Published in: IEEE Global Communications Conference (GLOBECOM), 2016.

Content Placement and Service Scheduling in Femtocell Caching Networks

Teng Liu and Alhussein A. Abouzeid
Department of Electrical, Computer and Systems Engineering
Rensselaer Polytechnic Institute
Troy, NY 12180-3590, USA
Email: liut7@rpi.edu, abouzeid@ecse.rpi.edu

***Abstract*—This work considers the joint problem of content placement and service scheduling in femtocell caching networks, to maximize the traffic volume served from the cache. The problem is modeled as a Markov decision process. We combine the Edmonds-Karp algorithm and the marginal allocation algorithm to develop an efficient centralized policy called Infinite CAche-filling (ICA), which can get arbitrarily close to optimal asymptotically as the estimation time window increases. We also design a randomized algorithm called Infinite CAche-filling with Probabilistic scheduling (ICAP) that takes into consideration the femtocells service capability due to interference or multiplexing techniques. We derive a lower bound on the expected discounted hit count of ICAP. We also derive an upper bound on the probability that the performance of ICAP degrades from this expected value. Numerical results show that ICAP scales well and converges relatively fast in response to request pattern changes.**

***Index Terms*—content-centric networking, femtocell network, Markov decision process, randomized algorithm.**

I. Introduction

As the demand for large content in cellular networks keeps increasing, small cell architectures, such as femtocells, are being deployed to better satisfy the ever-growing number of users in cellular networks[1]. It has been shown that enabling caching at base stations (BSs) and at small cell access points, termed *helpers*, is a promising approach to alleviate congestion at the network core and to decrease content access delay, so small cell deployment can get even denser without much efficiency degradation [2, 3]. [4] studied content placement in a single-cell with one BS and multiple cache-enabled helpers, given that each user has fixed connectivity with helpers. However, in [4], the BS has to keep track of the cached content in helpers and process two kinds of requests: one from users and one from helpers, which complicates the network architecture. In multiple cell architectures, BSs (and/or helpers) may have coverage overlaps [5]. Therefore, a user or a helper may have the opportunity of communicating with more than one BS. Taking the coverage overlaps into consideration, [6] developed a randomized content placement policy in cellular networks, but the implicit assumption that BSs can serve arbitrary number of users makes the policy inapplicable. [7] studied both content placement and request routing in multiple cells, while ignoring interference among users and not considering small cells that incorporate helpers between BS level and user level.

In this work, we study the joint problem of content placement and service scheduling to maximize a measure of the total traffic served from cache in order to alleviate the traffic volume in the network core. The service capability constraint of helpers is incorporated, which makes our model applicable to a more practical scenario where only a limited number of users can get service at a given time due to interference or multiplexing techniques. We first provide the Infinite CAche-filling algorithm (ICA) based on the marginal allocation algorithm that gives a near optimal solution if there is no service capability constraint. However, when this constraint is added to the formulation, ICA may not give a valid solution. So we develop the randomized algorithm called ICA with Probabilistic scheduling (ICAP) with the consideration of this constraint. We justify ICAP's performance by showing the expectation lower bound of the discounted hit count under ICAP. Numerical results show that ICAP scales well with the size of the network, and it performs significantly better than the derived lower bounds. Specifically, we make the following contributions:

- We incorporate the service capability constraint which can reflect the interference among users, and formulate the joint problem of content placement of service scheduling as a Markov Decision Process.
- We develop an efficient online randomized algorithm ICAP to alleviate the traffic in the network core by maximizing the discounted value of traffic volume served from cache.
- We derive the lower bound on the expected discounted hit count of ICAP. We also upper bound the probability that the performance of ICAP degrades from this expected value.

The remainder of this paper is organized as follows. In Section II we model the problem as a Markov decision process. In Section III we present the problem formulation. In Section IV, an efficient online algorithm called ICA is developed. In Section V we incorporate the service capability constraint and present a randomized algorithm ICAP, and justify its performance by deriving a lower bound on its expected value. Numerical results are presented in Section VI. We conclude our work and discuss future extensions in Section VII.

II. System Model

A. Topology

We consider multiple cells, each of which consists of a BS, a number of cache-enabled helpers, and a number of end users. Upstream network is simplified by direct duplex connections from BSs to a root server that stores all contents. The number of files in a helper's cache cannot exceed its cache size, which is referred to as the cache size constraint. At any given time, due to the interference or multiplexing techniques, a helper can serve a limited number of users, which we call the service capability constraint. Assume time is slotted, and a user sends one request to helpers only at the beginning of a time slot. Traffic is considered to be inelastic, which is a typical case in real time large content transmission, and any unserved request will be dropped at the next time slot before a new request is sent. We assume a strictly hierarchical topology where BSs can only serve helpers, and that helpers can only serve users.

Requests are handled as follows. Each user request is received at every helpers that can cover it. A helper cannot serve any user outside of its coverage, which is referred to as the coverage constraint. Since requests are typically short, their interference is negligible. A scheduling algorithm will then assign users to helpers to optimize certain performance metric. There is a cache miss if a helper is scheduled to serve a user that requests an uncached file. In this case, the request will be forwarded by the helper to the BS and get served by the upstream network. A helper serves the request directly if there is a cache hit.

B. Markov Decision Process (MDP) Modeling

We formulate the helpers' decision making process, which includes content placement and service scheduling, as an MDP problem. The MDP during time slot t is demonstrated in Fig. 1. Denote the set of users, helpers and files by U, H and F

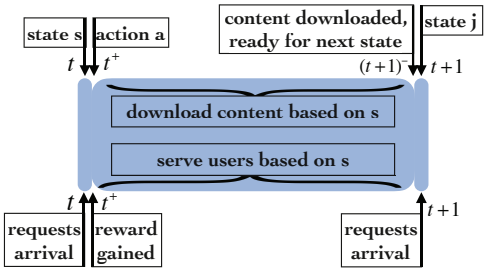


Fig. 1: MDP in the helper level during time slot t. Note that reward is gained immediately when action a is determined

respectively. Define a ground set E of size $|H||F|$ with the element $e_{h,f}$ representing caching file f in helper h. A state $s \in S$ is represented by a tuple $s = (\theta^s, \gamma^s)$, where $\theta^s \subseteq E$ is the caching state, and γ^s is the request state with γ^s_u denoting the file that user u requests. Let Γ be the set of all possible request states. After observing the current state, the helpers make caching decision $\hat{\theta}^a \subseteq E$, i.e. which files to download into cache during this time slot, and scheduling decision ω^a, i.e. which users to serve. The scheduling decision is an $|H| \times |U|$ matrix, where $\omega^a_{h,u} = 1$ if helper h schedules service for user u in the current time slot, otherwise $\omega^a_{h,u} = 0$. Thus an action can be expressed as a tuple $a = (\omega^a, \hat{\theta}^a)$. The states evolve according to the transition probability P. We assume the file popularity follows the Zipf-like distribution. Let $p(\gamma)$ be the probability that the requests vector from the users is γ. Further assume a request is independent of any previous request, then the transition probability from state s to state j after taking action a is given by

$$P(j|s,a) = P(j|\hat{\theta}^a) = \begin{cases} p(\gamma^j), \ \forall j : \theta^j = \hat{\theta}^a \\ 0, \ otherwise \end{cases}.$$

Define a hit matrix $\Omega(\gamma^s, \theta^s)$ as the adjacency matrix of the scheduling that accounts for cache hits between helpers and users, i.e. $\Omega_{h,u} = 1$ if helper h schedules service for user u that requests a cached file, otherwise $\Omega_{h,u} = 0$. Then the one-step reward can be defined as the total number of cache hits in the current time slot as follows

$$r(s,a) = |\Omega(\gamma^s, \theta^s)| \triangleq \sum_{h \in H} \sum_{u \in U} \Omega_{h,u}(\gamma^s, \theta^s).$$

We adopt the discounted hit count (DHC) as the performance metric, since the current network performance is valued more than its performance in the far future due to the changing wireless environment. Note that the discounted reward is finite as long as the discount parameter λ is less than 1, because the one-step reward is bounded for any state and action. Therefore, the problem definition is: *jointly determine the content placement and service scheduling in a centralized manner for maximizing the discounted hit count, subject to the service capability constraint, the coverage constraint, and the cache size constraint.*

III. Problem Formulation

In this section, we first define a θ-cluster that helps group the states, then we present the formulation of the joint problem of content placement and service scheduling.

Define the θ-cluster $c(\theta)$ as a set of states with the same caching state θ. Note that given a state $s \in c(\theta)$, if the caching action $\hat{\theta}^a$ is not changing the current cached contents, i.e. $\hat{\theta}^a = \theta^s = \theta$, then the next state is still in the same $c(\theta)$. So the expected one-step reward if the next state is in a specific θ-cluster $c(\theta^*)$ is given by

$$\begin{aligned} E[r(s,a) : \hat{\theta}^a = \theta^*] &= \sum_{j \in c(\theta^*)} P(j|\hat{\theta}^a = \theta^*) r(j,a) \\ &= \sum_{\gamma \in \Gamma} p(\gamma) |\Omega(\gamma, \theta^*)|. \end{aligned}$$

We now prove the optimal caching action is to cache content according to the $c(\theta^*)$ whose states have the largest expected one step reward. If we deviate from $c(\theta^*)$ from time slot t_1 to t_2, then all $E[r]$ from t_1 to t_2 are smaller than the $E[r]$ if we stay in $c(\theta^*)$. Since the DHC is the weighted sum of $E[r]$, any deviation from $c(\theta^*)$ results in a smaller DHC. Therefore, the optimal caching action will lead all states to $c(\theta^*)$. Note that if the file popularity changes, the current optimal θ-cluster may not be optimal anymore.

We now present some notations used in the problem formulation. Denote the scheduling hit matrix with the maximum hit count by $\Omega^*(\gamma, \theta)$. The helpers, users, and the helpers' coverage together form a bipartite graph G which we call the coverage graph. The adjacency matrix of this coverage graph is denoted by N with entry $N_{h,u} = 1$ if user u is in the coverage of helper h, otherwise $N_{h,u} = 0$. We use $|N|$ to represent the number of one entries in matrix N. Assume all helpers have the same service capability k, and denote the total service capability of all helpers by K. Let C_h and C indicate the cache size of helper h and the total cache size of all helpers respectively. Define the subset $E_h \subseteq E$ for every $h \in H$, which equals $\{e_{h,1}, e_{h,2}, \ldots, e_{h,|F|}\}$.

Assuming file popularities are known, the optimal deterministic action $a^* = (\omega^*, \hat{\theta}^*)$ at state s can be formulated as

$$\omega^* = \arg\max_{\omega^a} r(s,a) = \Omega^*(\gamma^s, \theta^s) \tag{1}$$

$$\hat{\theta}^* = \arg\max_{\theta} \sum_{\gamma \in \Gamma} p(\gamma)|\Omega^*(\gamma, \theta)| \tag{2}$$

$$s.t. \sum_{u \in U} \omega_{h,u} \leq k, \ \forall h \in H \tag{3}$$

$$N_{h,u} - \omega_{h,u} \geq 0, \ \forall h \in H, \ \forall u \in U \tag{4}$$

$$|\theta \cap E_h| \leq C_h, \ \forall h \in H \tag{5}$$

where (3) is the service capability constraint (SCC), (4) is the coverage constraint (CVC), and (5) is the cache size constraint (CSC). The DHC is given by:

$$\begin{aligned} \text{DHC} =& \sum_{t=1}^{\infty} (\sum_{\gamma \in \Gamma} p(\gamma)|\Omega(\gamma, \theta^*)|)\lambda^{t-1} \\ =& \frac{1}{1-\lambda} \sum_{\gamma \in \Gamma} p(\gamma)|\Omega(\gamma, \theta^*)|. \end{aligned} \tag{6}$$

IV. Formulation Analysis and ICA Algorithm

In this section, we first solve the scheduling problem using the Edmonds-Karp algorithm [8], then we prove if SCC is removed, the cache placement problem is weakly concave subject to a polymatroid constraint, so it can be solved optimally by the marginal allocation algorithm [9]. Lastly, we give an efficient approximation algorithm named ICA.

A. Scheduling Problem

The hit matrix in Eq. (1) can be solved for by some max-flow algorithm on a bipartite graph G' constructed from the coverage graph G by the following three steps.

Step 1: Add a source node v, and add an edge from v to each helper with the capacity equaling the helper's service capability. If there is no SCC, the capacity of this edge is set to the cardinality of the helper's coverage.

Step 2: Add a sink node t, and add an edge from each user to t with capacity 1. This step assures that one user can be served by only one helper at a time.

Step 3: For any edge (h, u) in graph G where user u requests an uncached file in helper h, remove it from graph G.

Now we prove the optimality of the solution obtained by running the max-flow algorithm on the new graph G'. The solution is in the form of a one-to-many matching between helpers and users, where a one-unit flow from a helper to a user represents a cache hit. According to the flow conservation, the amount of flow from the source to a helper equals the number of cache hits in the helper. The max-flow algorithm computes the maximum flow from the source to the network, so the solution gives the maximum number of cache hits.

The max-flow problem can be solved by the well-known Edmonds-Karp algorithm in $O(VL^2)$ time, where V and L is the number of nodes and the number of edges respectively. Therefore in the worst case, the scheduling problem can be solved in $O(|U|^3)$ if $|U| >> |H|$.

B. Content Placement Problem

The inner problem in (2) is finding the optimal scheduling given the state subject to SCC and CVC, and is solved in Section IV.A. Its outer problem is finding the optimal caching strategy given the optimal scheduling subject to CSC. We will prove the outer problem is weakly concave subject to a polymatroid constraint, so it can be solved optimally by the marginal allocation algorithm (MAA) by incrementally adding one more element to the solution which brings the most gain.

First we show that the CSC of the outer problem is a polymatroid constraint. Define a complete order R on the subset $\theta \subseteq E$ such that $\theta^x \leq_R \theta^y$ if and only if $f(\theta^x) \leq f(\theta^y)$, where $f()$ is some function of θ. Additionally, we say $\theta^x \leq \theta^y$ if all cached files in caching state θ^x are also cached in corresponding helpers in caching state θ^y, i.e. $\theta^x \subseteq \theta^y$. The CSC of the outer problem can be written as a partition matroid $P = \{\theta : \theta \subseteq E, |\theta \cap E_h| \leq C_h\}$. Since a partition matroid is a polymatroid, the CSC immediately satisfy (F1) through (F3) in [9].

Then we prove that without SCC the content placement problem is weakly concave by showing the complete order R satisfies (R1) and (R3) in [9], both of which are paraphrased and listed below. Since the affine combination of concave functions are still concave, we only need to prove $|\Omega^*(\theta^x)|$ is weakly concave given a specific request state $\gamma \in \Gamma$. So the complete order becomes: $\theta^x \leq_R \theta^y$ if and only if $|\Omega^*(\theta^x)| \leq |\Omega^*(\theta^y)|$ for some specific request state γ.

(R1): if $\theta^y \geq \theta^x, \theta^x \geq_R \theta^x \cup e_{h,f}$, then $\theta^y \geq_R \theta^y \cup e_{h,f}, h \in H, f \in F$.

Proof: adding file f to cache h does not decrease the overall cache hit, so $\theta^x \geq_R \theta^x \cup e_{h,f}$ implies $\theta^x =_R \theta^x \cup e_{h,f}$, which means no user within helper h's coverage requests file f. Therefore, adding $e_{h,f}$ to θ^y does not yield one more cache hit, i.e. $\theta^y =_R \theta^y \cup e_{h,f}$.

(R3): if $\theta^y \geq \theta^x, e_{h,f} \notin \theta^x, e_{h,f} \notin \theta^y, e_{p,q} \notin \theta^x, e_{p,q} \notin \theta^y$, and $\theta^x \cup e_{h,f} \geq_R \theta^x \cup e_{p,q}$, then $\theta^y \cup e_{h,f} \geq_R \theta^y \cup e_{p,q}$.

Proof by contradiction: suppose adding $e_{h,f}$ to θ^y yields less cache hits than adding $e_{p,q}$ to θ^y. This is true only if no user within helper h's coverage requests file f, while some user within helper p's coverage request file q. So in caching state θ^x, caching file q in helper p will yield a cache hit, while caching file f in helper h will not, i.e. $\theta^x \cup e_{h,f} <_R \theta^x \cup e_{p,q}$.

By contradiction, (R3) holds. Note that this proof fails with the presence of SCC.

Therefore, the complete order R is weakly concave if the SCC is removed. From Theorem 2 in [9], the MAA algorithm gives the optimal solution.

C. ICA algorithm

In this subsection, we modify the MAA in order to develop an efficient approximation algorithm. The original MAA is not efficient, which can be shown by the following observations.

The probability $p(\gamma)$ can be calculated as the joint p.m.f. of $|U|$ requests, each of which follows the Zipf-like distribution. The objective function (2) is the expectation of the maximum hit counts, i.e. $\mathbb{E}[|\Omega^*|]$. According to MAA, the objective function is evaluated for every additional file to be cached in the helpers. To ensure the polynomial complexity of MAA, the objective function should be evaluated in polynomial time. However, the objective evaluation takes $|F|^{|U|}$ computations of optimal hit counts, which is intractable as the file universe grows large. Moreover, since the file popularity is typically unknown in practice, $p(\gamma)$ cannot be calculated a-priori. Therefore, instead of brute-force calculation, we approximate $\mathbb{E}[|\Omega^*|]$ by sampling over an estimation time window.

We now describe the sampling approach with a time window of T time slots. Construct an $|H| \times |F|$ zero matrix A. After helpers received requests at the beginning of a time slot, compute the optimal scheduling $\Omega^*(\gamma, \theta \cup e_{h,f})$ and add the optimal hit count to entry $A_{h,f}$ for all h and f such that $e_{h,f} \notin \theta$. At the next time slot, the same calculation is executed, and the newly obtained hit counts are added to the corresponding entries of A. After T time slots, the entry $A_{h,f}$ is holding the sum of previous T hit counts of caching the content f to the helper h. Dividing all entries of A by T yields the sample mean of those hit counts during the time window T. Next, we allocate the file f to the helper h, which corresponds to the largest sample mean. Using this sampling approach, the objective evaluation takes $O(T)$ computations of optimal hit count as opposed to the original $O(|F|^{|U|})$.

We now present the overall algorithm and its complexity. Given the efficient evaluation method described above, we run MAA to fill the cache of all helpers, and we call it the cache-filling procedure, as demonstrated in Fig. 2. $T|H||F|$ computations of optimal hit count are needed to add one more file to the cache. Therefore, one run of caching-filling takes $O(TC|H||F|)$ computations of optimal hit count. Due to the sampling approach in the evaluation step, if we run the cache-filling procedure back to back over an infinite time horizon, the caching state can be updated in response to request pattern changes. This Infinite CAche-filling is referred to as ICA hereinafter. According to the Central Limit Theorem, ICA can get arbitrarily close to the optimal if we drive the sample mean close to the real mean by setting a large window T.

The ICA algorithm is computationally efficient. If the Edmonds-Karp algorithm is used for computing the optimal hit counts, during each time slot there are at most $|H||F|$ runs of Edmonds-Karp algorithm. To reduce the average

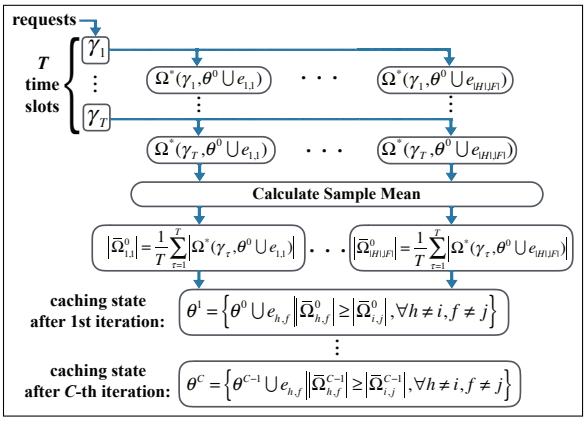


Fig. 2: Cache-filling procedure based on MAA. θ^0 is the initial caching state, i.e. nothing in cache. $|\bar{\Omega}^i_{h,f}|$ is the sample mean of the maximum hit count of the i-th iteration if file f is allocated to helper h.

complexity over time, we can sample the requests every n time slots instead of every time slot. To further reduce computational complexity, we can distributed the calculation across H helpers if the information of request arrivals, coverage, connectivity and cache size can be exchanged among them, which is one possible direction of our future work.

V. Probabilistic Scheduling Based on ICA

The above ICA algorithm gives a near-optimal solution with the absence of SCC. However, its performance is unbounded if we include SCC to model the more practical scenario where a helper can only serve a limited number of users due to interference or multiplexing techniques. Moreover, since we only model the cache hits in (2), the resulting scheduling will only serve those users who requests cached files, which has potential fairness issue. Therefore, we develop a probabilistic scheduling scheme based on ICA in Section V.A in order to satisfy SCC and provide some degree of fairness among users. We call the combination of the probabilistic scheduling and the ICA caching scheme as ICAP. We derive the expectation lower bound of the DHC under ICAP in Section V.B.

A. Algorithm Design

Denote the new probabilistic scheduling by the adjacency matrix $X(\gamma, \theta^*)$ (abbreviated as X^*), which is obtained by the following randomization steps on the scheduling $\Omega^*(\gamma, \theta^*)$ (abbreviated as Ω^*) returned by ICA. We abbreviate the notion "set the corresponding entry of the scheduling X^* to one/zero" as "add/delete an edge to/from X^*" hereinafter. We refer to the edges that corresponds to a one entry in Ω^*, as a "hit edge", otherwise it is a "miss edge".

Step 1: Assign a probability to every edge in the coverage graph G as follows: if it is a hit edge, assign it with probability $p_1(\Omega^*) = \frac{P}{|\Omega^*|}$, where $P \in [0, 1]$. Otherwise, assign it with $p_2(\Omega^*) = \frac{1-P}{|N|-|\Omega^*|}$. Suppose P is set in such way that the smallest possible $p_1(\Omega^*)$ is p_1.

Step 2: Pick an edge at random from the coverage graph G and add it to X^*. Each edge is picked with the assigned probability in Step 1. We repeat this random pick for M times

with replacement. If an edge is selected but it is already added to X^*, we just proceed to the next pick.

Step 3: After M picks, the number of incident edges to a user u (denoted by L_u) may be greater than one. For each of such user, we retain the hit edge and delete the rest. If none of the user's incident edges are hit edges, then we pick one of them uniformly at random and delete the rest.

Step 4: For each helper with the number of incident edges (denoted by L_h) more than its service capability, we repeatedly pick one of its incident edges uniformly at random with replacement, and stop until k different edges are picked or until we picked for $k \ln k$ times (whichever happens first). Then, we delete the edges that are not ever picked.

B. Lower bound on expected performance of ICAP

Denote the DHC under ICAP and ICA by Z_{ICAP} and Z_{ICA} respectively. Assume that the helpers have the same size of coverage. We have

$$E[Z_{ICAP}] \geq Z_{ICA}(1 - e^{-\frac{K}{|N|}})(1 - (1 - p_1)^M). \quad (7)$$

We can increase p_1 and M to improve the lower bound. However a large p_1 may decrease the fairness, which is left to our future work. The DHC can get arbitrarily low even with a high expected value, therefore, we also upper-bound the probability that Z_{ICAP} degrades from this lower bound:

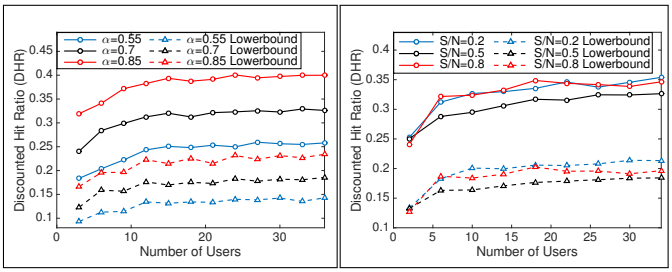
$$Pr\Big[E[Z_{ICAP}] - Z_{ICAP} \geq 1\Big] \leq 2e^{-(\ln|F|)^{|U|}(\frac{1-\lambda}{K/\sqrt{2}})^2}. \quad (8)$$

From (8) , we note it is almost impossible that ICAP's performance degrade from the lower bound of the expected DHC, given a relatively large file universe and network size. The derivation of the lower bound (7) and the upper bound (8) are presented in the appendix.

VI. Numerical Results

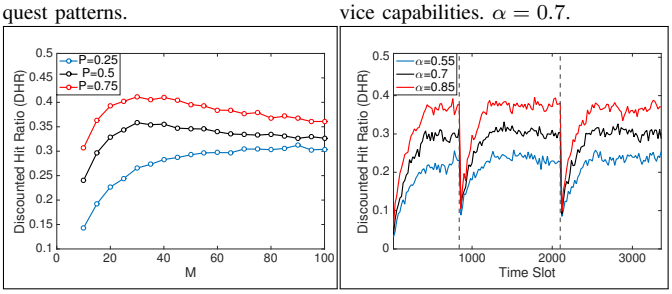
In this section, we study the scalability, the effect of the randomized edge-picking parameters P and M, and the convergence of ICAP under different settings via MATLAB simulation. In order to compare the ICAP performance against the performance upper bound, we measure the Discounted Hit Ratio (DHR), which is defined as the DHC normalized by an upper bound computed by assuming all requested files from the served users are cached. We approximate DHC by feeding the same request stream for additional 100 time slots (discounted parameter is set to 0.8). We compute the normalized lower bound in the same manner using (7). The request stream follows the Zipf-like distribution. Estimation time window is set to 20 time slots. There are 3 helpers in the network, each of which can cache at most 7 files. The coverage is randomly generated in a way that every user can be covered by at least one helper. The size of the file universe is set to 100. M and P are set to $|N|\ln(|N|)$ and $\frac{2|N|-|\Omega^*|}{|N|^2/|\Omega^*|}$ respectively, unless specified otherwise. We average over 20 simulations to get each data point, except for Fig. 3d where the DHRs are real time values when ICAP is running.

We first analyze the scalability of ICAP under different request patterns and service capabilities. Fig. 3a and Fig. 3b show that ICAP scales well with the growth of the number of users. The dashed lines in the two figures are the expectation lower bounds under corresponding settings. Fig. 3a plots the scalability of ICAP under different request patterns. If the exponent parameter of the Zipf-like distribution is increased, ICAP yields higher DHR with the same network size. A skewer Zipf-like distribution means higher ranked files are more likely to be requested, so the sampling procedure can detect these higher ranked files more accurately and download them to cache. Fig. 3b shows the scalability of ICAP under different service capabilities. Note that the service capability $k = N$ means a helper can serve all users within its coverage during a time slot. Fig. 3b shows there is no significant difference when the service capability changes, which demonstrates that ICAP solution scales well regardless of the SCC.



(a) Scalability under different request patterns.

(b) Scalability under different service capabilities. $\alpha = 0.7$.



(c) Impact of the edge-picking parameters P and M on discounted hit ratio. $\alpha = 0.7$.

(d) Convergence of ICAP. Gray dashed lines represent request pattern changes.

Fig. 3: Discounted hit ratio of ICAP under different network settings

Then we analyze the effect of the two parameters P and M in the randomized edge-picking procedure. If P is set to a large value, more hit edges will be added to the scheduling X in Step 1 of ICAP. Therefore the performance is better in terms of DHR, as shown in Fig. 3c. From the lower bounder (7), M should be large. However, if M increases to the point where all hit edges have been picked and the edge-picking procedure continues, more miss edges will be added to the scheduling X in Step 1 of ICAP. Therefore, there will be more miss edges incident to helpers after Step 3. This results in a higher probability that a hit edge is removed in Step 4, thus the performance slowly declines. Note that the fairness may increase at a cost of DHR performance by setting a proper P and M. We leave the study on fairness to our future work.

Finally, we discuss the convergence of ICAP and its performance upon request pattern changes. We consider a network of 20 users and 3 helpers. Each helper can serve at most 5 users during a time slot. We simulate ICAP for 3360 time

slots, where the ranking of files is changed at the 840-th time slot and at the 2100-th time slot. As shown in Fig. 3d, DHR converges after one run of cache-filling. Upon request pattern changes, the performance declines as expected, and then converges after one additional run of cache-filling. In addition, DHR is affected more by the request pattern change if the file popularity is skewer, because the file ranking change of a flat file popularity does not affect the joint distribution of requests as much as the same change of a skewer file popularity does. No matter how request pattern varies, DHR will surely converge after one run of cache-filling, that is TC time slots where C is the total cache capacity of all helpers.

VII. Conclusion and Future Work

In this paper, we study the joint problem of content placement and service scheduling in femtocell caching networks, with the objective of maximizing the discounted value of the traffic served from cache. A deterministic ICA algorithm is developed which can get arbitrarily close to optimal without SCC. We also proposed a probabilistic ICAP algorithm to satisfy SCC and provide some degree of fairness. However, the fairness performance is unknown and worth further study. In addition, a more efficient probabilistic scheduling and a tighter lower bound warrant further investigation. Moreover, if the BSs are also cache-enabled, we have a more general hierarchical caching network, which is one promising direction of our future work.

Acknowledgement

This material is based upon work supported by the U.S. National Science Foundation under grant numbers 1422153 and 1456887, and a Tekes FiDiPro Fellow award with University of Oulu, Oulu, Finland.

References

[1] V. Chandrasekhar, J. G. Andrews, and A. Gatherer, "Femtocell networks: A survey," *CoRR*, vol. abs/0803.0952, 2008.

[2] 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 Communications Magazine*, vol. 51, no. 4, pp. 142–149, 2013.

[3] N. Golrezaei, K. Shanmugam, A. G. Dimakis, A. F. Molisch, and G. Caire, "Wireless video content delivery through coded distributed caching," in *ICC*, pp. 2467–2472, IEEE, 2012.

[4] N. Golrezaei, K. Shanmugam, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless video content delivery through distributed caching helpers," in *INFOCOM*, pp. 1107–1115, IEEE, 2012.

[5] H. P. Keeler, B. Blaszczyszyn, and M. K. Karray, "Sinr-based k-coverage probability in cellular networks with arbitrary shadowing," in *ISIT*, pp. 1167–1171, IEEE, 2013.

[6] B. Blaszczyszyn and A. Giovanidis, "Optimal geographic caching in cellular networks," in *ICC*, pp. 3358–3363, 2015.

[7] K. Naveen, L. Massoulie, E. Baccelli, A. Carneiro Viana, and D. Towsley, "On the interaction between content caching and request assignment in cellular cache networks," in *5th Workshop on All Things Cellular: Operations, Applications and Challenges*, (New York, NY, USA), pp. 37–42, ACM, 2015.

[8] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. ACM*, vol. 19, pp. 248–264, Apr. 1972.

[9] A. Federgruen and H. Groenevelt, "The greedy procedure for resource allocation problems: necessary and sufficient conditions for optimality," *Operations Research*, vol. 34, pp. 909–918, 1986.

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

Appendix

We now derive a lower bound on the expected DHC of ICAP. An entry X^*_{hu} corresponds to a cache hit if $X^*_{hu}=1$ given that $\Omega^*_{hu}=1$. We denote the number of such entries in X^* by $g(X^*)$. From the linearity of expectation, we have

$$\begin{aligned} E[g(X^*)] &= \sum_{h,u:\ \Omega^*_{hu}=1} E[X^*_{hu}] \\ &= \sum_{h,u} Pr[X^*_{hu}=1|\Omega^*_{hu}=1]. \end{aligned} \tag{9}$$

We now derive the probability $Pr[X^*_{hu}=1|\Omega^*_{hu}=1]$. If $\Omega^*_{hu}=1$, the probability that $X^*_{hu}=1$ after Step 2 is $1-(1-p_1(\Omega^*))^M \geq 1-(1-p_1)^M$. Step 3 does not affect this edge if it is retained after Step 2. In Step 4, edge (h,u) will remain in X^* if $L_h \leq k$, otherwise the edge remains in X^* after d times of edge-picking with probability $1-(1-\frac{1}{L_h})^d \geq 1-(1-\frac{1}{|N|/|H|})^k \geq 1-e^{-\frac{K}{|N|}}$. So we have

$$\begin{aligned} Pr[X^*_{hu}=1|\Omega^*_{hu}=1] &= (1-(1-p_1(\Omega^*))^M)Pr[L_h\leq k]+ \\ &\quad (1-(1-\frac{1}{L_h})^k)(1-(1-p_1(\Omega^*))^M)Pr[L_h>k] \\ &\geq (1-e^{-\frac{K}{|N|}})(1-(1-p_1)^M). \end{aligned} \tag{10}$$

From (9) and (10), we have

$$E[g(X^*)] \geq |\Omega^*|(1-e^{-\frac{K}{|N|}})(1-(1-p_1)^M). \tag{11}$$

Denote the DHC under ICAP and ICA by Z_{ICAP} and Z_{ICA} respectively. From (11) we have

$$\begin{aligned} E[Z_{ICAP}] =& E\Big[\frac{1}{1-\lambda}\sum_{\gamma\in\Gamma} p(\gamma)g(X^*)\Big] \\ \geq& \frac{1}{1-\lambda}\sum_{\gamma\in\Gamma} p(\gamma)|\Omega^*|(1-e^{-\frac{K}{|N|}})(1-(1-p_1)^M) \\ =& Z_{ICA}(1-e^{-\frac{K}{|N|}})(1-(1-p_1)^M). \end{aligned}$$

We now derive the upper bound of the probability that Z_{ICAP} degrades from the lower bound (7). By Hoeffding bound, for any non-negative t we have

$$Pr[E[Z_{ICAP}]-Z_{ICAP}\geq |F|^{|U|}t] \leq 2e^{\frac{-2|F|^2|U|t^2}{\sum_{\gamma\in\Gamma}(\frac{Kp(\gamma)}{1-\lambda})^2}}. \tag{12}$$

Let $t=|F|^{-|U|}$ and let Q be the normalization parameter of the Zipf-like distribution, i.e. $Q=\frac{1}{\sum_{i=1}^{|F|} i^{-\alpha}} \leq \frac{1}{\ln|F|}$ [10]. So $\sum_{\gamma\in\Gamma} p^2(\gamma) \leq Q^{|U|}\sum_{\gamma\in\Gamma} p(\gamma) = Q^{|U|} \leq (\ln|F|)^{-|U|}$. Then (12) becomes

$$Pr\Big[E[Z_{ICAP}]-Z_{ICAP}\geq 1\Big] \leq 2e^{-(\ln|F|)^{|U|}(\frac{1-\lambda}{K/\sqrt{2}})^2}.$$