FemtoCaching: Wireless Video Content Delivery through Distributed Caching Helpers *

Negin Golrezaei, Student Member, IEEE, Karthikeyan Shanmugam, Alexandros G. Dimakis, Member, IEEE, Andreas F. Molisch, Fellow, IEEE, and Giuseppe Caire, Fellow, IEEE

Dept. of Electrical Eng.
University of Southern California
Los Angeles, CA, USA
emails: {golrezae,kshanmug,dimakis,molisch,caire}@usc.edu

Abstract—We propose and analyze a novel way for increasing the area spectral efficiency of video transmission in cellular communications systems. Small base stations called helpers with low-rate backhaul but high storage capacity cache popular video files. Mobile stations requesting video files receive them via short-range transmissions from helpers, if they are present in their caches; files not available from helpers are transmitted by the base station. We analyze the optimum way of assigning files to the different helpers, in order to minimize the expected file downloading time for files. We distinguish between the uncoded case (where only complete files are stored) and the coded case, where segments of Fountain-encoded versions of the video files are stored at helpers. We show that for the uncoded files, the optimum file assignment is NP-hard, and develop a greedy strategy that is provably within a factor 2 of the optimum. We furthermore show that for the coded case, the optimum file assignment can be formulated as a convex problem that can be further reduced to a linear program. Numerical simulations based on measured traces of YouTube requests demonstrate that our approach can improve video throughput by a factor 3 – 5 in realistic settings.

I. INTRODUCTION

Video is the main reason for the dramatic growth of data traffic over cellular communications networks - an increase by two orders of magnitude compared to current volume is expected in the next five years [1]. While this constitutes a great business opportunity, there is the danger that this increased demand will lead to a slowdown, or even breakdown, of cellular systems, frustrating customers, and essentially jeopardizing the success of more and more advanced wireless data services. Therefore, the tremendous increase in mobile video content demand will require a corresponding increase of area spectral efficiency of wireless networks. The most promising approach to achieve this is shrinking the cell sizes and essentially bringing the content closer to the users. This can be implemented by deploying small base stations that achieve localized communication and enable high-density spatial reuse of communication resources [2]. Such pico- and femto-cell networks, which are usually combined with macrocells into a heterogeneous network, are receiving a lot of attention in the recent literature, (see e.g. [3] and references therein).

However, the requirement for high-speed backhaul to the cellular operator network makes this an expensive option [3]. In this paper we suggest a radically new way to deal with the backhaul bottleneck problem. Our main idea is to provide the femto-base stations, typically served by rate-limited DSL backhaul links, with large storage capacity. In addition, we may also think of deploying fixed wireless caching nodes with no backhaul link at all, beyond the standard wireless connection to the cellular network. These stations, which we henceforth call caching helpers, or simply helpers, form a wireless distributed caching infrastructure.

This novel architecture is based on the following observations: (i) most video traffic is generated by encoded and pre-stored video in large video on demand servers such as Hulu, YouTube and Netflix, rather than live streaming, which amounts for a relatively tiny fraction of the overall video traffic. A large portion of the video on demand traffic corresponds to a few popular files; (ii) disk storage is a quantity that increases faster than brute-force increase of spatial reuse (with the associated increase of backhaul capacity requirement). Based on these seemingly simple ideas, we propose a system architecture that achieves the required throughput at considerably lower cost. The key insight is that if there is enough content reuse, i.e. many users are requesting the same few video files, caching can replace backhaul communication. The most popular files are stored in the cache, and are therefore always available locally to the UTs (User Terminal) that are requesting them.

More concretely, the helpers are placed in fixed positions in the cell and are assumed to have (i) large storage capacity, (ii) localized, high-bandwidth wireless communication capabilities which enable high frequency reuse, and (iii) low-rate backhaul links which can be wired or wireless. They can cache popular files and serve requests from mobile User Terminals (UTs) by enabling localized communication. Our approach is thus fundamentally different from a heterogeneous network using femto base stations, which do not have caches, and thus need to obtain any file through their backhaul network when it has been requested locally.

The helpers are working in conjunction with conventional macrocellular base stations, which provide to the UTs the video files that cannot be obtained from the helpers. Clearly, the smaller the percentage of file requests that has to be fulfilled by the macrocell, the larger the number of UTs that can be served. We will show later on in this paper that, with realistic setups, the number of users that can be served is increased by as much as 300 – 500%.

The key question for such a system is the wireless distributed caching problem, i.e., which files should be cached by which helpers. If every mobile device has only access to exactly one helper, then clearly each helper should cache
the same files, namely the most popular ones. However, for the case of dense helper deployment, each mobile device can access multiple caches and sees a distributed cache that is the union of the helpers’ caches. As we show later, in this case, the assignment of files to helpers becomes highly nontrivial.

We consider two versions of the distributed caching problem.
- In the first case, only complete files are stored in the helpers caches directly. We refer to this case as the uncoded distributed caching problem and we will show that finding an optimal placement is computationally intractable (NP-complete). Clearly, coding is used at the physical layer to achieve reliable communication. The uncoded nature of this problem consists of the fact that the raw data packets are directly stored in the caches.
- In the second case, we allow coding. For simplicity, we assume that it is implemented by an ideal MDS rateless code. An MDS rateless code generates an arbitrarily long sequence of parity symbols from an information packet of fixed size of \( B \) symbols, such that if the decoder obtains any \( B \) parity symbols, it can recover the original \( B \) information symbols. It is well-known that this behavior can be closely approached (at the cost of a small redundancy overhead) by Raptor codes [4]. Hence, in this case the individual identity of the packets is not relevant. Rather, what matters is how many parity symbols of a given file are retrieved from the helpers within the reach of each user. We will show that the coded distributed caching problem is a convex optimization problem that can be solved efficiently. By introducing additional auxiliary variables we can further reduce it to a linear program.

The contribution of this work is thus four-fold:
- Femtocaching: We propose a novel network architecture for video delivery that replaces (expensive) high-rate backhauls with (cheap) storage units combined with low-rate backhauls. We formalize the wireless distributed caching optimization problem, i.e., the question of which files should be assigned to which helpers. While there is a substantial amount of prior work on caching algorithms for web and video content (e.g., [5], [6], [7] and references therein), to the best of knowledge, all this related work focuses on wired and p2p networks and has a different focus and constraints, compared to the wireless problem we investigate here.
- Uncoded Femtocaching: The uncoded distributed caching problem is a special covering problem that involves placing files in helpers to minimize the total average delay of all UTs or equivalently maximize the weighted distributed caching that all the users see jointly. We show that finding the optimum placement of files is NP-complete. Further, we express the problem as a maximization of a submodular function subject to matroid constraints (see also [8] for other applications of matroid theory in networks).
- Coded Femtocaching: For coded distributed caching problem, we show that the expected delay minimization problem can be formulated as a convex optimization problem.
- Performance evaluation: We present a detailed simulation of a university campus scenario covered by a single macro-BS (using the cellular standard 3GPP LTE R8) and several helpers using a simplified 802.11n protocol. File requests are made according to real-world user request based on traces of YouTube videos from the Amherst campus [9] [10]. Our simulations show that there are very significant gains even when very simple caching algorithms are used.

The remainder of the paper is organized as follows: Section II presents the system model and assumptions. In Section III we formulate the uncoded distributed caching problem and present the solution for it. Section IV discusses the coded distributed caching problem. In Section V we verify our architecture by simulation results which are based on the experimental YouTube requests. A summary concludes the paper in Sec. VI.

II. DISTRIBUTED CACHING PLACEMENT MODEL AND ASSUMPTIONS

We consider a single cell, equipped with a base station (BS), serving a large number of User Terminals (UTs), with the help of dedicated content distribution nodes, or helpers. The helpers are placed in fixed positions in the cell and are assumed to have (i) large storage capacity, (ii) localized, high-bandwidth communication capabilities which enable high frequency reuse, and (iii) low-rate backhaul links which can be wired or wireless; they thus form a distributed caching infrastructure. Figure 1 illustrates the system layout. The key point is that if there is enough content reuse, i.e. many users are requesting the same video content, caching can replace backhaul communication.

The helpers are working in conjunction with conventional macrocellular base stations. If a UT requests a file that is cached in local helpers, the helpers handle the request; otherwise, the BS should serve the request. Clearly, the smaller the percentage of video requests that has to be fulfilled by the macrocell, the larger the number of UTs that can be served.

Fig. 1. An example of the single-cell layout. UTs are randomly distributed, while helpers can be optimally placed in the coverage region.

Here, we formulate the following wireless distributed caching problem: for given popularity distribution, storage capacity in the helpers and wireless communication model for the BS to UTs downlink and helpers to UTs links, how...
should the files be placed in the helpers such that the average sum delay of all users is minimized? Since users experience shorter delay when they locally download from helpers in their neighborhoods instead of the BS, minimizing the average delay for a given user is equivalent to maximizing the probability of finding the desired content in the helpers within the UT’s reach. The solution is trivial when there are few helpers in the cell and as a result each UT can connect only to a single helper. In this case, each helper should cache the most popular files, in sequence of popularity, until its cache is full. If the helper deployment is dense enough, UTs will be able to communicate with several such helpers and each sees a distributed cache that is the union of the helpers caches. In this situation, the question on how to best assign files to different helpers becomes a much more complicated issue, because each UT sees a different, but correlated, distributed cache. In section III we illustrate through examples the complexity of finding the optimum file placement when some users are connected to more than one helper. Moreover, we show that the uncoded distributed caching problem is NP-complete.

We assume that the popularity distribution of the files changes slowly. Typical examples include popular news, containing short videos, which are updated every 2-3 hours, new movies, which are posted every week, new music videos, which are posted (or change popularity) about every month. Due to time-scale decomposition, the popularity distribution of the files is effectively fixed; furthermore it can be learned by the system, and thus be assumed known for our further considerations. Moreover, the cost of refreshing the helpers’ content can be safely neglected. Once the optimal content placement is determined, the BS centrally populates the helpers’ caches using weak backhaul links.

There are \( H \) helpers, \( K \) user terminals, and a library of \( N \) files, denoted by \( \mathcal{F} \). All files are assumed to have the same size. This assumption is mainly used for notational convenience, and could be easily lifted by considering a finer packetization, and breaking longer files into blocks of the same length. The popularity distribution of the files conditioned on the event that a user makes a request is denoted by \( P_n \), for \( n = 1, \ldots, N \). The connectivity between users and helpers can be represented in a bipartite graph; one example is shown in Figure 2. If there is an edge between helper \( h \) and UT \( k \), it means that UT \( k \) can communicate reliably with helper \( h \). In practice, the connectivity graph is determined by the location of users and helpers and transmission radius of the helpers. We assume that the connectivity between UTs and helpers does not change during the transmission of a video file. This requires our users to be fairly static compared to the download time.

When a user requests some file \( n \), it first asks its local helpers, i.e., helpers in the neighborhood of the user in the user-helper connectivity graph. We assume that, at the cost of a small protocol overhead, the BS maintains a list of all the helpers caches content. A user makes a request, and it is redirected to a local helper if the request is available there, or it is handled by the BS directly if it is not available.

III. UNCODED DISTRIBUTED CACHING PLACEMENT

In the uncoded case, a file can be either entirely cached or not cached at all. As mentioned in Sec. I, if each UT can communicate to only one helper, the optimal caching policy is simple: each helper should cache the most popular files. When a user has connection to multiple helpers, however, the caching policy becomes non-trivial, as shown in the example of Figure 3. There are two helpers and four UTs. The dashed circles centered around helpers indicate the helper transmission radius. Assuming that each helper can cache \( M \) files, users \( U_1 \) and \( U_2 \) would prefer helper \( H_1 \) to cache the \( M \) most popular files since this minimizes their expected delay. Similarly, user \( U_4 \) would prefer that helper \( H_2 \) also caches the \( M \) most popular files. However \( U_3 \) would prefer \( H_1 \) to cache the \( M \) most popular files and \( H_2 \) the second \( M \) most popular (or the opposite). This effectively creates a distributed cache of size \( 2M \) for user \( U_3 \). As can be seen, in the distributed caching problem, the individual objectives of different users may be in conflict.

Assume that the set of available transmitters for user \( k \) is equal to \( \mathcal{N}(k) = \{ h_k^1, h_k^2, \ldots, h_k^{N(k)-1}, BS \} \). The reciprocal of the average data rates for the link between user \( k \) and its local helper \( h \) (including BS as the “helper” number \( h = |\mathcal{N}(k)| \) are denoted by \( \Omega_k = \{ \omega_k^1, \omega_k^2, \ldots, \omega_k^{N(k)-1}, \omega_k^{|\mathcal{N}(k)|} \} \). As a matter of fact, the instantaneous rate of these links fluctuates because of fading and multiple-access scheduling, so that the instantaneous rate might be zero on some slots. Nevertheless, assuming that each user is a “drop in the ocean”, i.e., that the mutual influence of the users on each other is negligible, we may consider each link as a channel of variable capacity, and given average rate that depends only on the link itself,
i.e., basically on the distance between the helper and the user. At this point, assuming that the file size is large enough so that long-term time averages are meaningful, the experienced delay for user $k$ for downloading $B$ bits from helper $h$ is given by $B \times \omega^k_h$ where the $\omega$ coefficients have the dimension of sec/bit, i.e., they are reciprocal of rates. Regarding $\omega^B_S$, it can be observed and shown analytically that under the standard proportional fairness downlink for high-speed data HSDPA in 3GPP, Ev-DO in 3GPP2, and LTE R8 [2], the delay of downloading from the BS depends on the overall number of active users in the system on the individual user position in the cell and it is linear with the file size [1]. To experience lower delay, users prefer to get files from transmitters that are closer to them. So, the average delay normalized to the file size for user $k$ will be:

$$D_k = \omega^1_k \sum_{n \in C^k_h} P_n + \omega^2_k \sum_{n \in C^k_h \setminus C^k_h} P_n + \omega^3_k \sum_{n \in C^k_h \setminus (C^k_h \cup C^k_h)} P_n + \ldots + \omega^B_S \sum_{n \in F \setminus \cup_{i=1}^{N(k)-1} C^k_h} P_n,$$

where $C_h$ is the files in the cache of helper $h$. Considering the finite (albeit large) storage capacity of helpers, we wish to find the optimal placement of the $N$ files on the helpers’ caches in order to minimize the total average delay of all UTs, i.e., $\sum_k D_k$. We can write the optimization problem as follows:

$$\text{maximize} \quad \sum_{k=1}^{K} \omega^B_S \sum_{n \in A_k} P_n$$

subject to

$$|C_h| \leq M, \quad \forall h,$$

$$A_k = \bigcup_{i=1}^{N(k)-1} C^k_h, \quad \forall k,$$

where $A_k$ is the set of accessible files by UT $k \in U$ and is defined in [3]. The set of coefficients $\omega^B_S$ for all UTs $k \in U$ is denoted by $\Omega^B_S$. Files belonging to the library $F$ are denoted by $f_1, f_2, \ldots, f_N$ and each file $f_n$ has popularity $P_n$. The popularity distribution for all files in the library $F$ is denoted by $P$. Let the problem instance be denoted by $HLP(U, H, E, F, \Omega^B_S, M, Q)$. It is easy to see that $HLP(U, H, E, F, \Omega^B_S, M, Q)$ is in the class NP. To show NP-hardness we will use a reduction from the following NP-complete problem.

Problem 2: (2-Disjoint Set Cover Problem) Consider a bipartite graph $G = (A, B, E)$ with edges $E$ between two disjoint vertex sets $A$ and $B$. For every $B \in B$, define a subset $\mathcal{N}(B)$ (neighborhood of $B$), clearly $A = \bigcup_{B \in B} \mathcal{N}(B)$. Do there exist two disjoint sets $B_1, B_2 \subset B$ such that $|B_1| + |B_2| = |B|$ and $A = \bigcup_{B \in B_1 \cup B_2} \mathcal{N}(B)$. Let us denote the problem instance by 2DSC($A, B, E$).

The above problem for finding the 2-disjoint cover is NP-complete [12]. We show in the following lemma that given a unit time oracle for Helper Decision Problem we can solve the 2-Disjoint Set Cover Problem in polynomial time (a polynomial time reduction is denoted by $\leq_L$).

Lemma 1: 2-Disjoint Set Cover Problem $\leq_L$ Helper Decision Problem

Proof: Consider an oracle that can solve any problem instance $HLP(\ldots)$ in unit time. Then solving an instance of 2DSC($A, B, E$) is equivalent to solving $HLP(A, B, E, \{f_1, f_2\}, \{1, \frac{1}{1+\epsilon}, \frac{1}{1+\epsilon^2}, \ldots\}, \{1, 1, \ldots, 1\}, K)$ where $|A| = K$ and $\epsilon < 1$.

Consider $A$ to be the set of UTs, $B$ to be the set of helpers and $E$ to be the edges representing connections between UTs and helpers. There are only two files $f_1$ and $f_2$ with popularity $P_1$ and $P_2$.
that might be placed in the cache of helper.

The optimum way of placing files in the helpers' caches. Every file

the helpers caching $f_1$ and helpers caching $f_2$ (both being disjoint since every helper can have either file only) cover the entire set of users $A$. This means there exist 2 disjoint set covers. An illustration is provided in Figure 4.

Conversely, if two disjoint set covers exist, then one can assign $f_1$ to the first set cover and assign $f_2$ to the second set cover and the $HLP$ instance will be satisfied since the sum of value seen by all users will be $K$.

\[ S = \{ f_1^1, f_1^2, \ldots, f_1^i, f_2^1, f_2^2, \ldots, f_2^j \}. \]  

(5)

The ground set can be partitioned into $H$ disjoint sets, $S_1, \ldots, S_H$ where $S_h = \{ f_1^1, f_2^1, \ldots, f_k^i \}$ is the set of all files that might be placed in the cache of helper $h$.

**Lemma 2:** The constraints in (2) can be written as partition matroid on the ground set $S$ defined in (5).

**Proof:** In the optimization problem (2), we want to find the optimum way of placing files in the helpers' caches. Every way of content placement can be expressed by a set $X \subseteq S$, called the placement set. So, for example if $f_n^h \in X$, the file $n$ is placed in the cache of helper $h$. A set of elements which are placed in the cache of helper $h$ are equal to $X_h \triangleq X \cap S_h$ where $S_h$ is a subset of the ground set $S$ associated to the helper $h$. So, the constraints on the cache capacity of helpers can be expressed as $X \subseteq I$ where

\[ I = \{ X \subseteq S : |X \cap S_h| \leq M, \ \forall \ h = 1, \ldots, H \}. \]  

(6)

Comparing $I$ in (6) and the definition of the partition matroid in (20), we can see that our constraints form a partition matroid with $l = H$ and $k = M$ for $i = 1, \ldots, H$. The partition matroid is denoted by $M = (S, I)$.

Having proven that the constraints form a matroid, we can rewrite the average delay of user $k$ as a function of placement set $X$.

\[ \bar{D}_k = \omega_k^1 \sum_n P_n X_n^k (n) + \omega_k^2 \sum_n P_n (X_n^h (n) \land \bar{X}_n^h (n)) + \omega_k^3 \sum_n P_n (X_n^h (n) \land \bar{X}_n^h (n)) + \ldots + \omega_{BS}^k \sum_n P_n (\bar{X}_n^h \ldots \land \bar{X}_n^{h_{k-1}}), \]  

(7)

where $\land$ is the 'AND' operation. $X_n^h$ is the boolean representation of $X_h = X \cap S_h$ and $\bar{X}_n^h$ is the complement of $X_h$. The $n$th element of $X_n^h$ is denoted by $X_n^h (n)$ which means that if the element $f_n^h$ is in the set $X_h$, the $n$th element of vector $X_n^h$ is $1$; otherwise, it is $0$.

**Lemma 3:** The objective function in the optimization problem in (2) is a monotone submodular function.

**Proof:** We first prove that the objective function is a submodular function. Since the sum of submodular functions is submodular, it is enough to prove that for a user $k$ the set function $F_k \triangleq \omega_{BS}^k - \bar{D}_k$ is a submodular function. We show that the marginal value of adding a new file to an arbitrary helper $h_j^k$ decreases as the placement set becomes larger. The marginal value of adding a new element to a placement set $X$ is the amount of increase in the set function due to the addition.

Let’s consider two placement sets $X_1$ and $X_2$ where $X_1 \subset X_2 \subset S$. We add an element $f_n^h \in S \setminus X_2$ to both placement sets. In other words, we add file $n$ to helper $h_j^k$. To prove that set function $F_k$ is submodular, we should distinguish between two cases. In the first case, according to the placement set $X_2$, user $k$ gets file from helper $h_j^k$ where $j < i$. Then, the marginal value of element $f_n^h$ is zero. For the placement set $X_1$ user $k$ downloads file from helper $h_j^k$, where $j' \geq j$. If $j' < i$, again the marginal value will be zero. However, when $j' > i$, by adding file $n$ to helper $h_j^k$ user $k$ downloads the file from the nearest helper $h_j^k$. So the marginal value of element $f_n^h$ will be $-P_n \omega_n^h + P_n \omega_n^{h_j^k} > 0$.

In the second case, according to placement set $X_2$, user $k$ accesses file $n$ through helper $h_j^k$ where $j > i$. So the marginal value of adding element $f_n^h$ is $-P_n \omega_n^h + P_n \omega_n^{h_j^k}$. Since in the placement set $X_1$, user $k$ downloads the file from helper $h_j^k$ where $j' \geq j$, the marginal value for the placement set $X_1$ which is equal to $f_n^h$ is $-P_n \omega_n^h + P_n \omega_n^{h_j^k}$ is equal and greater than the marginal value for the larger placement set $X_2$.

A greedy algorithm is a quite common way of maximizing a submodular function subject to a matroid constraint. The greedy algorithm starts with an empty set; at each step, it adds one element with the highest marginal value to the set while maintaining independence of the solution. The precise greedy algorithm is stated below. Classical results on approximations of submodular functions [13] established that the greedy algorithm achieves a performance that is provably within a factor $\frac{1}{2}$ of the optimal value.
In table I, with some abuse of notation, \( f(.) \) is the objective function in (7). As defined in (22), \( f_X(d) = f(X + d) - f(X) \) is the marginal value of an element \( d \) with respect to a placement set \( X \). At every iteration, the greedy algorithm selects the element \( f_{\alpha}^\beta \) with the highest marginal value. Selecting the element \( f_{\alpha}^\beta \) means that the file \( \alpha \) is cached by the helper \( \beta \). It can be seen that the marginal value of any element with respect to any placement set is greater than or equal to zero. Moreover, because the objective function is submodular, the marginal value of elements decreases as we add more elements to the placement set. Thus, if at one iteration, the marginal value of a selected element is zero, the marginal value of the next selected elements would be zero. Hence, provided that the marginal value of the selected element is zero at one iteration, the algorithm should stop. The selected element is chosen from all elements in set \( D \). The initial value of set \( D \) is the ground set \( S \). At every iteration, we remove the selected element \( f_{\alpha}^\beta \) from the set \( D \). If the cache of a helper \( \beta \) becomes full, we remove the entire set \( D_{\beta} \), the subset of \( D \) whose elements are all associated to helper \( \beta \), from the set \( D \). So, elements of the helper \( \beta \) will not be considered in subsequent iterations for selecting the element with the highest marginal value. The placement set \( X \) and a set of elements cached by each helper \( h \), i.e., \( X_h \), are also updated in each iteration. At the end, the greedy algorithm returns the placement set \( X \).

For maximization of a monotone submodular subject to matroid constraints, a randomized algorithm which gives a \( (1 - 1/e) \)-approximation has been proposed in [14]. This algorithm has two parts. In the first part, the combinatorial problem is replaced with the continuous one and the approximate solution of the continuous problem is found. In the second part, the fractional solution of the continuous problem is rounded using a technique called Pipage rounding [14]. Although this algorithm gives a better performance guarantee, when the size of the ground set, equal to \( H \times N \) in our problem, becomes large, it becomes too computationally demanding to implement.

### IV. Coded Distributed Caching Placement

In this section, we want to find the optimum way of placing fountain/MDS-encoded files in the helpers’ caches to minimize the total average delay of all users.

With fountain/MDS codes, user \( k \) should receive \( B \) coded symbols of file \( n \) in the union of the caches of transmitters in \( N(k) \). If the first \( j \leq |N(k)| \) transmitters in set \( N(k) \) are enough for the recovery of some file \( n \) while user \( k \) cannot recover the file by just receiving the parity symbols of the first \( j - 1 \) transmitters, after normalizing the number of coded symbols to the common file size \( B \), the delay for user \( k \) because of file \( n \) is equal to:

\[
\bar{D}_k^n = \sum_{h \in \{h_1^k, \ldots, h_{j-1}^k\}} \omega_k^h \rho_{hn} + (1 - \sum_{h \in \{h_1^k, \ldots, h_{j-1}^k\}} \rho_{hn}) \omega_j^k;
\]

where \( \rho_{hn} \) is the amount of coded symbols for file \( n \) in the cache of helper \( h \). Clearly \( \rho_{BSn} \) is equal to 1 for all \( n \). (1 - \( \sum_{h \in \{h_1^k, \ldots, h_{j-1}^k\}} \rho_{hn} \)) is the amount of parity symbols that should be transmitted by the \( j \)th transmitter in \( N(k) \). The above expression can be used if \( \sum_{h \in \{h_1^k, \ldots, h_{j-1}^k\}} \rho_{hn} < 1 \) and \( \sum_{h \in \{h_1^k, \ldots, h_{j-1}^k\}} \rho_{hn} \geq 1 \). The matrix whose element in the \( h \)-th row and \( n \)-th column is \( \rho = [\rho_{hn}] \), and it is referred to as the placement matrix. Because of the normalization, all the elements of placement matrix are in the range \([0, 1]\).

The delay \( \bar{D}_k^n \) incurred by user \( k \) because of downloading file \( n \) is a piecewise-defined affine function of the elements of the placement matrix \( \rho \).

We have the following result:

**Lemma 4**: \( \bar{D}_k^n \) is a convex function of \( \rho \).

**Proof**: A function that is pointwise maximum of finite number of affine functions is convex [15]. Now we show that:

\[
\bar{D}_k^n = \max_{j \in \{1, 2, \ldots, |N(k)|\}} \bar{D}_k^{n,j}.
\]

This means that if for some given \( j \), \( \sum_{h \in \{h_1^k, \ldots, h_{j-1}^k\}} \rho_{hn} < 1 \) and \( \sum_{h \in \{h_1^k, \ldots, h_{j-1}^k\}} \rho_{hn} \geq 1 \), and therefore \( \bar{D}_k^n = \bar{D}_k^{n,j} \), we have \( \bar{D}_k^{n,j} > \bar{D}_k^{n,i} \forall i \neq j \). Thus, from (8), we should show that:

\[
\sum_{h \in \{h_1^k, \ldots, h_{j-1}^k\}} \omega_k^h \rho_{hn} + (1 - \sum_{h \in \{h_1^k, \ldots, h_{j-1}^k\}} \rho_{hn}) \omega_j^k > \sum_{h \in \{h_1^k, \ldots, h_{j-1}^k\}} \omega_k^h \rho_{hn} + (1 - \sum_{h \in \{h_1^k, \ldots, h_{j-1}^k\}} \rho_{hn}) \omega_i^k,
\]

where the first and second expressions in the above equation are respectively \( \bar{D}_k^{n,j} \) and \( \bar{D}_k^{n,i} \). We only discuss the case \( i > j \) since the proof is similar for the case \( i < j \). After some manipulations, the above expression for \( i > j \) can be written as:
(\omega^k - \omega^h) \left( \sum_{h \in \{h_1^k, \ldots, h_{k-1}^k\}} \rho_{hn} - 1 \right) + \sum_{h \in \{h_{k+1}^k, \ldots, h_{k-1}^k\}} (\omega^k - \omega^h) \rho_{hn} > 0. \tag{11}

Since \( \sum_{h \in \{h_1^k, \ldots, h_{k-1}^k\}} \rho_{hn} \geq 1 \) and \( \omega^k > \omega^h \) for \( h \in \{h_1^k, \ldots, h_{k-1}^k\} \), the above expression is always true. \[ \blacksquare \]

The average delay of user \( k \) can be written as:

\[ D_k = \sum_{n=1}^{N} P_n \tilde{D}_k^n, \tag{12} \]

where \( P_n \) is the probability of requesting file \( n \). From [12] and [9], the expected delay of all users is:

\[ \bar{D} = \sum_{k=1}^{K} \sum_{n=1}^{N} P_n \max_{j \in \{1, 2, \ldots, |N(k)|\}} \tilde{D}_k^{n,j}. \]

Thus, the placement optimization problem takes on the form:

\[
\begin{align*}
\text{minimize} & \quad \sum_{k=1}^{K} \sum_{n=1}^{N} P_n \max_{j \in \{1, 2, \ldots, |N(k)|\}} \tilde{D}_k^{n,j} \\
\text{subject to} & \quad \sum_{n=1}^{N} \rho_{hn} \leq M, \quad \forall h \\
& \quad 0 \leq \rho_{hn} \leq 1, \quad \forall h, n, \tag{13}
\end{align*}
\]

where the optimization is with respect to \( \rho \). The first constraint indicates the cache capacity of each helper is equal to \( M \) files. The second constraint shows the amount of parity symbols of file \( n \) cached by a helper \( h \) is normalized to the size of files. The objective function in [13] is convex function since it is the weighted sum of convex piecewise linear functions [15]. Hence, we have minimization of convex function over linear constraints which can be solved using standard convex optimization. The number of involved variables in the optimization problem is quite large. However, some tricks and approximations (like bundling of video files with similar properties into groups that form the basis of optimization) can alleviate these problems. Furthermore, the optimization can be solved in a distributed way [16].

We can also solve the placement optimization problem by converting it to a linear program. To do this we introduce new variables and new constraints as follows:

\[
\begin{align*}
\text{minimize} & \quad \sum_{k=1}^{K} \sum_{n=1}^{N} P_n t_k^n \\
\text{subject to} & \quad D_k^{n,j} + s_k^{n,j} = t_k^n, \quad \forall k, n, j \in \{1, 2, \ldots, |N(k)|\} \\
& \quad s_k^{n,j} \geq 0, \quad \forall k, n, j \in \{1, 2, \ldots, |N(k)|\} \\
& \quad \sum_{n=1}^{N} \rho_{hn} \leq M, \quad \forall h \\
& \quad 0 \leq \rho_{hn} \leq 1, \quad \forall h, n, \tag{14}
\end{align*}
\]

where \( t_k^n = \max_{j \in \{1, 2, \ldots, |N(k)|\}} D_k^{n,j} \) and the optimization is with respect to \( \rho, t_k^n \)s and \( s_k^{n,j} \).

V. EXPERIMENTAL EVALUATION

In this section, evaluation of the coded and the uncoded cache placement optimizations through simulation in a cell based on 3GPP LTE Release 8 is presented. We assume a macro BS operating with a conventional proportional fair scheduling policy and helpers with some storage capacity serving users using WiFi-like links. We simulate the system with realistic user request pattern and evaluate the benefits of the femtocaching architecture in the uncoded and coded cache placement case, in terms of the achieved average file downloading delay.

We assume a single BS serving a circular-shaped cell with radius 400m (typical of urban macro cell [17]) and random statistically independent user positions, uniformly distributed over the cell. The pathloss function (in dB) is taken to be:

\[ PL(d(u,v)) = \begin{cases} 38 + 20 \log_{10}(d(u,v)) & , d(u,v) < 40 \\ 38 + 20 \log_{10}(40) + 35 \log_{10}(d(u,v)/40) & , d(u,v) > 40 \end{cases} \tag{15} \]

where \( d(u,v) \) denotes the distance, in meters, between the transmitter located at \( u \) and the receiver located at \( v \) where \( u, v \in \mathbb{R}^2 \). Let the ratio between the BS signal power and the noise power spectral density at the transmitter and receiver be \( g(u,v) \) and \( G_0 \) respectively. Then \( g(u,v) \) is given by:

\[ 10 \log_{10}(g(u,v)) = 10 \log_{10}(G_0) - PL(d(u,v)). \tag{16} \]

In a multi-cell scenario, users experience about \(-1\) to \(-4\) dB of SINR at the cell edge, for a system with universal frequency reuse (frequency reuse 1) [2]. Hence, we fix \( G_0 \) such that received SNR at the cell edge is equal to 0 dB.

We assume that the wireless channel is frequency selective and make a standard block-fading approximation of the small-scale Rayleigh fading, such that the fading channel coefficient is constant over a block of a certain duration in time (coherence time of \( N_t \times S \) OFDM symbols) and over a certain bandwidth in frequency (coherence bandwidth of \( N_f \) sub-carriers). Furthermore, we assume that the fading coefficients are i.i.d. from block to block, and independent across the users. We assume an OFDM TDMA cell system closely inspired by 3GPP LTE release 8 [18]. Figure 5 shows the time-frequency structure and the resource allocation slot (in yellow) of the downlink channel. More details are given in Table I.

\begin{table}[ht]
\centering
\caption{Simulation parameters based on LTE specifications.}
\begin{tabular}{|l|l|}
\hline
Parameter of the System & Value Assigned \\
\hline
Usable System bandwidth & 18 MHz \\
Sub-carrier Bandwidth & 15 KHz \\
Parameter \( N_t \) & 7 OFDM symb. = 0.5 ms \\
Parameter \( N_f \) & 12 Sub-carriers = 180 KHz \\
Smallest Resource allocation slot & \( N_t \times N_f \times S = 16,800 \) OFDM symbs. \( 0.5 \) ms \\
Coherence time & 100 ms = 1400 OFDM symbs. \\
Actual resource allocation block & \( N_f \times N_t \times S = 16,800 \) OFDM symbs. \\
Frequency blocks for allocation & 100 \\
\hline
\end{tabular}
\end{table}
The assumed signal model between the BS (located at the origin) and user $k$, located at $u_k$, over time-frequency slot $(t,f)$, with $t = 0, 1, 2, \ldots$ and $f \in \{1, \ldots, F\}$, is given by

$$y_k(t,f) = \sqrt{g(u_k,0)H_{k,0}(t,f)}x_0(t,f) + z_k(t,f),$$

(17)

where $H_{k,0}(t,f) \sim \mathcal{CN}(0,1)$ is the Rayleigh fading coefficient on slot $(t,f)$, and $x_0(t,f) \in \mathbb{C}^T$, $y_k(t,f) \in \mathbb{C}^T$ and $z_k(t,f) \in \mathbb{C}^T$ denote the transmit, received and noise vectors, with $z_k(t,f) \sim \mathcal{CN}(0, I)$, independent across time-frequency. The achievable rate of user $k$ on slot $(t,f)$ is given by

$$R_k(t,f) = \log \left(1 + g(u_k,0)|H_{k,0}(t,f)|^2\right).$$

(18)

**A. Scheduling Policy used**

A UT is said to be “idle” if it has no active download, and "active" otherwise. Active UTs cannot place more than one request at a time. If they do, all the previous requests are dropped. This models the fact that very closely spaced requests could predict the popularity distribution from the past activity in the cell; for a discussion of this assumption see Sec. II. We derive another trace by superimposing the busiest four hours of the day, which accounts for 848 unique users and about 4600 requests, and form trace1. We assume we know a priori the popularity distribution (number of views vs rank of videos in terms of views) for the day from the data of the entire day. This is used for assigning popularity to the files requested. This presupposes sophisticated algorithms that could predict the popularity distribution from the past activity in the cell.

**B. Trace based User Requests**

Since the advantage of the femtocaching architecture over the conventional cellular (BS only) case depends very strongly on how “peaky” the popularity distribution is, we can make the femtocaching gain be arbitrarily large just by assuming a suitable popularity distribution. Therefore, instead of choosing an artificial distribution, we used a measured trace of YouTube requests from a study conducted on the University of Massachusetts’ Amherst campus in 2008 [9] [10]. The study records YouTube requests arising from the wired campus network for several days; we assume that there is no significant difference in the user behavior to downloads with wireless devices. To be more specific, we target a scenario where the BS is swamped with YouTube-like (short videos) requests and consider the average downloading delay as the metric of interest. We assume that all requests are of size 30 MB, which is reasonable assuming a screen size 640 × 360, flash encoding and a few minutes (≈ 3 − 4 min) playback time. The simulations, analysis and conclusions also hold (in a scaled form) for larger file sizes. All traces used in simulations contain unique users (based on unique IP addresses from trace data) who request files during the time period. For every user, a time series involving the exact time of request (in seconds) is created and the corresponding video file number (can be simulated exactly as per these traces).

We use the trace data [21] for the day 02/19/08. There are a few thousand unique users for the entire day. We choose the busiest four hours of the day, which accounts for 848 unique users and about 4600 requests, and form trace1. We assume we know a priori the popularity distribution (number of views vs rank of videos in terms of views) for the day from the data of the entire day. This is used for assigning popularity to the files requested. This presupposes sophisticated algorithms that could predict the popularity distribution from the past activity in the cell; for a discussion of this assumption see Sec. II.

We derive another trace by superposing the busiest four hours and creating a merged trace containing 1719 (unique user lists from each four hour period are added up) users and about 9600 requests. This is in line with the recent predictions of the rise in video requests in mobile traffic in the next few years. We call this merged trace as trace2.

The scheduling and request handling take place as per the system specifications in the previous subsections. We compare the performance of three systems.

1) **Baseline system- Only the BS serves the users.**
2) **Uncoded Helper system**
3) **Coded Helper system**

The helpers are placed uniformly in a square grid spanning the cellular region. We assume a simple model for the helper-user communication. Every helper has a range of 100m. The rate dies down from 30 Mbps at 10m to 3 Mbps at 100m and decay curve follows the curve for case of 2x2 MIMO and 20 MHz in Fig 5.9 of [22] but scaled down by a factor.

---

**Fig. 5.** A pictorial representation of the time-frequency allocation slots in OFDM/TDMA.
of 4 accounting for overheads and the fact that this rate is guaranteed irrespective of other users who may request from the same helper concurrently. Although simplified, the model can be justified by facts about the latest WiFi standard quoted [23]. When a user requests a video file, and if the file lies in part or entirely in any of the helpers within range of the user, then the BS directs a suitable helper, depending on the distance from the user, to serve the user with the appropriate fraction of the request and the user gets it from the helper. This helper-user communication model supplies average delay, i.e \( \omega_{k}^{h} \), as in [24], from helper \( h \) to user \( k \) based on distance as inputs to the coded and uncoded placement algorithms.

The inputs \( \omega_{k}^{BS} \), as in (7), to the uncoded and the coded algorithm are the time averaged download delays for the files downloaded by user \( k \) from the base station. Generating these inputs to the algorithms poses a problem. The actual average delay experienced by user \( k \), from the BS, cannot be known prior to the placement of files. Since the BS uses a scheduling algorithm, the delays are determined by the exact traffic pattern. Traffic pattern at the BS changes when helpers and placement of files are introduced. Further, it also requires knowing the exact traffic pattern from the traces before placement. In essence, it can be determined only after the placement is made and only knowing the exact traffic pattern through simulations. We make the following observation to circumvent the problem. In essence, both the algorithms described try to store the most popular files, but with more sophistication. So, average download delays from BS are computed by simulation done when BS serves users along with the same number of helpers, as in the optimization, but storing the most popular files (\( M \) most popular files where \( M \) is the cache capacity). Since exact traffic pattern information cannot be used, we use an analytical model for user requests for this simulation.

In this model, every user behaves as follows: users when downloading do not make any additional request at all. The wait time between two requests is a random variable with a geometric distribution with parameter \( p \) [24]. The probability of success (probability of requesting when idle) is \( p \) every second. The expected wait time is \( 1/p \). \( p \) can be chosen to match the traffic intensity in the trace (No of requests and No of simultaneous active users per hour). This analytical model captures the 'realistic' coupling between the current service and future requests. That is, users who have poor service would request less frequently. Note, this does not use the exact user request pattern of the traces which means the inputs to the placement algorithm only needs some rough estimates of traffic intensity for choice of the parameter \( p \). We have chosen \( p = 0.003 \) and 300 users drawing requests i.i.d from the popularity distribution to generate the average delay values for the BS for our simulations. This choice of \( p \) represents reasonably high traffic intensity, i.e a significant fraction of the total number (300) of users being simultaneously active.

To compare the baseline and the helper systems, we define for every user, the Quality of Service (QoS) as the average download delay. A streaming user is satisfied if its average downloading delay is less than its playback time. This means that by allowing some pre-buffering and thanks to the playback buffer which absorbs the statistical fluctuations, the video can be watched without stall till the end. In all simulations, \( H \) represents the number of helpers, \( M \) represents the number of 30 MB files that a helper can cache. In some plots, storage capacity in GB is mentioned and \( M \) can be inferred from it. QoS is chosen to be 100 seconds and 200 seconds to measure satisfied users. Such a QoS is reasonable for a playback time of 3 minutes for the videos.

In general, we observe that the Greedy Helper system is strictly better than the Popular Helper System, which in turn is better than the Baseline. The number of satisfied users for the Coded and Uncoded Helper systems are very close. This indicates that helpers can store the entire files without using the fountain code and we still have a significant gain.

Figures 6 and 7 show the number of satisfied users versus the storage capacity of helpers. Figure 6 uses trace1 while Figure 7 uses trace2. As expected, increasing the storage capacity of each helper enhances the number of satisfied users. The difference between the Baseline system and the Helper systems in trace2 is larger than that in trace1. This indicates the significant advantages of helpers, in particular, in case of a large number of video requests. Thus, in order to have reasonable performance for large number of video requests, the use of helpers is more crucial.

Figures 8 and 9 show the number of satisfied users versus QoS for trace1 and trace2, respectively, with \( H = 45 \) and \( M = 2000 \). In Figure 8, the number of satisfied users increases more than 400\% in both helper systems. Better improvement is seen in (Figure 9).

Figures 10 and 11 show the number of satisfied users versus the number of helpers for trace1 and trace2. Increasing the number of helpers increases the number of satisfied users.

Let us recall that some requests are dropped if a user requests too frequently. We now compare the systems on the basis of successful requests (requests that are not dropped). Figures 12 and 13 show the percentage of successful requests versus the download time for trace1 and trace2, respectively, with \( H = 32 \) and \( M = 3000 \). It can be seen that even for large download times, the percentage of successful requests does not reach 100\%. The reason is that, in trace data, most users become active for a short time in a day and within that short time they request several video files consecutively. They usually request another file before the previous one is downloaded completely. Since the requests which are not downloaded completely are considered as dropped requests, dropped requests always exist. However, as it can be seen from the plots, helpers reduce the percentage of dropped requests. So, by having helpers, not only do we see an increase in the number of satisfied users, but also we see a reduction in the number of dropped requests.

VI. CONCLUSION

In this paper we introduced a new method for increasing the throughput of wireless video delivery networks. The key idea is the use of a distributed cache, i.e., helper stations that store the most popular video files, and transmit them, upon request, via short-range wireless links to the user terminals. The caches are low-cost because storage capacity has become exceptionally cheap (according to recent prices, two terabytes
cost approximately 100 dollars), while the loading of the files to the caches can occur through a low-rate (and thus cheap and robust) backhaul links at low demand times. We then formulated and solved the problem of which files should be assigned to which helpers. Our numerical evaluation uses a realistic LTE-based cellular simulator and a real trace of YouTube requests and demonstrates performance improvements on the order of $400 - 500\%$ more users at reasonable QoS levels. Our conclusion is that a wireless distributed helper system seems to be a promising way of alleviating the bottlenecks in wireless video delivery.

**APPENDIX**

**Matroids:** Matroids are structures that generalize this concept of independence, as known from linear algebra, to general sets. Informally, we need a finite ground set $S$ and a matroid is a way to label some subsets of $S$ as “independent”. In vector spaces, the ground set is a set of vectors, and subsets are called independent if their vectors are linearly independent, in the usual linear algebraic sense. Formally, we have [25]:

**Definition 1:** A matroid $\mathcal{M}$ is a tuple $\mathcal{M} = (S, \mathcal{I})$, where $S$ is a finite ground set and $\mathcal{I} \subseteq \mathcal{P}(S)$ (the power set of $S$) is a collection of independent sets, such that:
1. $\mathcal{I}$ is nonempty, in particular, $\emptyset \in \mathcal{I}$,
2. $\mathcal{I}$ is downward closed; i.e., if $Y \in \mathcal{I}$ and $X \subseteq Y$, then $X \in \mathcal{I}$,
3. If $X, Y \in \mathcal{I}$, and $|X| < |Y|$, then $\exists y \in Y \setminus X$ such that $X \cup \{y\} \in \mathcal{I}$. □

One example is the partition matroid. In a partition matroid, the ground set $S$ is partitioned into (disjoint) sets $S_1; S_2; \ldots; S_l$ and

$$\mathcal{I} = \{X \subseteq S : |X \cap S_i| \leq k_i \text{ for all } i = 1 \ldots l\},$$

for some given parameters $k_1, k_2, \ldots, k_l$. 

![Fig. 6. The number of satisfied users versus storage capacity of each helper for trace1, $H = 32$ and QoS=$100s$.](image1)

![Fig. 7. The number of satisfied users versus storage capacity of each helper for trace2, $H = 32$ and QoS=$200s$.](image2)

![Fig. 8. The number of satisfied users versus QoS for trace1, $H = 45$ and $M = 60GB$.](image3)

![Fig. 9. The number of satisfied users versus QoS for trace2, $H = 45$ and $M = 60GB$.](image4)

![Fig. 10. The number of satisfied users versus the number of helpers for trace1, $H = 32$ and QoS=$100s$.](image5)

![Fig. 11. The number of satisfied users versus the number of helpers for trace2, $H = 32$ and QoS=$200s$.](image6)
Submodular functions: Let $S$ be a finite ground set. A set function $f : 2^S \rightarrow \mathbb{R}$ is submodular if for all sets $A, B \subseteq S$,

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B). \quad (21)$$

Equivalently, submodularity can be defined by the following condition. Let $f_A(i) = f(A + i) - f(A)$ denote the marginal value of an element $i \in S$ with respect to a subset $A \subseteq S$. Then, $f$ is submodular if for all $A \subseteq B \subseteq S$ and for all $i \in S \setminus B$ we have:

$$f_A(i) \geq f_B(i). \quad (22)$$

Intuitively, submodular functions capture the concept of diminishing returns: as the set becomes larger the benefit of adding a new element to the set will decrease. Submodular functions can also be regarded as functions on the boolean hypercube $\{0, 1\}^{|S|} \rightarrow \mathbb{R}$. Every set has an equivalent boolean representation by assigning 1 to the elements in the set and 0 to other ones. We denote the boolean representation of a set $X$ by a vector $X^b \in \{0, 1\}^{|S|}$.

The function $f$ is monotone if for $A \subseteq B \subseteq S$, we have $f(A) \leq f(B)$.

Fig. 12. The percentage of successful requests versus QoS for trace1, $H = 32$ and $M = 90GB$.

Fig. 13. The percentage of successful requests versus QoS for trace2, $H = 32$ and $M = 90GB$.

REFERENCES

[1] “http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white/paper/e11-520862.html.”
[2] A. F. Molisch, Wireless communications. IEEE Press - Wiley, 2011.
[3] V. Chandrasekhar, J. Andrews, and A. Gatherer, “Femtocell networks: a survey,” Communications Magazine, IEEE, vol. 46, no. 9, pp. 59–67, 2008.
[4] A. Shokrollahi, “Raptor codes,” Information Theory, IEEE Transactions on, vol. 52, no. 6, pp. 2551–2567, 2006.
[5] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, “Web caching and zipf-like distributions: Evidence and implications,” in INFOCOM’99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, vol. 1. IEEE, 1999, pp. 126–134.
[6] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, “Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web,” in Proceedings of the twenty-ninth annual ACM symposium on Theory of computing. ACM, 1997, pp. 654–663.
[7] M. Rabinovich and O. Spatscheck, “Web caching and replication,” SIGMOD Record, vol. 32, no. 4, p. 107, 2003.
[8] S. El Rouayheb, A. Sprintson, and C. Georgiades, “On the index coding problem and its relation to network coding and matroid theory,” Information Theory, IEEE Transactions on, vol. 56, no. 7, pp. 3187–3195, 2010.
[9] M. Zink, K. Suh, Y. Gu, and J. Kurose, “Watch global, cache local: YouTube network traffic at a campus network-measurements and implications,” Proceeding of the 15th SPIE/ACM Multimedia Computing and Networking (MMCN08), 2008.
[10] ——, “Characteristics of youtube network traffic at a campus network-measurements, models, and implications,” Computer Networks, vol. 53, no. 4, pp. 501–514, 2009.
[11] S. Borst, “User-level performance of channel-aware scheduling algorithms in wireless data networks,” in INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies, vol. 1. IEEE, 2003, pp. 321–331.
[12] M. Cardei and D. Du, “Improving wireless sensor network lifetime through power aware organization,” Wireless Networks, vol. 11, no. 3, pp. 333–340, 2005.
[13] G. Nemhauser, L. Wolsey, and M. Fisher, “An analysis of approximations for maximizing submodular set functions,” Mathematical Programming, vol. 14, no. 1, pp. 265–294, 1978.
[14] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák, “Maximizing a submodular set function subject to a matroid constraint,” Integer programming and combinatorial optimization, pp. 182–196, 2007.
[15] M. Neely, “Stochastic Network Optimization with Application to Communication and Queueing Systems,” Synthesis Lectures on Communications and Networking (MMCN08), 2008.
[16] L. Georgiadis, M. Neely, M. Neely, and L. Tassiulas, Resource allocation and cross-layer control in wireless networks. Now Pub, 2006.