Proactive Small Cell Networks

Ejder Bastug, Jean-Louis Guénégo, Mérouane Debbah

Proactive Small Cell Networks

Ejder Başтuğ, Jean-Louis Guénégo, and Mérouane Debbah
Alcatel-Lucent Chair - SUPÉLEC, Gif-sur-Yvette, France
{ejder.bastug, jean-louis.guenego, merouane.debbah}@supelec.fr

***Abstract*—Proactive scheduling in mobile networks is known as a way of using network resources efficiently. In this work, we investigate proactive Small Cell Networks (SCNs) from a caching perspective. We first assume that these small base stations are deployed with high capacity storage units but have limited capacity backhaul links. We then describe the model and define a Quality of Experience (QoE) metric in order to satisfy a given file request. The optimization problem is formulated in order to maximize this QoE metric for all requests under the capacity constraints. We solve this problem by introducing an algorithm, called *PropCaching* (proactive popularity caching), which relies on the popularity statistics of the requested files. Since not all requested files can be cached due to storage constraints, the algorithm selects the files with the highest popularities until the total storage capacity is achieved. Consecutively, the proposed caching algorithm is compared with random caching. Given caching and sufficient capacity of the wireless links, numerical results illustrate that the number of satisfied requests increases. Moreover, we show that *PropCaching* performs better than random caching in most cases. For example, for $R = 192$ number of requests and a storage ratio $\gamma = 0.25$ (storage capacity over sum of length of all requested files), the satisfaction in *PropCaching* is 85% higher than random caching and the backhaul usage is reduced by 10%.**

***Index Terms*—Small cell networks, proactive caching, popularity caching**

I. Introduction

Market forecasts nowadays point out an explosion of mobile traffic [1]. The traffic generated by wireless devices is expected to become much higher than the traffic generated by wired devices. Even with the latest advancements, such as long term evolution (LTE) networks, wireless networks will not be able to sustain the demanded rates. To overcome this shortage, small cell networks (SCNs) have been proposed as a candidate solution [2], and they are expected to be the successor of LTE networks [3].

Deploying such high data rate SCNs to satisfy this demand requires high-speed dedicated backhaul. Due to the costly nature of this requirement, the current state of the art proposes to add high storage units (i.e., hard-disks, solid-state drives) to small cells (SCs) and use these units for caching purposes [4], (i.e., in order to offload the significant amount of backhaul usage). To decrease the cost and have additional benefits, the work in [5] proposes to use such a dense infrastructure opportunistically for cloud storage scenarios either for caching or persistent storage. Independently, another line of research focuses on proactivity to handle network resources efficiently, and can be seen as a complementary [6], [7].

In this work, we complementarily merge the two approaches for SCNs. We focus on proactive SCNs where we have small base stations deployed with high storage units but have limited backhaul links. In detail, we have the following observations:

- In classical networks, called hereafter *reactive* networks, user requests are satisfied right after they are initiated. In contrast, *proactive* SCNs can track, learn and then establish a user request prediction model. Therefore, we can achieve flexibility in scheduling efficient resources.
- Although human behaviour is highly predictable and correlated [8], [9], in reality, predicting the exact time of user requests might not be obtainable. However, statistical patterns such as file popularity distributions, may help to enable a certain level of prediction. By doing this, the predicted files can be cached in SCNs. Thus, the backhaul can be offloaded and mobile users can have a higher level of satisfaction.
- The caching in SCNs can be low-cost as the storage units have become exceptionally cheap. For instance, putting two terabytes of storage in a SC costs approximately 100 dollars.
- The network operators usually deploy transparent caching proxies to accelerate service requests and reduce bandwidth costs. We observe that these kinds of proactive caching approaches can be a complementary way of cost reduction by migrating the role of these proxies to SCNs.

Given these observations, the aim of this work is to investigate the impact of caching in SCNs. Looking to caching as a way of being proactive, we also introduce an algorithm called *PropCaching* (proactive popularity caching). The method relies on the popularity statistics of the requested files. We then compare our results with random caching.

The rest of the paper is organized as follows. We discuss the problem scenario and describe our model in Section II. We formulate an optimization problem and explain the *PropCaching* algorithm in Section III. Related numerical results are given in Section IV. Finally, we conclude in Section V.

The mathematical notation used in this paper is the following. Lower case and uppercase italic symbols (e.g., b, B) are scalar values. A lower case boldface symbol (e.g., $\mathbf{b}$) denotes a row vector and an upper case boldface symbol (e.g., $\mathbf{B}$) a matrix. $\mathbf{1}_{M\times N}$ represents a $M \times N$ matrix with all ones, and $\mathbf{0}_{M\times N}$ is again $M \times N$ sized matrix but with entries set to zeros. $\mathbf{\Lambda}(\mathbf{b})$ is a diagonal matrix constructed from vector $\mathbf{b}$. The indicator function $\mathbb{1}\{\cdot > \cdot\}$ returns 1 when the inequality holds, otherwise 0. Finally, the transpose of $\mathbf{b}$ is denoted with $\mathbf{b}^T$.

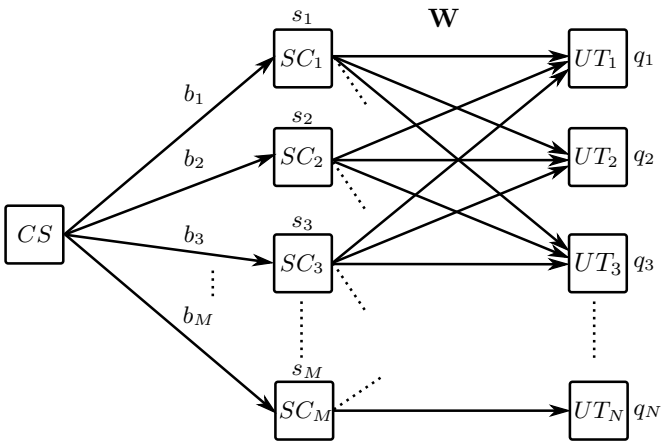


Figure 1. A SCN scenario consists of a central scheduler, small cells and user terminals.

II. Scenario

Consider a SCN scenario where a central scheduler (CS) and M SCs are in charge of serving N user terminals (UTs) as depicted in Fig. 1. In this setting, the CS is coordinating and providing broadband access to SCs over cheap backhaul links with given capacities $\mathbf{b} = [b_1, ..., b_M] \in \{0, \mathbb{Z}^+\}$. On the other hand, SCs have high storage units with the storage capacities $\mathbf{s} = [s_1, ..., s_M] \in \{0, \mathbb{Z}^+\}$, thus, they can cache information coming from the CS and serve their UTs over wireless links with the rate of:

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \vdots \\ \mathbf{w}_M \end{bmatrix} = \begin{bmatrix} w_{1,1} & \dots & w_{1,N} \\ w_{2,1} & \dots & w_{2,N} \\ \vdots & \vdots & \vdots \\ w_{M,1} & \dots & w_{M,N} \end{bmatrix} \in \{0, \mathbb{Z}^+\}^{M \times N}, \tag{1}$$

where $w_{i,j}$ represents the rate from i-th SC to j-th UT, in bits per timeslot.

Now, suppose that over a given time window T, users want to perform requests with the rates specified by $\mathbf{q} = [q_1, ..., q_N] \in \{0, \mathbb{Z}^+\}$. In other words, the j-th user wants to perform q_j number of requests during the time window T. Let us say for instance that users want to download files from internet. Thus, the CS keeps track of F different files indexed as $\mathbf{f} = [f_1, ..., f_F]$. Each file f_i is atomic and has a length specified by l_i. We denote $\mathbf{l} = [l_1, ..., l_F] \in \mathbb{Z}^+$.

In a *reactive* scenario, requests would be satisfied by the CS just after they are initiated by the user. In a *proactive* scenario, the CS tracks, learns and then predicts the user requests before the actual request arrival. This helps to decide which files should be stored in which SC before it is requested. Because the storage capacities of SCs are limited even if they are high, proactivity includes a file replacement policy. This policy tries to set or replace files in order to let the cache contain the right file at the right SC. This proactive storage mechanism would enable the network to use its resources efficiently especially in peak times where the load of the backhaul is very high. Thus, it would avoid large delays in file delivery.

Let us assume discrete popularity distributions of files of UTs:

$$\bar{\mathbf{P}} = \begin{bmatrix} \bar{\mathbf{p}}_1 \\ \bar{\mathbf{p}}_2 \\ \vdots \\ \bar{\mathbf{p}}_N \end{bmatrix} = \begin{bmatrix} \bar{p}_{1,1} & \dots & \bar{p}_{1,F} \\ \bar{p}_{2,1} & \dots & \bar{p}_{2,F} \\ \vdots & \vdots & \vdots \\ \bar{p}_{N,1} & \dots & \bar{p}_{N,F} \end{bmatrix} \in [0,1]^{N \times F}, \tag{2}$$

where $\bar{p}_{i,j}$ represents the probability of the j-th file being requested by the i-th UT. Note that the matrix $\bar{\mathbf{P}}$ is row-stochastic as i-th row represents the discrete probability distribution of i-th user, hence, the sum of all elements in i-th row equals to 1. The $\bar{\mathbf{P}}$ might be sparse as number of files grows by time.

As we defined above, $\bar{\mathbf{P}}$ represents the local file popularities of the UTs. The file popularity distributions observed at the SCs will be different than $\bar{\mathbf{P}}$ as all connected users will contribute with their local file popularities. To show this, let us first define the connectivity matrix, such that,

$$\mathbf{C} = \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \vdots \\ \mathbf{c}_M \end{bmatrix} = \begin{bmatrix} c_{1,1} & \dots & c_{1,N} \\ c_{2,1} & \dots & c_{2,N} \\ \vdots & \vdots & \vdots \\ c_{M,1} & \dots & c_{M,N} \end{bmatrix} \in \{0,1\}^{M \times N}, \tag{3}$$

where $c_{m,n} = \mathbb{1}\{w_{m,n} > 0\}$. Then, we can derive popularity distributions at the SCs, called $\mathbf{P}$, by multiplying $\bar{\mathbf{P}}$ with the user request rates $\mathbf{q}$, the connectivity matrix $\mathbf{C}$, and a normalization factor. The equation is given in (4), where $p_{i,j}$ represents the popularity of the j-th file seen by the i-th SC.

If the matrix $\mathbf{P}$ can be obtained by CS at $t = 0$, proactive caching strategies can be employed. In practice, this matrix can be computed at the CS by counting the number of times the files are requested. This would give the information about the file popularity of the future requests as the user behaviour is correlated. Assuming that the matrix $\mathbf{P}$ is perfectly given in our scenario, the following step is to decide how to distribute these files among SCs. This is detailed in the next section.

III. Proactivity Through Caching

In this section, we formulate an optimization problem for the satisfied requests over total requests and provide a *PropCaching* algorithm to heuristically maximize the objective.

Suppose that, over the time window T as assumed in the scenario, the CS is observing a number of R file request events listed in $\mathbf{r} = [r_1, ..., r_R]$. A request $r_i \in \mathbf{r}$ is done by a UT to download the file f^{r_i} with the corresponding length of l^{r_i}. Hence, the CS has to deliver this file to the user either from the internet or from the caches of the connected small cells. The delivery for the request r_i starts at $t = t^{r_i}$ and finishes until the file is completely downloaded by the UT at $t = \hat{t}^{r_i}$. For each file f_i, let us define its corresponding bandwidth requirement

$$
\begin{aligned}
\mathbf{P} =& \mathbf{\Lambda}\left(\frac{1}{\mathbf{c}_1\mathbf{q}^T}, ..., \frac{1}{\mathbf{c}_M\mathbf{q}^T}\right)\mathbf{C}\mathbf{\Lambda}(\mathbf{q})\bar{\mathbf{P}} \\
=& \underbrace{\begin{bmatrix} \frac{1}{\mathbf{c}_1\mathbf{q}^T} & 0 & \dots & 0 \\ 0 & \frac{1}{\mathbf{c}_2\mathbf{q}^T} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{\mathbf{c}_M\mathbf{q}^T} \end{bmatrix}}_{\text{normalization factor}} \underbrace{\begin{bmatrix} c_{1,1} & \dots & c_{1,N} \\ c_{2,1} & \dots & c_{2,N} \\ \vdots & \vdots & \vdots \\ c_{M,1} & \dots & c_{M,N} \end{bmatrix}}_{\text{connectivity}} \underbrace{\begin{bmatrix} q_1 & 0 & \dots & 0 \\ 0 & q_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & q_N \end{bmatrix}}_{\text{request rates}} \underbrace{\begin{bmatrix} \bar{p}_{1,1} & \dots & \bar{p}_{1,F} \\ \bar{p}_{2,1} & \dots & \bar{p}_{2,F} \\ \vdots & \vdots & \vdots \\ \bar{p}_{N,1} & \dots & \bar{p}_{N,F} \end{bmatrix}}_{\text{popularity distribution of files at UTs}} \\
=& \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \vdots \\ \mathbf{p}_M \end{bmatrix} = \begin{bmatrix} p_{1,1} & \dots & p_{1,F} \\ p_{2,1} & \dots & p_{2,F} \\ \vdots & \vdots & \vdots \\ p_{M,1} & \dots & p_{M,F} \end{bmatrix} \in [0,1]^{M\times F}.
\end{aligned} \tag{4}
$$

Algorithm 1 *PropCaching* algorithm.

Input: $\mathbf{s}$, $\mathbf{f}$, $\mathbf{l}$, $\mathbf{P}$

$\mathbf{\Theta} \leftarrow \mathbf{0}_{M\times F}$, $\hat{\mathbf{s}} \leftarrow \mathbf{0}_{M\times 1}$ ▷ Initializes the cache decision matrix $\mathbf{\Theta}$ and the current storage vector $\hat{\mathbf{s}}$
for $i = 1, ..., M$ **do**
  $[\mathbf{a}, \mathbf{b}] \leftarrow \text{SORT}(\mathbf{p}_i)$ ▷ Sorts $\mathbf{p}_i$ by descending order, returns $\mathbf{a}$ and $\mathbf{b}$ as ordered values and indices
  for $j = 1, ..., F$ **do**
    $k \leftarrow b_j$ ▷ Gets index of j-th most popular file
    if $l_k + \hat{s}_i \leq s_i$ **then** ▷ Checks if the j-th most popular file can be stored in the i-th SC
      $\Theta_{i,k} \leftarrow 1$ ▷ Sets the element of the cache decision matrix to 1, meaning that the file will be cached
      $\hat{s}_i \leftarrow \hat{s}_i + l_j$ ▷ Increases the current storage
    else
      break ▷ Stops inner loop if the storage capacity is achieved
    end if
  end for
end for

Output: $\mathbf{\Theta}$

as $d_i \in \mathbf{d} = [d_1, ..., d_F]$. Now, suppose that $d^{r_i} \in \mathbf{d}$ is the bandwidth requirement for the request r_i and $x^{r_i}(t)$ is the amount of delivered information up to the time t (see Fig. 2). By definition we consider that the request r_i is satisfied, if the average delivery rate in any time instance is superior to d^{r_i}. In this setting, the following formula holds:

$$
\frac{1}{\hat{t}^{r_i} - t^{r_i}} \sum_{t=t^{r_i}}^{\hat{t}^{r_i}} \mathbb{1}\left\{\frac{x^{r_i}(t)}{t - t^{r_i}} \geq d^{r_i}\right\} = 1. \tag{5}
$$

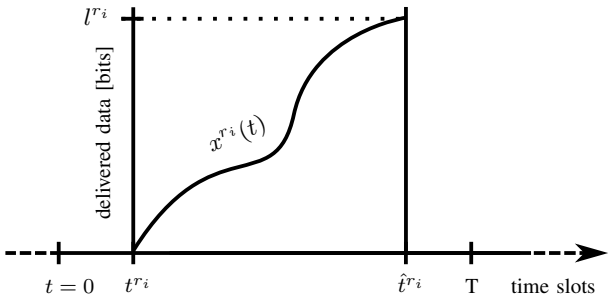


Figure 2. Deadline of a request r_i.

The idea in (5) is to ensure that the network delivers the data with enough speed such that waiting is not going to occur during the playback. Therefore, if the request is satisfied, UT will have a better quality of experience (QoE). The determination of the value d^{r_i} might be different for different types of content. Roughly speaking, watching a compressed high-definition (HD) video on a website (i.e., YouTube) requires $3 - 4$ Mbps of bandwidth for interruption-free playback. For our scenario, an optimization problem can be formulated in the sense of maximizing the number of satisfied requests under the capacity constraints. If we denote $\hat{R}$ as the ratio of the satisfied requests over the total requests R, or simply the *satisfaction ratio*, we have:

$$
\begin{aligned}
\underset{t^{r_i}}{\text{maximize}} \quad & \hat{R} = \frac{1}{R}\sum_{r_i\in\mathbf{r}} \frac{1}{\hat{t}^{r_i} - t^{r_i}} \sum_{t=t^{r_i}}^{\hat{t}^{r_i}} \mathbb{1}\left\{\frac{x^{r_i}(t)}{t - t^{r_i}} \geq d^{r_i}\right\} \\
\text{subject to} \quad & \mathbf{b} \preceq B^{max}, \\
& \mathbf{s} \preceq S^{max}, \\
& \mathbf{W} \preceq W^{max},
\end{aligned} \tag{6}
$$

where B^{max}, S^{max} and W^{max} are the capacity constraints of backhaul links, storage units and wireless links, respectively.

In general, even being able to predict the request times and storing the corresponding files in SCs before their arrival, all requests might not be satisfied due to the capacity constraints. Employing a brute-force search by varying the start of the delivery times would be hard due to the combinatorial behaviour of the problem. Therefore, instead of a time oriented approach, we focus on caching the popular files to achieve a certain amount of proactivity. Suppose that the CS has a *cache decision matrix* (also called *caching matrix*), such that

$$\mathbf{\Theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_M \end{bmatrix} = \begin{bmatrix} \theta_{1,1} & \dots & \theta_{1,F} \\ \theta_{2,1} & \dots & \theta_{2,F} \\ \vdots & \vdots & \vdots \\ \theta_{M,1} & \dots & \theta_{M,F} \end{bmatrix} \in \{0,1\}^{M \times F}, \quad (7)$$

where $\theta_{i,j} = 1$ means that the i-th SC has to store the j-th file, and $\theta_{i,j} = 0$ is the vice versa. The complete proactive case in our scenario would be achieved if all the files could be cached in the SCs. Hence, our caching matrix would be $\mathbf{\Theta} = \mathbf{1}_{M \times F}$, meaning that all SCs cache all files before the requests start. In the worst case, no file would be stored, so the caching matrix would be $\mathbf{\Theta} = \mathbf{0}_{M \times F}$.

The matrix $\mathbf{\Theta}$ in CS can be obtained using the *PropCaching* algorithm, as given in Algorithm 1. The algorithm basically chooses to store the files with the highest popularities until the storage capacity of SCs are achieved. The detailed explanation is given step by step in the algorithm. Recall that this cache method of decision is due to the fact that the exact request times are not practically obtainable. Even knowing the request times and then solving the problem would be hard with a brute-force search algorithm. Our heuristic approach is used to maximize our objective by caching the files according to the popularity statistics of the files and the storage capacities of SCs.

Assuming that the complexity of the initialization step is $\mathcal{O}(1)$ and the sorting operation has $\mathcal{O}(F \log F)$ worst-case complexity (i.e., by using Timsort), the complexity of Algorithm 1 becomes $\mathcal{O}(1 + M(F \log F + 4F)) \simeq \mathcal{O}(MF \log F)$, which is linear in M and almost linear in F. As we present *PropCaching* in a simple way for sake of clarity, in real-time systems where M and F is very big and the time window is moving, a more efficient method could be implemented by employing an iterative approach of the algorithm.

IV. Numerical Results

A discrete event simulator was implemented in order to obtain the numerical results for the scenario. The capacity of the backhaul links are assumed to be lower than the capacity of the wireless links. Over the time window T, the request times are pseudo-randomly generated with uniform distribution. The popularity of files are also generated using uniform distribution. The simulation parameters are given in Table I. After generating the request times with respect to the parameters, Algorithm 1 is simulated for different values of the *storage ratio* γ, which is defined as

Table I
Simulation parameters.

Parameter	Values	Description
T	1024	time window (time slot)
M	4	number of SCs
N	16	number of UTs
B^{max}	16	total capacity of backhaul links (Mbit/time slot)
W^{max}	128	total capacity of wireless links (Mbit/time slot)
F	128	number of files
l_i, $\forall i$	256	length of files (Mbit)
b_i, $\forall i$	4	backhaul link capacities (Mbit)
$w_{i,j}$, $\forall i,j$	2	wireless link capacities (Mbit)
d_i, $\forall i$	4	QoE requirement (Mbit)

$$\gamma = \frac{S^{max}}{M \sum_{i=1}^{F} l_i}. \quad (8)$$

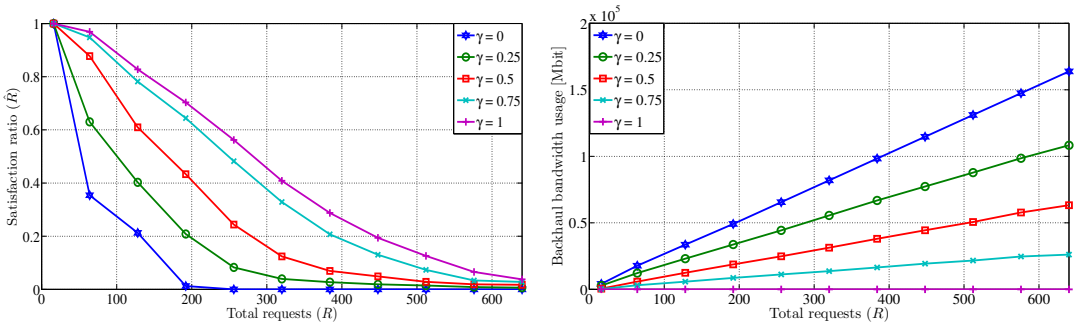
In order to see the impact of the storage in SC with the proactive caching approach, the following γ values are simulated: $\{0, 0.25, 0.50, 0.75, 1\}$. In the worst case, when $\gamma = 0$, SCs would have zero storage capacity. Hence, Algorithm 1 would compute the caching matrix as $\mathbf{\Theta} = \mathbf{0}_{M \times F}$. On the other hand, $\gamma = 1$ is the best case where each SC has enough storage to cache all requested files. Therefore the caching matrix would be $\mathbf{\Theta} = \mathbf{1}_{M \times F}$.

After estimating the caching matrix for each γ values using Algorithm 1, the corresponding files are assumed to be in SCs in order to avoid additional bandwidth usage for their delivery to SCs. In fact, this assumption is reasonable as some files might be either previously stored or can be stored on their first delivery. After this assumption, by iterating over time, the CS delivers the files either from the internet or from the cache of the connected SCs. In case of simultaneous transmissions due to the same request times and/or continuing transmissions, the network resources are fairly shared among the UTs. More precisely, the total backhaul and wireless bandwidths are equally divided among the requests.

The simulation using *PropCaching* is repeated while increasing the number of requests. These operations are repeated 100 times and averaged. To compare the gain of *PropCaching*, simulations of the random caching method is also performed by filling the $\mathbf{\Theta}$ with uniform distribution, until the storage capacity is achieved.

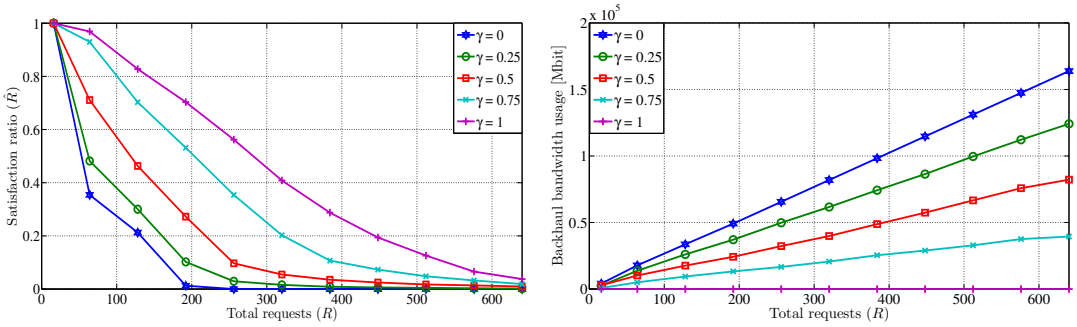
Numerical results for *PropCaching* are shown in Fig. 3(a). The figure shows the evolution of the satisfaction ratio $\hat{R}$ over the total requests R. For each γ value, the satisfaction ratio remains 1 until a certain threshold because of sufficient capacity of the links. After that, due to the congestion caused by many simultaneous deliveries, the number of satisfied requests starts to decrease and converges to 0. This is obvious due to the fair bandwidth sharing policy. As the total requests $R \to \infty$, the amount of delivery bandwidth given to a request tends to 0. Therefore, no request will be satisfied as $R \to \infty$.

The decrement in case of $\gamma = 0$ is strictly related to the backhaul capacity as the files are delivered over the backhaul. The more we increase γ the more we store the files in SCs. Thus, the decrease becomes more dependent on the wireless



(a) (b)

Figure 3. Numerical results of *PropCaching*: (a) evolution of the satisfaction ratio, (b) the backhaul bandwidth usage.



(a) (b)

Figure 4. Numerical results of random caching: (a) evolution of the satisfaction ratio, (b) the backhaul bandwidth usage.

link capacities. In general, as γ increases, the satisfaction ratio $\hat{R}$ becomes better compared to the non-caching case.

Fig. 3(b) illustrates the backhaul bandwidth consumption over the total requests R. It is seen that as γ increases, the amount of bandwidth consumption decreases. This is evident as more files are cached.

For the performance of random caching, Fig. 4(a) and Fig. 4(b) depict the evolution of the satisfaction ratio and the backhaul bandwidth consumption respectively. In case of $\gamma = 0$ and $\gamma = 1$, the performance is obviously identical to *PropCaching*. However, in between, *PropCaching* has different gains for different R values. For example, for $R = 192$ and $\gamma = 0.25$, the satisfaction of *PropCaching* is 85% higher than random caching and the backhaul usage is reduced by 10%.

V. Conclusions

In this paper, proactive SCNs equipped with low capacity backhaul links and high storage units are investigated from a caching point of view. By using the popularity statistics of the files and employing a caching strategy based on this, the impact of storage in SCs is studied. As the load of the network increases by the number of requests, our results show that caching has a better performance in satisfying the requests compared to the non-caching case. Moreover, *PropCaching* outperforms random caching in the most cases.

Other approaches of proactivity could be caching the files according to the *recent* and *trending* statistics of the files. We think that by storing recently downloaded files or trending files according to time varying behaviour of the file popularities, the performance of these networks can be improved.

VI. Acknowledgment

This research has been supported by the ERC Starting Grant 305123 MORE (Advanced Mathematical Tools for Complex Network Engineering).

References

[1] Cisco, "Cisco visual networking index: Global mobile data traffic forecast update, 2011–2016," *White Paper, [Online] http://goo.gl/lLkTl*, 2012.

[2] J. Hoydis, M. Kobayashi, and M. Debbah, "Green small-cell networks," *IEEE Vehicular Technology Magazine*, vol. 6(1), pp. 37–43, 2011.

[3] Small Cell Forum, "Small cells to make up almost 90% of all base stations by 2016," *[Online] http://goo.gl/qFkpO*, 2012.

[4] N. Golrezaei, K. Shanmugam, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless video content delivery through distributed caching helpers," *arXiv preprint: 1109.4179*, 2011.

[5] E. Baştuğ, J.-L. Guénégo, and M. Debbah, "Cloud storage for small cell networks," in *IEEE CloudNet'12*, (Paris, France), Nov. 2012.

[6] H. E. Gamal, J. Tadrous, and A. Eryilmaz, "Proactive resource allocation: Turning predictable behavior into spectral gain," in *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, pp. 427 – 434, 2010.

[7] J. Tadrous, A. Eryilmaz, and H. E. Gamal, "Proactive resource allocation: Harnessing the diversity and multicast gains," *submitted to IEEE Transactions on Information Theory, arXiv preprint: 1110.4703*, 2011.

[8] C. Song, Z. Qu, N. Blumm, and A.-L. Barabási, "Limits of predictability in human mobility," *Science*, vol. 327, no. 5968, pp. 1018–1021, 2010.

[9] V. Etter, M. Kafsi, and E. Kazemi, "Been There, Done That: What Your Mobility Traces Reveal about Your Behavior," in *Mobile Data Challenge by Nokia Workshop, in conjunction with Int. Conf. on Pervasive Computing*, 2012.