FemtoCaching: Wireless Video Content Delivery through Distributed Caching Helpers

Negin Golrezaei, 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 suggest a novel approach to handle the ongoing explosive increase in the demand for video content in wireless/mobile devices. We envision femtocell-like base stations, which we call *helpers*, with weak backhaul links but large storage capacity. These helpers form a wireless distributed caching network that assists the macro base station by handling requests of popular files that have been cached. Due to the short distances between helpers and requesting devices, the transmission of cached files can be done very efficiently.**

A 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 a exactly one helper, then clearly each helper should cache the same files, namely the most popular ones. However, for the case that each mobile device can access multiple caches, the assignment of files to helpers becomes nontrivial.

The theoretical contribution of our paper lies in (i) formalizing the distributed caching problem, (ii) showing that this problem is NP-hard, and (iii) presenting approximation algorithms that lie within a constant factor of the theoretical optimum.

On the practical side, we present a detailed simulation of a university campus scenario covered by a single 3GPP LTE R8 cell and several helpers using a simplified 802.11n protocol. We use a real campus trace of video requests and show how distributed caching can increase the number served users by as much as $400-500\%$.

I. Introduction

Data traffic to mobile wireless devices is predicted to increase by *two orders of magnitude* in the next few years (e.g., [1]). This explosive demand is fueled mainly [1] by demand for delivery of video content to Internet-capable smartphones and other portable devices. One of the most promising ways to achieve data throughput increase of such tremendous scale is a decrease in cell size, essentially bringing the content closer to the users. This is achieved 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 effectiveness of this approach is often impaired by the lack of cost-effective backhaul connectivity of these small base stations to the cellular operator network [3].

This work was supported, in part, by NSF Career Grant 1055099 and a Research Gift by Intel.

In this paper we suggest a new way to deal with the backhaul problem by replacing the femto base stations by *s*mall base stations that have a low-bandwidth backhaul link but high storage capacity. These stations, which we henceforth call caching helpers, or simply helpers, form a wireless distributed caching infrastructure. 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 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 and hence frequency reuse. 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*. This occurs because the most popular files are stored in the cache, and are thus always available locally to the UTs that are requesting it. 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 operating in conjunction with a traditional, macrocellular base station. For the sake of conceptual simplicity, we consider a single cell, equipped with a macro base station (BS), serving a large number of UTs with the help of dedicated helpers. If a UT requests a file that is cached in local helpers, the helpers handle the request; the macro BS manages the requests that cannot be handled locally. 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. The central question is, of course, how much gain we can expect in real systems. In this paper, with realistic setups the number of users that can be served is increased by as much as $400-500\%$.

If the distance between helpers is large, and each UT can connect only to a single helper, it is obvious that 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, as we will demonstrate.

The contribution of this work is three-fold:

- 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. We prove that our 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 optimal placement is NP-complete. Further, we express the distributed caching problem as a maximization of a submodular function subject to matroid constraints (see also [4] for other applications of matroid theory in networks). This formulation allows us to use approximation results for submodular functions to show that a simple greedy algorithm is within a factor of $1/2$ from optimality. A more complicated algorithm with a $1-1/e$ factor of approximation is also discussed.
- On the practical side we present a detailed simulation of a university campus scenario covered by a single 3GPP LTE R8 cell and several helpers using a simplified 802.11n protocol. We simulate using the user request trace of YouTube videos from the Amherst campus [5] [6]. Our main finding is that there are very significant gains even when very simple caching algorithms are used.

II. Theoretical analysis

A. Distributed caching placement model and assumptions

We consider a system where video files are requested randomly by the users. The users' requests are redundant, i.e, they may request the same file, at different times, according to some popularity distribution.

We note that there is a substantial amount of prior work on caching algorithms for web and video content, see e.g. [7], [8], [9] 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.

In particular, 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 in the reach. 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 1. There are two helpers and four UTs in figure 1. The dashed circles centered around helpers indicate the transmission radius of helpers, i.e., users within a dashed circle can communicate locally with the corresponding helper. 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.

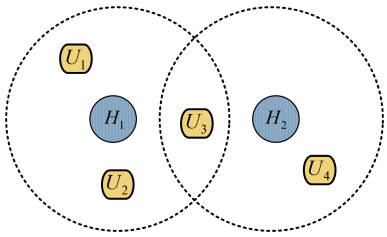

Fig. 1. Distributed Caching example: two helpers H_1, H_2 and four users with conflicting interests.

In our formulation, we make the following assumptions:

- The optimal content placement is determined centrally by the BS according to the optimization algorithms discussed later.
- 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. Invoking a time-scale decomposition, this has two important consequences: (i) 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. (ii) once the optimal content placement is determined, the operation of actually populating the helpers' caches can take place using weak backhaul links, since the cost of refreshing the helpers' content can be safely neglected.
- For our theoretical analysis, we assume that the communication from the helpers to the UTs is *instantaneous*. This leads to a clean optimization that relies on the assumption that the helpers provide much higher rates than the macrocellular link. In our experimental section we lift this assumption and evaluate our caching policies in a realistic rate regime.
- 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. We

also assume that files are either completely stored or not stored at all.
- 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.

There are H helpers, K user terminals, and a library of N files, denoted by $\mathcal{F}$. 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.

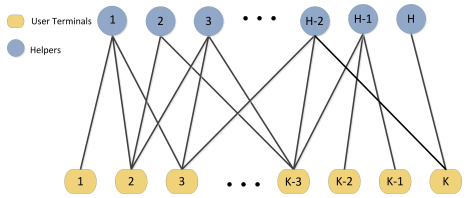

Fig. 2. Example of a connectivity bipartite graph indicating how UTs are connected to helpers

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. If these local helpers have the requested file, they send it; otherwise, the BS handles the request, incurring a higher delay.

It can be observed and shown analytically that under the standard proportional fairness downlink for high-speed data (HSDPA/Ev-Do) in 3GPP, also used in LTE R8, 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. Exact analysis of the proportional fairness scheduler is difficult. So all the above conclusions can be drawn analytically to hold under some approximation to proportional fairness scheduling [10]. Since the delay of local transmission between users and helpers is assumed to be zero, the average delay of UT k is proportional to the probability of not finding the request file in the union of the helpers' caches in the reach of user k, i.e., the helpers in the neighbourhood of user k in the connectivity graph. Therefore, after suitable normalization, the expected delay of user k can be written as:

$$\bar{D}_k = \omega_k[1 - \sum_{n \in A_k} P_n], \tag{1}$$

where ω_k is the delay of downloading a file of fixed size from BS for user k and A_k is the union of the helpers' caches in the neighbourhood of user k in the connectivity graph of the network, i.e., $A_k = \cup_{h \in \mathcal{N}(k)} C_h$. C_h is the set of files in the cache of helper h and $\mathcal{N}(k)$ denotes the neighbourhood of user k, i.e., all the helpers which user k can reliably connect to. In this way, $\sum_{n \in A_k} P_n$ is the probability that user k requests one of the files that are accessible through its local helpers.

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. Notice that the same file can be placed several times in different helpers. From (1), it can be seen that for each user, minimizing its delay is equivalent to maximizing the probability of finding the desired file through its local helpers. Thus, the optimization problem can be written as :

$$\begin{aligned} \text{maximize} \quad & \sum_{k=1}^{K} \omega_k \sum_{n \in A_k} P_n \\ \text{subject to} \quad & |C_h| \leq M, \quad \forall\, h \\ & A_k = \cup_{h \in \mathcal{N}(k)} C_h, \quad \forall\, k \end{aligned} \tag{2}$$

where the optimization is with respect to the sets $\{C_h \subseteq \mathcal{F} \ : \ \forall h\}$, M is the cache capacity of each helper and $|C_h|$ is the cardinality of set C_h. The constraint indicates that no more than M files are allowed to be placed in each helper's cache. The above objective function can be interpreted as the sum of *value* seen by each user. The value of each user k is equal to $\omega_k \sum_{n \in A_k} P_n$, which is proportional to the total popularity values of the union of files seen by user k through its neighborhood helpers. Thus, the more popular files a user k finds in its local helpers (in the set A_k) the larger value it sees. Our goal here is to maximize the sum of values seen by all users.

In the following, we will show that the above problem is NP-complete. Then, we formulate the problem as maximization of a monotone submodular function over matroid constraints. Moreover, we present an algorithm that gives results that are provably within a constant factor from optimality.

B. The distributed caching placement problem and computational intractability

To show that the optimization problem in (2) is NP-complete, we consider its corresponding decision problem, called Helper Decision Problem.

Problem 1: (*Helper Decision Problem*) Let's denote the connectivity graph in figure 2 by bipartite graph $G = (\mathcal{U}, \mathcal{H}, E)$ where $\mathcal{U}$ and $\mathcal{H}$ denote the sets of UTs and helpers, respectively, and E denotes the set of edges between elements (nodes) of $\mathcal{U}$ and $\mathcal{H}$. We ask the following question: given the graph $G = (\mathcal{U}, \mathcal{H}, E)$ and a library of files $\mathcal{F}$ and a real number $Q \geq 0$, does there exist any way of placing elements of $\mathcal{F}$ in the nodes of $\mathcal{H}$, such that $|C_h| \leq M$ for all $h \in \mathcal{H}$ and the objective function in (2) satisfies:

$$\sum_{k=1}^{K} \omega_k \sum_{n \in A_k} P_n \geq Q, \tag{3}$$

where A_k is the set of accessible files by UT $k \in \mathcal{U}$ and is defined in (2). The set of coefficients ω_k for all UTs

$k \in \mathcal{U}$ is denoted by Ω. Files belonging to the library $\mathcal{F}$ are denoted by $\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_N$ and each file $\mathbf{f}_n$ has popularity P_n. The popularity distribution for all files in the library $\mathcal{F}$ is denoted by $\mathbf{P}$. Let the problem instance be denoted by $HLP(\mathcal{U}, \mathcal{H}, E, \mathcal{F}, \mathbf{P}, \Omega, M, Q)$.

It is easy to see that $HLP(\mathcal{U}, \mathcal{H}, E, \mathcal{F}, \mathbf{P}, \Omega, M, Q)$ 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 $G = (\mathcal{A}, \mathcal{B}, E)$ with edges E between two disjoint vertex sets $\mathcal{A}$ and $\mathcal{B}$. For every $\mathbf{B} \in \mathcal{B}$, define a subset $\mathcal{N}(\mathbf{B})$ (neighborhood of $\mathbf{B}$), clearly $\mathcal{A} = \bigcup_{\mathbf{B} \in \mathcal{B}} \mathcal{N}(\mathbf{B})$. Do there exist two disjoint sets $\mathcal{B}_1, \mathcal{B}_2 \subset \mathcal{B}$ such that $|\mathcal{B}_1| + |\mathcal{B}_2| = |\mathcal{B}|$ and $\mathcal{A} = \bigcup_{\mathbf{B} \in \mathcal{B}_1} \mathcal{N}(\mathbf{B}) = \bigcup_{\mathbf{B} \in \mathcal{B}_2} \mathcal{N}(\mathbf{B})$. Let us denote the problem instance by $2DSC(\mathcal{A}, \mathcal{B}, E)$.

The above problem for finding 2-disjoint set cover is NP-complete [11]. We show in the following lemma that given a unit time oracle for *Helper Decision Problem* we can solve *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 $HLP(...)$ in unit time. Then solving an instance of $2DSC(\mathcal{A}, \mathcal{B}, E)$ is equivalent to solving $HLP(\mathcal{A}, \mathcal{B}, E, \{\mathbf{f}_1, \mathbf{f}_2\}, \{\frac{1}{1+\epsilon}, \frac{\epsilon}{1+\epsilon}\}, \{1, 1, ..., 1\}, 1, K)$ where $|\mathcal{A}| = K$ and $\epsilon < 1$.

Consider $\mathcal{A}$ to be the set of UTs, $\mathcal{B}$ to be the set of helpers and E to be the edges representing connections between UTs and helpers. There are only two files $\mathbf{f}_1$ and $\mathbf{f}_2$ with popularity $P_1 = \frac{1}{1+\epsilon}$ and $P_2 = \frac{\epsilon}{1+\epsilon}$. The user dependent coefficients Ω are assumed to be 1 for all UTs and the cache capacity of each helper is equal to 1. We check if the sum of value seen by all users can be greater than or equal to K. Since all the helpers $\mathbf{B}_i$ have capacity 1 they can either cache file $\mathbf{f}_1$ or $\mathbf{f}_2$. If the utility function in (3) is greater than or equal to K, then it has to be equal to K because every user can at most see 1. This can only happen if the helpers caching $\mathbf{f}_1$ and helpers caching $\mathbf{f}_2$ (both being disjoint since every helper can have either file only) cover the entire set of users $\mathcal{A}$. This means there exist 2 disjoint set covers. Illustration is provided in figure 3.

Conversely, if two disjoint set covers exist, then one can assign $\mathbf{f}_1$ to the first set cover and assign $\mathbf{f}_2$ to the second set cover and the HLP instance will be satisfied since the sum of value seen by all users will be K. ■

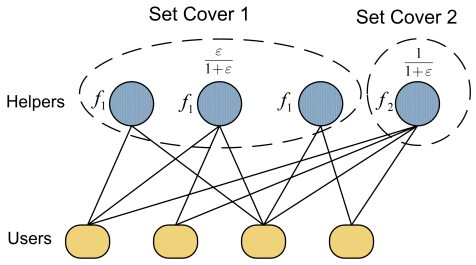

Fig. 3. Figure illustrating the reduction from 2-Disjoint Set Cover Problem

C. Expressing the placement as maximizing a monotone submodular function over matroid constraints

We start this section with some definitions. Subsequently, we show that the NP-complete optimization problem in (2) can be written as maximization of submodular function subject to matroid constraints.

1) Matroids: Linear independence is a well-known and useful concept. Matroids are structures that generalize this concept of independence, for 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 [12]:

Definition 1: A matroid $\mathcal{M}$ is a tuple $\mathcal{M} = (S, \mathcal{I})$, where S is a finite ground set and $\mathcal{I} \subseteq 2^S$ (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 $\exists y \in Y \backslash X$ 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 $S_1; S_2; ...; S_l$ and

$$\mathcal{I} = \{X \subseteq S : |X \cap S_i| \leq k_i \text{ for all } i = 1 \dots l\}, \quad (4)$$

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

2) Submodular functions: Let S be a finite ground set. A set function $f : 2^S \to \mathbb{R}$ is submodular if for all set $A, B \subseteq S$,

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B). \quad (5)$$

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 $i \in S \backslash B$ we have:

$$f_A(i) \geq f_B(i). \quad (6)$$

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 $\{0, 1\}^{|S|} \to \mathbb{R}$. Every set has an equivalent boolean presentation 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 $f(A) \leq f(B)$.

3) The original problem as the maximization of a submodular function subject to matroid constraints: In this part, we show that our constraints and our objective function are independent sets of a matroid and submodular function, respectively. First, we should define a ground set S. Denoting the placing of file n to the cache of helper h by the element $\mathbf{f}_n^h$, the ground set will be

$$S = \{\mathbf{f}_1^1, \mathbf{f}_2^1, ..., \mathbf{f}_N^1, ..., \mathbf{f}_1^H, \mathbf{f}_2^H, ..., \mathbf{f}_N^H\} \tag{7}$$

The ground set can be partitioned into H disjoint sets, $S_1, ..., S_H$ where $S_h = \{\mathbf{f}_1^h, \mathbf{f}_2^h, ..., \mathbf{f}_N^h\}$ is the set of all files that might be placed in the cache of helper h.

Lemma 2: The constraints in the (2) can be written as partition matroid on the ground set S defined in (7).

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 $X \subseteq \mathcal{I}$ where

$$\mathcal{I} = \{X \subseteq S : |X \cap S_h| \leq M, \ \ \forall\, h = 1, \ldots, H\}. \tag{8}$$

Comparing $\mathcal{I}$ in (8) and the definition of the partition matroid in (4), we can see that our constraints form a partition matroid with $l = H$ and $k_i = M$ for $i = 1, ..., H$. The partition matroid is denoted by $\mathcal{M} = (S, \mathcal{I})$. ∎

Having proven that the constraints form a matroid, we can rewrite the optimization problem in (2) as following:

$$\begin{aligned} \text{maximize} \ & \sum_{k=1}^{K} \omega_k \sum_{n=1}^{N} \mathbf{B}_X^k(n) P_n \\ \text{subject to} \ & X \in \mathcal{I} \\ & \mathbf{B}_X^k = \vee_{h \in \mathcal{N}(k)} X_h^b, \end{aligned} \tag{9}$$

where $\mathcal{I}$ is given in (8) and $X \in \mathcal{I}$ indicates that the number of files cached by each helper should not exceed the cache capacity M. X_h^b is the boolean representation of $X_h = X \cap S_h$. It means that if the element $\mathbf{f}_n^h$ is in the set X_h, the nth element of vector X_h^b is 1; otherwise, it is 0. $\mathbf{B}_X^k$ is a boolean vector associated to UT k with length of N (total number of files). The superscript X in $\mathbf{B}_X^k$ indicates that this vector is a function of the placement set X. The nth element of this vector, denoted by $\mathbf{B}_X^k(n)$, is equal to 1 if UT k has an access to the file n via at least one of helpers in its neighbourhood. This verifies the use of the component-wise 'or' operation $\vee$ in definition of $\mathbf{B}_X^k$.

Lemma 3: The objective function in the optimization problem in (9) is a monotone submodular function.

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 $\frac{1}{2}$ of the optimal value.

TABLE I
THE GREEDY ALGORITHM

algorithm **Greedy**

Initialize $D_h = S_h$ for h=1,...,H
$D = S$
$X_h \leftarrow \emptyset$ for h=1,...,H
$X \leftarrow \emptyset$
For $i = 1, 2, ..., M \times H$
$\mathbf{f}_\alpha^\beta = \arg\max_{d \in D} f_X(d)$
If $f_X(\mathbf{f}_\alpha^\beta) = 0$
break
EndIf
$X_\beta \leftarrow X_\beta + \mathbf{f}_\alpha^\beta$
$X \leftarrow X + \mathbf{f}_\alpha^\beta$
$D \leftarrow D \backslash \mathbf{f}_\alpha^\beta$
$D_\beta \leftarrow D_\beta \backslash \mathbf{f}_\alpha^\beta$
If $|X_\beta| = M$
$D \leftarrow D \backslash D_\beta$
EndIf
EndFor
Ouput X

In table I, with some abuse of notation, $f(.)$ is the objective function in (9). As defined in (6), $f_X(d) = f(X+d) - f(X)$ is the marginal value of an element d with respect to a placement set X. At every iteration, the greedy algorithm selects the element $\mathbf{f}_\alpha^\beta$ with the highest marginal value. Selecting the element $\mathbf{f}_\alpha^\beta$ means that the file α is cached by the helper β.

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 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 $\mathbf{f}_\alpha^\beta$ from the set D. If the cache of a helper β becomes full, we remove the entire set D_β, the subset of D whose elements are all associated to helper β, from the set D. So, elements of the helper β 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 function 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 gets too computationally demanding to implement.

III. Experimental evaluation

A. Simulation System Parameters

In this section, evaluation of the greedy algorithm through simulation in a cell based on 3GPP LTE Release 8 is presented. We assume a macro BS operating with a conventional scheduling policy and helpers with some storage capacity serving users using WiFi-like links. We simulate the system with realistic user request pattern and average delays for downloading are obtained to evaluate the placement algorithm.

We assume a circular cell with no inter-cell interference from other cells. The cell has a radius of $400m$ and users are randomly and independently distributed in the cell with uniform probability distribution. The assumed cell radius is typical in a urban macro cell [15]. The pathloss function (in dB) is taken to be:

$$PL(d(u,v)) = \left\{ \begin{array}{c} 38 + 20 * \log_{10}(d(u,v)) \quad , d(u,v) < 40 \\ 38 + 20 * \log_{10}(40) + \\ 35 * log_{10}\left(d(u,v)/40\right) \quad , d(u,v) > 40 \end{array} \right\} \tag{10}$$

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 received 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 the following equation:

$$10\log_{10}(g(u,v)) = 10\log_{10}(G_0) - PL(d(u,v)) \tag{11}$$

In a multi-cell scenario, users experience about -1dB to -4 dB of SINR at the cell edge, unless frequency reuse larger than 1 is used [2]. Since we don't consider any inter-cell interference, for realistic rates 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 [16]. Figure 4 shows the time-frequency structure and the resource allocation slot (in yellow) of the downlink channel. More details are given in Table II.

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

$$\mathbf{y}_k(t,f) = \sqrt{g(u_k,0)} H_{k,0}(t,f)\mathbf{x}_0(t,f) + \mathbf{z}_k(t,f), \tag{12}$$

where $H_{k,0}(t,f) \sim \mathcal{CN}(0,1)$ is the Rayleigh fading coefficient on slot (t,f), and $\mathbf{x}_0(t,f) \in \mathbb{C}^T$, $\mathbf{y}_k(t,f) \in \mathbb{C}^T$ and $\mathbf{z}_k(t,f) \in \mathbb{C}^T$ denote the transmit, received and noise vectors, with $\mathbf{z}_k(t,f) \sim \mathcal{CN}(0, \mathbf{I})$, independent across time-frequency. The achievable rate of user k on slot (t,f) is given by

$$R_k(t,f) = \log\left(1 + g(u_k,0)|H_{k,0}(t,f)|^2\right). \tag{13}$$

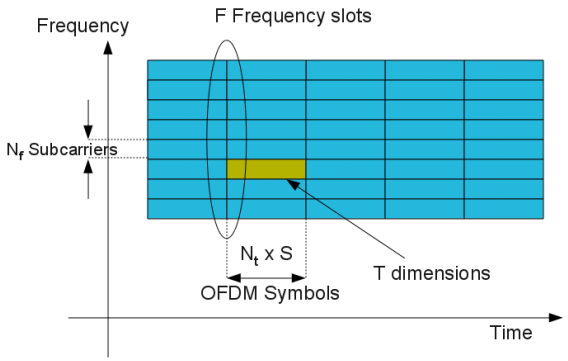

Fig. 4. A pictorial representation of the time-frequency allocation slots in OFDM/TDMA.

TABLE II
Simulation parameters based on LTE specifications.

Parameter of the System	Value Assigned
Usable System badnwidth	18 MHz
Sub-carrier Bandwidth	15 KHz
Parameter N_t	7 OFDM symbs. $= 0.5ms$
Parameter N_f	12 Sub-carriers $= 180KHz$
Smallest Resource allocation slot	$N_t = 7$ OFDM symbs. $\times$ $N_f = 12$ sub-carriers.
Coherence time	$100ms$ = 1400 OFDM symbs. $= S \times N_t$ OFDM symbs. across time
Parameter S	$\frac{100}{0.5} = 200$ slots (N_t OFDM symbs. each)
Actual resource allocation block	$N_t \times N_f \times S = 16800$ OFDM symbs.
Frequency blocks for allocation	100

B. 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 reflect a change of interest in the users.

We assume *Proportional Fairness Scheduling* (PFS), currently used in Ev-Do and HSDPA high-data rate downlink schemes in 3G [16], that assigns exactly one user to the frequency slots (TDMA constraint). Under the infinite backlog assumption, for all users the policy ensures that the system operates at a desirable point of the system ergodic rate region (defined by a utility function) [17] [18]. In our system, PFS policy is applied to active users only. Let $K_{ccrmon}(t)$ denote active users at time t. Assuming that $K_{\text{on}}(t)$ oscillates a little for a large number of slots (locally constant), it approximates an infinitely backlogged scenario with $K_{\text{on}}(t)$ active users, 'locally'. When a user becomes idle, it is eliminated from the scheduling policy. We just briefly descibe the PFS scheduling policy. At each slot time t, for each frequency slot f, choose the users $k(t,f) \in \{1, \ldots, K\}$ subject to the TDMA

constraint, that maximizes

$$\sum_{k=1}^{K_{\text{on}}(t)} Q_k(t) \sum_{f=1}^{F} R_k(t,f), \tag{14}$$

where $\{Q_k(t)\}$ are the scheduling weights obtained dynamically by an update scheme based on virtual queues to optimize the PFS utility function. $R_k(t,f)$ is defined in (13). The reader is referred to [17] [18] for more details.

C. Trace based User Requests

We target a scenario where the BS is swamped with YouTube (short videos) like video requests and consider the average download delay as the metric of interest. We use the YouTube request trace data from a study conducted on the University of Massachusetts' Amherst campus in 2008 [5] [6]. 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. 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 ($\approx$ 3 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 distinguished from the trace data) is created. The user requests are simulated exactly as per these traces.

We use the trace data [19] 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 two 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) *Popular Helper* system- Helpers and the BS serve the users. Helpers store beforehand the most popular files allowed by their storage capacity.
3) *Greedy Helper* system - Helpers are allocated files according to the greedy algorithm given in table (I).

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 $100m$. Once connected to a helper, the user located at distance d from a helper experiences a rate $R_k(d) = 3 + 4/(100) * (100 - d)$ Mbps. This means that the rate scales linearly from 7 Mbps to 3 Mbps as the user moves from 0 to 100m from the helper. This rate is guaranteed irrespective of other users who may request from the same helper concurrently. Although the model is very simplified, it can be justified by facts about the latest WiFi standard quoted in Section II-A.

When a user requests a video file, and if the file lies in one of the helpers within range of the user, then the BS directs that helper to serve the user and the user gets it from the helper. The user delay inputs for the greedy algorithm, i.e., ω_k's in (2) are the time average download delay for user k for undropped requests:

1) Obtained from the simulation done on the Baseline (without helpers) assuming a user request pattern directly from the traces used.
2) Obtained from the simulation done on the Baseline assuming an analytical request model for user requests. 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. The probability of success (probability of requesting when idle) is p every second. The expected wait time is $1/p$. p can be chosen such that expected wait times are about $4-7$ minutes considering video playback to be 3 min.

Both these approaches are found to give the almost the same placement results. Hence, the placement algorithm requires no apriori information of user request pattern from future but just a rough model based delays.

To compare the baseline and the helper systems, for every user, we define the Quality of Service (QOS) as the average download delay. A user is *satisfied* when his/her average download delay is below a given QOS (in seconds) threshold. 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 all the simulations, Baseline, Popular Helper and Greedy Helper systems are compared. Some common conclusions are that the Greedy Helper system is strictly better than the Popular Helper System which is better than the Baseline and increasing the number of helpers increases the number of satisfied users. Now we analyze the results in detail. Figures 5, 6 and 7 show the number of satisfied users versus the storage capacity of helpers (same across all helpers). Figure 5 uses trace1 with QOS$= 100s$ and $H = 32$. The number of satisfied users (in figure 5) roughly doubles for both Popular Helper and Greedy Helper systems compared to the Baseline system.

Figures 6 and 7 use trace2, with QOS$= 100s$ and QOS$=$

$200s$ respectively and $H = 32$. Recall that the number of video requests in trace2 is approximately twice that in trace1. In figure 6, the Baseline system supports less than 10 users. Thus, in order to have reasonable performance for large number of video requests, the use of helpers is more crucial. In figure 7, the number of satisfied users increases more than $400 - 500\%$ in both helper systems. Better improvement is seen for QOS$=$ $100s$ (figure 6).

Figures 8 and 9 show the number of satisfied users versus the number of helpers with QOS$=$ $100s$ and $M = 2000$ using trace1 and trace2, respectively. We notice that, when the number of helpers increases, the difference between the Greedy Helper system and the Popular Helper system becomes more significant.

Figures 10 and 11 show the number of satisfied users versus QOS for trace1 and trace2, respectively, with $H = 32$ and $M = 1000$. 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 case of a large number of video requests.

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 $M = 1000$. 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.

IV. Summary and Conclusions

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 experimental 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.

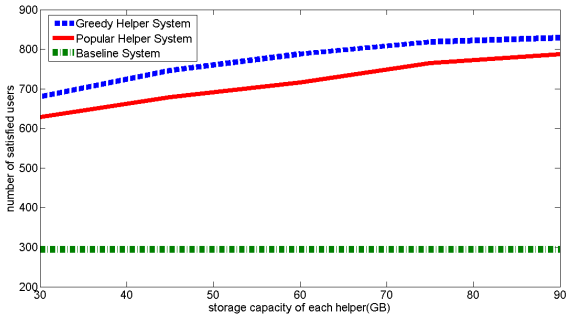

Fig. 5. The number of satisfied users versus the storage capacity of each helper for trace1, number of helpers=32, QOS=100 seconds.

References

[1] http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537 /ns705/ns827/white_paper_c11-520862.html.

[2] A.F. Molisch, *Wireless communications*. Wiley, 2011.

[3] V. Chandrasekhar, J. G. Andrews, and A. Gatherer,"Femtocell networks: a survey," *IEEE Commun. Mag.*, 46(9):59 – 67, Sept. 2008.

[4] S. El Rouayheb, A. Sprintson, and C. Georghiades, "On the index coding problem and its relation to network coding and matroid theory,". *IEEE Trans. Inf. Theory*, 56(7):3187 – 3195, 2010.

[5] M. Zink, K. Suh, Y. Gu, and J. Kurose. Watch global, "cache local: YouTube network traffic at a campus networkmeasurements and implications," *Proc. 15th SPIEACM Multimedia Computing and Networking*, 2008.

[6] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Characteristics of youtube network traffic at a campus network-measurements, models, and implications," *Computer Networks*, $53(4) : 501 - 514$, 2009.

[7] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," *IEEE Proc. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies,* v.1, p.126 – 134. IEEE, 1999.

[8] 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," *Proc. the twenty-ninth annual ACM symposium on Theory of computing*, p.654 – 663, 1997.

[9] M. Rabinovich and O. Spatscheck, "Web caching and replication," *SIGMOD Record*, 32(4):107, 2003.

[10] S. Borst, "User-level performance of channel-aware scheduling algorithms in wireless data networks," *IEEE Proc. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, v. 1, p. 321 – 331, 2003.

[11] M. Cardei and D.Z. Du, "Improving wireless sensor network lifetime through power aware organization," *Wireless Networks*, $11(3) : 333 -$ 340, 2005.

[12] A. Schrijver. *Combinatorial optimization: polyhedra and efficiency*, Springer Verlag, 2003.

[13] G.L. Nemhauser, L.A. Wolsey, and M.L. Fisher, "An analysis of approximations for maximizing submodular set functionsI," *Mathematical Programming*, $14(1) : 265 - 294$, 1978.

[14] G. Calinescu, C. Chekuri, M. Pal, and J. Vondrak, "Maximizing a monotone submodular function subject to a matroid constraint," to appear, 2009.

[15] *http://www.itu.int/pub/R-REP-M.2135-2008.*

[16] H. Holma and A. Toskala, *LTE for UMTS: OFDMA and SCFDMA based radio access.* John Wiley & Sons Inc, 2009.

[17] L. Georgiadis, M. Neely, M.J. Neely, and L. Tassiulas, "Resource allocation and cross-layer control in wireless networks," Now Pub, 2006.

[18] M.J. Neely, "Stochastic Network Optimization with Application to Communication and Queueing Systems," *Synthesis Lectures on Communication Networks*, 3(1) : 1 – 211, 2010.

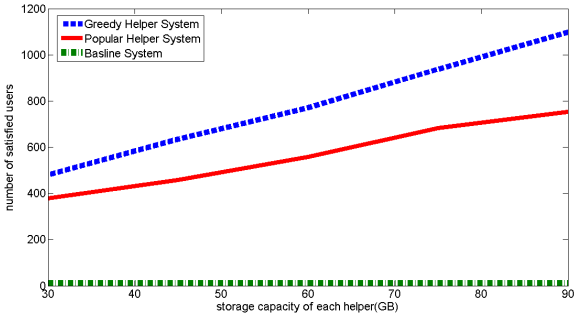

Fig. 6. The number of satisfied users versus the storage capacity of each helper for trace2, number of helpers=32, QOS=100 seconds.

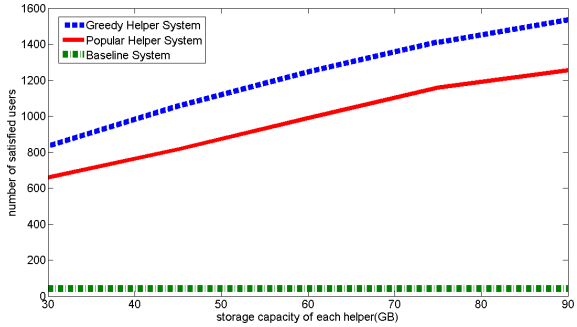

Fig. 7. The number of satisfied users versus the storage capacity of each helper for trace2, number of helpers=32, QOS=200 seconds.

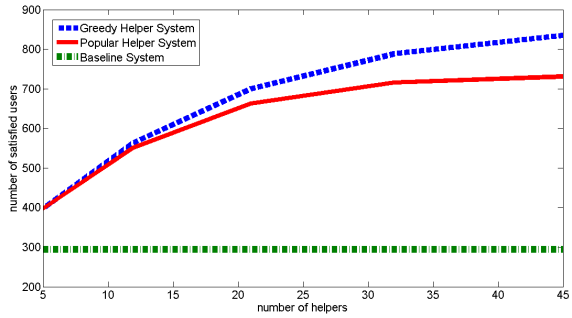

Fig. 8. The number of satisfied users versus the number of helpers for the trace1, the cache capacity of each helper is $60GB$, QOS is 100 seconds.

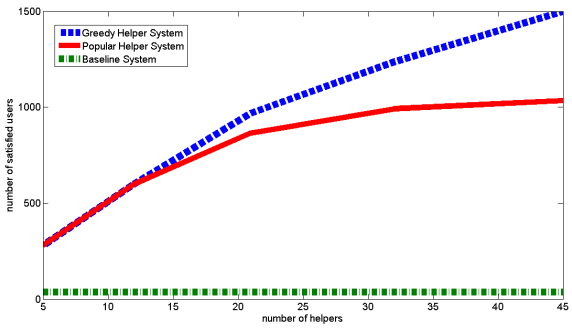

Fig. 9. The number of satisfied users versus the number of helpers for the trace2, the cache capacity of each helper is $60GB$, QOS is 200 seconds.

[19] *http://traces.cs.umass.edu/index.php/Network/Network.*

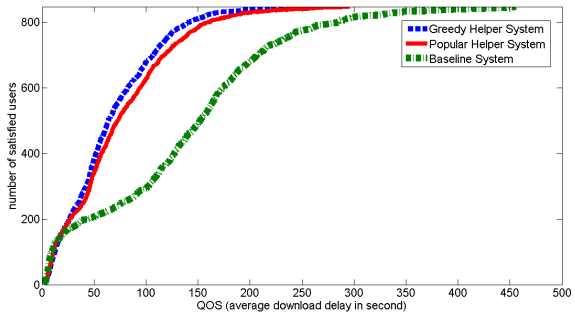

Fig. 10. The number of satisfied users versus QOS (the average download delay) for the trace1, the cache capacity of each helper is $30GB$ and the number of helpers is 32.

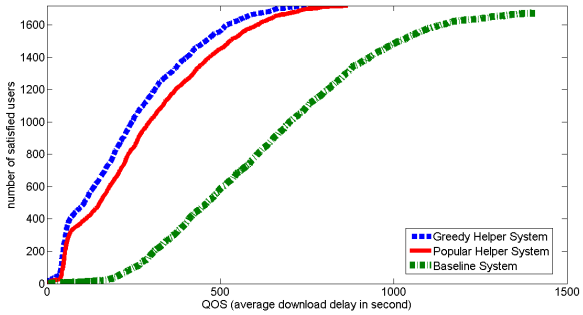

Fig. 11. The number of satisfied users versus QOS (the average download delay) for the trace2, the cache capacity of each helper is $30GB$ and the number of helpers is 32.

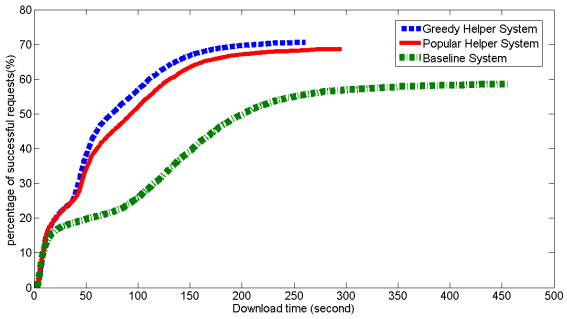

Fig. 12. The percentage of successful requests versus download time for the trace1, the cache capacity of each helper is $30GB$ and the number of helpers is 32.

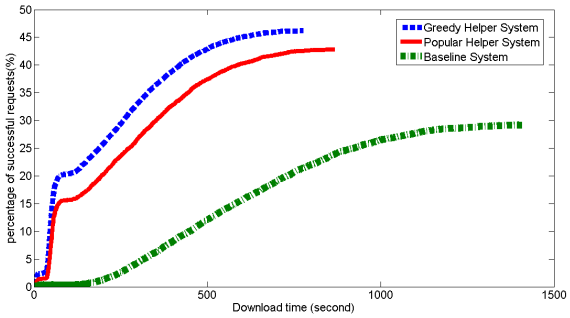

Fig. 13. The percentage of successful requests versus download time for the trace1, the cache capacity of each helper is $30GB$ and the number of helpers is 32.