Context-Aware Proactive Content Caching with Service Differentiation in Wireless Networks

Sabrina Müller*, Onur Atan†, Mihaela van der Schaar†, Anja Klein*
*Communications Engineering Lab, TU Darmstadt, Germany, {s.mueller, a.klein}@nt.tu-darmstadt.de
†Department of Electrical Engineering, University of California Los Angeles, USA, oatan@ucla.edu, mihaela@ee.ucla.edu

Abstract

Content caching in small base stations (SBSs) or wireless infostations is considered as a suitable approach to improve the efficiency in wireless content delivery. Due to storage limitations, placing the optimal content into local caches is crucial. Cache content placement is challenging since it requires knowledge about the content popularity distribution, which is often not available in advance. Moreover, content popularity is subject to fluctuations as mobile users with different interests connect to the caching entity over time. In this paper, we propose a novel algorithm for context-aware proactive cache content placement. By regularly observing context information of connected users, updating the cache content accordingly and observing the demands for cache content subsequently, the algorithm learns context-specific content popularity online over time. We derive a sub-linear regret bound, which characterizes the learning speed and proves that our algorithm asymptotically maximizes the average number of cache hits. Furthermore, our algorithm supports service differentiation by allowing operators of caching entities to prioritize groups of customers. Our numerical results confirm that by exploiting contextual information, our algorithm outperforms state-of-the-art algorithms in a real world data set, with an increase in the number of cache hits of at least 14%.

Index Terms

This work has been submitted to the IEEE for possible publication. Copyright may be transferred without notice, after which this version may no longer be accessible. Part of this work has been published at IEEE International Conference on Communications (ICC) 2016 [22].
I. INTRODUCTION

Nowadays, more than half of the consumer Internet traffic is video traffic, whose growth is foreseen to continue both within wired and wireless networks [1], [2]. In response to the requirements imposed by the growing demand for content of various types, which has to be delivered to a large number of end users, Content Delivery Networks (CDNs) were developed to serve data traffic [3]–[5]. Already today, more than half of the Internet video traffic is delivered by CDNs and this trend is expected to increase to 72% by 2019 [1]. One of the major components of CDNs is content caching. Its basic idea is to bring content closer to the end user. For this purpose, servers for content storage are placed across the network and users are served from the most suitable server that stores the requested content. Caching content and serving users from nearby servers alleviates both the traffic load in the network and the latency for the end user.

Wireless networks have been experiencing a steep increase in data traffic in recent years [2]. Fueled by the emergence of smart mobile devices with advanced multimedia capabilities and the trend towards high data rate applications such as video streaming, especially mobile video traffic is foreseen to account for the majority of mobile data traffic in a few years [2]. However, despite recent advancements in cellular mobile radio networks, these networks cannot keep up with the massive growth of mobile data traffic [6]. As in its wired counterpart, content caching is envisioned to improve the throughput in wireless content delivery. This is not only due to decreasing disk storage prices, but also due to the fact that typically only a small number of very popular contents account for the majority of data traffic [7].

Within wireless networks, caching at the edge has been extensively studied [8]–[22]. At the radio access network level, current approaches comprise two different types of local caching entities. The first type of caching entities are macro base stations (MBSs) as well as small base stations (SBSs) in wireless small cell networks that dispose of limited storage capacities and are typically owned by the mobile network operator (MNO). The second type of caching entities are wireless infostations that provide high bandwidth local data communication [23]. Caching-enabled infostations could be owned by content providers (CPs) to provide their end users with higher quality of experience or alternatively, third parties could offer caching at infostations as a service to CPs or their end users [19]. For both types of local caching entities, a fraction of
available popular content, such as popular videos, is stored at the caching entities in a *placement phase* to serve end users’ requests in their vicinity directly via localized communication in a *delivery phase*.

Since due to the vast amount of content available in multimedia platforms, not all available content can be stored in local caches, intelligent algorithms for *cache content placement* are required. Many challenges of cache content placement concern content popularity. Firstly, optimal cache content placement primarily depends on the content popularity distribution, but when caching content at a particular point in time, it is unknown which exact content will be requested in future and even an estimation of the content popularity distribution might not be at hand, but has to be learned by the local caching entity itself [15]–[22]. This assumption is not only legitimate from an overhead point of view, since else a periodic coordination with the global multimedia application would be required. More importantly, content popularity at a local caching entity might not even replicate global content popularity as monitored by the global multimedia application [24]–[26]. Hence, local caching entities should learn local content popularity for *proactive* cache content placement. Secondly, due to the mobility of users in the vicinity of a wireless caching entity, the local content popularity is in general not fixed, but may change according to the interests of fluctuating mobile users. The users’ interests may depend on various factors, such as their personal characteristics, their devices’ characteristics [27] or environmental conditions [26]. Hence, cache content placement should be *context-aware* by exploiting information that influences the users’ interests. Thirdly, while the typical goal of cache content placement might lie in the maximization of the number of cache hits, cache content placement should take into account the operator’s specific objective. In particular, appropriate caching algorithms should be capable of incorporating business models of operators to offer *service differentiation* to their customers, e.g., by optimizing cache content according to different prioritization levels [28], [29].

In this paper, we propose a novel context-aware proactive caching algorithm. Instead of assuming a priori knowledge about content popularity, which might be externally given or estimated in a separate training phase, our algorithm learns context-specific content popularity online over time by observing users’ requests for content. By taking into account the users’ context information, the algorithm learns context-specific content popularity, instead of an averaged content popularity over all possible users. The context information can include personal characteristics about the
user and his equipment as well as environmental conditions. Our algorithm learns online the best caching strategy to maximize the number of cache hits at the local caching entity. Moreover, the algorithm allows for operator-specific service differentiation by prioritizing different types of customers. The contributions of this paper are as follows:

- We present a context-aware proactive caching algorithm based on contextual multi-armed bandit optimization which supports service differentiation by prioritization. Our algorithm learns optimal cache content placement online over time in order to maximize the number of cache hits while exploiting the users’ context information.
- We theoretically bound the loss of the algorithm compared to the optimal benchmark, which assumes a priori knowledge about content popularity. By deriving a sub-linear regret bound, we characterize the speed of convergence and prove that our algorithm asymptotically maximizes the average number of cache hits.
- We present additional extensions of our approach, such as its combination with multicast transmissions and the incorporation of rating-based caching decisions.
- We numerically evaluate the context-aware proactive caching algorithm based on a real world data set. A comparison shows that by exploiting contextual information, our algorithm outperforms reference algorithms.

The remainder of the paper is organized as follows. Section II gives an overview of related works. In Section III, we describe the system model, including an architecture and a formal problem formulation for context-aware proactive caching. In Section IV, we propose a context-aware proactive caching algorithm. Theoretical analysis of regret and memory requirements are provided in Sections V and VI respectively. In Section VII, we present some extensions of the algorithm. Numerical results of the learning algorithm are presented in Section VIII. Section IX concludes the paper.

II. RELATED WORK

In this paper, we consider cache content placement for a caching problem with two phases, given by a placement phase and a delivery phase. However, practical caching systems often still use simple cache replacement algorithms that update the cache continuously during the delivery phase. Common examples of cache replacement algorithms are Least Recently Used (LRU) or Least Frequently Used (LFU) (for a survey, see [30]). While these simple cache
replacement algorithms do not consider future popularity of content, recent work has been devoted to developing sophisticated cache replacement algorithms by learning future content popularity [31].

In our overview given below, we focus on research on cache content placement for wireless caching problems with a placement phase and a delivery phase. We start by discussing related work that assumes a priori knowledge about content popularity. Reference [9] calculates information-theoretic gains achieved by combining cache content placement at end user devices with a coded multicast transmission in the content delivery phase. The proposed coded caching approach is optimal up to a constant factor. Reference [10] combines content caching at end devices with collaborative device-to-device communication to increase the efficiency of content delivery. In [11], an approximation algorithm is given for uncoded caching among cache-enabled SBSs which minimizes the average delay experienced by users that can be connected to several SBSs simultaneously. Additionally, it is proven that the problem is efficiently solvable in the coded caching case. Building upon the same caching architecture, in [12], an approximation algorithm is presented for minimizing the probability that moving users have to request parts of content from the MBS instead of the SBSs in distributed coded cache content placement. In [13], a multicast-aware caching scheme is proposed for minimizing the energy consumption in a small cell network, in which both the MBS as well as the SBSs can perform multicast transmissions. Reference [14] analytically calculates the outage probability and average content delivery rate in a network of SBSs equipped with caches.

Next, we discuss related work on cache content placement without prior knowledge about

|                          | [15] | [16] | [17]–[19] | [20] | [21] | This work |
|--------------------------|------|------|-----------|------|------|-----------|
| Model-Free               | Yes  | Yes  | No        | Yes  | Yes  | Yes       |
| Online/Offline-Learning  | Offline | Online | Online | Online | Online | Online    |
| Free of Training Phase   | No   | Yes  | Yes       | No   | Yes  | Yes       |
| Performance Guarantees   | No   | Yes  | No        | No   | Yes  | Yes       |
| Diversity in Content Popularity | No  | No   | No        | Yes  | Yes  | Yes       |
| User Context-Aware       | No   | No   | No        | No   | No   | Yes       |
| Service Differentiation  | No   | No   | No        | No   | No   | Yes       |
content popularity. A comparison of the characteristics of our proposed algorithm with related work of this type is given in Table I. Driven by a proactive caching paradigm, [15], [16] propose a cache content placement algorithm for small cell networks based on collaborative filtering. Fixed global content popularity is estimated using a training set and then exploited for caching decisions to maximize the average user request satisfaction based on their required rates of content delivery. While their approach requires a training set of known content popularities and only learns during a training phase, our proposed algorithm does not need a training phase, but learns content popularity online over time, thus also adapting to varying content popularities.

In [17], using a multi-armed bandit algorithm, an SBS learns a fixed content popularity online by refreshing its cache content and observing instantaneous demands for cached files over time. In this way, the cache content placement at an SBS is optimized over time to maximize the traffic that can be served by the SBS. The authors extend their framework for a wireless infostation network in [18], [19], where they additionally take into account the costs for adding new files to the cache. Moreover, they provide theoretical sub-linear regret bounds for their algorithms. A different extension of the multi-armed bandit framework is given in [20], which by incorporating coded cache content placement exploits the topology of users’ connections to the SBSs. The approach in [20] assumes a specific model of content popularity distribution. Since in practice the content popularity distribution is unknown a priori, such an assumption is restrictive. In contrast, our proposed algorithm is model-free since it does not assume a specific content popularity distribution. Moreover, in [17]–[20], the optimal caching strategy is learned over time based only on previous observations of instantaneous demands. In contrast, our proposed algorithm additionally exploits users’ context information, such that diversity of content popularity within the user population is taken into account. Diversity in content popularity among different users is for example taken into account in [21], but again without considering the users’ contexts. Users are clustered into groups of similar interests by a spectral clustering algorithm based on their requested contents in a training phase. Each user group is assigned to an SBS which then learns the content popularity of its fixed user group over time. Hence, each SBS learns a fixed content popularity under the assumption of a stable user population, whereas our approach allows reacting to arbitrary arrivals of users with content popularity depending on their contexts.

In our previous work [22], we presented a learning algorithm for smart caching at SBSs based on the contextual multi-armed bandit problem [32], [33], an extension of the standard multi-
armed bandit problem [34]. Our proposed algorithm in [22] was inspired by the distributed contextual learning algorithm presented in [33], which considers a general multi-agent learning setting. While several learners select one action each at a time in [33], we considered the case of one learner taking multiple actions at a time, since the sBS has to select multiple files to cache. Our algorithm in [22] took only aggregated context information of connected users into account. In this paper, we extend our approach from [22] in order to take into account context information at a single user level, which enables more fine-grained learning. Additionally, we incorporate service differentiation. Moreover, we present extensions to multicast transmission and rating-based caching decisions, provide a sophisticated regret analysis and evaluate our algorithm with a real world data set.

III. System Model

A. Wireless Caching Entity

We consider a wireless caching entity that can either be a caching-enabled SBS in a small cell network or a wireless infostation. The caching entity is characterized by a limited storage capacity and a reliable backhaul link to the core network. In its cache memory, the caching entity can store up to \( m \) files from a finite content library \( F \) containing \( |F| \) files, where we assume for simplicity that all files are of the same size. To inform users in its vicinity about available files, the caching entity broadcasts the information about currently cached files periodically [17]–[19]. If a user is interested in a file that the caching entity stored in its cache, he requests the file from the caching entity and is served via localized communication. In this case, no additional load is put on neither the macro cellular network nor the backhaul network. If the file is not stored in the caching entity, the user does not request the file from the caching entity. Instead, the user requests the file from the macro cellular network by connecting to an MBS. The MBS downloads the file from the core network via its backhaul connection, such that in this case, load is put on both the macro cellular as well as the backhaul network. In this way, the caching entity cannot observe the requests for files that it has not cached, but at the same time, it is not congested by such requests [17]–[19].

Figure [1] shows an illustration of the considered system model. Within this architecture, the caching entity can only observe the requests for cached files, i.e., cache hits, but it cannot observe the requests for non-cached files, i.e., cache misses. In order to reduce the load on the macro
cellular as well as the backhaul network, a caching entity operated by an MNO might aim at optimizing the cache content such that the traffic it can serve directly is maximized. Maximizing the traffic served by the caching entity corresponds to maximizing the number of cache hits, i.e., the number of requests for files cached at the caching entity. For this purpose, the caching entity should learn which files are most popular over time.

B. Customer-Specific Service Differentiation

Maximization of the number of cache hits as the goal of cache content placement might be adequate in case of an MNO operating an SBS, one reason being limitations imposed by net neutrality restrictions. However, for an operator of an infostation, e.g. a CP or third party operator, the goal might not simply be to cache the most popular content given a set of users. In contrast, according to his business model, the operator may want to provide differentiated services to his customers (that can be both end users and CPs), e.g., by offering different service types corresponding to different prioritization levels. These service types could refer to end users and for instance concern the pricing policy, e.g., a paying user might be prioritized compared to a user with free access to the cache content. This could mean that popular content among paying users is cached preferably. Another possibility is to differentiate end users by interest intensity, e.g., subscribers might obtain priority compared to one-time users. Yet another differentiation could be based on usage patterns. For example, experimental end users might indicate their interest in exploring diverse types of content, while conservative end users might want to be served with only their most preferred content.
Consider a set $S$ of different service types. For service type $s \in S$, let $v_s \geq 1$ denote a weight associated with serving one request of a user with service type $s$ by a cached file in the caching entity. The weights $v_s$ are fixed and known to the learning process. Let $v_{\text{max}} := \max_{s \in S} v_s$. The weights $v_s$ can be of monetary nature or correspond to some virtual value, depending on the purpose of service differentiation. For example, the weight of a service type might be related to the pricing policy applied to that service type, a priority factor for the influence of the service type in terms of advertisement or for the reputation of the operator, or describe a trade-off between experimental and conservative users. The goal of the operator for cache content placement then becomes the maximization of the number of weighted cache hits.

A second service differentiation can be applied in case of a third party operator whose customers are different CPs which buy caching at the wireless infostation as a service. The operator may want to prioritize content of certain CPs according to his business model. More general, the operator could prioritize specific content according to its value for the operator. In this case, cache content placement should take into account each contents’ priority. Consider a prioritization weight $w_f \geq 1$ for each file $f \in F$ and let $w_{\text{max}} := \max_{f \in F} w_f$. The prioritization weights could either be chosen individually for each file or per CP and are fixed and known as well.

The case without any service differentiation, where the goal of cache content placement is the pure maximization of the number of cache hits, is a special case of the scenario described above, in which there is only one service type $s$ with weight $v_s = 1$ and the prioritization weights satisfy $w_f = 1$ for all $f \in F$. While we will refer to the more general case in the subsequent sections, this special case is naturally contained in our analysis.

C. Context-Specific Content Popularity

Content popularity is in general not static, but depends on a user’s preferences. From a user’s point of view, his personal preferences for content may depend on various factors. We refer to such factors as context dimensions and categorize them in the following classes as summarized in Table II. Personal characteristics are for example demographic factors (e.g. age, gender), emotional state and previous experiences. User equipment refers to the type of device (e.g. smart phone, tablet) which is used to access and consume the content, as well as its capabilities and battery status. Environmental conditions are for example given by the location of the user.
and its characteristics, the time of day or the day of the week and external events (e.g. soccer match, concert) that can be both periodic as well as aperiodic. Clearly, this categorization of possible context dimensions is not exhaustive and the impact of each single dimension on content popularity is unknown a priori. Moreover, the caching entity may only have access to part of the context dimensions, for example due to privacy reasons. Under the assumption that the caching entity has access to some information, by observing not only demands for cached files, but also the contexts of users requesting the files, it can learn to cache the most suitable files for the contexts of currently connected users over time. For this purpose, the caching entity updates its cache content periodically.

D. Context-Aware Proactive Caching Architecture

In this subsection, we describe the architecture for context-aware proactive caching, which is designed similarly to an architecture for popularity-driven cache content replacement presented in [31]. An illustration of the context-aware proactive caching architecture is given in Figure 2. The main building blocks of the context-aware proactive caching architecture are a Cache Management entity, the Local Cache, a Storage Interface and a User Interface, a Context Monitor and a Learning Module. The Cache Management consists of a Cache Controller and a Request Handler. The Learning Module contains a Decision Engine, a Learning Database and a Context Database. The workflow consists of several phases as enumerated in Figure 2 and is described
Fig. 2. Context-Aware Proactive Caching Architecture

below.

- **Initialization**

  (1) The Learning Module is provided with the goal of content caching (i.e. pure maximization of number of cache hits or operator-specific goal). Moreover, the Learning Module fixes an appropriate time duration, which specifies the periodicity of context monitoring and cache refreshment and informs the Cache Management and the Context Monitor about the periodicity.

- **Periodic Context Monitoring and Cache Refreshment**

  (2) The Context Monitor periodically gathers context information by accessing the information available at the User Interface about currently connected users and by collecting information from external sources (such as additional information about users from social media platforms). If different service types exist, the Context Monitor also retrieves the service types of connected users. (3) The Context Monitor delivers the gathered information to the Context Database in the Learning Module. (4) The Decision Engine periodically extracts the newly monitored context information from the Context Database. (5) Upon comparison with the results from previous time slots as stored in the Learning Database, (6) the Decision Engine decides which files to cache for the next time slot. (7) The Decision Engine instructs the Cache Controller to refresh the cache content accordingly. (8) The Cache
Controller compares the current cache content with the required one and removes non-required content from the cache. If some required content is missing, the Cache Controller directs the Storage Interface to fetch the content from storage servers and to store it into the local cache. (9) Then, the Cache Controller informs the User Interface about the new cache content. (10) The User Interface pushes the information about the new cache content to currently connected users.

- User Requests
  (11) When a user requests a file available in the local cache, the User Interface forwards the request to the Request Handler. The Request Handler stores the request information. Then it retrieves the requested file from the local cache and serves the request.

- Periodic Learning
  (12) Upon completion of a time slot, the Request Handler hands the information about all user requests from that time slot to the Learning Module. The Learning Module updates the Learning Database with the context information from the beginning of the time slot and the number of requests for cached files in that time slot.

E. Formal Problem Formulation

Next, we give a formal problem formulation for context-aware proactive caching at the caching entity. As described above, the caching system operates in discrete time slots \( t = 1, 2, \ldots, T \), where \( T \) denotes the finite time horizon. Each time slot \( t \) consists of the following sequence of events: (i) The context of currently connected users and their service types are monitored and collected. Let \( U_t \) be the number of users currently connected to the caching entity. We assume that \( 1 \leq U_t \leq U_{\text{max}} \), where \( U_{\text{max}} \) is the maximum number of users that can be connected simultaneously. Let \( D \) be the number of monitored context dimensions per user. We denote the \( D \)-dimensional context space by \( \mathcal{X} \). It is assumed to be bounded and can hence be set to \( \mathcal{X} := [0, 1]^D \) without loss of generality. Let \( x_{t,i} \in \mathcal{X} \) be the context vector of user \( i \) observed in time slot \( t \). Let \( x_t = (x_{t,i})_{i=1}^{U_t} \) be the collection of contexts of all users in time slot \( t \). Let \( s_{t,i} \in S \) be the service type of user \( i \) in time slot \( t \) and let \( s_t = (s_{t,i})_{i=1}^{U_t} \) be the collection of service types of all users in time slot \( t \). (ii) Based on the contexts \( x_t \in \prod_{i=1}^{U_t} \mathcal{X} \), the service types \( s_t \) and their corresponding service weights, as well as the prioritization weights \( w_f \) of each file \( f \in F \) and knowledge from previous time slots, the cache content is refreshed. Then, all
users in the vicinity are informed about the set of currently cached files. This set is denoted by 
$C_t = \{c_{t,1}, \ldots, c_{t,m}\}$. (iii) Until the end of this time slot, users can request currently cached files. Their requests are served. The demands $d_{ct,j}(x_{t,i}, t)$ of each user $i = 1, \ldots, U_t$ for all cached files $c_{t,j} \in C_t$ in this time slot are observed, i.e., the number of requests for each single file currently stored in the cache is monitored. Then, the total sum of (weighted) cache hits in this time slot can be calculated.

The number of times a user with context vector $x \in \mathcal{X}$ requests a file $f \in F$ within one time slot is a random variable with unknown distribution. We denote this random demand by $d_f(x)$ and its expected value by $\mu_f(x)$. The random demand is assumed to take values in $[0, R_{\text{max}}]$, where $R_{\text{max}}$ is the maximum number of possible requests a user can submit within one time slot. This explicitly incorporates that a user can request the same file several times within one time slot. In time slot $t$, the random variables $d_f(x_{t,i}), i = 1, \ldots, U_t$, are assumed to be independent, i.e., given a set of currently connected users, their requests for content are independent of one another. This allows to learn for each of the users’ contexts separately. Moreover, each random variable $d_f(x_{t,i})$ is assumed to be independent of past caching decisions and previous demands. Note that compared to our formulation in [22], where the context space contained context dimensions based on averaged information over all currently connected user, here, we define the context space on a per-user level. Hence, in our new formulation, cache content placement takes the distribution of user contexts into account instead of averaging the context information over all currently connected users, which makes the cache content placement more accurate.

The goal of the caching entity is to optimize the cache content to maximize the total number of (weighted) cache hits up to the finite time horizon $T$. Suppose that for each context vector $x \in \mathcal{X}$, the expected demand $\mu_f(x)$ of all files $f \in F$ would be known. Then, in each time slot $t$, given the context $x_t = (x_{t,i})_{i=1,\ldots,U_t}$ and the service types $s_t = (s_{t,i})_{i=1,\ldots,U_t}$, the optimal solution would be to cache the $m$ files with highest expected (weighted) demands given the users’ contexts and service types. This can be formalized as follows. For contexts $x_t \in \prod_{i=1}^{U_t} \mathcal{X}$ and service types $s_t$, we define the top-$m$ files for the pair $(x_t, s_t)$ as the following $m$ files
\( f^*_1(\mathbf{x}_t, s_t), f^*_2(\mathbf{x}_t, s_t), \ldots, f^*_m(\mathbf{x}_t, s_t) \in F \) which satisfy

\[
 f^*_1(\mathbf{x}_t, s_t) \in \arg \max_{f \in F} \sum_{i=1}^{U_t} v_{s_{t,i}} \mu_f(x_{t,i})
\]

\[
 f^*_2(\mathbf{x}_t, s_t) \in \arg \max_{f \in F \setminus \{f_1^*(\mathbf{x}_t, s_t)\}} \sum_{i=1}^{U_t} v_{s_{t,i}} \mu_f(x_{t,i})
\]

\[
 \vdots
\]

\[
 f^*_m(\mathbf{x}_t, s_t) \in \arg \max_{f \in F \setminus \{f_1^*(\mathbf{x}_t, s_t), \ldots, f_{m-1}^*(\mathbf{x}_t, s_t)\}} \sum_{i=1}^{U_t} v_{s_{t,i}} \mu_f(x_{t,i}).
\]

Then, an optimal choice of files to cache given contexts \( \mathbf{x}_t \) and service types \( s_t \) is given by the set

\[
 C^*_t(\mathbf{x}_t, s_t) = \{ f^*_1(\mathbf{x}_t, s_t), \ldots, f^*_m(\mathbf{x}_t, s_t) \}. \tag{2}
\]

However, we assume that the expected demands are unknown a priori. In this case, the caching entity has to learn the expected demands over time. The optimal solution given in (2) then serves as a benchmark to evaluate the loss of learning the expected demands instead of knowing them a priori. Below, a formal definition of this loss is given, known as the regret of learning. Under the assumption that the context-specific expected demands are unknown a priori, the caching entity has to learn these expected demands over time. For this purpose, the caching entity has to find a trade-off between caching files about which little information is available (\textit{exploration}) and files of which it believes that they will yield the highest demands (\textit{exploitation}). In each time slot, the choice of files to be cached depends on the history of choices in the past and the corresponding observed demands. An algorithm which maps the history to the choices of files to cache is called a \textit{learning algorithm}.

The regret of learning with respect to the optimal benchmark solution is then given by

\[
 R(T) = \sum_{t=1}^{T} \sum_{j=1}^{m} \sum_{i=1}^{U_t} v_{s_{t,i}} \left( w_{f_j^*(\mathbf{x}_t, s_t)} E\left( d_{f_j^*(\mathbf{x}_t, s_t)}(x_{t,i}) \right) \right) - w_{c_{t,j}} E\left( d_{c_{t,j}}(x_{t,i}, t) \right) \tag{3}
\]

\(^1\)Several files may have the same expected demands, i.e., the optimal set of files may not be unique. This is also captured here.
where $d_{c_{t,j}}(x_{t,i}, t)$ denotes the random demand for the cached file $c_{t,j} \in C_t$ of user $i$ with context vector $x_{t,i}$ at time $t$. Here, the expectation is taken with respect to the choices made by the learning algorithm and the distributions of the demands.

IV. A CONTEXT-AWARE PROACTIVE CACHING ALGORITHM

To optimally select which files to cache given the context information about currently connected users, the caching entity should learn context-specific content popularity. The basic idea of our algorithm for context-aware proactive caching is based on the assumption that users with similar context information will more likely request similar files. If this natural assumption holds true, the users’ context information together with their requests for cached files can be exploited to learn for future caching decisions. For this purpose, our algorithm partitions the context space uniformly into smaller sets. Then, the caching entity learns the expected demands for files independently in each of the sets, by estimating the expected demands for files based on the observed demands of users whose contexts belonged to that set. In the algorithm, a time slot $t$ can either be an exploration or an exploitation phase. In exploration phases, the caching entity chooses a random set of files to cache. These phases are needed to update the estimated demands for files which have not been cached often before. In exploitation phases, the caching entity ranks the files according to their estimated demands and caches the ones with the highest estimated demands.

The algorithm for selecting $m$ files is called **Context-Aware Proactive Caching with Cache Size $m$** (m-CAC) and its pseudocode is given in Figure 3. Next, we describe the algorithm in more detail. In its initialization phase, m-CAC creates a partition $\mathcal{P}_T$ of the context space $\mathcal{X} = [0, 1]^D$ into $(h_T)^D$ sets, that are given by $D$-dimensional hypercubes of identical size $\frac{1}{h_T} \times \ldots \times \frac{1}{h_T}$. Here, $h_T$ is an input parameter which determines the number of sets in the partition. Additionally, m-CAC keeps a counter $N_{f,p}(t)$ for each pair consisting of a file $f \in F$ and a set $p \in \mathcal{P}_T$. The counter $N_{f,p}(t)$ is the number of times in which file $f \in F$ was cached after a user with context from set $p$ was connected to the caching entity up to time slot $t$ (i.e., when 2 users with context from set $p$ were connected in one time slot and file $f$ was cached, this counter is increased by 2). Moreover, m-CAC initializes the estimated demand $\hat{d}_{f,p}(t)$ of each pair consisting of a file $f \in F$ and a set $p \in \mathcal{P}_T$ up to time slot $t$. This estimated demand is calculated as follows: Let $\mathcal{E}_{f,p}(t)$ be the set of observed demands of users with context from set $p$ when file $f$ was cached
**m-CAC: Context-Aware Proactive Caching Algorithm**

1. **Input:** $T, h_T, K(t)$
2. Initialize context partition: Create partition $\mathcal{P}_T$ of context space $[0, 1]^D$ into $(h_T)^D$ hypercubes of identical size
3. Initialize counters: For all $f \in F$ and all $p \in \mathcal{P}_T$, set $N_{f,p} = 0$
4. Initialize estimates: For all $f \in F$ and all $p \in \mathcal{P}_T$, set $\hat{d}_{f,p} = 0$
5. **for each** $t = 1, \ldots, T$ **do**
   6. Observe number $U_t$ of currently connected users
   7. Observe user contexts $x_t = (x_{t,i})_{i=1,...,U_t}$ and service types $s_t = (s_{t,i})_{i=1,...,U_t}$
   8. Find the sets $p_{t,i} \in \mathcal{P}_T$ such that $x_{t,i} \in p_{t,i}$, set $p_t = (p_{t,i})_{i=1,...,U_t}$
   9. Compute the set of under-explored files $F_{\text{ue}}^{p_t}(t)$ in (4)
   10. **if** $F_{\text{ue}}^{p_t}(t) \neq \emptyset$ **then**
      11. $u = \text{size}(F_{\text{ue}}^{p_t}(t))$
      12. **if** $u \geq m$ **then**
          13. Select $c_{t,1}, \ldots, c_{t,m}$ randomly from $F_{\text{ue}}^{p_t}(t)$
      14. **else**
          15. Select $c_{t,1}, \ldots, c_{t,u}$ as the $u$ files from $F_{\text{ue}}^{p_t}(t)$
          16. Select $c_{t,u+1}, \ldots, c_{t,m}$ as the $(m-u)$ files $\hat{f}_{1,p_t,s_t}(t), \ldots, \hat{f}_{m-u,p_t,s_t}(t)$ from (5)
      17. **end if**
   18. **else**
      19. Select $c_{t,1}, \ldots, c_{t,m}$ as the $m$ files $\hat{f}_{1,p_t,s_t}(t), \ldots, \hat{f}_{m,p_t,s_t}(t)$ from (6)
   20. **end if**
   21. Observe user demands $(d_{j,i})_{i=1,...,U_t}$ for each file $c_{t,j}$, $j = 1, \ldots, m$
   22. **for** $i = 1, \ldots, U_t$ **do**
      23. **for** $j = 1, \ldots, m$ **do**
      24. $\hat{d}_{c_{t,j},p_{t,i}} = \frac{d_{c_{t,j},p_{t,i}} + d_{j,i}}{N_{c_{t,j},p_{t,i}} + 1}$ and $N_{c_{t,j},p_{t,i}} = N_{c_{t,j},p_{t,i}} + 1$
      25. **end for**
   26. **end for**
27. **end for**

Fig. 3. Pseudocode for m-CAC.
up to time slot $t$. Then, the estimated demand of file $f$ in set $p$ is given by the sample mean
\[ \hat{d}_{f,p}(t) := \frac{1}{|E_{f,p}(t)|} \sum_{d \in E_{f,p}(t)} d \]

In each time slot $t$, m-CAC first observes the number of currently connected users $U_t$, their contexts $x_t = (x_{t,i})_{i=1,...,U_t}$ and the service types $s_t = (s_{t,i})_{i=1,...,U_t}$. For each user context vector $x_{t,i}$ m-CAC determines the set $p_{t,i} \in \mathcal{P}_F$, to which the context vector belongs, i.e., such that $x_{t,i} \in p_{t,i}$ holds. The collection of all these sets is given by $p_t = (p_{t,1},...,p_{t,U_t})$. Then, the algorithm can be in one of the two phases mentioned above, in an exploration phase or in an exploitation phase. In order to determine the correct phase for the current time slot, the algorithm checks if there are files that have not been explored sufficiently often. For this purpose, the set of under-explored files $F_{pt}^{ue}(t)$ is calculated based on
\[ F_{pt}^{ue}(t) = \bigcup_{i=1}^{U_t} F_{p_{t,i}}^{ue}(t) \]
\[ := \bigcup_{i=1}^{U_t} \{ f \in F : N_{f,p_{t,i}}(t) \leq K(t) \}, \quad (4) \]
where $K(t)$ is a deterministic, monotonically increasing control function, which is an input to the algorithm. The control function has to be set adequately to balance the trade-off between exploration and exploitation. In Section \[ В \] we will select a control function that guarantees a good balance in terms of this trade-off.

If the set of under-explored files is non-empty, m-CAC enters the exploration phase. Let $u(t)$ be the size of the set of under-explored files. If the set of under-explored files contains at least $m$ elements, i.e., $u(t) \geq m$, the algorithm randomly selects $m$ files from $F_{pt}^{ue}(t)$ to cache. If the set of under-explored files contains less than $m$ elements, $u(t) < m$, it selects all $u(t)$ files from $F_{pt}^{ue}(t)$ to cache. Since the cache is not fully filled by $u(t) < m$ files, additionally $(m-u(t))$ other files can be cached. In order to exploit knowledge obtained so far, m-CAC selects $(m-u(t))$ files from $F \setminus F_{pt}^{ue}(t)$ based on a file ranking according to the estimated weighted demands, as defined by the files $\hat{f}_{1,p_{t},s_{t}}(t), ..., \hat{f}_{m-u(t),p_{t},s_{t}}(t) \in F \setminus F_{pt}^{ue}(t)$, which satisfy:
\[ \hat{f}_{j,p_{t},s_{t}}(t) \in \arg\max_{f \in F \setminus \bigcup_{i=1}^{m-u(t)} \{ \hat{f}_{k,p_{t},s_{t}}(t) \}} w_f \sum_{i=1}^{U_t} u_{s_{t,i}} \hat{d}_{f,p_{t,i}}(t) \]
\[ \hat{d}_{f,p}(t) := \frac{1}{|E_{f,p}(t)|} \sum_{d \in E_{f,p}(t)} d \]

\[ \hat{d}_{f,p}(t) \]

The set $E_{f,p}(t)$ does not have to be stored since the estimated demand $\hat{d}_{f,p}(t)$ can be updated based on $\hat{d}_{f,p}(t-1)$ and on the observed demands at time $t$.

\[ Note that in the pseudocode in Figure \[ 3 \] the argument $t$ is dropped from counters $N_{f,p}(t)$ and $\hat{d}_{f,p}(t)$ since previous values of these counters do not have to be stored.
for $j = 1, \ldots, m - u(t)$, where $\bigcup_{k=1}^{u(t)} \{\hat{f}_{k,pt,st}(t)\} = \emptyset$. If the set of files defined by (5) is not unique, ties are broken arbitrarily. Note that by this procedure, even in exploration phases, the algorithm additionally exploits, whenever the number of under-explored files is smaller than the cache size.

If the set of under-explored files $F_{p_{t}}^{ue}(t)$ is empty, m-CAC enters the exploitation phase. It selects $m$ files from $F$ based on a file ranking according to the estimated weighted demands, as defined by the files $\hat{f}_{1,pt,st}(t), \ldots, \hat{f}_{m,pt,st}(t) \in F$, which satisfy:

$$\hat{f}_{j,pt,st}(t) \in \arg\max_{f \in F \setminus (F_{p_{t}}^{ue}(t) \cup \bigcup_{k=1}^{j-1} \{f_{k,pt,st}(t)\})} w_{f} \sum_{i=1}^{u_{t}} v_{st,i} \hat{d}_{f,pt,i}(t)$$

for $j = 1, \ldots, m$. If the set of files defined by (6) is not unique, again ties are broken arbitrarily.

After selecting the files to be cached, the user demands for these files in this time slot are observed. Then, the estimated demands and the counters for cached files are updated.

V. ANALYSIS OF THE REGRET

In this section, we give an upper bound on the regret $R(T)$ of the learning algorithm m-CAC as given in (3).

The regret bound is based on the natural assumption that expected demands for files are similar for similar contexts. For example, at a similar time of day and for users with similar characteristics, the requests for files will also be similar. Such an assumption is crucial to learn from previous observations and can be captured by the following Hölder condition.

Assumption 1. There exists $L > 0$, $\alpha > 0$ such that for all $f \in F$ and for all $x, y \in \mathcal{X}$, it holds that

$$|\mu_{f}(x) - \mu_{f}(y)| \leq L ||x - y||^{\alpha},$$

where $|| \cdot ||$ denotes the Euclidian norm in $\mathbb{R}^{D}$. 

The Hölder condition allows us to derive a regret bound. Assumption [1] is needed for the analysis of the regret, but it should be noted that m-CAC can even be applied when this assumption does not hold true. However, in such a more general setting, a regret bound might not be guaranteed.
Next, we state our main theorem which shows that the regret of m-CAC is sublinear in the time horizon $T$, i.e., $R(T) = O(T^\gamma)$ with $\gamma < 1$. This bound on the regret guarantees that the algorithm has an asymptotically optimal performance, since then $\lim_{T \to \infty} \frac{R(T)}{T} = 0$ holds. This means, that asymptotically the average number of weighted cache hits is maximized with respect to the benchmark solution given by (2). In detail, the regret of m-CAC can be bounded as follows for any finite time horizon $T$.

**Theorem 1 (Bound for $R(T)$).** Let $K(t) = t^{\frac{2\alpha}{3\alpha + D}} \log(t)$ and $h_T = \lceil T^{\frac{1}{3\alpha + D}} \rceil$. If m-CAC is run with these parameters and Assumption 1 holds true, the leading order of the regret is $O\left(v_{\text{max}}w_{\text{max}}mU_{\text{max}}R_{\text{max}}|F|T^{\frac{2\alpha+D}{3\alpha+D}} \log(T)\right)$.

The proof can be found in our online appendix [35].

The regret bound given in Theorem 1 is sublinear in the time horizon $T$, proving that m-CAC asymptotically maximizes the average number of weighted cache hits. Additionally, Theorem 1 is applicable for any finite time horizon $T$, such that it provides a bound on the loss incurred by m-CAC for any finite number of cache replacement phases. Thus, Theorem 1 provides a characterization of the speed of convergence of the algorithm. Furthermore, from Theorem 1 it can be seen that the regret is a constant multiple of the regret bound for the special case without service differentiation, in which $v_{\text{max}} = 1$ and $w_{\text{max}} = 1$. Hence, also for the case of pure maximization of the number of cache hits, the order of the regret is $O(T^{\frac{2\alpha+D}{3\alpha+D}} \log(T))$.

**VI. MEMORY REQUIREMENTS**

The memory requirements of m-CAC are mainly determined by the counters that the algorithm keeps during its runtime. For each set $p$ in the partition $\mathcal{P}_T$ and each file $f$, the algorithm keeps the counters $N_{f,p}$ and $\hat{d}_{f,p}$. The number of files is $|F|$. If m-CAC is run with the parameters from Theorem 1 the number of sets in $\mathcal{P}_T$ is upper bounded by $\left(h_T\right)^D = \lceil T^{\frac{1}{3\alpha + D}} \rceil^D \leq 2^D T^{\frac{D}{3\alpha+D}}$. Hence, the required memory is upper bounded by $|F|2^D T^{\frac{D}{3\alpha+D}}$ and thus sublinear in the time horizon $T$. This means, that for $T \to \infty$, the algorithm would need infinite memory. However, for practical approaches, only the counters of such sets $p$ have to be kept to which at least one of the connected users’ context vectors has already belonged to. Hence, depending on the heterogeneity in the connecting users’ context vectors, the required number of counters that have to be kept can be much smaller than given by the upper bound above.
VII. Extensions

A. Exploiting the Multicast Gain

So far, we assumed that in the delivery phase of each time slot $t$, upon each content request, the requesting user is served with a file immediately. However, our algorithm can be extended to profit from the broadcast nature of the wireless medium by multicasting, which has been shown to be beneficial for caching in small cell networks [13]. For this purpose, each time slot $t$ is divided into a number of intervals. Within each of the intervals, user requests are monitored and accumulated. Then, after a short waiting time, the user requests that were monitored in the meanwhile are served. By such multicast transmissions, the traffic needed for data transmission is reduced. To exploit knowledge about content popularity learned so far, user requests for files with low estimated demand could, however, still be served immediately. In this way, unnecessary delays are prevented in cases in which another user request and thus a multicast transmission is not expected. Moreover, service differentiation could be taken into account. For example, high-priority users or high-priority content could be served immediately, such that their delay is not increased due to waiting times for multicast transmissions.

B. Rating-Based Context-Aware Proactive Caching

In the previous sections, we assumed that the caching entity performs cache content placement with respect to the demands of the contents, i.e., the number $d_f(x)$ of requests, to maximize the number of (weighted) cache hits. However, in case of a CP operating an infostation, his goal might be to cache not only the content that is requested often, but also receives high ratings from his end users. Consider the case that end users rate contents after their consumption in a range $[r_{\text{min}}, r_{\text{max}}] \subset \mathbb{R}_+$. For a context $x$, let $r_f(x)$ be the random variable describing the rating of a user with context $x$ if he requests file $f$ and makes a rating thereafter. Then, we define the random variable

\[
d_f(x) := \begin{cases} 
0, & \text{if file } f \text{ not requested} \\
r_f(x), & \text{if file } f \text{ requested}
\end{cases}
\]  

(7)
which instead of counting the number of requests, represents the rating of users with context $x$ if they request a content. By carefully designing the range of possible ratings, the CP can trade-off ratings and cache hits to a greater or lesser extend. With this definition, we can apply the algorithm from Section IV. In this case, m-CAC not only relies on observing the number of achieved cache hits after placing new content into the cache, but additionally, the users’ ratings have to be observed in order to learn content popularity in terms of ratings. If the users’ ratings are always available, the regret bound from Theorem 1 applies.

However, users might not always reveal a rating after consumption of a content. When the rating of a user is missing, we assume that m-CAC does not update the counters based on this user’s request. Hence, if no user within one time slot rates his requested content, m-CAC does not update its counters at all in this time slot. This results in a higher required number of exploration phases. Hence, the regret of the learning algorithm is influenced by the users’ willingness to reveal ratings for requested contents. Let $q \in (0, 1)$ be the probability that a user reveals his rating after requesting a content. Then, the regret of the learning algorithm is bounded as given below.

**Theorem 2** (Bound for $R(T)$ with missing ratings). Let $K(t) = t^{2\alpha + D} \log(t)$ and $h_T = \lceil T^{\frac{1}{3\alpha + D}} \rceil$. If m-CAC is run with these parameters, Assumption 7 holds true and a user reveals his rating with probability $q$, the leading order of the regret is $O\left(\frac{1}{q} v_{\text{max}} w_{\text{max}} m U_{\max} R_{\max} |F| T^{\frac{2\alpha + D}{3\alpha + D}} \log(T)\right)$.

The proof can be found in our online appendix [35].

Comparing Theorem 2 with the result from Theorem 1, the regret of m-CAC is scaled up by a factor $\frac{1}{q} > 1$ in case of rating-based context-aware proactive caching with missing ratings. This factor corresponds to the expected number of user requests until the caching entity receives one rating. However, the time order of the regret remains the same. Hence, the algorithm is robust under missing ratings in the sense that if some users refuse to rate requested contents, the algorithm is still asymptotically optimal.

**C. Asynchronous User Arrival**

So far, we assumed that in one time slot, exactly those users connected to the local caching entity when the user context is monitored, will request files within that time slot after the cache content is updated. However, if users connect to the local caching entity asynchronously, our
proposed algorithm should be adapted. If a user directly disconnects after the context monitoring and does not request any file, even though he was considered for cache content placement, it should be taken into account that his vanishing demand results from disconnection, i.e., he should be neglected when updating estimated demands. Hence, in m-CAC, the counters are not updated for disconnecting users. If a user connects to the local caching entity after cache content placement, his context was not considered in the caching decision. However, his requests can be used to learn faster by updating the estimated demands based on his requests. Hence, in m-CAC, the counters are updated based on these requests.

VIII. NUMERICAL RESULTS

In this section, we numerically evaluate the proposed learning algorithm m-CAC by comparing its solution for a real world data set to several reference algorithms.

A. Description of the Data Set

We use a data set from MovieLens to evaluate our presented algorithm. MovieLens is an online movie recommender operated by the research group GroupLens from the University of Minnesota. The MovieLens 1M Dataset [36] contains 1000209 ratings of 3952 movies. These ratings were made by 6040 users of MovieLens within the years 2000 to 2003. Each data set entry consists of an anonymous user ID, a movie ID, a rating (in whole numbers between 1 and 5) and a timestamp. Additionally, demographic information about the users is given: Their gender, age (in 7 categories), occupation (in 20 categories) as well as their Zip-code. For our numerical evaluations, we assume that the movie rating process in the data set corresponds to a content request process of users connected to a local caching entity (see [31] for a similar approach). Hence, a user rating a movie at a certain time in the data set for us corresponds to a request to either the local caching entity (in case the movie is cached at the local caching entity) or to the macro cellular network (in case the movie is not cached at the local caching entity). This approach is reasonable since users rate movies typically after they watched them.

In our simulations, we use only the data gathered within the first year of the data set, since around 94% of the user ratings were provided within this timeframe, while only very few ratings were made after this first year. Then, we divide a year’s time into 8760 time slots of one hour each ($T = 8760$), assuming that the caching entity updates its cache content at an hourly basis,
and assign the user requests to the time slots according to their timestamps. Figure 4 shows that the corresponding content request process is bursty and flattens out towards the end. As context dimensions, we select the dimensions gender and age.\(^5\)

**B. Reference Algorithms**

We compare the performance of m-CAC with five reference algorithms. The first algorithm is the *optimum oracle*. This algorithm works under the assumption of foresight, i.e., complete knowledge about user demands within the coming time slot. The optimum oracle selects the \(m\) files that will yield the highest number of requests within this time slot.\(^6\)

The second reference algorithm is called m-UCB, which consists of a variant of the UCB algorithm. UCB is a classical learning algorithm for multi-armed bandit problems, which has logarithmic regret order. However, it does not take into account context information, i.e., the logarithmic regret is with respect to the average expected demand over the whole context space. While in classical UCB, one action is taken in each period, we modify UCB to take \(m\) actions at a time, which corresponds to selecting \(m\) files.

\(^5\)We neglect the occupation as context dimension since by mapping them to a \([0,1]\) variable, we would have to classify which occupations are more similar to each other than others.

\(^6\)Note that the optimum oracle based on foresight yields even better results than the benchmark used to define the regret in \([3]\). In the definition of regret, the benchmark solution exploits only complete knowledge about expected content demands, instead of foresight of exact future demands. Since we use a data set with unknown expected content demands, we compare here to the optimum oracle.
The third reference algorithm is the m-\(\epsilon\)-Greedy. This is a variant of the simple \(\epsilon\)-Greedy \cite{34} algorithm, which does not consider context information. The m-\(\epsilon\)-Greedy caches a random set of \(m\) files with probability \(\epsilon \in (0, 1)\). With probability \((1 - \epsilon)\), the algorithm caches the \(m\) files with highest to \(m\)-th highest estimated demands. These estimated demands are calculated based on previous demands for cached files.

The forth reference algorithm is called m-Myopic. This is an algorithm taken from \cite{17}, which is investigated since it is comparable to the well known Least Recently Used algorithm (LRU) for caching. m-Myopic learns only from one period in the past. It starts with a random set of files and in each of the following periods discards all files that have not been requested in the previous period. Then it replaces the discarded files randomly by other files.

The fifth reference algorithm is called Random. It is a lower bound for the other algorithms, since it caches a random subset of files in each period.

C. Performance Measures

The following performance measures are used in our analysis. The evolution of per-time slot or aggregated number of cache hits allows comparing the absolute performance of the algorithms. A relative performance measure is given by the cache efficiency, which is defined as the ratio of cache hits compared to the overall demand, i.e.,

\[
\text{cache efficiency in } \% = \frac{\text{cache hits}}{\text{cache hits} + \text{cache misses}} \cdot 100.
\]

The cache efficiency describes the percentage of user requests which can be served by the files cached at the caching entity. Note that in general, even the optimum oracle cannot serve all user requests because of the limited cache size.

D. Results

In our simulations, we set \(\epsilon = 0.09\) in m-\(\epsilon\)-Greedy, which is the value at which heuristically the algorithm on average performed best. In m-CAC, we set the control function to \(K(t) = c \cdot t^{2\alpha + D} \log(t)\) with \(c = 1/(|F|D)\). Compared to the control function in Theorem \cite{1}, the additional factor reduces the number of exploration phases which allows for better performance.

First, we consider the case of pure cache hit maximization, i.e., without incorporating service differentiation. The long-term behavior of our proposed algorithm is investigated with the
following scenario. We assume that the caching entity can store \( m = 200 \) movies out of the \(|F| = 3952\) available movies. Hence, the cache size corresponds to about 5\% of the overall number of files \([17]\). We run all algorithms on the data set and study their results as a function of time, i.e. over the time slots \( t = 1, \ldots, T \). Figures 5 and 6 show the per-time slot and the aggregated number of cache hits up to period \( t \) as a function of time, respectively. As reflected in Figures 5 and 6 due to the bursty content request process (compare Figure 4), also the number of cache hits over time is bursty for all considered algorithms. As expected, the optimum oracle gives an upper bound to the other algorithms. Among the non-optimal algorithms, our proposed m-CAC, as well as m-\( \epsilon \)-Greedy and m-UCB clearly outperform m-Myopic and Random. This is due to the fact that these three algorithms learn from the history of observed demands, while m-Myopic only learns from one time slot in the past and Random does not learn at all. It can be observed that m-\( \epsilon \)-Greedy shows a better performance than m-UCB, even though it uses a simpler learning strategy. Overall, m-CAC outperforms all other non-optimal algorithms by additionally learning from context information. At the end of the time horizon, the aggregated number of cache hits of m-CAC is 14.6\%, 37.7\%, 298.5\% and 450.6\% higher than the ones of m-\( \epsilon \)-Greedy, m-UCB, m-Myopic and Random, respectively.

Next, we investigate the impact of the cache size \( m \) by varying it between 50 and 400 files. This range of cache sizes corresponds to about 1.3\% to 10.1\% of the overall number of files, which is a realistic assumption. All remaining parameters are kept as in the first scenario. Figure 7 shows the overall cache efficiency at the end of the time horizon \( T \) as a function of cache size, i.e., the total number of cache hits at \( T \) is normalized by the total number of requests. For
Fig. 6. Aggregated number of cache hits for $m = 200$ as a function of time

![Graph showing aggregated number of cache hits](image)

Fig. 7. Overall cache efficiency at $T$ as a function of cache size $m$

![Graph showing overall cache efficiency](image)

all algorithms, the overall cache efficiency is increasing for increasing cache size. Moreover, the results indicate that again $m$-CAC and $m$-$\epsilon$-Greedy slightly outperform $m$-UCB and clearly outperform $m$-Myopic and Random. Averaged over the range of cache sizes, the cache efficiency of $m$-CAC is 15.0%, 43.6%, 299.0% and 451.2% higher than the ones of $m$-$\epsilon$-Greedy, $m$-UCB, $m$-Myopic and Random, respectively.

Now we consider a case of service differentiation, in which two different service types 1 and 2 exist with weights $v_1 = 5$ and $v_2 = 1$. Hence, service type 1 should be prioritized due to the higher value it represents. We randomly assign 10% of the users to service type 1 and classify all remaining users as service type 2. Then we adjust all algorithms to take into account service differentiation by incorporating the weights according to the service types. Figure 8 shows the aggregated number of weighted cache hits up to period $t$ as a function of time. At the end of the
time horizon, the aggregated number of weighted cache hits of m-CAC is 15.6%, 21.9%, 291.4% and 436.2% higher than the ones of m-\(\epsilon\)-Greedy, m-UCB, m-Myopic and Random, respectively. A comparison with Figure 6 shows that the behaviour of the algorithms is similar to the case without service differentiation.

Finally, we investigate the extension of the algorithm to rating-based context-aware proactive caching. We use the same data set as above, but while we neglected the user ratings from the original data set so far by only counting every rating in the original data set as a content request, here, we take the users’ ratings into account as defined in (7). We do not consider service differentiation, but it could be applied here as well. Figure 9 shows the aggregated ratings achieved by all algorithms for a cache size of \(m = 200\) movies as a function of time. The long-term behavior shows that m-CAC outperforms m-UCB and m-\(\epsilon\)-Greedy, while m-Myopic and Random perform again much worse.

IX. CONCLUSION

In this paper, we presented a context-aware proactive caching algorithm based on contextual multi-armed bandits for cache content placement at a local caching entity in a wireless network. To cope with unknown and fluctuating content popularity among the permanently changing local user population, the algorithm regularly observes context information of connected users, updates the cache content and observes the demands for cached content. Thereby, the algorithm learns context-specific content popularity online over time, which allows for a proactive adaptation of cache content according to fluctuating local content popularity. We derived a sub-linear regret
Fig. 9. Aggregated ratings for \( m = 200 \) as a function of time

bound, which characterizes how fast the algorithm converges to optimal cache content placement and proves that our proposed algorithm asymptotically maximizes the average number of cache hits. Moreover, the algorithm supports customer prioritization. Additionally, the algorithm can be combined with multicast transmissions and it can be extended to allow rating-based caching decisions. Numerical studies showed that by exploiting contextual information, our algorithm outperforms state-of-the-art algorithms in a real world data set.

ACKNOWLEDGMENT

The work by Sabrina Müller and Anja Klein has been funded by the German Research Foundation (DFG) as part of project B03 within the Collaborative Research Center (CRC) 1053 – MAKI. The work by Onur Atan and Mihaela van der Schaar is supported by a DDDAS Airforce grant.

REFERENCES

[1] [Online]. Available: \[\text{http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white\_paper_c11-481360.pdf}\]

[2] [Online]. Available: \[\text{http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white\_paper_c11-520862.pdf}\]

[3] A. Vakali and G. Pallis, “Content delivery networks: status and trends,” *IEEE Internet Computing*, vol. 7, no. 6, pp. 68–74, Nov. 2003.

[4] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, “Globally distributed content delivery,” *IEEE Internet Computing*, vol. 6, no. 5, pp. 50–58, Sep. 2002.
[5] S. Borst, V. Gupta, and A. Walid, “Distributed caching algorithms for content distribution networks,” in Proc. IEEE INFOCOM, 2010, pp. 1–9.

[6] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. Leung, “Cache in the air: exploiting content caching and delivery techniques for 5G systems,” IEEE Communications Magazine, vol. 52, no. 2, pp. 131–139, Feb. 2014.

[7] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, “Web caching and zipf-like distributions: evidence and implications,” in Proc. IEEE INFOCOM, vol. 1, 1999, pp. 126–134.

[8] J. Erman, A. Gerber, M. Hajighayi, D. Pei, S. Sen, and O. Spatscheck, “To cache or not to cache: The 3G case,” IEEE Internet Computing, vol. 15, no. 2, pp. 27–34, Mar. 2011.

[9] M. Maddah-Ali and U. Niesen, “Fundamental limits of caching,” IEEE Transactions on Information Theory, vol. 60, no. 5, pp. 2856–2867, May 2014.

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

[11] K. Shammugam, N. Golrezaei, A. Dimakis, A. Molisch, and G. Caire, “Femtocaching: Wireless content delivery through distributed caching helpers,” IEEE Transactions on Information Theory, vol. 59, no. 12, pp. 8402–8413, Dec. 2013.

[12] K. Poularakis and L. Tassiulas, “Exploiting user mobility for wireless content delivery,” in Proc. IEEE International Symposium on Information Theory (ISIT), 2013, pp. 1017–1021.

[13] K. Poularakis, G. Iosifidis, V. Sourlas, and L. Tassiulas, “Exploiting caching and multicast for 5G wireless networks,” IEEE Transactions on Wireless Communications, vol. 15, no. 4, pp. 2995–3007, 2016.

[14] E. Bastug, M. Bennis, and M. Debbah, “Cache-enabled small cell networks: Modeling and tradeoffs,” in Proc. International Symposium on Wireless Communications Systems (ISWCS), 2014, pp. 649–653.

[15] ———, “Living on the edge: The role of proactive caching in 5G wireless networks,” IEEE Communications Magazine, vol. 52, no. 8, pp. 82–89, Aug. 2014.

[16] E. Bastug, M. Bennis, E. Zeydan, M. Abdel Kader, A. Karatepe, A. Salih Er, and M. Debbah, “Big data meets telcos: A proactive caching perspective,” Journal of Communications and Networks, Special Issue on Big Data Networking-Challenges and Applications, vol. 17, no. 6, pp. 549–558, Dec. 2015.

[17] P. Blasco and D. Gündüz, “Learning-based optimization of cache content in a small cell base station,” in Proc. IEEE International Conference on Communications (ICC), 2014, pp. 1897–1903.

[18] ———, “Multi-armed bandit optimization of cache content in wireless infostation networks,” in Proc. IEEE International Symposium on Information Theory (ISIT), 2014, pp. 51–55.

[19] ———, “Content-level selective offloading in heterogeneous networks: Multi-armed bandit optimization and regret bounds,” arXiv preprint, arXiv: 1407.6154, 2014.

[20] A. Sengupta, S. Amuru, R. Tandon, R. Buehrer, and T. Clancy, “Learning distributed caching strategies in small cell networks,” in Proc. IEEE International Symposium on Wireless Communications Systems (ISWCS), 2014, pp. 917–921.

[21] M. ElBamby, M. Bennis, W. Saad, and M. Latva-aho, “Content-aware user clustering and caching in wireless small cell networks,” in Proc. IEEE International Symposium on Wireless Communications Systems (ISWCS), 2014, pp. 945–949.

[22] S. Müller, O. Atan, M. van der Schaar, and A. Klein, “Smart caching in wireless small cell networks via contextual multi-armed bandits,” in Proc. IEEE International Conference on Communications (ICC), 2016, pp. 1–7.

[23] D. Goodman, J. Borras, N. B. Mandayam, and R. Yates, “Infostations: a new system model for data and messaging services,” in Proc. IEEE Vehicular Technology Conference, vol. 2, May 1997, pp. 969–973.
[24] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, “Youtube traffic characterization: A view from the edge,” in Proc. ACM SIGCOMM Conference on Internet Measurement, ser. IMC ’07, 2007, pp. 15–28.

[25] M. Zink, K. Suh, Y. Gu, and J. Kurose, “Characteristics of youtube network traffic at a campus network - measurements, models, and implications,” Comput. Netw., vol. 53, no. 4, pp. 501–514, Mar. 2009.

[26] A. Brodersen, S. Scellato, and M. Wattenhofer, “YouTube around the world: Geographic popularity of videos,” in Proc. ACM International Conference on World Wide Web, 2012, pp. 241–250.

[27] C. Zhou, Y. Guo, Y. Chen, X. Nie, and W. Zhu, “Characterizing user watching behavior and video quality in mobile devices,” in Proc. IEEE International Conference on Computer Communication and Networks (ICCCN), Aug 2014, pp. 1–6.

[28] B.-J. Ko, K.-W. Lee, K. Amiri, and S. Calo, “Scalable service differentiation in a shared storage cache,” in Proc. IEEE International Conference on Distributed Computing Systems, May 2003, pp. 184–193.

[29] Y. Lu, T. F. Abdelzaher, and A. Saxena, “Design, implementation, and evaluation of differentiated caching services,” IEEE Trans. Parallel Distrib. Syst., vol. 15, no. 5, pp. 440–452, May 2004.

[30] P. Cao and S. Irani, “Cost-aware www proxy caching algorithms,” in Proc. USENIX Symposium on Internet Technologies and Systems, 1997, pp. 193–206.

[31] S. Li, J. Xu, M. van der Schaar, and W. Li, “Popularity-driven content caching,” in Proc. IEEE INFOCOM, 2016.

[32] T. Lu, D. Pal, and M. Pal, “Contextual multi-armed bandits,” in Proc. International Conference on Artificial Intelligence and Statistics (AISTATS), 2010, pp. 485–492.

[33] C. Tekin and M. van der Schaar, “Distributed online learning via cooperative contextual bandits,” IEEE Transactions on Signal Processing, vol. 63, no. 14, pp. 3700–3714, Mar. 2015.

[34] P. Auer, N. Cesa-Bianchi, and P. Fischer, “Finite-time analysis of the multiarmed bandit problem,” Mach. Learn., vol. 47, no. 2-3, pp. 235–256, May 2002.

[35] [Online]. Available: http://kang.nt.e-technik.tu-darmstadt.de/nt/fileadmin/kt/Publikationen_PDFs/2016/TWC/TWC2016_Mueller_App.pdf

[36] [Online]. Available: http://grouplens.org/datasets/movielens/1m/

[37] W. Hoeffding, “Probability inequalities for sums of bounded random variables,” Journal of the American Statistical Association, vol. 58, no. 301, pp. 13–30, 1963.

[38] E. Chlebus, “An approximate formula for a partial sum of the divergent p-series,” Applied Mathematics Letters, vol. 22, no. 5, pp. 732 – 737, May 2009.