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

Sabrina Müller, *Student Member, IEEE*, Onur Atan, Mihaela van der Schaar, *Fellow, IEEE*, and Anja Klein, *Member, IEEE*

AQ:1 ***Abstract*— Content caching in small base stations or wireless infostations is considered to be a suitable approach to improving the efficiency in wireless content delivery. Placing the optimal content into local caches is crucial due to storage limitations, but it requires knowledge about the content popularity distribution, which is often not available in advance. Moreover, local content popularity is subject to fluctuations, since mobile users with different interests connect to the caching entity over time. Which content a user prefers may depend on the user's context. In this paper, we propose a novel algorithm for context-aware proactive caching. The algorithm learns context-specific content popularity online by regularly observing context information of connected users, updating the cache content and observing cache hits subsequently. We derive a sublinear regret bound, which characterizes the learning speed and proves that our algorithm converges to the optimal cache content placement strategy in terms of maximizing the number of cache hits. Furthermore, our algorithm supports service differentiation by allowing operators of caching entities to prioritize customer groups. Our numerical results confirm that 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*— Wireless networks, caching at the edge, cache content placement, online learning.**

I. Introduction

AQ:2 Wireless networks have been experiencing a steep increase in data traffic in recent years [2]. With 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 expected to increase and to account for the majority of mobile data traffic within the next few years [2]. However, despite recent advances in cellular mobile radio networks, these networks cannot keep up with the massive growth of mobile data traffic [3]. As already investigated for wired networks [4], *content caching* is envisioned to improve the efficiency 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 [5].

Within wireless networks, *caching at the edge* has been extensively studied [1], [6]–[19]. At the radio access network level, current approaches comprise two types of *wireless local caching entities*. The first type are *macro base stations* (MBSs) and *small base stations* (SBSs) that are implemented in wireless small cell networks, dispose of limited storage capacities and are typically owned by the *mobile network operator* (MNO). The second type are *wireless infostations* with limited storage capacities that provide high bandwidth local data communication [16], [17], [20], [21]. Wireless infostations could be installed in public or commercial areas and could use Wi-Fi for local data communication. They could be owned by *content providers* (CPs) aiming at increasing their users' quality of experience. Alternatively, third parties (e.g., the owner of a commercial area) could offer caching at infostations as a service to CPs or to the users [17]. Both types of caching entities store a fraction of available popular content in a *placement phase* and serve local users' requests via localized communication in a *delivery phase*.

Due to the vast amount of content available in multimedia platforms, not all available content can be stored in local caches. Hence, 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, however, when caching content at a particular point in time, it is unclear which content will be requested in future. Not even an estimate of the content popularity distribution might be at hand. It therefore must be computed by the caching entity itself [1], [13]–[19], which is not only legitimate from an overhead point of view, since else a periodic coordination with the global multimedia platform would be required. More importantly, local content popularity in a caching entity might not even replicate global content popularity as monitored by the global multimedia platform [22]–[24]. Hence, caching entities should learn local content popularity for a *proactive* cache content placement. Secondly, different content can be favored by different users. Consequently, local content popularity may change according to the different preferences of fluctuating mobile users in the vicinity of a

Manuscript received May 4, 2016; revised September 30, 2016; accepted November 20, 2016. The work by S. Müller and A. Klein was supported by the German Research Foundation as part of Projects B3 and C1 within the Collaborative Research Center 1053–MAKI. The work by O. Atan and M. van der Schaar was supported by the NSF under Grant CCF1524417 and Grant ECCS1407712. This paper was presented at the IEEE International Conference on Communications, May 2016 [1]. The associate editor coordinating the review of this paper and approving it for publication was B. Hamdaoui.

S. Müller and A. Klein are with the Communications Engineering Laboratory, Technische Universität Darmstadt, 64283 Darmstadt, Germany (e-mail: s.mueller@nt.tu-darmstadt.de; a.klein@nt.tu-darmstadt.de).

O. Atan and M. van der Schaar are with the Department of Electrical Engineering, University of California at Los Angeles, Los Angeles, CA 90095 USA (e-mail: oatan@ucla.edu; mihaela@ee.ucla.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TWC.2016.2636139

caching entity. Therefore, proactive cache content placement should take into account the *diversity in content popularity* across the local user population. Thirdly, the users' preferences in terms of consumed content may differ based on their contexts, such as their location [24], personal characteristics (e.g., age [25], gender [26], personality [27], mood [28]), or their devices' characteristics [29]. Hence, cache content placement should be *context-aware* by taking into account that content popularity depends on a user's context. Thereby, a caching entity can learn the preferences of users with different contexts. Fourthly, while its typical goal is to maximize the number of cache hits, cache content placement should also take into account the cache 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 [30], [31]. For example, if users with different preferences are connected to a caching entity, the operator could prioritize certain users by caching content favored by these users. Moreover, certain CPs' content could be prioritized in caching decisions.

In this paper, we propose a novel context-aware proactive caching algorithm, which for the first time *jointly* considers the above four aspects. Firstly, instead of assuming a priori knowledge about content popularity, which might be externally given or estimated in a separate training phase, our algorithm learns the content popularity online by observing the users' requests for cache content. Secondly, by explicitly allowing different content to be favored by different users, our algorithm is especially suitable for mobile scenarios, in which users with different preferences arrive at the wireless caching entity over time. Thirdly, we explicitly model that the content popularity depends on a user's context, such as his/her personal characteristics, equipment, or external factors, and propose an algorithm for content caching that learns this context-specific content popularity. Using our algorithm, a caching entity can proactively cache content for the currently connected users based on what it has previously learned, instead of simply caching the files that are popular "on average", across the entire population of users. The learned cache content placement strategy is proven to converge to the optimal cache content placement strategy which maximizes the expected number of cache hits. Fourthly, the algorithm allows for service differentiation by customer prioritization. The contributions of this paper are as follows:

- We present a context-aware proactive caching algorithm based on contextual multi-armed bandit optimization. Our algorithm incorporates diversity in content popularity across the user population and takes into account the dependence of users' preferences on their context. Additionally, it supports service differentiation by prioritization.
- We analytically bound the loss of the algorithm compared to an oracle, which assumes a priori knowledge about content popularity. We derive a sublinear regret bound, which characterizes the learning speed and proves that our algorithm converges to the optimal cache content placement strategy which maximizes the expected number of cache hits.
- We present additional extensions of our approach, such as its combination with multicast transmissions and the incorporation of caching decisions based on user ratings.
- We numerically evaluate our caching algorithm based on a real world data set. A comparison shows that by exploiting context information in order to proactively cache content for currently connected users, 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. 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 propose some extensions of the algorithm. Numerical results are presented in Section VIII. Section IX concludes the paper.

II. Related Work

Practical caching systems often 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) (see [32]). While these simple algorithms do not consider future content popularity, recent work has been devoted to developing sophisticated cache replacement algorithms by learning content popularity trends [33], [34].

In this paper, however, we focus 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. Information-theoretic gains achieved by combining caching at user devices with a coded multicast transmission in the delivery phase are calculated in [7]. The proposed coded caching approach is optimal up to a constant factor. Content caching at user devices and collaborative device-to-device communication are combined in [8] to increase the efficiency of content delivery. In [9], an approximation algorithm for uncoded caching among SBSs equipped with caches is given, which minimizes the average delay experienced by users that can be connected to several SBSs simultaneously. Building upon the same caching architecture, in [10], an approximation algorithm for distributed coded caching is presented for minimizing the probability that moving users have to request parts of content from the MBS instead of the SBSs. In [11], a multicast-aware caching scheme is proposed for minimizing the energy consumption in a small cell network, in which the MBS and the SBSs can perform multicast transmissions. The outage probability and average content delivery rate in a network of SBSs equipped with caches are analytically calculated in [12].

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

TABLE I

COMPARISON WITH RELATED WORK ON LEARNING-BASED CACHING WITH PLACEMENT AND DELIVERY PHASE

	[13], [14]	[15]–[17]	[18]	[19]	This work
Model-Free	Yes	Yes	No	Yes	Yes
Online/Offline-Learning	Offline	Online	Online	Online	Online
Free of Training Phase	No	Yes	Yes	No	Yes
Performance Guarantees	No	Yes	No	No	Yes
Diversity in Content Popularity	No	No	No	Yes	Yes
User Context-Aware	No	No	No	No	Yes
Service Differentiation	No	No	No	No	Yes

AQ:3

proactive caching paradigm, [13] and [14] propose a caching 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 ratio based on their required delivery rates. 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 the content popularity online, thus also adapting to varying content popularities. In [15], using a multi-armed bandit algorithm, an SBS learns a fixed content popularity distribution online by refreshing its cache content and observing instantaneous demands for cached files. In this way, cache content placement is optimized over time to maximize the traffic served by the SBS. The authors extend their framework for a wireless infostation in [16] and [17] by additionally taking into account the costs for adding files to the cache. Moreover, they provide theoretical sublinear regret bounds for their algorithms. A different extension of the multi-armed bandit framework is given in [18], which exploits the topology of users' connections to the SBSs by incorporating coded caching. The approach in [18] assumes a specific type of content popularity distribution. Since in practice the type of 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 type of content popularity distribution. Moreover, in [15]–[18], the optimal cache content placement strategy is learned over time based only on observations of instantaneous demands. In contrast, our proposed algorithm additionally takes diversity of content popularity across the user population into account and exploits users' context information. Diversity in content popularity across the user population is for example taken into account in [19], but again without considering the users' contexts. Users are clustered into groups of similar interests by a spectral clustering algorithm based on their requests in a training phase. Each user group is then assigned to an SBS which learns the content popularity of its fixed user group over time. Hence, in [19], each SBS learns a fixed content popularity distribution under the assumption of a stable user population, whereas our approach allows reacting to arbitrary arrivals of users preferring different content.

In summary, compared to related work on cache content placement (see Table I), our proposed algorithm for the first time *jointly* learns the content popularity online, allows for diversity in content popularity across the user population, takes into account the dependence of users' preferences on their context and supports service differentiation. Compared to our previous work [1], we now take into account context information at a single user level, instead of averaging context information over the currently connected users. This enables more fine-grained learning. Additionally, we incorporate service differentiation and present extensions, e.g., to multicast transmission and caching decisions based on user ratings.

We model the caching problem as a multi-armed bandit problem. Multi-armed bandit problems [35] have been applied to various scenarios in wireless communications before [36], such as cognitive jamming [37] or mobility management [38]. Our algorithm is based on *contextual multi-armed bandit* algorithms [39]–[42]. The closest related work is [42], in which several learners observe a single context arrival in each time slot and select a subset of actions to maximize the sum of expected rewards. While [42] considers multiple learners, our system has only one learner – the caching entity selecting a subset of files to cache in each time slot. Compared to [42], we extended the algorithm in the following directions: We allow multiple context arrivals in each time slot, and select a subset of actions which maximize the sum of expected rewards given the context arrivals. In the caching scenario, this translates to observing the contexts of all currently connected users and caching a subset of files which maximize the sum of expected numbers of cache hits given the users' contexts. In addition, we enable each arriving context to be annotated with a weight, so that if different contexts arrive within the same time slot, differentiated services can be provided per context, by selecting a subset of actions which maximize the sum of expected weighted rewards. In the caching scenario, this enables the caching entity to prioritize certain users when selecting the cache content, by placing more weight on files that are favored by prioritized users. Moreover, we enable each action to be annotated with a weight, such that actions can be prioritized for selection. In the caching scenario, this enables the caching entity to prioritize certain files when selecting the cache content.

III. SYSTEM MODEL

A. Wireless Local Caching Entity

We consider a wireless local caching entity that can either be an SBS equipped with a cache 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 file library F containing

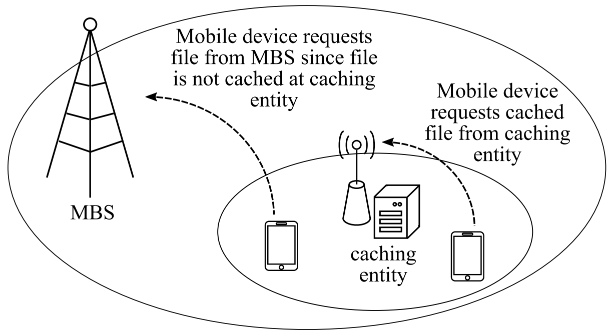


Fig. 1. System model.

$|F| \in \mathbb{N}$ files, where we assume for simplicity that all files are of the same size. Users located in the coverage area can connect to the caching entity. The set of currently connected users may change dynamically over time due to the users' mobility. At most $U_{\max} \in \mathbb{N}$ users can be simultaneously connected to the caching entity. To inform connected users about available files, the caching entity periodically broadcasts the information about the current cache content [15]–[17]. If a user is interested in a file that the caching entity stored in its cache, the user's device 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's device does not request the file from the caching entity. Instead, it 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. Hence, the caching entity can only observe requests for cached files, i.e., *cache hits*, but it cannot observe requests for non-cached files, i.e., *cache misses*. Note that this restriction is specific to wireless caching and is usually not used in wired caching scenarios. In this way, the caching entity is not congested by cache misses [15]–[17], but learning content popularity is more difficult. Fig. 1 shows an illustration of the considered system model.

In order to reduce the load on the macro cellular network and the backhaul network, a caching entity might aim at optimizing the cache content such that the traffic it can serve is maximized, which corresponds to maximizing the number of cache hits. For this purpose, the caching entity should learn which files are most popular over time.

B. Service Differentiation

Maximizing the number of cache hits might be an adequate goal of cache content placement in case of an MNO operating an SBS, one reason being net neutrality restrictions. However, the operator of an infostation, e.g., a CP or third party operator, may want to provide differentiated services to its customers (those can be both users and CPs). For example, if users with different preferences are connected to an infostation, the operator can prioritize certain users by caching content favored by these users. In this case, a cache hit by a prioritized user is associated with a higher value than a cache hit by a regular user. For this purpose, we consider a finite set S of service types. For service type $s \in S$, let $v_s \geq 1$ denote a fixed and known weight associated with receiving one cache hit by a user of service type s. Let $v_{\max} := \max_{s \in S} v_s$. The weights might be selected based on a pricing policy, e.g., by paying a monthly fee, a user can buy a higher weight. Alternatively, the weights might be selected based on a subscription policy, e.g., subscribers might obtain priority compared to one-time users. Yet another prioritization might be based on the importance of users in terms of advertisement or their influence on the operator's reputation. Finally, prioritization could be based on usage patterns, e.g., users might indicate their degree of openness in exploring other than their most preferred content. Taking into account the service weights, the caching entity's goal becomes to maximize the number of *weighted* cache hits. Clearly, the above service differentiation only takes effect if users with different preferences are present, i.e., if content popularity is heterogeneous across the user population.

Another service differentiation can be applied in case of a third party operator whose customers are different CPs. The operator may want to prioritize certain CPs by caching their content. In this case, each content is associated with a weight. Here, we consider a fixed and known prioritization weight $w_f \geq 1$ for each file $f \in F$ and let $w_{\max} := \max_{f \in F} w_f$. The prioritization weights can either be chosen individually for each file or per CP.

The case without service differentiation, where the goal is to maximize the number of (non-weighted) cache hits, is a special case, 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 refer to the more general case in the subsequent sections, this special case is naturally contained in our analysis.

TABLE II
EXAMPLES OF CONTEXT DIMENSIONS

Class	Context Dimension
personal characteristics	demographic factors personality mood
user equipment	type of device device capabilities battery status
external factors	location time of day, day of the week events

C. Context-Specific Content Popularity

Content popularity may vary across a user population since different users may prefer different content. A user's preferences might be linked to various factors. We refer to such factors as *context dimensions* and give some examples in Table II. Relevant *personal characteristics* may, for example, be demographic factors (e.g., age, gender), personality, or mood. In addition, a user's preferences may be influenced by

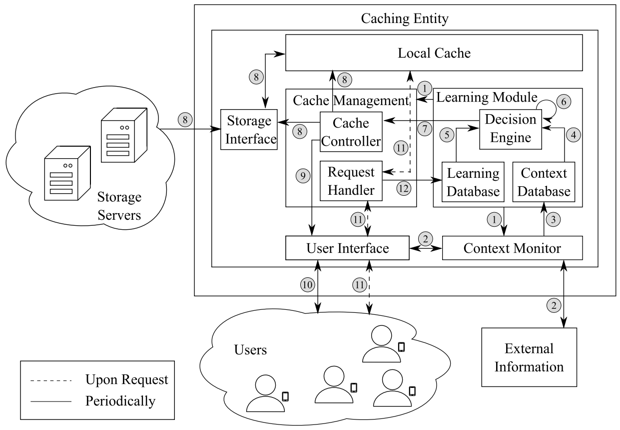


Fig. 2. Context-aware proactive caching architecture.

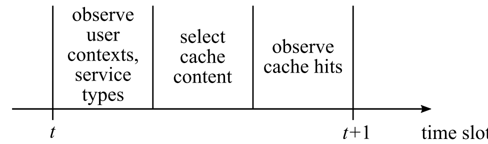


Fig. 3. Sequence of operations of context-aware proactive caching in time slot t.

user equipment, such as the type of device used to access and consume the content (e.g., smart phone, tablet), as well as its capabilities, or its battery status. Besides, *external factors* may have an impact on a user's preferences, such as the user's location, the time of day, the day of the week, and the taking place of events (e.g., soccer match, concert). Clearly, this categorization is not exhaustive and the impact of each single context dimension on content popularity is unknown a priori. Moreover, a caching entity may only have access to some of the context dimensions, e.g., due to privacy reasons. However, our model *does not* rely on *specific* context dimensions; it can use the information that *is* collected from the user. If the caching entity does have access to some relevant context dimensions, these can be exploited to learn context-specific content popularity.

D. Context-Aware Proactive Caching Architecture

Next, we describe the architecture for context-aware proactive caching, which is designed similarly to an architecture presented in [33]. An illustration of the context-aware proactive caching architecture is given in Fig. 2. Its main building blocks are the *Local Cache*, a *Cache Management* entity, a *Learning Module*, a *Storage Interface*, a *User Interface*, and a *Context Monitor*. 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 Fig. 2 and is described below.

- Initialization
  (1) The Learning Module is provided with the goal of caching (i.e., maximize number of cache hits or achieve operator-specific goal). It fixes the appropriate periodicity of context monitoring and cache refreshment. Then, it 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 information about currently connected users available at the User Interface and optionally by collecting additional information from external sources (e.g., 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 results from previous time slots as stored in the Learning Database, (6) the Decision Engine decides which files to cache in the coming time slot. (7) The Decision Engine instructs the Cache Controller to refresh the cache content accordingly. (8) The Cache Controller compares the current and the required cache content 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 new cache content to currently connected users.
- User Requests
  (11) When a user requests a cached file, the User Interface forwards the request to the Request Handler. The Request Handler stores the request information, retrieves the 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 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. The caching system operates in discrete time slots $t = 1, 2, ..., T$, where T denotes the finite time horizon. As illustrated in Fig. 3, each time slot t consists of the following sequence of operations: (i) The context of currently connected users and their service types are monitored. Let U_t be the number of currently connected users. We assume that $1 \leq U_t \leq U_{\max}$ and we specifically allow the set of currently connected users to change in between the time slots of the algorithm, so that user mobility is taken into account. 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 $\mathbf{x_t} = (x_{t,i})_{i=1,...,U_t}$ 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 $\mathbf{s_t} = (s_{t,i})_{i=1,...,U_t}$ be the collection of service types of all users in time slot t. (ii) The cache content is refreshed based on the contexts $\mathbf{x_t}$, the service types $\mathbf{s_t}$ and their service weights, the file prioritization weights w_f, $f \in F$, and knowledge from previous time slots. Then, connected users are informed about the current cache content, which is denoted by $C_t = \{c_{t,1}, ..., c_{t,m}\}$. (iii) Until the end of the time slot, users can request currently cached files. Their requests are served. The demand $d_{c_{t,j}}(x_{t,i}, t)$ of each user $i = 1, ..., U_t$ for each cached file $c_{t,j} \in C_t$, $j = 1, ..., m$, in this time slot is observed, i.e., the number of cache hits for each cached file is monitored.

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 $\mu_f(x) := E(d_f(x))$. The random demand is assumed to take values in $[0, R_{\max}]$, where $R_{\max} \in \mathbb{N}$ is the maximum possible number of requests a user can submit within one time slot. This explicitly incorporates that a user may request the same file repeatedly within one time slot. In time slot t, the random variables $(d_f(x_{t,i}))_{i=1,..,U_t, f\in F}$, are assumed to be independent, i.e., the requests of currently connected users and between different files are independent of each other. Moreover, each random variable $d_f(x_{t,i})$ is assumed to be independent of past caching decisions and previous demands.

The goal of the caching entity is to select the cache content in order to maximize the expected cumulative number of (weighted) cache hits up to the finite time horizon T. We introduce a binary variable $y_{t,f}$, which describes if file f is cached in time slot t, where $y_{t,f} = 1$, if $f \in C_t$, and 0 otherwise. Then, the problem of cache content placement can be formally written as

$$\begin{aligned} \max \quad & \sum_{t=1}^{T} \sum_{f \in F} y_{t,f} w_f \sum_{i=1}^{U_t} v_{s_{t,i}} \mu_f(x_{t,i}) \\ \text{s.t.} \quad & \sum_{f \in F} y_{t,f} \leq m, \quad t = 1, ..., T, \\ & y_{t,f} \in \{0, 1\}, \quad f \in F, \ t = 1, ..., T. \end{aligned} \tag{1}$$

Let us now first assume that the caching entity *had* a priori knowledge about context-specific content popularity *like an omniscient oracle*, i.e., suppose that for each context vector $x \in \mathcal{X}$ and for each file $f \in F$, the caching entity would know the expected demand $\mu_f(x) = E(d_f(x))$. In this case, problem (1) corresponds to an integer linear programming problem. The problem can be decoupled into T independent sub-problems, one for each time slot t. Each sub-problem is a special case of the knapsack problem [43] with a knapsack of capacity m and with items of non-negative profit and unit weights. Hence, its optimal solution can be easily computed in a running time of $O(|F|\log(|F|))$ as follows. In time slot t, given the contexts $\mathbf{x_t}$ and the service types $\mathbf{s_t}$, the optimal solution is given by ranking the files according to their (weighted) expected demands and by selecting the m highest ranked files. We denote these *top-m files for pair ($\mathbf{x_t}$, $\mathbf{s_t}$)* by $f_1^*(\mathbf{x_t}, \mathbf{s_t}), f_2^*(\mathbf{x_t}, \mathbf{s_t}), ..., f_m^*(\mathbf{x_t}, \mathbf{s_t}) \in F$. Formally, for $j = 1, ..., m$, they satisfy [1]

$$f_j^*(\mathbf{x_t}, \mathbf{s_t}) \in \underset{f \in F \setminus (\bigcup_{k=1}^{j-1} \{f_k^*(\mathbf{x_t}, \mathbf{s_t})\})}{\operatorname{argmax}} \quad w_f \sum_{i=1}^{U_t} v_{s_{t,i}} \mu_f(x_{t,i}), \tag{2}$$

where $\bigcup_{k=1}^{0} \{f_k^*(\mathbf{x_t}, \mathbf{s_t})\} := \emptyset$. We denote by $C_t^*(\mathbf{x_t}, \mathbf{s_t}) := \bigcup_{k=1}^{m} \{f_k^*(\mathbf{x_t}, \mathbf{s_t})\}$ an optimal choice of files to cache in time slot t. Consequently, the collection

$$(C_t^*(\mathbf{x_t}, \mathbf{s_t}))_{t=1,...,T} \tag{3}$$

is an optimal solution to problem (1). Since this solution can be achieved by an omniscient oracle under a priori knowledge about content popularity, we call it the *oracle solution*.

However, in this paper we assume that the caching entity *does not* have a priori knowledge about content popularity. In this case, the caching entity cannot simply solve problem (1) as described above, since the expected demands $\mu_f(x) = E(d_f(x))$ are unknown. Hence, the caching entity has to learn these expected demands over time by observing the users' demands for cached files given the users' contexts. For this purpose, over time, the caching entity has to find a trade-off between caching files about which little information is available (*exploration*) and files of which it believes that they will yield the highest demands (*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 *learning algorithm*. The oracle solution given in (3) can be used as a benchmark to evaluate the loss of learning. Formally, the *regret* of learning with respect to the oracle solution is given by

$$\begin{aligned} R(T) = \sum_{t=1}^{T} \sum_{j=1}^{m} \sum_{i=1}^{U_t} v_{s_{t,i}} \Big(& w_{f_j^*(\mathbf{x_t}, \mathbf{s_t})} E\left(d_{f_j^*(\mathbf{x_t}, \mathbf{s_t})}(x_{t,i})\right) \\ & - E\left(w_{c_{t,j}} d_{c_{t,j}}(x_{t,i}, t)\right)\Big), \end{aligned} \tag{4}$$

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

In order to proactively cache the most suitable files given the context information about currently connected users, the caching entity should learn context-specific content popularity. Due to the above formal problem formulation, this problem corresponds to a contextual multi-armed bandit problem and we can adapt and extend a contextual learning algorithm [41], [42] to our setting. Our algorithm is based

[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.

on the assumption that users with similar context information will 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 starts by partitioning the context space uniformly into smaller sets, i.e., it splits the context space into parts of similar contexts. Then, the caching entity learns the content popularity independently in each of these sets of similar contexts. The algorithm operates in discrete time slots. In each time slot, the algorithm first observes the contexts of currently connected users. Then, the algorithm selects which files to cache in this time slot. Based on a certain control function, the algorithm is either in an exploration phase, in which it chooses a random set of files to cache. Theses phases are needed to learn the popularity of files which have not been cached often before. Otherwise, the algorithm is in an exploitation phase, in which it caches files which on average were requested most when cached in previous time slots with similar user contexts. After caching the new set of files, the algorithm observes the users' requests for these files. In this way, over time, the algorithm learns context-specific content popularity.

The algorithm for selecting m files is called *Context-Aware Proactive Caching with Cache Size m* (m-CAC) and its pseudocode is given in Fig. 4. 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., if 2 users with context from set p are connected in one time slot and file f is cached, this counter is increased by 2). Moreover, m-CAC keeps the estimated demand $\hat{d}_{f,p}(t)$ up to time slot t of each pair consisting of a file $f \in F$ and a set $p \in \mathcal{P}_T$. 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 up to time slot t. Then, the estimated demand of file f in set p is given by the sample mean $\hat{d}_{f,p}(t) := \frac{1}{|\mathcal{E}_{f,p}(t)|} \sum_{d \in \mathcal{E}_{f,p}(t)} d$.[2,3]

In each time slot t, m-CAC first observes the number of currently connected users U_t, their contexts $\mathbf{x_t} = (x_{t,i})_{i=1,\ldots,U_t}$ and the service types $\mathbf{s_t} = (s_{t,i})_{i=1,\ldots,U_t}$. For each context vector $x_{t,i}$, m-CAC determines the set $p_{t,i} \in \mathcal{P}_T$, to which the context vector belongs, i.e., such that $x_{t,i} \in p_{t,i}$ holds. The collection of these sets is given by $\mathbf{p_t} = (p_{t,i})_{i=1,\ldots,U_t}$. Then, the algorithm can either be in an exploration phase or in an exploitation phase. In order to determine the correct phase in 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* $F^{\text{ue}}_{\mathbf{p_t}}(t)$ is calculated based on

$$F^{\text{ue}}_{\mathbf{p_t}}(t) := \cup_{i=1}^{U_t} F^{\text{ue}}_{p_{t,i}}(t)$$
$$:= \cup_{i=1}^{U_t} \{f \in F : N_{f,p_{t,i}}(t) \leq K(t)\}, \tag{5}$$

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 V, 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 $F^{\text{ue}}_{\mathbf{p_t}}(t)$ to cache. If the set of under-explored files contains less than m elements, i.e., $u(t) < m$, it

m-CAC: Context-Aware Proactive Caching Algorithm

Input: T, h_T, $K(t)$
Initialize context partition: Create partition $\mathcal{P}_T$ of context space $[0,1]^D$ into $(h_T)^D$ hypercubes of identical size
Initialize counters: For all $f \in F$ and all $p \in \mathcal{P}_T$, set $N_{f,p} = 0$
Initialize estimated demands: For all $f \in F$ and all $p \in \mathcal{P}_T$, set $\hat{d}_{f,p} = 0$
for each $t = 1, ..., T$ **do**
  Observe number U_t of currently connected users
  Observe user contexts $\mathbf{x_t} = (x_{t,i})_{i=1,...,U_t}$ and service types $\mathbf{s_t} = (s_{t,i})_{i=1,...,U_t}$
  Find $\mathbf{p_t} = (p_{t,i})_{i=1,...,U_t}$ such that $x_{t,i} \in p_{t,i} \in \mathcal{P}_T, i = 1, ..., U_t$
  Compute the set of under-explored files $F^{\text{ue}}_{\mathbf{p_t}}(t)$ in (5)
  if $F^{\text{ue}}_{\mathbf{p_t}}(t) \neq \emptyset$ **then** ▷ Exploration
    $u = \text{size}(F^{\text{ue}}_{\mathbf{p_t}}(t))$
    if $u \geq m$ **then**
      Select $c_{t,1}, ..., c_{t,m}$ randomly from $F^{\text{ue}}_{\mathbf{p_t}}(t)$
    else
      Select $c_{t,1}, ..., c_{t,u}$ as the u files from $F^{\text{ue}}_{\mathbf{p_t}}(t)$
      Select $c_{t,u+1}, ..., c_{t,m}$ as the $(m-u)$ files $\hat{f}_{1,\mathbf{p_t},\mathbf{s_t}}(t), ..., \hat{f}_{m-u,\mathbf{p_t},\mathbf{s_t}}(t)$ from (6)
    end if
  else ▷ Exploitation
    Select $c_{t,1}, ..., c_{t,m}$ as the m files $\hat{f}_{1,\mathbf{p_t},\mathbf{s_t}}(t), ..., \hat{f}_{m,\mathbf{p_t},\mathbf{s_t}}(t)$ from (7)
  end if
  Observe demand $(d_{j,i})$ of each user $i = 1, ..., U_t$ for each file $c_{t,j}, j = 1, ..., m$
  for $i = 1, ..., U_t$ **do**
    for $j = 1, ..., m$ **do**
      $\hat{d}_{c_{t,j},p_{t,i}} = \frac{\hat{d}_{c_{t,j},p_{t,i}} N_{c_{t,j},p_{t,i}} + d_{j,i}}{N_{c_{t,j},p_{t,i}} + 1}$ and $N_{c_{t,j},p_{t,i}} = N_{c_{t,j},p_{t,i}} + 1$
    end for
  end for
end for

Fig. 4. Pseudocode of m-CAC.

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

[3]Note that in the pseudocode in Fig. 4, 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.

selects all $u(t)$ files from $F^{\mathrm{ue}}_{\mathbf{p_t}}(t)$ to cache. Since the cache is not fully filled by $u(t) < m$ files, $(m - u(t))$ additional files can be cached. In order to exploit knowledge obtained so far, m-CAC selects $(m - u(t))$ files from $F \setminus F^{\mathrm{ue}}_{\mathbf{p_t}}(t)$ based on a file ranking according to the estimated weighted demands, as defined by the files $\hat{f}_{1,\mathbf{p_t},\mathbf{s_t}}(t), \dots, \hat{f}_{m-u(t),\mathbf{p_t},\mathbf{s_t}}(t) \in F \setminus F^{\mathrm{ue}}_{\mathbf{p_t}}(t)$, which satisfy for $j = 1, \dots, m - u(t)$:

$$\hat{f}_{j,\mathbf{p_t},\mathbf{s_t}}(t) \in \underset{f \in F \setminus (F^{\mathrm{ue}}_{\mathbf{p_t}}(t) \cup \bigcup_{k=1}^{j-1}\{\hat{f}_{k,\mathbf{p_t},\mathbf{s_t}}(t)\})}{\operatorname{argmax}} w_f \sum_{i=1}^{U_t} v_{s_{t,i}} \hat{d}_{f,p_{t,i}}(t). \tag{6}$$

If the set of files defined by (6) 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 $F^{\mathrm{ue}}_{\mathbf{p_t}}(t)$ 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 $\hat{f}_{1,\mathbf{p_t},\mathbf{s_t}}(t), \dots, \hat{f}_{m,\mathbf{p_t},\mathbf{s_t}}(t) \in F$, which satisfy for $j = 1, \dots, m$:

$$\hat{f}_{j,\mathbf{p_t},\mathbf{s_t}}(t) \in \underset{f \in F \setminus \left(\bigcup_{k=1}^{j-1}\{\hat{f}_{k,\mathbf{p_t},\mathbf{s_t}}(t)\}\right)}{\operatorname{argmax}} w_f \sum_{i=1}^{U_t} v_{s_{t,i}} \hat{d}_{f,p_{t,i}}(t). \tag{7}$$

If the set of files defined by (7) is not unique, ties are again broken arbitrarily.

After selecting the subset of files to cache, the algorithm observes the users' requests for these files in this time slot. Then, it updates the estimated demands and the counters of cached files.

V. Analysis of the Regret

In this section, we give an upper bound on the regret $R(T)$ of m-CAC in (4). The regret bound is based on the natural assumption that expected demands for files are similar in similar contexts, i.e., that users with similar characteristics are likely to consume similar content. This assumption is realistic since the users' preferences in terms of consumed content differ based on the users' contexts, so that it is plausible to divide the user population into segments of users with similar context and similar preferences. Formally, the similarity assumption is 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 Euclidean norm in $\mathbb{R}^D$.

Assumption 1 is needed for the analysis of the regret, but it should be noted that m-CAC can also be applied if this assumption does not hold true. However, a regret bound might not be guaranteed in this case.

The following theorem 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 m-CAC converges to the oracle solution strategy. In other words, m-CAC converges to the optimal cache content placement strategy, which maximizes the expected number of cache hits. 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 $R(T)$ is $O\left(v_{\max} w_{\max} m U_{\max} R_{\max} |F| T^{\frac{2\alpha+D}{3\alpha+D}} \log(T)\right)$.

The proof can be found in our online appendix [44]. The regret bound given in Theorem 1 is sublinear in the time horizon T, proving that m-CAC converges to the optimal cache content placement strategy. 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 placement phases. Thus, Theorem 1 characterizes m-CAC's speed of convergence Furthermore, Theorem 1 shows that the regret bound is a constant multiple of the regret bound in the special case without service differentiation, in which $v_{\max} = 1$ and $w_{\max} = 1$. Hence, the order of the regret is $O\left(T^{\frac{2\alpha+D}{3\alpha+D}} \log(T)\right)$ in the special case as well.

VI. Memory Requirements

The memory requirements of m-CAC are mainly determined by the counters kept by the algorithm during its runtime (see also [41]). For each set p in the partition $\mathcal{P}_T$ and each file $f \in F$, the algorithm keeps the counters $N_{f,p}$ and $\hat{d}_{f,p}$. The number of files is $|F|$. If m-CAC runs with the parameters from Theorem 1, the number of sets in $\mathcal{P}_T$ is upper bounded by $(h_T)^D = \lceil T^{\frac{1}{3\alpha+D}} \rceil^D \leq 2^D T^{\frac{D}{3\alpha+D}}$. Hence, the required memory is upper bounded by $|F| 2^D T^{\frac{D}{3\alpha+D}}$ and is thus sublinear in the time horizon T. This means, that for $T \to \infty$, the algorithm would require 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 connected users' context vectors, the required number of counters that have to be kept can be much smaller than given by the upper bound.

VII. Extensions

A. Exploiting the Multicast Gain

So far, we assumed that each request for a cached file is immediately served by a unicast transmission. However, our algorithm can be extended to multicasting, which has been shown to be beneficial in combination with caching [7], [11]. For this purpose, to extend our algorithm, each time slot t is divided into a fixed number of intervals. In each interval, incoming requests are monitored and accumulated. At the end of the interval, requests for the same file are served by a multicast transmission. In order to exploit knowledge about content popularity learned so far, a request for a file with low estimated demand could, however, still be served by a unicast transmission. In this way, unnecessary delays are prevented in cases in which another request and thus a multicast transmission are not expected. Moreover, service differentiation could be taken into account. For example, high-priority users could be served by unicast transmissions, such that their delay is not increased due to waiting times for multicast transmissions.

B. Rating-Based Context-Aware Proactive Caching

So far, we considered cache content placement with respect to the demands $d_f(x)$ in order to maximize the number of (weighted) cache hits. However, a CP operating an infostation might want to cache not only content that is requested often, but which also receives high ratings from the users. Consider the case that after consumption users rate content in a range $[r_{\min}, r_{\max}] \subset \mathbb{R}_+$. 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

$$\tilde{d}_f(x) := r_f(x) d_f(x), \tag{8}$$

which combines the demand and the rating for file f of a user with context x. By carefully designing the range of ratings, the CP chooses the trade-off between ratings and cache hits. Now, we can apply m-CAC with respect to $\tilde{d}_f(x)$. In this case, m-CAC additionally needs to observe the users' ratings in order to learn content popularity in terms of ratings. If the users' ratings are always available, Theorem 1 applies and provides a regret bound of $O\left(v_{\max} w_{\max} r_{\max} m U_{\max} R_{\max} |F| T^{\frac{2\alpha+D}{3\alpha+D}} \log(T)\right)$.

However, users might not always reveal a rating after consuming a content. When a user's rating is missing, we assume that m-CAC does not update the counters based on this user's request. This may result in a higher required number of exploration phases. Hence, the regret of the learning algorithm is influenced by the users' willingness to reveal ratings of requested content. Let $q \in (0, 1)$ be the probability that a user reveals his rating after requesting a file. Then, the regret of the learning algorithm is bounded as given below.

Theorem 2 (Bound for $R(T)$ for Rating-Based Caching With Missing Ratings): 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 with respect to $\tilde{d}_f(x)$, Assumption 1 holds true for $\tilde{d}_f(x)$ and a user reveals his rating with probability q, the leading order of the regret $R(T)$ is $O\left(\frac{1}{q} v_{\max} w_{\max} r_{\max} m U_{\max} R_{\max} |F| T^{\frac{2\alpha+D}{3\alpha+D}} \log(T)\right)$.

The proof can be found in our online appendix [44]. Comparing Theorem 2 with Theorem 1, the regret of m-CAC is scaled up by a factor $\frac{1}{q} > 1$ in case of rating-based caching with missing ratings. This factor corresponds to the expected number of requests until the caching entity receives one rating. However, the time order of the regret remains the same. Hence, m-CAC is robust under missing ratings in the sense that if some users refuse to rate requested content, the algorithm still converges to the optimal cache content placement strategy.

C. Asynchronous User Arrival

So far, we assumed that the set of currently connected users only changes in between the time slots of our algorithm. This means, that only those users connected to the caching entity at the beginning of a time slot, will request files within that time slot. However, if users connect to the caching entity asynchronously, m-CAC should be adapted. If a user directly disconnects after the context monitoring without requesting any file, he should be excluded from learning. Hence, in m-CAC, the counters are not updated for disconnecting users. If a user connects to the caching entity after cache content placement, his context was not considered in the caching decision. However, his requests can be used to learn faster. Hence, in m-CAC, the counters are updated based on this user's requests.

D. Multiple Wireless Local Caching Entities

So far, we considered online learning for cache content placement in a single caching entity. However, real caching systems contain multiple caching entities, each of which should learn local content popularity. In a network of multiple caching entities, m-CAC could be applied separately and independently by each caching entity. For the case that coverage areas of caching entities overlap, in this subsection, we present m-CACao, an extension of m-CAC to *Context-Aware Proactive Caching with Area Overlap*. The idea of m-CACao is that caching entities can learn content popularity faster by not only relying on their own cache hits, but also on cache hits at neighboring caching entities with overlapping coverage area. For this purpose, the caching entities overhear cache hits produced by users in the intersection to neighboring coverage areas.

In detail, m-CAC is extended to m-CACao as follows: The context monitoring and the selection of cache content works as in m-CAC. However, the caching entity not only observes its own cache hits (line 21 in Fig. 4), but it overhears cache hits at neighboring caching entities of users in the intersection. Then, the caching entity not only updates the counters of its own cached files (lines 22-26 in Fig. 4), but it additionally updates the counters of files of which it overheard cache hits at neighboring caches. This helps the caching entity to learn faster.

VIII. Numerical Results

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

A. Description of the Data Set

We use a data set from MovieLens [45] to evaluate our proposed algorithm. MovieLens is an online movie recommender operated by the research group GroupLens from the University of Minnesota. The MovieLens 1M DataSet [46] 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

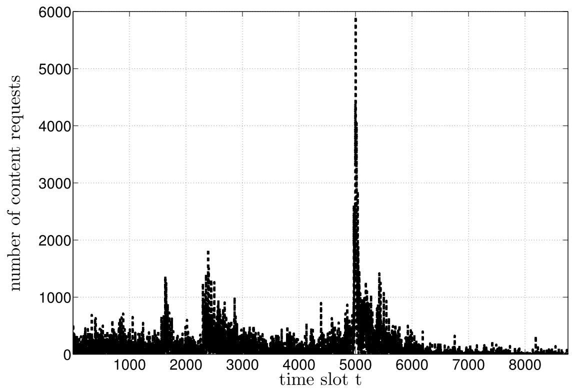


Fig. 5. Number of content requests in used data set as a function of time slots. Time slots at an hourly basis.

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 wireless local caching entity (see [33], [34] 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 caching entity (in case the movie is cached in the caching entity) or to the macro cellular network (in case the movie is not cached in the caching entity). This approach is reasonable since users typically rate movies after watching them.

In our simulations, we only use the data gathered within the first year of the data set, since around 94% of the ratings were provided within this time frame. 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. Then, we assign the requests and corresponding user contexts to the time slots according to their timestamps and we interpret each request as if it was coming from a separate user. At the beginning of a time slot, we assume to have access to the context of each user responsible for a request in the coming time slot. Fig. 5 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.[4]

B. Reference Algorithms

We compare m-CAC with five reference algorithms. The first algorithm is the omniscient Oracle, which has complete knowledge about the exact future demands. In each time slot, the oracle selects the optimal m files that will maximize the number of cache hits in this time slot.[5]

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 [35], 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 time slot, we modify UCB to take m actions at a time, which corresponds to selecting m files.

The third reference algorithm is the m-ϵ-Greedy. This is a variant of the simple ϵ-Greedy [35] algorithm, which does not consider context information. The m-ϵ-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 fourth reference algorithm is called m-Myopic. This is an algorithm taken from [15], which is investigated since it is comparable to the well-known Least Recently Used algorithm (LRU) for caching. m-Myopic only learns from one time slot in the past. It starts with a random set of files and in each of the following time slots discards all files that have not been requested in the previous time slot. Then, it randomly replaces the discarded files by other files.

The fifth reference algorithm, called Random, caches a random set of files in each time slot.

C. Performance Measures

The following performance measures are used in our analysis. The evolution of per-time slot or cumulative *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 requests which can be served by cached files.

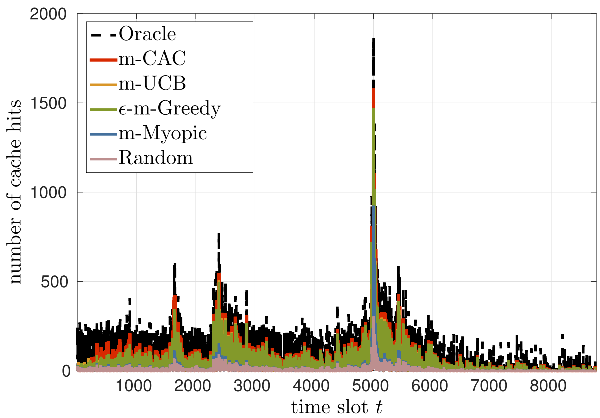
D. Results

In our simulations, we set $\epsilon = 0.09$ in m-ϵ-Greedy, which is the value at which heuristically the algorithm on average performed best. In m-CAC, we set the control function to $K(t) = c \cdot t^{\frac{2\alpha}{3\alpha+D}} \log(t)$ with $c = 1/(|F|D)$.[6] The simulation results are obtained by averaging over 100 runs of the algorithms. First, we consider the case without service differentiation. The long-term behavior of m-CAC 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 file library size. We run all algorithms on the data set and study their results as a function of time, i.e., over the time slots $t = 1, ..., T$. Fig. 6(a) and 6(b) show the per-time slot and the cumulative numbers of cache hits up to time slot t as a function of time, respectively. Due to the bursty content request process (compare Fig. 5), also the number of cache hits achieved by the different algorithms is bursty over time. As expected, the Oracle gives an upper bound to the other algorithms. Among the other algorithms, m-CAC, m-ϵ-Greedy and m-UCB clearly

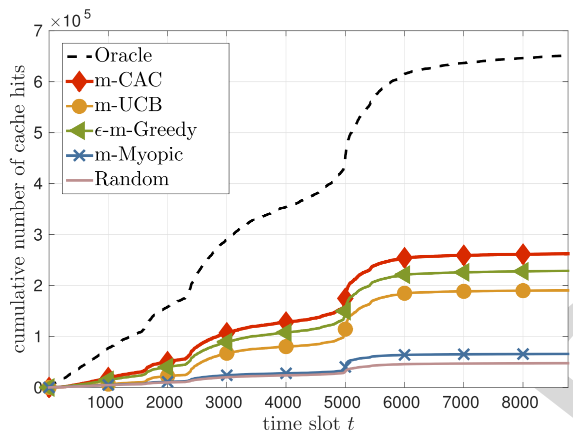
[4] We neglect the occupation as context dimension since by mapping them to a [0,1] variable, we would have to classify which occupations are more and which are less similar to each other.

[5] Note that this oracle yields even better results than the oracle used as a benchmark to define the regret in (4). In the definition of regret, the oracle only exploits knowledge about expected demands, instead of exact future demands.

[6] Compared to the control function in Theorem 1, the additional factor reduces the number of exploration phases which allows for better performance.



(a) Number of cache hits per time slot.



(b) Cumulative number of cache hits.

Fig. 6. Time evolution of algorithms for $m = 200$.

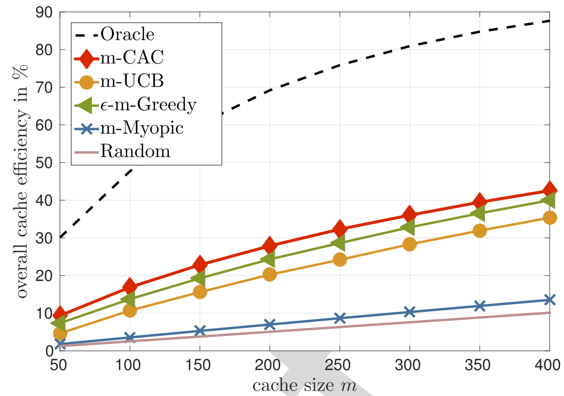


Fig. 7. Overall cache efficiency at T as a function of cache size m.

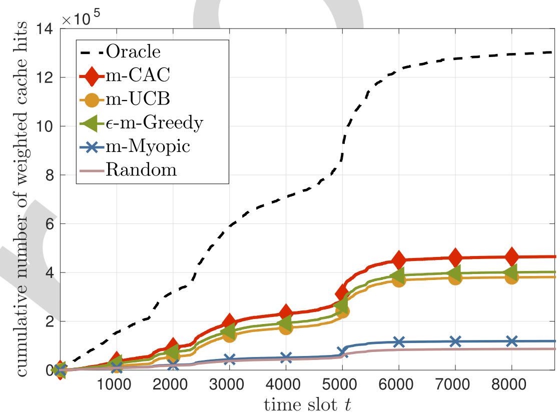


Fig. 8. Cumulative number of weighted cache hits for $m = 200$ as a function of time.

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-ϵ-Greedy shows a better performance than m-UCB, even though it uses a simpler learning strategy. Overall, m-CAC outperforms the other algorithms by additionally learning from context information. At the time horizon, the cumulative number of cache hits achieved by m-CAC is 1.146, 1.377, 3.985 and 5.506 times higher than the ones achieved by m-ϵ-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, which corresponds to between 1.3% and 10.1% of the file library size, which is a realistic assumption. All remaining parameters are kept as before. Fig. 7 shows the overall cache efficiency achieved at the time horizon T as a function of cache size, i.e., the cumulative number of cache hits up to T is normalized by the cumulative number of requests up to T. The overall cache efficiency of all algorithms is increasing with increasing cache size. Moreover, the results indicate that again m-CAC and m-ϵ-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 28.4%, compared to an average cache efficiency of 25.3%, 21.4%, 7.76% and 5.69% achieved by m-ϵ-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 with weights $v_1 = 5$ and $v_2 = 1$ exist. 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 each algorithm to take into account service differentiation by incorporating the weights according to the service types. Fig. 8 shows the cumulative number of weighted cache hits up to time slot t as a function of time. At the time horizon, the cumulative number of weighted cache hits achieved by m-CAC is 1.156, 1.219, 3.914 and 5.362 times higher than the ones achieved by m-ϵ-Greedy, m-UCB, m-Myopic and Random, respectively. A comparison with Fig. 6(b) shows that the behavior of the algorithms is similar to the case without service differentiation.

Finally, we investigate the extension to multiple caching entities and compare the performance of the proposed algorithms m-CAC and m-CACao. We consider a scenario with two caching entities and divide the data set as follows: A fraction $o \in [0, 0.3]$ of randomly selected requests is considered to be made in the intersection of the two coverage areas. We use the parameter o as a measure of the overlap between the caching entities. The remaining requests are randomly assigned to either one of the caching entities.

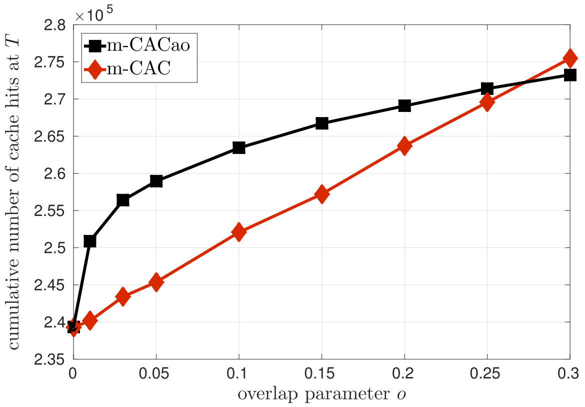


Fig. 9. Cumulative number of cache hits at T as a function of the overlap parameter o.

These requests are considered to be made by users solely connected to one caching entity. Then, on the one hand we run m-CAC separately on each caching entity and on the other hand we run m-CACao on both caching entities. Fig. 9 shows the cumulative number of cache hits achieved in sum by the two caching entities at the time horizon T as a function of the overlap parameter o. As expected, m-CAC and m-CACao perform identically for non-overlapping coverage areas. With increasing overlap, the number of cache hits achieved by both m-CAC and m-CACao increases. The reason is that users in the intersection can more likely be served since they have access to both caches. Hence, even though the caching entities do not coordinate their cache content, more cache hits occur. For up to 25% of overlap ($o \leq 0.25$), m-CACao outperforms m-CAC. Clearly, m-CACao performs better since by overhearing cache hits at the neighboring caching entity, both caching entities learn content popularity faster. For very large overlap ($o > 0.25$), m-CAC yields higher numbers of cache hits. The reason is that when applying m-CACao in case of a large overlap, neighboring caching entities overhear such a large number of cache hits, that they learn very similar content popularity distributions. Hence, over time it is likely that their caches contain the same files. In contrast, applying m-CAC, a higher diversity in cache content is maintained over time. Clearly, further gains in cache hits could be achieved by jointly optimizing the cache content of all caching entities. However, this would either require coordination among the caching entities or a central planner deciding on the cache content of all caching entities, which results in a high communication overhead. In contrast, our heuristic algorithm m-CACao does not require additional coordination or communication and yields good results for small overlaps.

IX. Conclusion

In this paper, we presented a context-aware proactive caching algorithm for wireless caching entities based on contextual multi-armed bandits. To cope with unknown and fluctuating content popularity among the dynamically arriving and leaving users, the algorithm regularly observes context information of connected users, updates the cache content and subsequently observes cache hits. In this way, the algorithm learns context-specific content popularity online, which allows for a proactive adaptation of cache content according to fluctuating local content popularity. We derived a sublinear regret bound, which characterizes the learning speed and proves that our proposed algorithm converges to the optimal cache content placement strategy, which maximizes the expected number of cache hits. Moreover, the algorithm supports customer prioritization and can be combined with multicast transmissions and rating-based caching decisions. Numerical studies showed that by exploiting context information, our algorithm outperforms state-of-the-art algorithms in a real world data set.

References

[1] 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 Int. Conf. Commun. (ICC)*, May 2016, pp. 1–7.

[2] [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html AQ:4

[3] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. M. Leung, "Cache in the air: Exploiting content caching and delivery techniques for 5G systems," *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 131–139, Feb. 2014.

[4] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Mar. 2010, pp. 1–9.

[5] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *Proc. IEEE Comput. Commun. 18th Annu. Joint Conf. Soc. (INFOCOM)*, vol. 1. Mar. 1999, pp. 126–134.

[6] J. Erman, A. Gerber, M. Hajiaghayi, D. Pei, S. Sen, and O. Spatscheck, "To cache or not to cache: The 3G case," *IEEE Internet Comput.*, vol. 15, no. 2, pp. 27–34, Mar. 2011.

[7] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Trans. Inf. Theory*, vol. 60, no. 5, pp. 2856–2867, May 2014.

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

[9] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "FemtoCaching: Wireless content delivery through distributed caching helpers," *IEEE Trans. Inf. Theory*, vol. 59, no. 12, pp. 8402–8413, Dec. 2013.

[10] K. Poularakis and L. Tassiulas, "Exploiting user mobility for wireless content delivery," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2013, pp. 1017–1021.

[11] K. Poularakis, G. Iosifidis, V. Sourlas, and L. Tassiulas, "Exploiting caching and multicast for 5G wireless networks," *IEEE Trans. Wireless Commun.*, vol. 15, no. 4, pp. 2995–3007, Apr. 2016.

[12] E. Bastug, M. Bennis, and M. Debbah, "Cache-enabled small cell networks: Modeling and tradeoffs," in *Proc. Int. Symp. Wireless Commun. Syst. (ISWCS)*, Aug. 2014, pp. 649–653.

[13] E. Baştuğ, M. Bennis, and M. Debbah, "Living on the edge: The role of proactive caching in 5G wireless networks," *IEEE Commun. Mag.*, vol. 52, no. 8, pp. 82–89, Aug. 2014.

[14] E. Baştuğ *et al.*, "Big data meets telcos: A proactive caching perspective," *J. Commun. Netw.*, vol. 17, no. 6, pp. 549–557, Dec. 2015.

[15] P. Blasco and D. Gündüz, "Learning-based optimization of cache content in a small cell base station," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2014, pp. 1897–1903.

[16] P. Blasco and D. Gündüz, "Multi-armed bandit optimization of cache content in wireless infostation networks," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2014, pp. 51–55.

[17] P. Blasco and D. Gündüz. (Jul. 2014). "Content-level selective offloading in heterogeneous networks: Multi-armed bandit optimization and regret bounds." [Online]. Available: https://arxiv.org/abs/1407.6154

[18] A. Sengupta, S. Amuru, R. Tandon, R. M. Buehrer, and T. C. Clancy, "Learning distributed caching strategies in small cell networks," in *Proc. IEEE Int. Symp. Wireless Commun. Syst. (ISWCS)*, Aug. 2014, pp. 917–921.

[19] M. ElBamby, M. Bennis, W. Saad, and M. Latva-aho, "Content-aware user clustering and caching in wireless small cell networks," in *Proc. IEEE Int. Symp. Wireless Commun. Syst. (ISWCS)*, Aug. 2014, pp. 945–949.

[20] D. Goodman, J. Borras, N. B. Mandayam, and R. D. Yates, "INFOSTATIONS: A new system model for data and messaging services," in *Proc. IEEE 47th Veh. Technol. Conf.*, vol. 2. May 1997, pp. 969–973.

[21] A. L. Iacono and C. Rose, "Infostations: New perspectives on wireless data networks," in *Next Generation Wireless Networks*, S. Tekinay, Ed. Boston, MA, USA: Springer, 2002, pp. 3–63.

[22] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "YouTube traffic characterization: A view from the edge," in *Proc. ACM Conf. Internet Meas. (IMC)*, Oct. 2007, pp. 15–28.

[23] 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.

[24] A. Brodersen, S. Scellato, and M. Wattenhofer, "YouTube around the world: Geographic popularity of videos," in *Proc. ACM Int. Conf. World Wide Web (WWW)*, Apr. 2012, pp. 241–250.

[25] M.-L. Mares and Y. Sun, "The multiple meanings of age for television content preferences," *Human Commun. Res.*, vol. 36, no. 3, pp. 372–396, Jul. 2010.

[26] C. A. Hoffner and K. J. Levine, "Enjoyment of mediated fright and violence: A meta-analysis," *Media Psychol.*, vol. 7, no. 2, pp. 207–237, Nov. 2005.

[27] P. J. Rentfrow, L. R. Goldberg, and R. Zilca, "Listening, watching, and reading: The structure and correlates of entertainment preferences," *J. Personality*, vol. 79, no. 2, pp. 223–258, Apr. 2011.

[28] D. Zillmann, "Mood management through communication choices," *Amer. Behavioral Sci.*, vol. 31, no. 3, pp. 327–340, Jan. 1988.

[29] C. Zhou, Y. Guo, Y. Chen, X. Nie, and W. Zhu, "Characterizing user watching behavior and video quality in mobile devices," in *Proc. IEEE Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2014, pp. 1–6.

[30] B.-J. Ko, K.-W. Lee, K. Amiri, and S. Calo, "Scalable service differentiation in a shared storage cache," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst. (ICDCS)*, May 2003, pp. 184–193.

[31] 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.

[32] P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms," in *Proc. USENIX Symp. Internet Technol. Syst.*, 1997, pp. 193–206.

[33] S. Li, J. Xu, M. van der Schaar, and W. Li, "Popularity-driven content caching," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.

[34] S. Li, J. Xu, M. van der Schaar, and W. Li, "Trend-aware video caching through Online learning," *IEEE Trans. Multimedia*, vol. 18, no. 12, pp. 2503–2516, Dec. 2016.

[35] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, nos. 2–3, pp. 235–256, May 2002.

[36] S. Maghsudi and E. Hossain, "Multi-armed bandits with application to 5G small cells," *IEEE Wireless Commun.*, vol. 23, no. 3, pp. 64–73, Jun. 2016.

[37] S. Amuru, C. Tekin, M. van der Schaar, and R. M. Buehrer, "Jamming bandits—A novel learning method for optimal jamming," *IEEE Trans. Wireless Commun.*, vol. 15, no. 4, pp. 2792–2808, Apr. 2016.

[38] C. Shen, C. Tekin, and M. van der Schaar, "A non-stochastic learning approach to energy efficient mobility management," *IEEE J. Sel. Areas Commun.*, to be published. AQ:5

[39] T. Lu, D. Pal, and M. Pal, "Contextual multi-armed bandits," in *Proc. Int. Conf. Artif. Intell. Statist. (AISTATS)*, 2010, pp. 485–492.

[40] A. Slivkins, "Contextual bandits with similarity information," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 2533–2568, Jan. 2014.

[41] C. Tekin and M. van der Schaar, "Distributed Online learning via cooperative contextual bandits," *IEEE Trans. Signal Process.*, vol. 63, no. 14, pp. 3700–3714, Mar. 2015.

[42] C. Tekin, S. Zhang, and M. van der Schaar, "Distributed Online learning in social recommender systems," *IEEE J. Sel. Topics Signal Process.*, vol. 8, no. 4, pp. 638–652, Aug. 2014.

[43] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Berlin, Germany: Springer, 2004.

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

[45] F. M. Harper and J. A. Konstan, "The MovieLens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 1–19, Dec. 2015.

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

[47] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *J. Amer. Statist. Assoc.*, vol. 58, no. 301, pp. 13–30, Mar. 1963.

[48] E. Chlebus, "An approximate formula for a partial sum of the divergent pp-series," *Appl. Math. Lett.*, vol. 22, no. 5, pp. 732–737, May 2009.

Sabrina Müller (S'15) received the B.Sc. and the M.Sc. degrees in mathematics from the Technische Universität Darmstadt, Germany, in 2012 and 2014, respectively, where she is currently pursuing the Ph.D. degree in electrical engineering. She is a member of the Communications Engineering Laboratory, Technische Universität Darmstadt. Her research interests include machine learning and optimization methods and their applications to wireless networks.

Onur Atan received the B.Sc. degree from Bilkent University, Ankara, Turkey, in 2013 and the M.Sc. degree from the University of California at Los Angeles, Los Angeles, CA, USA, in 2014, where he is currently pursuing the Ph.D. degree in electrical engineering. He received the Best M.Sc. Thesis Award in electrical engineering from the University of California at Los Angeles. His research interests include online learning and multi-armed bandit problems.

Mihaela van der Schaar (M'99–SM'04–F'10) was a Distinguished Lecturer with the Communications Society from 2011 to 2012. She is currently a Chancellor's Professor of Electrical Engineering with the University of California at Los Angeles, Los Angeles, CA, USA. She holds 33 granted U.S. patents. Her research interests include machine learning for decision making, medical informatics and education, online learning, real-time stream mining, network science, engineering economics, social networks, game theory, wireless networks, and multimedia. She was a member of the Editorial Board of the IEEE JOURNAL OF SELECTED TOPICS IN SIGNAL PROCESSING in 2011. She was a recipient of the NSF CAREER Award in 2004, the Best Paper Award from the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY in 2005, the Okawa Foundation Award in 2006, the IBM Faculty Award in 2005, 2007, and 2008, the Most Cited Paper Award from the EURASIP *Journal on Image Communications* in 2006, the Gamenets Conference Best Paper Award in 2011, and the IEEE CIRCUITS AND SYSTEMS Society Darlington Award Best Paper Award in 2011. She received three ISO Awards for her contributions to the MPEG video compression and streaming international standardization activities. She was the Editor-in-Chief of the IEEE TRANSACTIONS ON MULTIMEDIA from 2011 to 2013. AQ:6

Anja Klein (M'96) received the Diploma and Dr.-Ing. (Ph.D.) degrees in electrical engineering from the University of Kaiserslautern, Germany, in 1991 and 1996, respectively. She was involved in the standardization of third generation mobile radio in ETSI and in 3GPP, for instance leading the TDD Group in RAN1 of 3GPP. She was a Vice-President, heading the Development Department and the Systems Engineering Department. In 1996, she joined Siemens AG, Mobile Networks Division, Munich and Berlin. In 2004, she joined the Technische Universität Darmstadt, Germany, as a Full Professor, heading the Communications Engineering Laboratory. She has authored over 290 refereed papers and has contributed to 12 books. She is an inventor and a co-inventor of over 45 patents in the field of mobile radio. Her main research interests are in mobile radio, including interference management, cross-layer design, relaying and multi-hop, computation offloading, smart caching, and energy harvesting. In 1999, she was named the Inventor of the Year by Siemens AG. She is a member of Verband Deutscher Elektrotechniker-Informationstechnische Gesellschaft. AQ:7 AQ:8

AUTHOR QUERIES

AUTHOR PLEASE ANSWER ALL QUERIES

PLEASE NOTE: We cannot accept new source files as corrections for your paper. If possible, please annotate the PDF proof we have sent you with your corrections and upload it via the Author Gateway. Alternatively, you may send us your corrections in list format. You may also upload revised graphics via the Author Gateway.

AQ:1 = Please be advised that per instructions from the Communications Society this proof was formatted in Times Roman font and therefore some of the fonts will appear different from the fonts in your originally submitted manuscript. For instance, the math calligraphy font may appear different due to usage of the usepackage[mathcal]euscript. We are no longer permitted to use Computer Modern fonts.
AQ:2 = In the provided source PDF, 48 references were provided, whereas in the source tex, only 46 were provided. We have followed the source tex in this case. Please confirm.
AQ:3 = Note that if you require corrections/changes to tables or figures, you must supply the revised files, as these items are not edited for you.
AQ:4 = Please provide the accessed date for refs. [2], [44], and [46].
AQ:5 = Please provide the volume no., issue no., page range, month, and year for ref. [38].
AQ:6 = Please provide the author photo for "Mihaela van der Schaar."
AQ:7 = Please confirm whether the edits made in the sentence "She is an..."
AQ:8 = Please confirm whether the retention of the sentence "In 1999, she... Gesellschaft."

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

Sabrina Müller, *Student Member, IEEE*, Onur Atan, Mihaela van der Schaar, *Fellow, IEEE*, and Anja Klein, *Member, IEEE*

AQ:1 ***Abstract*— Content caching in small base stations or wireless infostations is considered to be a suitable approach to improving the efficiency in wireless content delivery. Placing the optimal content into local caches is crucial due to storage limitations, but it requires knowledge about the content popularity distribution, which is often not available in advance. Moreover, local content popularity is subject to fluctuations, since mobile users with different interests connect to the caching entity over time. Which content a user prefers may depend on the user's context. In this paper, we propose a novel algorithm for context-aware proactive caching. The algorithm learns context-specific content popularity online by regularly observing context information of connected users, updating the cache content and observing cache hits subsequently. We derive a sublinear regret bound, which characterizes the learning speed and proves that our algorithm converges to the optimal cache content placement strategy in terms of maximizing the number of cache hits. Furthermore, our algorithm supports service differentiation by allowing operators of caching entities to prioritize customer groups. Our numerical results confirm that 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*— Wireless networks, caching at the edge, cache content placement, online learning.**

I. Introduction

AQ:2 WIRELESS networks have been experiencing a steep increase in data traffic in recent years [2]. With 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 expected to increase and to account for the majority of mobile data traffic within the next few years [2]. However, despite recent advances in cellular mobile radio networks, these networks cannot keep up with the massive growth of mobile data traffic [3]. As already investigated for wired networks [4], *content caching* is envisioned to improve the efficiency 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 [5].

Within wireless networks, *caching at the edge* has been extensively studied [1], [6]–[19]. At the radio access network level, current approaches comprise two types of *wireless local caching entities*. The first type are *macro base stations* (MBSs) and *small base stations* (SBSs) that are implemented in wireless small cell networks, dispose of limited storage capacities and are typically owned by the *mobile network operator* (MNO). The second type are *wireless infostations* with limited storage capacities that provide high bandwidth local data communication [16], [17], [20], [21]. Wireless infostations could be installed in public or commercial areas and could use Wi-Fi for local data communication. They could be owned by *content providers* (CPs) aiming at increasing their users' quality of experience. Alternatively, third parties (e.g., the owner of a commercial area) could offer caching at infostations as a service to CPs or to the users [17]. Both types of caching entities store a fraction of available popular content in a *placement phase* and serve local users' requests via localized communication in a *delivery phase*.

Due to the vast amount of content available in multimedia platforms, not all available content can be stored in local caches. Hence, 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, however, when caching content at a particular point in time, it is unclear which content will be requested in future. Not even an estimate of the content popularity distribution might be at hand. It therefore must be computed by the caching entity itself [1], [13]–[19], which is not only legitimate from an overhead point of view, since else a periodic coordination with the global multimedia platform would be required. More importantly, local content popularity in a caching entity might not even replicate global content popularity as monitored by the global multimedia platform [22]–[24]. Hence, caching entities should learn local content popularity for a *proactive* cache content placement. Secondly, different content can be favored by different users. Consequently, local content popularity may change according to the different preferences of fluctuating mobile users in the vicinity of a

Manuscript received May 4, 2016; revised September 30, 2016; accepted November 20, 2016. The work by S. Müller and A. Klein was supported by the German Research Foundation as part of Projects B3 and C1 within the Collaborative Research Center 1053–MAKI. The work by O. Atan and M. van der Schaar was supported by the NSF under Grant CCF1524417 and Grant ECCS1407712. This paper was presented at the IEEE International Conference on Communications, May 2016 [1]. The associate editor coordinating the review of this paper and approving it for publication was B. Hamdaoui.

S. Müller and A. Klein are with the Communications Engineering Laboratory, Technische Universität Darmstadt, 64283 Darmstadt, Germany (e-mail: s.mueller@nt.tu-darmstadt.de; a.klein@nt.tu-darmstadt.de).

O. Atan and M. van der Schaar are with the Department of Electrical Engineering, University of California at Los Angeles, Los Angeles, CA 90095 USA (e-mail: oatan@ucla.edu; mihaela@ee.ucla.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TWC.2016.2636139

caching entity. Therefore, proactive cache content placement should take into account the *diversity in content popularity* across the local user population. Thirdly, the users' preferences in terms of consumed content may differ based on their contexts, such as their location [24], personal characteristics (e.g., age [25], gender [26], personality [27], mood [28]), or their devices' characteristics [29]. Hence, cache content placement should be *context-aware* by taking into account that content popularity depends on a user's context. Thereby, a caching entity can learn the preferences of users with different contexts. Fourthly, while its typical goal is to maximize the number of cache hits, cache content placement should also take into account the cache 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 [30], [31]. For example, if users with different preferences are connected to a caching entity, the operator could prioritize certain users by caching content favored by these users. Moreover, certain CPs' content could be prioritized in caching decisions.

In this paper, we propose a novel context-aware proactive caching algorithm, which for the first time *jointly* considers the above four aspects. Firstly, instead of assuming a priori knowledge about content popularity, which might be externally given or estimated in a separate training phase, our algorithm learns the content popularity online by observing the users' requests for cache content. Secondly, by explicitly allowing different content to be favored by different users, our algorithm is especially suitable for mobile scenarios, in which users with different preferences arrive at the wireless caching entity over time. Thirdly, we explicitly model that the content popularity depends on a user's context, such as his/her personal characteristics, equipment, or external factors, and propose an algorithm for content caching that learns this context-specific content popularity. Using our algorithm, a caching entity can proactively cache content for the currently connected users based on what it has previously learned, instead of simply caching the files that are popular "on average", across the entire population of users. The learned cache content placement strategy is proven to converge to the optimal cache content placement strategy which maximizes the expected number of cache hits. Fourthly, the algorithm allows for service differentiation by customer prioritization. The contributions of this paper are as follows:

- We present a context-aware proactive caching algorithm based on contextual multi-armed bandit optimization. Our algorithm incorporates diversity in content popularity across the user population and takes into account the dependence of users' preferences on their context. Additionally, it supports service differentiation by prioritization.
- We analytically bound the loss of the algorithm compared to an oracle, which assumes a priori knowledge about content popularity. We derive a sublinear regret bound, which characterizes the learning speed and proves that our algorithm converges to the optimal cache content placement strategy which maximizes the expected number of cache hits.
- We present additional extensions of our approach, such as its combination with multicast transmissions and the incorporation of caching decisions based on user ratings.
- We numerically evaluate our caching algorithm based on a real world data set. A comparison shows that by exploiting context information in order to proactively cache content for currently connected users, 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. 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 propose some extensions of the algorithm. Numerical results are presented in Section VIII. Section IX concludes the paper.

II. Related Work

Practical caching systems often 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) (see [32]). While these simple algorithms do not consider future content popularity, recent work has been devoted to developing sophisticated cache replacement algorithms by learning content popularity trends [33], [34].

In this paper, however, we focus 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. Information-theoretic gains achieved by combining caching at user devices with a coded multicast transmission in the delivery phase are calculated in [7]. The proposed coded caching approach is optimal up to a constant factor. Content caching at user devices and collaborative device-to-device communication are combined in [8] to increase the efficiency of content delivery. In [9], an approximation algorithm for uncoded caching among SBSs equipped with caches is given, which minimizes the average delay experienced by users that can be connected to several SBSs simultaneously. Building upon the same caching architecture, in [10], an approximation algorithm for distributed coded caching is presented for minimizing the probability that moving users have to request parts of content from the MBS instead of the SBSs. In [11], a multicast-aware caching scheme is proposed for minimizing the energy consumption in a small cell network, in which the MBS and the SBSs can perform multicast transmissions. The outage probability and average content delivery rate in a network of SBSs equipped with caches are analytically calculated in [12].

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

TABLE I
COMPARISON WITH RELATED WORK ON LEARNING-BASED CACHING WITH PLACEMENT AND DELIVERY PHASE

AQ:3

	[13], [14]	[15]–[17]	[18]	[19]	This work
Model-Free	Yes	Yes	No	Yes	Yes
Online/Offline-Learning	Offline	Online	Online	Online	Online
Free of Training Phase	No	Yes	Yes	No	Yes
Performance Guarantees	No	Yes	No	No	Yes
Diversity in Content Popularity	No	No	No	Yes	Yes
User Context-Aware	No	No	No	No	Yes
Service Differentiation	No	No	No	No	Yes

proactive caching paradigm, [13] and [14] propose a caching 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 ratio based on their required delivery rates. 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 the content popularity online, thus also adapting to varying content popularities. In [15], using a multi-armed bandit algorithm, an SBS learns a fixed content popularity distribution online by refreshing its cache content and observing instantaneous demands for cached files. In this way, cache content placement is optimized over time to maximize the traffic served by the SBS. The authors extend their framework for a wireless infostation in [16] and [17] by additionally taking into account the costs for adding files to the cache. Moreover, they provide theoretical sublinear regret bounds for their algorithms. A different extension of the multi-armed bandit framework is given in [18], which exploits the topology of users' connections to the SBSs by incorporating coded caching. The approach in [18] assumes a specific type of content popularity distribution. Since in practice the type of 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 type of content popularity distribution. Moreover, in [15]–[18], the optimal cache content placement strategy is learned over time based only on observations of instantaneous demands. In contrast, our proposed algorithm additionally takes diversity of content popularity across the user population into account and exploits users' context information. Diversity in content popularity across the user population is for example taken into account in [19], but again without considering the users' contexts. Users are clustered into groups of similar interests by a spectral clustering algorithm based on their requests in a training phase. Each user group is then assigned to an SBS which learns the content popularity of its fixed user group over time. Hence, in [19], each SBS learns a fixed content popularity distribution under the assumption of a stable user population, whereas our approach allows reacting to arbitrary arrivals of users preferring different content.

In summary, compared to related work on cache content placement (see Table I), our proposed algorithm for the first time *jointly* learns the content popularity online, allows for diversity in content popularity across the user population, takes into account the dependence of users' preferences on their context and supports service differentiation. Compared to our previous work [1], we now take into account context information at a single user level, instead of averaging context information over the currently connected users. This enables more fine-grained learning. Additionally, we incorporate service differentiation and present extensions, e.g., to multicast transmission and caching decisions based on user ratings.

We model the caching problem as a multi-armed bandit problem. Multi-armed bandit problems [35] have been applied to various scenarios in wireless communications before [36], such as cognitive jamming [37] or mobility management [38]. Our algorithm is based on *contextual multi-armed bandit* algorithms [39]–[42]. The closest related work is [42], in which several learners observe a single context arrival in each time slot and select a subset of actions to maximize the sum of expected rewards. While [42] considers multiple learners, our system has only one learner – the caching entity selecting a subset of files to cache in each time slot. Compared to [42], we extended the algorithm in the following directions: We allow multiple context arrivals in each time slot, and select a subset of actions which maximize the sum of expected rewards given the context arrivals. In the caching scenario, this translates to observing the contexts of all currently connected users and caching a subset of files which maximize the sum of expected numbers of cache hits given the users' contexts. In addition, we enable each arriving context to be annotated with a weight, so that if different contexts arrive within the same time slot, differentiated services can be provided per context, by selecting a subset of actions which maximize the sum of expected weighted rewards. In the caching scenario, this enables the caching entity to prioritize certain users when selecting the cache content, by placing more weight on files that are favored by prioritized users. Moreover, we enable each action to be annotated with a weight, such that actions can be prioritized for selection. In the caching scenario, this enables the caching entity to prioritize certain files when selecting the cache content.

III. SYSTEM MODEL

A. Wireless Local Caching Entity

We consider a wireless local caching entity that can either be an SBS equipped with a cache 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 file library F containing

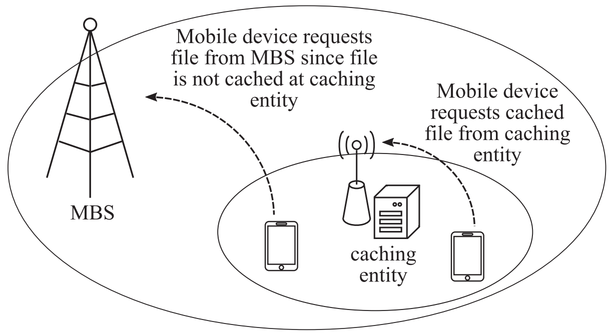


Fig. 1. System model.

TABLE II

EXAMPLES OF CONTEXT DIMENSIONS

Class	Context Dimension
personal characteristics	demographic factors personality mood
user equipment	type of device device capabilities battery status
external factors	location time of day, day of the week events

$|F| \in \mathbb{N}$ files, where we assume for simplicity that all files are of the same size. Users located in the coverage area can connect to the caching entity. The set of currently connected users may change dynamically over time due to the users' mobility. At most $U_{\max} \in \mathbb{N}$ users can be simultaneously connected to the caching entity. To inform connected users about available files, the caching entity periodically broadcasts the information about the current cache content [15]–[17]. If a user is interested in a file that the caching entity stored in its cache, the user's device 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's device does not request the file from the caching entity. Instead, it 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. Hence, the caching entity can only observe requests for cached files, i.e., *cache hits*, but it cannot observe requests for non-cached files, i.e., *cache misses*. Note that this restriction is specific to wireless caching and is usually not used in wired caching scenarios. In this way, the caching entity is not congested by cache misses [15]–[17], but learning content popularity is more difficult. Fig. 1 shows an illustration of the considered system model.

In order to reduce the load on the macro cellular network and the backhaul network, a caching entity might aim at optimizing the cache content such that the traffic it can serve is maximized, which corresponds to maximizing the number of cache hits. For this purpose, the caching entity should learn which files are most popular over time.

B. Service Differentiation

Maximizing the number of cache hits might be an adequate goal of cache content placement in case of an MNO operating an SBS, one reason being net neutrality restrictions. However, the operator of an infostation, e.g., a CP or third party operator, may want to provide differentiated services to its customers (those can be both users and CPs). For example, if users with different preferences are connected to an infostation, the operator can prioritize certain users by caching content favored by these users. In this case, a cache hit by a prioritized user is associated with a higher value than a cache hit by a regular user. For this purpose, we consider a finite set S of service types. For service type $s \in S$, let $v_s \geq 1$ denote a fixed and known weight associated with receiving one cache hit by a user of service type s. Let $v_{\max} := \max_{s \in S} v_s$. The weights might be selected based on a pricing policy, e.g., by paying a monthly fee, a user can buy a higher weight. Alternatively, the weights might be selected based on a subscription policy, e.g., subscribers might obtain priority compared to one-time users. Yet another prioritization might be based on the importance of users in terms of advertisement or their influence on the operator's reputation. Finally, prioritization could be based on usage patterns, e.g., users might indicate their degree of openness in exploring other than their most preferred content. Taking into account the service weights, the caching entity's goal becomes to maximize the number of *weighted* cache hits. Clearly, the above service differentiation only takes effect if users with different preferences are present, i.e., if content popularity is heterogeneous across the user population.

Another service differentiation can be applied in case of a third party operator whose customers are different CPs. The operator may want to prioritize certain CPs by caching their content. In this case, each content is associated with a weight. Here, we consider a fixed and known prioritization weight $w_f \geq 1$ for each file $f \in F$ and let $w_{\max} := \max_{f \in F} w_f$. The prioritization weights can either be chosen individually for each file or per CP.

The case without service differentiation, where the goal is to maximize the number of (non-weighted) cache hits, is a special case, 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 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 may vary across a user population since different users may prefer different content. A user's preferences might be linked to various factors. We refer to such factors as *context dimensions* and give some examples in Table II. Relevant *personal characteristics* may, for example, be demographic factors (e.g., age, gender), personality, or mood. In addition, a user's preferences may be influenced by

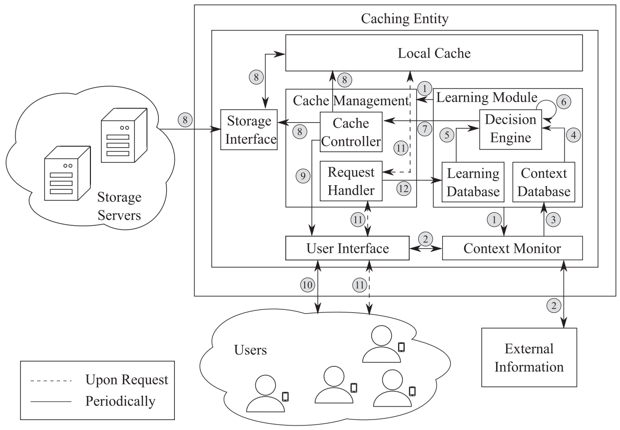


Fig. 2. Context-aware proactive caching architecture.

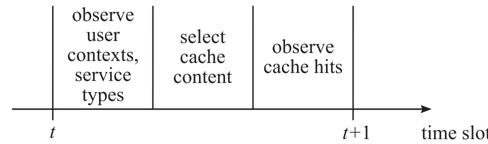


Fig. 3. Sequence of operations of context-aware proactive caching in time slot t.

user equipment, such as the type of device used to access and consume the content (e.g., smart phone, tablet), as well as its capabilities, or its battery status. Besides, *external factors* may have an impact on a user's preferences, such as the user's location, the time of day, the day of the week, and the taking place of events (e.g., soccer match, concert). Clearly, this categorization is not exhaustive and the impact of each single context dimension on content popularity is unknown a priori. Moreover, a caching entity may only have access to some of the context dimensions, e.g., due to privacy reasons. However, our model *does not* rely on *specific* context dimensions; it can use the information that *is* collected from the user. If the caching entity does have access to some relevant context dimensions, these can be exploited to learn context-specific content popularity.

D. Context-Aware Proactive Caching Architecture

Next, we describe the architecture for context-aware proactive caching, which is designed similarly to an architecture presented in [33]. An illustration of the context-aware proactive caching architecture is given in Fig. 2. Its main building blocks are the *Local Cache*, a *Cache Management* entity, a *Learning Module*, a *Storage Interface*, a *User Interface*, and a *Context Monitor*. 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 Fig. 2 and is described below.

- Initialization
  (1) The Learning Module is provided with the goal of caching (i.e., maximize number of cache hits or achieve operator-specific goal). It fixes the appropriate periodicity of context monitoring and cache refreshment. Then, it 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 information about currently connected users available at the User Interface and optionally by collecting additional information from external sources (e.g., 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 results from previous time slots as stored in the Learning Database, (6) the Decision Engine decides which files to cache in the coming time slot. (7) The Decision Engine instructs the Cache Controller to refresh the cache content accordingly. (8) The Cache Controller compares the current and the required cache content 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 new cache content to currently connected users.
- User Requests
  (11) When a user requests a cached file, the User Interface forwards the request to the Request Handler. The Request Handler stores the request information, retrieves the 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 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. The caching system operates in discrete time slots $t = 1, 2, ..., T$, where T denotes the finite time horizon. As illustrated in Fig. 3, each time slot t consists of the following sequence of operations: (i) The context of currently connected users and their service types are monitored. Let U_t be the number of currently connected users. We assume that $1 \leq U_t \leq U_{\max}$ and we specifically allow the set of currently connected users to change in between the time slots of the algorithm, so that user mobility is taken into account. 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 $\mathbf{x_t} = (x_{t,i})_{i=1,\ldots,U_t}$ 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 $\mathbf{s_t} = (s_{t,i})_{i=1,\ldots,U_t}$ be the collection of service types of all users in time slot t. (ii) The cache content is refreshed based on the contexts $\mathbf{x_t}$, the service types $\mathbf{s_t}$ and their service weights, the file prioritization weights w_f, $f \in F$, and knowledge from previous time slots. Then, connected users are informed about the current cache content, which is denoted by $C_t = \{c_{t,1}, \ldots, c_{t,m}\}$. (iii) Until the end of the time slot, users can request currently cached files. Their requests are served. The demand $d_{c_{t,j}}(x_{t,i}, t)$ of each user $i = 1, \ldots, U_t$ for each cached file $c_{t,j} \in C_t$, $j = 1, \ldots, m$, in this time slot is observed, i.e., the number of cache hits for each cached file is monitored.

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 $\mu_f(x) := E(d_f(x))$. The random demand is assumed to take values in $[0, R_{\max}]$, where $R_{\max} \in \mathbb{N}$ is the maximum possible number of requests a user can submit within one time slot. This explicitly incorporates that a user may request the same file repeatedly within one time slot. In time slot t, the random variables $(d_f(x_{t,i}))_{i=1,..,U_t, f \in F}$, are assumed to be independent, i.e., the requests of currently connected users and between different files are independent of each other. Moreover, each random variable $d_f(x_{t,i})$ is assumed to be independent of past caching decisions and previous demands.

The goal of the caching entity is to select the cache content in order to maximize the expected cumulative number of (weighted) cache hits up to the finite time horizon T. We introduce a binary variable $y_{t,f}$, which describes if file f is cached in time slot t, where $y_{t,f} = 1$, if $f \in C_t$, and 0 otherwise. Then, the problem of cache content placement can be formally written as

$$\begin{aligned} \max \quad & \sum_{t=1}^{T} \sum_{f \in F} y_{t,f} w_f \sum_{i=1}^{U_t} v_{s_{t,i}} \mu_f(x_{t,i}) \\ \text{s.t.} \quad & \sum_{f \in F} y_{t,f} \le m, \quad t = 1, \ldots, T, \\ & y_{t,f} \in \{0, 1\}, \quad f \in F, \ t = 1, \ldots, T. \end{aligned} \tag{1}$$

Let us now first assume that the caching entity *had* a priori knowledge about context-specific content popularity *like an omniscient oracle*, i.e., suppose that for each context vector $x \in \mathcal{X}$ and for each file $f \in F$, the caching entity would know the expected demand $\mu_f(x) = E(d_f(x))$. In this case, problem (1) corresponds to an integer linear programming problem. The problem can be decoupled into T independent sub-problems, one for each time slot t. Each sub-problem is a special case of the knapsack problem [43] with a knapsack of capacity m and with items of non-negative profit and unit weights. Hence, its optimal solution can be easily computed in a running time of $O(|F| \log(|F|))$ as follows. In time slot t, given the contexts $\mathbf{x_t}$ and the service types $\mathbf{s_t}$, the optimal solution is given by ranking the files according to their (weighted) expected demands and by selecting the m highest ranked files. We denote these *top-m files for pair $(\mathbf{x_t}, \mathbf{s_t})$* by $f_1^*(\mathbf{x_t}, \mathbf{s_t}), f_2^*(\mathbf{x_t}, \mathbf{s_t}), \ldots, f_m^*(\mathbf{x_t}, \mathbf{s_t}) \in F$. Formally, for $j = 1, \ldots, m$, they satisfy [1]

$$f_j^*(\mathbf{x_t}, \mathbf{s_t}) \in \underset{f \in F \setminus (\bigcup_{k=1}^{j-1} \{f_k^*(\mathbf{x_t}, \mathbf{s_t})\})}{\operatorname{argmax}} \quad w_f \sum_{i=1}^{U_t} v_{s_{t,i}} \mu_f(x_{t,i}), \tag{2}$$

where $\bigcup_{k=1}^{0} \{f_k^*(\mathbf{x_t}, \mathbf{s_t})\} := \emptyset$. We denote by $C_t^*(\mathbf{x_t}, \mathbf{s_t}) := \bigcup_{k=1}^{m} \{f_k^*(\mathbf{x_t}, \mathbf{s_t})\}$ an optimal choice of files to cache in time slot t. Consequently, the collection

$$(C_t^*(\mathbf{x_t}, \mathbf{s_t}))_{t=1,\ldots,T} \tag{3}$$

is an optimal solution to problem (1). Since this solution can be achieved by an omniscient oracle under a priori knowledge about content popularity, we call it the *oracle solution*.

However, in this paper we assume that the caching entity *does not* have a priori knowledge about content popularity. In this case, the caching entity cannot simply solve problem (1) as described above, since the expected demands $\mu_f(x) = E(d_f(x))$ are unknown. Hence, the caching entity has to learn these expected demands over time by observing the users' demands for cached files given the users' contexts. For this purpose, over time, the caching entity has to find a trade-off between caching files about which little information is available (*exploration*) and files of which it believes that they will yield the highest demands (*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 *learning algorithm*. The oracle solution given in (3) can be used as a benchmark to evaluate the loss of learning. Formally, the *regret* of learning with respect to the oracle solution is given by

$$\begin{aligned} R(T) = \sum_{t=1}^{T} \sum_{j=1}^{m} \sum_{i=1}^{U_t} v_{s_{t,i}} \Big(& w_{f_j^*(\mathbf{x_t}, \mathbf{s_t})} E\left(d_{f_j^*(\mathbf{x_t}, \mathbf{s_t})}(x_{t,i})\right) \\ & - E\left(w_{c_{t,j}} d_{c_{t,j}}(x_{t,i}, t)\right)\Big), \end{aligned} \tag{4}$$

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

In order to proactively cache the most suitable files given the context information about currently connected users, the caching entity should learn context-specific content popularity. Due to the above formal problem formulation, this problem corresponds to a contextual multi-armed bandit problem and we can adapt and extend a contextual learning algorithm [41], [42] to our setting. Our algorithm is based

[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.

on the assumption that users with similar context information will 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 starts by partitioning the context space uniformly into smaller sets, i.e., it splits the context space into parts of similar contexts. Then, the caching entity learns the content popularity independently in each of these sets of similar contexts. The algorithm operates in discrete time slots. In each time slot, the algorithm first observes the contexts of currently connected users. Then, the algorithm selects which files to cache in this time slot. Based on a certain control function, the algorithm is either in an exploration phase, in which it chooses a random set of files to cache. Theses phases are needed to learn the popularity of files which have not been cached often before. Otherwise, the algorithm is in an exploitation phase, in which it caches files which on average were requested most when cached in previous time slots with similar user contexts. After caching the new set of files, the algorithm observes the users' requests for these files. In this way, over time, the algorithm learns context-specific content popularity.

The algorithm for selecting m files is called *Context-Aware Proactive Caching with Cache Size m* (m-CAC) and its pseudocode is given in Fig. 4. 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., if 2 users with context from set p are connected in one time slot and file f is cached, this counter is increased by 2). Moreover, m-CAC keeps the estimated demand $\hat{d}_{f,p}(t)$ up to time slot t of each pair consisting of a file $f \in F$ and a set $p \in \mathcal{P}_T$. 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 up to time slot t. Then, the estimated demand of file f in set p is given by the sample mean $\hat{d}_{f,p}(t) := \frac{1}{|\mathcal{E}_{f,p}(t)|} \sum_{d \in \mathcal{E}_{f,p}(t)} d$.[2,3]

In each time slot t, m-CAC first observes the number of currently connected users U_t, their contexts $\mathbf{x_t} = (x_{t,i})_{i=1,\ldots,U_t}$ and the service types $\mathbf{s_t} = (s_{t,i})_{i=1,\ldots,U_t}$. For each context vector $x_{t,i}$, m-CAC determines the set $p_{t,i} \in \mathcal{P}_T$, to which the context vector belongs, i.e., such that $x_{t,i} \in p_{t,i}$ holds. The collection of these sets is given by $\mathbf{p_t} = (p_{t,i})_{i=1,\ldots,U_t}$. Then, the algorithm can either be in an exploration phase or in an exploitation phase. In order to determine the correct phase in 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* $F^{\text{ue}}_{\mathbf{p_t}}(t)$ is calculated based on

$$\begin{aligned} F^{\text{ue}}_{\mathbf{p_t}}(t) &:= \cup_{i=1}^{U_t} F^{\text{ue}}_{p_{t,i}}(t) \\ &:= \cup_{i=1}^{U_t} \{f \in F : N_{f,p_{t,i}}(t) \leq K(t)\}, \end{aligned} \tag{5}$$

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 V, 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 $F^{\text{ue}}_{\mathbf{p_t}}(t)$ to cache. If the set of under-explored files contains less than m elements, i.e., $u(t) < m$, it

m-CAC: Context-Aware Proactive Caching Algorithm

Input: T, h_T, $K(t)$
Initialize context partition: Create partition $\mathcal{P}_T$ of context space $[0,1]^D$ into $(h_T)^D$ hypercubes of identical size
Initialize counters: For all $f \in F$ and all $p \in \mathcal{P}_T$, set $N_{f,p} = 0$
Initialize estimated demands: For all $f \in F$ and all $p \in \mathcal{P}_T$, set $\hat{d}_{f,p} = 0$
for each $t = 1, ..., T$ **do**
  Observe number U_t of currently connected users
  Observe user contexts $\mathbf{x_t} = (x_{t,i})_{i=1,...,U_t}$ and service types $\mathbf{s_t} = (s_{t,i})_{i=1,...,U_t}$
  Find $\mathbf{p_t} = (p_{t,i})_{i=1,...,U_t}$ such that $x_{t,i} \in p_{t,i} \in \mathcal{P}_T, i = 1, ..., U_t$
  Compute the set of under-explored files $F^{\text{ue}}_{\mathbf{p_t}}(t)$ in (5)
  if $F^{\text{ue}}_{\mathbf{p_t}}(t) \neq \emptyset$ **then** ▷ Exploration
    $u = \text{size}(F^{\text{ue}}_{\mathbf{p_t}}(t))$
    if $u \geq m$ **then**
      Select $c_{t,1}, ..., c_{t,m}$ randomly from $F^{\text{ue}}_{\mathbf{p_t}}(t)$
    else
      Select $c_{t,1}, ..., c_{t,u}$ as the u files from $F^{\text{ue}}_{\mathbf{p_t}}(t)$
      Select $c_{t,u+1}, ..., c_{t,m}$ as the $(m-u)$ files $\hat{f}_{1,\mathbf{p_t},\mathbf{s_t}}(t), ..., \hat{f}_{m-u,\mathbf{p_t},\mathbf{s_t}}(t)$ from (6)
    end if
  else ▷ Exploitation
    Select $c_{t,1}, ..., c_{t,m}$ as the m files $\hat{f}_{1,\mathbf{p_t},\mathbf{s_t}}(t), ..., \hat{f}_{m,\mathbf{p_t},\mathbf{s_t}}(t)$ from (7)
  end if
  Observe demand $(d_{j,i})$ of each user $i = 1, ..., U_t$ for each file $c_{t,j}, j = 1, ..., m$
  for $i = 1, ..., U_t$ **do**
    for $j = 1, ..., m$ **do**
      $\hat{d}_{c_{t,j},p_{t,i}} = \frac{\hat{d}_{c_{t,j},p_{t,i}} N_{c_{t,j},p_{t,i}} + d_{j,i}}{N_{c_{t,j},p_{t,i}} + 1}$ and $N_{c_{t,j},p_{t,i}} = N_{c_{t,j},p_{t,i}} + 1$
    end for
  end for
end for

Fig. 4. Pseudocode of m-CAC.

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

[3]Note that in the pseudocode in Fig. 4, 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.

selects all $u(t)$ files from $F^{\mathrm{ue}}_{\mathbf{p_t}}(t)$ to cache. Since the cache is not fully filled by $u(t) < m$ files, $(m - u(t))$ additional files can be cached. In order to exploit knowledge obtained so far, m-CAC selects $(m - u(t))$ files from $F \setminus F^{\mathrm{ue}}_{\mathbf{p_t}}(t)$ based on a file ranking according to the estimated weighted demands, as defined by the files $\hat{f}_{1,\mathbf{p_t},\mathbf{s_t}}(t), \ldots, \hat{f}_{m-u(t),\mathbf{p_t},\mathbf{s_t}}(t) \in F \setminus F^{\mathrm{ue}}_{\mathbf{p_t}}(t)$, which satisfy for $j = 1, \ldots, m - u(t)$:

$$\hat{f}_{j,\mathbf{p_t},\mathbf{s_t}}(t) \in \underset{f \in F \setminus (F^{\mathrm{ue}}_{\mathbf{p_t}}(t) \cup \bigcup_{k=1}^{j-1} \{\hat{f}_{k,\mathbf{p_t},\mathbf{s_t}}(t)\})}{\operatorname{argmax}} w_f \sum_{i=1}^{U_t} v_{s_{t,i}} \hat{d}_{f,p_{t,i}}(t). \tag{6}$$

If the set of files defined by (6) 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 $F^{\mathrm{ue}}_{\mathbf{p_t}}(t)$ 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 $\hat{f}_{1,\mathbf{p_t},\mathbf{s_t}}(t), \ldots, \hat{f}_{m,\mathbf{p_t},\mathbf{s_t}}(t) \in F$, which satisfy for $j = 1, \ldots, m$:

$$\hat{f}_{j,\mathbf{p_t},\mathbf{s_t}}(t) \in \underset{f \in F \setminus \left(\bigcup_{k=1}^{j-1} \{\hat{f}_{k,\mathbf{p_t},\mathbf{s_t}}(t)\}\right)}{\operatorname{argmax}} w_f \sum_{i=1}^{U_t} v_{s_{t,i}} \hat{d}_{f,p_{t,i}}(t). \tag{7}$$

If the set of files defined by (7) is not unique, ties are again broken arbitrarily.

After selecting the subset of files to cache, the algorithm observes the users' requests for these files in this time slot. Then, it updates the estimated demands and the counters of cached files.

V. Analysis of the Regret

In this section, we give an upper bound on the regret $R(T)$ of m-CAC in (4). The regret bound is based on the natural assumption that expected demands for files are similar in similar contexts, i.e., that users with similar characteristics are likely to consume similar content. This assumption is realistic since the users' preferences in terms of consumed content differ based on the users' contexts, so that it is plausible to divide the user population into segments of users with similar context and similar preferences. Formally, the similarity assumption is 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

$$|\mu_f(x) - \mu_f(y)| \le L||x - y||^{\alpha},$$

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

Assumption 1 is needed for the analysis of the regret, but it should be noted that m-CAC can also be applied if this assumption does not hold true. However, a regret bound might not be guaranteed in this case.

The following theorem 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 m-CAC converges to the oracle solution strategy. In other words, m-CAC converges to the optimal cache content placement strategy, which maximizes the expected number of cache hits. 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 $R(T)$ is $O\left(v_{\max} w_{\max} m U_{\max} R_{\max} |F| T^{\frac{2\alpha+D}{3\alpha+D}} \log(T)\right)$.

The proof can be found in our online appendix [44]. The regret bound given in Theorem 1 is sublinear in the time horizon T, proving that m-CAC converges to the optimal cache content placement strategy. 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 placement phases. Thus, Theorem 1 characterizes m-CAC's speed of convergence Furthermore, Theorem 1 shows that the regret bound is a constant multiple of the regret bound in the special case without service differentiation, in which $v_{\max} = 1$ and $w_{\max} = 1$. Hence, the order of the regret is $O\left(T^{\frac{2\alpha+D}{3\alpha+D}} \log(T)\right)$ in the special case as well.

VI. Memory Requirements

The memory requirements of m-CAC are mainly determined by the counters kept by the algorithm during its runtime (see also [41]). For each set p in the partition $\mathcal{P}_T$ and each file $f \in F$, the algorithm keeps the counters $N_{f,p}$ and $\hat{d}_{f,p}$. The number of files is $|F|$. If m-CAC runs with the parameters from Theorem 1, the number of sets in $\mathcal{P}_T$ is upper bounded by $(h_T)^D = \lceil T^{\frac{1}{3\alpha+D}} \rceil^D \le 2^D T^{\frac{D}{3\alpha+D}}$. Hence, the required memory is upper bounded by $|F| 2^D T^{\frac{D}{3\alpha+D}}$ and is thus sublinear in the time horizon T. This means, that for $T \to \infty$, the algorithm would require 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 connected users' context vectors, the required number of counters that have to be kept can be much smaller than given by the upper bound.

VII. Extensions

A. Exploiting the Multicast Gain

So far, we assumed that each request for a cached file is immediately served by a unicast transmission. However, our algorithm can be extended to multicasting, which has been shown to be beneficial in combination with caching [7], [11]. For this purpose, to extend our algorithm, each time slot t is divided into a fixed number of intervals. In each interval, incoming requests are monitored and accumulated. At the end of the interval, requests for the same file are served by a multicast transmission. In order to exploit knowledge about content popularity learned so far, a request for a file with low estimated demand could, however, still be served by a unicast transmission. In this way, unnecessary delays are prevented in cases in which another request and thus a multicast transmission are not expected. Moreover, service differentiation could be taken into account. For example, high-priority users could be served by unicast transmissions, such that their delay is not increased due to waiting times for multicast transmissions.

B. Rating-Based Context-Aware Proactive Caching

So far, we considered cache content placement with respect to the demands $d_f(x)$ in order to maximize the number of (weighted) cache hits. However, a CP operating an infostation might want to cache not only content that is requested often, but which also receives high ratings from the users. Consider the case that after consumption users rate content in a range $[r_{\min}, r_{\max}] \subset \mathbb{R}_+$. 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

$$\tilde{d}_f(x) := r_f(x) d_f(x), \tag{8}$$

which combines the demand and the rating for file f of a user with context x. By carefully designing the range of ratings, the CP chooses the trade-off between ratings and cache hits. Now, we can apply m-CAC with respect to $\tilde{d}_f(x)$. In this case, m-CAC additionally needs to observe the users' ratings in order to learn content popularity in terms of ratings. If the users' ratings are always available, Theorem 1 applies and provides a regret bound of $O\left(v_{\max} w_{\max} r_{\max} m U_{\max} R_{\max} |F| T^{\frac{2\alpha+D}{3\alpha+D}} \log(T)\right)$.

However, users might not always reveal a rating after consuming a content. When a user's rating is missing, we assume that m-CAC does not update the counters based on this user's request. This may result in a higher required number of exploration phases. Hence, the regret of the learning algorithm is influenced by the users' willingness to reveal ratings of requested content. Let $q \in (0, 1)$ be the probability that a user reveals his rating after requesting a file. Then, the regret of the learning algorithm is bounded as given below.

Theorem 2 (Bound for $R(T)$ for Rating-Based Caching With Missing Ratings): 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 with respect to $\tilde{d}_f(x)$, Assumption 1 holds true for $\tilde{d}_f(x)$ and a user reveals his rating with probability q, the leading order of the regret $R(T)$ is $O\left(\frac{1}{q} v_{\max} w_{\max} r_{\max} m U_{\max} R_{\max} |F| T^{\frac{2\alpha+D}{3\alpha+D}} \log(T)\right)$.

The proof can be found in our online appendix [44]. Comparing Theorem 2 with Theorem 1, the regret of m-CAC is scaled up by a factor $\frac{1}{q} > 1$ in case of rating-based caching with missing ratings. This factor corresponds to the expected number of requests until the caching entity receives one rating. However, the time order of the regret remains the same. Hence, m-CAC is robust under missing ratings in the sense that if some users refuse to rate requested content, the algorithm still converges to the optimal cache content placement strategy.

C. Asynchronous User Arrival

So far, we assumed that the set of currently connected users only changes in between the time slots of our algorithm. This means, that only those users connected to the caching entity at the beginning of a time slot, will request files within that time slot. However, if users connect to the caching entity asynchronously, m-CAC should be adapted. If a user directly disconnects after the context monitoring without requesting any file, he should be excluded from learning. Hence, in m-CAC, the counters are not updated for disconnecting users. If a user connects to the caching entity after cache content placement, his context was not considered in the caching decision. However, his requests can be used to learn faster. Hence, in m-CAC, the counters are updated based on this user's requests.

D. Multiple Wireless Local Caching Entities

So far, we considered online learning for cache content placement in a single caching entity. However, real caching systems contain multiple caching entities, each of which should learn local content popularity. In a network of multiple caching entities, m-CAC could be applied separately and independently by each caching entity. For the case that coverage areas of caching entities overlap, in this subsection, we present m-CACao, an extension of m-CAC to *Context-Aware Proactive Caching with Area Overlap*. The idea of m-CACao is that caching entities can learn content popularity faster by not only relying on their own cache hits, but also on cache hits at neighboring caching entities with overlapping coverage area. For this purpose, the caching entities overhear cache hits produced by users in the intersection to neighboring coverage areas.

In detail, m-CAC is extended to m-CACao as follows: The context monitoring and the selection of cache content works as in m-CAC. However, the caching entity not only observes its own cache hits (line 21 in Fig. 4), but it overhears cache hits at neighboring caching entities of users in the intersection. Then, the caching entity not only updates the counters of its own cached files (lines 22-26 in Fig. 4), but it additionally updates the counters of files of which it overheard cache hits at neighboring caches. This helps the caching entity to learn faster.

VIII. NUMERICAL RESULTS

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

A. Description of the Data Set

We use a data set from MovieLens [45] to evaluate our proposed algorithm. MovieLens is an online movie recommender operated by the research group GroupLens from the University of Minnesota. The MovieLens 1M DataSet [46] 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

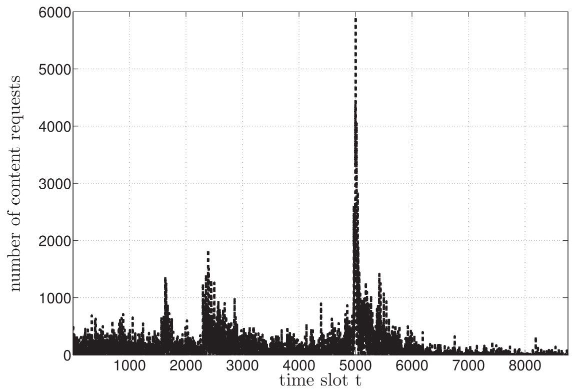


Fig. 5. Number of content requests in used data set as a function of time slots. Time slots at an hourly basis.

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 wireless local caching entity (see [33], [34] 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 caching entity (in case the movie is cached in the caching entity) or to the macro cellular network (in case the movie is not cached in the caching entity). This approach is reasonable since users typically rate movies after watching them.

In our simulations, we only use the data gathered within the first year of the data set, since around 94% of the ratings were provided within this time frame. 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. Then, we assign the requests and corresponding user contexts to the time slots according to their timestamps and we interpret each request as if it was coming from a separate user. At the beginning of a time slot, we assume to have access to the context of each user responsible for a request in the coming time slot. Fig. 5 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.[4]

B. Reference Algorithms

We compare m-CAC with five reference algorithms. The first algorithm is the omniscient Oracle, which has complete knowledge about the exact future demands. In each time slot, the oracle selects the optimal m files that will maximize the number of cache hits in this time slot.[5]

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 [35], 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 time slot, we modify UCB to take m actions at a time, which corresponds to selecting m files.

The third reference algorithm is the m-ϵ-Greedy. This is a variant of the simple ϵ-Greedy [35] algorithm, which does not consider context information. The m-ϵ-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 fourth reference algorithm is called m-Myopic. This is an algorithm taken from [15], which is investigated since it is comparable to the well-known Least Recently Used algorithm (LRU) for caching. m-Myopic only learns from one time slot in the past. It starts with a random set of files and in each of the following time slots discards all files that have not been requested in the previous time slot. Then, it randomly replaces the discarded files by other files.

The fifth reference algorithm, called Random, caches a random set of files in each time slot.

C. Performance Measures

The following performance measures are used in our analysis. The evolution of per-time slot or cumulative *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 requests which can be served by cached files.

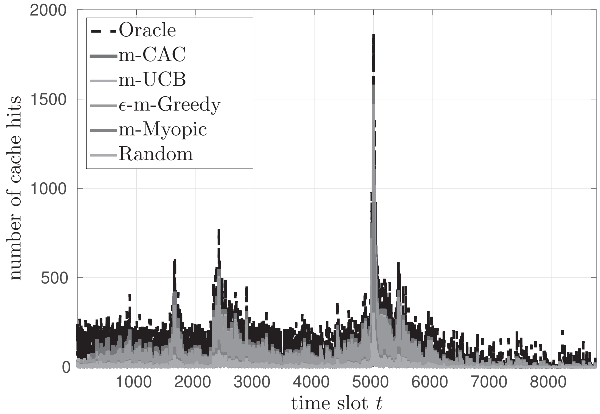
D. Results

In our simulations, we set $\epsilon = 0.09$ in m-ϵ-Greedy, which is the value at which heuristically the algorithm on average performed best. In m-CAC, we set the control function to $K(t) = c \cdot t^{\frac{2\alpha}{3\alpha+D}} \log(t)$ with $c = 1/(|F|D)$.[6] The simulation results are obtained by averaging over 100 runs of the algorithms. First, we consider the case without service differentiation. The long-term behavior of m-CAC 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 file library size. We run all algorithms on the data set and study their results as a function of time, i.e., over the time slots $t = 1, ..., T$. Fig. 6(a) and 6(b) show the per-time slot and the cumulative numbers of cache hits up to time slot t as a function of time, respectively. Due to the bursty content request process (compare Fig. 5), also the number of cache hits achieved by the different algorithms is bursty over time. As expected, the Oracle gives an upper bound to the other algorithms. Among the other algorithms, m-CAC, m-ϵ-Greedy and m-UCB clearly

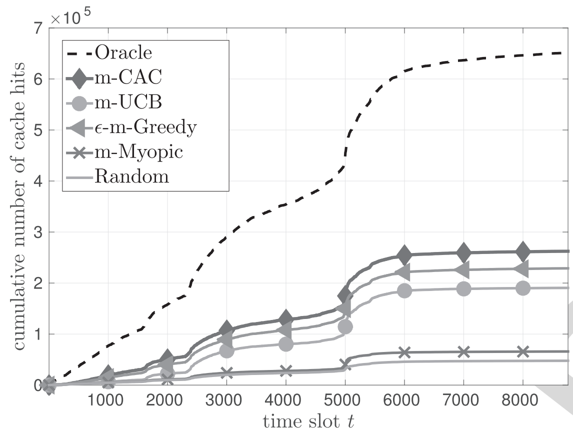
[4] We neglect the occupation as context dimension since by mapping them to a [0,1] variable, we would have to classify which occupations are more and which are less similar to each other.

[5] Note that this oracle yields even better results than the oracle used as a benchmark to define the regret in (4). In the definition of regret, the oracle only exploits knowledge about expected demands, instead of exact future demands.

[6] Compared to the control function in Theorem 1, the additional factor reduces the number of exploration phases which allows for better performance.



(a) Number of cache hits per time slot.



(b) Cumulative number of cache hits.

Fig. 6. Time evolution of algorithms for $m = 200$.

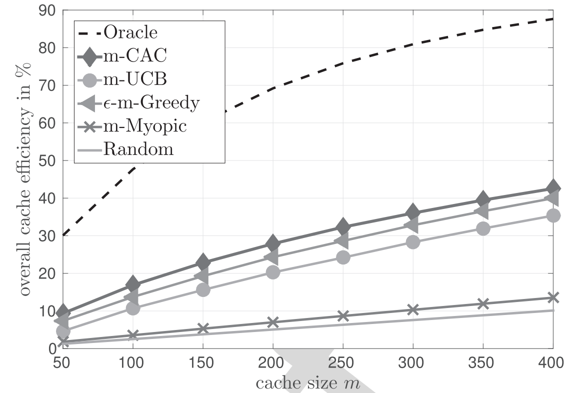


Fig. 7. Overall cache efficiency at T as a function of cache size m.

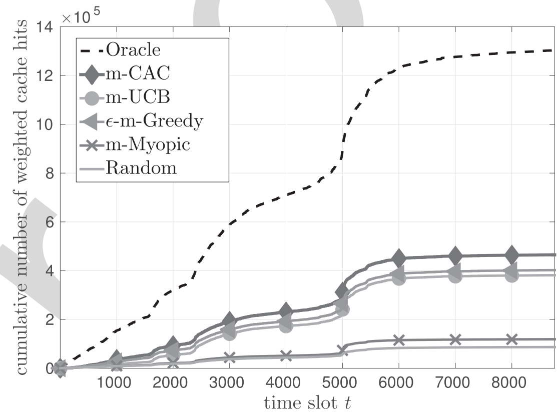


Fig. 8. Cumulative number of weighted cache hits for $m = 200$ as a function of time.

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-ϵ-Greedy shows a better performance than m-UCB, even though it uses a simpler learning strategy. Overall, m-CAC outperforms the other algorithms by additionally learning from context information. At the time horizon, the cumulative number of cache hits achieved by m-CAC is 1.146, 1.377, 3.985 and 5.506 times higher than the ones achieved by m-ϵ-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, which corresponds to between 1.3% and 10.1% of the file library size, which is a realistic assumption. All remaining parameters are kept as before. Fig. 7 shows the overall cache efficiency achieved at the time horizon T as a function of cache size, i.e., the cumulative number of cache hits up to T is normalized by the cumulative number of requests up to T. The overall cache efficiency of all algorithms is increasing with increasing cache size. Moreover, the results indicate that again m-CAC and m-ϵ-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 28.4%, compared to an average cache efficiency of 25.3%, 21.4%, 7.76% and 5.69% achieved by m-ϵ-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 with weights $v_1 = 5$ and $v_2 = 1$ exist. 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 each algorithm to take into account service differentiation by incorporating the weights according to the service types. Fig. 8 shows the cumulative number of weighted cache hits up to time slot t as a function of time. At the time horizon, the cumulative number of weighted cache hits achieved by m-CAC is 1.156, 1.219, 3.914 and 5.362 times higher than the ones achieved by m-ϵ-Greedy, m-UCB, m-Myopic and Random, respectively. A comparison with Fig. 6(b) shows that the behavior of the algorithms is similar to the case without service differentiation.

Finally, we investigate the extension to multiple caching entities and compare the performance of the proposed algorithms m-CAC and m-CACao. We consider a scenario with two caching entities and divide the data set as follows: A fraction $o \in [0, 0.3]$ of randomly selected requests is considered to be made in the intersection of the two coverage areas. We use the parameter o as a measure of the overlap between the caching entities. The remaining requests are randomly assigned to either one of the caching entities.

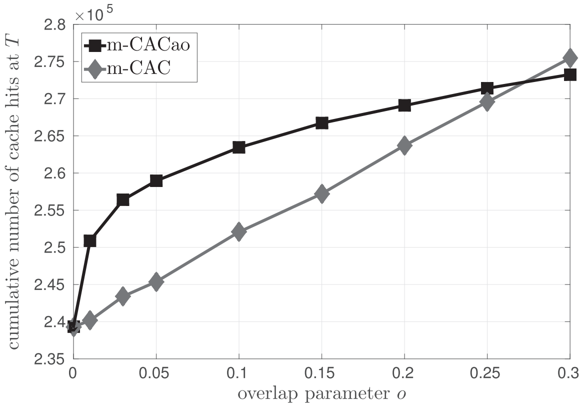


Fig. 9. Cumulative number of cache hits at T as a function of the overlap parameter o.

These requests are considered to be made by users solely connected to one caching entity. Then, on the one hand we run m-CAC separately on each caching entity and on the other hand we run m-CACao on both caching entities. Fig. 9 shows the cumulative number of cache hits achieved in sum by the two caching entities at the time horizon T as a function of the overlap parameter o. As expected, m-CAC and m-CACao perform identically for non-overlapping coverage areas. With increasing overlap, the number of cache hits achieved by both m-CAC and m-CACao increases. The reason is that users in the intersection can more likely be served since they have access to both caches. Hence, even though the caching entities do not coordinate their cache content, more cache hits occur. For up to 25% of overlap ($o \leq 0.25$), m-CACao outperforms m-CAC. Clearly, m-CACao performs better since by overhearing cache hits at the neighboring caching entity, both caching entities learn content popularity faster. For very large overlap ($o > 0.25$), m-CAC yields higher numbers of cache hits. The reason is that when applying m-CACao in case of a large overlap, neighboring caching entities overhear such a large number of cache hits, that they learn very similar content popularity distributions. Hence, over time it is likely that their caches contain the same files. In contrast, applying m-CAC, a higher diversity in cache content is maintained over time. Clearly, further gains in cache hits could be achieved by jointly optimizing the cache content of all caching entities. However, this would either require coordination among the caching entities or a central planner deciding on the cache content of all caching entities, which results in a high communication overhead. In contrast, our heuristic algorithm m-CACao does not require additional coordination or communication and yields good results for small overlaps.

IX. Conclusion

In this paper, we presented a context-aware proactive caching algorithm for wireless caching entities based on contextual multi-armed bandits. To cope with unknown and fluctuating content popularity among the dynamically arriving and leaving users, the algorithm regularly observes context information of connected users, updates the cache content and subsequently observes cache hits. In this way, the algorithm learns context-specific content popularity online, which allows for a proactive adaptation of cache content according to fluctuating local content popularity. We derived a sublinear regret bound, which characterizes the learning speed and proves that our proposed algorithm converges to the optimal cache content placement strategy, which maximizes the expected number of cache hits. Moreover, the algorithm supports customer prioritization and can be combined with multicast transmissions and rating-based caching decisions. Numerical studies showed that by exploiting context information, our algorithm outperforms state-of-the-art algorithms in a real world data set.

References

[1] 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 Int. Conf. Commun. (ICC)*, May 2016, pp. 1–7.

[2] [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html AQ:4

[3] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. M. Leung, "Cache in the air: Exploiting content caching and delivery techniques for 5G systems," *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 131–139, Feb. 2014.

[4] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Mar. 2010, pp. 1–9.

[5] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *Proc. IEEE Comput. Commun. 18th Annu. Joint Conf. Soc. (INFOCOM)*, vol. 1. Mar. 1999, pp. 126–134.

[6] J. Erman, A. Gerber, M. Hajiaghayi, D. Pei, S. Sen, and O. Spatscheck, "To cache or not to cache: The 3G case," *IEEE Internet Comput.*, vol. 15, no. 2, pp. 27–34, Mar. 2011.

[7] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Trans. Inf. Theory*, vol. 60, no. 5, pp. 2856–2867, May 2014.

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

[9] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "FemtoCaching: Wireless content delivery through distributed caching helpers," *IEEE Trans. Inf. Theory*, vol. 59, no. 12, pp. 8402–8413, Dec. 2013.

[10] K. Poularakis and L. Tassiulas, "Exploiting user mobility for wireless content delivery," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2013, pp. 1017–1021.

[11] K. Poularakis, G. Iosifidis, V. Sourlas, and L. Tassiulas, "Exploiting caching and multicast for 5G wireless networks," *IEEE Trans. Wireless Commun.*, vol. 15, no. 4, pp. 2995–3007, Apr. 2016.

[12] E. Bastug, M. Bennis, and M. Debbah, "Cache-enabled small cell networks: Modeling and tradeoffs," in *Proc. Int. Symp. Wireless Commun. Syst. (ISWCS)*, Aug. 2014, pp. 649–653.

[13] E. Baştuğ, M. Bennis, and M. Debbah, "Living on the edge: The role of proactive caching in 5G wireless networks," *IEEE Commun. Mag.*, vol. 52, no. 8, pp. 82–89, Aug. 2014.

[14] E. Baştuğ *et al.*, "Big data meets telcos: A proactive caching perspective," *J. Commun. Netw.*, vol. 17, no. 6, pp. 549–557, Dec. 2015.

[15] P. Blasco and D. Gündüz, "Learning-based optimization of cache content in a small cell base station," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2014, pp. 1897–1903.

[16] P. Blasco and D. Gündüz, "Multi-armed bandit optimization of cache content in wireless infostation networks," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2014, pp. 51–55.

[17] P. Blasco and D. Gündüz. (Jul. 2014). "Content-level selective offloading in heterogeneous networks: Multi-armed bandit optimization and regret bounds." [Online]. Available: https://arxiv.org/abs/1407.6154

[18] A. Sengupta, S. Amuru, R. Tandon, R. M. Buehrer, and T. C. Clancy, "Learning distributed caching strategies in small cell networks," in *Proc. IEEE Int. Symp. Wireless Commun. Syst. (ISWCS)*, Aug. 2014, pp. 917–921.

[19] M. ElBamby, M. Bennis, W. Saad, and M. Latva-aho, "Content-aware user clustering and caching in wireless small cell networks," in *Proc. IEEE Int. Symp. Wireless Commun. Syst. (ISWCS)*, Aug. 2014, pp. 945–949.

[20] D. Goodman, J. Borras, N. B. Mandayam, and R. D. Yates, "INFOSTATIONS: A new system model for data and messaging services," in *Proc. IEEE 47th Veh. Technol. Conf.*, vol. 2. May 1997, pp. 969–973.

[21] A. L. Iacono and C. Rose, "Infostations: New perspectives on wireless data networks," in *Next Generation Wireless Networks*, S. Tekinay, Ed. Boston, MA, USA: Springer, 2002, pp. 3–63.

[22] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "YouTube traffic characterization: A view from the edge," in *Proc. ACM Conf. Internet Meas. (IMC)*, Oct. 2007, pp. 15–28.

[23] 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.

[24] A. Brodersen, S. Scellato, and M. Wattenhofer, "YouTube around the world: Geographic popularity of videos," in *Proc. ACM Int. Conf. World Wide Web (WWW)*, Apr. 2012, pp. 241–250.

[25] M.-L. Mares and Y. Sun, "The multiple meanings of age for television content preferences," *Human Commun. Res.*, vol. 36, no. 3, pp. 372–396, Jul. 2010.

[26] C. A. Hoffner and K. J. Levine, "Enjoyment of mediated fright and violence: A meta-analysis," *Media Psychol.*, vol. 7, no. 2, pp. 207–237, Nov. 2005.

[27] P. J. Rentfrow, L. R. Goldberg, and R. Zilca, "Listening, watching, and reading: The structure and correlates of entertainment preferences," *J. Personality*, vol. 79, no. 2, pp. 223–258, Apr. 2011.

[28] D. Zillmann, "Mood management through communication choices," *Amer. Behavioral Sci.*, vol. 31, no. 3, pp. 327–340, Jan. 1988.

[29] C. Zhou, Y. Guo, Y. Chen, X. Nie, and W. Zhu, "Characterizing user watching behavior and video quality in mobile devices," in *Proc. IEEE Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2014, pp. 1–6.

[30] B.-J. Ko, K.-W. Lee, K. Amiri, and S. Calo, "Scalable service differentiation in a shared storage cache," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst. (ICDCS)*, May 2003, pp. 184–193.

[31] 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.

[32] P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms," in *Proc. USENIX Symp. Internet Technol. Syst.*, 1997, pp. 193–206.

[33] S. Li, J. Xu, M. van der Schaar, and W. Li, "Popularity-driven content caching," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.

[34] S. Li, J. Xu, M. van der Schaar, and W. Li, "Trend-aware video caching through Online learning," *IEEE Trans. Multimedia*, vol. 18, no. 12, pp. 2503–2516, Dec. 2016.

[35] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, nos. 2–3, pp. 235–256, May 2002.

[36] S. Maghsudi and E. Hossain, "Multi-armed bandits with application to 5G small cells," *IEEE Wireless Commun.*, vol. 23, no. 3, pp. 64–73, Jun. 2016.

[37] S. Amuru, C. Tekin, M. van der Schaar, and R. M. Buehrer, "Jamming bandits—A novel learning method for optimal jamming," *IEEE Trans. Wireless Commun.*, vol. 15, no. 4, pp. 2792–2808, Apr. 2016.

[38] C. Shen, C. Tekin, and M. van der Schaar, "A non-stochastic learning approach to energy efficient mobility management," *IEEE J. Sel. Areas Commun.*, to be published. AQ:5

[39] T. Lu, D. Pal, and M. Pal, "Contextual multi-armed bandits," in *Proc. Int. Conf. Artif. Intell. Statist. (AISTATS)*, 2010, pp. 485–492.

[40] A. Slivkins, "Contextual bandits with similarity information," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 2533–2568, Jan. 2014.

[41] C. Tekin and M. van der Schaar, "Distributed Online learning via cooperative contextual bandits," *IEEE Trans. Signal Process.*, vol. 63, no. 14, pp. 3700–3714, Mar. 2015.

[42] C. Tekin, S. Zhang, and M. van der Schaar, "Distributed Online learning in social recommender systems," *IEEE J. Sel. Topics Signal Process.*, vol. 8, no. 4, pp. 638–652, Aug. 2014.

[43] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Berlin, Germany: Springer, 2004.

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

[45] F. M. Harper and J. A. Konstan, "The MovieLens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 1–19, Dec. 2015.

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

[47] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *J. Amer. Statist. Assoc.*, vol. 58, no. 301, pp. 13–30, Mar. 1963.

[48] E. Chlebus, "An approximate formula for a partial sum of the divergent pp-series," *Appl. Math. Lett.*, vol. 22, no. 5, pp. 732–737, May 2009.

Sabrina Müller (S'15) received the B.Sc. and the M.Sc. degrees in mathematics from the Technische Universität Darmstadt, Germany, in 2012 and 2014, respectively, where she is currently pursuing the Ph.D. degree in electrical engineering. She is a member of the Communications Engineering Laboratory, Technische Universität Darmstadt. Her research interests include machine learning and optimization methods and their applications to wireless networks.

Onur Atan received the B.Sc. degree from Bilkent University, Ankara, Turkey, in 2013 and the M.Sc. degree from the University of California at Los Angeles, Los Angeles, CA, USA, in 2014, where he is currently pursuing the Ph.D. degree in electrical engineering. He received the Best M.Sc. Thesis Award in electrical engineering from the University of California at Los Angeles. His research interests include online learning and multi-armed bandit problems.

Mihaela van der Schaar (M'99–SM'04–F'10) was a Distinguished Lecturer with the Communications Society from 2011 to 2012. She is currently a Chancellor's Professor of Electrical Engineering with the University of California at Los Angeles, Los Angeles, CA, USA. She holds 33 granted U.S. patents. Her research interests include machine learning for decision making, medical informatics and education, online learning, real-time stream mining, network science, engineering economics, social networks, game theory, wireless networks, and multimedia. She was a member of the Editorial Board of the IEEE JOURNAL OF SELECTED TOPICS IN SIGNAL PROCESSING in 2011. She was a recipient of the NSF CAREER Award in 2004, the Best Paper Award from the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY in 2005, the Okawa Foundation Award in 2006, the IBM Faculty Award in 2005, 2007, and 2008, the Most Cited Paper Award from the EURASIP *Journal on Image Communications* in 2006, the Gamenets Conference Best Paper Award in 2011, and the IEEE CIRCUITS AND SYSTEMS Society Darlington Award Best Paper Award in 2011. She received three ISO Awards for her contributions to the MPEG video compression and streaming international standardization activities. She was the Editor-in-Chief of the IEEE TRANSACTIONS ON MULTIMEDIA from 2011 to 2013. AQ:6

Anja Klein (M'96) received the Diploma and Dr.-Ing. (Ph.D.) degrees in electrical engineering from the University of Kaiserslautern, Germany, in 1991 and 1996, respectively. She was involved in the standardization of third generation mobile radio in ETSI and in 3GPP, for instance leading the TDD Group in RAN1 of 3GPP. She was a Vice-President, heading the Development Department and the Systems Engineering Department. In 1996, she joined Siemens AG, Mobile Networks Division, Munich and Berlin. In 2004, she joined the Technische Universität Darmstadt, Germany, as a Full Professor, heading the Communications Engineering Laboratory. She has authored over 290 refereed papers and has AQ:7 contributed to 12 books. She is an inventor and a co-inventor of over 45 patents in the field of mobile radio. Her main research interests are in mobile radio, including interference management, cross-layer design, relaying and multi-hop, computation offloading, smart caching, and energy harvesting. In 1999, AQ:8 she was named the Inventor of the Year by Siemens AG. She is a member of Verband Deutscher Elektrotechniker-Informationstechnische Gesellschaft.

AUTHOR QUERIES

AUTHOR PLEASE ANSWER ALL QUERIES

PLEASE NOTE: We cannot accept new source files as corrections for your paper. If possible, please annotate the PDF proof we have sent you with your corrections and upload it via the Author Gateway. Alternatively, you may send us your corrections in list format. You may also upload revised graphics via the Author Gateway.

AQ:1 = Please be advised that per instructions from the Communications Society this proof was formatted in Times Roman font and therefore some of the fonts will appear different from the fonts in your originally submitted manuscript. For instance, the math calligraphy font may appear different due to usage of the usepackage[mathcal]euscript. We are no longer permitted to use Computer Modern fonts.

AQ:2 = In the provided source PDF, 48 references were provided, whereas in the source tex, only 46 were provided. We have followed the source tex in this case. Please confirm.

AQ:3 = Note that if you require corrections/changes to tables or figures, you must supply the revised files, as these items are not edited for you.

AQ:4 = Please provide the accessed date for refs. [2], [44], and [46].

AQ:5 = Please provide the volume no., issue no., page range, month, and year for ref. [38].

AQ:6 = Please provide the author photo for “Mihaela van der Schaar.”

AQ:7 = Please confirm whether the edits made in the sentence “She is an...”

AQ:8 = Please confirm whether the retention of the sentence “In 1999, she... Gesellschaft.”