Joint Consideration of Content Popularity and Size in Device-to-Device Caching Scenarios

Georgios Kollias*, Angelos Antonopoulos†
*Universitat Politecnica de Catalunya (UPC), Barcelona, Spain
†Centre Tecnologic de Telecomunicacions de Catalunya (CTTC/CERCA), Barcelona, Spain
Email: georgios.kollias@upc.edu, aantonopoulos@cttc.es

***Abstract*—Content caching has been considered by both academia and industry as an efficient solution to tackle the problem of the back-haul becoming the bottleneck in the service of users in future heterogeneous cellular networks. Most of the related caching-oriented studies are based on the content popularity, overlooking the impact of content size on their analysis. In this context, this work studies content caching in an environment where cellular users are equipped with cache memories. In particular, we formulate the content caching as an optimization problem, where the objective is to minimize the average download latency of popular videos through self-caching and device-to-device (D2D) caching and, consequently, increase the network throughput. In addition, in order to solve this problem in real-time scenarios, we introduce a low-complexity utility-based algorithm, which accounts for parameters such as the size and the popularity of the requested contents, as well as the density of the end users. Finally, we provide extensive simulation results that validate our analysis and prove that our innovative scheme outperforms other existing solutions.**

***Index Terms*—Caching, Device-to-Device communication, 5G.**

I. Introduction

Cellular networks will have to come up against a tremendous traffic growth over the next few years, which is expected to reach 77 exabytes per month by 2022. Interestingly, 79% of this traffic will come from video which is likely to increase 9-fold over the same period [1]. Densifying traditional networks with small cells (SCs) has been considered a solution to the envisioned capacity crunch over the next years [2]. In order to support the requested high data rates and given the increased cost for installing high capacity mid-haul or/and back-haul between the numerous SCs and the overlaid macro base stations (MBSs) or/and the core network, the concept of content caching has emerged [3]. The necessity for high rates and low latency with regard to video downloading is outlined in the white paper from Akamai [4], where it can been seen that even few seconds of startup delay could result in almost 50% of users abandoning the view of a requested video.

In this framework, it has been considered that caching popular contents in the cache memory of the mobile devices could result in a two-fold gain: i) offload traffic from the core network, since requests for the same content can be satisfied locally through device-to-device (D2D) communication and ii) achieve high downloading times given the high rates that D2D links can provide [5]. In particular, installing cache memories to the end users allows a content request to be satisfied locally, thus avoiding using x-haul (back/mid/front-haul) connections and, consequently, decreasing service time. Specifically, a user equipment (UE) could find part of the requested content either in its own cache or in the cache memory of neighbouring UEs. Hence, the density of UEs impacts content caching, given that it affects the number of contents to which a potential requesting UE has access to, as well as the time needed for a request to be satisfied. In the case where the request cannot be satisfied through caching, the missing part of the request should be fetched from the content server and delivered to the user through the x-haul.

What is common in the majority of the cache-related works in the literature is that they underestimate or overlook the impact of content size on defining an efficient content placement technique. The latter could be due to the fact that the relationship between unequal content size and popularity of videos has not been quantified and analysed yet [6], and could explain why most works either consider in their analysis equal sized contents [7] or even that the size of all the available contents is equal to the unit [8].

More specifically, in a scenario with equal sized contents, it can be easily proved that the preferred caching strategy would be to cache higher portions of the most popular ones, given that these will be requested more times, and therefore less traffic will have to be offloaded from the cellular network. However, when the requested contents are characterized by unequal size, the decision about which contents to cache and at which portion, is not straightforward. Specifically, it has to be investigated whether a content of low popularity and large size should be cached more over a more popular and smaller file in order to reduce network's traffic load, given that multiple requests for the same content might be generated in a relatively short time period.

In this paper, motivated from the above, we investigate content caching aiming at reducing the average downloading latency for popular videos. In our study, we consider content size as an important parameter that should be taken into account when designing caching strategies. In order to i) quantify the impact of the most relevant parameters (i.e., user density, content popularity and size) on the average content downloading delay, and ii) choose the most efficient content placement scheme with respect to the network's parameters, we formulate a content download time minimization problem to provide closed-form expressions. Finally, these expressions are integrated in a proposed utility-based low-complexity algorithm to solve this problem in real-time scenarios and provide the appropriate content placement each time, which can be easily adapted in existing network architectures.

The main contributions of this paper are summarized as follows:

- We study and analyze how content size, jointly with content popularity and density of cache-enabled UEs, can affect the average content downloading delay. Furthermore, we derive closed form expressions to quantify the impact of the aforementioned parameters.
- We formulate a caching optimization problem for the minimization of the average content downloading delay for a set of popular videos, and we introduce a low-complexity *D2D* caching algorithm based on *Popularity* and *Size* of popular contents, from now on denoted as *DPS*, to solve the problem of average content downloading delay. The algorithm, is based in a utility function that uses the derived closed form expressions, and takes into account the content size and popularity, the density of UEs and the limitations in the caching memories.
- We provide extensive simulation results to validate our analysis and show that our proposed scheme outperforms existing solutions in terms of average content download delay, while it also adapts better to varying network parameters.

The remainder of this paper is organized as follows: we introduce our system model in Section II, while in Section III, the caching analysis is presented. Section IV contains our video download latency analysis and the optimization problem for minimizing the average video downloading delay, through caching, while a low complexity algorithm is also proposed. Finally, numerical results are presented in Section V. Section VI concludes the paper.

II. System Model

In our approach, we consider a network topology that consists of a number of SCs, which are connected to an overlaid MBS through a mid-haul connection with finite capacity (Fig. 1). The MBS, which is used to offer full-area coverage, is directly connected to a content server. Inside the coverage area of each SC, defined with a radius R_{SC}, UEs are distributed according to a homogeneous Poisson Point Process (PPP) with intensity λ. We define two sets of UEs, namely $\mathcal{U}_{nr}$ and $\mathcal{U}_r$, with $\mathcal{U}_{nr} \cap \mathcal{U}_r = \emptyset$, that stand for the UEs that request non-real and real-time traffic services, respectively. A UE requests a real-time content with probability p_r and consequently the density of such UEs, denoted as λ_r, equals $\lambda_r = p_r \lambda$. Similarly, the rest of the UEs that request non-real-time contents are deployed according to a homogeneous PPP with intensity $\lambda_{nr} = (1 - p_r)\lambda$. It holds that $\lambda = \lambda_r + \lambda_{nr}$. All UEs are equipped with a cache memory of size M_{UE}. Moreover, UEs may communicate among each other using D2D communication protocols, as long as their distance is lower than the maximum communication range of D2D communications, defined as R_{D2D}. Finally, throughout this paper, we consider that M_{UE} is divided to segments of equal size, denoted as s_{seg}.

In our scenario, we focus on the service of non-real-time traffic UEs that request files[1] asynchronously from a content library, denoted as $\mathcal{F} = \{1, ..., N\}$. Each file is characterized by a unique profile $f_i = (s_i, q_i)$, $\forall i \in [1, N]$, where s_i and q_i stand for the size and the popularity of file i, respectively. It should be noted that segmented content caching [9] and, specifically, erasure coding is considered ([10]), while we also consider that the number of encoded segments is high enough to promote erasure coding over whole-file replication. Therefore, a content's, f_i, reception is successful when the number of segments received from a UE equals n_i, which is defined as

$$n_i = \frac{s_i}{s_{seg}}. \tag{1}$$

With regard to q_i, the popularity of the requested contents could be modeled by a zipf-like distribution [11], i.e.,

$$q_i = \frac{1/i^{\gamma}}{\sum_{k=1}^{N} 1/k^{\gamma}} \tag{2}$$

and, consequently, $q_i \geq q_{i+1}$, where γ is a fixed parameter that defines the skewness of the file popularity distribution.

According to the considered network topology, a demand for a content can be satisfied from the cache memory of the requested UEs, from the cache memory of a neighbouring UE or through the mid-haul connecting the overlaid MBS with the nearby SC.

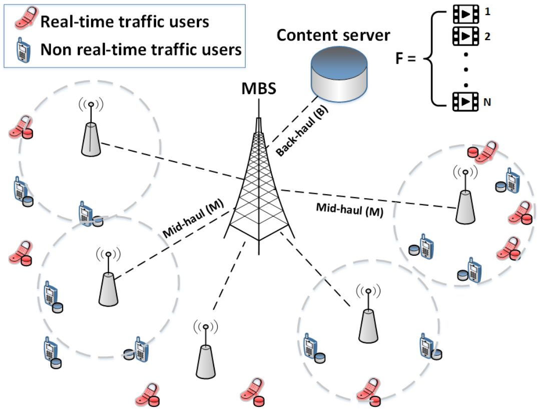

Fig. 1: System model of the cache assisted network

In more detail, when a request for a content f_i is generated by a non-real-time UE, service is offered according to the following steps:

- UE searches in its own cache for the requested content or for segments of it. This process is denoted as *self-caching*.
- When self-caching is not sufficient for a UE to be served entirely, a second caching layer is offered from the cache memories of the neighbouring UEs. The D2D communication between two UEs for caching purposes is characterized throughout this work as *D2D-caching*.
- Finally, in the case where the aforementioned caching options cannot satisfy the demands of a UE, the content

[1] Please note that the terms "file" and "content" are used interchangeably throughout this paper.

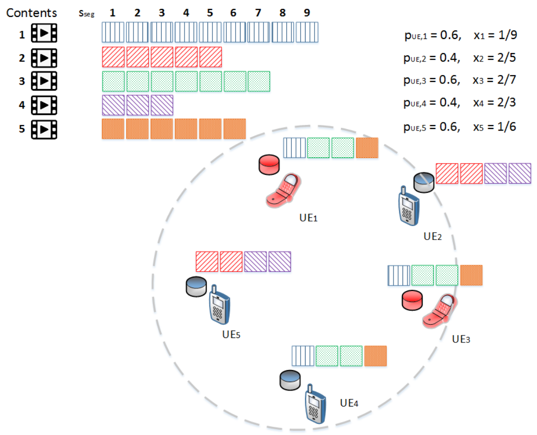

Fig. 2: Toy example: Content caching at the UE side

needs to be downloaded from the overlaid MBS to the nearby SC through the mid-haul connection.

In our model, we consider in-band overlay D2D communication [12] and, therefore, D2D communication does not intervene to cellular downlink traffic. Finally, the convention followed in this work is that the caching procedure is distinguished to the placement phase during off-peak hours, where caches are updated with popular videos, and the delivery phase, where the requested contents are forwarded to the end users during high traffic hours.

III. Caching Analysis

This section presents the caching policies that we consider in our scenario, specifically self-caching and D2D-caching. Furthermore, our analysis on the average number of segments for each requested content i that a UE gets through the aforementioned policies is provided.

A. Caching Policies

As we have already explained, the UE content requests from the library $\mathcal{F}$ could be potentially served from all the network elements with caching capabilities within the UE's communication range. However, the received service depends on parameters such as the number of potential D2D-neighbours and the portion of the requested file cached at the different elements. For instance, a UE with a large number of D2D-neighbours is more likely to receive higher portions of a requested file through D2D-caching rather from the content server.

Motivated by the above, in this section, we study how a non-real-time traffic UE receives a requested content through caching, while we also derive expressions to measure the expected portion of each content i a UE gets from self-caching and D2D-caching. At this point, let us define two probabilities which are related to caching at the UE side. First, we define the caching probability at UEs as

$$P_{UE,i} = [p_{UE,1}, .., p_{UE,N}], \tag{3}$$

where $p_{UE,i}, \forall i \in \mathcal{F}$ stands for the percentage of UEs that have cached part of the i-th file in their cache memory. It is also considered that, for every content i cached at the UE side, the caching percentage is the same. This caching probability, which is denoted as X_i, can be expressed as

$$X_i = [x_1, ..., x_N], \tag{4}$$

where $x_i, \forall i \in \mathcal{F}$ stands for the percentage of content i cached at the caching memories of UEs.

The above concepts can be better seen in the following toy example (Fig. 2)

Toy Example 1. In this toy example (Fig. 2), content caching at the UE side is illustrated. Specifically, five UEs, three non-real-time and two real-time traffic ones, equipped with a cache memory of size four (i.e., $M_{UE} = 4$), are deployed inside the coverage area of a SC. We assume that the content library consists of five files with unequal size, however each content consists of a number of equal sized segments (s_{seg}), and we observe the percentage of UEs that cache each content, as well as the percentage of each content cached at every cache memory. Therefore, it can be observed that three out of the five UEs (i.e., UE_1, UE_3, UE_4) cache part of the content 1, i.e $p_{UE,1} = 0.6$, while each one of the aforementioned UEs caches the same percentage of content 1, specifically one segment out of the nine available, which implies that $x_1 = 1/9$. Similarly, contents 2 and 4 are cached in two UEs with $x_2 = 2/5$ and $x_4 = 2/3$ respectively, while three UEs cache contents 3 and 5 with $x_3 = 2/7$ and $x_5 = 1/6$.

B. X-caching

A UE can receive a content through self-caching or D2D-caching. If the aforementioned ways are not sufficient, then the content should be downloaded from the core network through the MBS. In the sequel, we derive the expressions for the average amount of the requested files, denoted as $\mathbb{E}[R_{i,W}]$ with $W \in \{self, D\}$, that a non-real-time traffic UE receives through caching.

1) Self-caching: According to the definitions presented in Section III-A, a non-real-time traffic user self-caches a portion of a content i with probability $p_{UE,i}$. Therefore, it can be derived that the expected amount of a file i that can be fetched through self-caching can be expressed as

$$\mathbb{E}[R_{i,self}] = s_{seg} \cdot p_{UE,i} \cdot n_{UE,i}, \tag{5}$$

where $n_{UE,i}$ is the number of segments of content i cached in the memory of a UE, calculated as

$$n_{UE,i} = \frac{x_i s_i}{s_{seg}}. \tag{6}$$

2) D2D-caching: A UE, who cannot fetch entirely a content i from its own cache memory, will search for the remaining segments in the cache memories of the neighbouring UEs. The portion that will be received through D2D-caching is a function of the portion that has already been retrieved through self-caching (see Eq. (5)) and the number of D2D-neighbours that may have cached the specific content.

Considering that UEs follow a homogeneous PPP with intensity λ, let us calculate the probability of a non-real-time traffic UE having k-1 D2D-neighbours, similar to [13], as

$$P_{pssn}(k-1, R_D) = \frac{(\lambda \pi R_D^2)^k}{k!} e^{-(\lambda \pi R_D^2)}, \ \forall k \geq 1. \tag{7}$$

Furthermore, let us also define the number of UEs required for a UE to get the requested content, denoted as $k_{UE,i}$, as

$$k_{UE,i} = \left\lceil \frac{1}{x_i} \right\rceil. \tag{8}$$

For instance, going back to our toy example, regarding content 2, it holds that $n_{UE,2} = 2$ and $k_{UE,2} = 3$.

Two cases are distinguished from the above: i) The case where the reference non-real-time traffic UE caches part of the requested content with probability $p_{UE,i}$ and, therefore, $k_{UE,i} - 1$ D2D-neighbours are needed to fetch the entire requested amount, and ii) the case where a UE does not cache the requested content i, meaning that $k_{UE,i}$ D2D-neighbours with the desired content should be within its communication range.

Consequently, it is derived that the expected amount of the file i that can be received through D2D-neighbours can be expressed by

$$\mathbb{E}[R_{i,D}] = s_{seg}(p_{UE,i}\mathbb{E}[\theta_{D,ue,i}] + (1-p_{UE,i})\mathbb{E}[\theta_{D,ue',i}]), \tag{9}$$

where $\mathbb{E}[\theta_{D,ue,i}]$ and $\mathbb{E}[\theta_{D,ue',i}]$ stand for the expected number of segments of content i that can be received through D2D-caching from a UE that has either cached part of content i (ue) or not (ue'). Considering also that a UE needs either $k_{UE,i} - 1$ or $k_{UE,i}$ D2D-neighbours to receive a content i, with the help of Eqs. (6)-(8), we calculate in Eqs. (10) and (11) the two parts of Eq. (9). The last two expressions (i.e., Eqs. (10) and (11)) identify the probability of having less or equal than the required number D2D-neighbours (i.e. $k_{UE,i}-1$ or $k_{UE,i}$) to get a content through D2D-caching and calculate the expected number of segments that can be fetched through D2D-caching in such case. Furthermore, the above expressions account for the case where a UE might have more that the required number of D2D-neighbours (i.e., $k \in (k_{UE,i}, \mathcal{U}_{UE})$, or $k \in (k_{UE,i}+1, \mathcal{U}_{UE})$.

IV. Video Download latency

In this section, we formulate the latency minimization problem and we provide a low-complexity solution for real-time scenarios.

A. Problem Definition

When the mid-haul is saturated and the MBS is congested, the problem that arises is that the video download time cannot meet users' requirements. Therefore, caching popular contents directly to the users' cache memory could prevent users from experiencing such problems. In this section, we study the video download latency and we calculate the average time needed in order to serve the user's demands.

At this point, let us define as $T^x_{s_{seg},i}$, $x \in \{S, D, M\}$ the time needed to transfer a segment of a video (content i) to a requesting user from its own caching memory (S), from a D2D-neighbour (D) or from the remote content server through the available mid-haul (i.e., M). Using Eqs. (5) and (9), it can be derived that the average content download time for a content i can be expressed as

$$\begin{aligned}\mathbb{E}[T_i] = T^S_{s_{seg},i} p_{UE,i} n_{UE,i} + T^D_{s_{seg},i}(p_{UE,i}\mathbb{E}[\theta_{D,ue,i}] + \\ (1-p_{UE,i})\mathbb{E}[\theta_{D,ue',i}]) + T^M_{s_{seg},i}(1 - p_{UE,i}n_{UE,i} - \\ (p_{UE,i}\mathbb{E}[\theta_{D,ue,i}] + (1-p_{UE,i})\mathbb{E}[\theta_{D,ue',i}])).\end{aligned} \tag{12}$$

Finally, using also the contents' popularity distribution, we can define the average download latency (i.e., $\mathbb{E}[T]$) for $\mathcal{F}$ as

$$\mathbb{E}[T] = \sum_{i=1}^{N} q_i \mathbb{E}[T_i], \forall i \in \mathcal{F}. \tag{13}$$

B. Caching Optimization Problem

In this section, considering the analysis in Section III and Section IV-A, we formulate an optimization problem to minimize traffic that travels through the mid-haul and, consequently, minimize the expected video download latency achieved through caching for non-real-time traffic users:

$$\begin{aligned} \min \quad & \mathbb{E}[T] & & (14) \\ \text{s.t.} \quad & 0 \leq p_{ue,i} \leq 1, & \forall i \in \mathcal{F}, \quad & (14a) \\ & 0 \leq x_i \leq 1, & \forall i \in \mathcal{F}, \quad & (14b) \\ & \sum_{i=1}^{N} x_i s_i \leq M_{ue}, & \forall i \in \mathcal{F}, \forall j \in \mathcal{U} \quad & (14c) \end{aligned}$$

In the optimization problem defined in (14), (14a) ensures that the portion of UEs caching content i is bounded by the number of deployed UEs. Constraint (14b) denotes that the amount of a cached content in each cache memory of UEs cannot exceed its actual size. Finally, (14c) guarantees that caching is limited by the maximum allowed capacity of installed caches.

The optimization problem in Eq. (14) is a mixed-integer linear problem (MILP) that can be solved with the help of exhaustive search algorithms [14]. In order to address the computational complexity that MILP problems have, and to be able to get results in real-time scenarios, we have developed a low complexity caching algorithm (DPS) which is presented in the sequel.

C. Utility-based Solution

As it can be seen from Eq. (13), content profile, expressed in the form of both content popularity and size, which is encapsulated in Eq. (12), affects the average content download time. Furthermore, the density of the D2D UEs, in combination with their integrated cache memories, are parameters that also affect the average video download latency.

In this section, we propose a utility-based solution where the objective is to minimize the average download time for the non-real-time traffic UEs by properly caching segments of the requested popular contents.

Considering the optimization problem in Eq. (14), and in order to reduce the complexity of the proposed solution, let us handle the cache memory of the UEs as one. The latter implies that a segment of a popular video is either cached at every UE

$$\mathbb{E}[\theta_{D,ue,i}] = \sum_{k=1}^{k_{UE,i}} P_{pssn}(k-1,R_D)\sum_{m=0}^{k-1} m n_{UE,i}\binom{k-1}{m} p_{UE,i}^{m}(1-p_{UE,i})^{k-1-m} + \sum_{k=k_{UE,i}+1}^{\mathcal{U}_{UE}} P_{pssn}(k-1,R_D)$$
$$\Bigg[\sum_{m=0}^{k_{UE,i}-1} m n_{UE,i}\binom{k-1}{m} p_{UE,i}^{m}(1-p_{UE,i})^{k-1-m} + \sum_{m=k_{UE,i}}^{k} (k_{UE,i}-1) n_{UE,i}\binom{k-1}{m} p_{UE,i}^{m}(1-p_{UE,i})^{k-1-m}\Bigg]. \quad (10)$$
$$\mathbb{E}[\theta_{D,ue',i}] = \sum_{k=1}^{k_{UE,i}+1} P_{pssn}(k-1,R_D)\sum_{m=0}^{k-1} m n_{UE,i}\binom{k-1}{m} p_{UE,i}^{m}(1-p_{UE,i})^{k-1-m} + \sum_{k=k_{UE,i}+2}^{\mathcal{U}_{UE}} P_{pssn}(k-1,R_D)$$
$$\Bigg[\sum_{m=0}^{k_{UE,i}} m n_{UE,i}\binom{k-1}{m} p_{UE,i}^{m}(1-p_{UE,i})^{k-1-m} + \sum_{m=k_{UE,i}+1}^{k} k_{UE,i} n_{UE,i}\binom{k-1}{m} p_{ue,i}^{m}(1-p_{UE,i})^{k-1-m}\Bigg]. \quad (11)$$

or not cached at all (i.e., $p_{UE.i} = \{0,1\}$). Furthermore, as it was aforementioned, the cache memory of UEs is divided into segments of equal size. The maximum number of segments in a UE's cache memory, denoted as N_{UE}^{max}, equals $N_{UE}^{max} = \frac{M_{UE}}{s_{seg}}$. Therefore, in our solution, we define a utility function, U_i, that calculates the impact of the aforedescribed parameters when each available segment is allocated to a different content.

At each step h, of an iterative procedure that will be explained in detail in the sequel, the contents compete with each other for every available segment. The criterion for assigning a cache segment to a content, is the highest reduction in the amount of traffic that will have to be fetched through the mid-haul, denoted as caching gain G_i. The caching gain for each content i is defined as

$$G_i = U_i(n_{UE,i}+1) - U_i(n_{UE,i}), \quad (15)$$

where $U_i(n_{UE,i})$ stands for the current value of the utility function and $U_i(n_{UE,i}+1)$ stands for the value of the utility if the next segment will be allocated to content i. Please note that for $U_i(n_{UE,i})$ it holds that

$$U_i(n_{UE,i}) = q_i\big(p_{UE,i}n_{UE,i} + (p_{UE,i}\mathbb{E}[\theta_{D,ue,i}] + (1-p_{UE,i})\mathbb{E}[\theta_{D,ue',i}])\big). \quad (16)$$

Algorithm 1: DPS

```
   Data: $q_i, s_i, \lambda$
   Result: Caching allocation
 1 Calculate $U_i(n_{UE,i}), \forall i \in \mathcal{F}$
 2 Calculate
 3 for $h \in N_{tot}^{max}$ do
 4     if $h = 1$ then
 5         Calculate $U_i(n_{UE,i}+1)$,$G_i$, $\forall i \in \mathcal{F}$
 6         $i^* = argmax(G_i), \forall i \in \mathcal{F}$
 7         $n_{UE,i^*} \leftarrow n_{UE,i^*} + 1$
 8         Calculate $U_{i^*}(n_{UE,i^*}+1)$,$G_{i^*}$,
 9     else
10         $i^* = argmax(G_i), \forall i \in \mathcal{F}$
11         $n_{UE,i^*} \leftarrow n_{UE,i^*} + 1$
12         Calculate $U_{i^*}(n_{UE,i^*}+1)$,$G_{i^*}$,
13     end
14 end
```

In order to assign cache segments to contents in a beneficial for the average download time way, we propose an iterative caching algorithm, denoted as *DPS* (Algorithm 1). *DPS* is of low complexity, i.e., $O(N_{tot}^{max})$, since the number of iterations cannot exceed N_{tot}^{max}, meaning that it depends on the total number of available segments in the cache memory of UEs. Furthermore, at each step, only the utility functions for one content are calculated, specifically for the one that was allocated a segment in the previous step and, therefore, its new caching gain should be calculated.

More specifically, *DPS* (Algorithm 1), works as follows

- In its first iteration (lines 5-8) the utility function and the caching gain for each content i is calculated (line 5) and then the content i^* with the highest caching gain that will occupy the first segment is selected (lines 6-7). Subsequently, the new value of the utility function and caching gain is calculated for the content that was allocated in the first cache segment (lines 8).
- In every other iteration, and if the number N_{tot}^{max} is not exceeded, the content with the highest caching gain occupies the respective cache segment (line 10) and the new value for both its utility function and caching gain is calculated (lines 11-12).

V. Performance Evaluation

In this section, we evaluate the performance of *DPS* with both numerical and simulation results obtained from our custom-made simulator (implemented in C) with regard to the zipf parameter, $\gamma \in \{0.1, 0.3, 0.5, 0.8, 1, 1.5, 2, 2.5\}$, and the size of content library, $F \in \{20, 40\}$. Furthermore, we compare its performance in terms of average service delay with a static scheme based on popular caching, denoted as *Pop*, and a scheme based on random caching of available contents, denoted as $Random$ [14].

By definition, *Pop* outperforms *Random* for high values of γ, where a small subset of F accounts for the majority of the requests and consequently the requested traffic. On the contrary, *Random* performs better for low values of γ, where the difference in popularity between the most and the least popular file is lower and, therefore, the effect of content size is more prominent.

Figure 3 depicts the service latency of the different caching schemes, considering $N = 40$ and various values of the zipf parameter. As it can be seen, *DPS* outperforms all the other schemes, regardless of the zipf value.

In more detail, it can be observed that, for low values of γ (i.e., $\gamma \in \{0.1, 0.3, 0.5\}$), our proposed scheme surpasses the *Pop* approach by approximately 10%. This is due to the fact that *DPS* manages to capture better the impact of content size on the service delay when the contents' requests are almost equally distributed. On the contrary, for high values of γ (i.e., $\gamma \in \{2, 2.5\}$) and given that the impact of the popularity becomes more intense, *Pop* converges to our proposed algorithm. However, it should be noted that even in these cases, *DPS* reduces service latency compared to *Pop* (i.e., 19% and 9%, respectively). Regarding the performance for medium values of γ (i.e., $\gamma \in \{0.8, 1, 1.5\}$), the different size of the most popular contents is the factor taken in account from *DPS* that results in high latency reductions compared to *Pop*. Specifically *DPS* outperforms *Pop* by 15%, 18% and 22%, respectively, for the aforementioned values of γ. Finally, although *DPS* and *Random* show similar performance for $\gamma = 0.1$, as γ increases, our proposed algorithm outperforms *Random* by even 65% (i.e., when $\gamma = 2.5$).

Similar behavior is observed for lower size of the content library (N=20) (Fig. 4). Specifically, it can be seen that *DPS* outperforms *Pop* and *Random* by even 23% and 34% for low and medium values of γ. However, it can be seen that, in the extreme case where $\gamma \in \{2, 2.5\}$, which according to Eq. (2) is translated to popularity of 63% and 75% for the most popular file, probabilities that are not common in real scenarios, *Pop* outperforms our proposed solution while *DPS* surpasses *Random* by 48%.

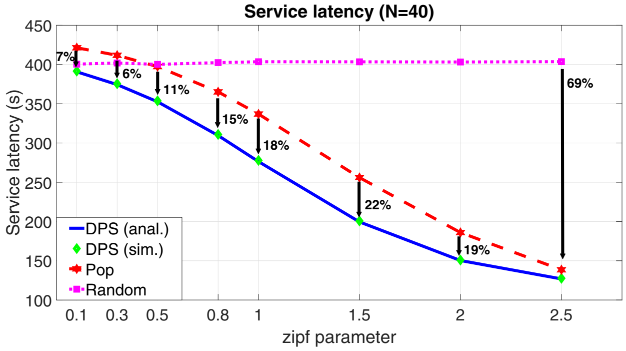

Fig. 3: Service latency, N=40

VI. Conclusions

In this work, we have studied segmented caching in backhaul limited cellular networks where D2D-caching is considered. Specifically, we analyzed the impact of content popularity and size on the latency of non-real-time users, taking also into account parameters such as the user density and the limitations in the capacity of the available cache memories. It has been shown that our dynamic caching algorithm adapts better to different network parameters and outperforms existing schemes based on popular and random caching by 7% to 34% for low to medium values for the zipf parameter γ, while it also shows better performance for high values of γ and increased size of content library.

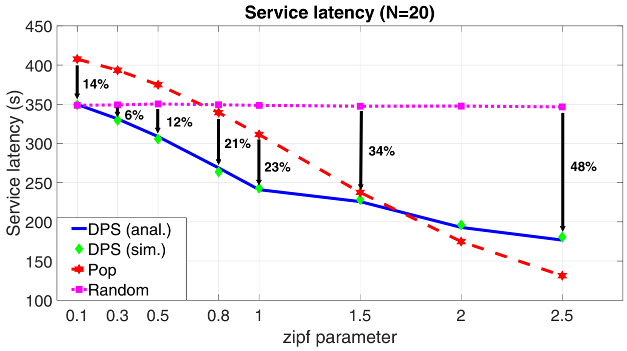

Fig. 4: Service latency, N=20

Acknowledgments

This work has been supported in part by the research projects AGAUR (2017-SGR-891), SPOT5G (TEC2017-87456-P) and 5G-SOLUTIONS (856691).

References

[1] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2017-2022, Cisco Systems, Inc, Feb 2019.

[2] J. G. Andrews, T. Bai, M. N. Kulkarni, A. Alkhateeb, A. K. Gupta and R. W. Heath, "Modeling and Analyzing Millimeter Wave Cellular Systems," in IEEE Transactions on Communications, vol. 65, no. 1, pp. 403-430, Jan 2017.

[3] V. Chandrasekhar, J. G. Andrews and A. Gatherer, "Femtocell networks: a survey," in IEEE Communications Magazine, vol. 46, no. 9, pp. 59-67, Sep 2008.

[4] White Paper "Maximizing Audience Engagement: How online video performance impacts viewer behavior", Akamai, Jan 2015.

[5] A. Afzal, S. A. R. Zaidi, D. McLernon and M. Ghogho, "On the analysis of cellular networks with caching and coordinated device-to-device communication," 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, pp. 1-7. Jul 2016.

[6] H. Feng, Z. Chen and H. Liu, "Design and Optimization of VoD Schemes With Client Caching in Wireless Multicast Networks," in IEEE Transactions on Vehicular Technology, vol. 67, no. 1, pp. 765-780, Jan 2018.

[7] Y. Zhu, G. Zheng, K. Wong, S. Jin and S. Lambotharan, "Performance Analysis of Cache-Enabled Millimeter Wave Small Cell Networks," in IEEE Transactions on Vehicular Technology, vol. 67, no. 7, pp. 6695-6699, Jul 2018.

[8] D. Liu and C. Yang, "Cache-enabled heterogeneous cellular networks: Comparison and tradeoffs," 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, pp. 1-6. Jul 2016.

[9] Z. Chen, Y. Liu, B. Zhou and M. Tao, "Caching incentive design in wireless D2D networks: A Stackelberg game approach," 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, pp. 1-6. Jul 2016.

[10] N. Golrezaei, A. F. Molisch, A. G. Dimakis and G. Caire, "Femtocaching and device-to-device collaboration: A new architecture for wireless video distribution," in IEEE Communications Magazine, vol. 51, no. 4, pp. 142-149, Apr 2013.

[11] L. Breslau, Pei Cao, Li Fan, G. Phillips and S. Shenker, "Web caching and Zipf-like distributions: evidence and implications," INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, New York, NY, 1999, pp. 126-134 vol.1. Aug 2002.

[12] H. Hsu and K. C. Chen, "A Resource Allocation Perspective on Caching to Achieve Low Latency," in IEEE Communications Letters, vol. 20, no. 1, pp. 145-148, Jan 2016.

[13] D. Stoyan, W. S. Kendall, J. Mecke, and L. Ruschendorf. Stochastic geometry and its applications, Wiley Chichester, Jan 2013.

[14] Z. Chang, Y. Gu, Z. Han, X. Chen and T. Ristaniemi, "Context-aware data caching for 5G heterogeneous small cells networks," 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, pp. 1-6. Jul 2016.