Delay-Aware Coded Caching for Mobile Users

Emre Ozfatura*, Thomas Rarris*, Deniz Gündüz*, and Ozgur Ercetin†
*Information Processing and Communications Lab
Department of Electrical and Electronic Engineering, Imperial College London
{m.ozfatura, thomas.rarris14, d.gunduz}@imperial.ac.uk
†Sabanci University, Turkey, oercetin@sabanciuniv.edu

***Abstract*—Cache capacity-delay trade-off is studied for cooperative coded caching among small-cell base stations (SBSs) considering mobile users. First, a delay-aware coded caching policy is introduced, taking into account the popularity of the files and the maximum re-buffering delay constraint, which minimizes the average re-buffering delay of a mobile user under a given cache capacity constraint. Subsequently, a given average re-buffering delay constraint is considered to ensure a certain quality-of-service (QoS) target, and certain files are served by the macro-cell base station (MBS) when the cache capacity of the SBSs is not sufficient to store all the files in the library. A coded caching policy that minimizes the average amount of data served by the MBS is proposed for the latter scenario.**

I. Introduction

In the last decade, on-demand video streaming have started to dominate the Internet traffic. In 2016, Youtube alone was responsible for %21 of the mobile Internet traffic in North America [1]. By 2021 the size of the Internet video traffic is expected to be four times larger [2]. This rapid growth calls for a paradigm shift in the design of cellular networks. Caching popular contents at the network edge has been proposed as an effective technique to mitigate the excessive video traffic and to reduce latency.

In heterogeneous cellular networks, small-cell base stations (SBSs) can also be equipped with cache memories to store popular video files, and in a network of densely deployed SBSs, there are often multiple SBSs that can serve a mobile user (MU). This flexibility in MU-SBS association leads to the design of cooperative caching policies [3]–[5], which, in a broad sense, aim at maximizing the number of files served locally through SBSs to reduce the load on the macro-cell base stations (MBSs). Furthermore, storing the contents in an encoded form, particularly using maximum distance separable (MDS) codes, can allow users to download a content from multiple SBSs without coordinating the particular portions downloaded from each SBS; and hence, further increases the amount of data served locally [6], [7].

All the aforementioned works seek an optimal cooperative caching policy based on a given static user access topology; however, in ultra dense networks, due to the limited coverage area of SBSs, user access patterns are not static, and the mobility patterns of users have a significant impact on the amount of content that can be delivered locally [8]. Mobility-aware cooperative caching policies have been recently studied in [9], [10]. In these works, the goal is to maximize the amount of data that is served locally while satisfying a given content downloading delay constraint. However, when the contents are stored in coded form [9], [10], a user cannot start displaying the video before collecting all the parity bits, which may cause significant initial *buffering delay*, especially for streaming applications. Proactive content caching for continuous video display scenario, in which users can start displaying the video before downloading all its fragments, has been previously studied in [11], where SBSs fetch contents dynamically in advance, prior to user arrivals, using the instantaneous user mobility information. Instead of a dynamic content fetching policy, in this paper, we consider a static caching policy similarly to [9] and [10], but focus on continuous display of video.

This work received generous support from the European Commission through projects SCAVENGE (grant no. 675891), TACTILENet (grant no. 690893), and BEACON (grant no. 725731).

II. System Model and Problem Formulation

We consider an heterogeneous cellular network with one MBS and N SBSs, $\{\mathrm{SBS}_1, \ldots, \mathrm{SBS}_\mathrm{N}\}$, each with a disjoint coverage area of the same size. Each SBS is equipped with a cache memory of size C bits. Due to disjoint coverage, a MU is served by only one SBS at any particular time. We assume that time is divided into equal-length time slots, the duration of which corresponds to the minimum time a MU remains in the coverage area of the same SBS. We also assume that each SBS is capable of transmitting B bits to a MU within its coverage area in a single time slot.

We consider a content library of K files, $\mathbb{V} = \{v_1, \ldots, v_K\}$, each of size F bits. The request probability of v_k is p_k, where $p_1 \geq p_2 \geq \cdots \geq p_K$. Since a video file has F bits and the transmission rate of a SBS is B bits per time slot, a MU cannot download a video file before $T = F/B$ time slots. For simplicity we assume T is an integer, and call the T time slots following a user request a *download session*. Although a MU is connected to only one SBS at each time slot, due to mobility, it connects to multiple SBSs over the same download session. We note that, due to the limited

cache size, an SBS cannot store all the library, and the uncached video files are offloaded to the MBS.

A. User Mobility

The *mobility path* of a MU is defined as the sequence of SBSs it visits within the same download session. For instance, for $T = 5$, $SBS_1, SBS_3, SBS_4, SBS_5, SBS_6$ is a possible mobility path. We consider a *high mobility* scenario, where at the end of each time slot the MU moves to a neighboring cell it has not visited within the same download session. Under this assumption a mobility path is a sequence of T distinct SBSs.

B. Delay-aware coded caching

We explain here the coding scheme that is used to encode the stored video files. Each file is divided into T disjoint segments of size B bits each, i.e., $v_k = \left(s_k^{(1)}, \ldots, s_k^{(T)}\right)$. These segments are grouped into M_k disjoint fragments $f_k^{(1)}, \ldots, f_k^{(M_k)}$; that is,

$$v_k = \bigcup_{m=1}^{M_k} f_k^{(m)}, \quad \text{and } f_k^{(i)} \cap f_k^{(j)} = \emptyset, \tag{1}$$

for any $i, j \in \{1, \ldots, M_k\}$ and $i \neq j$. Then, the segments in fragment m are jointly encoded using an $\left(|f_k^{(m)}|, N\right)$ MDS code, and each coded segment is cached by a different SBS. Hence, fragment $f_k^{(m)}$ can be recovered from any $|f_k^{(m)}|B$ parity bits collected from any $|f_k^{(m)}|$ different SBSs within $|f_k^{(m)}|$ time slots (please refer to [12] for further discussion on the coded storage strategy).

The reason for constructing N coded segments is to ensure that a MU does not receive the same coded segment more than once in any possible path. We remark that, for a given T, certain cells can not be visited within the same mobility path; hence, depending on T, less than N coded segments may be sufficient to achieve this [13].

Definition 1. *A coded caching policy* $\mathbf{X}$ *defines how each file* v_k *is divided into fragments, i.e.,* $\mathbf{X} \triangleq \{\mathbf{X}_k\}_{k=1}^K$*, where* $\mathbf{X}_k = \left\{f_k^{(1)}, \ldots, f_k^{(M_k)}\right\}$*. A caching policy is feasible, if* $\sum_{k=1}^K M_k B \leq C$*.*

C. Continuous video display and delay analysis

The *video display rate*, λ, defines the average amount of data (bits) required to display a unit duration (normalized to one time slot) of a video file. In this work, we consider the scenario in which the service rate of the SBSs and the video display rate of MUs are approximately equal, i.e., $B \approx \lambda$. Hence, at each time slot the MU displays one segment and can also download data of the size of one segment. In order to display a segment, it should be available at the MU buffer in an uncoded form. If a segment is not available in the buffer, then the user waits until it becomes available. This waiting time is called the *re-buffering delay*.

The cumulative re-buffering delay for file v_k, under policy $\mathbf{X}_k$, is denoted by $D_k(\mathbf{X}_k)$, and it is equal to the sum of re-buffering delays experienced within a video streaming session. For the delay analysis, consider a particular file with M fragments, i.e., $\left\{f^{(1)}, \ldots, f^{(M)}\right\}$. The *display duration* of a fragment is the number of segments in it. If there is only one fragment, the display duration of that fragment is equal to the video duration. Let $d^{(m)}$ denote the display duration of fragment $f^{(m)}$, i.e., $d^{(m)} = |f^{(m)}|B/\lambda \approx |f^{(m)}|$. Furthermore, let $t_d^{(m)}$ and $t_p^{(m)}$ denote the time instants at which the mth fragment is downloaded and started to be displayed, respectively. If $t_p^{(m)} > t_d^{(m)}$, the user displays the mth fragment without experiencing any stalling; however, if $t_d^{(m)} > t_p^{(m)}$, then the user enters a re-buffering period and it stops displaying the video until $t_d^{(m)}$. Accordingly, the re-buffering duration for the mth fragment, $\Delta^{(m)}$, can be formulated as

$$\Delta^{(m)} = \max\left\{t_d^{(m)} - t_p^{(m)}, 0\right\}. \tag{2}$$

Note that $t_p^{(m)}$ is equivalent to the sum of the display times and re-buffering delays experienced by the previously displayed fragments, i.e., $t_p^{(m)} = \sum_{i=1}^{m-1} \Delta^{(i)} + d^{(i)}$.

Similarly, assuming that the fragments are downloaded in order, $t_d^{(m)}$ is the total download time of all the previous fragments, i.e., $t_d^{(m)} = \sum_{i=1}^{m} d^{(i)}$. Hence, (2) can be rewritten as $\Delta^{(m)} = \max\left\{d^{(m)} - \sum_{i=1}^{m-1} \Delta^{(i)}, 0\right\}$. We observe that if $\Delta^{(m)} > 0$, then we have $\sum_{i=1}^{m} \Delta^{(i)} = d^{(m)}$. Let D be the cumulative re-buffering delay experienced over all fragments of the video, which is derived by the following lemma.

Lemma 1. *The cumulative re-buffering delay* D *is equal to the display duration of the largest fragment, i.e.,*

$$D = \sum_{m=1}^{M} \Delta^{(m)} = \max\left\{d^{(1)}, \ldots, d^{(m)}\right\}. \tag{3}$$

Lemma 1 can be easily proved by induction using the equality $\sum_{i=1}^{m} \Delta^{(i)} = d^{(m)}$, and the fact that $\Delta^{(1)} = d^{(1)}$. Note that if the first fragment has the largest display duration, then $D = \Delta^{(1)}$, and the cumulative re-buffering delay is equal to the initial buffering delay.

D. Problem formulation

Our goal is to find the optimal coded caching policy $\mathbf{X}$ that minimizes the cumulative re-buffering delay averaged over all files, i.e., $D_{avg}(\mathbf{X}) = \sum_{k=1}^{K} p_k D_k(\mathbf{X}_k)$. We first focus on a particular file and highlight the delay-cache capacity trade-off with an example. If the number of fragments is equal to the number of segments, i.e., $f^{(m)} = \left\{s^{(m)}\right\}$, $\forall m \in \{1, \ldots, T\}$, then each SBS caches all the segments. This requires a cache memory

of $F = TB$ bits for the corresponding file. On the other hand, if there is only one fragment that contains all the segments, i.e., $f^{(1)} = \left\{s^{(1)}, \ldots, s^{(T)}\right\}$, then all the segments are jointly encoded, and each SBS caches only B bits for the corresponding file. However, while the re-buffering delay in the first scenario is only 1, it is T in the second.

Recall that the required cache size for a file is MB bits, which depends only on the number of fragments M. However, the cumulative re-buffering delay is equal to the display time (the number of segments) of the largest fragment. Hence, for given M the cumulative re-buffering delay can be minimized by choosing fragment sizes approximately equal, i.e., for any $i, j \in \{1, \ldots, M\}$, and $i \neq j$, $|d^{(i)} - d^{(j)}| \leq 1$. Consequently, for a given memory constraint of MB bits the minimum achievable cumulative re-buffering delay is $\lceil T/M \rceil$.

To mathematically capture this relationship, we introduce the *delay-cache capacity* function $\Omega(M) \triangleq \lceil T/M \rceil$, which maps the number of fragments in a file to the minimum achievable re-buffering delay D. We note that $\Omega(M)$ is a monotonically decreasing step function. To analyze $\Omega(M)$, we introduce two new parameters: the *delay level* and the *decrement point*. Any possible value of $\Omega(M)$ is called a delay level, denoted by $D^{(l)}$.

Recall that the popularity of the files are not identical, which implies that re-buffering delay of popular files has a greater impact on the average. Hence, for each file v_k, we consider a weighted delay-cache capacity function $\Omega_k(M_k)$, where $\Omega_k(M_k) \triangleq p_k \lceil T/M_k \rceil$. Note that for a given number of fragments M, we know the optimal caching decision, i.e., the number of segments in each fragment. Hence, from now on, we use $\mathbf{M} \triangleq (M_1, \ldots, M_K)$ to denote the caching policy (instead of $\mathbf{X}$). Then the average re-buffering delay is rewritten as $D_{avg}(\mathbf{M}) = \sum_{k=1}^{K} \Omega_k(M_k)$. Eventually, we have the following optimization problem

$$\textbf{P1:} \quad \min_{\mathbf{M}} D_{avg}(\mathbf{M})$$
$$\text{subject to: } D_k(M_k) \leq D_{max}, \ \forall k, \tag{4}$$
$$\sum_{k=1}^{K} M_K B \leq C, \tag{5}$$

where (4) is the fairness constraint which ensures that the cumulative re-buffering delay is less than D_{max} for any video file, and (5) is the cache capacity constraint.

III. Solution Approach

Lets denote the minimum l that satisfies $D^{(l)} < D_{\max}$ in **P1** by $l_{\min}$. Then, the optimization problem **P1** can be reformulated as

$$\textbf{P2:} \quad \min_{\mathbf{M}} D_{avg}(\mathbf{M})$$
$$\text{subject to: } M_k \geq m^{(l_{\min})}, \forall k, \tag{6}$$
$$\sum_{k=1}^{K} M_k \leq C/B. \tag{7}$$

Algorithm 1: Cost-free delay minimization

Input : B,C,$\left\{\left\{\gamma_{k,l}\right\}_{l=1}^{L}\right\}_{k=1}^{K}$
Output: $\mathbf{M}$

```
M_k ← m^(l_min), γ_k ← γ_{k,l_min}, k ∈ {1,...,K};
l_k ← l_min, C̃ ← C/B;
while C̃ > 0 do
    ḱ = argmax {γ_1,...,γ_K};
    if C̃ ≥ (m^(l_ḱ+1) − m^(l_ḱ)) then
        l_ḱ ← l_ḱ + 1;
        γ_ḱ ← γ_{k,l_ḱ};
        M_k ← m^(l_ḱ);
        C_B ← C_B − (m^(l_ḱ) − m^(l_ḱ−1))
    else
        M_ḱ ← M_ḱ + C_B, C̃ ← 0;
    end
end
```

Note that we simply converted the delay constraint to a cache capacity constraint, such that each file requires a cache capacity of at least $m^{(l_{\min})}B$ bits. In order to find a feasible solution to **P2**, we need $C \geq Km^{(l_{\min})}B$ bits. In the following section, first we solve **P2** assuming that this condition holds. The other case will be considered in the subsequent section. Note that, if (6) does not hold for all the files, some of the least popular files are not cached at all, and a MU requesting one of these files is offloaded to the MBS causing additional overhead. Later we will show how this overhead can be modeled. We call a caching strategy *cost-free* if allthe video files are cached by SBSs.

A. Cost-free delay minimization

P2 can be shown to be an NP hard problem, as it can be reduced to the knapsack problem. However, if we use a piecewise linear approximation of the delay-cache capacity function $\Omega_k(M_K)$, which is denoted by $\tilde{\Omega}_k(M_K)$, then the objective function becomes the sum of piecewise monotonic linear functions. Let $\gamma_{k,l}$ be the slope of the function $\tilde{\Omega}_k(M_K)$, in the interval $(m^{(l)}m^{(l+1)}]$. Then, it is easy to observe that $|\gamma_{k,l}| > |\gamma_{k,l+1}|$ holds for all l. Hence, if the objective function is replaced by $\tilde{D}_{avg}(\mathbf{M}) = \sum_{k=1}^{K} \tilde{\Omega}_k(M_K)$, we obtain the following convex optimization problem:

$$\textbf{P3:} \quad \min_{\mathbf{M}} \tilde{D}_{avg}(\mathbf{M}) = \sum_{k=1}^{K} \tilde{\Omega}_k(M_K)$$
$$\text{subject to: } M_k \geq m^{(l_{\min})} \text{ for all } k \tag{8}$$
$$\sum_{k=1}^{K} M_k \leq C/B. \tag{9}$$

Note that the solution of **P3** is not equivalent to the solution of the original problem **P2**. However, we will show that with a small perturbation in the cache size C, the two solutions become identical. Since the objective is a convex function of sum of piecewise linear functions, we follow a similar strategy to the one used in [10]. The proposed algorithm first allocates each file a cache

memory of size $m^{(l_{\min})}B$ bits, which corresponds to the delay level of $D^{(l_{\min})}$. After this initial phase, it searches for the $\tilde{\Omega}_k(M_k)$ that has the minimum slope (maximum negative slope), and updates the delay level of file v_k to the next one, i.e., $D^{(l)}$ to $D^{(l+1)}$, and updates M_k accordingly. The procedure is repeated until (9) is satisfied with equality. The overall coded caching strategy is detailed in Algorithm 1. Please refer to [12] for further discussions on the optimality of Algorithm 1.

B. Average delay constrained cost minimization

In some cases, it may not be possible to satisfy the D_{max} constraint for all the files in the library due to the cache capacity constraints. Furthermore, the average re-buffering delay can be a predefined system parameter, denoted by D_{avgMax}, in order to offer a certain QoS to the users; however, the average delay obtained from the solution of **P2** may not satisfy this requirement. As a result, some of the least popular files are not cached at all and the requests for these files are offloaded to MBS.

We denote the average amount of data that needs to be downloaded from the MBS by Θ, and the set of cached videos by $A = \{k : M_k > 0\}$. Then, we have $\Theta = \sum_{k \notin A} p_k$. Our goal is to find the coded caching policy $\mathbf{M}$ that minimizes Θ:

$$\textbf{P4:} \quad \min_{\mathbf{M}} \Theta(\mathbf{M}) = \sum_{k \notin A} p_k$$

$$\text{subject to: } D_{avg}(\mathbf{M}) \leq D_{avgMax}, \tag{10}$$

$$M_k \geq m^{(l_{\min})}, \ \forall \ k \in A, \tag{11}$$

$$\sum_{k=1}^{K} M_k \leq C/B. \tag{12}$$

Constraint (10) is for the maximum average delay requirement, (11) is the fairness constraint for the locally cached files; and (12) imposes the cache capacity constraint. Due to (11), at most $\acute{K} = min(\frac{C/B}{m^{(l_{\min})}}, K)$ different files can be stored in the SBS caches. If the most popular $\acute{K}$ files are cached according to the delay constraint D_{max}, $m^{(l_{\min})}B$ bits allocated to each file, then the cache memory size and the fairness constraints are satisfied. If the constraint (10) is also satisfied, i.e., $D_{avgMax} = D_{max}$, then the aforementioned assignment is optimal, and no further steps are needed. Otherwise, in order to decrease D_{avgMax}, the least popular file in A is removed and Algorithm 1 is applied to find the optimal cache allocation for the remaining files.

The use of Algorithm 1 ensures that the allocation yields the minimum possible average cumulative re-buffering delay under the given cache capacity constraint. Using this procedure, we increase the average cost by the least possible amount while decreasing the average delay by the highest possible amount. This step is repeated until all the constraints are satisfied. The overall procedure is illustrated in Algorithm 2.

Algorithm 2: Delay constrained cost minimization

Input : B,C, D_{avgMax}
Output: $\mathbf{M}$
$M_k \leftarrow 0, k \in \{1, \ldots, K\}$,$\tilde{C} \leftarrow C/B$;
for $k \in \{1, \ldots, K\}$ **do**
  if $\tilde{C} \geq m^{(l_{\min})}$ **then**
    $M_k \leftarrow m^{(l_{\min})}$, $\tilde{C} \leftarrow \tilde{C} - m^{(l_{\min})}$;
  end
end
execute Algorithm 1;
while $D_{avg} > D_{avgMax}$ **do**
  $k = \text{argmin}\{p_i\}, i \in \{1, \ldots, K : M_i > 0\}$;
  $M_k \leftarrow 0$;
  execute Algorithm 1;
end

IV. Numerical results

A. Simulation setup

We consider a video library of $K = 10000$ files, where the popularity of the files follows a Zipf distribution with parameter $w = 0.85$, which adjusts its skewness. We set $T = 10$ and $D_{max} = 10$. we consider two different scenarios. First, we consider the cache sizes, normalized over the library size, $\hat{C} \in [0.1, 0.7]$ so that D_{max} can be satisfied for each video file. For this scenario we analyze the average cumulative re-buffering delay with respect to the cache size. In the second scenario, we consider $\hat{C} = 0.08$, where D_{max} constraint cannot be satisfied for all the files; and thus, for the second scenario we analyze the trade-off between the average cost and the average cumulative re-buffering delay.

B. Simulation results

In the simulations we consider tow benchmarks, namely; *most popular file caching* (MPFC) and the *equal file caching* (EFC). In MPFC, initially, a cache size enough to satisfy D_{max} is allocated to all files, then, starting from the most popular file, allocated cache size is made equal to file size until no space is left in the caches of SBSs. In EFC, again we use the same initial cache size allocation, then starting from the most popular file the allocated cache size is increased to the next decrement point. Once, the cache size of the each file is aligned to the next decrement point, we go back to the most popular file and repeat the process until no empty space is left in the caches.

In the first simulation scenario, the cost-free delay minimization algorithm is executed and the results are shown in Figure 1. The average cumulative re-buffering delay of the system is plotted against the available cache size for the proposed caching scheme and the two benchmarks. The proposed caching policy is observed to have better performance than the two benchmarks in all the scenarios, and in some points the average delay is reduced up to 35% with respect to the benchmark with the best performance at this point.

In the second simulation scenario, in which the cache

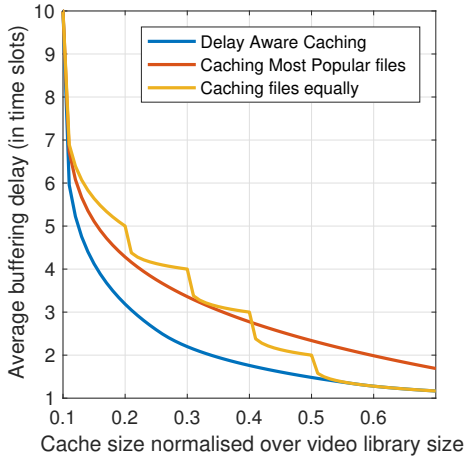

Fig. 1: Average buffering delay vs cache size

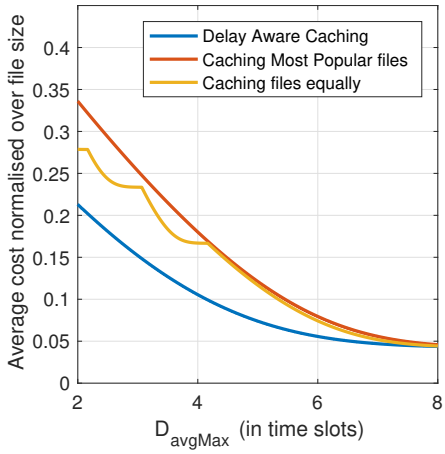

Fig. 2: Average cost vs maximum average delay

size is not sufficient to satisfy delay constraint D_{avgMax}, Algorithm 2 is executed, and its performance is compared with MPFC and EFC policies in Figure 2.

MPFC with a given D_{avgMax} constraint is executed according to the following strategy: first the most popular $\frac{C/B}{m^{(t_{\min})}}$ files are cached according to the maximum allowed delay D_{Max}. If the average delay constraint is not satisfied, i.e., $D_{max} > D_{avgMax}$ then the least popular file that is cached is removed, and the corresponding cache memory is used for the most popular file that is not cached up to the maximum level. This procedure is repeated until the average delay constraint D_{avgMax} is satisfied for all the cached files. For the EFC benchmark, again after the initial step, if the average delay constraint D_{avgMax} is not satisfied, then the least popular file in the cache is removed. The equal file caching algorithm described above is applied subsequently on the files that are still in the cache. This procedure is repeated until the average delay constraint is satisfied for all the cached files. The graph portrays the relationship between the average cost and the average delay constraint D_{avgMax}. Our proposed solution exhibits significant improvement in comparison with the benchmark policies. For example, $D_{avgMax} = 2$, the average cost is improved by 23% and 36% with respect to EFC and MPFC, respectively. As it is expected, the tighter the average delay constraint D_{avgMax}, the higher the cost.

V. Conclusion

We first proposed a caching policy that minimizes the average cumulative re-buffering delay under the high mobility assumption, where a MU does not visit the same SBS within the same download session. We then considered a scenario in which the average cumulative re-buffering delay is a given system requirement, and introduced a caching policy that minimizes the amount of data downloaded from the MBS while satisfying this requirement. Numerical simulations have been presented, showcasing the improved performance of the proposed caching policy in comparison with other benchmark caching policies. General user mobility patterns will be studied as a future extension.

References

[1] Sandvine, "2016 Global internet phenomena report: LATIN AMERICA & NORTH AMERICA," June 2016, White Paper.

[2] Cisco, "Cisco visual networking : Forecast and methodology, 2016-2021," June 2017, White Paper.

[3] W. Jiang, G. Feng, and S. Qin, "Optimal cooperative content caching and delivery policy for heterogeneous cellular networks," *IEEE Trans. Mobile Comput.*, vol. 16, no. 5, May 2017.

[4] M. Dehghan, B. Jiang, A. Seetharam, T. He, T. Salonidis, J. Kurose, D. Towsley, and R. Sitaraman, "On the complexity of optimal request routing and content caching in heterogeneous cache networks," *IEEE/ACM Trans. on Netw.*, vol. 25, June 2017.

[5] K. Poularakis, G. Iosifidis, and L. Tassiulas, "Approximation algorithms for mobile data caching in small cell networks," *IEEE Trans. on Comms.*, vol. 62, no. 10, pp. 3665–3677, Oct. 2014.

[6] K. Shanmugam, N. Golrezaei, A. Dimakis, A. Molisch, and G. Caire, "FemtoCaching: Wireless content delivery through distributed caching helpers," *IEEE Trans. Inf. Theory*, Dec. 2013.

[7] J. Liao, K. K. Wong, Y. Zhang, Z. Zheng, and K. Yang, "Coding, multicast, and cooperation for cache- enabled heterogeneous small cell networks," *IEEE Trans. Wireless Comm.*, vol. 16, Oct. 2017.

[8] R. Wang, X. Peng, J. Zhang, and K. Letaief, "Mobility-aware caching for content-centric wireless networks: modeling and methodology," *IEEE Comms. Mag.*, vol. 54, no. 8, Aug. 2016.

[9] K. Poularakis and L. Tassiulas, "Code, cache and deliver on the move: A novel caching paradigm in hyper-dense small-cell networks," *IEEE Trans. Mobile Comput.*, vol. 16, Mar. 2017.

[10] E. Ozfatura and D. Gündüz, "Mobility and popularity-aware coded small-cell caching," *IEEE Commun. Lett.*, vol. 22, Feb. 2018.

[11] K. Kanai, T. Muto, J. Katto, S. Yamamura, T. Furutono, T. Saito, H. Mikami, K. Kusachi, T. Tsuda, W. Kameyama, Y. J. Park, and T. Sato, "Proactive content caching for mobile video utilizing transportation systems and evaluation through field experiments," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 8, Aug. 2016.

[12] E. Ozfatura, T. Rarris, O. Ercetin, and D. Gündüz, "Delay-aware coded caching for mobile users," will be available on arXiv.

[13] E. Ozfatura and D. Gündüz, "Mobility-aware coded storage and delivery," in *Int'l ITG Workshop on Smart Antennas*, March 2018.