Achieving Freshness in Single/Multi-User Caching of Dynamic Content over the Wireless Edge

Bahman Abolhassani[1], John Tadrous[2], Atilla Eryilmaz[1]
[1,2] Department of Electrical and Computer Engineering
[1] The Ohio State University, Columbus, 43210. Email: abolhassani.2@osu.edu, eryilmaz.2@osu.edu
[2] Gonzaga University, Spokane, WA 99202. Email: tadrous@gonzaga.edu

***Abstract*—Existing content caching mechanisms are predominantly geared towards easy-access of content that is static once created. However, numerous applications, such as news and dynamic sources with time-varying states, generate 'dynamic' content where new updates replace previous versions. This motivates us in this work to study the freshness-driven caching algorithm for dynamic content, which accounts for the changing nature of data content. In particular, we provide new models and analyses of the average operational cost both for the single-user and multi-user scenarios. In both scenarios, we characterize the performance of the optimal solution and develop algorithms to select the content and the update rate that the user(s) must employ to have low-cost access to fresh content. Moreover, our work reveals new and easy-to-calculate key metrics for quantifying the caching value of dynamic content in terms of their refresh rates, popularity, number of users in the multi-user group, and the fetching and update costs associated with the optimal decisions. We compare the proposed freshness-driven caching strategies with benchmark caching strategies like cache the most popular content. Results demonstrate that freshness-driven caching strategies considerably enhance the utilization of the edge caches with possibly orders-of-magnitude cost reduction. Furthermore, our investigations reveals that multi-user scenario, benefiting from the multicasting property of wireless service to update the cache content, can be cost effective compared to single user caching, as the number of users increases.**

***Index Terms*—Wireless Content Distribution, Caching, Dynamic Content.**

I. Introduction

With the wide availability of content delivery networks, many applications utilize edge cache at end-users to deliver dynamic contents, reducing the network latency and system congestion during the peak-traffic time. By caching a large number of dynamic contents in the edge caches, the average response time can be reduced, benefiting from higher cache hit rates. However higher hit rates come at the expense of a less fresh content, resulting in higher overall system cost.

Numerous works study the content delivery in caching systems and effective strategies have been proposed. In [1], [2] and [3], authors study the benefits of caching with the focus being mainly on exploiting the history or statistics of the user demand. These works are based on the promise that the content stored in the cache will ultimately be used. An important factor that may greatly affect the caching decision is the content generation dynamics. However, these studies fail to take into consideration the possibility of content refreshment which renders the current version of the cached content less relevant or possibly obsolete. These types of dynamic contents include news and social network updates where the users prefer to have the most fresh version of the content while also making sure that the total cost of the network remains low.

As the data gets updated in data sources, currently cached version becomes out of date or stale since users are interested in the latest version of data [4]. Most caching policies, however, do not consider the content generation dynamics and focus alternatively on the content popularity. It turns out that the content generation rate plays a crucial role in deciding which data to be cached and with what rate should the cached data be updated to account for the dynamically varying content at the data source. In [5], Candan, et al. propose a framework which enables dynamic content caching for database-driven e-commerce sites by intelligently invalidating dynamically generated web pages in the caches. In [6], authors mention that great benefits can be reached by incorporating the freshness in caching but do not investigate the case due to complexity of it. In [7], authors propose a dynamic cache management policy based on the history of requests and age of the content to update the existing content of the cache. They show that the optimal policy for minimizing the number of missed requests is to keep the packets that have the highest instantaneous request value in the cache. In [8], authors study a least recently used (LRU) policy for cache management in a web browser but they suggest that finding a good caching policy that is conscious of document size and delay may be difficult. In [9], Chen et al. propose LA2U and LAUD policies to implement the update rate in caching. LA2U computes the access-to-update ratio for the cached data items, and evicts the one with the smallest ratio. Notably, LA2U is equivalent to least frequently used (LFU), in the absence of content updates. LAUD works in the same way as LA2U except that LAUD uses popularity-to-update differences rather than access-to-update ratios to decide which items to cache. In [10], Akon et al., present OUR as a cache replacement scheme which uses both update rates and content popularities to achieve superior-guaranteed performance. They define a performance

This work was supported primarily by the ONR Grant N00014-19-1-2621, and in part by the NSF grants: CNS-NeTS-1514260, CNS-NeTS-1717045, CMMISMOR-1562065, CNS-ICN-WEN-1719371, and CNS-SpecEES-18243; and the DTRA grant HDTRA1-18-1-0050.

factor (PF) for each data item. If the newly requested item has a higher PF than that of any cached item, the item with the lowest PF is evicted, and the new item is stored in the cache. Otherwise, the requested item is not cached. However, having a closed form metric that also consider the freshness and can be used to sort the items and achieve a close to optimal solution is not fully investigated.

In this paper, we focus on the design of new caching strategies in the presence of dynamically changing data content and provide a design framework and performance analysis of relevant efficient caching strategies. With dynamically changing data content, the older content versions lose their value at different rates. A freshness-driven caching paradigm must account for these dynamics so as to optimally balance the costs of caching a content and the costs of serving the content non-fresh.

In particular, we propose a freshness-driven caching algorithm for dynamic content, which accounts for the update rate of data content both for the single-user and multi-user cases and provide an analysis of the average operational cost for both cases. We aim to reveal the gains of freshness-driven caching compared to other basic caching strategies. Our contributions, along with the organization of the paper, are as follows.

- In Section II, we present a tractable caching model for serving dynamic content over wireless broadcast channels.
- In Section III, for a database of N data items with an arbitrary popularity distribution that serves a single user with a limited cache space, we propose a suboptimal caching algorithm, Algorithm 1, that gives the cache checking and update rate together with the set of items to be cached in order to minimize the average system cost. We prove that our proposed algorithm optimally minimizes the average cost for any given cache check and update rate, and always outperforms the traditional cache the most popular items strategy, even with optimized cach check and update rates.
- In Section IV, by distributing the cache capacity among multiple local users, we develop an optimal caching algorithm, Algorithm 2, that reveals the potential benefits of the *multicasting* property in wireless networks for optimal caching. We show that our proposed algorithm always minimizes the aggregate average cost of the system. Finally, we conclude the work in Section V.

II. System Model

Consider the network setup shown in Fig. 1, with a database hosting a set $\mathcal{N}$ of N data items and serving M users. Each data item $n \in \mathcal{N}$ is dynamically refreshed with a content refresh being sufficient for the user to consume without the need for older content from the same data item. Content refreshes arrive to data item n according to a Poisson process with rate $\lambda_n \geq 0$. We consider the vector $\boldsymbol{\lambda} = (\lambda_n)_{n=1}^N$ as the collection of the data items refresh rates.

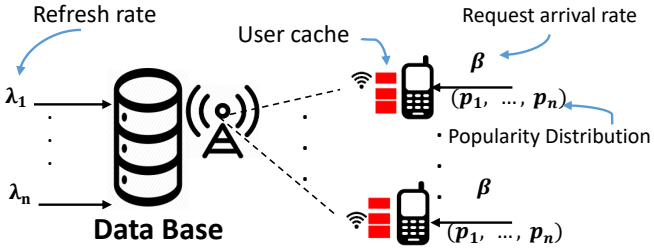

Fig. 1: Caching with freshness dynamics.

Each user m generates requests according to a Poisson process with rate $\beta \geq 0$. A generated request from any user targets data item n with probability p_n. That is, the vector $\mathbf{p} = (p_n)_{n=1}^N$ captures the popularity profile of the data items. Users are equipped with a limited-storage cache that can hold K different items.

When a user generates a request to a data item that is found in the cache of that user, the request is fulfilled immediately from that user's cache under a "freshness cost". The freshness cost is incurred due to the fact that the cached content may not be the most fresh version. We associate a freshness cost with such events which increases linearly with the *age* of the cached content. In particular, we define the *age* of a cached content from item n as the number of refreshes that item n has received in the database and not reflected on the content in the cache. If the user, thus, consumes a content from the cache with age k, the user will incur a freshness cost of $k \cdot C_0$, where $C_0 \geq 0$ is a constant showing the freshness cost per stale version. If the requested data is not in the cache, the user has to fetch the data from the database and incur a constant fetching cost of $C_f \geq 0$.

In this paper, we will study the caching strategies to minimize the overall system cost in presence of dynamically refreshing content which adversely impacts the caching utility. We will investigate *which* items to cache and how many items to cache for the single-user and multi-user scenarios.

Single-user scenario concerns a user with a limited cache space that keeps local copies of the dynamic content for local-access. If the requested item is in the local cache, it is directly served with the possible age-cost described above. In order to prevent the age-cost from dominating the overall cost, the local cache needs to check for updates of stored content at appropriate rates. Therefore, in this scenario the questions of interest are which data items are worth storing and at what rate their updates must be checked to minimize the overall cost. We will address this question in Section III.

Multi-user scenario, in contrast, concerns the distributed caching setting whereby each user receives its independent requests for the dynamic content for local consumption. The key new component in this case is the broadcast nature of the wireless medium whereby transmissions of content made to one user can be received and used to opportunistically update content in other users' cache at no additional transmission cost. This *multicasting property* non-trivially couples the decisions across the distributed cache space for optimal caching solution. In Section IV, we undertake this interesting

setting to provide optimal distributed allocation strategy for minimum overall cost.

In both the single and multi-user cases, we prove the optimality characteristics of our proposed caching and update strategies, and compare their gains over natural benchmarks that do not account for the dynamic nature of the content. In Section IV, we also compare the optimal solutions for the single and multi-user scenarios for equal request rates and equal total cache spaces in order to reveal the benefits of distributed caching over common caching that emerge due to the dynamic nature of the content.

III. Optimal Caching and Updating for Dynamic Content: Single-User Scenario

In this scenario, the user requests are served individually and no other user can benefit from such a service. Therefore, we drop the dependence on the user index m, i.e., the user generates requests with a rate of β. The cache size at the user is K data items. To avoid excessive freshness cost, the user employs a cache *check and update* mechanism through which the user generates random cache check and update requests to check the items in the cache and update them from the database if they have been already refreshed in the database. We assume that the cache check and update requests are generated according to a Poisson process with rate $\mu \geq 0$. Each checking request costs an amount $C_{ch} \geq 0$ which accounts for the communication overhead with the database. If the content in the cache is found to be not the most updated version, then the user will fetch the most fresh version from the database at an additional caching cost of $C_{ca} \geq 0$ which accounts for the resource consumption to deliver the fresh content to the user's cache. As discussed earlier, if an older content with age k is served from cache, the user will incur a freshness cost of $k \cdot C_0$, where $C_0 \geq 0$ is a constant. If the requested data is not in the cache, the user has to urgently fetch the data from the back-end database at a higher fetching cost of $C_f \geq 0$. The checking, caching and urgent fetching costs are constants and satisfy the relation $C_{ch} \leq C_{ca} \leq C_f$.

A. Problem Formulation

Let $\mathcal{I}_K \subseteq \mathcal{N}$ be the set of items that are stored in the user cache and let μ be the checking rate of cache content for the freshness. Note that K is the caching capacity of the user and due to the high refresh rate, the user may not necessarily fill the cache. As such $|\mathcal{I}_K| \leq K$.

Lemma 1: Let $C^{\mathcal{S}}_{\mathcal{I}_K}(\mu)$ be the average system cost in the $\mathcal{S}$ingle-user scenario as the user caches the set of items $\mathcal{I}_K$ and checks the cache freshness with the Poisson process of rate of μ. Then:

$$C^{\mathcal{S}}_{\mathcal{I}_K}(\mu) = \beta C_f + |\mathcal{I}_K|\mu C_{ch} + \mu C_{ca} \sum_{i\in\mathcal{I}_K} \frac{\lambda_i}{\lambda_i+\mu} + \beta \sum_{i\in\mathcal{I}_K} p_i \left(\frac{\lambda_i C_0}{\mu} - C_f\right), \quad (1)$$

Proof. Let $\{\Pi^i_\mu(t), t \geq 0\}, \forall i \in \mathcal{I}_K$ be the Markov process describing the freshness age of cached item i at time t under a given checking rate μ. The evolution of this process is shown in Fig. 2.

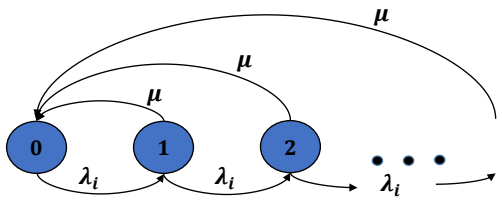

Fig. 2: Markov chain diagram for freshness $\{\Pi^i_\mu(t), t \geq 0\}$ under checking rate μ.

As it can be seen in Fig. 2, every arriving content update to the item i in the database that occurs with rate λ_i increases the age of that item in the cache by one. Checking and updating the cache content will occur with rate μ and upon occurrence, it will move the system back to state zero, the most fresh version. We are interested in the limit of $\Pi^i_\mu(t) \xrightarrow[t\to\infty]{d} \bar{\Pi}^i_\mu$, i.e., the steady state distribution of $\Pi^i_\mu(t)$. Let $\pi^i_k(\mu) = P(\bar{\Pi}^i_\mu = k), k \in \{0,1,2,\ldots\}$ be the probability of item $i \in \mathcal{I}_K$ having the age of k under the checking rate μ, then:

$$\pi^i_k(\mu) = \pi^i_0(\mu)\left(\frac{\lambda_i}{\lambda_i+\mu}\right)^k, \forall k \in \{0,1,2,\ldots\}. \quad (2)$$

Setting $\sum_{k=0}^{\infty} \pi^i_k(\mu) = 1$, gives $\pi^i_0(\mu) = \frac{\mu}{\lambda_i+\mu}$. Hence, the average age of any item $i \in \mathcal{I}_K$ in the cache is given by:

$$\mathbb{E}[\bar{\Pi}^i_\mu] = \sum_{k=0}^{\infty} k\pi^i_k(\mu) = \frac{\lambda_i}{\mu}. \quad (3)$$

The average system cost in the $\mathcal{S}$ingle-user scenario as the user caches the set of items $\mathcal{I}_K$ and checks the cache freshness with rate μ comprises four main terms as follows:

$$C^{\mathcal{S}}_{\mathcal{I}_K}(\mu) = |\mathcal{I}_K|\mu C_{ch} + \beta C_f\left(1 - \sum_{i\in\mathcal{I}_K} p_i\right) + \mu C_{ca} \sum_{i\in\mathcal{I}_K} \left(1 - \pi^i_0(\mu)\right) + \beta C_0 \sum_{i\in\mathcal{I}_K} p_i \mathbb{E}[\bar{\Pi}^i_\mu]. \quad (4)$$

The first term in Equation (4) shows the average *checking cost* for a cache capacity of K that caches the set of items $\mathcal{I}_K$ and updates the cache content with the rate of μ and each checking process has a cost of C_{ch}. The second term in Equation (4) shows the average *fetching cost* for a cache set of $\mathcal{I}_K$ and the request arrival rate of β as a function of miss rate $\beta(1-\sum_{i\in\mathcal{I}_K})$. For any arrival request, $1-\sum_{i\in\mathcal{I}_K} p_i$ is the probability that the requested content is not in the cache, so the content should be fetched from the database which incurs the cost of C_f.

The third term in Equation (4) shows the average *caching cost* for a cache set of $\mathcal{I}_K$ and checking rate of μ. For a given μ, $\pi^i_0(\mu), \forall i \in \mathcal{I}_K$ is the probability that item i in the cache is the most updated version, i.e., has age 0. So $1-\pi^i_0(\mu)$ is the probability that item i existing in the cache is not fresh. For every checking process that happens with rate μ, if the content in the cache is not fresh, we cache the

most updated version from the database and put it in the user cache which incurs the cost of C_{ca}.

The fourth term in Equation (4) shows the average *freshness cost* for a cache set of $\mathcal{I}_K$ and checking rate of μ. For each item $i \in \mathcal{I}_K$ existing in the cache, the arrival request will be served from the cache. The arrival request of item i is βp_i and since the item with age k incurs the cost of $k.C_0$, so the average cost of freshness will be $C_0\mathbb{E}[\bar{\Pi}^i_\mu]$ which $\mathbb{E}[\bar{\Pi}^i_\mu]$ is the average age of cached item i given in (3).

Replacing the results of Equations (2) and (3) in the cost function given in Equation (4) completes the proof. ■

The cost minimization problem for the single-user scenario would thus be:

$$\min_{\mu\geq 0, \mathcal{I}_K \subseteq \mathcal{N}} C^{\mathcal{S}}_{\mathcal{I}_K}(\mu). \tag{5}$$

A traditional (suboptimal) approach to tackle the caching problem (5) is to cache the first K most popular items.

Definition 1 (Cache the Most Popular): Define the $\mathcal{I}^p_K \subseteq \mathcal{N}$ to be the set of K most popular items. That is,

$$\mathcal{I}^p_K := \{i \in \mathcal{N} : |\mathcal{I}^p_K| = K, p_i \geq p_n \forall i \in \mathcal{I}^p_K, n \in \mathcal{N}\backslash\mathcal{I}^p_K\}.$$

Then the cache the most popular strategy will assign the cached set of items as $\mathcal{I}_K = \mathcal{I}^p_K$ and optimizes the cache check and update rate as $\mu = \mu^p$, where

$$\mu^p := \arg\min_{\mu\geq 0} C^{\mathcal{S}}_{\mathcal{I}^p_K}(\mu).$$

Since the cost in (1) is convex over μ, such μ_p exists.

The cache most popular strategy does not consider the content refresh rate, and the associated freshness costs. Hence it is a suboptimal strategy. We then note that, the optimization (5) is computationally formidable to solve as it necessitates a discrete search process which involves finding the jointly optimal subset of items to be cached from a large database of N items and the best cache check and update rate. We, therefore, investigate the design of suboptimal, yet simpler caching strategies that account for the dynamic content refreshing and lead to more performance merits than the traditional cache most popular strategy.

B. Proposed Algorithm

We propose an algorithm, Algorithm 1, with a selected set of cached items $\hat{\mathcal{I}}_K$ and a cached check and update rate $\hat{\mu}$, based on the refreshing rate of $\boldsymbol{\lambda}$ and other system parameters to minimize the expected system cost.

In particular, and as used in Algorithm 1, for item i, we define the metric $\delta^{\mathcal{S}}(\lambda_i, p_i, \mu) = \delta^{\mathcal{S}}_i(\mu)$ as follows:

$$\delta^{\mathcal{S}}_i(\mu) := \mu C_{ch} + \frac{\mu\lambda_i}{\lambda_i+\mu}\mathrm{C_{ca}} + \beta p_i\left[C_0\frac{\lambda_i}{\mu} - C_f\right], \forall i \in \mathcal{N}$$

to capture the marginal cost of adding the item i to the cache for a given μ.

Our proposed algorithm sorts the items based on $\delta^{\mathcal{S}}_i(\mu)$ and starts filling the cache with items that have the least $\delta^{\mathcal{S}}_i(\mu)$ and keeps adding until either all the items with negative $\delta^{\mathcal{S}}_i(\mu)$ are placed in the cache or the cache becomes full, i.e., K items have been already cached. Then for the new set of cached items, the algorithm computes the corresponding optimal cache check and update rate $\hat{\mu}$ and modifies the values of $\delta^{\mathcal{S}}_i(\hat{\mu})$ based on new $\hat{\mu}$.

Algorithm 1 Single-user caching strategy

Input: $P = (p_1, ..., p_N), \lambda = (\lambda_1, ..., \lambda_N), \mu^p, \mathcal{I}^p_K$
Initialization : $\hat{\mathcal{I}}_K = \emptyset$, $\mathcal{I}^{Old}_K = \mathcal{I}^p_K$
1: Set $\hat{\mu} = \mu_p$
2: Compute $\delta^{\mathcal{S}}_i(\hat{\mu}) = \hat{\mu}C_{ch} + \frac{\hat{\mu}\lambda_i}{\lambda_i+\hat{\mu}}\mathrm{C_{ca}} + \beta p_i\left[C_0\frac{\lambda_i}{\hat{\mu}} - C_f\right], \forall i \in \mathcal{N}$
3: Update $\hat{\mathcal{I}}_K$ as follows:

$$\hat{\mathcal{I}}_K = \{i \in \mathcal{N} : |\hat{\mathcal{I}}_K| \leq K, \delta^{\mathcal{S}}_i(\hat{\mu}) < 0, \delta^{\mathcal{S}}_i(\hat{\mu}) \leq \delta^{\mathcal{S}}_n(\hat{\mu}), \forall i \in \hat{\mathcal{I}}_K, n \in \mathcal{N}\backslash\hat{\mathcal{I}}_K\}.$$

4: **while** $\hat{\mathcal{I}}_K \neq \mathcal{I}^{Old}_K$ **do**
5: $\mathcal{I}^{Old}_K = \hat{\mathcal{I}}_K$
6: $\hat{\mu} = \arg\min_{\mu\geq 0} C^{\mathcal{S}}_{\hat{\mathcal{I}}_K}(\mu)$
7: Update $\delta^{\mathcal{S}}_i(\hat{\mu})$ from step 2.
8: Update $\hat{\mathcal{I}}_K$ from step 3.
9: **end while**
10: **return** $\hat{\mu}$, $\hat{\mathcal{I}}_K$.

Notice that all data items with positive $\delta^{\mathcal{S}}_i(\mu)$ can only increase the average cost if cached. The metric $\delta^{\mathcal{S}}_i(\mu)$ reveals the effect of refresh rate alongside the popularity on gains that can be achieved by caching an item. For example, if an item has a high probability of being requested and a high refresh rate, the high refresh rate will increase the values of $\delta^{\mathcal{S}}_i(\mu)$ and therefore renders that item less likely to be cached even if there is available cache storage.

C. Performance Analysis

In the following, we provide a proof of optimality for the proposed algorithm under a given cache check and update rate μ and show that it always outperforms the cache most popular content strategy.

Proposition 1: For a given cache check and update rate μ, Algorithm 1 minimizes the average cost in (1).

Proof. For a given μ and the set of items $\mathcal{I}_K$ in the cache, if we add any item i to the cache such that $i \notin \mathcal{I}_K$, then we can write the resulting cost as:

$$C^{\mathcal{S}}_{\mathcal{I}_K\cup\{i\}}(\mu) = C^{\mathcal{S}}_{\mathcal{I}_K}(\mu) + \delta^{\mathcal{S}}_i(\mu) \quad \forall i \notin \mathcal{I}_K$$

By induction, if we set $\mathcal{I}_K = \{\emptyset\}$ and add the item i to the cache, the cost will decrease by $\delta^{\mathcal{S}}_i(\mu)$. If we keep adding items i with $\delta^{\mathcal{S}}_i(\mu) < 0$, the average cost will continue to decrease. Therefore:

$$C^{\mathcal{S}}_{\mathcal{I}_K}(\mu) = C^{\mathcal{S}}_{\{\emptyset\}}(\mu) + \sum_{i\in\mathcal{I}_K}\delta^{\mathcal{S}}_i(\mu)$$

Since the proposed algorithm at each step chooses the items with minimum negative $\delta^{\mathcal{S}}_i(\mu)$ for a given μ, it results in the optimal solution. ■

Proposition 2: The proposed algorithm, Algorithm 1, always outperforms the cache most popular strategy, i.e.,

$$C^{\mathcal{S}}_{\hat{\mathcal{I}}_K}(\hat{\mu}) \leq C^{\mathcal{S}}_{\mathcal{I}^p_K}(\mu^p)$$

Proof. We prove this by showing that in each iteration of the proposed algorithm, the resulting average cost decreases. Proposition 1, suggests that for a given μ, our algorithm gives the optimal solution. At any given iteration t, we have:

$$C^{\mathcal{S}}_{\hat{\mathcal{I}}_K(t)}(\hat{\mu}(t)) \geq C^{\mathcal{S}}_{\hat{\mathcal{I}}_K(t+1)}(\hat{\mu}(t)).$$

Since at each step we choose $\hat{\mu}(t+1)$ to minimize the average cost for a given $\hat{\mathcal{I}}_K(t+1)$, in other words, $\mu(t+1) = \underset{\mu}{\operatorname{argmin}}\, C_{\mathcal{I}_K(t+1)}(\mu)$, we have:

$$C^{\mathcal{S}}_{\hat{\mathcal{I}}_K(t+1)}(\hat{\mu}(t)) \geq C^{\mathcal{S}}_{\hat{\mathcal{I}}_K(t+1)}(\hat{\mu}(t+1)).$$

Combining the two equations gives:

$$C^{\mathcal{S}}_{\hat{\mathcal{I}}_K(t)}(\hat{\mu}(t)) \geq C^{\mathcal{S}}_{\hat{\mathcal{I}}_K(t+1)}(\hat{\mu}(t+1)),$$

which shows at each iteration, the proposed algorithm reduces the cost. Since we start the algorithm with $\hat{\mu}(1) = \mu^p$ and $\hat{\mathcal{I}}_K(1) = \mathcal{I}^p_K$, so the proposed algorithm always outperforms cache the most popular strategy. ■

We next investigate the efficiency of our algorithm compared to cache the most popular strategy.

D. Numerical Investigation

We let the total number of data items be $N = 10^6$, data items' popularity be $p_n = c/n^{\alpha}$ with $\alpha = 1.2$ and content refresh rates be $\lambda_n = \lambda/n^z$, for some $z \geq 0$. We consider the normalized costs of fetching, checking, caching and freshness to be $C_f = 1, C_{ca} = 0.1, C_{ch} = 0.05, C_0 = 0.025$.

Setting the cache size K to be 500, we compare the average cost achieved by the proposed algorithm, Algorithm 1, and the average cost of cache the most popular items strategy under the same system variables declared above. We adopt the percentage cost reduction of our proposed algorithm to the cache the most popular strategy's cost as our performance metric. Such a metric is defined as:

$$\text{Cost Reduction}(\%) = 100 \times \frac{C^{\mathcal{S}}_{\mathcal{I}^p_K}(\mu^p) - C^{\mathcal{S}}_{\hat{\mathcal{I}}_K}(\hat{\mu})}{C^{\mathcal{S}}_{\mathcal{I}^p_K}(\mu^p)}.$$

The percentage cost reduction is depicted in Fig. 3. The figure shows substantial gains (between $50-90\%$ reduction in the cost) compared to the predominant popularity-based design, are achievable with our proposed preliminary design. It also reveals that the gains become more substantial as the refresh rate of different items becomes more non-uniform (as the parameter z increases).

Note that adding cache capacity to users is not always an effective way to reduce the average system cost, specially in presence of highly dynamic content.

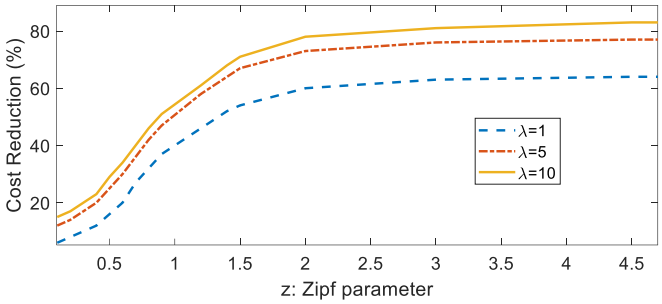

Fig. 3: Average cost reduction by the proposed algorithm over the cache the most popular for the single-user scenario.

IV. Optimal Caching and Updating for Dynamic Content: Multi-User Scenario

Consider the scenario shown in Fig. 1, with M users. To gain a clear insight of the potential *wireless multicasting* gain and how distributed caching can relate to the single-user scenario with a cache of size K, we assume that each user in the multi-user scenario has the capacity to cache *only one* of the date items. However, the number of users is set equal to the number of items that the user can cache in the single-user scenario. That is, $M = K$. In other words, we distribute the K caching capacity over the users with each user can cache one item.

In this section, we investigate what items to be cached and how should the cached items be replicated over the set of users. In the multi-user scenario, due to the broadcast capability of wireless service, it is not necessary to employ a cache check and update mechanism as is the case in the single-user scenario. Instead, users that have a certain item in their cache can update it for free if another user that does not have it, requests its most fresh version from the database.

Let $\mathbf{r} = (r_1, ..., r_N)$ be the vector of the number of times each item has been cached among the K users. In other words, r_i is the number of replicas of item i that exist in the users' caches. Also recall that C_0 is the freshness cost per an age unit. As the age of a cached content increases, the freshness cost grows linearly. The average cost of urgently fetching a data item from the database is C_f and the the freshness cost of consuming an item from the cache is $k.C_0$ where k is the age of the cached content.

A. Problem Formulation

For K users, each equipped with one cache, let $\mathbf{r} = (r_1, ..., r_N)$ be the vector of replication. Define the feasible set of solutions as:

$$\mathcal{F}_K = \left\{ \mathbf{r} = (r_1, \ldots, r_N) \,|\, \sum_{i=1}^{n} r_i \leq K, r_i \in \{0, 1, 2 \ldots\} \right\}.$$

Lemma 2: Let $C^{\mathcal{M}}(\beta, \mathbf{r})$ be the average expected system cost in the $\mathcal{M}$ulti-user scenario with K users and request arrival rate of β under vector of replication $\mathbf{r} \in \mathcal{F}_K$. Then:

$$C^{\mathcal{M}}(\mathbf{r}) = K\beta C_f + \sum_{i=1}^{N} r_i \left(\frac{C_0 \lambda_i}{K - r_i} - \beta p_i C_f \right). \quad (6)$$

Proof. Let $\{\Pi^i_{r_i}(t), t \geq 0\}$, $\forall i \in \mathcal{N}$ be the Markov process describing the freshness age of cached item i at time t under the number of replicas r_i. The evolution of this process is shown in Fig. 4. As discussed earlier, in

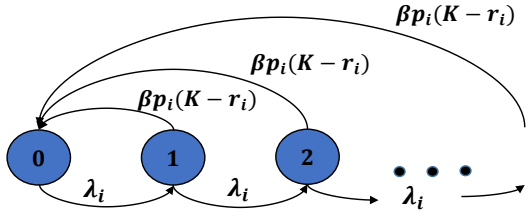

Fig. 4: Markov chain diagram for freshness $\{\Pi^i_{r_i}(t), t \geq 0\}$ under the number of replicas r_i.

the multi-user scenario, the broadcast capability of wireless service acts as a natural update mechanism. In other words, users update their cached content for free by overhearing that content while being served to other users who do not have it in their cache. For any item i in the cache, since there are K users and r_i of them have item i in their cache, the service rate of item i is equal to $\beta p_i(K - r_i)$. As it can be seen in Fig. 4, every service of item i acts as an update mechanism for the users that hold item i in their cache and upon occurrence, the service of item i from the database will move the system back to state zero, the most fresh version. Every arriving content update to the item i in the database that occurs with rate λ_i increases the age of that item in the cache by one.

Letting $\Pi^i_{r_i}(t) \xrightarrow[t\to\infty]{d} \bar{\Pi}^i_{r_i}$ and using the steady state distribution of $\Pi^i_{r_i}(t)$ define $\pi^i_k(r_i) = P(\bar{\Pi}^i_{r_i} = k), k \in \{0, 1, 2, \ldots\}$ be the probability of item i having the age of k under the number of replicas r_i, then the average age of item i is given by:

$$\mathbb{E}[\bar{\Pi}^i_{r_i}] = \frac{\lambda_i}{\beta p_i(K - r_i)}. \tag{7}$$

The average system cost in the $\mathcal{M}$ulti-user scenario as K users cache according to the vector of replication $\mathbf{r} = (r_1, ..., r_N) \in \mathcal{F}_K$, comprises two main terms and is given by:

$$C^{\mathcal{M}}(\mathbf{r}) = \beta C_f \left(K - \sum_{i=1}^{N} r_i p_i\right) + \beta C_0 \sum_{i=1}^{n} p_i r_i \mathbb{E}[\bar{\Pi}^i_{r_i}]. \tag{8}$$

The first term in Equation (8), shows the average *fetching cost* for any $\mathbf{r} \in \mathcal{F}_K$ and request arrival rate β as a function of miss rate $\beta\left(K - \sum_{i=1}^{N} r_i p_i\right)$. For any of the K users, if a requested item is in the user's cache, it will be immediately served from the cache with the freshness cost, otherwise it will be fetched from the database and the urgent fetching cost C_f is incurred. Since there are r_i users that have item i in their cache, the miss rate for item i is $\beta r_i(1 - p_i)$. Summing over all the N items and remembering that $\mathbf{r} \in \mathcal{F}_K$, gives the total miss rate as $\beta\left(K - \sum_{i=1}^{N} r_i p_i\right)$.

The second term in Equation (8), shows the average *freshness cost* for any $\mathbf{r} \in \mathcal{F}_K$ and request arrival rate of β. For each item i in the cache, the arrival request rate is βp_i and since the item with age k incurs the cost of $k \cdot C_0$, so the average cost of freshness for item i will be $C_0\mathbb{E}[\bar{\Pi}^i_{r_i}]$. Since r_i is the number of users having item i in their cache, the total freshness cost incurred by item i is given by $\beta p_i r_i C_0 \mathbb{E}[\bar{\Pi}^i_{r_i}]$. Summing over all the items gives the total freshness cost of the system. Substituting Equation (7) in Equation (8) gives the average cost of system. ■

Algorithm 2 Multi-user caching strategy

Input: $\mathbf{p} = (p_1, ..., p_N), \lambda = (\lambda_1, ..., \lambda_N), K$
Initialization : $r^*_i = 0 \quad \forall i \in \mathcal{N}$
1: Calculate $\delta^{\mathcal{M}}_i(r^*_i) = \frac{KC_0\lambda_i}{(K-r^*_i)(K-r^*_i-1)} - \beta p_i C_f \quad \forall i \in \mathcal{N}$.
2: $j = \arg\min_{i\in\mathcal{N}} \delta^{\mathcal{M}}_i(r^*_i)$
3: **while** $\delta^{\mathcal{M}}_j(r^*_j) < 0$ and $\sum_{i=1}^{N} r^*_i < K$ **do**
4: $r^*_j = r^*_j + 1$
5: update $\delta_j(r^*_j)$ from Step 1.
6: update $j = \arg\min_{i\in\mathcal{N}} \delta^{\mathcal{M}}_i(r^*_i)$
7: **end while**
8: **return** $\mathbf{r}^* = (r^*_1, \ldots, r^*_N)$

Our objective is thus to choose the content to be cached at the users in order to minimize the average cost of system, that is:

$$\arg\min_{\mathbf{r}\in\mathcal{F}_K} C^{\mathcal{M}}(\mathbf{r}). \tag{9}$$

The traditional cache the most popular strategy in this context reduces to caching the K most popular items[1] to the users' caches, one item per user cache.

Definition 2 (Cache the Most Popular): Define the $\mathcal{I}^p_K \subseteq \mathcal{N}$ to be the set of K most popular items. Then cache the most popular strategy for the K users, each with a unit caching capacity, is given by:

$$r^p_i := \begin{cases} 1, & i \in \mathcal{I}^p_K, \\ 0, & i \in \mathcal{N} \setminus \mathcal{I}^p_K, \end{cases}$$

with $\mathbf{r}^p := (r^p_1, ..., r^p_N)$.
Such strategy does not consider the freshness of items, yet we address the question of whether the system can achieve better performance through lower cost.

B. Proposed Algorithm

We propose Algorithm 2 based on the data items refresh rate $\boldsymbol{\lambda}$ to solve (9).

In particular, as it can be seen in Algorithm 2, for item i, we define the metric $\delta^{\mathcal{M}}(\lambda_i, p_i, l) = \delta^{\mathcal{M}}_i(l)$ as follows:

$$\delta^{\mathcal{M}}_i(l) := \frac{KC_0\lambda_i}{(K-l)(K-l-1)} - \beta p_i C_f \quad \forall i \in \mathcal{N}. \tag{10}$$

The metric $\delta^{\mathcal{M}}_i(l)$ captures the marginal cost of adding item i to the cache given that l of the users have already cached item i. Our proposed algorithm, at each step, sorts the

[1]Note that caching the same item at all users (i.e., setting $r_i = K$ for some $i \in \mathcal{N}$, $r_j = 0$,$\forall j \neq i$) can only result in an infinite cost due to the fact that the item cached will never be requested from the database yielding a freshness cost that grows indefinitely.

items based on $\delta_i^{\mathcal{M}}(l)$, caches the item with the minimum $\delta_i^{\mathcal{M}}(l)$ and iterates until either all the items with negative $\delta_i(l)$ are cached or there no more users are available to cache more items (i.e., no available cache storage). Complexity of proposed algorithm is similar to the sort algorithm.

Notice that items with positive $\delta_i^{\mathcal{M}}(l)$ can only increase the average cost if cached. Similar to single user-scenario, $\delta_i^{\mathcal{M}}(l)$ reveals the effect of refresh rate alongside the popularity on gains that can be achieved by caching an item.

C. Performance Analysis

In the following, we provide a proof of optimality for the proposed caching algorithm by showing that $\mathbf{r}^*$ satisfies all the necessary conditions for optimality.

Theorem 1: Algorithm 2 solves the problem (9) optimally.

Proof. We start the proof by first discussing the necessary conditions for the optimal solution.

Lemma 3 (Necessary conditions for optimality): Any optimal solution $\bar{\mathbf{r}} = (\bar{r}_1, \ldots, \bar{r}_N)$ to the problem defined in Equation (9) must satisfy all the following conditions.

$$\delta_i^{\mathcal{M}}(\bar{r}_i - 1) \leq 0 \quad \forall i \in \mathcal{N}, \text{ with } \bar{r}_i > 0, \tag{11}$$

$$\delta_i^{\mathcal{M}}(\bar{r}_i - 1) \leq \delta_j^{\mathcal{M}}(\bar{r}_j) \quad \forall j \neq i, \text{ with } \bar{r}_i > 0, \tag{12}$$

$$\sum_{i=1}^{n} \bar{r}_i = K \text{ or } \delta_i^{\mathcal{M}}(\bar{r}_i) > 0 \; \forall i \in \mathcal{N}. \tag{13}$$

Proof. We use contradictions to prove that all the three conditions are necessary for the optimal solution.

To prove that Equation (11) is necessary for optimality, we use contradiction. Assume that Equation (11) does not hold, so there exists $j \in \mathcal{N}$ with $\bar{r}_j > 0$ such that $\delta_j^{\mathcal{M}}(\bar{r}_j - 1) > 0$. Then construct $\mathbf{r} = \bar{\mathbf{r}} - e_j$, where $\mathbf{r} \in \mathcal{F}_K$ and we have that $C^{\mathcal{M}}(\mathbf{r}) = C^{\mathcal{M}}(\bar{\mathbf{r}}) - \delta_j^{\mathcal{M}}(\bar{r}_j - 1)$. Since $\delta_j^{\mathcal{M}}(\bar{r}_j - 1) > 0$, so $C^{\mathcal{M}}(\mathbf{r}) < C^{\mathcal{M}}(\bar{\mathbf{r}})$ which contradicts the fact that $\bar{\mathbf{r}}$ was the optimal solution.

To prove that Equation (12) is necessary for optimality, assume that there exist $i, j \in \mathcal{N}$ such that $\delta_i^{\mathcal{M}}(\bar{r}_i - 1) > \delta_j^{\mathcal{M}}(\bar{r}_j)$. Then construct $\mathbf{r} = \bar{\mathbf{r}} - e_i + e_j$, where $\mathbf{r} \in \mathcal{F}_K$ and we have that $C^{\mathcal{M}}(\mathbf{r}) = C^{\mathcal{M}}(\bar{\mathbf{r}}) - \delta_i(\bar{r}_i - 1) + \delta_j(\bar{r}_j)$. So there exists $\mathbf{r} \in \mathcal{F}_K$ with $C^{\mathcal{M}}(\mathbf{r}) < C^{\mathcal{M}}(\bar{\mathbf{r}})$ which contradicts the fact that $\bar{\mathbf{r}}$ was the optimal solution.

To prove that Equation (12) is necessary for optimality, assume that $\sum_{i=1}^{N} \bar{r}_i < K$ and $j \in \mathcal{N}$ such that $\delta_j^{\mathcal{M}}(\bar{r}_j) < 0$. Construct $\mathbf{r} = \bar{\mathbf{r}} - e_j$, where $\mathbf{r} \in \mathcal{F}_K$ and we have that $C^{\mathcal{M}}(\mathbf{r}) = C^{\mathcal{M}}(\bar{\mathbf{r}}) + \delta_j^{\mathcal{M}}(\bar{r}_j)$. Since $\delta_j^{\mathcal{M}}(\bar{r}_j) < 0$, so there exists $\mathbf{r} \in \mathcal{F}_K$ with $C^{\mathcal{M}}(\mathbf{r}) < C^{\mathcal{M}}(\bar{\mathbf{r}})$ which contradicts the fact that $\bar{\mathbf{r}}$ was the optimal solution. ■

Now we prove that any solution $\mathbf{r} \in \mathcal{F}_K$ to the optimization problem defined in Equation (9) that satisfies all the necessary conditions for optimality given in Lemma 3, results in the same average cost.

Lemma 4: Any solution $\mathbf{r} \in \mathcal{F}_K$ satisfying the Equations (11), (12) and (13) will result in the same average cost.

Proof. To prove the lemma, we show that for any $\mathbf{r}, \bar{\mathbf{r}} \in \mathcal{F}_K$ and arbitrary $\mathbf{a} = (a_1, \ldots, a_N)$ such that $\mathbf{r} = \bar{\mathbf{r}} + \mathbf{a}$ if both $\mathbf{r}$ and $\bar{\mathbf{r}}$ satisfy the conditions of Lemma 3, then either $\mathbf{a} = 0$ or $C^{\mathcal{M}}(\mathbf{r}) = C^{\mathcal{M}}(\bar{\mathbf{r}})$. Assume $\mathbf{a} \neq 0$, we consider two cases separately.

Case 1: if $\delta_i^{\mathcal{M}}(\bar{r}_i) > 0 \; \forall i \in \mathcal{N}$, then if there exists j such that $a_j > 0$, we have that:

$$\delta_j^{\mathcal{M}}(r_j - 1) = \delta_j^{\mathcal{M}}(\bar{r}_j + a_j - 1) > \delta_j^{\mathcal{M}}(\bar{r}_j)$$

So Equation (11) does not hold for $\mathbf{r}$, which is a contradiction. Hence $a_i \leq 0 \; \forall i \in \mathcal{N}$. If $a_i = 0 \; \forall i$ then the problem is solved, but if there exists j such that $a_j < 0$, then we have:

$$\delta_j^{\mathcal{M}}(r_j) = \delta_j^{\mathcal{M}}(\bar{r}_j + a_j) \leq \delta_j(\bar{r}_j - 1) \leq 0. \tag{14}$$

According to Equation (13), $\sum_{i=1}^{n} r_i = K$ should hold for $\mathbf{r}$, but $\sum_{i=1}^{N} r_i = \sum_{i=1}^{N} \bar{r}_i + \sum_{i=1}^{n} a_i < K$, since $\sum_{i=1}^{N} \bar{r}_i \leq K$ and $\sum_{i=1}^{N} a_i < 0$, which is a contradiction.

Case 2: If $\delta_i^{\mathcal{M}}(\bar{r}_i) > 0$ does not hold for all $i \in \mathcal{N}$, then according to Equation (13), $\sum_{i=1}^{N} \bar{r}_i = K$ should hold. Since $\sum_{i=1}^{N} r_i = \sum_{i=1}^{N} \bar{r}_i + \sum_{i=1}^{N} a_i \leq K$, then $\sum_{i=1}^{N} a_i \leq 0$. If $a_i \leq 0$ for all i, then in order to have $\mathbf{a} \neq 0$, there exists $j \in \mathcal{N}$ with $a_j < 0$ such that Equation (14) holds. Now, from Equation (13), $\sum_{i=1}^{N} r_i = K$ should hold for $\mathbf{r}$, but since $\sum_{i=1}^{N} a_i < 0$, it is not possible. So if there exists $j \in \mathcal{N}$ with $a_j < 0$, there must exist $v \in \mathcal{N}$ with $a_v > 0$ such that $\sum_{i=1}^{N} a_i = 0$ since we should have $\sum_{i=1}^{N} r_i = K$ as shown before. Since $\mathbf{r}$ satisfies all the necessary conditions of Lemma 3, Equation (12) holds for $\mathbf{r}$ over v and j.
$\delta_v^{\mathcal{M}}(\bar{r}_v) \leq \delta_v^{\mathcal{M}}(\bar{r}_v + a_v - 1) \leq \delta_j^{\mathcal{M}}(\bar{r}_j + a_j) \leq \delta_j^{\mathcal{M}}(\bar{r}_j - 1)$.

Now if $\delta_v^{\mathcal{M}}(\bar{r}_v) < \delta_j^{\mathcal{M}}(\bar{r}_j - 1)$, the condition of Equation (12) does not hold for $\bar{\mathbf{r}}$ which is a contradiction and if $\delta_v^{\mathcal{M}}(\bar{r}_v) = \delta_j(\bar{r}_j - 1)$, construct the $\mathbf{r} = \bar{\mathbf{r}} + e_v - e_j$. Then $C^{\mathcal{M}}(\mathbf{r}) = C^{\mathcal{M}}(\bar{\mathbf{r}}) + \delta_v^{\mathcal{M}}(\bar{r}_v) - \delta_j^{\mathcal{M}}(\bar{r}_j - 1) = C^{\mathcal{M}}(\bar{\mathbf{r}})$. which completes the proof. ■

The solution reached by Algorithm 2 satisfies all the necessary conditions in Lemma 3 and according to Lemma 4, such a solution is optimal. ■

It is worth noting that the cache allocation strategy for the multi-user scenario supported with wireless multicasting can lead to some users caching less popular items than those cached at other users. Such a diversity in cached iterms' popularities empowers the need for requests from the database which in turn brings the most recent version of content to users for free, thanks to wireless multicasting.

Knowing that the proposed algorithm, Algorithm 2, gives the optimal solution, we investigate its performance merits compared to other basic caching strategies like cache the most popular strategy.

D. Numerical Investigations

Using the same parameter values defined in Section III.D with $z = 1.2$ and changing the number of users K, we set the performance metric to be the percentage cost reduction of our proposed algorithm, Algorithm 2, to cache the most popular strategy's cost. Define:

$$\text{Cost Reduction}(\%) = 100 \times \frac{C^{\mathcal{M}}(\mathbf{r}^p) - C^{\mathcal{M}}(\mathbf{r}^*)}{C^{\mathcal{M}}(\mathbf{r}^p)},$$

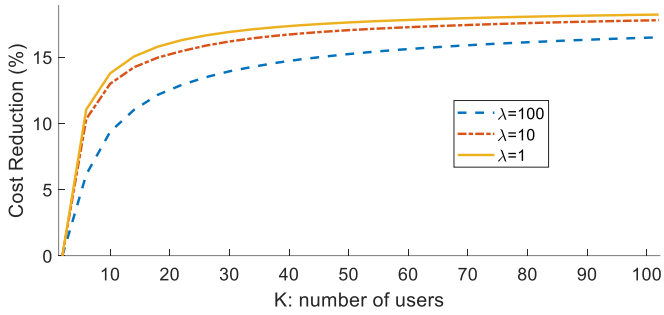

Fig. 5: Average cost reduction by the proposed algorithm over the cache the most popular for the multi-user scenario.

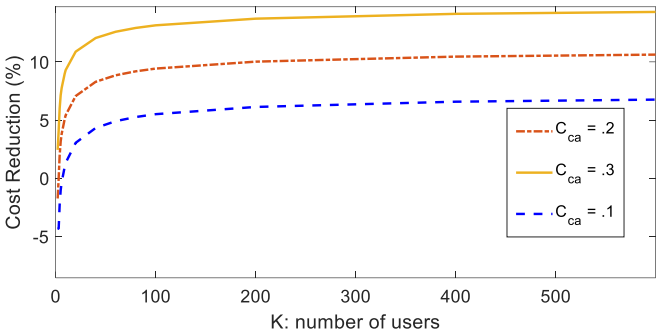

Fig. 6: Average cost reduction per user by distributing cache space between multiple users.

The percentage cost reduction is depicted in Fig. 5. The figure shows considerable gains compared to the predominant popularity-based design are achievable with our proposed preliminary design. It also reveals that the gains become more substantial as the number of users increases, reveling that proposed algorithm, Algorithm 2, can more effectively incorporate the broadcasting gain to reduce the cost. It also reveals that the gain increases as the refresh rate of different items decreases. Also as the refresh rate of different items decreases, with less users we can achieve higher gains, benefiting more from the multicasting gain.

We also compare the average cost of the single-user with the multi-user scenario. Running the Algorithm 1 for the single-user scenario and Algorithm 2 for the multi-user scenario, with $C^{\mathcal{S}}_{\hat{\mathcal{I}}_K}(\hat{\mu})$ and $C^{\mathcal{M}}(\mathbf{r}^*)/K$ representing the average cost per user for these two scenarios respectively. Setting the performance metric to be the percentage cost reduction per user for multi-user scenario compared to the single-user scenario's cost, we define:

$$\text{Cost Reduction}(\%) = 100 \times \frac{C^{\mathcal{S}}_{\hat{\mathcal{I}}_K}(\hat{\mu}) - C^{\mathcal{M}}(\mathbf{r}^*)/K}{C^{\mathcal{S}}_{\hat{\mathcal{I}}_K}(\hat{\mu})}.$$

Fig. 6 shows the percentage cost reduction for different caching costs of C_{ca}. According to the figure, for small cache size, single cache outperforms distributed caching in the sense of average cost per user, but as the number of users grows, the multi-user scenario, benefiting more through the multicasting property, will outperform the single-user scenario. In other words, through distributed caching aided with multicast cache update, the per-user cost in the multi-user system decreases as the number of users K grows, while the single-user's cost does not benefit from more cache storage, K, because of the associated cache check and update requests. Recall that the multi-user scenario, despite the single-user scenario, does not employ any optimized cache checking and updating mechanism.

V. Conclusion

In this work, we have proposed caching algorithms for wireless content distribution networks serving dynamically changing data content such as news updates, social network stories, and any other system with time-varying states. We have developed a design framework together with the performance analysis for efficient freshness-driven caching strategies. We have characterized the average operational cost both for the single-user and multi-user scenarios. Our results have revealed that, in the presence of dynamic content, adding more cache space to edge-users may solve the system congestion problem at the expense of a high freshness cost. In the multi-user scenario, as the number of users increases, our proposed algorithm benefits more from the multicasting property as a mechanism to update the cache content and outperforms single-user caching. Our results have also demonstrated that freshness-driven design considerably reduces the average cost and optimizes the cache space more effectively than the predominant existing strategies such as cache the most popular content.

References

[1] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *2010 Proceedings IEEE INFOCOM*. IEEE, 2010, pp. 1–9.

[2] A. Meyerson, K. Munagala, and S. Plotkin, "Web caching using access statistics," in *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2001, pp. 354–363.

[3] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, "Placement algorithms for hierarchical cooperative caching," *Journal of Algorithms*, vol. 38, no. 1, pp. 260–302, 2001.

[4] R. E. Craig, S. D. Ims, Y. Li, D. E. Poirier, S. Sarkar, Y.-s. Tan, and M. R. Villari, "Caching dynamic content," Jun. 29 2004, uS Patent 6,757,708.

[5] K. S. Candan, W.-S. Li, Q. Luo, W.-P. Hsiung, and D. Agrawal, "Enabling dynamic content caching for database-driven web sites," in *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, 2001, pp. 532–543.

[6] C. Yuan, Y. Chen, and Z. Zhang, "Evaluation of edge caching/off loading for dynamic content delivery," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 11, pp. 1411–1423, 2004.

[7] C. Kam, S. Kompella, G. D. Nguyen, J. E. Wieselthier, and A. Ephremides, "Information freshness and popularity in mobile caching," in *2017 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2017, pp. 136–140.

[8] V. S. Mookerjee and Y. Tan, "Analysis of a least recently used cache management policy for web browsers," *Operations Research*, vol. 50, no. 2, pp. 345–357, 2002.

[9] H. Chen, Y. Xiao, and X. Shen, "Update-based cache access and replacement in wireless data access," *IEEE Transactions on Mobile Computing*, vol. 5, no. 12, pp. 1734–1748, 2006.

[10] M. Akon, M. T. Islam, X. Shen, and A. Singh, "OUR: Optimal update-based replacement policy for cache in wireless data access networks with optimal effective hits and bandwidth requirements," *Wireless Communications and Mobile Computing*, vol. 13, no. 15, pp. 1337–1352, 2013.