Maximizing Information Freshness in Caching Systems with Limited Cache Storage Capacity

Melih Bastopcu Sennur Ulukus
Department of Electrical and Computer Engineering
University of Maryland, College Park, MD 20742
bastopcu@umd.edu *ulukus@umd.edu*

***Abstract*—We consider a cache updating system with a source, a cache with limited storage capacity and a user. There are n files. The source keeps the freshest versions of the files which are updated with known rates. The cache gets fresh files from the source, but it can only store the latest downloaded versions of K files where $K \leq n$. The user gets the files either from the cache or from the source. If the user gets the files from the cache, the received files might be outdated depending on the file status at the source. If the user gets the files directly from the source, then the received files are always fresh, but the extra transmission times between the source and the user decreases the freshness at the user. Thus, we study the trade-off between storing the files at the cache and directly obtaining the files from the source at the expense of additional transmission times. We find analytical expressions for the average freshness of the files at the user for both of these scenarios. Then, we find the optimal caching status for each file (i.e., whether to store the file at the cache or not) and the corresponding file update rates at the cache to maximize the overall freshness at the user. We observe that when the total update rate of the cache is high, caching files improves the freshness at the user. However, when the total update rate of the cache is low, the optimal policy for the user is to obtain the frequently changing files and the files that have relatively small transmission times directly from the source.**

I. Introduction

Time sensitive information has become ever more important especially with emerging technologies such as autonomous driving, augmented reality, social networking, high-frequency automated trading, online gaming, etc. Age of information has been introduced to measure the timeliness of information in communication networks. Age of information has been widely studied in the context of web crawling, queueing networks, caching systems, remote estimation, energy harvesting systems, scheduling in networks, and so on [1]–[18].

In this work, we consider a cache updating system that consists of a source, a cache with limited cache (i.e., storage) capacity and a user as shown in Fig. 1. In this system, the source keeps the freshest versions of all the files that are refreshed with known rates λ_i. The cache gets the freshest versions of the files from the source, but its cache capacity is limited, i.e., it can only store the freshest versions of K files where $K \leq n$. The user gets files either from the cache or from the source. If the user gets a file from the cache, the updated file at the user might still be outdated depending on the file status at the source. If the user gets a file directly from the source, the received file is always fresh. However, as the channel between the user and the source is not perfect, there is a file transmission time which decreases the freshness at the user. Thus, in this paper, we study the trade-off between storing the files at the cache to decrease the file transmission times versus directly obtaining the fresh files from the source at the expense of higher transmission times. Our aim is to find the optimal caching status for each file (i.e., whether to store the file at the cache or not) and the corresponding optimal file update rates at the cache.

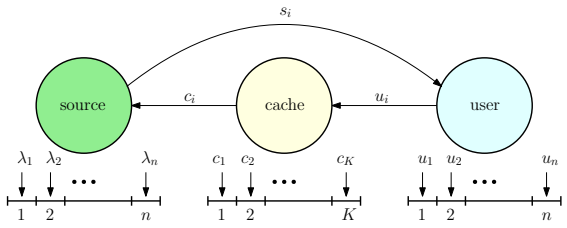


Fig. 1. A cache updating system with a source, a cache and a user.

References that are most closely related to our work are [7] and [18]. Reference [7] considers a model where a resource constrained remote server wants to keep the items at a local cache as fresh as possible. Reference [7] shows that the update rates of the files should be chosen proportional to the square roots of their popularity indices. Different from [7] where the freshness of the local cache is considered, we consider the freshness at the end-user. Furthermore, the freshness metric that we use in this paper is different than the traditional age metric used in [7], and hence our overall work is distinct compared to [7]. In comparison to our earlier work in [18], here, we consider a cache with limited caching capacity, and we study the trade-off between storing the files at the cache and obtaining the files directly from the source.

In this paper, we find an analytical expression for the average freshness of the files at the user when the files are cached and not cached. We impose a total update rate constraint for the cache due to limited nature of resources. We find the optimal caching status for each file and the corresponding optimal file update rates at the cache. We observe that due to binary nature of file caching status, the optimization problem is NP-hard. However, for a given set of caching status of the files, the problem becomes a convex optimization problem in terms of the file update rates at the cache. For a given set of caching

This work was supported by NSF Grants CCF 17-13977 and ECCS 18-07348.

status of the files, the optimal rate allocation policy at the cache is a *threshold policy* where the rapidly changing files at the source may not be updated. We observe that when the total update rate of the cache is high, storing files at the cache improves the freshness of the user. However, when the total update rate of the cache is low, it is optimal for the user to obtain the rapidly changing files and the files that have relatively small transmission times directly from the source.

II. System Model

We consider an information updating system where there is a source, a cache and a user as shown in Fig. 1. The source keeps the freshest version of n files which are updated with exponential inter-arrival times with rate λ_i. The file updates at the source are independent of each other. The cache gets fresh files from the source, but it may store only K files where $K = 1, \ldots, n$. We assume that the channel between the source and the cache is perfect and the transmission times are negligible. Thus, if the cache requests an update for a stored file, it receives the file from the source right away. We model the inter-update request times for the ith file at the cache as exponential with rate c_i. The cache is subject to a total update rate constraint, i.e., $\sum_{i=1}^{n} c_i \leq C$ as in [7], [18].

The inter-update request times of the user for the ith file are exponential with rate u_i. The channel between the user and the cache is also assumed to be perfect and the transmission times are negligible. Thus, if the requested file is stored at the cache, the user gets the stored file at the cache right away. If the user requests a file which is not cached, the cache forwards the file update request from the user to the source. Since the cache only forwards the user requests for uncached files (i.e., without creating requests of its own), $c_i > u_i$ is not possible, and we have $c_i \leq u_i$. For uncached files, the file update requests at the cache are fully synchronized with the user's requests which means that when the user requests an update for an uncached file, this request reaches the source immediately if the cache forwards it. Thus, for each file update request of the user for the uncached file i, the cache forwards the request to the source with probability $p_i = \frac{c_i}{u_i}$. From [19, Thm. 13.6], the effective inter-update request times of the user for an uncached file are exponential with rates c_i. We assume that the channel between the source and the user is imperfect and the transmission time for the ith file is exponential with rate s_i.

We note that each file at the source is always *fresh*. However, when a file is updated at the source, the stored versions of the same file at the cache and at the user become *outdated*. When the cache gets an update for an outdated file, the updated file in the cache becomes *fresh* again until the next update arrival at the source. The user gets files either from the cache or from the source. If the user gets a file from the cache, it will receive the file update immediately, but the received file can be outdated if the file at the cache is not fresh. If the user gets a file directly from the source, the received file is always fresh, but the transmission takes time. We note that since the cache and the user are unaware of the file updates at the source, they do not know whether they have the freshest versions of the files or not. Thus, they may still unknowingly request an update even though they have the freshest version of a file.

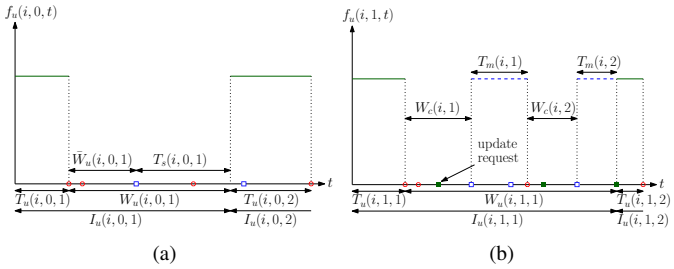


Fig. 2. Sample evolution of the freshness of the ith file at the user when the ith file is (a) not cached and (b) cached. Red circles represent the update arrivals at the source, blue squares represent the update requests from the cache, and green filled squares represent the update requests from the user.

We use k_i which is a binary variable to indicate the caching status of the ith file, i.e., $k_i = 1$ when the ith file is cached and $k_i = 0$ when it is not cached. We define $f_u(i, k_i, t)$ as the freshness function of the ith file at the user as,

$$f_u(i, k_i, t) = \begin{cases} 1, & \text{if the } i\text{th file is fresh at time } t, \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

where the instantaneous freshness function is a binary function taking values of fresh, "1", or not fresh, "0", at any time t. A sample $f_u(i, k_i, t)$ is shown in Fig. 2(a) when $k_i = 0$ and in Fig. 2(b) when $k_i = 1$.

File updates that replace an outdated version of the file with the freshest one are denoted as *successful* updates. We define the time interval between the jth and the $(j+1)$th successful updates for the ith file at the user as the jth update cycle and denote it by $I_u(i, k_i, j)$. We denote the time duration when the ith file at the user is fresh during the jth update cycle as $T_u(i, k_i, j)$. We denote $f_c(i, t)$ as the freshness function of the ith file at the cache. Similarly, update cycles and duration of freshness at the cache are denoted by $I_c(i, j)$ and $T_c(i, j)$. Then, we denote $F_u(i, 1)$ (resp. $F_u(i, 0)$) as the long term average freshness of the ith file at the user when the file is cached (resp. when the file is not cached), i.e., $k_i = 1$ (resp. $k_i = 0$). $F_u(i, k_i)$ is equal to

$$F_u(i, k_i) = \lim_{T\to\infty} \frac{1}{T} \int_0^T f_u(i, k_i, t) dt. \tag{2}$$

Similar to [1], we have

$$F_u(i, k_i) = \lim_{T\to\infty} \frac{N}{T} \left(\frac{1}{N} \sum_{j=1}^{N} T_u(i, k_i, j) \right) = \frac{\mathbb{E}[T_u(i, k_i)]}{\mathbb{E}[I_u(i, k_i)]},$$

where N is the number of update cycles in time duration T. We define the total freshness over all files at the user F_u as

$$F_u = \sum_{i=1}^{n} k_i F_u(i, 1) + (1 - k_i) F_u(i, 0). \tag{3}$$

Our aim is to find the optimal file caching status k_i, and the corresponding file update rates at the cache c_i for $i = 1, \ldots, n$, such that the total average freshness of the user F_u

is maximized while satisfying the constraints on the cache capacity, i.e., $\sum_{i=1}^{n} k_i \leq K$, the total update rate of the cache, $\sum_{i=1}^{n} c_i \leq C$, and the feasibility constraints i.e., $c_i \leq u_i$ for uncached files (for files with $k_i = 0$). Thus, our problem is,

$$\begin{aligned} \max_{\{k_i,c_i\}} \quad & F_u \\ \text{s.t.} \quad & \sum_{i=1}^{n} k_i \leq K \\ & \sum_{i=1}^{n} c_i \leq C \\ & (1-k_i)c_i \leq u_i, \qquad i = 1,\dots,n \\ & c_i \geq 0, \quad k_i \in \{0,1\}, \quad i = 1,\dots,n. \end{aligned} \tag{4}$$

In the following section, we find the long term average freshness of the ith file at the user $F_u(i,k_i)$ when the ith file is cached and when it is not cached. Once we find $F_u(i,k_i)$, this will determine the objective function of (4) via (3).

III. Average Freshness Analysis

In this section, we find the long term average freshness for the ith file at the user $F_u(i,k_i)$ for $k_i \in \{0,1\}$. In the following theorem, we first find the long term average freshness of the ith file at the user when the ith file is cached.

Theorem 1 *If the ith file is cached, the long term average freshness of the ith file at the user $F_u(i,1)$ is equal to*

$$F_u(i,1) = \frac{\mathbb{E}[T_u(i,1)]}{\mathbb{E}[I_u(i,1)]} = \frac{u_i}{u_i+\lambda_i}\frac{c_i}{c_i+\lambda_i}. \tag{5}$$

The proof of the Theorem 1 follows from [18, Section III]. Since the user gets fresh files more frequently from the cache for higher values of c_i, the freshness of the ith file at the user $F_u(i,1)$ in (5) increases with c_i. In addition, $F_u(i,1)$ in (5) is a concave function of c_i. If the user was directly connected to the source, freshness of the ith file at the user would be equal to $\frac{u_i}{u_i+\lambda_i}$ as in [18]. However, as the user is connected to the source via the cache, the freshness experienced by the user proportionally decreases with the freshness of the cache which is $\frac{c_i}{c_i+\lambda_i}$. Note that $\frac{c_i}{c_i+\lambda_i} < 1$ for all c_i.

Next, we find the long term average freshness of the ith file at the user $F_u(i,0)$ when the ith file is not cached.

Theorem 2 *If the ith file is not cached, the long term average freshness of the ith file at the user $F_u(i,0)$ is equal to*

$$F_u(i,0) = \frac{\mathbb{E}[T_u(i,0)]}{\mathbb{E}[I_u(i,0)]} = \frac{c_i}{c_i+\lambda_i+\frac{c_i\lambda_i}{s_i}}. \tag{6}$$

Proof: When the ith file at the user becomes fresh, the time until the next file update arrival at the source is still exponentially distributed with rate λ_i due to the memoryless property of the exponential distribution. Thus, $\mathbb{E}[T_u(i,0)] = \frac{1}{\lambda_i}$.

After the ith file is updated at the source, the stored version of the ith file at the user becomes outdated, i.e., the instantaneous freshness function $f_u(i,0,t)$ becomes 0 again. We denote the time interval until the source gets a file update request for the ith file after the file at the user becomes outdated as $\bar{W}_u(i,0)$ which is exponentially distributed with rate c_i as discussed in Section II. After receiving the file update request from the user, the source sends the ith file directly to the user. If the ith file at the source is updated during a file transfer, then the file transfer is interrupted and the fresh file is sent until the freshest version of the ith file is successfully transmitted to the user. We denote the total transmission time for the ith file as $T_s(i,0)$. Due to [19, Prob. 9.4.1], $T_s(i,0)$ is also exponentially distributed with rate s_i. Thus, we have $\mathbb{E}[T_s(i,0)] = \frac{1}{s_i}$. We denote the time interval when the ith file at the user is outdated during the jth update cycle as $W_u(i,0,j)$, i.e., $W_u(i,0,j) = I_u(i,0,j) - T_u(i,0,j)$, which is also equal to $W_u(i,0,j) = \bar{W}_u(i,0,j) + T_s(i,0,j)$. We denote the typical random variables for $W_u(i,0,j)$ and $I_u(i,0,j)$ as $W_u(i,0)$ and $I_u(i,0)$, respectively. Then, we have $\mathbb{E}[W_u(i,0)] = \mathbb{E}[\bar{W}_u(i,0)] + \mathbb{E}[T_s(i,0)] = \frac{1}{c_i} + \frac{1}{s_i}$ and

$$\mathbb{E}[I_u(i,0)] = \mathbb{E}[T_u(i,0)] + \mathbb{E}[W_u(i,0)] = \frac{1}{\lambda_i} + \frac{1}{c_i} + \frac{1}{s_i}.$$

Thus, we get $F_u(i,0)$ in (6) by using $F_u(i,0) = \frac{\mathbb{E}[T_u(i,0)]}{\mathbb{E}[I_u(i,0)]}$. ■

We note that $F_u(i,0)$ in (6) is an increasing function of c_i and also is concave in c_i. When the user gets a file from the source directly, the received file is always fresh, but due to the transmission time between the source and the user, the average time that the ith file is outdated at the user increases. Thus, the freshness of the ith file at the user $F_u(i,0)$ in (6) increases with s_i. Further, $F_u(i,1) > F_u(i,0)$ implies that $\frac{c_i}{c_i+\lambda_i} > \frac{s_i}{u_i}$. In other words, if the file update rate of the ith file at the cache c_i is high enough, it is better to cache file i. However, if file i is updated too frequently at the source, i.e., λ_i is too large, or file i has small transmission times, i.e., s_i is too high, then it is better to get the file from the source. Thus, there is a trade-off: If a file is stored at the cache, this enables the user to obtain the file more quickly, but the received file might be outdated. On the other hand, if the user gets the file directly from the source, the file will always be fresh, but the file transmission time decreases the freshness at the user.

IV. Freshness Maximization

In this section, we solve the optimization problem in (4). Using $F_u(i,k_i)$ in (5) and (6) and F_u in (3), we rewrite the freshness maximization problem in (4) as

$$\begin{aligned} \max_{\{k_i,c_i\}} \quad & \sum_{i=1}^{n} k_i \frac{u_i}{u_i+\lambda_i}\frac{c_i}{c_i+\lambda_i} + (1-k_i)\frac{c_i}{c_i+\lambda_i+\frac{c_i\lambda_i}{s_i}} \\ \text{s.t.} \quad & \sum_{i=1}^{n} k_i \leq K \\ & \sum_{i=1}^{n} c_i \leq C \\ & (1-k_i)c_i \leq u_i, \qquad i = 1,\dots,n \\ & c_i \geq 0, \quad k_i \in \{0,1\}, \quad i = 1,\dots,n. \end{aligned} \tag{7}$$

In order to solve the optimization problem in (7), we need to determine the optimal caching status for each file k_i and find the optimal file update rates at the cache c_i. We note that the optimization problem in (7) is NP-hard due to the presence of binary variables k_i. However, for a given $(k_1, k_2, \ldots, k_n)$ tuple, (7) becomes a convex optimization problem in c_i. Thus, the optimal solution can be found by searching over all possible $(k_1, k_2, \ldots, k_n)$ tuples and finding the corresponding optimal c_i values for each $(k_1, k_2, \ldots, k_n)$ tuple.

Next, for a given set of $(k_1, k_2, \ldots, k_n)$ values, we find the corresponding optimal c_i values. For that, we introduce the Lagrangian function [20] for (7) as

$$\mathcal{L} = -\sum_{i=1}^{n} k_i \frac{u_i}{u_i+\lambda_i}\frac{c_i}{c_i+\lambda_i} + (1-k_i)\frac{c_i}{c_i+\lambda_i+\frac{c_i\lambda_i}{s_i}} + \beta\left(\sum_{i=1}^{n} c_i - C\right) + \sum_{i=1}^{n}\eta_i((1-k_i)c_i - u_i) - \sum_{i=1}^{n}\nu_i c_i,$$

where $\beta \geq 0$, $\eta_i \geq 0$ and $\nu_i \geq 0$. The KKT conditions are

$$\frac{\partial \mathcal{L}}{\partial c_i} = -\frac{u_i}{u_i+\lambda_i}\frac{\lambda_i}{(c_i+\lambda_i)^2} + \beta - \nu_i = 0, \tag{8}$$

for all i with $k_i = 1$, and

$$\frac{\partial \mathcal{L}}{\partial c_i} = -\frac{\lambda_i}{\left(c_i+\lambda_i+\frac{c_i\lambda_i}{s_i}\right)^2} + \beta + \eta_i - \nu_i = 0, \tag{9}$$

for all i with $k_i = 0$. Complementary slackness conditions are

$$\beta\left(\sum_{i=1}^{n} c_i - C\right) = 0, \tag{10}$$

$$\eta_i((1-k_i)c_i - u_i) = 0, \tag{11}$$

$$\nu_i c_i = 0. \tag{12}$$

For given k_is with $k_i = 1$, we rewrite (8) as

$$(c_i+\lambda_i)^2 = \frac{1}{\beta-\nu_i}\frac{u_i\lambda_i}{u_i+\lambda_i}. \tag{13}$$

If $c_i > 0$, we have $\nu_i = 0$ from (12). Thus, we have

$$c_i = \left(\frac{1}{\sqrt{\beta}}\sqrt{\frac{u_i\lambda_i}{u_i+\lambda_i}} - \lambda_i\right)^+, \tag{14}$$

for all i with $k_i = 1$, where $(x)^+ = \max(x, 0)$. Similarly, for given k_is with $k_i = 0$, we rewrite (9) as

$$\left(c_i+\lambda_i+\frac{c_i\lambda_i}{s_i}\right)^2 = \frac{\lambda_i}{\beta+\eta_i-\nu_i}. \tag{15}$$

If $c_i > 0$, we have $\nu_i = 0$ from (12). Furthermore, if $c_i < u_i$, then we have $\eta_i = 0$ from (11). Otherwise, we have $c_i = u_i$ and $\eta_i \geq 0$ from (11). Thus, we have

$$c_i = \min\left(\frac{s_i}{s_i+\lambda_i}\left(\sqrt{\frac{\lambda_i}{\beta}} - \lambda_i\right)^+, u_i\right), \tag{16}$$

for all i with $k_i = 0$.

Note that $c_i > 0$ in (14) requires $\frac{1}{\lambda_i}\frac{u_i}{u_i+\lambda_i} > \beta$ which also implies that if $\frac{1}{\lambda_i}\frac{u_i}{u_i+\lambda_i} \leq \beta$, then we must have $c_i = 0$. Similarly, $c_i > 0$ in (16) requires $\frac{1}{\lambda_i} > \beta$ which also implies that if $\frac{1}{\lambda_i} \leq \beta$, then we must have $c_i = 0$. Thus, for given k_is, we observe that the optimal rate allocation policy for the cache is a *threshold policy* in which the optimal update rates are equal to zero when the file update rates λ_is are too large, i.e., when the files are updated too frequently at the source. In the optimal policy, the total update rate constraint for the cache, i.e., $\sum_{i=1}^{n} c_i \leq C$, should be satisfied with equality as the objective function in (7) is an increasing function of c_i.

For given k_is and u_is, we define ϕ_i as

$$\phi_i = \begin{cases} \frac{1}{\lambda_i}\frac{u_i}{u_i+\lambda_i}, & \text{if } k_i = 1, \\ \frac{1}{\lambda_i}, & \text{if } k_i = 0. \end{cases} \tag{17}$$

Similar to [18, Lemma 3], for given k_is and u_is, if $c_i > 0$ for some i, then we have $c_j > 0$ for all j with $\phi_j \geq \phi_i$.

Next, for a given set of k_is and u_is, we find the optimal c_is. First, we obtain ϕ_i from (17). We initially assume that $c_i < u_i$ for all i with $k_i = 0$, i.e., c_i in (16) is equal to $\frac{s_i}{s_i+\lambda_i}\left(\sqrt{\frac{\lambda_i}{\beta}} - \lambda_i\right)^+$. Then, we rewrite (14) and (16) as

$$c_i = \begin{cases} \frac{\lambda_i}{\sqrt{\beta}}\left(\sqrt{\phi_i} - \sqrt{\beta}\right)^+, & \text{if } k_i = 1, \\ \frac{s_i}{s_i+\lambda_i}\frac{\lambda_i}{\sqrt{\beta}}\left(\sqrt{\phi_i} - \sqrt{\beta}\right)^+, & \text{if } k_i = 0. \end{cases} \tag{18}$$

As we discussed earlier, in the optimal policy, we must have $\sum_{i=1}^{n} c_i = C$. Similar to the solution method in [18], we solve $\sum_{i=1}^{n} c_i = C$ for β by assuming that $\phi_i \geq \beta$ for all i, i.e., by ignoring $(\cdot)^+$ in (18). Then, we compare the smallest ϕ_i with β. If the smallest ϕ_i is larger than or equal to β, it implies that $c_i > 0$ for all i as we assumed before, and we have obtained c_i values for given k_is. If the smallest ϕ_i is smaller than β, it implies that the corresponding c_i was negative and it must be chosen as zero. In this case, we choose $c_i = 0$ for the smallest ϕ_i. Then, we repeat this process again until the smallest ϕ_i among the remaining c_is satisfies $\phi_i \geq \beta$.

Finally, when we find all c_i values, we go back to our initial assumption which is $c_i < u_i$ for all i with $k_i = 0$ and check whether it holds or not. We define the set $S = \{i | c_i > u_i, k_i = 0\}$. If we have $c_i < u_i$ for all i with $k_i = 0$, i.e., when S is empty, then we obtain the optimal c_i values. If we have $c_i > u_i$ for some i with $k_i = 0$, then in the optimal policy, we have $c_i = u_i$ for all $i \in S$. Then, for remaining c_is with $i \notin S$, we repeat this process again with the remaining total update rate, i.e., $C - \sum_{i\in S} u_i$, until we have $c_i \leq u_i$ for all i with $k_i = 0$.

V. Numerical Results

In this section, we provide two numerical results for the optimal solution obtained in Section IV for $n = 8$. For these results, we consider the update arrival rates at the source $\lambda_i = bq^i$ with $q = 0.7$ such that $\sum_{i=1}^{n} \lambda_i = 10$. We take the file request rates at the user $u_i = dr^i$ with $r = 0.8$ such that $\sum_{i=1}^{n} u_i = 20$. Finally, we take the file transmission rates at the source $s_i = hp^i$ with $p = 1.25$ such that $\sum_{i=1}^{n} s_i = 3$.

In the first example, we increase the cache capacity K from 0 to n when the total update rate at the cache is $C = 1, 4, 8$. We

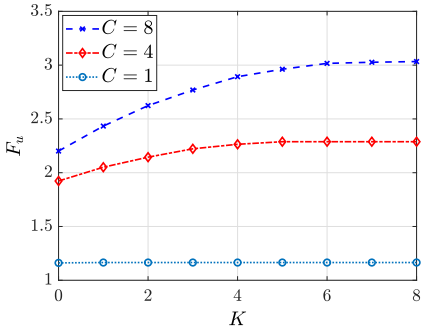
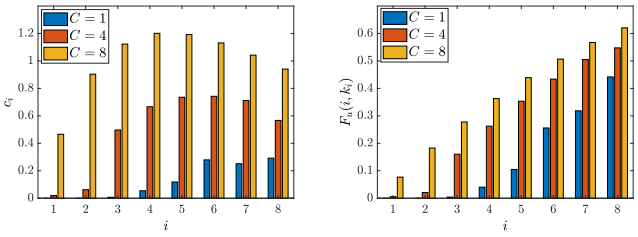


Fig. 3. Total freshness of the user F_u with respect to the cache capacity K when the total cache update rate is $C = 1, 4, 8$.

observe in Fig. 3 that when the total cache update rate is small, i.e., when $C = 1$, increasing the cache capacity K does not improve the freshness of the user much, i.e., F_u stays constant for $K \geq 1$. As the total cache update rate C is too low, if a file is stored at the cache, the user gets obsolete versions of the file most of the time. Thus, we observe that even though the cache capacity is high, the optimal policy is to cache only one file. In this case, the cache mostly forwards the update requests from the user to the source, i.e., the cache behaves like a relay node. When the total cache update rate increases, i.e., when $C = 4$, we observe in Fig. 3 that increasing the cache capacity K increases the user freshness up to $K = 5$ and does not improve it for $K > 5$. Similarly, when $C = 8$, we observe in Fig. 3 that the user freshness increases with the cache capacity. In this case, as the total cache update rate is high enough, the optimal policy is to cache every file.

In the second example, we consider the same system as in the first example, but we take $K = n$ and find the optimal caching status for each file k_i and the corresponding file update rates at the cache c_i. When $C = 1$, the optimal policy is to cache only the 6th file, i.e., $k_6 = 1$ and $k_i = 0$ for $i \neq 6$. When $C = 4$, the optimal caching status is $k_i = 1$, for $i = 3, 4, 5, 6, 7$ and $k_i = 0$, otherwise. Thus, we observe that the files that change too fast at the source are not cached. Furthermore, as the file transmission rate of the 8th file is too high, we see that the 8th file is not cached. When $C = 8$, it is optimal to cache every file, i.e., $k_i = 1$ for all i. Thus, when the total cache update rate is high enough, the optimal policy is to cache every file as caching helps user to avoid the transmission time between the source and the user. However, when the total cache update rate is limited, the optimal policy is not to cache the files that are frequently updated at the source or the files that have smaller transmission times.

The optimal file update rate of the cache c_i is shown in Fig. 4(a). When $C = 1$, i.e., when the total cache update rate is too small, the first two files which are updated at the source most frequently are not updated by the cache, i.e., $c_1 = c_2 = 0$. Furthermore, we observe in Fig. 4(a) that the file update rates at the cache initially increase with the file indices up to $i = 6$ when $C = 4$ and up to $i = 4$ when $C = 8$, and then decrease for the remaining files. The freshness of the files at the user $F_u(i, k_i)$ is shown in Fig. 4(b). We see in Fig. 4(b) that the files that change slowly at the source have higher file freshness



(a) (b)

Fig. 4. (a) Update rate allocation at the cache for each file, and (b) the corresponding freshness $F_u(i, k_i)$, when $C = 1, 4, 8$.

at the user even though the file update rates at the cache get lower. We observe that increasing the total cache update rate C improves the freshness of the files. In addition, we note that the freshness improvement is higher on the rapidly changing files compared to the others.

REFERENCES

[1] J. Cho and H. Garcia-Molina. Effective page refresh policies for web crawlers. *ACM Transactions on Database Systems*, 28(4):390–426, December 2003.

[2] A. Kolobov, Y. Peres, E. Lubetzky, and E. Horvitz. Optimal freshness crawl under politeness constraints. In *ACM SIGIR Conference*, July 2019.

[3] S. K. Kaul, R. D. Yates, and M. Gruteser. Real-time status: How often should one update? In *IEEE Infocom*, March 2012.

[4] M. Costa, M. Codrenau, and A. Ephremides. Age of information with packet management. In *IEEE ISIT*, June 2014.

[5] A. Soysal and S. Ulukus. Age of information in G/G/1/1 systems: Age expressions, bounds, special cases, and optimization. May 2019. Available on arXiv: 1905.13743.

[6] W. Gao, G. Cao, M. Srivatsa, and A. Iyengar. Distributed maintenance of cache freshness in opportunistic mobile networks. In *IEEE ICDCS*, June 2012.

[7] R. D. Yates, P. Ciblat, A. Yener, and M. Wigger. Age-optimal constrained cache updating. In *IEEE ISIT*, June 2017.

[8] C. Kam, S. Kompella, G. D. Nguyen, J. Wieselthier, and A. Ephremides. Information freshness and popularity in mobile caching. In *IEEE ISIT*, June 2017.

[9] J. Zhong, R. D. Yates, and E. Soljanin. Two freshness metrics for local cache refresh. In *IEEE ISIT*, June 2018.

[10] 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 *IEEE WCSP*, October 2018.

[11] H. Tang, P. Ciblat, J. Wang, M. Wigger, and R. D. Yates. Age of information aware cache updating with file- and age-dependent update durations. September 2019. Available on arXiv: 1909.05930.

[12] L. Yang, Y. Zhong, F. Zheng, and S. Jin. Edge caching with real-time guarantees. December 2019. Available on arXiv:1912.11847.

[13] Y. Sun, Y. Polyanskiy, and E. Uysal-Biyikoglu. Remote estimation of the Wiener process over a channel with random delay. In *IEEE ISIT*, June 2017.

[14] A. Arafa, J. Yang, S. Ulukus, and H. V. Poor. Age-minimal transmission for energy harvesting sensors with finite batteries: Online policies. *IEEE Transactions on Information Theory*, 66(1):534–556, January 2020.

[15] M. Bastopcu and S. Ulukus. Age of information with soft updates. In *Allerton Conference*, October 2018.

[16] B. Buyukates, A. Soysal, and S. Ulukus. Age of information scaling in large networks. In *IEEE ICC*, May 2019.

[17] M. Bastopcu and S. Ulukus. Who should Google Scholar update more often? In *IEEE Infocom*, July 2020.

[18] M. Bastopcu and S. Ulukus. Information freshness in cache updating systems. April 2020. Available on arXiv:2004.09475.

[19] R. D. Yates and D. J. Goodman. *Probability and Stochastic Processes*. Wiley, 2014.

[20] S. P. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.