INTERNATIONAL HELLENIC UNIVERSITY

Caching at the radio access network edge for 5G networks

Abdullah Alsabbagh

SID: 3301160001

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Information and Communication Systems

December 2017

THESSALONIKI – GREECE

INTERNATIONAL HELLENIC UNIVERSITY

Caching at the radio access network edge for 5G networks

Abdullah Alsabbagh

SID: 3301160001

Supervisor: **Dr. Merkouris Karaliopoulos**

Supervising Committee Members:

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Information and Communication Systems

December 2017

THESSALONIKI – GREECE

Abstract

This dissertation was written as a part of the MSc in ICT Systems at the International Hellenic University.

Caching at the network edge is a recently proposed technique for the upcoming mobile generation 5G, to reduce the backhaul rates at the peak hours by prefetching popular contents and store them into memories at or near to the end users. However, we focus on a new revolutionary caching scheme named as coded caching that take advantage of the multicast medium of the mobile network to offer a considerable gain through information theory coding techniques. In this work, we analyze the performance of two dominant approaches. A comparative simulation-based study has been established of uncoded and coded caching under various levels of spatial locality of the user contents.

Our simulation results show that LFU (Least frequently used) uncoded caching scheme provides a better performance than coded caching schemes for real-life scenarios which were represented in our simulation as non-uniform content popularity. In addition, coded caching scheme still needs additional improvements regarding the supported number of users as well as the computational complexity imposed on users and server sides.

Abdullah Alsabbagh
December 2017

Table of Content

Table of Figures

Acknowledgments

I would like to express my sincere gratitude to my supervisor Dr. Merkouris Karaliopoulos who has supported me enthusiastically and constantly during the entire project; His suggestions, comments, and advice throughout the project were invaluable and helped me a lot to achieve this project. Last but not least, I would like to thank my family who has been hugely supportive throughout my project.

Chapter 1 Introduction

1.1 Motivation

In the last few years, smartphones have become an essential part of our daily life. It is not just a communications-oriented service anymore as most of our daily tasks can be done through our phones, i.e. booking, emails, controling our social media accounts etc. Nowadays, with our phones, we can: 1) access popular VoD (video on demand) websites like YouTube, Netflix and many others, 2) download high-quality videos up to 4K and 8K, generating massive data traffic that travels all the way from the data centers through Content Delivery Networks (CDNs) and intermediate network access links to reach our phones.

The response of the mobile network operators to the capacity demands these trends raise, has typically been to increase their subscribers' bandwidth by upgrading their network to the last generation of radio transmission technologies. From one system generation to another, two have been the main ways towards more data delivery capacity. On the one hand, the network access points are densified: more cells of smaller size emerge closer to the end user. In parallel, new radio transmission technologies and access schemes are deployed that can scale up the transmission rate over the wireless links and accommodate more bits over the air for given spectrum slice. Nevertheless, mobile video content is projected to account for 75% of mobile data traffic volumes by 2020, when the total mobile data traffic is predicted to grow by a factor ranging from eight to ten by 2020. At the same time, the 5G community is far more aggressive targeting a 1000-fold increase in the mobile data volume served by their networks. The support of such data volumes over the radio links poses far greater challenges to the radio link designers; at the same time, even if these rates are achieved over the radio links, it renders backhaul links bottlenecks of the mobile network.

One of the most promising solutions towards overcoming the challenge for higher data delivery capacities in the mobile networks is the use of caching techniques *within* the mobile cellular network and, more precisely, *at its edges*. Implementing in-network caching by placing CDN (content delivery networks) nodes at the gateway nodes of the mobile network will already benefit the mobile network by accelerating the data transfer to end users and reducing the transit traffic to other networks. Nevertheless, mitigating backhaul and wireless links' overload requires moving beyond the placement of CDN nodes at the network border, and caching

content also at the radio network access points (base stations) and even at the users' mobile devices. Those caches will operate in two phases: the *content placement* phase during off-peak hours, where caches are populated with (most popular) content; and the delivery phase during peak hours, when users' requests for content are submitted to and, hopefully, served by these caches so that the congestion at the core network and the backhaul links is reduced.

In the last five years, multiple research efforts have been dedicated to fitting promising advanced caching concepts to the reality and constraints of the wireless mobile network. Two are the dominant caching techniques that have been proposed to this end: *femtocaching* [1] and *coded caching* [4]. Femtocaching resembles the common caching technique used in Internet for Web content, namely the whole (video) file must be stored at the cache. On the other hand, with coded caching, users populate their caches with smaller *coded* parts of more content items and can recover the full content with the help of encoded broadcast transmissions.

In what follows, a review of the main available results about these two techniques has been presented.

1.2 State-of-the-art Work

1.2.1 Caching in General

Caching is currently being revisited in the context of mobile cellular networks and proposed to be implemented in 5G Networks. This has motivated a number of papers and articles addressing its design and implementation in the wireless network but also the main system features that have an impact on its performance and proper ways to model them. Most of these issues are more generic and do not concern exclusively the mobile wireless networks.

Temporal and spatial locality of content demand and their implications: The demand for content exhibits variations both across time and space. For example, some TV series' episodes become rapidly popular within a small period of time and then become unpopular again, when they are outdated by newer episodes. Moreover, the content requested, say, during working hours is often different than the one accessed over evening leisure time. We refer to these temporal variations of content demand as time locality.

Likewise, the demand may differentiate from one geographical area to another. Beyond globally popular content, there is much content of more local interest. Indeed, the demand for almost all contents available online today may present variability at multiple geographical

levels: continental, national, regional, city or even neighborhood level. For a cellular network, this implies that different network cells may present different statistical patterns of aggregate content demand. We refer to these variations, which coexist with the temporal ones, as the spatial locality of content demand.

These two fundamental properties of the content demand have also direct implications for (a) the modeling work in the area of caching systems since they invalidate basic assumptions of de facto popularity models; (b) the actual algorithms that control the content of the cache each moment in time.

Hence, the standard model used in the study of web caching, called independence reference model (IRM), is designed for static content popularity and is not a valid choice under rapidly changing content popularity [2]. The shot noise model (SNM) proposed in [6] shows more accurate results than the IRM with respect to caching performance analysis [6]. Figure 1.1 shows that fitting real traces of content requests, collected from YouTube, with the SNM offers a far better closer match than fitting them with the IRM since the former better captures the strong correlations between content popularity and time.

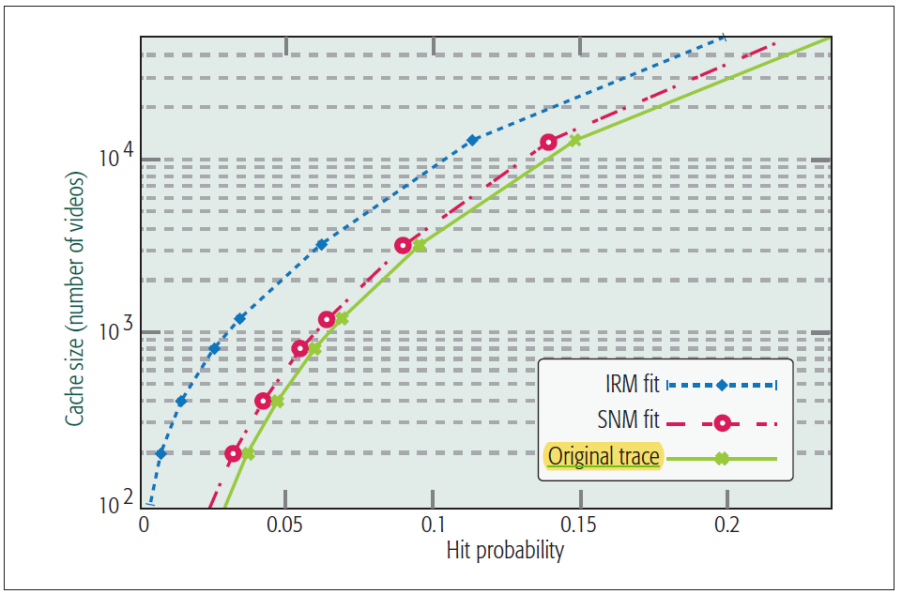

Figure 1.1 Hit probability comparison between best fit of the IRM, SNM, and YouTube traces (extracted from [6]).

The temporal locality properties also call for novel dynamic cache management policies. The de-facto policy used in web caching, the LRU (least recently used) caching algorithm, evicts the content item with the highest time at the cache without a request every time a request for a

new content item is made, which is not in the cache. LRU is proven to be optimal under the IRM model, but it is suboptimal when it is applied to time-varying content popularity [2].

Finally, both types of locality make the actual online detection of demand dynamics a particularly challenging task. The spatial locality, in particular, gives rise to a tradeoff between prediction accuracy and speed. Matching the cache content to the local demand implies learning it out of the local requests only. However, aggregating data at BS level turns out to be much slower than doing so at the network level since collecting a statistically significant and reliable sample for training the prediction algorithm requires more time. To remedy the situation, new caching architectures have been proposed that combine information obtained at different aggregation layers [7].

User privacy and HTTP encryption: any type of in-network caching, but also in-network processing, assumes that network operators can intervene on the information path, extract part of the exchanged information (e.g., HTTP headers with information on accessed content) and use it intelligently to infer the content demand and accordingly optimize the cache content and offer the quality of service (QoS) over their networks. However, the widespread use of end-to-end encryption mechanisms such as HTTPS prevents the cache system from interrupting and serving user requests. Solutions that have been proposed to this problem involve replacing end-to-end encryption protocols such as HTTPS with new protocols that perform caching on encrypted contents while preserving user privacy [8].

Memory size constraints: implementing cache nodes on each base station in the network will impose additional costs proportional to its capacity. Thus, deciding the optimal size of this memories depends on the following parameters [2]:

- Cost considerations.
- Skewness of content popularity
- Local traffic distribution in cells.

In the following paragraphs we will review the latest state of art progress regarding coded caching and uncoded femtocaching.

1.2.2 Uncoded Femtocaching

Femtocaching is initially introduced in [1] as a novel way to increase the spectral efficiency of the video transmission over the cellular communication system. It involves the deployment and use of caching helper nodes that store the most popular video contents and make it available

locally to User Terminals when they request it. These nodes operate in parallel to the conventional macro base stations, which provide the UTs with the video files that cannot be obtained from the helpers. The principle is that if there is enough content reuse, i.e. many users are requesting the same few video files, caching can replace backhaul communication. Figure 1.2 shows a possible distribution of the helper nodes within the coverage area of the Macro BS. The question that comes up in such a setting is which content should be stored in each cache - we can distinguish between two scenarios:

a) Each UT (User terminal) is connected to (associated with) only one helper node at each point in time. In this case, the solution is trivial as each cache will store the content that is most popular over the set of users served by the respective helper node.

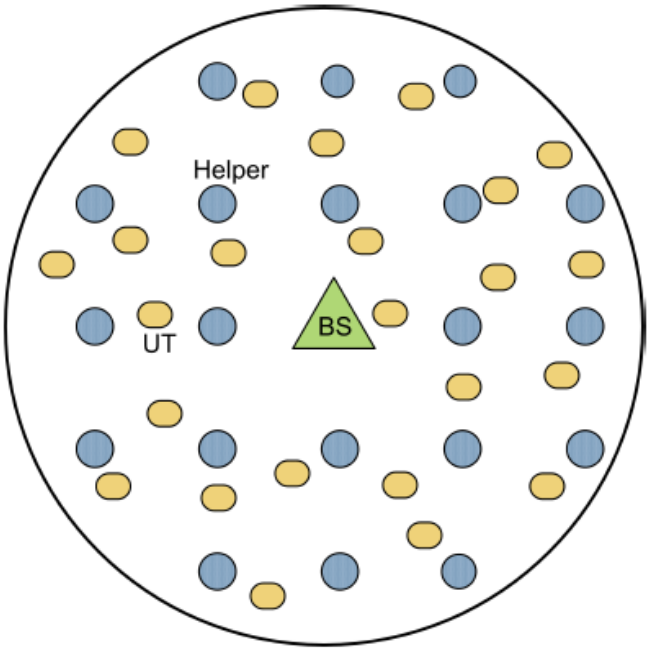

Figure 1.2 An example of the single-cell layout. UTs are randomly distributed, while helpers can be optimally placed in the coverage region of one BS (extracted from [1]).

b) Each UT may be simultaneously connected to more than one helper nodes: in the example presented in [1] for simplicity, they assume that each user is connected to two helpers, each of capacity *M* content items. In this case, the first helper will store the most popular files while the second helper will store the next most popular files. This will create a distributed cache of size 2M for this user. However, the individual objectives of different users may be in conflict and determining the optimal file placement when some users are connected to more than one helper nodes turns out to be an NP-complete problem [1].

In [1], the authors formulate the problem as an instance of maximizing a monotone submodular function over matroid constraints and propose a greedy algorithm for its solution:

- Start with an empty set.
- At each step, add the content item with the highest marginal value to the set while maintaining independence of the solution.

Drawing on a solid theory behind the use of greedy algorithms in such settings, these results suggest that when the objective function of an optimization problem is a submodular function, a greedy algorithm achieves a performance that is provably within a factor 1/2 of the optimal value [9].

Thus, if the user's requested files are already available in the cache "helper" memory, we experience a *local* caching gain [4], which is proportional to the cache memory size. We have a large gain for a large cache as it can store more content likely to be requested; or a small gain, when the hit probability (i.e., the fraction of requests served by the cache) is small, which results in downloading the content over the backhaul connection.

1.2.3 Coded Caching

Authors in [4] proposed a radically different approach to caching content in a network, which can be applied both within a cell (caching takes place at user devices) and across cells (caching takes place at radio access points, as with the femtocaching helper nodes). What is called coded caching deviates from femtocaching in the following ways:

- Helper nodes cache content files partially rather than fully, *i.e.*, they store file segments rather than whole files;
- These segments may generally be encoded versions of the original files;
- When a file is actually requested, it is broadcast together with other requested files as part of a single coded multicast transmission addressing the file requests from all users and caches.

More specifically, under coded caching, all the popular video content will be coded and split to a number of files of a fixed size related to the number of UTs. Moreover, each UT has internal storage memory working as a local cache. Thus, during the placement phase in off-peak hours each user will populate its cache with parts of popular contents. This contents will be chosen properly in order to ensure symmetric properties. Then during the request phase, each user might request different file. The work in [4] shows that we can minimize the number

of transmissions to satisfy all users by applying index coding. Additionally, from the required resource blocks equation K (1 – M/N)/(1 + KM/N), it shows that for a fixed cacheable fraction of the catalogue size M/N, the required number of resource blocks does not increase with the number of users K, as shown in figure 1.3; this is a great advantage over the conventional uncoded caching scheme.

The following toy example illustrates the concept of coded caching [4]. It shows 2 users (K = 2) connected to the cache memory via a shared link with rate R. The catalog size consists of two files A and B (N = 2). For simplicity, we assume that both users have the same memory size. We will have three different cases depending on the local cache size.

Firstly, If the users' caches can store the 2 files (M = 2), both files can be accessed through the local caches so that the broadcast link will not be used (R = 0) and the gain is maximized.

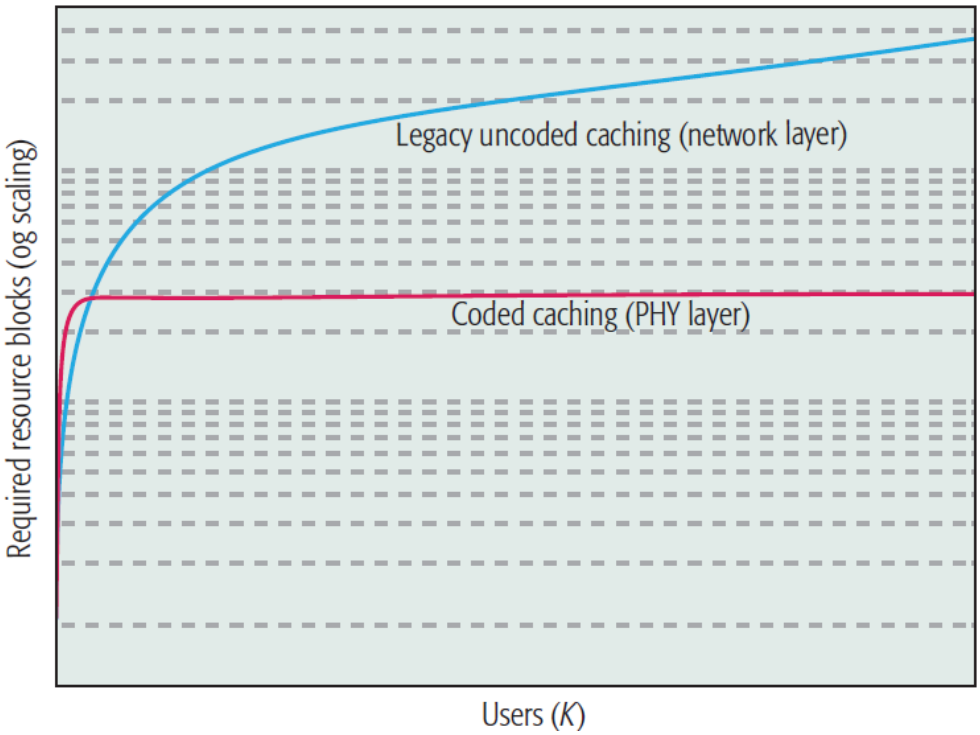

Figure 1.3 Coded caching can serve an arbitrarily large population of users with a fixed number of resource blocks ([extracted from [2]).

In the other extreme case, if M = 0 then no file will be cached in the local memory and the number of files that has to be sent over the broadcast link is R=2. Finally, if M = 1, each user will store one part of each file; for example, user 1 will store A1, B1 and user 2 will store A2, B2. Then during the peak hour, each user will request one of the missing files. If user 1 requests A and user 2 requests B, we can apply the bitwise XOR operator on the B1 and A2 file segments

and broadcast the coded sequence of bits so that both users can recover the requested file by applying bitwise XOR with the corresponding pre-cached file in their local cache. For example, user 1 has already the file A1 and it can extract A2 by bitwise XORing the broadcast transmission with B1. The required transmission rate, in this case, is R = 1 since one coded multicast transmission can satisfy both users' requests. Figure 1.4 shows the cached content and multicast transmissions for all four possible combinations of users' file requests.

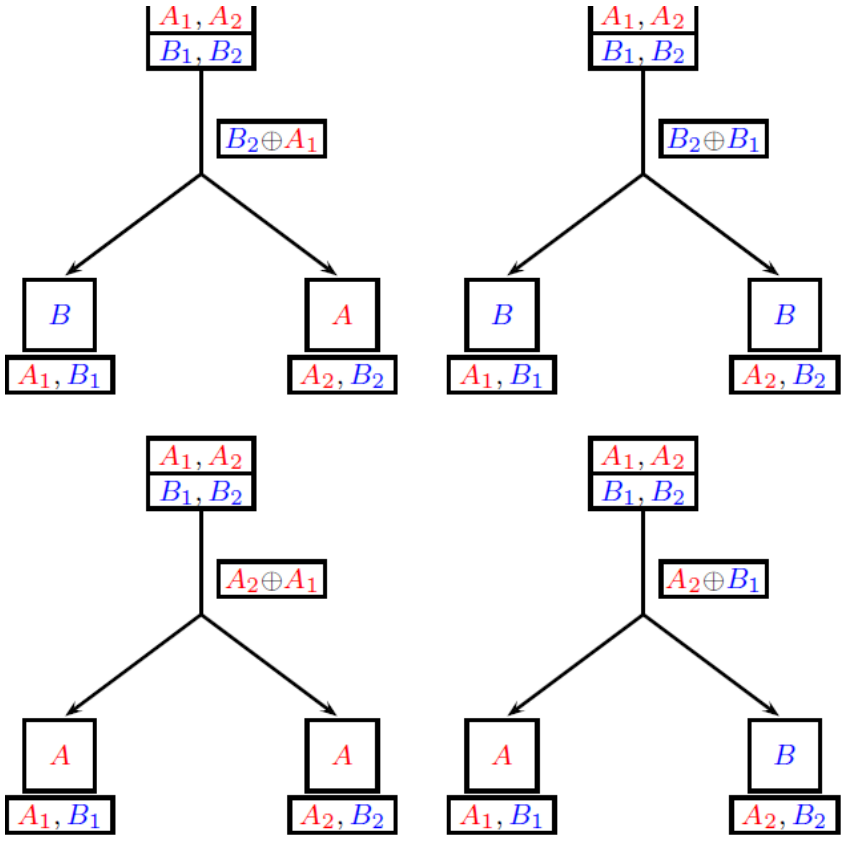

Figure 1.4 Code caching configurations under all four possible combinations of file requests by the two users (M =2, N=2, K=2) (extracted from [4])

The main savings with coded caching is in what is actually sent over the broadcast link that feeds the caches. Using information-theoretic formulations, the technique achieves *a global caching gain*, on top of the local caching gain of conventional schemes, leading to multiplicative enhancement of the overall caching efficiency.

The algorithm that Maddah-Ali and Niesen proposed for the coded caching configuration assumes uniform demands, i.e., all users are equally interested in all items, and proceeds in the following manner:

Algorithm

Placement phase:

a. Each one of the N files in the catalogue is split into overlapping subfiles of equal size F/t, where t = MK/N; (M: cache size, K: number of users, F: file size)

b. Each of these subfiles is stored in a subset of caches of size t. Hence, each cache stores a total of N subfiles

Delivery phase:

a. Users transmit requests for specific files –in the worst case, for K different files

b. For each subset of users of size t+1, we XOR t+1 subfiles, each with the property that one of the t+1 users is missing it whereas the other t have it in their caches.

This algorithm is centralized, i.e., the choices of what to store in each cache during the placement phase are centrally coordinated.

In [10] the same group relax the assumption for central coordination during the content placement phase. Each user independently stores the same number of (random) bits from each of the N files. The transfer/request phase and the assumptions about the popularity of content are identical with the original scheme in [4]. The coded rate of this scheme is

$$R_D(M) \simeq K.(1 - M/N).min\{\frac{N}{KM}(1-(1-M/N)^K), \frac{N}{K}\}.$$

We note that if $N \geq K$ or $M \geq 1$, then the minimum in $R_D(M)$ is achieved by the first term so that

$$R_D(M) = K.(1 - M/N).\frac{N}{KM}(1-(1-M/N)^K) .$$

In parallel, in [11], researchers from the same research group relax the assumption of uniform demand for all items. They consider arbitrary demands but still identical distributions at the cache/user level, i.e., there is no differentiation between users. The proposed algorithm steps in each phase are as follows:

Placement phase:

a. Files are organized into *L* groups featuring similar demand, that is, after they are sorted in order of decreasing popularity, group k includes files with demand popularity ranging in [p_k,p_k/2], namely all files with a popularity differing by a factor of 2.

b. Different amounts of memory are assigned to each group of files. These amounts are proportional to their popularity. The amount of memory assigned to each group is the same over all caches. Each file in a given group reserves the same portion of the group-specific memory.

c. Users store randomly the same number of bits per file within the same group, the space occupied by each file depending on the (popularity of) the group it belongs to.

Delivery phase:

a. Users transmit requests for specific files –in the worst case for K different files. These requests are now partitioned into L groups, depending on which group the requested file belongs to.
b. The decentralized coded caching scheme is applied separately for each one of the L groups, for the number of files N_l, users K_l and memory M_l that correspond to each group l.

The *expected* coded rate of this scheme was derived to be

$$\sum_{d \in N^K} (\prod_{k=1}^{K} P_{d_k}) R_d.$$

Where (d_k: is the request of user K, R_d: the corresponding rate, P_{d_k}: the probability of user request).

Besides and beyond the theoretical work, in [5] the authors have implemented a fully working prototype on CorteXlab that facilitates the wireless multi-user communication scenarios, which help them in testing state-of-art of both coded and uncoded schemes in real-world environment. The results of their experiments show that the coding overhead does not significantly affect the promising performance gains of coded multicasting in small-scale real-world scenarios, practically enforcing its potential to become a key next generation 5G technology.

1.3 Objectives of the Dissertation

Standard uncoded caching techniques and coded caching have emerged so far as two alternative approaches towards the proliferation of edge caching in mobile cellular networks. Both concepts have seen much theoretical work that has shed light to fundamental properties, both advantages and drawbacks, of the two schemes.

One persistent element in most of these theoretical studies is the (simplified) assumption about the temporal and spatial dynamics of content demand. The original coded caching scheme was studied under the assumption of uniform demand for all content items, identical over all caches [4]. Although subsequent work [11] allowed for arbitrary distributions of content demand, however it preserved the assumption for identical demand distributions over all caches (users). Likewise, in the case of the femtocaching work, the reference popularity distribution of content is the global one, i.e., the aggregate of all users' demands under all helper nodes in the network. Our intention in this work is to look deeper into the implications of the spatial locality of content demand for the two caching schemes. Intuitively, in an extreme setting where each user in a cell (cache in a radio network) present completely distinct preferences for content (different

than its peers), it might make far more sense to use a scheme like LFU (Least Frequently Used) for determining the cache placement, working independently for each cache/user. On the other hand, under identical demand for content, coded caching presents a non-negligible gain. The question that plausibly emerges is when exactly, i.e., under what characteristics of the content popularity, is the one scheme preferable to the other.

Therefore, the overarching objective of this dissertation is **to carry out a systematic comparative study of the two techniques through simulations that will thoroughly explore the impact of spatial locality in content demand**.

To this end, it will aim to

- Study and model the spatial locality of content demand over the geographical area of some hundreds of network cells. This task will draw on synthetic distributions but may benefit from real data that may be become available in the course of the dissertation;
- Define plausible metrics for quantifying the spatial locality of content demand across an area of the network;
- Determine conditions, e.g., values of the metrics assessing the spatial locality, that could indicate when coded caching is more efficient than femtocaching and when the opposite holds.

1.4 Outline of the Dissertation

The remainder of this report will be organized as follows:

Chapter 2 presents the system model and assumptions as well as the methodology that is adopted in the comparison of the two caching techniques. This chapter also discusses our work on spatial locality characterization and the definition of metrics we use for this comparison. In chapter 3 we present the results of our simulation study, insisting on the sensitivity of the comparison outcome to the different system parameters.

Chapter 4 summarizes the findings of the experimental study and outline directions for future work.

Chapter 2 System Model and Methodology

This chapter includes two subsections. The first one presents the system model for our investigation. This includes the assumption about the network layout, the users, and the cache placement as well as the model for the content popularity. The second part presents the methodology that will be used for carrying out the comparison of the two caching techniques. This encompasses a description of the implementation of the caching algorithms and the performance metrics that drive the comparison.

2.1 System Model

The system model is outlined in Fig. 2.1 and includes:

2.1 Small cells, caches, and users

We consider a number K of small cells (small base stations) that serve varying numbers of users each. The cells provide coverage to a large area, which may include business districts and residence areas.

The network users issue requests for content. The content catalogue, in its entirety, is stored at a powerful content server. Let N be the size of the content catalogue and assume, for simplicity, that file sizes in bits are identical and denoted by F.

The cells are equipped with caches that can store locally content that users may request. The normalized capacity of these caches, in terms of number of files, is finite, M, letting them store only a small part of the overall content catalogue. These caches are connected to the content server through an error-free shared backhaul link, that could generally be wireless or wired.

Users are at each point in time associated with a single cell. Their association may change over time but exactly one cell can serve them content at any particular time instance.

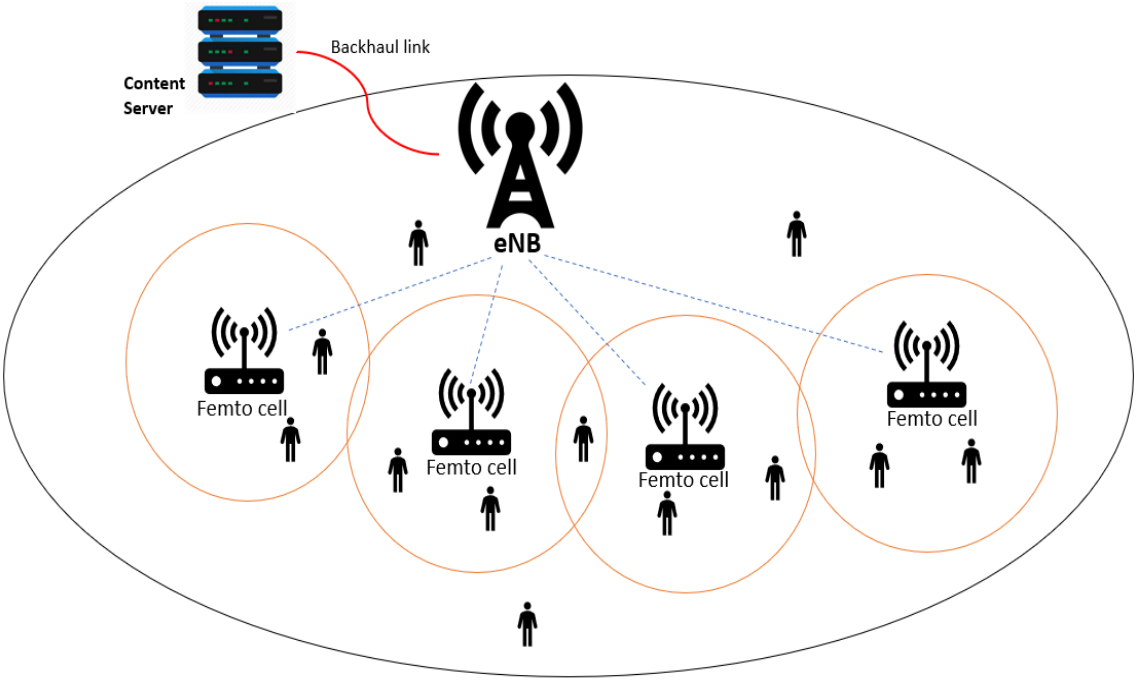

Figure 2.1 A sketch of the network model. Users are served by small cache-enabled cells that store locally content of appeal to the users. These cells are fed with content from a central server with far more capacity, which can store practically the whole catalogue of files that is available to users. A backhaul, which may be realized either through wired or wireless technologies feeds the caches of the small cells

The assumption for the content provision is that it operates in two phases: a placement phase and a delivery phase.

The *content placement phase*, is carried out periodically during the off-peak hours. It determines the content to be stored at the caches of the small cells and places it there. This content then remains the same till the next execution of the content placement phase. In the time interval between two content placement instances, the *content delivery phase*, the cache serves the requests it receives by users associated with the cell it is attached to.
What will be stored during the placement phase and what will be delivered during the delivery phase changes dramatically according to the implemented edge caching approach, uncoded or coded. We discuss this point next.

2.1.2 Caching Algorithms

Our comparison considers two main approaches to caching, uncoded and the coded.

2.1.2.1 Uncoded caching: Highest-Popularity First (HPF)

This is one of the simplest and most popular caching strategies, which is used for non-uniform content distribution, when the files' popularity is well-known. This caching strategy is an off-line equivalent to the online Least Frequently Used (LFU) caching algorithm, hereafter the

terms LFU and HPF denote the same things and are used interchangeably, whereby the item that is least-frequently used is evicted from the cache. HPF is optimal for a system with a single cache [11] when the cache stores the M most popular files. However, HPF can be arbitrarily suboptimal in the multi-cache setting (K >1) [11].

To model the HPF caching algorithm, we sort the files in descending popularity order and populate the caches with first M (depending on the cache size) files. The aggregate popularity of these M files yields an estimate for the expected cache hit ratio experienced at the cache. The aggregated (global) popularity illustrated in the following equation:

fpopg(i) = 1-product_{k=1:K} (1-fpop({k}(i))

Thus, to infer the global demand fpopg(i) of an item i over all K caches out of the individual ones, it would be the probability that item i is requested by at least one cache is 1 minus the probability that no cache requests it.

We attach the code that represents our HPF model in appendix A.

2.1.2.2. Coded caching: Decentralized Coded Caching algorithm for non-uniform demand distribution

The Decentralized coded caching algorithm introduced in [10] generalizes the concept of centralized coded caching, as stated earlier in chapter 1, and solves the following restrictions: first, under centralized CC, the number of users must be known well in advance of the delivery phase. Secondly, there must be tight coordination of the small cell caches as to which file parts need to be stored during the placement phase. On the contrary, the DCC algorithm introduced in [10] creates a simultaneous coded-multicasting opportunity without coordination in the placement phase. In the delivery phase, the small cell caches still need to inform the server about their cached contents and the file requests they get. The algorithm efficiently exploits the multicasting opportunities created during the placement phase and allows DCC to achieve a performance close to the performance of the ideal centralized CC. The decentralized CC algorithm is outlined below:

Placement phase:

Each of K small cells with cache size M, independently caches a subset of $\frac{M F}{N}$bits of each file $n \in [N]$ that has the size F. those bits are chosen uniformly at random.

Delivery phase:

This algorithm provides two delivery procedures, the server chooses the procedure minimizing the resulting rate over the shared link out of the two following ones:

- The first procedure: the servers organizes the caches in all possible groups of sizes 1,2,…K-1. For each of these groups *s*, and depending on the file bits that are common among the caches in *s*, the server XORs and multicasts bits, which are requested by a user in cache k and are only stored at every cache in s excluding k.
- The second procedure: for every file $n \in [N]$, the server sends enough random linear combinations of bits from file *n* for all users requesting it to decode.

We attached our implementation of the decentralized CC algorithm in appendix B.

This algorithm assumes the files have a uniform popularity distribution. Therefore, the follow-up work in [11] proposed a way to deal with Zipf-distributed file popularities by grouping together files that have "similar" popularities, as described earlier in section 1.2.
In our experimental study, we compare this adaptation of the decentralized CC scheme for non-uniform file demand against the HPF algorithm.

2.2 Evaluation Methodology

- **Generating local and global content popularity distributions:**

  We used power law function as a simple approach that demonstrates content popularity. It has the parameter theta that controls the skewness of the popularity with value ranges between 0.1 and 2.

  We generated the local popularity for N files, and then we normalize and replicate this popularity to k number of users. The output is K cells with N files popularity each. Finally, we generate the global popularity by summing the popularity of the file n with its corresponding file in all K cells.

2.2.1 Content Popularity

In our evaluation, we explicitly distinguish between local popularity (at the level of one cache) vs. global popularity, when these local popularity distributions are aggregated over all (or a subset of) small cells. This interplay between the popularity distributions at different spatial scale is important for assessing the impact of demand's spatial locality.
Thus, we consider between two main scenarios regarding the content popularity (a.k.a demand distribution):

a. Uniform demand

In this scenario, which serves as the baseline for our comparisons, all N files of the catalogue are assumed to be requested with the same probability, i.e., the probability that a request from any small cell cache K is for a file n is equal to 1/N. For the coded caching algorithm, this implies that each cache stores an equal portion, M/N, of every file in the content catalogue.

b. Non-uniform file demand

This is the more realistic scenario, which lies at the focus of our study. The popularity varies from one file to another as well as from one cache to the other. We consider a number of models for capturing this heterogeneity, all of them drawing one way or another to the *Zipf* popularity distribution. According to it, the content item *n* can be requested with the rate $\lambda\, p_n$ [2] where λ denotes the rate of requests and $p_n$the power-law distribution ($p_n \simeq n^{-\theta}, \theta \succ 0$); θ is used to control the skewness of the popularity. Specifically, large θ (highly right skewed histogram) represents a few files with similar popularity. On the other hand, small θ (flat histogram) represents a lot of files with similar popularity as shown in figure 2.2.

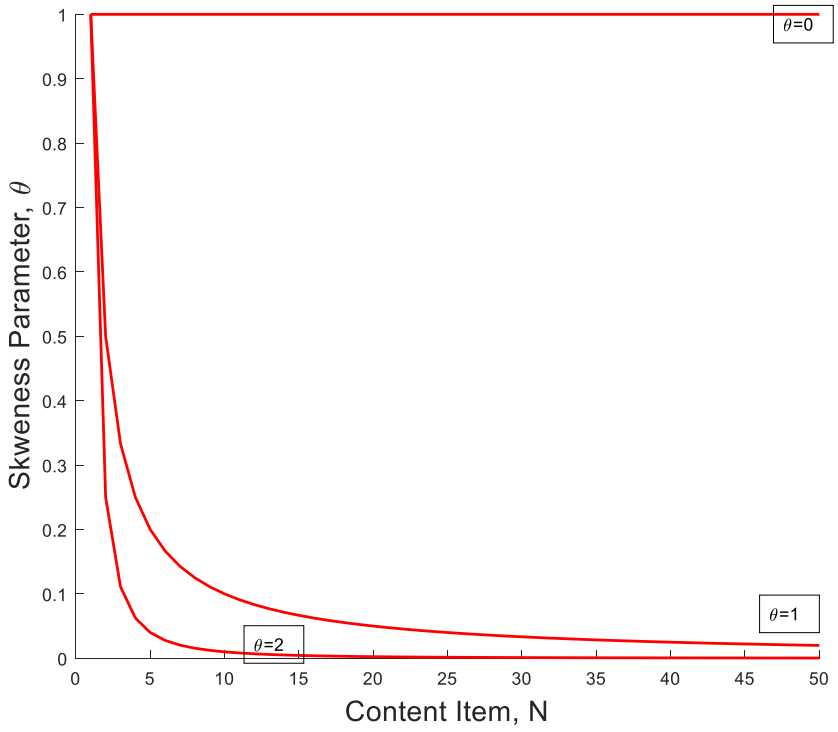

Figure 2.2 Power law distribution for different values of θ.

The models for the demand heterogeneity across the network are the following:

- Identical file set ranking of items and skewness parameters across the *K* caches (**POPa**):

In this scenario, for catalog of size N file. Every user k shares an identical content demand with K-1 users. In other words: that the users have identical file distribution for every value of theta.

- Identical file set skewness parameters but different ranking of content items across the *K* caches (**POPb**): In this model, the assumption is that users have random permutation regarding the order of files.
- Different file set and skewness parameter across the *K* caches (**POPc**):
  In this model, the assumption is, for a catalog of size N, each user has an interest in a subset of files of size N_S, fixed for all users, each user has an interest in a different subset of N. However, there is a partial overlap of size O between subsets of each subsequent users.

2.2.2 Performance metrics

Two performance metrics are mainly relevant to our comparison, the cache hit ratio and the bandwidth savings at the backhaul links that are possible with the two techniques.

2.2.2.1 Cache hit-ratio

The effectiveness of cache memory is measured in terms of the cache hit ratio, which is defined as the ratio of the number of cache hits (the fraction of requests that can be served by the cache) over the number of requests, usually expressed as a percentage. The hit ratio is generally influenced by factors such as the cache policy, the number of cacheable objects, the cache memory size. Typically, what is sought after is an efficient caching algorithm that maximizes the hit ratio and minimizes the cache misses. A cache hit ratio of 90% and higher implies that most of the requests are satisfied by the cache. A value below 80% on static files indicates inefficient caching due to poor configuration [13].

2.2.2.2 Backhaul rate

In the mobile networks, the backhaul link represents the connection between the network edge (base station) and the core network. It also comprises the intermediate links between the core network. The rate of the backhaul link could be at the order of E1 or ADSL level. Therefore, mitigating backhaul rate requires implementing caching algorithms that provide maximum gain (law backhaul rate).

The main promise of coded caching approaches is that they can significantly reduce the rate that is required at the backhaul links, yielding gains that are orders of size better than

conventional uncoded caching policies. Below, we summarize the achievable rates under the HPF scheme and the coded caching algorithms we have discussed so far (under aggregate global demand distributions).

1- Uncoded caching (HPF).

$$R(M;K,N) = K\,(1 - M/N)$$

2- Centralized coded caching.

$$R_C(M;K,N) = K.(1 - M/N)(1/(1 + KM/N))$$

3-Decentralized coded caching.

$$R_D(M;K,N) = K.(1 - M/N).((1 - (1 - M/N)^K)/(KM/N))$$

4. Decentralized coded caching with file grouping under non-uniform demand

$$R_F = \sum_{l=1}^{L} R_D(M_l;K,N_l)$$

where L is the number of file groups with "similar" demand determined by the scheme, Nl is the number of files in each one of these groups and Ml is the amount of storage space allocated to each one of these groups in each cache.

It is obvious that with all caching techniques we can achieve a minimum gain $(1 - M/N)$, which is called "*local caching gain"*. This gain becomes significant when the local cache size M in the order of N. In coded caching, we can also have an additional gain called "*Global caching gain"* $(1/(1 + KM/N)$, which can be significant when KM is in the order of N. The optimality of these rates is a separate research thread within the Information Theory community [4].

The software that we used for carrying out the simulation study is MATLAB. Input to the analysis are the synthetic distributions for the popularity of content at local scale, generated in line with section 2.2.1

Chapter 3 Evaluation

This chapter presents and discusses the results of the comparative simulation study undertaken in the context of the dissertation. As it is mentioned in chapter 2, we used MATLAB software to simulate the performance of the three caching schemes under different assumptions and scenarios. In the first section, we compare HPF "high popularity first", Decentralized (DCC), and Centralized coded caching (CC) under uniform content popularity assumption. Then, in the second section, we test the performance of HPF and DCC schemes under non-uniform content popularity.

3.1 Uniform Content Popularity: HPF vs DCC vs CC

This section reports the comparison of HPF, DCC, and CC caching algorithms under uniform content popularity assumption. These reports show the performance of these caching algorithms as a result of changes in the cache size M and the number of files N.

3.1.1 Impact of the cache size, *M*

Figure 3.1 presents the rates of studied algorithms as we increase the cache capacity M, for a fixed number of users K=9 and catalog size N=600.

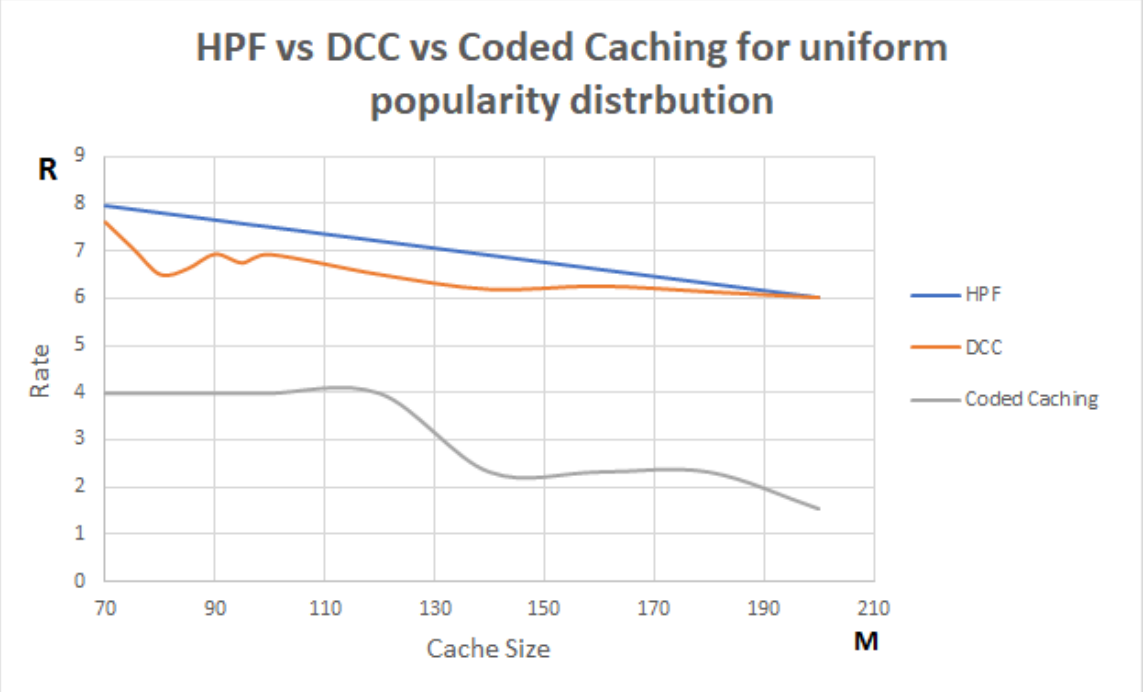

Figure 3.1 Cache Size Vs rate performance comparison of HPF, DCC and Coded caching algorithms for uniform content popularity

Figure 3.1 demonstrates that the coded caching algorithms provide higher gains; namely, smaller rates compared to the HPF and DCC schemes. In addition, it shows that the rates of CC and DCC drop more slowly than under HPF (linear decrease) as we increase the memory size *M*.

3.1.2 The impact of the number of files, N

While the number of files (N) is the variable in this experiment, the cache size (M) and number of users (K) are chosen to be 30 and 9 respectively. Figure 3.2 demonstrates that both HPF and DCC data rate go up slowly with the growing value of the file size N, leveling off around 7.5 at N=200. On the other hand, coded caching algorithm slowly responds to the large increase in the catalog size. The CC rate is 4 for catalog sizes ranging from 140 to 200 files.

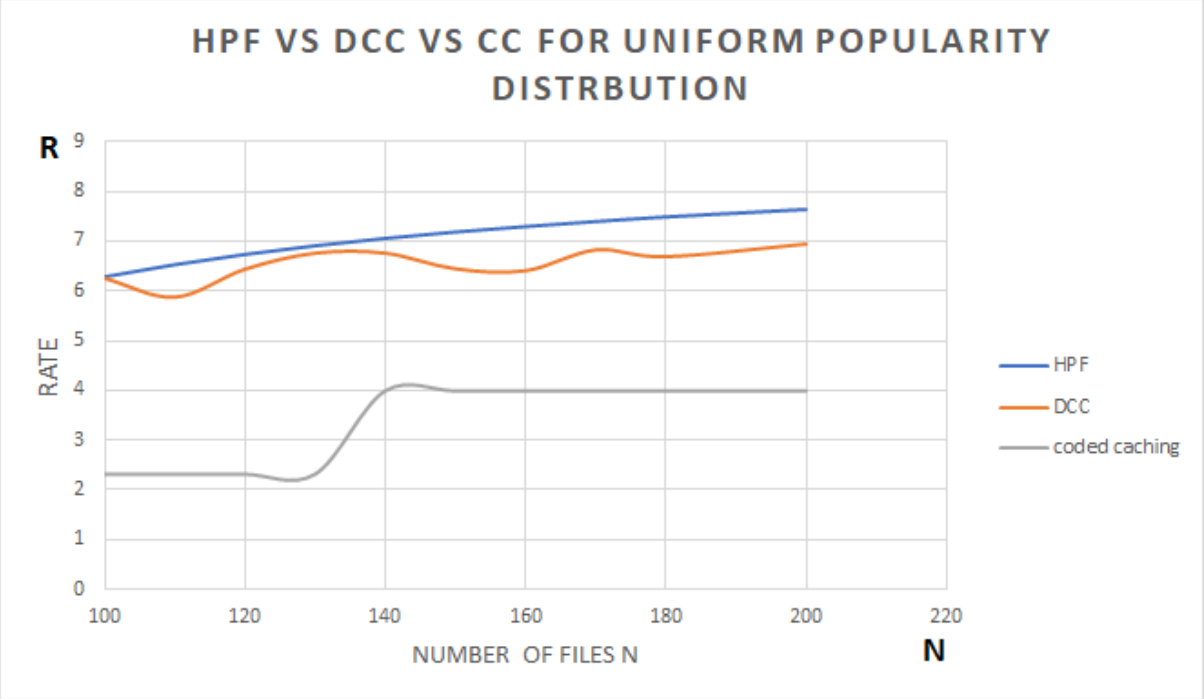

Figure 3.2 number of files N Vs rate performance comparison of HPF, DCC and CC caching algorithms for uniform content popularity

Remark: we notice from simulations plots in Figures 3.1 and 3.2 that CC algorithms has plateau behavior at different values of N and M as they are used to calculate the number of segments as well as segment size (section 2.1.2.2.), Thus, those parameters should be selected carefully to achieve the optimum performance of the algorithm.

3.2 Non-Uniform Content Popularity: HPF vs DCC with file grouping

In this section, we report the performance of HPF and DCC caching schemes under non-uniform content popularity assumption. First, we compare the mentioned schemes in response to changes in the skewness parameter θ and cache size M for three different scenarios of demand heterogeneity: POPa, POPb and POPc. Second, we report the comparison of those schemes with previous file order scenarios for a selected skewness parameter θ values as a function of the cache size M.

Remark: In the following figures, each point in the DCC curve corresponds to the average of 10 repetitions for each placement and request phase. Therefore, the resulting DCC curve has small deviations with more realistic values.

3.2.1 Impact of the Zipf skewness parameter θ

In this section, we focus on the skewness parameter θ, which is let range from 0.1 to 2, for all three scenarios of demand heterogeneity.

a) Identical file set ranking of items and skewness parameters across the K caches (POPa):

For catalog size N=300, users K=9, file size F=2000 bits and three different values of cache size M= (10,30,60) the simulation produces the following figure:

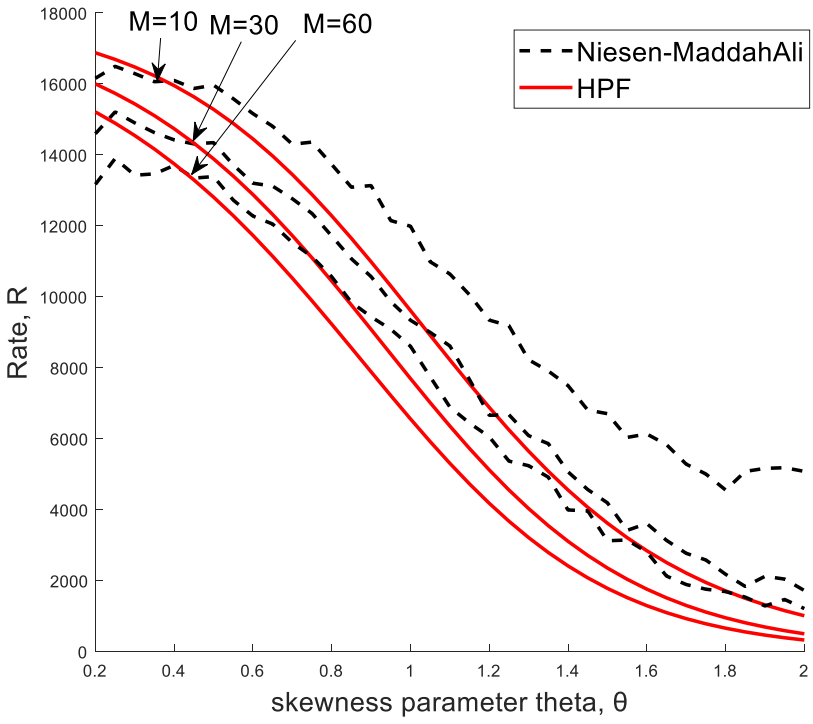

Figure 3.3 Backhaul rate vs. Zipf skewness parameter", performance comparison of HPF and DCC caching algorithms under scenario POPa

Inspecting the previous figure reveals some remarkable trends. In both schemes, the required rates decrease as we increase the skewness parameter θ. In addition, it shows the fluctuation of the DCC curves as a function of the small variations of theta. However, this is not the case for HPF, which has smooth curves without any fluctuation.

Moreover, with the same previously mentioned parameters as an input, we repeated the same experiment for three different values of the cache size M that produced the three curves shown in the figure. It can be noticed that in both schemes we have a considerable gain produced when the cache size M increases from 10 to 30 and 60. This gain varies with different values of theta. Another significant behavior is for a small cache size M=10, HPF demonstrates better performance for most values of theta. Moreover, DCC shows better results for theta values between 0.1 and 0.45 compared to HPF for M values equal to 30, 60 files. However, this is not true for the rest of theta values. The turning point change as we change the cache size. In other words, for M=10 the turning point is around 0.3 where the HPF start outperform DCC, However, the turning point become around 0.45 when the cache size increase to 60.

b) Identical file set skewness parameters but different ranking of content items across the *K* caches (POPb):

Figure 3.4 shows the comparison of the HPF and DCC (Niesen-MaddahAli) schemes for K=8 users, N=200 files, and F=2000 bits. Three curves are produced for each scheme, for cache sizes M=10, 30, and 60 files.

Considering the performance shown in figure 3.5, it can be concluded that the HPF rate drops proportionally with theta. On the contrary, this is not the case for DCC, which shows a different response for different theta values. Moreover, in the DCC scheme we have a huge gain imposed when the cache size increases from 10 to 30 and 60. For example, for theta =0.2 the DCC rate drops from 14000 to 5000 bits at M=60. In contrast, HPF shows smooth response without deviation with a considerable gain resulted in increasing the cache size to 30 or 60; however, this gain decreases as a function of theta. Last but not least, it can be viewed that, in the 3rd curve where M=60, the DCC scheme performs better for theta values between 0.1 and 0.6 (this range gets smaller for small M values) and has a considerable gain compared to HPF< which performs better for theta ranging between 0.6 and 2.

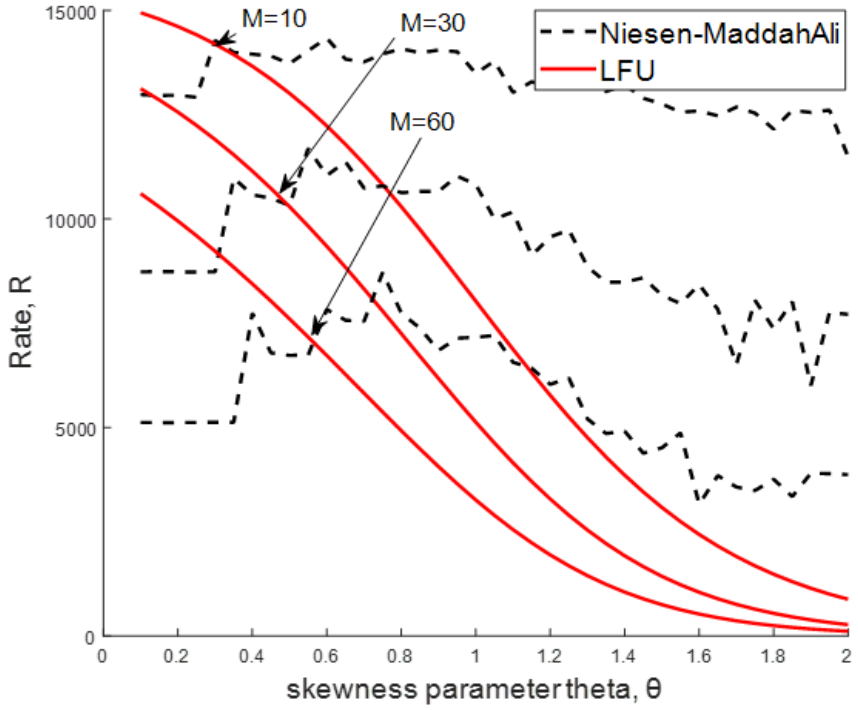

Figure 3.4 "Backhaul rate vs. Zipf skewness parameter", performance comparison of HPF and DCC caching algorithms under scenario POPb

c) Different file set and skewness parameter across the *K* caches (POPc):

Figure 3.5 shows the comparison of the HPF and DCC schemes for K=8 users, N=260 files, F=2000 bits, subset size N_S=50 and partial overlaps of size O=20 bits. Three curves are produce for each scheme for cache sizes M=10,30 and 60 files.

It can be noticed that HPF outperforms DCC for all theta values. In addition, LFU curves drop faster as a function of theta compared to DCC, which drop far more slowly. Moreover, for both schemes, there is a huge gain imposed by increasing the cache size from 10 to 30 and 60. However, the required rate under HPF decreases to zero for cache size=60 files, which is a plausible result since each user can cache all the requested files.

We measured the running times required for the simulation runs, needed for Fig. 3.6; these are recorded in the table below:

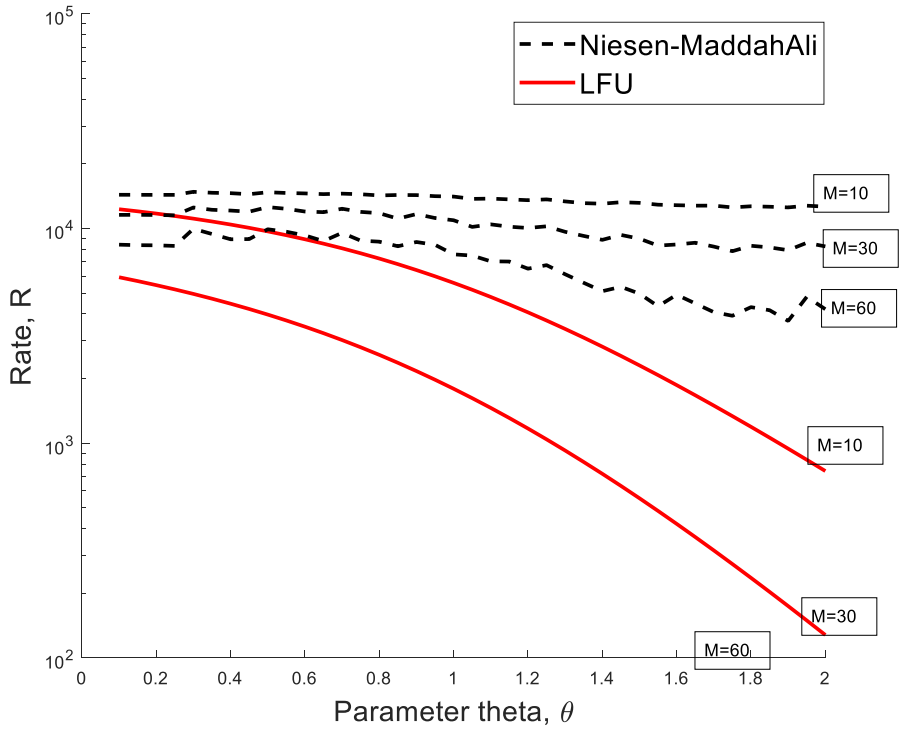

Figure 3.5 "Backhaul rate vs. Zipf skewness parameter", performance comparison of HPF and DCC caching algorithms under scenario POPc

We measured that time consumed of running the simulation, skewness parameter θ Vs rate, a performance comparison of HPF and DCC caching algorithms for the three mentioned scenarios recorded it in the table below:

Table 1 Simulation run times for deriving the Fig. 3.6 plot, performance comparison of HPF and DCC caching algorithms for the three different scenarios

Scenario	Run Time
POPa	18
POPb	64 minutes
POPc	31 minutes

Viewing the table 3.1, it can be noticed that POPb consume considerable time compared to the other two scenarios. In our simulation we consider a system with 8 users. It would be inefficient to implement such caching schemes that support a large number of users without improving their computational complexity.

3.2.2 Impact of the cache capacity, M

This section discusses the performance of two studied schemes regarding the change in the cache capacity M for selected values of the skewness parameter θ based on the behavior in figures 3.4, 3.5 and 3.6 considering the scenarios discussed earlier in the previous section.

1- theta θ =0.3.

Figure 3.6 shows the comparison of HPF vs DCC caching schemes under scenario POPa. It shows that both rates drop proportional to the cash capacity M. However, DCC scheme shows better results where its curve drops faster.

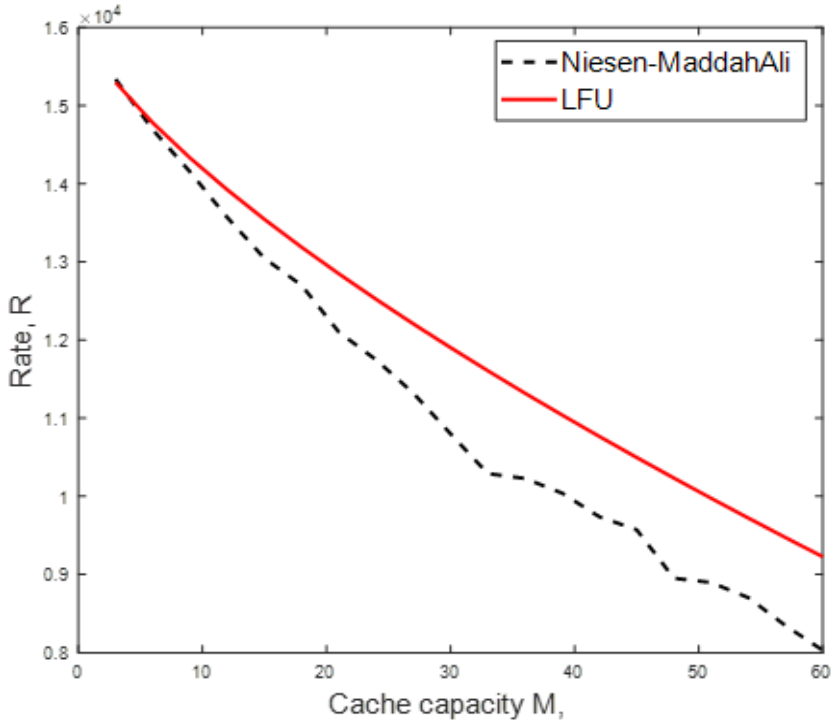

Figure 3.6 "Backhaul rate vs Cache Capacity M", performance comparison of HPF and DCC caching algorithms under scenario POPa (theta = 0.3)

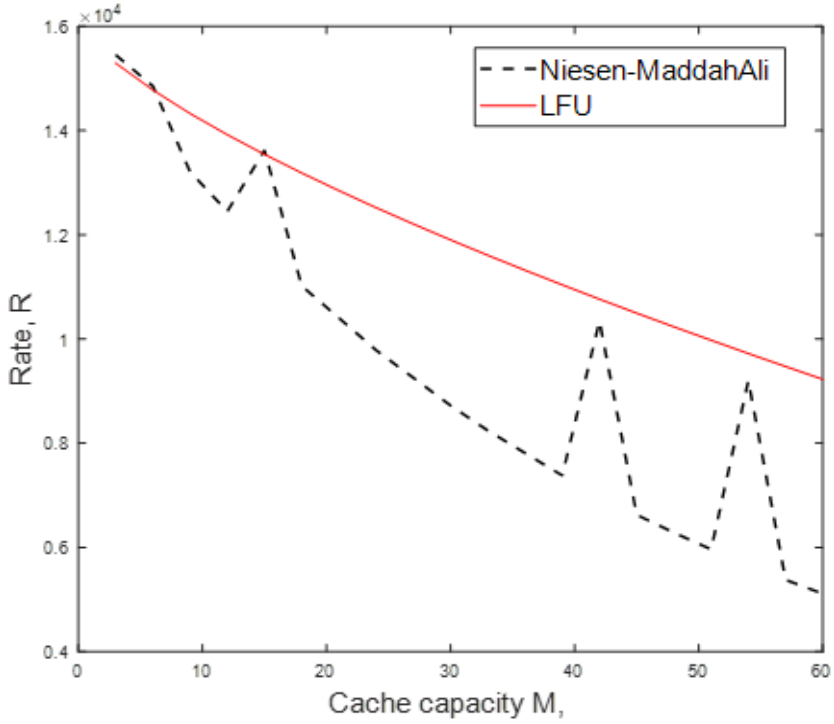

Figure 3.7 "Backhaul rate vs Cache Capacity M", performance comparison of HPF and DCC caching algorithms under scenario POPb (theta = 0.3)

Figure 3.7 shows the comparison of HPF vs DCC caching schemes under scenario POPb. It can be viewed that both rates drop proportional to the cash capacity M. However, DCC curve drops faster with multiple fluctuations at multiple cache size values.

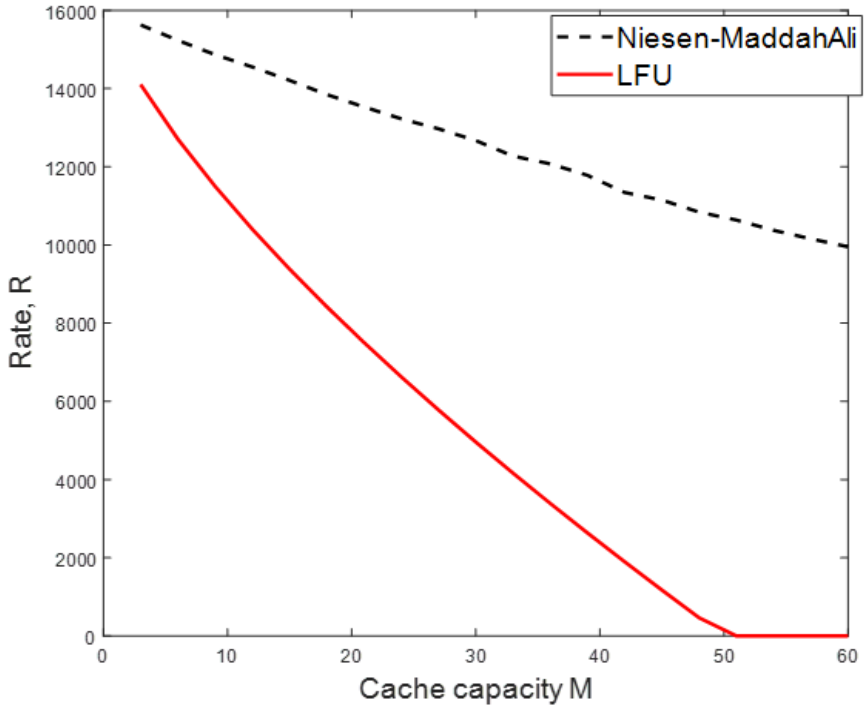

Figure 3.8 "Backhaul rate vs Cache Capacity M", performance comparison of HPF and DCC caching algorithms under scenario POPc (theta = 0.3)

Figure 3.8 shows the comparison of HPF vs DCC caching schemes under scenario POPc. Unlike the curves shown in figure 3.8 and 3.7, it can be noticed that LFU show a better performance than DCC for skewness parameter θ=0.3. Furthermore, it shows that LFU rate drops linearly as a function of the M compared to DCC rate which decreases slowly with M.

2 - theta θ =0.4:

Figure 3.9 shows the comparison of HPF vs DCC caching schemes under scenario POPa. It can be noticed that both rates have similar performance for this popularity parameter. It also shows that both rates drop linearly proportional to the cash capacity M. Moreover, DCC curve shows few fluctuations compared to HPF curve which has a smooth drop proportional to M.

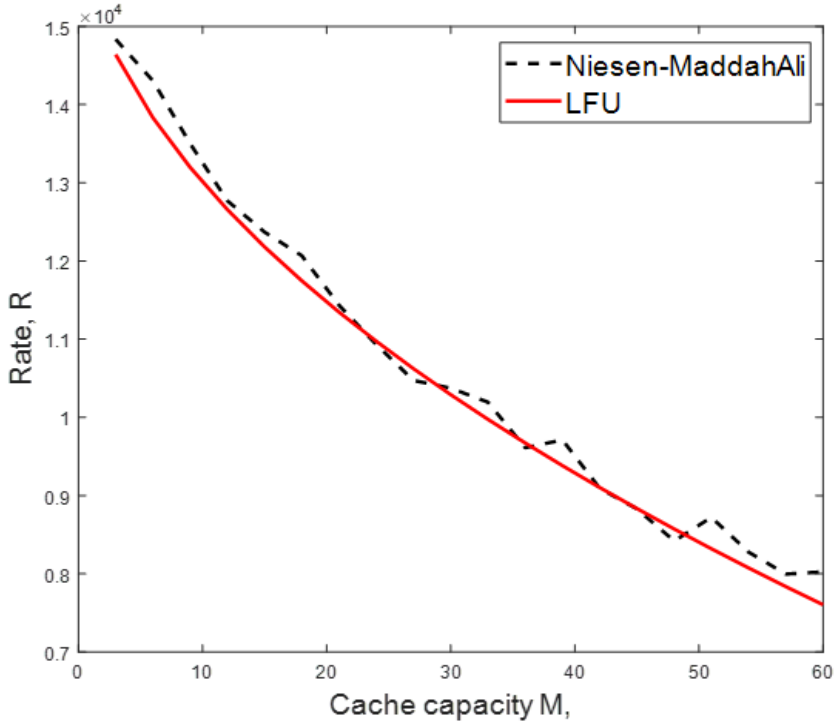

Figure 3.9 "Backhaul rate vs Cache Capacity M", performance comparison of HPF and DCC caching algorithms under scenario POPa (theta = 0.4)

Figure 3.10 shows the comparison of HPF vs DCC caching under scenario POPb. It can be noticed that both rates drop proportionally to the cache capacity M. It also shows that both schemes have similar performance for small cache size between 0 and 15 files. However, DCC scheme does a better performance for the cache size range 15 to 60 files per cache. Besides, DCC curve has small fluctuation compared to HPF which has a smooth curve.

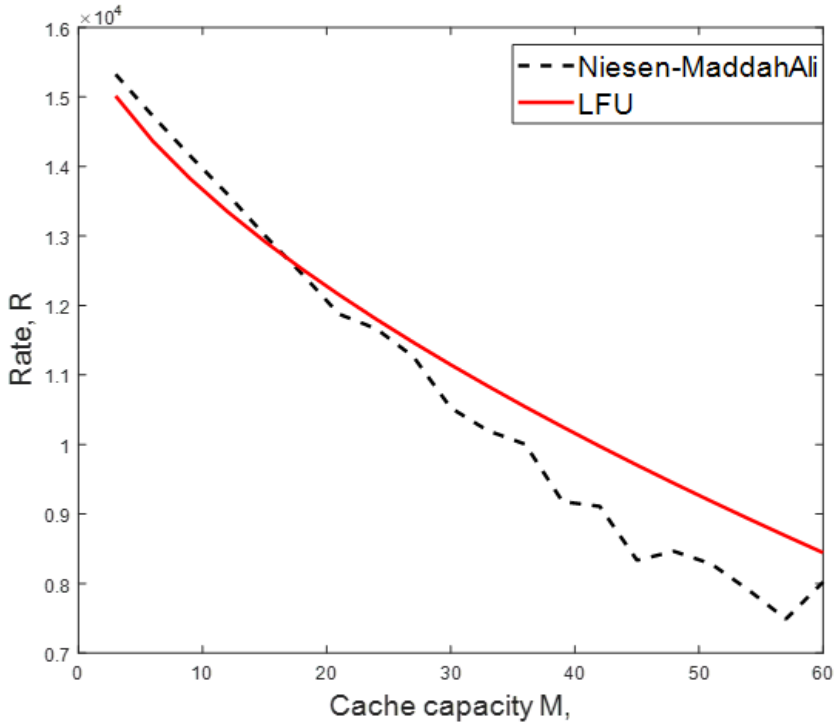

Figure 3.10 "Backhaul rate vs Cache Capacity M", performance comparison of HPF and DCC caching algorithms under scenario POPb (theta = 0.4)

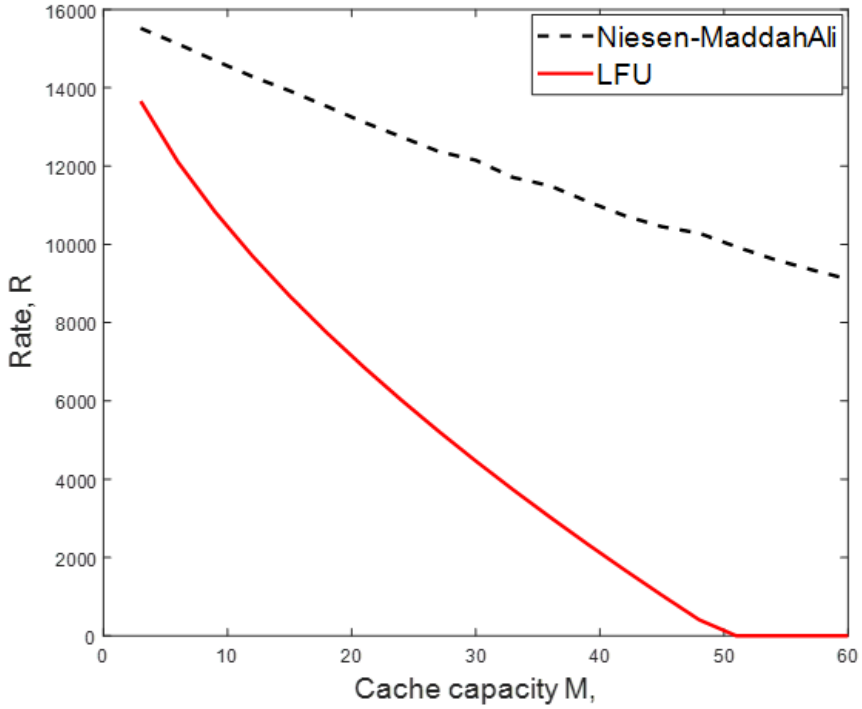

Figure 3.11 "Backhaul rate vs Cache Capacity M", performance comparison of HPF and DCC caching algorithms under scenario POPc (theta = 0.4)

Figure 3.11 shows the comparison of HPF vs DCC caching schemes under scenario POPc. In this figure, it can be noticed that both curves show the same performance shown in figure 3.8 where the popularity parameter=0.3.

3 - theta θ =0.5:

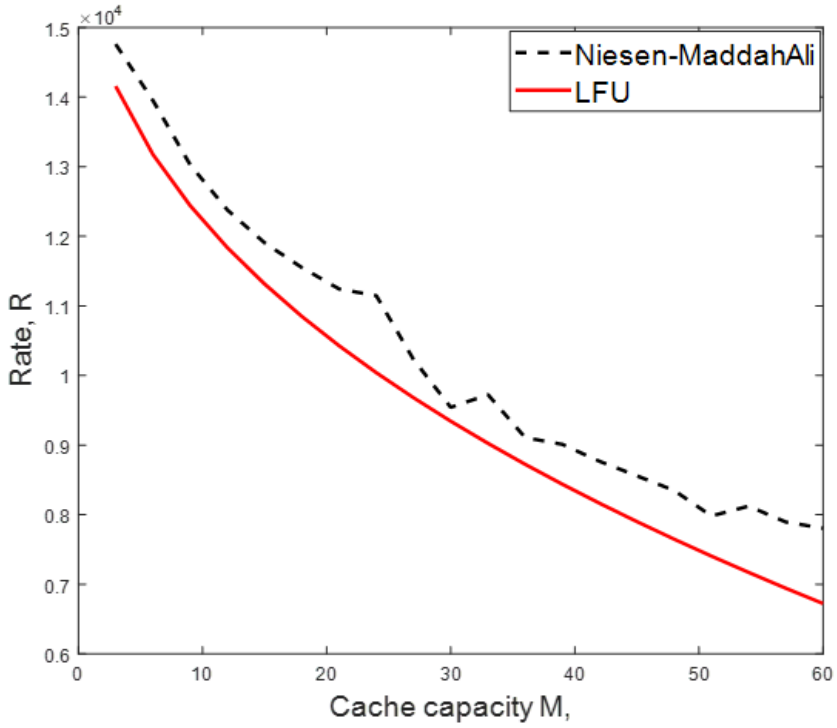

Figure 3.12 "Backhaul rate vs Cache Capacity M", performance comparison of HPF and DCC caching algorithms under scenario POPa (theta = 0.5)

Figure 3.12 shows the comparison of HPF vs DCC caching under scenario POPa. It can be noticed that starting from the current value of the popularity parameter θ=0.5, HPF will show a better performance as we will show in the next figures. It also shows that both rates drop proportionally to the cash capacity M. Moreover, DCC curve shows few fluctuations compared to HPF curve which has a smooth drop proportional to M.

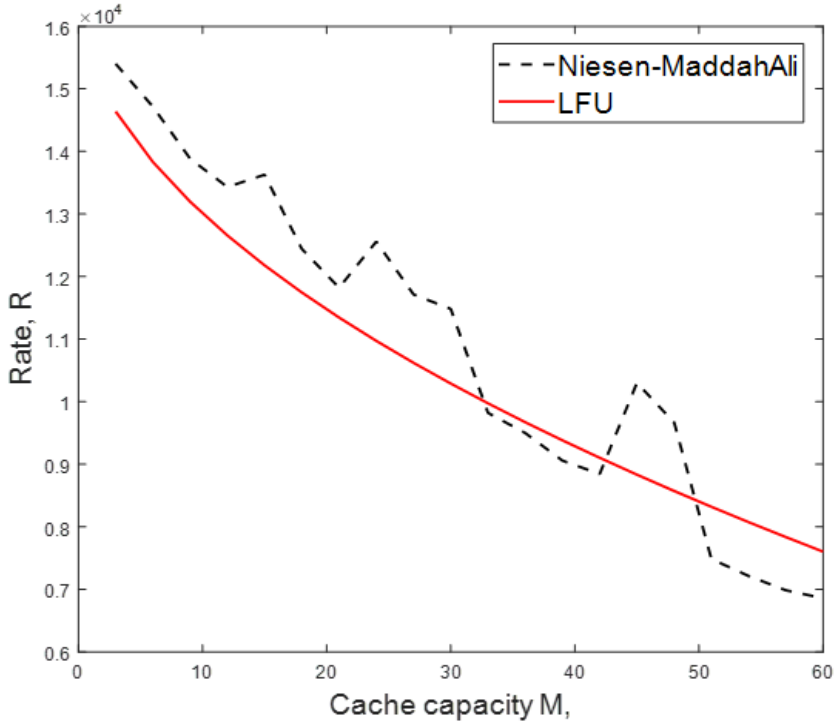

Figure 3.13 "Backhaul rate vs Cache Capacity M", performance comparison of HPF and DCC caching algorithms under scenario POPb (theta = 0.5)

Figure 3.13 shows the comparison of HPF vs DCC caching schemes under scenario POPb. Unlike the homogenous scenario, here both schemes still have similar performance. It can be viewed that both rate decrease as a function of M. despite the fact DCC curve has fluctuated as a function of M.

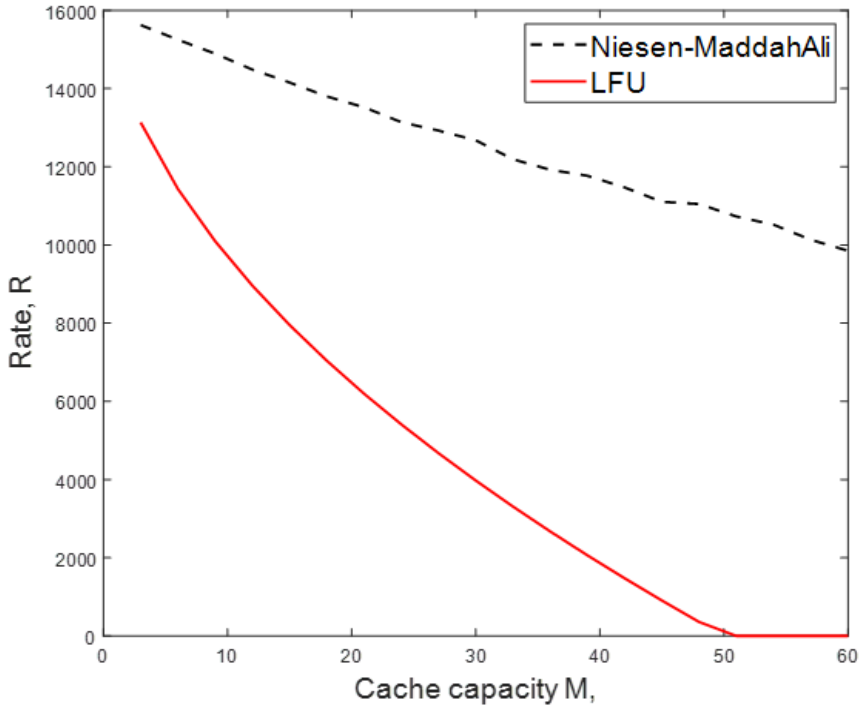

Figure 3.14 "Backhaul rate vs Cache Capacity M", performance comparison of HPF and DCC caching algorithms under scenario POPc (theta = 0.5)

Figure 3.14 shows the comparison of HPF vs DCC caching schemes under scenario POPc. In this figure, it can be noticed that both curves show the same performance shown in figures 3.9 and 3.12 where the popularity parameter=0.3. HPF scheme performs better than DCC.

4 - theta θ =1:

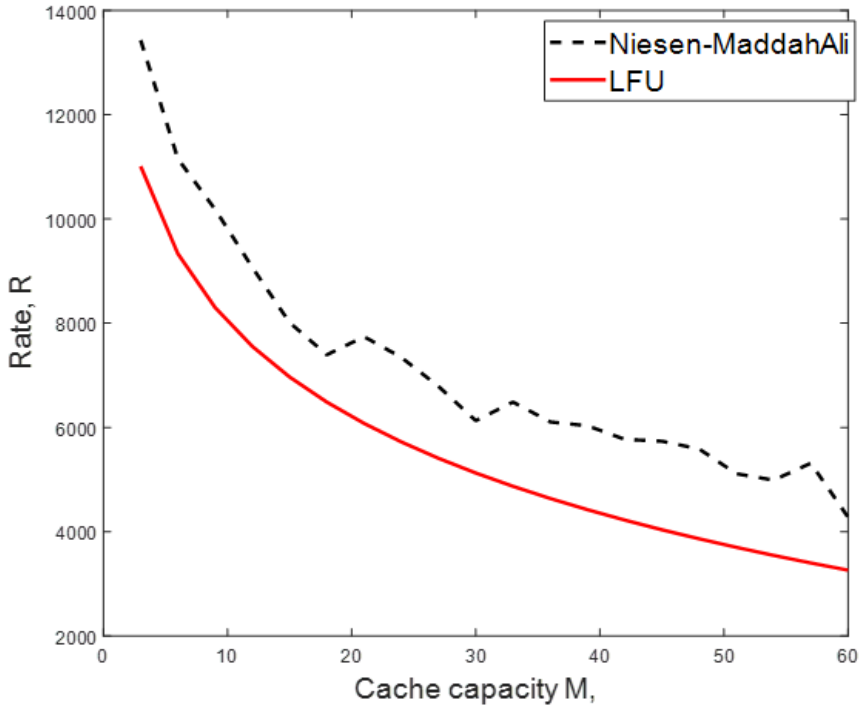

Figure 3.15 "Backhaul rate vs Cache Capacity M", performance comparison of HPF and DCC caching algorithms under scenario POPa (theta = 1)

Figure 3.15 shows the comparison of HPF vs DCC caching under scenario POPa. It shows that HPF more preferable and has a better performance compared to DCC scheme. While both schemes rates decrease proportionally to the cache size M.

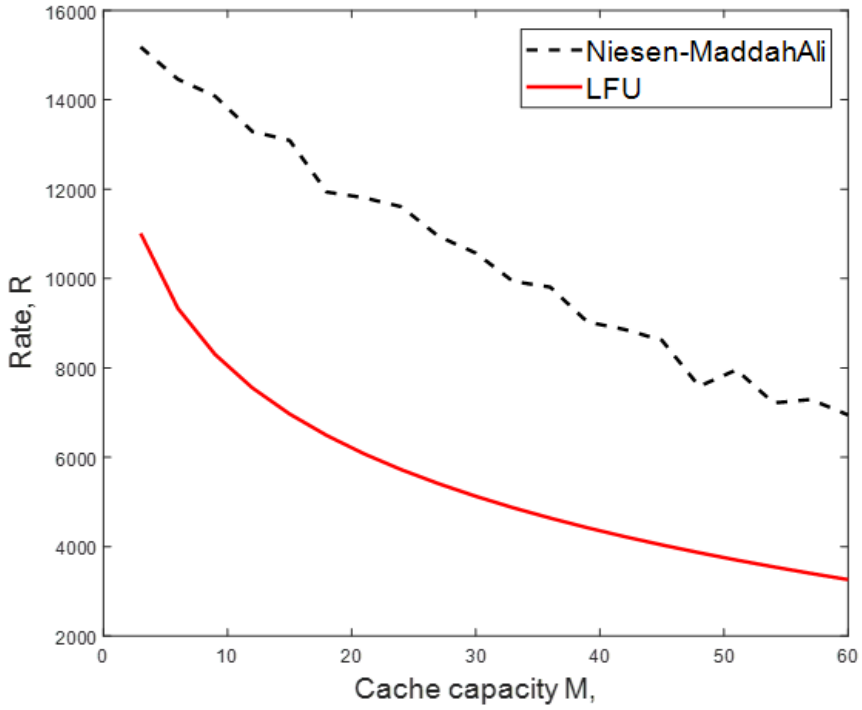

Figure 3.16 "Backhaul rate vs Cache Capacity M", performance comparison of HPF and DCC caching algorithms under scenario POPb (theta =1)

Figure 3.16 shows that HPF scheme become much better than DCC algorithm and it provides huge gain for the system compared to DCC scheme. Finally, in both schemes rates drop as we increase the cache size M. However, it can be noticed that DCC has small deviations compared to HPF curve.

Figure 3.17 shows the comparison of HPF vs DCC caching under scenario POPc for theta=1. In this figure, it can be noticed HPF curve started at 9000 bits for small cache size compared to HPF curve shown in figure 3.15 where the curve started at 13000 bits. Thus, the HPF rate decreased when skewness parameter increased from 0.5 to 1. In contrast, DCC scheme still has almost the same performance for the mentioned skewness parameter range

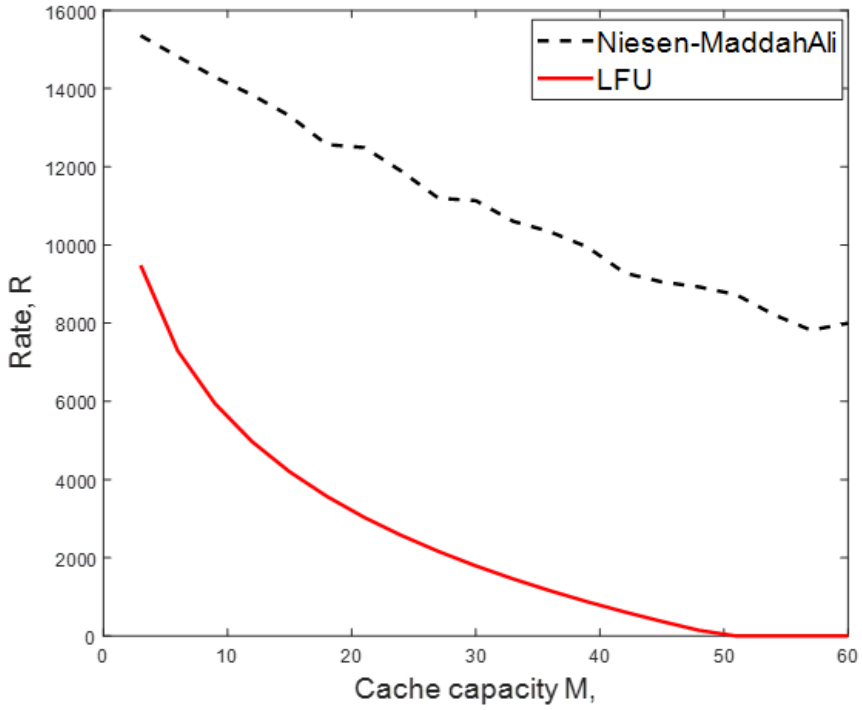

Figure 3.17 "Backhaul rate vs Cache Capacity M", performance comparison of HPF and DCC caching algorithms under scenario POPc (theta = 1)

5 - theta θ =2:

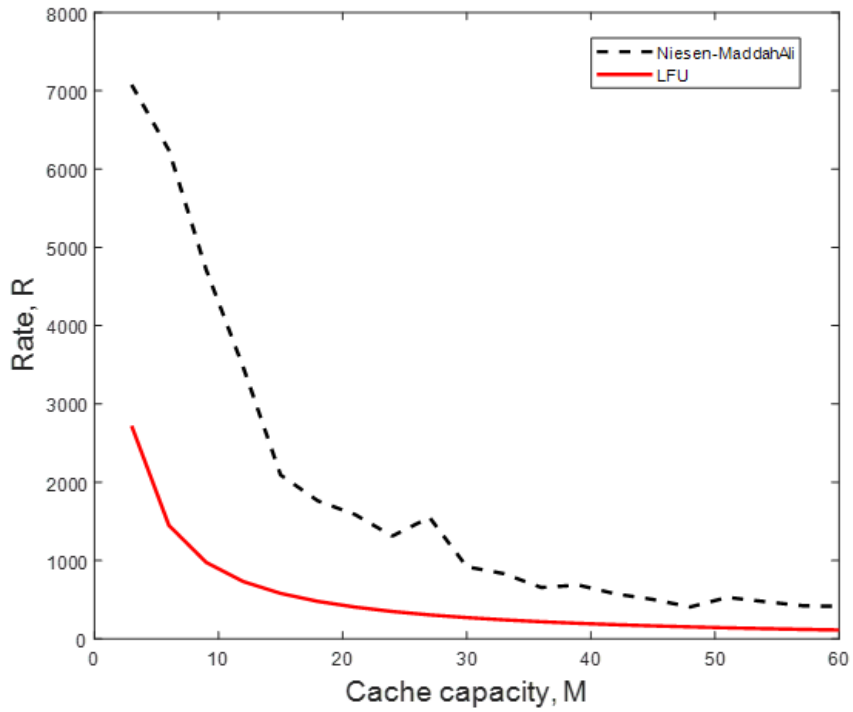

Figure 3.18 "Backhaul rate vs Cache Capacity M", performance comparison of HPF and DCC caching algorithms under scenario POPa (theta = 2)

Figure 3.18 shows the comparison of HPF vs DCC caching under scenario POPa. For theta = 2, it can be noticed both schemes rates have an approximately exponential decrease in the rate

as the cache size increases. In addition, HPF shows a very good performance with the rate started at 2800 bits which keep dropping as a function of M until it becomes very small compared to DCC rate which started at 7000 bits for small cache size M and, yet it has a lower gain for the rest of M values.

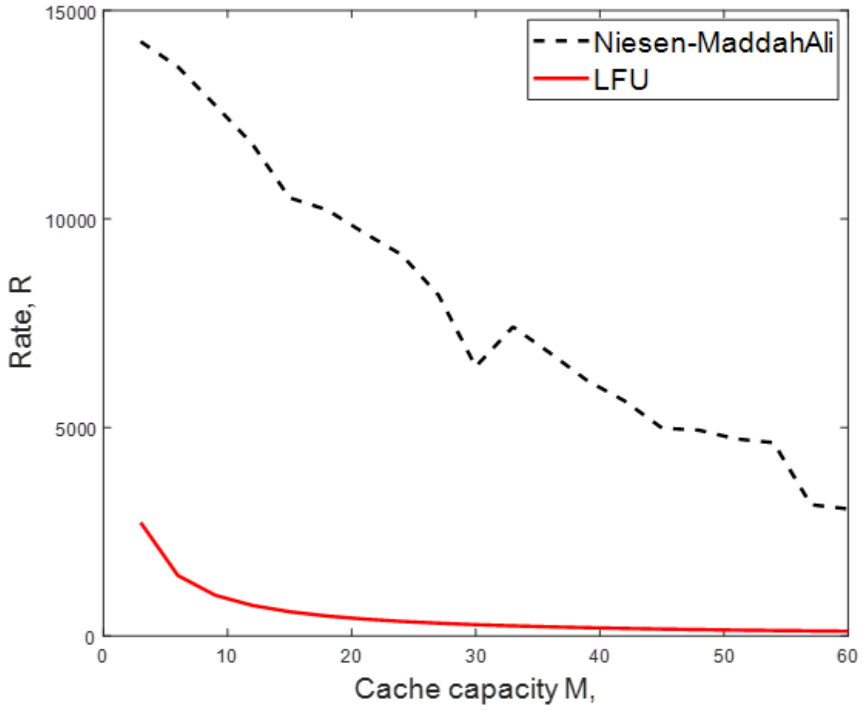

Figure 3.19 "Backhaul rate vs Cache Capacity M", performance comparison of HPF and DCC caching algorithms under scenario POPb (theta = 2)

Figure 3.19 shows the comparison of HPF vs DCC caching schemes under scenario POPb. Moreover, unlike the figure 3.19, it can be noticed the DCC rate drops slowly in approximately linear way. Moreover, the DCC rate started at 15000 bits for small cache size for heterogeneous scenario compared to 7000 for homogenous one. In contrast, HPF has a huge gain with the very small rate close to zero for large values of cache capacity.

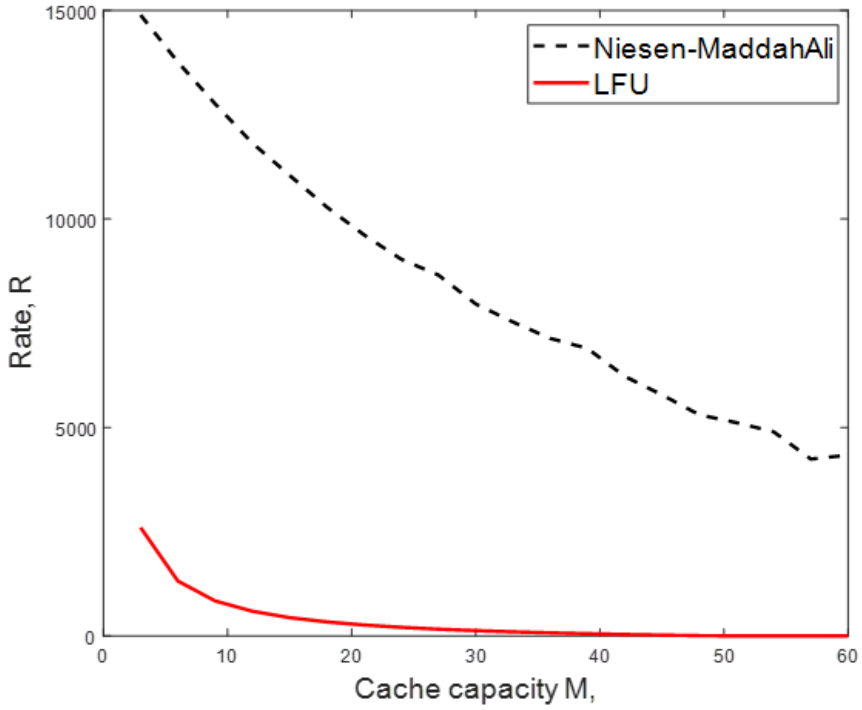

Figure 3.20 "Backhaul rate vs Cache Capacity M", performance comparison of HPF and DCC caching algorithms under scenario POPc (theta = 2)

Figure 3.20 shows the comparison of HPF vs DCC caching under scenario POPc. In this figure, it can be noticed HPF curve started around 2500 bits for small cache size showing similar performance to the heterogeneous scenario. In addition, DCC scheme has similar performance with some deviation to heterogeneous case. Moreover, HPF scheme requires a small rate that decrease to zero for cache size greater than 30. In contrast, the DCC rate drop linearly as function of M.

Chapter 4 Conclusions and Discussion

This last chapter concludes the work that has been done during this thesis as well as draws the attention about its outcomes and the potential avenues that ought to be explored in the future.

4.1 Conclusion

Caching at the network edge is a recently proposed technique for next mobile generation, 5G network, in order to reduce the traffic on the backhaul links by caching the content at or near the end users. The focus of this thesis has been on analyzing and comparing two dominant caching approaches, uncoded and coded caching, under different assumptions about the spatial locality of content demand. The comparison has been performed through MATLAB in two steps: the first step, compares Decentralized and Centralized coded caching with the Least Frequently Used (LFU) caching scheme under uniform content popularity distribution; Thus, all the files are equally popular at the small cell cache level. The control parameters for this step are the number of users K, the number of files N, and the cache size M. The main outcome of this step is:

- The centralized coded caching provides a huge gain compared to LFU and Decentralized coded caching, this gain is proportional to M and K, which confirm the theoretical results provided in [4].
- LFU and DCC schemes have approximately similar performance and the rates they achieve are inversely proportional to K.
- DCC, LFU, and CC have a slow response to a huge increase of catalog size N.

The second step consists in comparing the LFU caching scheme with the Decentralized coded caching under three scenarios of non-uniform file popularity distribution. The control parameters of this step are the skewness parameter, θ, and cache size, M. The main outcomes of this step are:

- There is a critical value of θ_{cr}, a turning point marking a change in how the two schemes compare with each other. The decentralized caching scheme provides a considerable gain over a small range of skewness parameter values below θ_{cr},. In contrast, the LFU scheme provides a very good performance that supports a wider range of θ values.
- The gain provided by LFU schemes increases proportionally with skewness parameter θ.
- Increasing the cache size M would add a considerable gain to both caching schemes.

Moreover, here we provide an additional result that had been noticed in both steps:

- There is a computational burden on the server and the users imposed by Decentralized and Centralized coded caching. This burden increases with K which increases the simulation run time. In contrast, LFU scheme doesn't require any computational complexity.

Implementing caching at the network edge involves a collaboration of three key stakeholders: the users, the mobile operator, and the content providers. Each stakeholder will be affected by the following:

The users of the mobile operator would benefit by having a better network performance. However, they would share their own resources, for example, the memory and processing for coded caching. In addition to the privacy issues resulted from intercepting their transmission by the mobile operator.

The mobile operator would get the most benefit of implementing wireless caching by minimizing the load on the backhaul links as well as increase their user satisfaction by providing high-speed data rates. Nevertheless, their considerable cost imposed by installing the infrastructure of the Big data processing servers. Moreover, providing new protocols that would maintain users' privacy while allowing the mobile operator to intercept their transmission.

The content providers who can also benefit from implementing the wireless caching system by providing their content near the end users. This leads to a reduction of content transmission costs. Moreover, they will avoid the costs of having large memory units. However, installing the caches at the network edge wouldn't be beneficial if they stick to legacy CDN techniques.

Our tests of aforementioned coded caching schemes are limited to 8 caches. This restriction is due to the computational complexity that consumes considerable time. However, in the practical implementation, the system could have many users and caches. Therefore, one approach mentioned in [4] to deal with it by using coded caching only among smaller subgroups of caches. Nevertheless, this decreases the computational load at the cost of higher rates over the shared link.

In conclusion, our simulation results show that LFU caching scheme provides a better performance than coded caching schemes for real-life scenarios which represented it in our simulation as non-uniform content popularity. In addition, coded caching schemes still need additional improvements regarding the supported number of users as well as the computational complexity imposed on the user and server side.

4.2 Future work

Coded caching at the network edge is still new, it is a hot topic of ongoing research for exploring and examining implementation feasibility. This project is among the first few that have generated a comparative study between coded and uncoded caching techniques under conditions of spatial locality for the demanded content. Some of the essential avenues to be explored in the future are:

1. Including machine learning and clustering techniques as a way to group cells of similar demand and keep K (the number of users) in the coded caching design small.
2. Performing more experiments on real data in order to track what is requested from each cell.
3. New Coded Caching designs that can cope with the challenge of K and group files/caches/users more intelligently to make CC realistically feasible.
4. Including online caching updates in the coded caching design. In other words, the current design of coded caching technique updates the cache contents only during the placement phase. However, in practical use, many caching systems update their caches during the delivery phase.

Appendix A: MATLAB Code

HPF code:

In the following code, we model a cache system of K users, N files and M cache size.

we distinguish between two cases, when the caches rank the files in similar or different way.

In what follows, we present the code of both cases:

a) the caches rank the files similarly:

1- The main function (trainsim.m)

```
% The code start here, the main function

clc; clear all;

% number of files
N = 100;

% number of caches
K = 10;

% cache size
M = 10;

% generate the popularity distribution of the files

counter=1; % for ploting the figures

for theta =0.2:0.1:1
   [fpop] = generate_localdistr(N, K, theta); %note different
  users will have the same popularity victor


% generate the global distribution
for n=1:N
    fpopg(n) = 0;
    for k=1:K
        fpopg(n) = fpopg(n) + fpop{k}(n);
    end
    product=1-fpop{1}(n);
    for j=2:K
    product=product*(1-fpop{j}(n));
    end
    fpopg(n) = 1-product;

end
```

```
fprintf(1,'Generated the local and global content
   distributions \n Press enter to continue\n');
%pause;

[hitratio placements] = PopCache(fpop,N,K,M);
fprintf(1,'Computed the hit ratio and placement under
   popularity-based caching \n');
fprintf(1,'The hitratios per cache are: \n');
hitratio_array{counter}=hitratio;
placements_array{counter}=placements;
counter=counter+1;
% what needs to be sent over the broadcast link is the portion
   of the
% requests that cannot be satisfied by the local caches
R_hpf = sum(1-hitratio);
fprintf(1,'The rate over the shared link is R_hpf=%f
   \n',R_hpf);
end
p=numSubplots(9); % this function to calculate the number of
   columns and rows in subplot
figure
title('Hitratio VS theta');

theta=0.2;
for i=1:9
    %%this loop to plot the sub graphs, we note that each
   graph is a point
    %%becasuse we have one value for theta and one value in
   all hitratio
    %%vector
    subplot(p(1),p(2),i)
        plot(theta,hitratio_array{i},'+')
        xlim([0 1])
        ylim([0 1])
        ylabel('Hit-ratio') % y-axis label

        xlabel('Theta')
        theta=theta+0.1;
end

% end of the main function
```

1-Generating local distribution of the file (generate_localdistr.m)

a) For heterogenous and homogenous scenarios:

```
function [fpop] = generate_localdistr(N, K, theta,N_S,O, type)
%  generate_localdistr
```

```
%
%  generates the local content popularity distributions
%  inputs :  N (num files), K (num caches), theta (the
parameter of local
%  Zipf distribution), type(homo or hetero)
%  output :  the KxN cell array fpop, with fpop{i}(j) being
the probability
%  that item j is requested from cache i

for i = 1:N
    f(i) = power(1/i,theta);
end
sumf = sum(f);
f = f/sumf;
fpop{1}= f;
t=1;
 if strcmp(type,'hetero_overlap')

        for i=1:K
           fpop{i}(1:N)=0.0000000000000001; % the files that
is not of interest of the users have a very small pop.
                                % I couldn't make the value
zero
                                % the minimum value I can use
here is 0.01
                                % otherwise I will get an error
        end
        f= f(randperm(N));
            fpop{1}(1:N_S)= f(1:N_S); % assign the first
subset to the first user
             t=t+N_S-1;
                t=t-O
end
% produce K-1 permutations of the popularity vectors
for j=2:K
    if strcmp(type,'hetero')
       fpop{j}(1:N) = f(randperm(N));
    elseif strcmp(type,'homo') % all users are identical
       fpop{j}(1:N) = fpop{1}(1:N);

end

end
```

b) For heterogeneous over subset of files with partial overlaps

```
function [fpop] = generate_localdistr2(N,K,theta,N1,O)
%  generate_localdistr2
%
%  generates the local content popularity distributions
```

```
%  inputs :  N (num files), K (num caches), theta (the
parameter of local
%  Zipf distribution
%  output :  the KxN cell array fpop, with fpop{i}(j) being
the probability
%  that item j is requested from cache i
for i = 1:N1
    f(i) = power(1/i,theta);
end
for i=N1+1:N
    f(i) = 0;
end
sumf = sum(f);
f = f/sumf;
fpop{1}= f;

for i=2:K
    range = [(i-1)*(N1-O)+1:(i-1)*(N1-O)+N1];
    fpop{i}(range) = f(1:N1);
    fpop{i}(setdiff(1:N,range))=0;
end
```

1- Populating cache memories and calculating hit-ratio (PopCache.m)

```
function [hitratio, placements] = PopCache(fpop,N,K,M);
% computes the aggregate popularity of the K most popular
  items that are
% stored in each of the K caches
% for a caching technique that stores the K most popular items
  (i.e., LFU)
% this is equal to the hit ratios that are expected to be seen
  in thgese
% caches

for k =1:K
    [sorted{k} ind] = sort(fpop{k},'descend');
    hitratio(k) = sum(sorted{k}(1:M));
    placements{k}= ind(1:M);
    miss(k) = 1-hitratio(k);
end
```

b) the caches rank the files differently:

in this case, we only modify the `generate_localdistr` function on order to randomly produce popularity victors

```
function [fpop] = generate_localdistr(N, K, theta)
```

```
%  generate_localdistr
%
%  generates the local content popularity distributions
%  inputs :  N (num files), K (num caches), theta (the
  parameter of local
%  Zipf distribution)
%  output :  the KxN cell array fpop, with fpop{i}(j) being
  the probability
%  that item j is requested from cache i

for i = 1:N
    f(i) = power(1/i,theta);
end
sumf = sum(f);
f = f/sumf;
fpop{1}= f;


% produce random k-1 popularity vectors.
for j=2:K
    rand_index=randperm(length(f)); %return random indcies
  from 0 to the  length of (f);
    rand_f=f( rand_index); %reshuffel the f files to produce
  random k  popularity victors
    fpop{j}(1:N) = rand_f(1:N);
    end
end
```

CC MATLAB Code:

```
function [bitsCC] =
NiesenMaddahAliCC(M,N,K,fsize,L,sortedfpopg,sortedfiles,placer
eps,reqreps,localfd)
% INPUTS
% reps  : number of rounds of file requests
% L     : number of groups
% fsize : file size in bits
% localf : local distributions of demand

% fragment files into L popularity groups
maxpop = sortedfpopg(1);
maxind = 1;
minind = 0;
divisor = power(2,1);
for grpid=1:L
    if L == 1
        F{grpid} = sortedfiles;
        Nl(1) = N;
    else
        [a lastgroup] = find( sortedfpopg < maxpop/divisor );
        if isempty(lastgroup)
            F{grpid} = sortedfiles(minind+1:end);
            Nl(grpid) = length(F{grpid});
        else
            minind = lastgroup(1)-1;
            F{grpid} = sortedfiles(maxind:minind);
            Nl(grpid) = length(F{grpid});
            %maxpop = sortedfpopg(lastgroup(1)-1)
            maxind = lastgroup(1);
            divisor = divisor*2;
        end
    end
end
fprintf(1,'Finished with the file partitioning \n Press enter
to continue\n');
Nl

% determine the memory assigned to each group
% baseline : uniform assignment at group level
for grpid = 1:L
    Ml(grpid) = M/L;
end

% Make the original placements of bits from each file
% returns 2^K possible subsets of users and the bits that are
common
% to each one of those per file
for count = 1:placereps
```

```
    [cached subsetsizes] = DCCplacementNMA(Ml,Nl,K,F,L,fsize);
    fprintf(1,'Determined the initial bit placements at user
caches \n');

    % now generate one or more file request vectors according
to
    % file popularity distribution
    for iter=1:reqreps
        for i=1:L
            requested{i} = '';
            Kl{i} = '';
        end
        for j = 1:K
            [sortedlocal sortedlocalind] =
sort(localfd{j},'descend');
            ecdf = cumsum(sortedlocal);
            ecdf
            draw = rand(1,1);
            draw
            inds = find(ecdf > draw)
            request(j) = sortedlocalind(inds(1));
            srchind = 1;
            while ~ismember(request(j),F{srchind}) && (
srchind <= L )
                srchind = srchind+1
            end
            Kl{srchind} = [Kl{srchind} {j}];
            requested{srchind} = [requested{srchind}
{request(j)}];
        end

        % At this point I know which files out of each group
have been requested
        % and how many users have requested files from each
group.
        % The two are not identical since more than one
user(s) may have
        % asked the same file
        fprintf(1,'Finished with the user requests and
partitioning into file groups -- Users per group: \n');
        for g=1:L
            Kl{g}
        end
     %   fprintf(1,'Files requested by users :\n');srchind

        for g=1:L
            requested{g}
        end

        % now compute the delivery rates
        for grpid = 1:L
```

```
            if ~isempty(requested{grpid})
                [sentbits_nonunique{grpid} sentbits(grpid)
subsets subsetbits] =
DCCdeliveryNMA(Kl{grpid},fsize,cell2mat(requested{grpid}),cach
ed);
            else
                sentbits(grpid) = 0;
            end
        end

        bitsCC((count-1)*reqreps+iter) = sum(sentbits);
        %fprintf(1,'\n\nTotal bits that have to be sent with
%d rounds of CC : %d \n',L,bitsCC((count-1)*reqreps+iter));
        %fprintf(1,'Press enter to continue\n');
        %pause;
    end
end
```

DCC MATLAB Code:

1- Placement Phase Code:

```
function [user_cache subset_size] =
DCCplacementNMA(Ml,Nl,K,filegrps,L,fsize)
% clear all; clc; close all;
% N : number of files
% K : number of users
% M : cache size
% F : file size
% requests : the file choices of users in a call of the function
% this is different than the script where files are generated within
the
% code out of a uniform distribution
%fidp = fopen('NMAplacement.txt','w');

N = sum(Nl); % total number of files

for l=1:L
    % set of bits to store for each file in group l
    subset_size(l) = min(fsize,fix(Ml(l)*fsize/Nl(l)));
  %  fprintf(1,'file bits stored per file in group %d =
%d\n',l,subset_size(l));
    for u=1:K
        for j=1:N
            if ismember(j,filegrps{l})

user_cache{u}(j,1:subset_size(l))=(randperm(fsize,subset_size(l)));
% each user can store a subset of bits from each file
            end
        end
    end
end
% the possible number of file parts is equal to the power set of the
user
% set --> cardinality : 2^|K|
file_fragments = power(2,K);
%fprintf(1,'Each file is effectively split into maximally %d
segments, each stored at a different subset of caches
\n',file_fragments);
```

2- Delivery Phase Code:

```
function [totalbits, totalbits_unique, subsets, subsetbits] =
DCCdeliveryNMA(Kg,fsize,requests,user_cache)
% K : user ids with file requests within the specific file
group
% fsize : file size
% requests : the file choices of users in a call of the
function
% subsets : the different subgroups of users that might emerge
(and corresponding file
% fragments)
```

```
% subsetbits : the common bits at the caches of each subset of
users per file
% "requests" number files according to the original indexing
in [1,N]
% subtract the min index -1 to scale them down to the range
[1, Nl]
fidp = fopen('NMAsubsets.txt','w');
%d = requests - min(requests)+1
fprintf(1,'The requested files are :\n');
d = requests
%indset = length(subsets);

% get all possible subsets : store them in subsets with indset
enumerating them
K = length(Kg);
indset = 0;
for s = K:-1:1
   % consider all possible sets of size s
   msets = combnk(Kg,s);
   for j=1:size(msets,1)
       indset = indset+1;
       subsets{indset} = msets(j,:);
   end
end
indset = indset+1;
subsets{indset} = ''; % the last subset is the empty set
fprintf('\n\n Number of subsets = %d\n',indset);
%subsets

% for each file requested compute all 2^K subsets, i.e., the
part of the
% file that is common at the caches of each of the 2^K
possible subsets of
% users
for fileind = 1:length(requests) % for each file
    indF = requests(fileind);
    totalsetbits{indF} = [];
    for j=1:indset-1 % for each of the 2^K subsets of users
        subset = cell2mat(subsets{j});
        subsetbits{indF}{j} = [user_cache{subset(1)}(indF,:)];
        %totalset{indF} = union(totalset{indF},
user_cache{subset(1)}(indF,:));
        for c = 2:length(subset)
           subsetbits{indF}{j} =
intersect(subsetbits{indF}{j},user_cache{subset(c)}(indF,:));
        end
        subsetbits{indF}{j} =
setdiff(subsetbits{indF}{j},totalsetbits{indF});
        totalsetbits{indF} = union(totalsetbits{indF},
subsetbits{indF}{j});
        if indF == 1
```

```
            fprintf(fidp,'subset j =%d    num bits =%d
\n',j,length(subsetbits{indF}{j}));
        end
    end % end j
    subsetbits{indF}{indset} =
setdiff([1:fsize],totalsetbits{indF}); % bits that are not
stored by any user
    fprintf(fidp,'\nFile = %d  bits not stored anywhere = %d
\n',indF,length(subsetbits{indF}{indset}));
end % end indF

% now try to compute how many bits are sent over the air for
each of the
% 2^K-1 transmissions
totalbits = 0;
for j=1:indset-1 % the last one is the empty subset
    subset = cell2mat(subsets{j}); % get the subset
    if
isempty(setxor(cell2mat(Kg),union(subset,cell2mat(Kg)))) %
identical subsets
        sentbits(j) = 0; % bits to be sent for a specific
multicast coded transmission
        for memb = 1:length(subset)
            reduced = setxor(subset(memb),subset);
            % in the following if-else I search for the index
k of the user
            % subset in the reduced var
            if isempty(reduced)
                k = indset; % the empty set
            else
                k = 1;
                % conditions for search success: a) the size
of reduced should be equal to the that of the subset
                % we are after and b) the common elements
should also be equal
                % in number with the size of reduced
                while ( length(subsets{k}) ~= length(reduced)
) || ( length(reduced) ~= sum(cell2mat(subsets{k}) == reduced)
)
                    k = k+1;
                end
            end
            % get the file requested by the user subset(memb)
            ind2filerequests = find(cell2mat(Kg) ==
subset(memb));
            %fprintf(1,'In delivery part: userid =%d fileid in
requested files= %d  subset ind =%d
\n',subset(memb),d(ind2filerequests),k);
            bits =  subsetbits{d(ind2filerequests)}{k}; % bits
to be sent from file user memb requested
```

```
            if length(bits) > sentbits(j) % padd the smaller
files in XOR
                sentbits(j) = length(bits);
            end
        end % here I have computed how many bits do I need to
send for TX no j
    totalbits = totalbits+sentbits(j);
    end
end
%fprintf(1,'Total bits that have to be sent WITHOUT accounting
for similar file choices by users: %d \n',totalbits);

% subtract bits that are common when two or more users choose
the same
% file, savings are in terms of the bit set A_0 (i.e. bits not
stored in
% any cache for a specific file)
uniqued = unique(d);
for n=1:length(uniqued)
    repeats(n) = length(find(d == uniqued(n)));
    totalbits_unique = totalbits - (repeats(n)-
1)*length(subsetbits{uniqued(n)}{indset});
end
if ~exist('totalbits_unique')
    totalbits_unique = totalbits;
end
%fprintf(1,'Total bits that have to be sent, accounting for
similar file choices by users: %d \n',totalbits_unique);

fclose(fidp);
```

Main code for running all Schemes:

```
tstart = tic;

fid =fopen('CCvsHPF.txt','w');
%theta_v = [0.1:0.05:2];
% it must be K*N1-(K-1)*O = N
N = 260; K = 8;  fsize = 2000; Mt = [3:3:60]

N_S = 50;
O = 20;

theta_v = 0.5;

% number of placement iterations
placereps = 10;
% number of request vectors that is input to the code for each
placement
reqreps = 10;

for th=1:length(Mt)
    M=Mt(th)
    theta = theta_v;
    % original file distributions - hetero means heterogeneous
    [fpop] = generate_localdistr2(N, K, theta, N_S, O);
     % generate the global distribution
    for n=1:N
        sump = 0;
        for k = 1:K
           sump = sump+ fpop{k}(n);
        end
        fpopg(n) = sump;
%       prod = 1;
%       for k = 1:K
%           prod = prod*(1-fpop{k}(n));
%       end
%       fpopg(n) = 1-prod;
    end
    % requires normalization
    sumd = sum(fpopg);
    fpopg = fpopg/sumd;
%    fprintf(1,'Generated the local and global content
distributions\n Press enter to continue\n');
%   pause;

    % partition the files into sets of similar popularity
    [sortedfpopg sortedfiles] = sort(fpopg,'descend');

    % compute the number of groups
    L(th) = ceil(log2(sortedfpopg(1)/sortedfpopg(N)))
```

```
    %[bitsCC bitsHPF Nl Kl] =
NonUniformDemandCCfun(M,N,K,Fsize,L,sortedfpopg,sortedfiles);
    [bitsCC] =
NiesenMaddahAliCC(M,N,K,fsize,L(th),sortedfpopg,sortedfiles,pl
acereps,reqreps,fpop);

    rateNMA(th) = mean(bitsCC);

    fprintf(1,'theta = %f  average number of bits sent over
the %d file request iterations with Niesen-MaddahAli algorithm
=%d\n',theta,placereps*reqreps,rateNMA(th));
   % fprintf(1,'\nPress enter to continue\n');
   % pause;

    bitsHPF(th) = 0;
    for k=1:K
        [sortedfpopl] = sort(fpop{k},'descend');
        bitsHPF(th) =
bitsHPF(th)+fsize*sum(sortedfpopl(M+1:N));
    end
   % fprintf(1,'\n For comparison, the bits I would send with
an uncoded caching scheme HPF is : %d\n',bitsHPF(th));
   % pause;
end

fclose(fid);
figure;
plot(Mt, rateNMA,'k--',Mt,bitsHPF,'r-','LineWidth',1.5);
xlabel('Cache capacity M,');
ylabel('Rate, R');
legend('Niesen-MaddahAli','LFU','Location','NorthWest');

toc(tstart)
```

REFERENCES

[1] Golrezaei, Negin, et al. "Femtocaching: Wireless video content delivery through distributed caching helpers." in Proc. IEEE *INFOCOM 2012*

[2] Paschos, Georgios, et al. "Wireless caching: Technical misconceptions and business barriers." *IEEE Communications Magazine* 54.8 (2016): 16-22.

[3] Andreev, Sergey, et al. "Exploring synergy between communications, caching, and computing in 5G-grade deployments." *IEEE Communications Magazine* 54.8 (2016): 60-69.

[4] Maddah-Ali, Mohammad Ali, and Urs Niesen. "Fundamental limits of caching." *IEEE Transactions on Information Theory* 60.5 (2014): 2856-2867.

[5] Fadlallah, Yasser, et al. "Coding for Caching in 5G Networks." *IEEE Communications Magazine* 55.2 (2017): 106-113.

[6] S. Traverso et al., “Temporal Locality in Today’s Content Caching: Why It Matters and How to Model It,” in Proc. ACM SIGCOMM, 2013

[7] M. Leconte et al., “Placing Dynamic Content in Caches with Small Population,” in Proc. IEEE INFOCOM, 2016

[8] D. Naylor et al., “Multi-Context TLS (mcTLS): Enabling Secure in-Network Functionality in TLS,” in Proc. ACM SIGCOMM, 2015.

[9] 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.

[10] M. A. Maddah-Ali, U. Niesen, "Decentralized coded caching attains order-optimal memory-rate tradeoff, IEEE/ACM Transactions on Networking (TON), vol. 23, no. 4, pp. 1029-1040, 2015

[11] U. Niesen, M. A. Maddah-Ali, "Coded caching with nonuniform demands," IEEE INFOCOM WKSHPS, pp. 221-226, 2014

[12] R. Pedarsani, M. A. Maddah-Ali, U. Niesen, “Online coded caching,” IEEE/ACM Transactions on Networking, vol. 24, no. 2, pp. 836-845, 2016

[13] Amos Ndegwa, what is Cache Hit Ratio. https://blog.stackpath.com/glossary/cache-hit-ratio/. (2/11/2017).