.

He is the recipient of the Defense Threat reduction Agency Young Investigator Award (2009) and the NSF Career Award (2012), as well as research awards from Cisco (2008) and Google (2010). He also received an Outstanding Professor Award (2013) and was selected as a TEES Select Young Faculty Fellow (2014) at Texas A&M University.

**John C. S. Lui** received the Ph.D. degree in computer science from UCLA. He is currently the Choh-Ming Li Chair Professor in the Department of Computer Science and Engineering, The Chinese University of Hong Kong. His current research interests include communication networks, network/system security, network economics, network sciences, cloud computing, large-scale distributed systems, and performance evaluation theory. He serves on the editorial boards of the IEEE/ACM Transactions on Networking, the IEEE Transactions on Computers, the IEEE Transactions on Parallel and Distributed Systems, the Journal of Performance Evaluation, and the International Journal of Network Security. He was the chairman of the CSE Department from 2005-2011. His personal interests include films and general reading. He received various departmental teaching awards and the CUHK Vice-Chancellor?s Exemplary Teaching Award. John is a co-recipient of the best paper award in the IFIP WG 7.3 Performance 2005, IEEE/IFIP NOMS 2006, SIMPLEX 2013, and ACM RecSys 2017. He is an elected member of the IFIP WG 7.3, Fellow of ACM, Fellow of IEEE, Senior Research Fellow of the Croucher Foundation and was the past chair of the ACM SIGMETRICS (2011-2015).

**Vijay Subramanian** is an Associate Professor in the EECS Department at the Univ. of Michigan, Ann Arbor. He got his Ph.D. from the ECE Department at UIUC in 1999. Thereafter, he spent a few years as a researcher at Motorola Inc. and the Hamilton Insitute in Maynooth, Ireland, and a research faculty in the EECS Department at Northwestern University before moving to the Univ. of Michigan in Fall 2014. His research interests are in stochastic analysis, random graphs, game theory and mechanism design with applications to social, economic and technological networks.