Single vs Distributed Edge Caching for Dynamic Content

Bahman Abolhassani, John Tadrous, Atilla Eryilmaz

***Abstract*—Existing content caching mechanisms are predominantly geared towards easy-access to 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 and distributed edge caching 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 distribute edge caching 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 reveal that the distributed edge caching scenario, benefiting from the multicasting property of wireless service to update the cached content, can be cost-effective compared to the single edge caching, as the number of edge caches 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 a higher cache hit rates. However higher hit rates come at the expense of a less fresh content, resulting in a higher overall system cost.

Numerous works study the content delivery in caching systems (see [1], [2], and [3]) and effective strategies have been proposed [4], [5]. In [6], authors consider the problem of video on-demand streaming through distributed caching helpers to direct the traffic to the nearest small cell access points using short-range links. The challenges of demand fluctuations and storage prices are addressed in [7]. In [5] and [8], 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 (see [9], [10] and [11]). 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 [12]. 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 the backend databases, currently cached content at local caches becomes out of date or stale since users are interested in the latest version of data [13], [14]. Most caching policies, however, do not consider the content generation dynamics and focus alternatively on the content popularity [15]. 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 [16]. On the other hand, most works that consider the caching for dynamic content, focus mainly on minimizing the miss rate [14] or minimizing the average age of the cached content (see [17], [16]). In [18], 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 [19], 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 [20], 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 [21], 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

Manuscript received December 23, 2020; revised June 11, 2021 and accepted October 13, 2021; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Richard La. Date of publication –, 2021; date of current version October 13, 2021. This work is funded in part from the ONR Grant N00014-19-1-2621 and the NSF grants: CNS-NeTS-1717045, CNS-SpecEES-1824337, CNS-NeTS-2007231, CNS-NeTS-2106679; IIS-2112471.

B. Abolhassani and A. Eryilmaz are with the Department of Electrical and Computer Engineering, The Ohio State University, Columbus, OH 43210 USA (e-mail:abolhassani.2@osu.edu; eryilmaz.2@osu.edu).

J. Tadrous is with the Department of Electrical and Computer Engineering, Gonzaga University, Spokane, WA 99202 (e-mail:tadrous@gonzaga.edu).

decide which items to cache.

References that are most closely related to our work are [22], [13] and [23]. Reference [22] studies the problem of cache updating system with a source, a single cache, and a user. Authors provide a maximization-based method to find the update rates for the cache and for the user to maximize the total freshness of the files at the user. Even though the problem setting in [22] is similar to ours, our freshness metric is different and our goal is to jointly minimize the freshness cost and the cost of fetching the fresh item from the database.

Reference [23] considers a similar model to our distributed edge caching scenario where the BS refreshes the cached content based on AoI upon user requests. The average AoI and service delay have been derived in closed forms, demonstrating a trade-off relationship concerning the refreshing window size. Different from [23] where the freshness of the local cache is measured using the AoI metric, we use a completely new metric to measure the freshness of content and call it *Age-of-Version* (AoV).

While AoI is a meaningful metric for measuring the freshness of content in some systems, there are many real-world scenarios where content does not lose its value simply because of elapsed times since it put into the cache. These types of dynamic contents include news and social network updates where the users prefer to have the freshest version but so long as there is no new update, that content is considered to be the freshest version. 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 [24]. In this work, we use our proposed fresh metric AoV which counts the integer difference between the versions at the database and the local caches [25], [26] and [27]. We also introduce a new cost function for dynamic content caching which captures both the cost due to the missed event and the cost due to content freshness [28] which grows with the AoV metric. Moreover, our model utilizes the multicasting property of the wireless medium to opportunistically update the cached contents over the edge-caches [29] and [30].

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 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 distributed edge caching and single edge caching scenarios 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 that utilizes distributed edge caches for serving dynamic content over wireless broadcast channels.
- In Section III, we provide an overview of the main findings.
- In Section IV, for a database of N data items with an arbitrary popularity distribution that locally serves a group of users by utilizing a single local edge cache 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 cache check and update rates.
- In Section V, using the distributed edge caches to serve the requests of the neighbouring 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.
- In Section VI, comparing the average cost per user for the distributed edge caching with the average cost of the system with a single local edge cache, we highlight scenarios in which each of these approaches are more cost effective. Finally, we conclude the work in Section VII.

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 groups of users by utilizing K distributed edge caches. Each edge cache is equipped with a limited-storage cache that can hold M different items.

Content update dynamic: 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.

Content popularity: Each edge cache k serves a group of users which generate 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.

Channel Failure is incurred due to the unreliability of the wireless transmission. We assume every transmission over the wireless medium is successful with probability $\alpha > 0$. Therefore, $1 - \alpha$ is the probability of channel failure. We also assume that channel failure is independent over the K edge caches.

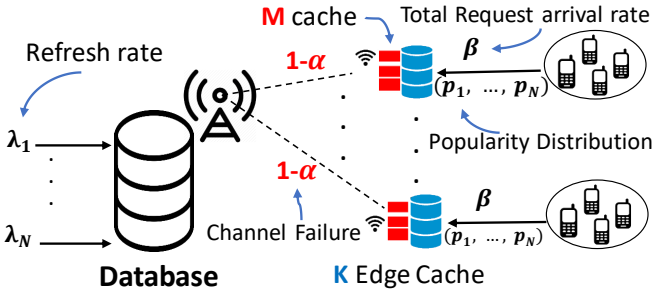

Fig. 1: Caching with freshness dynamics.

Fetching cost is incurred when the requested data is not in the cache. When a request arrives to the edge cache for an item that is not stored in the cache, the edge cache will attempt to fetch the most fresh version from the database and incur a constant cost $C_f \geq 0$. As such attempts over the wireless channel will be subject to failure with probability $1-\alpha$, the edge cache will keep attempting until the data item is successfully received by the user requesting that item, incurring the cost C_f for each attempt.

Freshness cost is incurred due to the fact that the cached content may not be the most fresh version. When a user generates a request to a data item that is found in its assigned edge cache, the request is fulfilled immediately from the edge cache under a "freshness cost". We associate a freshness cost with such events which increases linearly with the *age* of the cached content. Age is defined to be the number of update events occurred to the content in the database after it has been last cached at the edge cache. Therefore, age i means that the content has been updated i times in the database since it has been placed in the edge cache. If the user, thus, consumes a content from the cache with age i, the user will incur a freshness cost of $i \cdot C_0$, where $C_0 \geq 0$ is a constant showing the freshness cost per stale version.

Cache check and update mechanism is utilized by each local edge cache to keep the cached content fresh. We assume that single local edge cache employs a cache check and update mechanism generated according to a Poisson process with rate $\mu \geq 0$. The local edge cache is not aware of the age of the cached content, unless it checks with the database. Each checking process costs $C_{ch} \geq 0$ and if the cached content is found to be not the most updated version, then the edge cache will fetch the most fresh version from the database at an additional caching cost of C_{ca}.

Number of replicas indicates the number of times each item has been cached among the K distributed edge caches. Let r_n be the number of replicas of item n that exist among the K edge caches. Thus, $\mathbf{r} = (r_1, ..., r_N)$ is the vector of the number of times each item has been cached.

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, both for the distributed and single edge caching scenarios.

Single Edge Caching concerns a single edge cache 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 IV.

Distributed Edge Caching, in contrast, concerns the distributed edge caches whereby each edge cache, locally serves the requests for the dynamic content coming from the neighbouring users. The key new component in this case is the broadcast nature of the wireless medium whereby transmissions of content made to one edge cache can be received and used to opportunistically update content in other edge 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 V, we undertake this interesting setting to provide optimal distributed allocation strategy for minimum overall cost.

For example, consider a football stadium filled with people watching the game. Assume there are WiFi access points distributed in the stadium, each serving the requests coming from the neighboring users. As it is shown in Fig. 1, assume there are K WiFi access points, each equipped with M-item storage capacity. The data set of N items is not static and is continuously changing with a refresh rate vector $\boldsymbol{\lambda} = (\lambda_n)_{n=1}^N$. For instance, users may request to watch the video clips of the most recent tackle, which is subject to change as a new tackle occurs and renders the previous versions of the cached videos obsolete. This motivates us to study the different approaches that can utilize edge caches to serve a group of users requesting from a dynamic set of data items. In this scenario, there are two approaches in order to update the cached contents of each WiFi access point. One approach is to update the cache of each WiFi access point individually, i.e., single edge caching. The other approach is to take advantage of the broadcast nature of the wireless communication and use multicasting to update the cached content of all the K WiFi access points, i.e., Distributed Edge Caching. Caching the most popular items in edge caches has been proven to be a good strategy when the data items are static [31] but what does happen if the data items become dynamic and change over time? Of course, in reality, the refresh rate and popularity would not be known beforehand and they need to be learned overtime, but our work focuses on understanding their impact on caching decisions.

In both the local and distributed edge caching scenarios, 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 VI, we compare the optimal

solutions for the single and distributed edge caching scenarios for equal request rates and equal cache spaces per edge cache in order to reveal the benefits of the distributed caching over common caching that emerges due to the dynamic nature of the content.

III. Overview And Discussion of Main Findings

This section presents an overview of this paper's main findings and highlights the key insights on both single and distributed edge caching. It also furnishes a review of the impact of the popularity and refresh rate on the caching decisions.

We explicitly characterize the performance and introduce the following metrics for optimizing the single caching as a function of the cache check and update rate μ (see Section IV) and for the distributed caching as a function of the number of replicas r_n (see Section V) respectively as:

$$\delta_n^{\mathcal{L}}(\mu) = \frac{\beta C_f}{\alpha} p_n - \frac{\beta C_0}{\alpha\mu} p_n \lambda_n - \frac{\mu \mathrm{C_{ca}}}{\lambda_n - \alpha\mu} \lambda_n - \mu C_{ch},$$

$$\delta_n^{\mathcal{D}}(r_n) = \frac{\beta C_f}{\alpha} p_n - \frac{C_0}{K - r_n} \lambda_n.$$

In both metrics, the first term, $\frac{\beta C_f}{\alpha} p_n$, is the popularity related term, and the other terms are related to the refresh rate. By investigating the proposed metrics, we can observe the following facts:

- For the single caching, the proposed metric $\delta_n^{\mathcal{L}}(\mu)$ reveals the need for an update mechanism carried out with a rate μ to keep the cached content fresh. On the contrary, the distributed edge caching, benefiting from the wireless multicast to update the cached content, relies solely on the number of replicas of each item distributed over the edge caches to keep the cached content fresh. Therefore, as the number of distributed edge caches K increases, popularity related term dominates the other term related to the refresh rate.
- For the case of static data items with $\lambda_n = 0, \forall n \in \mathcal{N}$, both caching scenarios result in the cache the most popular items strategy.
- For the case of highly dynamic data items with $\lambda_n \to \infty, \forall n \in \mathcal{N}$, both metrics will be negative, preventing any items from being cached. This is due to the high price that should be paid to keep the cache fresh.
- For the general refresh rates, the proposed metrics explicitly indicate the effect of refresh rates together with the popularity distribution on the caching decisions. Both for the local and distributed edge caching, highly popular items will yield in larger metric values and are more likely to be cached. On the other hand, if such popular items are also highly dynamic, the high refresh rates will reduce the metric value, leading to such items less likely to be cached. Furthermore, items with negative metric values will never be cached even if there is available storage capacity at the edge caches.
- An optimal caching strategy for the single edge caching is achieved by jointly optimizing the cache check rate μ together with the set of items to be cached. We address this in our proposed algorithm, Algorithm 1. The optimal caching strategy for the distributed edge caching is achieved by finding the optimal number of replicas for each item over the edge caches. We address this in our proposed algorithm, Algorithm 2.

Comparing the cost per edge cache for the single and distributed edge caching scenarios, we glean the following insights:

- Less surprisingly, as the number of distributed edge caches K or the cache update cost C_{ca} increases, distributed edge caching outperforms the single edge caching in the sense of the average cost per edge cache. This is expected since single edge caching relies on the cache update to keep its cache fresh, and this cost increases linearly with C_{ca}. On the other hand, distributed edge caching, solely relying on multicast for cache update, can reduce its cost per edge cache by providing more fresh cached content. In particular, as the number of edge caches increases, there would be more multicast *diversity* that can be used to update the cached content with no additional cost.
- More surprisingly, for small cache sizes M or highly unreliable channels with $\alpha \ll 1$, single caching is preferable over distributed caching. In the single edge caching scenario, for small cache sizes, we can keep the cached content fresh without paying too much update cost that linearly increases with the cache size. On the other hand, distributed edge caching solely relying on the multicast to keep its cache fresh will suffer more as the channel gets unreliable.

IV. Optimal Caching and Updating for Dynamic Content: Single Edge Caching

In this scenario, edge caches are updated individually and no other edge cache can benefit from content updated destined to another cache. Therefore, we drop the dependence on the edge cache index k, i.e., the users generate requests with a rate of β to the edge cache. The cache size at the edge cache is M data items. To avoid excessive freshness cost, the edge cache employs a cache *check and update* mechanism through which the edge cache 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 edge cache 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 edge cache. As discussed earlier, if an older content with age i is served from the cache, the user will incur a freshness

cost of $i \cdot C_0$, where $C_0 \geq 0$ is a constant. If the requested data is not in the cache, the edge cache 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 $\mathcal{I}_M \subseteq \mathcal{N}$ be the set of items that are stored in the edge cache and let μ be the checking rate of cache content for the freshness. Note that M is the caching capacity of the edge cache and due to the high refresh rate, the user may not necessarily fill the cache. As such $|\mathcal{I}_M| \leq M$.

Lemma 1: Let $C^{\mathcal{L}}_{\mathcal{I}_M}(\mu)$ be the average system cost in the single Edge Caching scenario as the edge cache stores the set of items $\mathcal{I}_M$ and checks the cache freshness with the Poisson process of rate of μ. Then:

$$C^{\mathcal{L}}_{\mathcal{I}_M}(\mu) = \frac{\beta C_f}{\alpha} + |\mathcal{I}_M|\mu C_{ch} + \mu C_{ca} \sum_{n\in\mathcal{I}_M} \frac{\lambda_n}{\lambda_n + \alpha\mu} + \beta \sum_{n\in\mathcal{I}_M} p_n \left(\frac{\lambda_n C_0}{\alpha\mu} - \frac{C_f}{\alpha} \right), \quad (1)$$

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

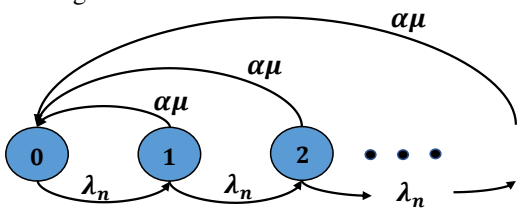

Fig. 2: Markov chain diagram for freshness $\{\Pi^n_\mu(t), t \geq 0\}$ under checking rate μ.

As it can be seen in Fig. 2, every arriving content update to the item n in the database that occurs with rate λ_n 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, with probability α, it will move the system back to state zero, the most fresh version, where $1-\alpha$ is the probability of channel failure. We are interested in the limit of $\Pi^n_\mu(t) \xrightarrow[t\to\infty]{d} \bar{\Pi}^n_\mu$, i.e., the steady state distribution of $\Pi^n_\mu(t)$. Let $\pi^n_i(\mu) = P(\bar{\Pi}^n_\mu = i), i \in \{0,1,2,\ldots\}$ be the probability of item $n \in \mathcal{I}_M$ having the age of i under the checking rate μ, then:

$$\pi^n_i(\mu) = \pi^n_0(\mu) \left(\frac{\lambda_n}{\lambda_n + \alpha\mu} \right)^i, \forall i \in \{0,1,2,\ldots\}. \quad (2)$$

Setting $\sum_{i=0}^{\infty} \pi^n_i(\mu) = 1$, gives $\pi^n_0(\mu) = \frac{\alpha\mu}{\lambda_n + \alpha\mu}$. Hence, the average age of any item $n \in \mathcal{I}_M$ in the cache is given by:

$$\mathbb{E}[\bar{\Pi}^n_\mu] = \sum_{i=0}^{\infty} i\Pi^n_i(\mu) = \frac{\lambda_n}{\alpha\mu}. \quad (3)$$

The average system cost in the single Edge Caching scenario as the edge cache stores the set of items $\mathcal{I}_M$ and checks the cache freshness with rate μ comprises four main terms as follows:

$$C^{\mathcal{L}}_{\mathcal{I}_M}(\mu) = \mu|\mathcal{I}_M|C_{ch} + \frac{\beta C_f}{\alpha}\left(1 - \sum_{n\in\mathcal{I}_M} p_n\right) + \mu C_{ca} \sum_{n\in\mathcal{I}_M} (1 - \pi^n_0(\mu)) + \beta C_0 \sum_{n\in\mathcal{I}_M} p_n \mathbb{E}[\bar{\Pi}^n_\mu]. \quad (4)$$

The first term in Equation (4) shows the average *checking cost* for a cache capacity of M that caches the set of items $\mathcal{I}_M$ 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 $\mathcal{I}_M$ and the request arrival rate of β as a function of miss rate $\beta(1 - \sum_{n\in\mathcal{I}_M} p_n)$. For any arrival request, the miss probability $1 - \sum_{n\in\mathcal{I}_M} p_n$ 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. Due to the channel failure, fetching might not be successful with probability $1-\alpha$. In that case, the user will keep requesting the item, until that content is successfully received by the user, resulting on the average number of $\frac{1}{\alpha}$ fetches for each miss event, each incurring the fetching cost C_f.

The third term in Equation (4) shows the average *caching cost* for a cache set of $\mathcal{I}_M$ and checking rate of μ. For a given μ, $\pi^n_0(\mu), \forall n \in \mathcal{I}_M$ is the probability that item n in the cache is the most updated version, i.e., has age 0. So staleness probability $1-\pi^n_0(\mu)$ is the probability that item n 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 edge cache which incurs the cost of C_{ca}.

The fourth term in Equation (4) shows the average *freshness cost* for a cache set of $\mathcal{I}_M$ and checking rate of μ. For each item $n \in \mathcal{I}_M$ existing in the cache, the arrival request will be served from the cache. The arrival request of item n is βp_n and since the item with age i incurs the freshness cost of $i.C_0$, so the average cost of freshness will be $C_0\mathbb{E}[\bar{\Pi}^n_\mu]$ which $\mathbb{E}[\bar{\Pi}^n_\mu]$ is the average age of the cached item n 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 edge caching scenario would thus be:

$$\min_{\mu\geq 0, \mathcal{I}_M \subseteq \mathcal{N}} C^{\mathcal{L}}_{\mathcal{I}_M}(\mu). \quad (5)$$

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

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

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

Then the cache the most popular strategy will assign the cached set of items as $\mathcal{I}_M = \mathcal{I}^p_M$ and optimizes the cache

Algorithm 1 Single Edge Caching Strategy

Input: $P = (p_1, ..., p_N), \lambda = (\lambda_1, ..., \lambda_N), \mu^p, \mathcal{I}_M^p$

Initialization : $\hat{\mathcal{I}}_M = \emptyset$, $\mathcal{I}_M^{Old} = \mathcal{I}_M^p$

1: Set $\hat{\mu} = \mu_p$

2: Compute $\delta_n^{\mathcal{L}}(\hat{\mu}) = \frac{\beta p_n}{\alpha}\left[C_f - C_0\frac{\lambda_n}{\hat{\mu}}\right] - \hat{\mu}C_{ch} - \frac{\hat{\mu}\lambda_n}{\lambda_n+\alpha\hat{\mu}}\mathrm{C_{ca}}, \forall n \in \mathcal{N}$

3: Update $\hat{\mathcal{I}}_M$ as follows:

$$\hat{\mathcal{I}}_M = \{n \in \mathcal{N} : |\hat{\mathcal{I}}_M| \leq M, \delta_n^{\mathcal{L}}(\hat{\mu}) > 0, \\ \delta_n^{\mathcal{L}}(\hat{\mu}) \geq \delta_i^{\mathcal{L}}(\hat{\mu}), \forall n \in \hat{\mathcal{I}}_M, i \in \mathcal{N}\backslash\hat{\mathcal{I}}_M\}.$$

4: **while** $\hat{\mathcal{I}}_M \neq \mathcal{I}_M^{Old}$ **do**

5: $\mathcal{I}_M^{Old} = \hat{\mathcal{I}}_M$

6: $\hat{\mu} = \underset{\mu\geq 0}{\arg\min}\ C_{\hat{\mathcal{I}}_M}^{\mathcal{L}}(\mu)$

7: Update $\delta_n^{\mathcal{L}}(\hat{\mu})$ from step 2.

8: Update $\hat{\mathcal{I}}_M$ from step 3.

9: **end while**

10: **return** $\hat{\mu}$, $\hat{\mathcal{I}}_M$.

check and update rate as $\mu = \mu^p$, where

$$\mu^p := \underset{\mu\geq 0}{\arg\min}\ C_{\mathcal{I}_M^p}^{\mathcal{L}}(\mu).$$

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 $\hat{\mathcal{I}}_M$ 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 n, we define the metric $\delta^{\mathcal{L}}(\lambda_n, p_n, \mu) = \delta_n^{\mathcal{L}}(\mu)$ as follows:

$$\delta_n^{\mathcal{L}}(\mu) := \frac{\beta p_n}{\alpha}\left[C_f - C_0\frac{\lambda_n}{\mu}\right] - \mu C_{ch} - \frac{\mu\lambda_n}{\lambda_n+\alpha\mu}\mathrm{C_{ca}}, \forall n \in \mathcal{N} \tag{6}$$

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

Our proposed algorithm sorts the items based on $\delta_n^{\mathcal{L}}(\mu)$ and starts filling the cache with items that have the greatest $\delta_n^{\mathcal{L}}(\mu)$ and keeps adding until either all the items with positive $\delta_n^{\mathcal{L}}(\mu)$ are placed in the cache or the cache becomes full, i.e., M 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 $\delta_n^{\mathcal{L}}(\hat{\mu})$ based on new $\hat{\mu}$.

Notice that all data items with negative $\delta_n^{\mathcal{L}}(\mu)$ can only increase the average cost if cached. The metric $\delta_n^{\mathcal{L}}(\mu)$ 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 decrease the values of $\delta_n^{\mathcal{L}}(\mu)$ 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 optimally minimizes the average cost in (1).

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

$$C_{\mathcal{I}_M\cup\{n\}}^{\mathcal{L}}(\mu) = C_{\mathcal{I}_M}^{S}(\mu) - \delta_n^{\mathcal{L}}(\mu) \quad \forall n \notin \mathcal{I}_M$$

By induction, if we set $\mathcal{I}_M = \{\emptyset\}$ and add the item n to the cache, the cost will decrease by $-\delta_n^{\mathcal{L}}(\mu)$. If we keep adding items n with $\delta_n^{\mathcal{L}}(\mu) > 0$, the average cost will continue to decrease. Therefore:

$$C_{\mathcal{I}_M}^{\mathcal{L}}(\mu) = C_{\{\emptyset\}}^{\mathcal{L}}(\mu) - \sum_{n\in\mathcal{I}_M}\delta_n^{\mathcal{L}}(\mu)$$

Since the proposed algorithm at each step chooses the items with greatest positive $\delta_n^{\mathcal{L}}(\mu)$ 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.,

$$C_{\hat{\mathcal{I}}_M}^{\mathcal{L}}(\hat{\mu}) \leq C_{\mathcal{I}_M^p}^{\mathcal{L}}(\mu^p)$$

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:

$$C_{\hat{\mathcal{I}}_M(t)}^{\mathcal{L}}(\hat{\mu}(t)) \geq C_{\hat{\mathcal{I}}_M(t+1)}^{\mathcal{L}}(\hat{\mu}(t)).$$

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

$$C_{\hat{\mathcal{I}}_M(t+1)}^{\mathcal{L}}(\hat{\mu}(t)) \geq C_{\hat{\mathcal{I}}_M(t+1)}^{\mathcal{L}}(\hat{\mu}(t+1)).$$

Combining the two equations gives:

$$C_{\hat{\mathcal{I}}_M(t)}^{\mathcal{L}}(\hat{\mu}(t)) \geq C_{\hat{\mathcal{I}}_M(t+1)}^{\mathcal{L}}(\hat{\mu}(t+1)),$$

which shows at each iteration, the proposed algorithm reduces the cost. Since we start the algorithm with $\hat{\mu}(1) = \mu^p$

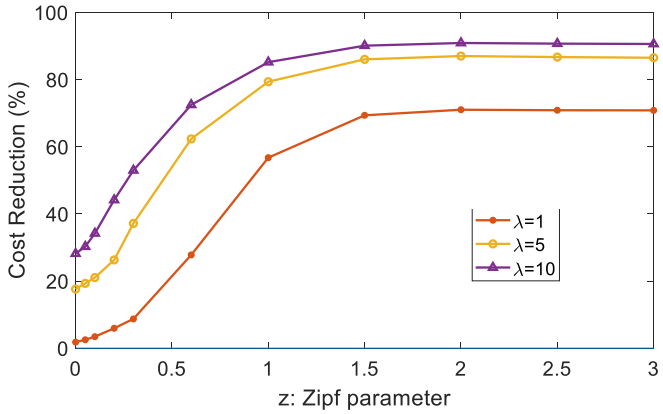

Fig. 3: Average cost reduction by the proposed algorithm over the cache the most popular for the single edge caching.

and $\hat{\mathcal{I}}_M(1) = \mathcal{I}_M^p$, 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 $N = 10^3$, data items' popularity be $p_n = c/n^\gamma$ with $\gamma = 1$ 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 $C_f = 1, C_{ca} = 0.1, C_{ch} = 0.05, C_0 = 0.01$.

Setting the cache size M to be 100, $\alpha = 0.9$ and $\beta = 5$, 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:

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

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 and the more popular items become highly dynamic (as the parameter z increases).

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

E. Analytical Investigation For Uniform Popularity and Refresh Rate

While uniform popularity and symmetric update rate are not common, they facilitate closed-form analysis which reveals performance insights. In such scenario, the local edge cache serves the users requesting from a set of N data items with uniform popularity, i.e., $p_n = \frac{1}{N}, \forall n \in \mathcal{N}$ and same refresh rate over all the items, i.e., $\lambda_n = \lambda, \forall n \in \mathcal{N}$.

The following Proposition shows the optimality of our proposed algorithm, Algorithm 1, when applied to the set of data items with uniform popularity and constant refresh rate.

Proposition 3: For the case of uniform popularity and refresh rate, Algorithm 1 converges to the optimal solution in just one step, where the local edge cache stores the optimal set of items $\mathcal{I}_M^*$ and checks their freshness with the optimal rate μ^* such that:

$$\mu^* = -\frac{\lambda}{2\alpha} + S + \frac{1}{2}\sqrt{-4S^2 - 2p + \frac{q}{S}}, \tag{7}$$

$$|\mathcal{I}_M^*| = \begin{cases} M, & \delta^{\mathcal{L}}(\mu^*) > 0, \\ 0, & \delta^{\mathcal{L}}(\mu^*) \leq 0, \end{cases} \tag{8}$$

where $p = \frac{\lambda^2}{\alpha^2}\left(\frac{C_{ca}}{C_{ch}} - \frac{1}{2}\right)$, $q = \frac{\lambda^3}{\alpha^3}\frac{C_{ca}}{C_{ch}} - \frac{\lambda^2}{\alpha^2}\frac{\beta C_0}{4NC_{ch}}$ and $S = \frac{1}{2}\sqrt{-\frac{2}{3}p + \frac{1}{3}\left(Q + \frac{\Delta_0}{Q}\right)}$ such that $Q = \sqrt[3]{\frac{\Delta_1 + \sqrt{\Delta_1^2 - 4\Delta_0^3}}{2}}$, $\Delta_0 = \frac{\lambda^4}{\alpha^4}\left(\frac{C_{ca}}{C_{ch}} + 1\right)$ and $\Delta_1 = 2\frac{\lambda^6}{\alpha^6}\left(\frac{C_{ca}}{C_{ch}} + 1\right)^3 + \frac{\lambda^5}{\alpha^5}\left(\frac{108\beta C_0 C_{ca}}{NC_{ch}}\right)^2$, and $\delta^{\mathcal{L}}(\mu)$ is given by:

$$\delta^{\mathcal{L}}(\mu) = \frac{\beta C_f}{\alpha N} - \mu C_{ch} - \frac{\mu\lambda}{\lambda + \alpha\mu}\mathrm{C_{ca}} - \frac{\beta C_0 \lambda}{N\alpha\mu}.$$

Proof. According to Lemma 1, for the case of uniform popularity and λ constant, the average system cost as the edge cache stores the set of items $\mathcal{I}_M$ and checks the cache freshness with rate μ is given by:

$$C^{\mathcal{L}}_{\mathcal{I}_M}(\mu) = \frac{\beta C_f}{\alpha}(1 - \frac{|\mathcal{I}_M|}{N}) + |\mathcal{I}_M|\mu C_{ch} + \frac{|\mathcal{I}_M|\beta C_0 \lambda}{N\alpha\mu} + |\mathcal{I}_M|\mu C_{ca}\frac{\lambda}{\lambda + \alpha\mu},$$

Solving the cost minimization in Equation (5) for the case of uniform popularity and λ constant will give the optimal μ^* as (7). Under such μ^* the marginal cost reduction by adding each item to the cache is $-\delta^{\mathcal{L}}(\mu^*)$ where $\delta^{\mathcal{L}}(\mu)$ is given in Equation (6) for the general case. Our proposed algorithm, Algorithm 1, starts filling the cache until either all the items with positive $\delta_n^{\mathcal{L}}(\mu)$ are placed in the cache or the cache becomes full, i.e., M items have been already cached. Since for the case of uniform popularity distribution and λ constant, we have $\delta_n^{\mathcal{L}}(\mu) = \delta^{\mathcal{L}}(\mu), \forall n \in \mathcal{N}$, so for the optimal checking rate μ^*, either $\delta^{\mathcal{L}}(\mu^*) > 0$ where the proposed algorithm has decided to fill the cache by choosing M of the data items to be placed in the cache, or $\delta^{\mathcal{L}}(\mu^*) \leq 0$ where the proposed algorithm has decided to not cache any items, since it can only increase the average system cost. This gives the optimal set of cached items given in the Equation (8). As can be seen from Equation (7), for the case of uniform popularity with λ constant, optimal cache checking rate μ^* is independent of the set of cached data

items $\mathcal{I}_M$ and therefore Algorithm 1 converges to μ^* in just one step. This completes the proof. ■

To further elaborate on the results of the Proposition 3, we consider a special case when $\frac{\lambda}{\alpha}$ is considerably large. This can happen when the set of data items is highly dynamic, i.e., $\lambda \to \infty$ or the communication channel becomes very unreliable, i.e., $\alpha \to 0$, or a combination of the two. Under any of these scenarios, the optimal checking rate μ^* given in (7) can be approximated as:

$$\mu^* \approx \sqrt{\frac{\beta C_0 \lambda}{N\alpha\left(C_{ca}+C_{ch}\right)}}.$$

This approximation clearly shows the effect of system parameters on the optimal checking rate. Assuming that for the given parameters, $\delta^{\mathcal{L}}(\mu^*)$ is positive (i.e., $\lambda < \frac{\beta C_f^2}{4C_0(C_{ca}+C_{ch})\alpha N}$), the average cost of a single user with $M \leq N$ cache capacity requesting of a set of items with uniform popularity and refresh rate is given by:

$$C^*\left(\mu^*\right) = \frac{\beta C_f}{\alpha}\left(1-\frac{M}{N}\right) + 2M\sqrt{\frac{\beta C_0 \lambda\left(C_{ca}+C_{ch}\right)}{N\alpha}},$$

where $C^*\left(\mu^*\right)$ shows a linear cost reduction as the cache capacity M increases. Thus, the minimum achievable cost by the single user caching system is reached when $M = N$ and is given by $\frac{2}{\sqrt{\alpha}}\sqrt{\beta C_0 \lambda N(C_{ch}+C_{ca})}$.

V. Optimal Caching and Updating for Dynamic Content:Distributed Edge Caching Scenario

Consider the scenario shown in Fig. 1, with K edge caches. To gain a clear insight of the potential *wireless multicasting* gain and how distributed caching can relate to the single edge caching scenario with a cache of size M, we assume that each edge cache in the distributed edge caching scenario has also the capacity to cache M of the date items.

In this section, we investigate what items to be cached and how should the cached items be replicated over the set of edge caches. In the distributed edge caching 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 edge caching scenario. Instead, edge caches that have a certain item in their cache can update it for free if another edge cache 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 edge caches. In other words, r_n is the number of replicas of item n that exist in the edge 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 freshness cost of consuming an item from the cache is $i.C_0$ where i is the age of the cached content.

A. Problem Formulation

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

$$\mathcal{F}_K^M = \left\{ \mathbf{r} = (r_1, \ldots, r_N) \,|\, \sum_{n=1}^{N} r_n \leq KM, r_n \in \{0, 1, \ldots\} \right\},$$

where KM is the total cache available in the system.

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

$$C^{\mathcal{D}}(\mathbf{r}) = \frac{K\beta C_f}{\alpha} + \sum_{n=1}^{N} r_n \left(\frac{C_0 \lambda_n}{K - r_n} - \frac{\beta p_n C_f}{\alpha} \right). \quad (9)$$

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

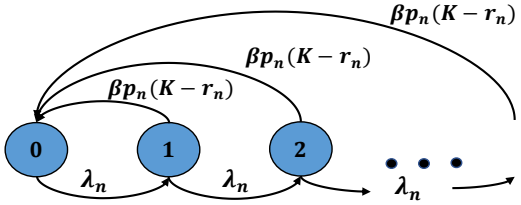

Fig. 4: Markov chain diagram for freshness $\{\Pi_{r_n}^n(t), t \geq 0\}$ under the number of replicas r_n.

distributed edge caching scenario, the broadcast capability of wireless service acts as a natural update mechanism. In other words, edge caches update their cached content for free by overhearing that content while being served to other edge caches who do not have it in their cache. For any item n in the cache, since there are K edge caches and r_n of them have item n in their cache, so the remaining $K - r_n$ edge caches don't have the item n stored in their cache. Each request for item n from these edge caches will result in a miss event and trigger fetching which can be overheated by the other edge caches to update their cached content of item n. Therefore, the service rate of item n for any of the edge caches that have it in their cache is equal to $\frac{\beta p_n (K - r_n)}{\alpha}$. Notice that because of the channel failure, each fetching is successful with probability α and the edge cache will keep fetching the requested item until it is successfully received, resulting in the average number of $\frac{1}{\alpha}$ fetches for each miss event. As it can be seen in Fig. 4, every service of item n acts as an update mechanism for the edge caches that hold item n in their cache and upon occurrence, the service of item n from the database, with probability α of successful communication, will move the system back to state zero, the most fresh version. Giving the rate $\alpha \frac{\beta p_n (K - r_n)}{\alpha} = \beta p_n (K - r_n)$ to state zero. Every arriving content update to the item n in the database that occurs with rate λ_n increases the age of that item in the cache by one.

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

$$\mathbb{E}[\bar{\Pi}^n_{r_n}] = \frac{\lambda_n}{\beta p_n (K - r_n)}. \tag{10}$$

The average system cost in the Distributed edge scenario as K edge caches store according to the vector of replication $\mathbf{r} = (r_1, ..., r_N) \in \mathcal{F}^M_K$, comprises two main terms and is given by:

$$C^{\mathcal{D}}(\mathbf{r}) = \frac{\beta C_f}{\alpha}\left(K - \sum_{n=1}^{N} r_n p_n\right) + \beta C_0 \sum_{n=1}^{N} p_n r_n \mathbb{E}[\bar{\Pi}^n_{r_n}]. \tag{11}$$

The first term in Equation (11), shows the average *fetching cost* for any $\mathbf{r} \in \mathcal{F}^M_K$ and request arrival rate β as a function of miss rate $\beta\left(K - \sum_{n=1}^{N} r_n p_n\right)$. For any of the K edge caches, if a requested item is in the edge cache of the user requesting that item, 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. Fetching will be successful with probability α and we will keep fetching until the data item is successfully received by the user requesting that content, resulting on the average number of $\frac{1}{\alpha}$ fetches for each miss event. Since there are r_n edge caches that have item n in their cache, $K - r_n$ edge caches will not have item n in their cache. Therefore, the miss rate for item n is $\beta p_n (K - r_n)$. Summing over all the N items and remembering that $\mathbf{r} \in \mathcal{F}^M_K$, gives the total miss rate as $\beta\left(K - \sum_{n=1}^{N} r_n p_n\right)$.

The second term in Equation (11), shows the average *freshness cost* for any $\mathbf{r} \in \mathcal{F}^M_K$ and request arrival rate of β. For each item n in the cache, the arrival request rate is βp_n and since the item with age i incurs the cost of $i \cdot C_0$, so the average cost of freshness for item n will be $C_0 \mathbb{E}[\bar{\Pi}^n_{r_n}]$. Since r_n is the number of users having item n in their cache, the total freshness cost incurred by item n is given by $\beta p_n r_n C_0 \mathbb{E}[\bar{\Pi}^n_{r_n}]$. Summing over all the items gives the total freshness cost of the system. Substituting Equation (10) in Equation (11) gives the average cost of the system. ■

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

$$\arg\min_{\mathbf{r} \in \mathcal{F}^M_K} C^{\mathcal{D}}(\mathbf{r}). \tag{12}$$

The traditional cache the most popular strategy in this context reduces to caching the KM most popular items[1] to the users' caches, M items per edge cache.

[1]Note that caching the same item at all users (i.e., setting $r_n = K$ for some $n \in \mathcal{N}$, $r_j = 0$,$\forall j \neq n$) 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.

Algorithm 2 Distributed Edge Caching Strategy

Input: $\mathbf{p} = (p_1, ..., p_N), \boldsymbol{\lambda} = (\lambda_1, ..., \lambda_N), K$
Initialization : $r^*_n = 0 \quad \forall n \in \mathcal{N}$
1: Calculate $\delta^{\mathcal{D}}_n(r^*_n) = \frac{\beta p_n C_f}{\alpha} - \frac{KC_0\lambda_n}{(K - r^*_n)(K - r^*_n - 1)} \quad \forall n \in \mathcal{N}$.
2: $j = \arg\max_{n \in \mathcal{N}} \delta^{\mathcal{D}}_n(r^*_n)$
3: **while** $\delta^{\mathcal{D}}_j(r^*_j) > 0$ and $\sum_{n=1}^{N} r^*_n < KM$ **do**
4: $\quad r^*_j = r^*_j + 1$
5: $\quad$ update $\delta^{\mathcal{D}}_j(r^*_j)$ from Step 1.
6: $\quad$ update $j = \arg\max_{n \in \mathcal{N}} \delta^{\mathcal{D}}_n(r^*_n)$
7: **end while**
8: **return** $\mathbf{r}^* = (r^*_1, \ldots, r^*_N)$

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

$$r^p_n := \begin{cases} 1, & n \in \mathcal{I}^p_{K,M}, \\ 0, & n \in \mathcal{N} \setminus \mathcal{I}^p_{K,M}, \end{cases}$$

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 (12).

In particular, as it can be seen in Algorithm 2, for item n, we define the metric $\delta^{\mathcal{D}}(\lambda_n, p_n, l) = \delta^{\mathcal{D}}_n(l)$ as follows:

$$\delta^{\mathcal{D}}_n(l) := \frac{\beta p_n C_f}{\alpha} - \frac{KC_0\lambda_n}{(K - l)(K - l - 1)} \quad \forall n \in \mathcal{N}. \tag{13}$$

The metric $-\delta^{\mathcal{D}}_n(l)$ captures the marginal cost of adding item n to the caches given that l of the edge caches have already cached item n. Our proposed algorithm, at each step, sorts the items based on $\delta^{\mathcal{D}}_n(l)$, caches the item with the maximum $\delta^{\mathcal{D}}_n(l)$ and iterates until either all the items with positive $\delta^{\mathcal{D}}_n(l)$ are cached or no more edge caches 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 negative $\delta^{\mathcal{D}}_n(l)$ can only increase the average cost if cached. Similar to single edge caching scenario, $\delta^{\mathcal{D}}_n(l)$ 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 (12) 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 (12) must satisfy all the following conditions.

$$\delta_n^{\mathcal{D}}(\bar{r}_n - 1) \geq 0 \quad \forall n \in \mathcal{N}, \text{ with } \bar{r}_n > 0, \tag{14}$$

$$\delta_n^{\mathcal{D}}(\bar{r}_n - 1) \geq \delta_j^{\mathcal{D}}(\bar{r}_j) \quad \forall j \neq n, \text{ with } \bar{r}_n > 0, \tag{15}$$

$$\sum_{n=1}^{N} \bar{r}_n = KM \text{ or } \delta_n^{\mathcal{D}}(\bar{r}_n) < 0 \; \forall n \in \mathcal{N}. \tag{16}$$

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

To prove that Equation (14) is necessary for optimality, we use contradiction. Assume that Equation (14) does not hold, so there exists $j \in \mathcal{N}$ with $\bar{r}_j > 0$ such that $\delta_j^{\mathcal{D}}(\bar{r}_j - 1) < 0$. Then construct $\mathbf{r} = \bar{\mathbf{r}} - e_j$, where $\mathbf{r} \in \mathcal{F}_K$ and we have that $C^{\mathcal{D}}(\mathbf{r}) = C^{\mathcal{D}}(\bar{\mathbf{r}}) + \delta_j^{\mathcal{D}}(\bar{r}_j - 1)$. Since $\delta_j^{\mathcal{D}}(\bar{r}_j - 1) < 0$, so $C^{\mathcal{D}}(\mathbf{r}) < C^{\mathcal{D}}(\bar{\mathbf{r}})$ which contradicts the fact that $\bar{\mathbf{r}}$ was the optimal solution.

To prove that Equation (15) is necessary for optimality, assume that there exist $n, j \in \mathcal{N}$ such that $\delta_n^{\mathcal{D}}(\bar{r}_n - 1) < \delta_j^{\mathcal{D}}(\bar{r}_j)$. Then construct $\mathbf{r} = \bar{\mathbf{r}} - e_n + e_j$, where $\mathbf{r} \in \mathcal{F}_K$ and we have that $C^{\mathcal{D}}(\mathbf{r}) = C^{\mathcal{D}}(\bar{\mathbf{r}}) + \delta_n^{\mathcal{D}}(\bar{r}_n - 1) - \delta_j^{\mathcal{D}}(\bar{r}_j)$. So there exists $\mathbf{r} \in \mathcal{F}_K$ with $C^{\mathcal{D}}(\mathbf{r}) < C^{\mathcal{D}}(\bar{\mathbf{r}})$ which contradicts the fact that $\bar{\mathbf{r}}$ was the optimal solution.

To prove that Equation (15) is necessary for optimality, assume that $\sum_{n=1}^{N} \bar{r}_n < KM$ and $j \in \mathcal{N}$ such that $\delta_j^{\mathcal{D}}(\bar{r}_j) > 0$. Construct $\mathbf{r} = \bar{\mathbf{r}} + e_j$, where $\mathbf{r} \in \mathcal{F}_K$ and we have that $C^{\mathcal{D}}(\mathbf{r}) = C^{\mathcal{D}}(\bar{\mathbf{r}}) - \delta_j^{\mathcal{D}}(\bar{r}_j)$. Since $\delta_j^{\mathcal{D}}(\bar{r}_j) > 0$, so there exists $\mathbf{r} \in \mathcal{F}_K$ with $C^{\mathcal{D}}(\mathbf{r}) < C^{\mathcal{D}}(\bar{\mathbf{r}})$ 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 (12) 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 (14), (15) and (16) will result in the same average cost.

Proof. To prove the lemma, we show that for any $\bar{\mathbf{r}} \in \mathcal{F}_K$ 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 $C^{\mathcal{D}}(\mathbf{r}) = C^{\mathcal{D}}(\bar{\mathbf{r}})$. Assume $\mathbf{a} \neq 0$, we consider two cases separately.

Case 1: if $\delta_n^{\mathcal{D}}(\bar{r}_n) < 0 \; \forall n \in \mathcal{N}$, then if there exists j such that $a_j > 0$, we have that:

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

So Equation (14) does not hold for $\mathbf{r}$, which is a contradiction. Hence $a_n \leq 0 \; \forall n \in \mathcal{N}$. If $a_n = 0 \; \forall n$ then the problem is solved, but if there exists j such that $a_j < 0$, then we have:

$$\delta_j^{\mathcal{D}}(r_j) = \delta_j^{\mathcal{D}}(\bar{r}_j + a_j) \geq \delta_j^{\mathcal{D}}(\bar{r}_j - 1) \geq 0. \tag{17}$$

According to Equation (16), $\sum_{n=1}^{N} r_n = KM$ should hold for $\mathbf{r}$, but $\sum_{n=1}^{N} r_n = \sum_{n=1}^{N} \bar{r}_n + \sum_{n=1}^{N} a_n < KM$, since $\sum_{n=1}^{N} \bar{r}_n \leq K$ and $\sum_{n=1}^{N} a_n < 0$, which is a contradiction.

Case 2: If $\delta_n^{\mathcal{D}}(\bar{r}_n) < 0$ does not hold for all $n \in \mathcal{N}$, then according to Equation (16), $\sum_{n=1}^{N} \bar{r}_n = KM$ should hold. Since $\sum_{n=1}^{N} r_n = \sum_{n=1}^{N} \bar{r}_n + \sum_{n=1}^{N} a_n \leq KM$, then $\sum_{n=1}^{N} a_n \leq 0$. If $a_n \leq 0$ for all n, then in order to have $\mathbf{a} \neq 0$, there exists $j \in \mathcal{N}$ with $a_j < 0$ such that Equation (17) holds. Now, from Equation (16), $\sum_{n=1}^{N} r_n = KM$ should hold for $\mathbf{r}$, but since $\sum_{n=1}^{N} a_n < 0$, 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 $\sum_{n=1}^{N} a_n = 0$ since we should have $\sum_{n=1}^{N} r_n = KM$ as shown before. Since $\mathbf{r}$ satisfies all the necessary conditions of Lemma 3, Equation (15) holds for $\mathbf{r}$ over v and j.

$$\delta_v^{\mathcal{D}}(\bar{r}_v) \geq \delta_v^{\mathcal{D}}(\bar{r}_v + a_v - 1) \geq \delta_j^{\mathcal{D}}(\bar{r}_j + a_j) \geq \delta_j^{\mathcal{D}}(\bar{r}_j - 1).$$

Now if $\delta_v^{\mathcal{D}}(\bar{r}_v) > \delta_j^{\mathcal{D}}(\bar{r}_j - 1)$, the condition of Equation (15) does not hold for $\bar{\mathbf{r}}$ which is a contradiction and if $\delta_v^{\mathcal{D}}(\bar{r}_v) = \delta_j(\bar{r}_j - 1)$, construct the $\mathbf{r} = \bar{\mathbf{r}} + e_v - e_j$. Then $C^{\mathcal{D}}(\mathbf{r}) = C^{\mathcal{D}}(\bar{\mathbf{r}}) - \delta_v^{\mathcal{D}}(\bar{r}_v) + \delta_j^{\mathcal{D}}(\bar{r}_j - 1) = C^{\mathcal{D}}(\bar{\mathbf{r}})$. 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 distributed edge caching scenario supported with wireless multicasting can lead to some edge caches storing less popular items than those cached at other edge caches. 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 the edge caches 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 $z = 1, \alpha = 0.9$ 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:

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

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 edge caches increases, reveling that the proposed algorithm, Algorithm 2, can more effectively incorporate the broadcasting gain to reduce the cost. It also reveals that the gain increases as the cache

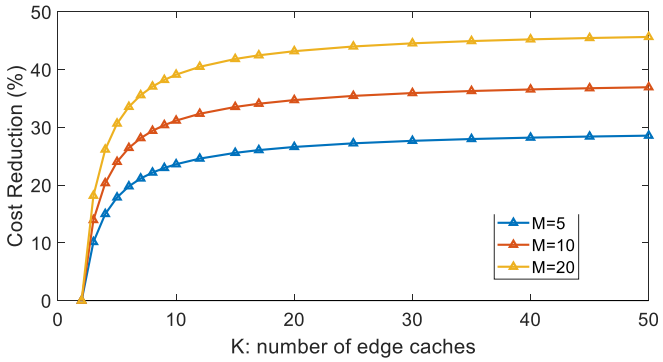

Fig. 5: Average cost reduction by the proposed algorithm over the cache the most popular for distributed edge caching.

capacity per edge cache increases. Also as the refresh rate of different items decreases, with less edge caches we can achieve higher gains, benefiting more from the multicasting gain.

E. Analytical Investigation For Uniform Popularity and Refresh Rate

While uniform popularity and symmetric update rate are not common, they facilitate closed-form analysis which reveals performance insights. In this scenario, the local edge cache serves the users requesting from a set of N data items with uniform popularity, i.e., $p_n = \frac{1}{N}, \forall i \in \mathcal{N}$ and same refresh rate over all the items, i.e., $\lambda_n = \lambda, \forall n \in \mathcal{N}$.

Proposition 4: The average cost per edge cache of a system shown in Fig. 1 comprising of K edge caches each equipped with $M \leq M^*$ cache requesting of a data set of N items with uniform popularity and constant refresh rate, is given by:

$$C^{\mathcal{D}}\left(\mathbf{r}^*\right)/K = C_0\lambda\frac{NM}{K(N-M)} + \frac{\beta C_f}{\alpha}\left(1-\frac{M}{N}\right), \quad (18)$$

where we have assumed that $\frac{KM}{N} \in \{0,1,2,\ldots\}$ and the optimal storage capacity per edge cache M^* is given by:

$$M^* = \max\{M \in \mathbb{Z}^+ | M \leq N(1-\sqrt{\frac{C_0\lambda N\alpha}{K\beta C_f}})\}.$$

Notice that M^* is the maximum storage capacity per edge cache that can be used by the distributed edge caching system to reduce the average cost and if storage per edge cache $M > M^*$, the proposed Algorithm 2 that optimally minimizes the cost will not use the extra cache, since doing so can only increase the average system cost.

Proof. Total average cost of the system shown in Fig. 1 with K edge caches each with M storage is given in Equation (9). Assuming that $\frac{KM}{N} \in \{0,1,2,\ldots\}$, then applying Algorithm 2 will give $r_n^* = \frac{KM}{N}, \forall n \in \mathcal{N}$. Finally, dividing the total average cost by the number of distributed caches K gives the average cost per edge cache. The average cost per edge cache given in the Equation (18) is a convex function of the cache capacity per edge cache M. Minimizing over M and considering that cache capacity can only be an integer will give us the optimal storage capacity per edge cache M^* under the above assumptions. ■

According to Proposition 4, the average cost per edge cache for the distributed edge caching scenario is minimized when each edge cache has at least M^* storage capacity. Assuming that $N(1-\sqrt{\frac{C_0\lambda N\alpha}{K\beta C_f}}) \in \mathbb{Z}^+$, the minimum achievable cost per edge cache by the distributed edge caching system is given by $2\sqrt{\frac{\beta C_f C_0\lambda N}{\alpha K}} - \frac{C_0\lambda N}{K}$.

VI. Numerical Results

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

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

Note that the positive percentages of cost reduction means that cost per edge cache for the distributed edge caching is smaller than the single edge caching.

We let the total number of data items be $N = 1000$, data items' popularity be $p_n = c/n^\gamma$ with $\gamma = 1$ and content refresh rates be $\lambda_n = \lambda/n^z$ with $z = 1$. We consider the normalized costs of fetching, checking, and freshness to be $C_f = 1, C_{ch} = 0.1, C_0 = 0.01$. Varying the following parameters: cache storage per edge cache M, normalized caching cost C_{ca} and the number of edge caches K, we compare the average cost per edge cache achieved by Algorithm 2 for the distributed edge caching, and the average cost of a single edge cache performing under the Algorithm 1. We assume all the edge caches have the same cache space to store M items in their cache. In the following we show the effect of these parameters on the performance of distributed edge caching compared to the single edge caching.

Fig. 6 shows the percentage cost reduction for different caching capacity M as a function of the cache update cost C_{ca} for the single edge caching. According to the figure, for small cache sizes, single cache outperforms distributed caching in the sense of average cost per edge cache, but as the cache update cost C_{ca} grows, distributed edge caching, benefiting more through the multicasting property, will outperform the single edge caching. This is due to the fact that the single edge cache depends on the C_{ca} to keep its cached contents fresh. Therefore, as C_{ca} increases, the single edge cache needs to pay more cost to maintain a fresh cache. Also, as the number of cache capacity M increases, there are more cached items that need to be checked and updated, and as any such update will incur a cost C_{ca}, the single edge cache

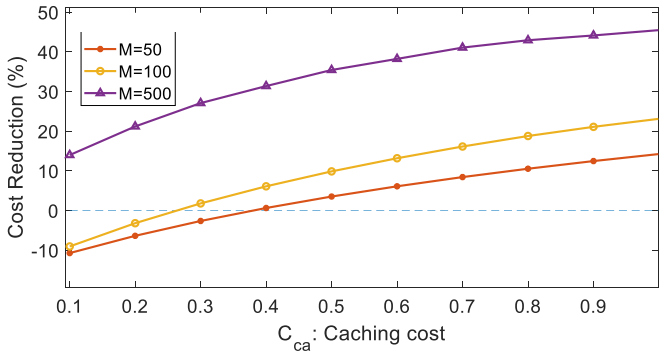

Fig. 6: Average cost reduction per edge cache by $K = 3$ distributed caching with $\alpha = 1$.

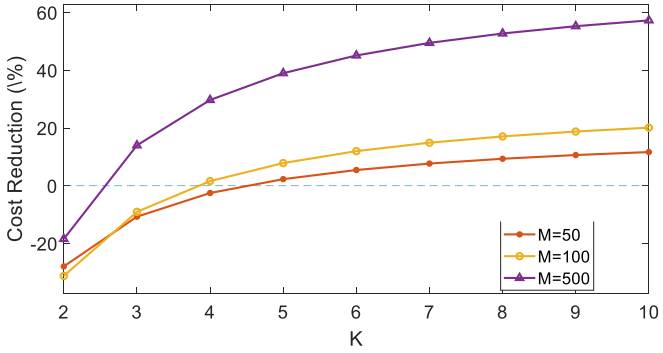

Fig. 7: Average cost reduction per edge cache by distributed edge caching with $\alpha = 1$ and $C_{ca} = 0.1$.

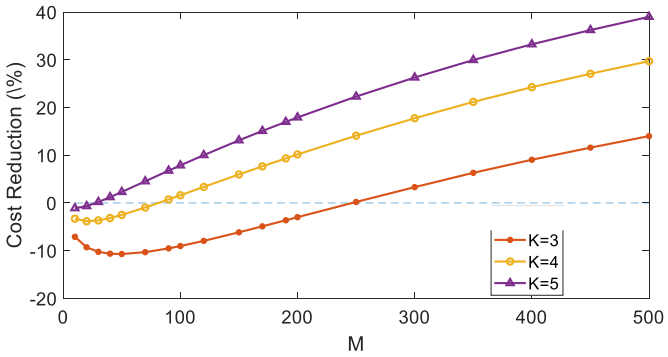

Fig. 8: Average cost reduction per edge cache by distributed edge caching with $\alpha = 1$ and $C_{ca} = 0.1$.

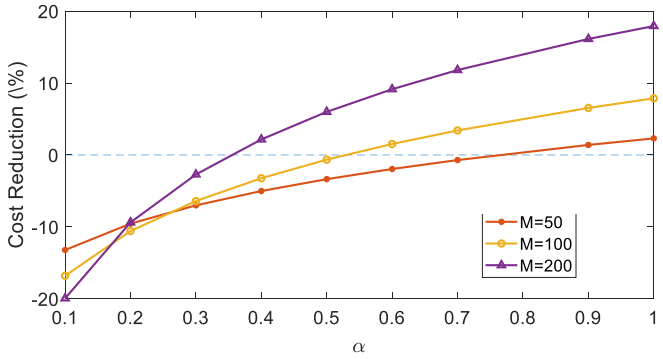

Fig. 9: Average cost reduction per edge cache by $K = 5$ distributed edge caching with $C_{ca} = 0.1$.

performance will fall behind the distributed edge caching performance that can benefit free updates by utilizing the multicast nature of the wireless communication.

Fig. 7 shows the percentage cost reduction for different caching capacity M as a function of the number of distributed edge caches K. According to the figure, for small number of available edge caches, single cache outperforms distributed caching in the sense of average cost per edge cache. This is due to the fact that the distributed edge caches solely rely on multicast to keep their cache fresh, and if the number of edge caches is small, there would not be enough multicast diversity to update the cached content, this will result to increase in the freshness cost and edge caches may decide not to cache items to keep the average cost low, resulting in unutilized cache space. But as the number of edge caches K increases, due to the increased diversity, there will be enough multicast to keep the cached content fresh. Therefore, distributed edge caching will always outperform the single edge caching as the number of edge caches K increases. The performance gap will further grow larger as the number of cache capacity per edge cache M increases. Because the single edge cache, solely depending on the cache update cost C_{ca} to keep the cached content fresh, will struggle to keep its cache fresh as the number of cached items increases.

Fig. 8 shows the percentage cost reduction for different numbers of edge caches K as a function of cache space per edge cache M. According to the figure, if the cache per edge cache M and the number of edge caches K, both are small, the single cache outperforms distributed caching in the sense of average cost per edge cache. But as any of these parameters increases, distributed edge caching, benefiting more from the multicast, becomes more cost effective. Increasing K, introduces more diversity to the distributed edge caching scenario and therefore lowers the average cost per edge cache, while the single edge caching performance does not depend on K. On the other hand, as the cache capacity per edge cache M increases, there would be more cached items that need to be updated frequently. While distributed edge caching scenario can use multicast to update their cache, the single edge cache will have to spend the average updating cost C_{ca} for each cache update. Therefore, updating cost for the single edge cache can add up as the cache capacity per edge cache increases, since there would be more items that need to be kept fresh.

Fig. 9 shows the percentage cost reduction for different caching capacity M as a function of the probability of successful communication α. According to the figure, for small values of α when the channel is unreliable, the single edge caching outperforms the distributed edge caching. Moreover, for extremely unreliable channels, as the cache capacity per edge cache M increases, the performance gap grows. But as the α increases and channel becomes more reliable, distributed edge caching outperforms the single edge caching. This is due to the fact that distributed edge caching depends on the successful multicast communication

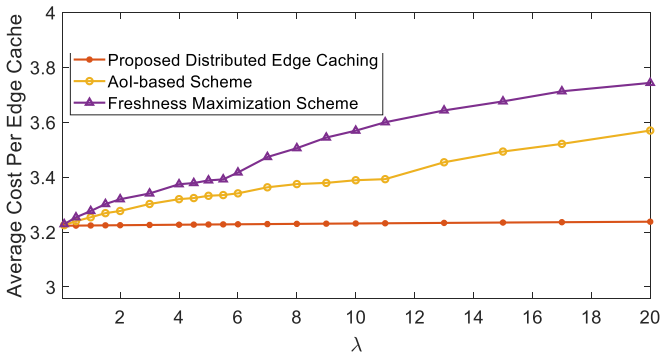

Fig. 10: Average cost of distributed edge caching per edge cache.

to update the cache and as the channel becomes less reliable, its performance will suffer more compared to the single edge caching.

To show the practicality of our proposed algorithms, we use an extensive set of real-world data, namely the data set of the BBC iPlayer [32], [33], [34], to obtain realistic video demand distributions. The BBC iPlayer is a video streaming service from BBC that provides video content for several BBC channels without charge. Content on the BBC iPlayer is available for up to 30 days depending on the policies. We consider the dataset covering June 2014, which includes 192,120,311 recorded access sessions, resulting in request rates $\beta_t = 74.1205$ requests per second. The number of files according to the iPlayer database is larger than $n = 10000$. According to [35], the popularity distribution of video files of the BBC iPlayer requested by the users in June 2014 can be approximated by the Zipf distribution with parameter $z = 0.86$. We also assume that the content of the dataset have refresh rates according to $\lambda_n = \lambda/n^z$ [22] with parameter $z = 0.86$. We consider the normalized costs of fetching, checking, and freshness to be $C_f = 1, C_{ch} = 0.1, C_0 = 0.01$. Finally, we consider $K = 20$ edge caches deployed over the system where each serves the fraction $\beta = \frac{\beta_t}{K} = 3.7$ requests per seconds, and each edge cache is equipped with $M = 20$ storage capacity. We aim to conduct a simulation using the data set of the BBC iPlayer to compare our proposed Algorithm with the dynamic caching schemes proposed in [23] and [22]. We modify the AoI-based caching scheme proposed in [23] which does not consider the λ and the freshness maximization scheme proposed in [22] which considers the λ, to fit into our system model and use similar parameters to do the comparison.

Fig. 10 shows the average cost of distributed edge caching per edge cache as a function of refresh rate λ for channel success probability $\alpha = 0.9$. According to the figure, if the items are almost static, all caching strategies have the same performance, but as the items become more dynamic, our proposed algorithm outperforms both the AoI-based scheme and freshness maximization scheme. The reason is that the AoI-based scheme uses AoI as a freshness metric where items lose their freshness with the same rates simply because of the elapsed times since they last receive any updates. It also fails to consider that different item lose their freshness at different rates. On the other hand, the freshness maximization scheme only focuses to maximize the freshness of cached items, failing to consider the fetching costs associated with miss events. Whereas, our proposed algorithm uses AoV as a freshness metric where items do not lose their freshness simply by elapsed times, and different items can have different refresh rates. It also takes into account the fetching cost and attempts to maximize the freshness while also minimizing the fetching cost by considering the trade-off between freshness and fetching costs. As the refresh rate increases, the performance gap between our proposed algorithm and the previously mentioned dynamic caching schemes increases. Our simulation results also confirm that as the wireless channel becomes more reliable, the performance gap increases.

VII. 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 and distributed edge caching scenarios. Our results have revealed that, in the presence of dynamic content, adding more cache space to edge caches may solve the system congestion problem at the expense of a high freshness cost. In the distributed edge caching scenario, as the number of edge caches increases, our proposed algorithm benefits more from the multicasting property as a mechanism to update the cache content and outperforms single edge 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] J. Zhang, "A literature survey of cooperative caching in content distribution networks," *arXiv preprint arXiv:1210.0071*, 2012.

[2] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Transactions on Information Theory*, vol. 60, no. 5, pp. 2856–2867, 2014.

[3] W. Jiang, G. Feng, and S. Qin, "Optimal cooperative content caching and delivery policy for heterogeneous cellular networks," *IEEE Transactions on Mobile Computing*, vol. 16, no. 5, pp. 1382–1393, 2016.

[4] J. Ni and D. H. Tsang, "Large-scale cooperative caching and application-level multicast in multimedia content delivery networks," *IEEE Communications Magazine*, vol. 43, no. 5, pp. 98–105, 2005.

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

[6] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.

[7] J. Kwak, G. Paschos, and G. Iosifidis, "Elastic femtocaching: Scale, cache, and route," *IEEE Transactions on Wireless Communications*, 2021.

[8] 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.

[9] Z. Zheng and Z. Zheng, "Towards an improved heuristic genetic algorithm for static content delivery in cloud storage," *Computers & Electrical Engineering*, vol. 69, pp. 422–434, 2018.

[10] M. Rabinovich and O. Spatscheck, *Web caching and replication*. Addison-Wesley Boston, USA, 2002, vol. 67.

[11] T. Tang, "Multicast enabled caching service," Oct. 10 2002, uS Patent App. 09/934,013.

[12] 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.

[13] Z. Su, Y. Hui, Q. Xu, T. Yang, J. Liu, and Y. Jia, "An edge caching scheme to distribute content in vehicular networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 6, 2018.

[14] 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.

[15] A. Giovanidis and A. Avranas, "Spatial multi-lru caching for wireless networks with coverage overlaps," *ACM SIGMETRICS Performance Evaluation Review*, vol. 44, no. 1, pp. 403–405, 2016.

[16] E. Najm and R. Nasser, "Age of information: The gamma awakening," in *2016 IEEE International Symposium on Information Theory (ISIT)*. Ieee, 2016, pp. 2574–2578.

[17] C. Kam, S. Kompella, and A. Ephremides, "Age of information under random updates," in *2013 IEEE International Symposium on Information Theory*. IEEE, 2013, pp. 66–70.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] M. Bastopcu and S. Ulukus, "Information freshness in cache updating systems," *IEEE Transactions on Wireless Communications*, 2020.

[23] S. Zhang, L. Wang, H. Luo, X. Ma, and S. Zhou, "Aoi-delay tradeoff in mobile edge caching with freshness-aware content refreshing," *IEEE Transactions on Wireless Communications*, 2021.

[24] 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," in *2018 10th International Conference on Wireless Communications and Signal Processing (WCSP)*. IEEE, 2018.

[25] B. Abolhassani, J. Tadrous, A. Eryilmaz, and E. Yeh, "Fresh caching for dynamic content," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[26] B. Abolhassani, J. Tadrous, and A. Eryilmaz, "Optimal load-splitting and distributed-caching for dynamic content," in *2021 19th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOPT)*. IEEE, 2021, pp. 1–8.

[27] ——, "Achieving freshness in single/multi-user caching of dynamic content over the wireless edge," in *2020 18th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOPT)*. IEEE, 2020, pp. 1–8.

[28] G. Ahani and D. Yuan, "Accounting for information freshness in scheduling of content caching," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 2020, pp. 1–6.

[29] B. Abolhassani, J. Tadrous, and A. Eryilmaz, "Wireless multicasting for content distribution: Stability and delay gain analysis," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1–9.

[30] ——, "Delay gain analysis of wireless multicasting for content distribution," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 529–542, 2020.

[31] J. Famaey, F. Iterbeke, T. Wauters, and F. De Turck, "Towards a predictive cache replacement strategy for multimedia content," *Journal of Network and Computer Applications*, 2013.

[32] D. Karamshuk, N. Sastry, M. Al-Bassam, A. Secker, and J. Chandaria, "Take-away tv: Recharging work commutes with predictive preloading of catch-up tv content," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 8, pp. 2091–2101, 2016.

[33] G. Nencioni, N. Sastry, G. Tyson, V. Badrinarayanan, D. Karamshuk, J. Chandaria, and J. Crowcroft, "Score: Exploiting global broadcasts to create offline personal channels for on-demand access," *IEEE/ACM Transactions on Networking*, vol. 24, no. 4, pp. 2429–2442, 2015.

[34] M.-C. Lee, A. F. Molisch, N. Sastry, and A. Raman, "Individual preference probability modeling and parameterization for video content in wireless caching networks," *IEEE/ACM Transactions on Networking*.

[35] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003, pp. 314–329.

Bahman Abolhassani received the B.Sc. and M.Sc. degrees in electrical engineering from Sharif University of Technology (SUT), Tehran, Iran, in 2015 and 2017, respectively. He is currently pursuing the Ph.D. degree with the Electrical and Computer Engineering Department, The Ohio State University, Columbus, OH, USA. Between 2015 and 2017, he was a researcher at the Optical Networks Research Laboratory, SUT. His research interests include communication networks, optimization theory, caching and algorithm design.

John Tadrous (S'10–M'15–SM'20) received the B.Sc. degree from the EE Department, Cairo University, the M.Sc. degree in wireless communications from the Center of Information Technology, Nile University, in 2010, and the Ph.D. degree in electrical engineering from the ECE Department, The Ohio State University, in 2014. He was a Research Assistant at the Wireless Intelligent Networks Center, Nile University, from 2008 to 2010, where he worked on resource allocation and power control for cognitive radio networks. From 2010 to 2014, he was a Research Associate at the Information Processing Systems Laboratory, where he worked on proactive resource allocation and scheduling, smart data pricing, and information theory. From 2014 to 2016, he was with the Center for Multimedia Communication, Rice University, as a Post-Doctoral Research Associate, where he worked on modeling and analysis of interactive data traffic, full-duplex communications, and beamforming design for massive MIMO systems. He served as an assistant professor of Electrical and Computer Engineering at Gonzaga University from September 2016 to May 2021. He has been an Associate Professor since September 2021. He received Gonzaga University's Faculty Award for Professional Contributions in May 2020.

Atilla Eryilmaz (S'00 / M'06 / SM'17) received his M.S. and Ph.D. degrees in Electrical and Computer Engineering from the University of Illinois at Urbana-Champaign in 2001 and 2005, respectively. Between 2005 and 2007, he worked as a Postdoctoral Associate at the Laboratory for Information and Decision Systems at the Massachusetts Institute of Technology. Since 2007, he has been at The Ohio State University, where he is currently a Professor and the Graduate Studies Chair of the Electrical and Computer Engineering Department. Dr. Eryilmaz's research interests span optimal control of stochastic networks, machine learning, optimization, and information theory. He received the NSF-CAREER Award in 2010 and two Lumley Research Awards for Research Excellence in 2010 and 2015. He is a co-author of the 2012 IEEE WiOpt Conference Best Student Paper, subsequently received the 2016 IEEE Infocom, 2017 IEEE WiOpt, 2018 IEEE WiOpt, and 2019 IEEE Infocom Best Paper Awards. He has served as: a TPC co-chair of IEEE WiOpt in 2014 and of ACM Mobihoc in 2017; an Associate Editor (AE) of IEEE/ACM Transactions on Networking between 2015 and 2019; and is an AE of IEEE Transactions on Network Science and Engineering since 2017.