4-1-2021

The Design of Dynamic Probabilistic Caching with Time-Varying Content Popularity

Jie Gao
University of Waterloo

Shan Zhang
Beihang University

Lian Zhao
Ryerson University

Xuemin Shen
University of Waterloo

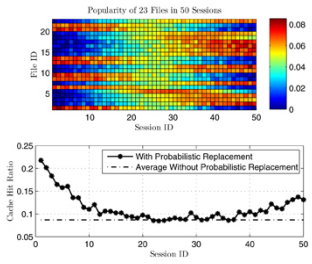
IEEE Transactions on Mobile Computing, Vol. 20, No. 4 (April 1, 2021): 1672-1684. DOI.

The Design of Dynamic Probabilistic Caching with Time-Varying Content Popularity

Jie Gao
Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada
Shan Zhang
School of Computer Science and Engineering, Beihang Univesity, Beijing, China
State Key Laboratory of Software Development Environment, Beijing, China
Beijing Key Laboratory of Computer Networks, Beijing, China
Lian Zhao
Department of Electrical, Computer, and Biomedical Engineering, Ryerson University, Toronto, ON, Canada
Xuemin Shen
Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada

Abstract:

In this paper, we design dynamic probabilistic caching for the scenario when the instantaneous content popularity may vary with time while it is possible to predict the average content popularity over a time window. Based on the average content popularity, optimal content caching probabilities can be found, e.g., from solving optimization problems, and existing results in the literature can implement the optimal caching probabilities via static content placement. The objective of this work is to design dynamic probabilistic caching that: i) converge (in distribution) to the optimal content caching probabilities under time-invariant content popularity, and ii) adapt to the time-varying instantaneous content popularity under time-varying content popularity. Achieving the above objective requires a novel design of dynamic content replacement because static caching cannot adapt to varying content popularity while classic dynamic replacement policies, such as LRU, cannot converge to target caching probabilities (as they do not exploit any content popularity information). We model the design of dynamic probabilistic replacement policy as the problem of finding the state transition probability matrix of a Markov chain and propose a method to generate and refine the transition probability matrix. Extensive numerical results are provided to validate the effectiveness of the proposed design.



SECTION 1 Introduction

Data traffic volume in cellular networks has experienced a tremendous growth since the deployment of the LTE, and a major portion of the traffic is related to content delivery [1]. This feature is expected to be even more prominent in the next generation cellular network [2]. In this background, mobile edge caching (MEC) has attracted increasing research attention [3], [4], [5]. By placing selected contents in the cache, a base station (BS) can resolve a part of content requests locally without fetching the requested content over the backhaul. This benefits both the network and the users by alleviating the pressure on the backhaul and reducing the end-to-end delay in content delivery, respectively.

In practice, caches deployed in a network have limited sizes and can only accommodate selected contents. The problem of selecting the contents to be stored at available caches is referred to as *content placement*. In the case of static caching, the cached contents will not change once the content placement problem is solved. By contrast, the cached content can be updated, e.g., as content request and download status changes, in the case of dynamic caching. The problem of selecting new content to cache while replacing existing contents is referred to as *content replacement*. The term caching can include content placement, content replacement, or both. Various principles and approaches for caching can be found in the literature [6], [7], [8], [9], [10], [11].

When the content popularity is unknown, dynamic caching is usually adopted to allow adjustments to the content placement based on the content request and download status. With heuristic principles such as the least-frequently-used (LFU) or least-recently-used (LRU), a less popular content is replaced when a new content is accepted in classic dynamic caching. Probabilistic dynamic caching, which originates from computer networks, implements dynamic caching probabilistically. The idea is that a user-requested content is cached with a certain probability [12], [13]. Because of the dynamic cache-and-replace process, dynamic probabilistic caching may adapt to time-varying content popularity without knowing the popularity. Moreover, dynamic probabilistic caching can achieve fair and efficient content placement in a network with low redundancy in a distributed

setting [12], [14]. However, it is difficult to establish a bound on the performance of probabilistic caching, which is typically evaluated numerically.

When the content popularity is known, the optimal content placement can be calculated, and static caching based on the optimal placement is usually adopted. A common approach for content placement in such case is for caches at different locations in a network to cooperate and find a joint optimal caching solution. For example, Applegate *et al.* studied the placement of video-on-demand contents considering an arbitrary network topology and time-varying content demand [15]. Using relaxation and approximation methods [16], the authors found a near-optimal solution that can serve all requests with significantly less bandwidth consumption when compared to LRU/LFU. Common objectives for joint optimal caching include maximizing the cache hit probability [17], maximizing the caching capacity of a network [18], and minimizing the content provisioning delay [19], [20] or cost [21]. Alternatively, the problem of content placement can also be formulated as a non-cooperative game [22] or an auction [23] between cache servers, content providers, and/or network operators, and the solution can be found from the resulting equilibrium or outcome in a decentralized manner [24], [25]. Probabilistic content placement was adopted in many recent works in the literature of optimization-based MEC with known content popularity [26], [27], [28], [29], [30]. Chen *et al.* analyzed the performance of probabilistic content placement at small base stations (SBSs) and found the optimal joint caching probability to maximize the successful download probability [26]. Li *et al.* studied the joint optimization of caching probabilities for maximizing the successful delivery probability in the scenario of an N-tier heterogeneous network [27]. In [28], SBSs cooperate and form clusters, and an efficient solution of content placement to reduce the cooperative strategy was proposed. Liu and Yang studied the problem of optimal probabilistic content placement policy to maximize the area spectral efficiency in a heterogeneous network and analyzed the impact of transmission power, node density, and rate requirements on the result [29]. The optimal tier-level content placement with probabilistic content placement for maximizing the cache hit probability, in which BSs at the same tier are assigned the same caching probabilities, was derived in [30]. However, most of these works do not consider dynamic content replacement as it becomes unnecessary when the exact content popularity is known.

It is well known that content popularity can be subject to variations over time [31], [32]. For example, Traverso *et al.* proposed a shot noise model to capture the temporal locality in content popularity in [33], while Leonardi and Torrisi analyzed LRU under the shot noise model [34]. Garetto *et al.* proposed a unified approach to analyze caching policies, taking the temporal locality into account [35]. Given potential variations in content popularity over time, assuming the knowledge of instantaneous content popularity might not always be practical. However, the average content popularity over a time window may be obtained, e.g., from prediction [36].[1] In such case, the knowledge of the average content popularity can be used to derive the target optimal overall content caching probabilities. It is not difficult to implement the target caching probabilities using static caching. Alternatively, one could use dynamic caching such as LRU. The former option maximizes the average cache hit ratio (among all static caching) but cannot adapt to the time-varying content popularity. The latter, by contrast, may cope with variations in the instantaneous content popularity but generally does not exploit the average content popularity information. Depending on the content request statistics, either static caching based on the target caching probabilities or dynamic caching can be the better option. The objective of this paper is to design dynamic probabilistic caching that inherit advantages from both options. Specifically, we aim at such dynamic probabilistic caching that can converge (in distribution) to target optimal content caching probabilities if the content popularity is time-invariant and can adapt to the time-varying instantaneous content popularity otherwise. To do that, we exploit the average content popularity information while designing the content replacement policy. The contributions of this work are as follows.

First, we propose the idea of dynamic probabilistic caching based on average content popularity and demonstrate that it can outperform both the static caching and classic dynamic caching such as LRU. The

proposed dynamic probabilistic caching can adapt to the time-varying instantaneous content popularity and thereby increase the cache hit ratio when compared to the optimal static caching in the case of time-varying instantaneous content popularity. In addition, unlike classic dynamic caching, the proposed design can converge (in distribution) to a target set of optimal content caching probabilities in the case of time-invariant content popularity.

Second, we establish a connection between a set of caching probabilities and a probabilistic content placement policy. We show that the probabilistic content placement policy that can implement a given set of caching probabilities can be non-unique in most cases and derive a general-form solution of probabilistic content placement policies, which includes existing solutions, e.g., the one in [37], as special cases. Moreover, we show that different probabilistic content placement policies are not equivalent when implemented in dynamic caching.

Third, we formulate the problem of designing a probabilistic content replacement policy based on average content popularity into an equivalent problem of designing the state transition probability matrix of a Markov chain and propose a method to solve the latter problem. In classic dynamic caching such as LRU, the content request statistics determine the state transitions. However, our design builds the state transition probability matrix by exploiting the average content popularity information. The proposed method finds the state transition probability matrix so that the underlying Markov chain is irreducible and ergodic, and its unique steady state implements the target probabilistic content placement policy if the instantaneous content popularity converges to the average content popularity.

SECTION 2 System Model

Consider an edge caching system (Fig. 1 shows an example), where each SBS has a cache. Similar scenarios of MEC can be found in the literature, e.g., [38], [39], [40]. Here we focus on the dynamic caching strategy of each individual cache. Consider a target cache with size c and assume there are N_{f} contents in total. Contents have an identical length (normalized to 1), and the set of contents is denoted by $\mathcal{F}$. The instantaneous content request probability at the target cache can be varying with time and hard to track. However, we assume that it is possible to predict the average probability of the cache receiving a request for content $k \in \mathcal{F}$ over a predefined time window and denote this probability as φ_k[2]. We do not consider the issue of user mobility since the impact of user mobility is reflected through the average content popularity at the target cache while our focus is not on the popularity prediction. For research on mobility management in MEC, interested readers can check references [41], [42], [43].

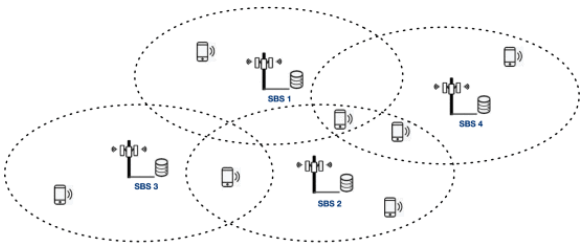


Fig. 1. An illustration of an edge caching system.

As probabilistic caching is considered, contents can be cached with probabilities. Denote the probability that content k is stored at the target cache as p_k. The cache size limit requires that

$$\sum_{\forall k \in \mathcal{F}} p_k \leq c\,.$$

(1)

To maximize its cache hit ratio, an SBS would use all available cache spaces. In the sequel, we assume that all cache space is used (which implies that using cache space does not incur a cost). Depending on the specific scenario and objective, the caches in the system may find their optimal caching probabilities either locally or jointly through an optimization problem or a game using the knowledge of $\{\varphi_k\}_{k\in\mathcal{F}}$. Denote the optimal caching probabilities of the target cache by $\{p_k^\star\}_{\forall k}$.

In our model, we focus on the caching strategy after $\{p_k^\star\}_{\forall k}$ are obtained instead of finding $\{p_k^\star\}_{\forall k}$. After knowing $\{p_k^\star\}_{\forall k}$, the target cache could implement $\{p_k^\star\}_{\forall k}$ using static caching. Alternatively, the target cache could also use dynamic caching such as LRU. The former maximizes the average cache hit ratio among all static caching in the predefined time window, while the latter may cope with time-varying content popularity. We aim to design dynamic caching that can implement $\{p_k^\star\}_{\forall k}$ when the content popularity is time-invariant and can adapt when the content popularity is time-varying.

TABLE 1 List of Symbols

c	Cache size of the target SBS
$\mathcal{F}$	Set of all contents
N_f	Number of all contents
φ_k	Probability that content k is requested
$\mathbf{s}^m$	The mth cache state, represented by a $N_\mathrm{f} \times 1$ vector
$\mathbf{S}$	The cache state matrix
η^l	The probability of the cache in state $\mathbf{s}^l$
η	The state caching probability vector
η^*	The target (optimal) state caching probability vector
$\mathcal{G}^m$	The set of contents cached by cache state $\mathbf{s}^m$
$\mathcal{H}_m$	The neighbor states of state m
$\mathcal{H}_m^k$	The neighbor states of state m linked by content k
p_k	The probability of caching content k
$\mathbf{P}$	Content caching probability vector, i.e., $[p_1^*, \dots, p_{N_\mathrm{f}}]$
$\mathbf{P}^*$	The target (optimal) content caching probability vector
$\tau_{m',m}$	The conditional probability that the content in $\mathcal{G}^{m'} - \mathcal{G}^m$ replaces that in $\mathcal{G}^m \quad - \mathcal{G}^{m'}$
S_l	The lth ordered sequence of states
$B(l)$	The branch point of sequence l
$M(l)$	The merge point of sequence l

Given the above objective, the next two sections formulate static caching in terms of probabilistic content placement policy and dynamic caching in terms of probabilistic content replacement, respectively. A list of symbols used in this paper is given in Table 1.

SECTION 3 Probabilistic Content Placement

A cache accommodates a combination of contents, subject to its size limit, at any given time. Therefore, in probabilistic caching, it is necessary for a cache to map a given set of content caching probabilities into specific combinations of contents and the corresponding probabilities of caching these combinations. In this section, we first define cache state and then investigate static probabilistic caching by studying the connection between content caching probabilities and state caching probabilities.

3.1 Cache States

A cache state can be represented by a 0-1 vector: the kth element is 1 if content k is cached and 0 otherwise. Given the cache limit c, the cache has $n = \binom{N_\mathrm{f}}{c}$ different states that cache exactly c files. Denote the mth state of the cache and the set of contents cached by the mth state by $\mathbf{s}^m$ and $\mathcal{G}^m$, respectively. It follows that $\mathbf{s}^m = [s_1^m, \dots, s_{N_\mathrm{f}}^m]^\mathrm{T}$ is a $N_\mathrm{f} \times 1$ vector, where the superscript $(\cdot)^\mathrm{T}$ denotes the transpose. In the vector $\mathbf{s}^m$, element s_k^m is 1 if $k \in \mathcal{G}^m$ and 0 if $k \notin \mathcal{G}^m$. We can organize all state vectors into an $N_\mathrm{f} \times n$ cache state matrix $\mathbf{S}$, which is defined as $\mathbf{S} = [\mathbf{s}^1, \dots, \mathbf{s}^n]$.

3.2 State Caching Probability

State caching probabilities are the probabilities assigned to the cache states. Denote the probability of cache state $\mathbf{s}^l$ as η^l. It follows that $\sum_{l=1}^{n} \eta^l = 1$. The state caching probabilities $\{\eta^l\}_{\forall l}$ and the content caching probabilities $\{p_k\}_{\forall k}$ must satisfy the following conditions

$$\eta^l = 0, \forall l | \mathrm{s}_k^l = 0, \text{if } p_k = 1,$$

(2a)

$$\eta^l = 0, \forall l | \mathrm{s}_k^l = 1, \text{if } p_k = 0.$$

(2b)

It can be seen that the number of cache states with nonzero state caching probabilities reduces for each content k such that $p_k = 1$ or $p_k = 0$. The state caching probabilities can be organized into an $n \times 1$ state caching probability vector $\boldsymbol{\eta}$, as follows:

$$\boldsymbol{\eta} = [\eta^1, \dots, \eta^n]^\mathrm{T}.$$

(3)

3.3 Static Probabilistic Caching

Implementing a target set of content caching probabilities, e.g., $\{p_k^\star\}_{\forall k}$, is equivalent to determining the vector η. This is because η specifies the cache states and the corresponding state caching probabilities. Define the content caching probability vector $\mathbf{p} = [p_1, \dots, p_{N_\mathrm{f}}]$. The content caching probability vector $\mathbf{p}$, state caching probability vector $\boldsymbol{\eta}$, and the cache state matrix $\mathbf{S}$ are connected through the following equation:

$$\mathbf{S}\boldsymbol{\eta} = \mathbf{p},$$

(4)

where $\mathbf{S}$, $\boldsymbol{\eta}$, $\boldsymbol{p}$ are of size $N_\mathrm{f} \times n$, $n \times 1$, and $N_\mathrm{f} \times 1$, respectively.

Lemma 1.
A given set of caching probabilities $\{p_k\}_{\forall k}$ could be implemented by more than one state caching probability vector $\boldsymbol{\eta}$.

The proof of Lemma 1 is straightforward. Since $n = \binom{N_\mathrm{f}}{c} \geq N_\mathrm{f}$, it can be seen that Eq. (4) is in general an under-determined system that may have more than one solution of $\boldsymbol{\eta}$, corresponding to different probabilistic content placement policies.

Note that Eq. (4) incorporates the requirement that $\sum_l \eta^l = 1$. This can be seen by multiplying an all-one vector of size $1 \times N_\mathrm{f}$ from the left to both sides of Eq. (4).

To implement a target set of content caching probabilities $\{p_k^\star\}_{\forall k}$ (equivalently, the content caching probability vector $\mathbf{p}^\star$) using static caching, the method is to randomly draw a cache state based on $\boldsymbol{\eta}^\star$. Once the cache state is drawn, there will be no change of cache state (i.e., no content replacement). As long as $\mathbf{S}\boldsymbol{\eta}^\star = \mathbf{p}^\star$, the random draw of the cache state implements the target content caching probabilities. Lemma 1 suggest that, the static probabilistic caching that can implement a target set of content caching probabilities is non-unique.

Lemma 2.
The cache state matrix $\mathbf{S}$ has the following two properties:

- $\mathbf{S}$ has full row rank;
- the minimum singular value of $\mathbf{S}$ is no less than $\sqrt{c}$.

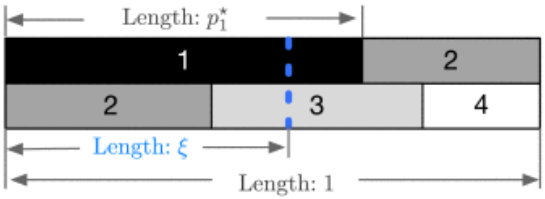
Proof.
See Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/TMC.2020.2967038.

Lemma 2 suggests that the matrix $\mathbf{S}\mathbf{S}^\mathrm{T}$ has full rank. Therefore, given a set of optimal caching probabilities $\mathbf{p}^\star$, the solution to Eq. (4) can be written as:

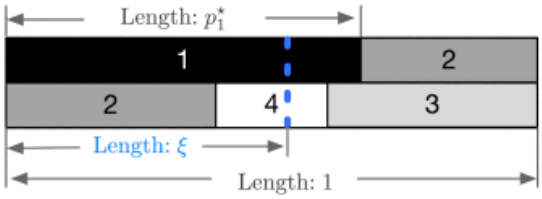
$$\boldsymbol{\eta}^\star = \mathbf{S}^\mathrm{T}(\mathbf{S}\mathbf{S}^\mathrm{T})^{-1}\mathbf{p}^\star + \mathbf{v}, \tag{5}$$

where v can be any vector in the null space of $\mathbf{S}$ that renders $\boldsymbol{\eta}^\star$ a valid probability vector, i.e., $\mathbf{0} \preceq \boldsymbol{\eta}^\star \preceq \mathbf{1}$ and $\mathbf{1}^\mathrm{T}\boldsymbol{\eta}^\star = 1$. Here, the inequality sign $\preceq$ represents element-wise comparison, i.e., $\mathbf{x} \preceq \mathbf{y}$ if $x_i \leq y_i, \forall i$ (in which x_i represents the ith element of vector $\mathbf{x}$). The solution in Eq. (5) includes all possible static probabilistic caching that can implement $\mathbf{p}^\star$.

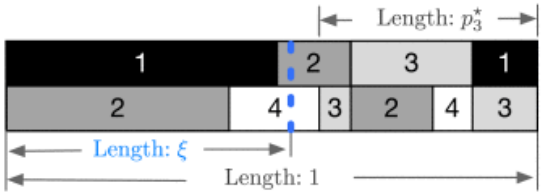
A heuristic method for implementing the given $\mathbf{p}^\star$ is introduced in [37]. Specifically, one can generate c (i.e., the cache size) rows, all with length 1, and then a block for each content, where the length of block k is equal to $p_k^\star$. Next, the blocks are placed into the rows one by one. If the vacant part of a row is not long enough for a block, the rest of this block continues from the beginning of the next row. Fig. 2a illustrates the above procedure using a total of 5 contents and a cache of size 2 as an example. In this example, the given caching probabilities for contents 1 to 4 are nonzero while the given caching probability for content 5 is zero. We represent the blocks for content 1 to content 4 using decreasing shades. After placing all blocks, the next step is to generate a random number ξ in [0,1] based on uniform distribution and draw a vertical line (the dashed line in Fig. 2a) at the corresponding location. Then, the blocks that the vertical line crosses indicate the contents to be cached to implement $\mathbf{p}^\star$. For example, in the case of Fig. 2a, contents 1 and 3 will be cached.



(a) Choosing a cache state probabilistically based on the method proposed in [37]. The result shown corresponds to one solution of $\boldsymbol{\eta}$ in eq. (5).



(b) An alternative result, which correpsonds to a different solution of $\boldsymbol{\eta}$ in eq. (5).



(c) A third result.

Fig. 2. Generating cache state probabilistically given $\mathbf{p}^\star$ in static probabilistic caching.

Although the aforementioned method does not explicitly solve $\boldsymbol{\eta}$ from the given $\mathbf{p}^\star$, the probability of a chosen cache state corresponds to one solution of $\boldsymbol{\eta}^\star$. Specifically, if the dashed vertical line in Fig. 2a sweeps across the rows from one side to the other under a constant speed, the portion of time that the vertical line crosses state l is equal to η^l. However, the solution in Eq. (5) is non-unique in general, and probabilistic caching based on the aforementioned method corresponds to only one solution of $\boldsymbol{\eta}^\star$. If we rearrange the blocks or divide the blocks into pieces, the result may correspond to different solutions of $\boldsymbol{\eta}^\star$. Figs. 2b and 2c give two examples.

As subtle as the aforementioned heuristic method is, it can be more complicated than finding a solution using Eq. (5) when N_f is large. More importantly, it is worth noting that, while the results in Figs. 2a, 2b, and 2c are equivalent in the case of static caching, different solutions of $\boldsymbol{\eta}$ may not be equivalent in the case of dynamic caching with content replacement. Consider the same example with 5 contents and a cache size of 2. Suppose that the optimal caching probabilities for the 5 contents found from optimization are given by $\mathbf{p}^\star = [0.9,0.7,0.3,0.1,0]^\mathrm{T}$. The matrix $\mathbf{S}$ is given as follows:

$$\mathbf{S} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

Then, it can be shown that

$$\boldsymbol{\eta}_1 = [0.6,0.2,0.1,0,0.1,0,0,0,0,0]^{\mathrm{T}},$$

which corresponds to the case similar to Fig. 2b and

$$\boldsymbol{\eta}_2 = [0.62,0.22,0.06,0,0.06,0.02,0,0.02,0,0]^{\mathrm{T}},$$

which corresponds to the case similar to Fig. 2c, can both implement $\mathbf{p}^{\star} = [0.9,0.7,0.3,0.1,0]^{\mathrm{T}}$. Therefore, for static caching, content placement generated from $\boldsymbol{\eta}_1$ and $\boldsymbol{\eta}_2$ are equivalent. However, for dynamic caching, dynamic content replacement based on $\boldsymbol{\eta}_1$ can lead to a less number of cache replacements than that based on $\boldsymbol{\eta}_2$. This is because $\boldsymbol{\eta}_1$ has less nonzero elements, corresponding to fewer cache states. For example, dynamic implementation of $\mathbf{p}^{\star}$ based on $\boldsymbol{\eta}_2$ may cause a replacement of content 3 by content 4 when the cached contents are $\{2,3\}$ and a replacement of content 1 by content 3 when the cached contents are $\{1,4\}$. However, such replacements will not happen if dynamic caching based on $\boldsymbol{\eta}_1$ is used because $\boldsymbol{\eta}_1$ assigns zero probability to the states that cache $\{2,4\}$ and $\{3,4\}$.

Note that the choice of $\boldsymbol{\eta}^{\star}$ based on the optimal content caching probability $\mathbf{p}^{\star}$ can determine the initial content placement. Accordingly, the initial contents stored in the target cache at the beginning of the considered time window are pre-fetched based on $\boldsymbol{\eta}^{\star}$, which occurs at the beginning of the time window. Then, for dynamic caching, the target cache updates its cached contents as it receives content requests, as studied in the next section.

SECTION 4 Probabilistic Content Replacement: the Markov Chain of Content Replacement

In classic dynamic caching such as LRU, replacements are usually determined solely by the content request statistics. Such replacements provide adaptivity to time-varying content popularity without any *a priori* information. However, in our considered scenario, in which average content popularity $\{\varphi_k\}_{k\in\mathcal{F}}$ and target content caching probabilities $\{p_k^{\star}\}_{\forall k}$ are available, using classic dynamic caching while neglecting the average content popularity can lead to a waste of useful information. Therefore, we aim to design dynamic probabilistic caching that can exploit the average content popularity $\{\varphi_k\}_{k\in\mathcal{F}}$ to adapt to the content popularity. A logical requirement is that the resulting caching probability should converge to the optimal content placement policy $\boldsymbol{\eta}^{\star}$ based on the average content popularity in the case of time-invariant content popularity. This section addresses two questions regarding the design of such dynamic caching: a) what should be the probability of accepting a new content into the cache? and b) which existing content should be replaced, and with what probability, if the new content is accepted. In the rest of this section, we assume that the target cache makes replacement decisions locally, i.e., without needing to check with other caches.

4.1 The Content Replacement Markov Chain

The content replacement process at the target cache can be modeled using a Markov chain. Note that the idea of using Markov chain and related tools in designing content replacement policy can be found in the literature. For example, using a Markov decision process, Bahat and Makowski proved that the optimal content replacement policy is a Markov stationary policy under the independent reference model [44]. In their recent work, Shukla and Abouzeid modeled content replacement using a Markov decision process and found the optimal content retention time to jointly minimize content retrieval delay and cache wearout [45]. Our focus here, however, is the design of content replacement that can converge to a target stationary point in the case when the instantaneous content popularity is time-invariant and adapt to the variations in the case when the instantaneous content popularity is time-varying.

In the content replacement Markov chain, a state transition may happen when a requested content is not in the cache. For state m, denote the set of states that state m can transit into by replacing one cached content with content k and the entire set of states that state m can transit into by replacing one cached content with any other content as $\mathcal{H}_m^k$ and $\mathcal{H}_m$, respectively. It is not difficult to see that state m and any state in $\mathcal{H}_m$ must differ in one and only one cached content. Moreover, the following results hold:

$$\mathcal{H}_m = \bigcup_{k \in \mathcal{F} \setminus \mathcal{G}^m} \mathcal{H}_m^k, |\mathcal{H}_m^k| = c, |\mathcal{H}_m| = c(N_\mathrm{f} - c) \ .$$

(6)

Consider a pair of neighbor states m and $m' \in \mathcal{H}_m^k$. The question that arises is: with what probability should the cache transition into state m' given that content k is requested while the cache is currently in state m? Denote this conditional probability as $\tau_{m',m}$. Then, the design of dynamic probabilistic caching boils down to finding $\tau_{m',m}$ for every m and $m' \in \mathcal{H}_m^k$ where $k \in \mathcal{F} \setminus \mathcal{G}^m$.

The overall state transition probability matrix of the content replacement Markov chain is determined by two sets of probabilities: the content request probabilities and $\{\tau_{m',m}\}$. Since the instantaneous content request probabilities are unknown, we can only exploit the average content request probabilities $\{\varphi_k\}_{k \in \mathcal{F}}$. Therefore, the design of dynamic content replacement uses $\{\varphi_k\}_{k \in \mathcal{F}}$ instead of the instantaneous content request probabilities. A discussion on the effect of time-varying content popularity will be given later.

Denote the overall state transition probability matrix by $\mathbf{\Theta}$. The element at the mth column and the m'th row of $\mathbf{\Theta}$ represents the probability that state m transitions into state m'. Then, $\mathbf{\Theta}$ can be written as the summation of content-specific conditional state transition matrices as follows:

$$\mathbf{\Theta} = \sum_{k \in \mathcal{F}} \varphi_k \mathbf{\Theta}_k \ .$$

(7)

where $\mathbf{\Theta}_k$ is the conditional state transition probability matrix given that content k is being requested. It can be seen that $\mathbf{\Theta}$ and $\mathbf{\Theta}_k, \forall k$ have many zero elements because $\mathbf{\Theta}(m, m')$ and $\mathbf{\Theta}(m', m)$ are both zero if $m' \notin \mathcal{H}_m$, and $\mathbf{\Theta}_k(m, m')$ and $\mathbf{\Theta}_k(m', m)$ are both zero if $m' \notin \mathcal{H}_m^k$. Specifically, based on the conditions in Eq. (6), each column of $\mathbf{\Theta}$ only has $c(N_\mathrm{f} - c) + 1$ nonzero elements with one on and the rest off the main diagonal. For a column in $\mathbf{\Theta}_k$, two cases are possible. If state m caches content k, then the diagonal element $\mathbf{\Theta}_k(m, m)$ is the only nonzero element in the mth column. Otherwise, the mth column has c nonzero elements off the main diagonal.

In order to highlight the state transition, the content request probability φ_k at the target cache is alternatively denoted by $\varphi_{m',m}$ if $m' \in \mathcal{H}_m^k$, i.e., if content k must be requested for the cache to transition from state m into state m'. Denote the mth element on the main diagonal of $\mathbf{\Theta}$ by $\alpha_{m,m}$. Fig. 3 illustrates the state transitions. In the illustrated example, $N_\mathrm{f} = 5$ and $c = 2$, and therefore $c(N_\mathrm{f} - c) + 1 = 7$. As a result, each state can transition into itself and 6 other states.

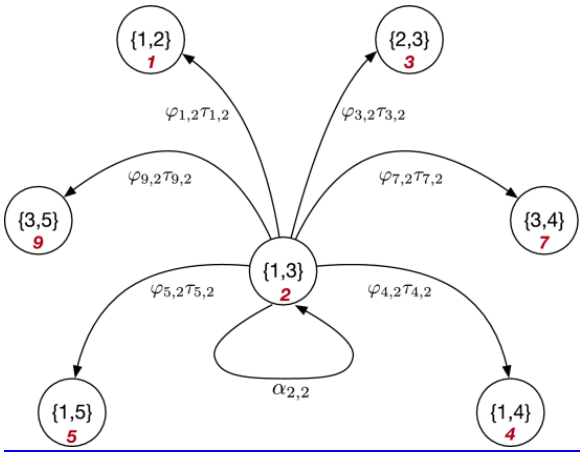


Fig. 3. An illustration of the variables in the state transition using the example of state 2, where $N_{\mathrm{f}} = 5$ and $c = 2$. Each circle represents a state, the bracketed numbers in the center represent the cached contents in that state, and the italic number at the bottom represents the state ID.

Given the above notations, the overall transition probability matrix $\boldsymbol{\Theta}$ corresponding to the content replacement Markov chain for the target cache can be written as:

$$\boldsymbol{\Theta} = \begin{array}{c} \\ 1 \\ \vdots \\ m \\ \vdots \\ n \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & \cdots & m & \cdots & n \end{array} \\ \begin{bmatrix} \alpha_{1,1} & \cdots & \varphi_{1,m}\tau_{1,m} & \cdots & \varphi_{1,n}\tau_{1,n} \\ \vdots & \ddots & \vdots & & \vdots \\ \varphi_{m,1}\tau_{m,1} & & \alpha_{m,m} & & \varphi_{m,n}\tau_{m,n} \\ \vdots & & \vdots & \ddots & \vdots \\ \varphi_{n,1}\tau_{n,1} & \cdots & \varphi_{n,m}\tau_{n,m} & \cdots & \alpha_{n,n} \end{bmatrix} \end{array},$$

(8)

where $\varphi_{m',m} \in [0,1]$ and $\tau_{m',m} \in [0,1], \forall m' \in \mathcal{H}_m, \forall m$. The diagonal element $\alpha_{m,m}$ in (8) can be found as follows:

$$\alpha_{m,m} = \sum_{k \in \mathcal{G}^m} \varphi_k + \sum_{m' \in \mathcal{H}_m} \varphi_{m',m}(1 - \tau_{m',m}) .$$

(9)

The first item in Eq. (9) represents the probability that a content currently in the cache is requested, while the second item represents the probability that a content not in the cache is requested (and downloaded) but not accepted into the cached (i.e., no replacement occurred).

In Section 3, we have investigated the problem of implementing target content caching probabilities $\mathbf{p}^\star$ by finding the corresponding state caching probabilities $\boldsymbol{\eta}^\star$. Given $\boldsymbol{\eta}^\star$, the design of dynamic probabilistic caching so that the content caching probabilities converge to $\mathbf{p}^\star$ in the case of time-invariant content popularity is equivalent to finding the transition matrix $\boldsymbol{\Theta}$ such that

$$\boldsymbol{\Theta}\boldsymbol{\eta}^\star = \boldsymbol{\eta}^\star .$$

(10)

If the elements of $\boldsymbol{\Theta}$ could be arbitrarily chosen in the range of [0,1], the problem can be solved with existing methods, e.g., the Metropolis-Hastings Algorithm [46]. However, as can be seen from Eq. (8), there are additional constraints on the elements of $\boldsymbol{\Theta}$. First, each off-diagonal element is a product of two items, and the (m', m) th element should be bounded by $\varphi_{m',m}$. Second, the summation of multiple elements in the same column should also be bounded, i.e.,

$$\sum_{\forall m' \in \mathcal{H}_m^k} \varphi_{m',m} \tau_{m',m} \leq \varphi_k, \forall k \in \mathcal{G}^m, \forall m\,.$$

(11)

Consequently, the Metropolis-Hastings Algorithm cannot be applied to the considered problem. In the next section, an approach is proposed to construct an irreducible and ergodic Markov chain by designing the matrix $\boldsymbol{\Theta}$ for the target cache so that $\boldsymbol{\eta}^\star$ is the unique steady state.

4.2 Generating the State Transition Matrix $\boldsymbol{\Theta}$

Without loss of generality, it is assumed that the elements of $\boldsymbol{\eta}^\star$ are non-zero and arranged in a non-increasing order, i.e., $\eta^q \geq \eta^l$ if $q < l$. An extension to the case when η^j is zero for some j is straightforward.

When a content not in the cache is requested, a replacement may or may not happen. In order to control this factor in our design, we introduce parameters $\{\omega_{k,m',m}\}$ to represent the upper-limit on state transition probabilities. Specifically, for any given m and $m' \in \mathcal{H}_m^k$, the parameter $\omega_{k,m',m}$ represents the upper limit that state m transits into state m' given that content k is requested. As a result, the following condition must be satisfied

$$\sum_{m' \in \mathcal{H}_m^k} \omega_{k,m',m} \leq 1, \forall m, \forall k\,.$$

(12)

If strict equality holds in the above condition, then content k is always accepted into the cache when the cache state is m. Otherwise, content k may not be accepted into the cache even after it is requested and downloaded.

The next step is to determine which state transitions could happen and with what probabilities. Recall that the elements of $\boldsymbol{\eta}^\star$ are arranged in non-increasing order. Note that adjacent states, e.g., state m and state $m+1$, may not be neighbor states using such an order. For any given m, define the functions $V(m)$ for $m \in \{2, \dots, n\}$ and $X(m)$ for $m \in \{1, \dots, n-1\}$, both mapping from a state to one of its neighbor states as follows

$$V(m) = \underset{\hat{m} \in \mathcal{H}_m}{\operatorname{argmin}} \{\eta^{\hat{m}} | \eta^{\hat{m}} \geq \eta^m\},$$

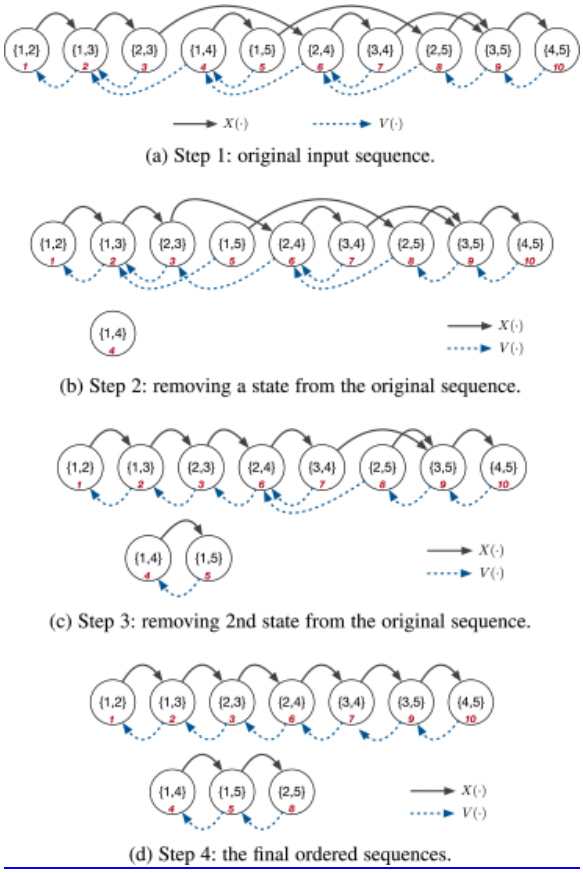
(13a)

$$X(m) = \underset{\check{m} \in \mathcal{H}_m}{\operatorname{argmax}} \{\eta^{\check{m}} | \eta^{\check{m}} \leq \eta^m\}.$$

(13b)

An illustration of $V(\cdot)$ and $X(\cdot)$ when $N_f = 5$ and $c = 2$ is given in Fig. 4a. Using $V(m)$ and $X(m)$ on state m, two cases are possible.

- $X(m) = m + 1$ and $V(m + 1) = m$: In such case, states m and $m + 1$ are both adjacent and neighbor states (e.g., states 1 and 2 in Fig. 4a).



(a) Step 1: original input sequence.

(b) Step 2: removing a state from the original sequence.

(c) Step 3: removing 2nd state from the original sequence.

(d) Step 4: the final ordered sequences.

Fig. 4. An illustration of using $V(m)$ and $X(m)$ to group states into ordered sequences. The states satisfy $\eta_1^\star \geq \eta_2^\star \geq \ldots \geq \eta_{10}^\star$.

- $X(m) \neq m + 1$ and $V(m + 1) \neq m$: In such case, m and $m + 1$ are adjacent but not neighbor states (e.g., states 3 and 4 in Fig. 4a).

To facilitate the design of state transitions, we organize the states into several sequences so that: 1). $\eta_m^\star$ in each sequence is sorted in a non-increasing order; and 2). adjacent states in the same sequence are always neighbors. Denote the original sequence of all states by S_0. Using the procedure in Algorithm 1 repeatedly (by setting the output L as the input sequence S_0 of the next run until the output L is empty), the above-mentioned ordered sequences can be obtained. The procedure is illustrated in 4 steps in Fig. 4.

The connection points identified in Step 3 and Step 5 of Algorithm 1 are the points where the next sequence of states may connect with the current sequence in the Markov chain. We refer to the points where the first state and the last state of the next sequence can connect to as branch and merge points, respectively. Determine one branch point and one merge point for each sequence is not difficult and neglected here. Denote the lth sequence of states as S_l and the length of S_l as K_l. Denote the kth state, the branch point, and the merge point of sequence S_l as $S_l(k)$, $B(l)$, and $M(l)$, respectively. This is illustrated in Fig. 5a.

Algorithm 1. Generating Ordered Sequence of Neighbor States

Input: S_0

Output: S, L

Initialization: $S = S_0$; L set to an empty sequence.

for State $m = 2$ to n **do**

if $X(V(m)) \neq m$ **then**

Mark $V(m)$ as a potential connection point;

Remove m from sequence $\mathbf{S}$;

Add m to the end of sequence L;

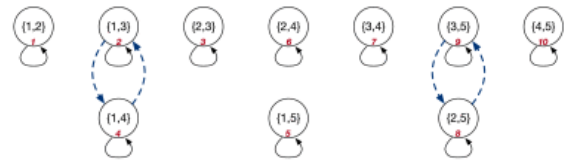
else if $V(X(m)) \neq m$ **then**

Mark $X(m)$ as a potential connection point;

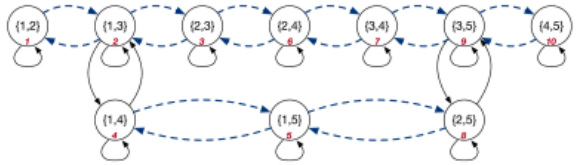
end if

end for

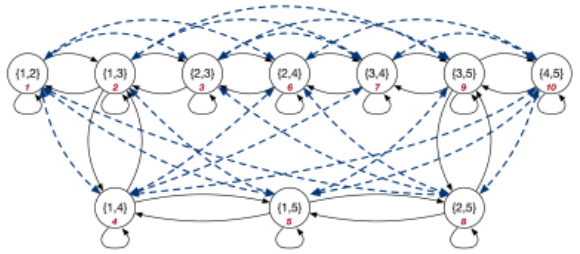
return S, L



(a) Ordered state sequence S_1 consists of the 7 states in the first row, and S_2 consists of the 3 states in the second row. The states 2 and 9 are the branch and merge points of S_2, respectively. The dashed transition links are created by Steps 2 and 3 of Algorithm 2.



(b) The dashed transition links are created by the update on Θ by Steps 4 to 6 of Algorithm 2.



(c) The Markov chain created after Θ is refined by Algorithm 3. To reduce the number of connections, bi-direcitional links are used.

Fig. 5. An illustration on generating and refining the underlying Markov chain by updating $\boldsymbol{\Theta}$ using Algorithms 2 and 3.

Given the ordered sequences, we next determine the state transition probabilities iteratively, considering one pair of states in each iteration. Initialize the state transition probability matrix $\boldsymbol{\Theta}$ to be $\mathbf{I}$. The basic procedure for

updating the state transition probabilities for a pair of neighbor states m and m such that $m' = \mathcal{H}_{i,m}^{k}$ and $m = \mathcal{H}_{m'}^{k'}$ is given as follows:

$$\delta = min\{\omega_{k,m',m}\varphi_{m',m}, \omega_{k',m,m'}\varphi_{m,m'}\frac{\eta^{m'}}{\eta^{m}}\},$$

(14a)

$$\boldsymbol{\Theta}(m', m) = \delta,$$

(14b)

$$\boldsymbol{\Theta}(m, m') = \delta\frac{\eta^{m}}{\eta^{m'}},$$

(14c)

$$\boldsymbol{\Theta}(m', m') \Leftarrow \boldsymbol{\Theta}(m', m') - \delta\frac{\eta^{m}}{\eta^{m'}},$$

(14d)

$$\boldsymbol{\Theta}(m, m) \Leftarrow \boldsymbol{\Theta}(m, m) - \delta,$$

(14e)

where $\Leftarrow$ represents the operation of assigning the value of the right-hand side expression to the left-hand side. The above procedure guarantees that, assuming $\boldsymbol{\Theta\eta}^{\star} = \boldsymbol{\eta}^{\star}$ before the update, the equality still holds after updating the transition probabilities for states m and m'.

Algorithm 2. Generating the Stochastic Matrix $\boldsymbol{\Theta}$

Input: $\boldsymbol{\eta}^{\star}$, $\{\varphi_{m,m'}\}_{\forall m,\forall m'}$, $\{\mathrm{S}_l\}$, $\{B(l)\}$, $\{M(l)\}$

Output: $\boldsymbol{\Theta}$

Initialization: Set $\boldsymbol{\Theta} = \mathbf{I}$.

for each sequence l **do**

Run the procedure (14a)-(14e) with $m = B(l), m' = \mathrm{S}_l(1)$.

Run the procedure (14a)-(14e) with $m = M(l), m' = \mathrm{S}_l(K_l)$.

for State $q = K_l$ to 2 **do**

Run the procedure (14a)-(14e) with $m' = \mathrm{S}_l(q)$ and $m = \mathrm{S}_l(q-1)$.

end for

end for

return $\boldsymbol{\Theta}$

Based on the above definitions and procedures, Algorithm 2 is proposed to generate a basic transition probability matrix $\boldsymbol{\Theta}$ that has the steady state $\boldsymbol{\eta}^{\star}$. An illustration of Algorithm 2 is given in Figs. 5a and 5b. Steps 2 and 3 of Algorithm 2 "connect" the ordered sequences found using Algorithm 1 by generating the state transition probabilities for the connection points, e.g., states 2 and 4 and states 8 and 9 as illustrated in Fig. 5a. Then, Steps 4 to 6 of Algorithm 2 generate state transitions only between adjacent and neighbor states within each ordered sequence of states. This is illustrated in Fig. 5b, which shows a basic irreducible Markov chain at the output of Algorithm 2. The generated Markov chain satisfies $\boldsymbol{\Theta\eta}^{\star} = \boldsymbol{\eta}^{\star}$ but most of the off-diagonal elements in $\boldsymbol{\Theta}$ are 0. As a result, the mixing time can be long. In order to reduce the mixing time, we use a refinement procedure to connect more states, which is given in Algorithm 3 and illustrated in Fig. 5c. The refinement in Algorithm 3 is designed based on the fact that the mixing time of the Markov chain is determined by the second largest eigenvalue of the transition matrix [47]. Simulation examples in Section 5 will demonstrate the performance of the refinement. Detailed analysis, however, is beyond the scope of this work.

Theorem 1.

In the case when the instantaneous content popularity converges to the average content popularity, the matrix $\boldsymbol{\Theta}$ generated by Algorithm 2 has the following properties:

- $\boldsymbol{\Theta}$ is a valid state transition matrix;
- $\boldsymbol{\Theta}$ guarantees that $\tau_{m,m'} \in [0,1]$ for all m and $m' \neq m$;
- $\boldsymbol{\Theta}$ satisfies Eq. (10);
- The underlying Markov chain specified by $\boldsymbol{\Theta}$ has a unique steady state which is $\boldsymbol{\eta}^{\star}$.

Proof.

See Appendix B, available in the online supplemental material.

Based on Theorem 1, the underlying Markov chain can converge in distribution to $\boldsymbol{\eta}^{\star}$ when the instantaneous content popularity is constant. Accordingly, the content caching probabilities converge to the optimal caching probabilities $\mathbf{p}^{\star}$. When the instantaneous content popularity is time-varying, dynamic caching based on the designed state transitions may adapt to varying content popularity. Nevertheless, it should be noted that the adaptivity depends on the specific content request statistics, and therefore neither dynamic or static caching can claim to outperform the other in all cases. We will demonstrate and discuss this in Section 5.

Algorithm 3. Refining the Matrix $\boldsymbol{\Theta}$

Input: Θ, $\{\varphi_{m,m'}\}_{\forall m,\forall m'}$

Output: $\boldsymbol{\Theta}$

for State $m = 1$ to n **do**

for State $m' = m + 1$ to n **do**

if $m' \in \mathcal{H}_m$ and $\boldsymbol{\Theta}(m', m) = 0$ **then**

Run the procedure in (14a)-(14e)

end if

end for

end for

return $\boldsymbol{\Theta}$

In the case of time-varying content popularity, assume that there are N_{r} requests in the considered time window. Denote the instantaneous content popularity at the qth request by $\{\varphi_k^{(q)}\}_{\forall k\in\mathcal{F}}$, where $q \in \{1,\dots,N_{\mathrm{r}}\}$. The average content popularity is specified by $\{\varphi_k\}_{\forall k\in\mathcal{F}}$. The instantaneous state transition matrix $\boldsymbol{\Theta}^{(q)}$ is time-varying as a result of the time-varying instantaneous content popularity.

Lemma 3.
Under time-varying content popularity, the overall state transition matrix averaged over the N_{r} requests, denoted by $\overline{\boldsymbol{\Theta}}$, is equal to the matrix $\boldsymbol{\Theta}$ generated by Algorithm 2.

Proof.
See Appendix C, available in the online supplemental material.

It should be noted that the result $\overline{\boldsymbol{\Theta}} = \boldsymbol{\Theta}$ in Lemma 3 does not imply that the average content caching probabilities are equal to the target content caching probabilities when the instantaneous content popularity is time-varying. To the best of our knowledge, designing dynamic caching which can achieve target average content caching probabilities without knowing the instantaneous content request probabilities is impossible. After all, the ability of adapting to content popularity implies that the resulting content caching probabilities depend on the content request statistics (not just the average content popularity).

4.3 Discussion on the Scalability

The proposed design involves finding the transition probabilities for each cache state, and the overall number of cache states, i.e., $\binom{N_{\mathrm{f}}}{c}$, can be prohibitively large in practice. Nevertheless, we can limit the number of states to be considered. Next, we provide some methods for reducing the number of states when the number of contents is large. Section. 5 will use a numerical example to demonstrate the simplification of the proposed dynamic caching and the resulting performance.

On the Content Level. Although the number of contents can be large, the number of "popular" contents which are worth caching can be small. The study in [48] shows that a large portion of YouTube videos ($> 70\%$) are requested only once from an edge network. It follows that a significant portion of contents will be assigned with a caching probability of zero. This can be seen in the example illustrating Fig. 2b in Section 3.3. Denote the number of all contents which are assigned with a nonnegative caching probability as N_{p}. Then, the number of cache states to be considered decreases from $\binom{N_{\mathrm{f}}}{c}$ to $\binom{N_{\mathrm{p}}}{c}$, which is a significant reduction when N_{p} is much smaller than N_{f}. Moreover, the "very popular" contents which are assigned with a caching probability of 1 also reduces the number of cache states to be considered. If Ne contents are assigned a caching probability in $(0,1)$, then the number of cache states decreases to $\binom{N_{\mathrm{e}}}{c}$.

On the State Level. We have illustrated in Fig. 2 and the related example on how to choose a state caching probability vector that has less non-negative elements. On top of that, a further and more effective simplification can be used. Specifically, we could limit the considered states to a small number of states with large overall cached content request probabilities. This will significantly reduce the number of states to be considered and the resulting dimension of the state transition matrix. For example, if $N_{\mathrm{e}} = 100$ and $c = 20$, there are more than 1020 states. However, we can consider the top 1000 states that cache the most popular contents only. By setting a proper cut-off threshold, the proposed dynamic caching can still yield satisfactory performance. We will show this with an example in Section 5, in which there are 10000 contents but we only consider 30 states.

SECTION 5 Numerical Examples

5

Example 1. (The convergence speed of the underlying Markov chain corresponding to the designed $\boldsymbol{\Theta}$).

In this illustrative example, 5 contents and a cache with size 2 is considered. Thus, there are 10 cache states and the state caching probability vector is a vector with 10 elements. The transition probability matrix $\boldsymbol{\Theta}$ is first generated by Algorithms 2 and then refined by Algorithms 3. For the purpose of illustrating the convergence performance of the proposed $\boldsymbol{\Theta}$, a constant instantaneous content popularity based on Zipf distribution is used in this example. The convergence speed of the underlying Markov chains of $\boldsymbol{\Theta}$ at the output of Algorithms 2 and Algorithm 3 are shown in the top and bottom subplots of Fig. 6, respectively. In each figure, 10000 tests with randomly generated initial $\boldsymbol{\eta_0}$ are conducted. Three observations can be made from Fig. 6. First, $\boldsymbol{\eta}$ always converge (in distribution) to the target $\boldsymbol{\eta^\star}$ with the designed replacement policy represented by $\boldsymbol{\Theta}$ in the 20000 tests regardless of the initial caching strategy, which validates the result in Theorem 1. Second, the convergence speed is shorter on average when the $\boldsymbol{\eta_0}$ is closer to $\boldsymbol{\eta^\star}$ and vice versa. Third, the refinement of $\boldsymbol{\Theta}$ by Algorithm 3 significantly reduces the convergence speed, i.e., by a factor of 10.

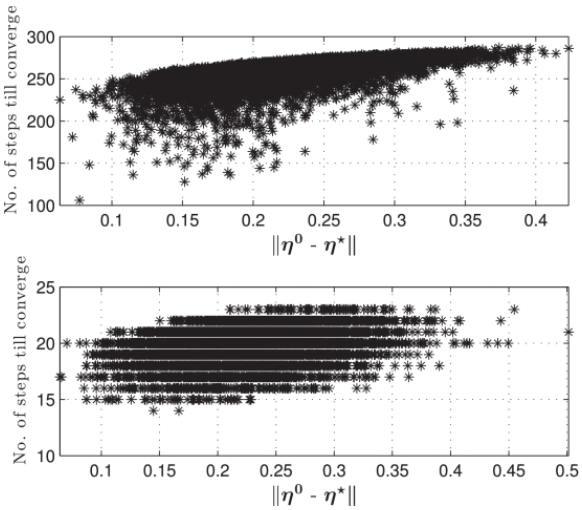


Fig. 6. Demonstration of the convergence speed of $\boldsymbol{\Theta}$ generated by Algorithms 2 and 3 with 10000 random initial states.

5

Example 2. (The comparison of the convergence (in distribution) of replacement policies).

In this illustrative example, 15 contents and a cache with size 8 is considered. Thus, there are 6435 cache states, and the state caching probability vector has 6435 elements. The convergence performance of three replacement policies under constant content popularity is compared: the proposed policy corresponding to the designed $\boldsymbol{\Theta}$, LRU, and LFU. The content requests from UE are randomly generated and follows a Zipf distribution. The convergence is represented through the square norm of the difference between the current caching strategy $\boldsymbol{\eta}$ and the target caching strategy $\boldsymbol{\eta^\star}$ versus the number of content requests since the beginning of the simulations. The comparison of the convergence performance is shown in Fig. 7. It can be seen from this figure that the proposed replacement policy can converge to $\boldsymbol{\eta^\star}$ in distribution and thereby implement a given set of caching probabilities when the instantaneous content popularity is constant. By contrast, LRU or LFU cannot converge to $\boldsymbol{\eta^\star}$ under the same condition.

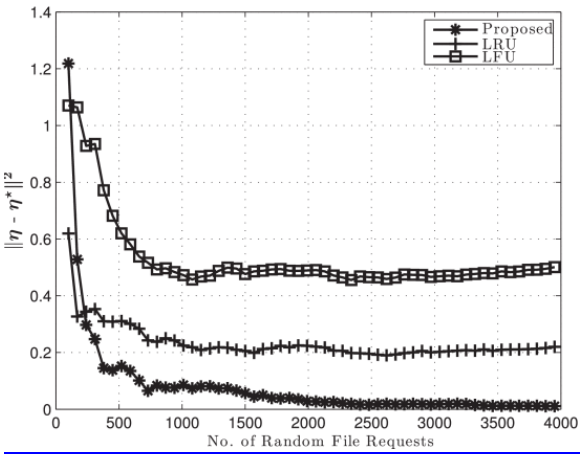


Fig. 7. Comparison of replacement policies: the proposed, LRU, and LFU.

5

Example 3. (The benefit of probabilistic content replacement).

In this illustrative example, the benefit of dynamic probabilistic content replacement with the designed replacement policy {τm,m′} **is demonstrated with 23 contents and a cache of size 2. A time duration divided into 50 sessions is considered, and** 2×10^6 **content requests are generated in total. The 23 contents are equally popular overall but the popularity of each content varies in each session. Two different cases of variations in content popularity (in terms of content request probability) are considered: random fluctuation and smooth change, as shown in the top subplots of Figs. 8 and 9, respectively. The corresponding cache hit ratio by using dynamic probabilistic content replacement is given in the bottom subplots of Figs. 8 and 9, respectively. As the overall popularity is the same for each content, caching any two content without replacement would lead to a cache hit ratio of 2/23, or 0.087 approximately. It can be seen that from Figs. 8 and 9 that the cache hit ratio is improved by using probabilistic content replacement in either case. The overall cache hit ratio is 0.1063 and 0.1085, equivalent to an increase of 22 and 25 percent, in the cases of Figs. 8 and 9, respectively. The figures demonstrate that designed probabilistic content replacement based on the average content popularity can improve cache hit ratio by adapting to the varying content popularity when the instantaneous content popularity changes over time.**

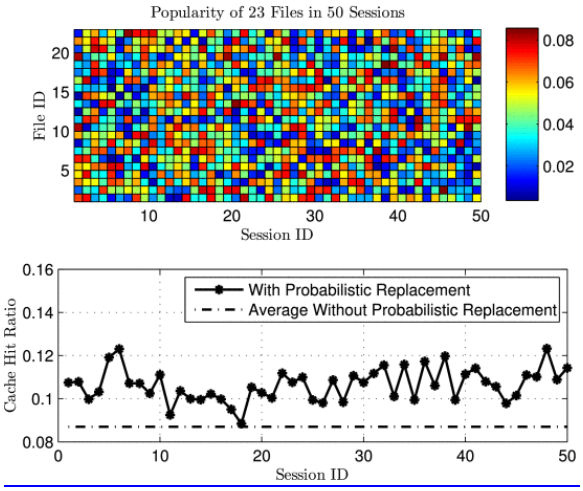


Fig. 8. Cache hit ratio with probabilistic replacement when content popularity varies over time - the case of random fluctuation.

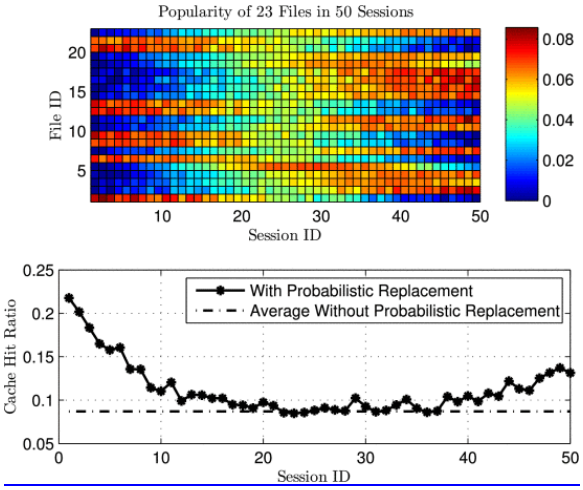


Fig. 9. Cache hit ratio with probabilistic replacement when content popularity varies over time - the case of smooth change.

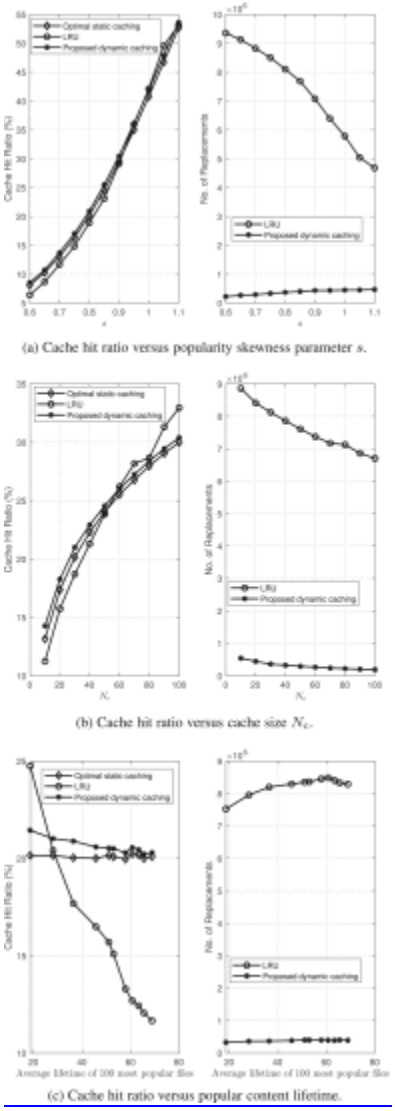
5

Example 4. (The case of large content number and shot noise request model).

In this example, we consider a total of 10000 contents and demonstrate how to apply the proposed idea of dynamic caching when there are a large number of contents. The requests for the contents are generated in a window of 100 minutes based on the shot noise model using the exponential shape [33]. The average number of requests for each content in the considered time window follows a Zipf distribution with parameter s. The average content popularity is based on the overall number of requests in the considered 100 minutes. The first request for the contents follows a uniform distribution in $[0,40]$ minutes. Contents have different lifetime, while the lifetime of every content is short so that almost all requests occur in the considered 100-minute time window. In such a scenario, there are two problems in applying the proposed dynamic caching. First, the number of cache state is prohibitively large due to the 10000 contents. Second, the optimal static caching strategy reduces to caching the c most popular contents deterministically (i.e., $\eta_1^\star = 1$ and $\eta_l^\star = 0, \forall l > 1$). To apply and evaluate the idea of the proposed dynamic caching, we make the following two simplifications of the proposed design. First, we only consider 30 cache states with the largest static cache hit ratio. Second, in order to apply dynamic state transition, we use a non-optimal $\boldsymbol{\eta}^\star$ (of size 30×1), in which the state caching probability of state l is proportional to $\sum_{k \in \mathcal{G}^l} \varphi_k$. Note that the above two settings significantly simplify our original design and can lead to performance degradations. We compare the cache hit ratio of the optimal static caching, LRU, and the (simplified) proposed dynamic caching as well as the number of replacements of LRU and the (simplified) proposed dynamic caching under various cache sizes, values of s, and content lifetime (i.e., the duration between the first and the last requests).

Fig. 10a shows the cache hit ratio of the three schemes as well as the number of replacements of LRU and the (simplified) proposed dynamic caching versus the content popularity skewness parameter s in the range of $[0.6,1.1]$. The cache size is 30, and the average content lifetime for the 100 most popular files is around 32.7 minutes. Each point in the figure is an average over 240 runs of simulations, each of which randomly generates all content requests for the 10000 contents. The plot on the left-hand side shows that, when s is small, the optimal static caching outperform LRU. Moreover, the proposed dynamic caching outperforms both LRU and the optimal static caching. The reason behind the two observation is that the replacement made by LRU can be less targeted when the skewness of popularity is small (i.e., the chance that the new content is less likely to be requested than the replaced content could be large) while the replacements made by the proposed method are

guided by the average content popularity information. When s becomes large and the number of "most popular" contents become smaller, LRU can gain an advantage since all three schemes cache the most popular contents with large probabilities, but LRU has the strongest adaptivity and uses the rest of the cache to store temporarily popular contents. However, overall, the proposed dynamic caching, even after significant simplification, still outperforms both the optimal static caching and LRU in a wide range of s. Furthermore, the plot on the right-hand side shows that the proposed dynamic caching achieves such a performance with much less number of replacements compared to LRU (i.e., less than 1/40 of the replacements by LRU when $s = 0.6$ and less than 1/8 when $s = 1.1$). Therefore, the replacements in the proposed dynamic caching are more effective due to the exploitation of the average content popularity information.



(a) Cache hit ratio versus popularity skewness parameter s.

(b) Cache hit ratio versus cache size N_c.

(c) Cache hit ratio versus popular content lifetime.

Fig. 10. Comparison of the optimal static caching, LRU, and the proposed dynamic caching under different settings.

Fig. 10b shows the cache hit ratio of the three schemes as well as the number of replacements of LRU and the (simplified) proposed dynamic caching versus the cache size N_c in the range of [10,100]. The popularity skewness parameter s is 0.8, and the average content lifetime for the 100 most popular files is 32.7 minutes. Each point in the figure is an average over 240 runs of simulations, each of which randomly generates all content requests for the 10000 contents. The logarithmic growth in the cache hit ratio versus the cache size shown

in Fig. 10b is consistent with the observations in existing research, e.g., [49]. The proposed dynamic caching always has an advantage over LRU in the cache hit ratio when the cache size N_c is no larger than 60 (i.e., 6‰ of the size of all contents). In addition, the proposed dynamic caching always outperforms the optimal static caching. Furthermore, similar to the case shown in Fig. 10a, the proposed dynamic caching uses a significantly less number of replacements compared to that of LRU.

Fig. 10c shows the cache hit ratio of the three schemes as well as the number of replacements of LRU and the (simplified) proposed dynamic caching versus the average content lifetime for the 100 most popular files. The popularity skewness parameter s is 0.8, and the cache size N_c is 30. Each point in the figure is an average over 240 runs of simulations, each of which randomly generates all content requests for the 10000 contents. The left-hand side plot shows that, when the average lifetime of the 100 most popular contents is small, e.g., less than 30 minutes in the considered 100-minute window, dynamic caching (including LRU and the proposed caching) has an advantage over the static caching. This is because, given a fixed time window, a short lifetime can lead to a stronger temporal locality in content popularity. Moreover, when the content lifetime increases, the performance of LRU decreases very fast while the proposed dynamic caching still achieves a better cache hit ratio compared to the optimal static caching. The plot on the right-hand side shows again that the proposed dynamic caching uses a significantly less number of replacements than LRU.

SECTION 6 Conclusion

In this work, we have studied dynamic probabilistic caching when the instantaneous content popularity may vary with time but the average content popularity is known. Specifically, we have designed probabilistic content placement and replacement policies with the objective of increasing cache hit ratio under varying instantaneous content popularity while converging to the target content caching probabilities under constant instantaneous content popularity. On the probabilistic content placement, we have established a novel connection between caching probabilities and probabilistic content placement policies through mixed strategies. On the probabilistic content replacement, we have designed two algorithms to generate and refine the transition probability matrix so that the Markov chain has a unique steady state which achieves the target content placement. Our study has demonstrated that the proposed dynamic probabilistic caching can be applied on top of the existing results on finding the optimal caching probabilities to improve the cache hit ratio under time-varying content popularity while the optimal caching probabilities can be used as the target content caching probabilities. When combined with content popularity prediction, the proposed design can provide a competitive approach for dynamic caching under time-varying content popularity.

ACKNOWLEDGMENTS

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada under Grant STPGP-493787 and in part by the Nature Science Foundation of China under Grant 61801011.

References

1. X. Wang, M. Chen, T. Taleb, A. Ksentini and V. C. M. Leung, "Cache in the air: Exploiting content caching and delivery techniques for 5G systems", *IEEE Commun. Magazine*, vol. 52, no. 2, pp. 131-139, Feb. 2014.

2. I. Parvez, A. Rahmati, I. Guvenc, A. I. Sarwat and H. Dai, "A survey on low latency towards 5G: RAN core network and caching solutions", *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 3098-3130, 2018.

3. E. K. Markakis, K. Karras, A. Sideris, G. Alexiou and E. Pallis, "Computing caching and communication at the Edge: The cornerstone for building a versatile 5G ecosystem", *IEEE Commun. Magazine*, vol. 55, no. 11, pp. 152-157, Nov. 2017.
4. X. Zhang and Q. Zhu, "Hierarchical caching for statistical QoS guaranteed multimedia transmissions over 5G Edge computing mobile wireless networks", *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 12-20, Jun. 2018.
5. S. Zhang, J. Li, H. Luo, J. Gao, L. Zhao and X. S. Shen, "Towards fresh and low-latency content delivery in vehicular networks: An Edge caching aspect", *Proc. 10th Int. Conf. Wireless Commun. Signal Process.*, pp. 1-6, 2018.
6. K. Zhang, S. Leng, Y. He, S. Maharjan and Y. Zhang, "Cooperative content caching in 5G networks with mobile Edge computing", *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 80-87, Jun. 2018.
7. T. Liu, J. Li, F. Shu, H. Guan, S. Yan and D. N. K. Jayakody, "On the incentive mechanisms for commercial Edge caching in 5G Wireless networks", *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 72-78, Jun. 2018.
8. M. Ji, G. Caire and A. F. Molisch, "Wireless device-to-device caching networks: Basic principles and system performance", *IEEE J. Sel. Areas Commun.*, vol. 34, no. 1, pp. 176-189, Jan. 2016.
9. B. M. Maggs and R. K. Sitaraman, "Algorithmic nuggets in content delivery", *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 3, pp. 52-66, Jul. 2015.
10. S. Jeon, S. Hong, M. Ji, G. Caire and A. F. Molisch, "Wireless multihop device-to-device caching networks", *IEEE Trans. Inf. Theory*, vol. 63, no. 3, pp. 1662-1676, Mar. 2017.
11. L. Qiu and G. Cao, "Popularity-aware caching increases the capacity of Wireless networks", *IEEE Trans. Mobile Comput.*, vol. 19, no. 1, pp. 173-187, Jan. 2020.
12. S. Tarnoi, K. Suksomboon, W. Kumwilaisak and Y. Ji, "Performance of probabilistic caching and cache replacement policies for content-centric networks", *Proc. 39th IEEE Conf. Local Comput. Netw.*, pp. 99-106, 2014.
13. W. Bao, D. Yuan, K. Shi, W. Ju and A. Y. Zomaya, "Ins and outs: Optimal caching and re-caching policies in mobile networks", *Proc. 18th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, pp. 41-50, 2018.
14. I. Psaras, W. K. Chai and G. Pavlou, "In-network cache management and resource allocation for information-centric networks", *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 11, pp. 2920-2931, Nov. 2014.
15. D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee and K. K. Ramakrishnan, "Optimal content placement for a large-scale VoD system", *IEEE/ACM Trans. Netw.*, vol. 24, no. 4, pp. 2114-2127, Aug. 2016.
16. I. D. Baev, R. Rajaraman and C. Swamy, "Approximation algorithms for data placement problems", *SIAM J. Comput.*, vol. 38, no. 4, pp. 1411-1429, 2008.
17. N. Deng and M. Haenggi, "The benefits of hybrid caching in Gauss–Poisson D2D networks", *IEEE J. Sel. Areas Commun.*, vol. 36, no. 6, pp. 1217-1230, Jun. 2018.
18. S. Zhang, N. Zhang, P. Yang and X. Shen, "Cost-effective cache deployment in mobile heterogeneous networks", *IEEE Trans. Veh. Technol.*, vol. 66, no. 12, pp. 11264-11276, Dec. 2017.
19. S. Zhang, P. He, K. Suto, P. Yang, L. Zhao and X. Shen, "Cooperative edge caching in user-centric clustered mobile networks", *IEEE Trans. Mobile Comput.*, vol. 17, no. 8, pp. 1791-1805, Aug. 2018.
20. Z. Su, Y. Hui, Q. Xu, T. Yang, J. Liu and Y. Jia, "An edge caching scheme to distribute content in vehicular networks", *IEEE Trans. Veh. Technol.*, vol. 67, no. 6, pp. 5346-5356, Jun. 2018.

21. Q. Li, W. Shi, X. Ge and Z. Niu, "Cooperative edge caching in software-defined hyper-cellular networks", *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2596-2605, Nov. 2017.
22. Y. Cui, Z. Wang, Y. Yang, F. Yang, L. Ding and L. Qian, "Joint and competitive caching designs in large-scale multi-tier wireless multicasting networks", *IEEE Trans. Commun.*, vol. 66, no. 7, pp. 3108-3121, Jul. 2018.
23. M. Mangili, F. Martignon, S. Paris and A. Capone, "Bandwidth and cache leasing in Wireless information-centric networks: A game-theoretic study", *IEEE Trans. Veh. Technol.*, vol. 66, no. 1, pp. 679-695, Jan. 2017.
24. Y. Yang, Y. Wu, N. Chen, K. Wang, S. Chen and S. Yao, "LOCASS: Local optimal caching algorithm with social selfishness for mixed cooperative and selfish devices", *IEEE Access*, vol. 6, pp. 30060-30072, May 2018.
25. W. Wang, D. Niyato, P. Wang and A. Leshem, "Decentralized caching for content delivery based on blockchain: A game theoretic perspective", *Proc. IEEE Int. Conf. Commun.*, pp. 1-6, 2018.
26. Y. Chen, M. Ding, J. Li, Z. Lin, G. Mao and L. Hanzo, "Probabilistic small-cell caching: Performance analysis and optimization", *IEEE Trans. Veh. Technol.*, vol. 66, no. 5, pp. 4341-4354, May 2017.
27. K. Li, C. Yang, Z. Chen and M. Tao, " Optimization and analysis of probabilistic caching in \$N\$ N - Tier heterogeneous networks ", *IEEE Trans. Wireless Commun.*, vol. 17, no. 2, pp. 1283-1297, Feb. 2018.
28. Y. Zhou, Z. Zhao, R. Li, H. Zhang and Y. Louet, "Cooperation-based probabilistic caching strategy in clustered cellular networks", *IEEE Trans. Commun. Lett.*, vol. 21, no. 9, pp. 2029-2032, Sep. 2017.
29. D. Liu and C. Yang, "Caching policy toward maximal success probability and area spectral efficiency of cache-enabled HetNets", *IEEE Trans. Wireless Commun.*, vol. 65, no. 6, pp. 2699-2714, Jun. 2017.
30. J. Wen, K. Huang, S. Yang and V. O. K. Li, "Cache-enabled heterogeneous cellular networks: Optimal tier-level content placement", *IEEE Trans. Wireless Commun.*, vol. 16, no. 9, pp. 5939-5952, Sep. 2017.
31. M. Zeng et al., "Temporal-spatial mobile application usage understanding and popularity prediction for edge caching", *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 36-42, Jun. 2018.
32. T. Tanaka, S. Ata and M. Murata, "Analysis of popularity pattern of user generated contents and its application to content-aware networking", *Proc. IEEE Globecom Workshops*, pp. 1-6, 2016.
33. S. Traverso, M. Ahmed, M. Garetto, P. Giaccone, E. Leonardi and S. Niccolini, "Temporal locality in today's content caching: Why it matters and how to model it", *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 5, pp. 5-12, Nov. 2013.
34. E. Leonardi and G. L. Torrisi, "Least recently used caches under the shot noise model", *Proc. IEEE Conf. Comput. Commun.*, pp. 2281-2289, 2015.
35. M. Garetto, E. Leonardi and V. Martina, "A unified approach to the performance analysis of caching systems", *ACM Trans. Modeling Perform. Eval. Comput. Syst.*, vol. 1, no. 3, pp. 12:1-12:28, May 2016.
36. G. Gürsun, M. Crovella and I. Matta, "Describing and forecasting video access patterns", *Proc. IEEE INFOCOM*, pp. 16-20, 2011.
37. B. Blaszczyszyn and A. Giovanidis, "Optimal geographic caching in cellular networks", *Proc. IEEE Int. Conf. Commun.*, pp. 3358-3363, 2015.
38. N. Golrezaei, A. F. Molisch, A. G. Dimakis and G. Caire, "Femtocaching and device-to-device collaboration: A new architecture for wireless video distribution", *IEEE Commun. Magazine*, vol. 51, no. 4, pp. 142-149, Apr. 2013.

39. J. Gao, L. Zhao and X. Shen, "The study of dynamic caching via state transition field—the case of time-invariant popularity", *IEEE Trans. Wireless Commun.*, vol. 18, no. 12, pp. 5924-5937, Dec. 2019.
40. J. Gao, L. Zhao and X. Shen, "The study of dynamic caching via state transition field—the case of time-varying popularity", *IEEE Trans. Wireless Commun.*, vol. 18, no. 12, pp. 5938-5951, Dec. 2019.
41. A. Gaddah and T. Kunz, "Extending mobility to publish/subscribe systems using a pro-active caching approach", *Mobile Inf. Syst.*, vol. 6, no. 4, pp. 293-324, 2010.
42. X. Vasilakos, V. A. Siris and G. C. Polyzos, "Addressing niche demand based on joint mobility prediction and content popularity caching", *Comput. Netw.*, vol. 110, pp. 306-323, 2016.
43.F. Zhang et al., "EdgeBuffer: Caching and prefetching content at the edge in the MobilityFirst future internet architecture", *Proc. IEEE 16th Int. Symp. a World Wireless Mobile Multimedia Netw.*, pp. 1-9, 2015.
44.O. Bahat and A. M. Makowski, "Optimal replacement policies for nonuniform cache objects with optional eviction", *Proc. IEEE 22nd Annu. Joint Conf. IEEE Comput. Commun. Societies*, pp. 427-437, 2003.
45.S. Shukla and A. A. Abouzeid, "Optimal device-aware caching", *IEEE Trans. Mobile Comput.*, vol. 16, no. 7, pp. 1994-2007, Jul. 2017.
46.C. P. Robert and G. Casella, Monte Carlo Statistical Methods, New York, NY, USA:Springer, 2004.
47.S. Boyd, P. Diaconis and L. Xiao, "Fastest mixing markov chain on a graph", *SIAM Rev.*, vol. 46, no. 4, pp. 667-689, Dec. 2004.
48.N. Carlsson and D. Eager, "Ephemeral content popularity at the edge and implications for on-demand caching", *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 6, pp. 1621-1634, Jun. 2017.
49.A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan and J. R. Wilcox, "Information-centric networking: Seeing the forest for the trees", *Proc. 10th ACM Workshop Hot Topics Netw.*, pp. 1-6, 2011.