On the Computational Aspect of Coded Caching with Uncoded Prefetching

Sotirios K. Michos, *Student Member, IEEE,* Panagiotis D. Diamantoulakis, *Senior Member, IEEE,* Leonidas Georgiadis, *Senior Member, IEEE,* and George K. Karagiannidis, *Fellow, IEEE*

***Abstract*—Coded caching is the distribution of content across a communication system using techniques from coding theory in order to create multicasting opportunities among the users receiving the content. This enables a multiplicative improvement over the classic uncoded caching with respect to the transmission rates required in the delivery phase of the content. Since its introduction, coded caching has attracted significant research interest and several different schemes have been proposed over the last years. This work focuses on the fundamental case of coded caching with uncoded prefetching. In this case, the users' caches are filled with uncoded content during a prefetching phase in order to best serve each user's request during a subsequent delivery phase. This important case has recently received a complete information-theoretic characterization. However, reaching the information-theoretic optimality imposes a significant computational imbalance among the users. To mitigate this imbalance, we perform a complete computational analysis of the two major forms of coded caching with uncoded prefetching, namely *centralized* and *decentralized*. Furthermore, we propose a new information-theoretically optimal method for the delivery phase that achieves a significant computational improvement compared to the state of the art.**

***Index Terms*—centralized coded caching, decentralized coded caching, computational analysis, computational improvement, network coding.**

I. Introduction

Caching is the technique of duplicating content in distributed memories across a system with the goal of reducing the traffic load and the service times whenever this content is requested. It is naturally comprised of two phases, the placement phase where the content is placed around in the system caches and the delivery phase where content requests are served, taking into account the data already present in the cache of each user. Conventional caching has a long line of research [1]–[8], where the main goal is to either maximize the hit rate, which is the probability that a requested content is found at the cache memory, or to optimize the placement of contents in the caches based on various criteria, most important of which being their popularity [9], [10].

S. K. Michos, P. D. Diamantoulakis L. Georgiadis and G. K. Karagiannidis are with the Department of Electrical and Computer Engineering, Aristotle University of Thessaloniki, Greece (e-mail: sotmichos@auth.gr; padiaman@ieee.org; leonid@auth.gr; geokarag@auth.gr).

This research is co-financed by Greece and the European Union (European Social Fund- ESF) through the Operational Programme "Human Resources Development, Education and Lifelong Learning" in the context of the project "Strengthen ing Human Resources Research Potential via Doctorate Research" (MIS-5000432), implemented by the State Scholarships Foundation (IKY).

A. Motivation

In their seminal paper [11] Maddah-Ali and Niesen proposed the use of coding in the placement and delivery phases in order to create simultaneous multicasting opportunities among the users. This provided a multiplicative gain in the transmission rates over conventional caching. One of the most important characteristics of their approach is that it can offer this significant gain, even for scenarios where the content popularity is unknown or it is considered uniform. Due to its advantages, coded caching has attracted considerable research interest. More specifically, further research on this topic has been mainly focused on the investigation of its information-theoretic limits [12]–[17], as well as exploring the various forms it can take such as decentralized caching [18], online caching [19], hierarchical caching [20]–[22], D2D caching [23], caching with non-uniform demands [24]–[27], cache-aided interference channels [28]–[31] and others [32]–[44].

After a number of publications [11], [12], [28], [45], [46] that explored the optimal information theoretic rate-memory tradeoff in coded caching with uncoded prefetching, an exact characterization for the cases of centralized and decentralized caching was provided in [47] by Yu et al. Centralized caching is the fundamental paradigm around which all other coded caching schemes are developed, making any results regarding it being of principal importance. In this sense, [47] offers the potential of immediate improvement of all other relevant schemes of coded caching by extending the insights therein to them, like in [48], [49]. There has also been significant progress towards characterizing the exact rate-memory tradeoff for the case of coded prefetching [13]–[17], [34], [50], [51] with [52] setting the state of the art to within a factor of 2 with respect to the, as of yet unknown, optimal.

Along with these developments, research also turned towards investigating and mitigating the implementation challenges and limitations of coded caching. To this direction, two major issues have been identified. The first issue is the combinatorial explosion which happens in coded caching, where the number of subfiles, into which the files are broken, increase exponentially with respect to the number of users. This quickly makes the subfile size corresponding to any finite file size fall below the single bit level [53]. To this end, several finite-file packetization schemes have been proposed [27], [49], [54]–[62] that try to contain this explosion while keeping the multiplicative gains of the original methods.

The second important issue, which is also the focus of this work, is the increased computational cost of the information-theoretically optimal caching methods for a specific set of users called "non-leaders". By *computational cost* we mean the number of XOR operations performed during a particular method. We give the proper description of the concept in Definition 1 along with some important clarifications about it that one should take into consideration. The methods of [47] achieve information-theoretic optimality by utilizing the so-called commonality in user requests, which is the fact that many users may be requesting the same content in the delivery phase (while having different parts of it cached during the placement phase). Among these users, one is arbitrarily called leader and the rest are called non-leaders. The transmissions then target groups of users, mixing a subfile that each user needs, with the subfiles they already have but are also missing from the rest in the group. The qualitative difference between a leader and a non-leader is that the only transmissions that do take place are the ones implicating the leaders. The transmissions implicating only non-leaders, are completely omitted. On a first approach, this leaves a non-leader with a partial list of the subfiles they would require in order to compose their requested file. However, due to the fact that both the leader as well as their other non-leaders are requesting the same file, these missing subfiles are buried within the transmissions that have taken place, just not in an easily accessible way. So the non-leader has to go through an additional and elaborate computational process in order to reveal this information and be able to fully recover their requested file.

This introduces a computational imbalance between a leader and their non-leaders. The authors of [47] realized that the computational manipulations a non-leader needs to perform impose a significant extra burden on them in comparison to the leaders and ask for a more efficient decoding scheme. Looking as an example at the more general case of decentralized caching, one can easily realize the exponential nature of this computational imbalance by utilizing the results we present in this paper. In particular, using (34) we can see that the computational cost for a leader changes linearly with the amount of users in the system. However, observing (37) we can see that there is an additional term which is due to the computational derivation of the non-transmitted signals a non-leader has to perform. To see how this term scales, we can assume that all the K users, each having cache size of M files, request N_e distinct files among a library of N files, such that the ratio $r = K/N_e$ remains constant. Then, after some straightforward manipulations of this term we find that it scales as $\Theta\left[(\frac{M}{N}(1-\frac{M}{N})(r-1+\frac{M}{N}))^{N_e}\right]$ that corresponds to an exponentially increasing extra burden for each non-leader compared to their leader.

To further illustrate the effect of this computational imbalance in the presence of commonality among the requests, we can compare the computational cost of a leader in centralized caching, given in (19) with the (averaged) computational cost of a non-leader, given by (40), as in Fig. 1. The system's parameters are $K = 30$ users and $t = 5$, meaning the cumulative cache size among the users is five times the size of the library. As the parameter N_e ranges from 1 to 24, the (averaged) computational cost $\bar{C}_{nl}$ for a non-leader in order to recover all its subfiles quickly increases up to a peak value of almost 2.8 times the constant computational cost of a leader. This reflects the the situation where for high commonality among the requests (small values of N_e) the low number of leaders lead to fewer transmissions taking place. This in turn means that a non-leader has to recover more subfiles computationally out of these fewer transmissions. On the other hand, for decreased commonality (high values of N_e) all the users end up being leaders and the imbalance ceases to exist. However, in the high commonality regime, as K and t increase, the multiplication factor we observe can increase very rapidly, and leads to an overly large computational burden for a non-leader compared to a leader.

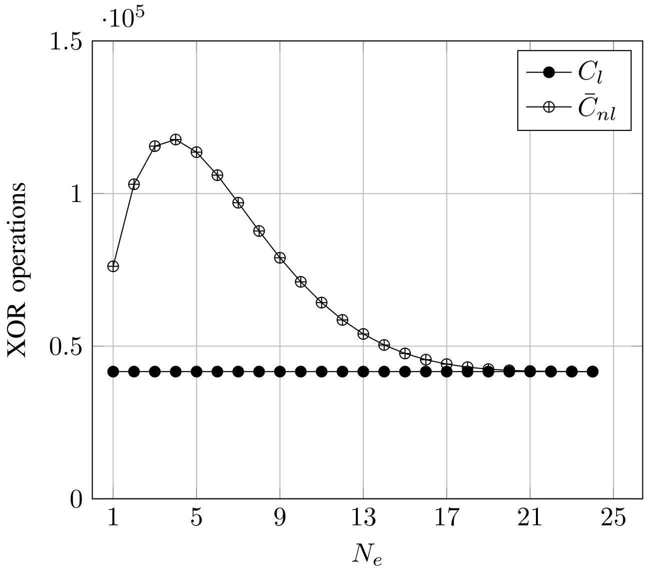


Fig. 1: The computational cost of a leader and the averaged computational cost of a non-leader for a coded caching system with $K = 30$ users and $t = 5$ for the various demands with $N_e \in 1, ..., 24$. We observe that as the commonality among the demands N_e changes, the computational cost for a non-leader can be multiple times the one for a leader.

In [47] a motivating example is provided which shows that the computations performed by a non-leader can potentially be reduced and its generalization is posed as an open problem. However, to the best of the author's knowledge, no such generalization has been provided. Furthermore, an exact characterization of the computational cost of the existing methods that would also enable their comparison to new approaches is also missing from the literature. To this end, the mitigation of the aforementioned computational imbalance as well as the formalization of a framework to characterize its computational aspects, are the main focus of this work.

B. Contribution

The contribution of this work is twofold and can be summarized to the following points:

i) We develop analytic expressions to precisely characterize the computational cost of coded caching with uncoded prefetching.

ii) We propose an improvement over this coded caching scheme with respect to the required computational resources.

In particular, we derive the computational cost of both centralized and decentralized caching with uncoded prefetching from a leader's and a non-leader's aspect as well as a system-wide point of view. With the computational manipulations being readily translatable to energy consumption, the same expressions provide a characterization of the schemes' energy demands.

Furthermore, we introduce a novel algorithm that directly improves the computational cost of the state of the art by utilizing a shortcut in the computations performed by a non-leader. We apply this improvement to both centralized and decentralized versions of coded caching and observe some significant computational improvements, especially in the second more realistic case. Due to the multi-parameter dependence of the exact analytic expressions for the computational cost, we show that there is a particular tractable averaging procedure that facilities a meaningful comparison between the methods.

C. Organization

The remainder of the paper is organized as follows. Section II gives a motivating example for our work and describes the system model of centralized and decentralized caching with uncoded prefetching. In Section III we perform a computational analysis of centralized and decentralized caching. In particular, we provide a complete characterization of the computational costs involved with the different parts of the system and observe that decentralized caching accepts a quite more compact characterization when compared to centralized caching. We proceed with the development of our proposed method in Section IV for both centralized and decentralized caching and present a comparison of its various aspects with the state of the art in Section V using two cases, one small user case and one large user case. The main paper closes with Section VI containing our conclusions and some comments on future work. The proofs of some more technical results are offered in the appendices.

II. System Model and Preliminaries

A. Motivating Example

In this section we present a motivating example with the purpose to illustrate the difference between the method provided in [47] and our work. As shown in Fig. 2, we consider a simple system with a library of $N = 2$ files, called A and B, and $K = 6$ users each with a cache size of $M = 1$ file.

The main goal in the prefetching phase is to fill up the users' cache memories in such a way that in the ensuing delivery phase we will be able to serve all the users concurrently for any potential demand vector. Following the prefetching method of [11] and [47] we break up each file into $\binom{6}{3} = 20$ files and index them according to the 3−subsets of the 6 users. In this sense, file A will break up into subfiles $A_{\{1,2,3\}}$, $A_{\{1,2,4\}}$, $A_{\{1,2,5\}}$, $A_{\{1,2,6\}}$, $A_{\{1,3,4\}}$, $A_{\{1,3,5\}}$, $A_{\{1,3,6\}}$, $A_{\{1,4,5\}}$, $A_{\{1,4,6\}}$, $A_{\{1,5,6\}}$, $A_{\{2,3,4\}}$, $A_{\{2,3,5\}}$,

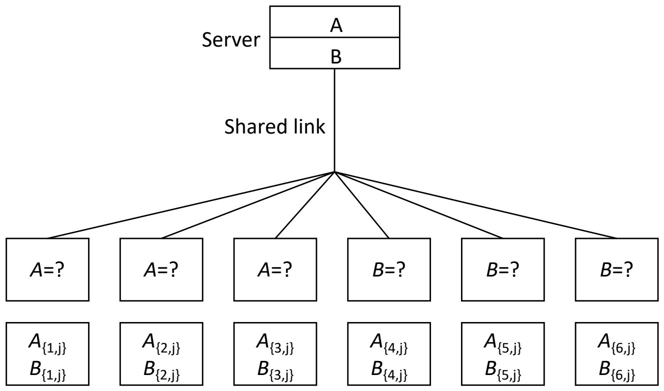

Server
A
B
Shared link
A=?
A=?
A=?
B=?
B=?
B=?
$A_{\{1,j\}}$ $B_{\{1,j\}}$
$A_{\{2,j\}}$ $B_{\{2,j\}}$
$A_{\{3,j\}}$ $B_{\{3,j\}}$
$A_{\{4,j\}}$ $B_{\{4,j\}}$
$A_{\{5,j\}}$ $B_{\{5,j\}}$
$A_{\{6,j\}}$ $B_{\{6,j\}}$

Fig. 2: A caching system with K=6 users, N=2 files, a local cache of M=1 file at each user and a demand vector where each file is requested by 3 users.

$A_{\{2,3,6\}}$, $A_{\{2,4,5\}}$, $A_{\{2,4,6\}}$, $A_{\{2,5,6\}}$, $A_{\{3,4,5\}}$, $A_{\{3,4,6\}}$, $A_{\{3,5,6\}}$ and $A_{\{4,5,6\}}$. The same happens with file B. Afterward, each subfile is sent to all the users contained in its index set. So, for example, user 1 will receive the subfiles $A_{\{1,2,3\}}$, $A_{\{1,2,4\}}$, $A_{\{1,2,5\}}$, $A_{\{1,2,6\}}$, $A_{\{1,3,4\}}$, $A_{\{1,3,5\}}$, $A_{\{1,3,6\}}$, $A_{\{1,4,5\}}$, $A_{\{1,4,6\}}$ and $A_{\{1,5,6\}}$ of file A as well as the corresponding subfiles of file B. Since each subfile is $1/20$ the size of each file and each user receives 20 subfiles, this process fills up their caches completely.

During the delivery phase, let us assume that the first three users request file A and the three latter ones file B. Focusing on an arbitrary 4−user subset, for example $\{1, 2, 3, 4\}$, we can observe that user 1 requires the subfile $A_{\{2,3,4\}}$ that is present in users 3, 4 and 5. Similarly, user 2 requires the subfile $A_{\{1,3,4\}}$ present on users $1, 3$ and 4, user 3 requires the subfile $A_{\{1,2,4\}}$ present in users $1, 2$ and 4 and user 4 requires the subfile $A_{\{1,2,3\}}$ present in users $1, 2$ and 3. Thus, if the base station transmits the signal $A_{\{2,3,4\}} \oplus A_{\{1,3,4\}} \oplus A_{\{1,2,4\}} \oplus A_{\{1,2,3\}}$ each of the users will be able to decode their required subfile. In this sense, transmitting the $\binom{6}{4} = 15$ corresponding signals $Y_{\mathcal{A}}$, one for each of the 4−user subsets $\mathcal{A} \subset \{1, 2, 3, 4, 5, 6\}$, we can ensure that all the users are able to recover their requested file. This was the approach presented in [11].

The authors of [47] realized that we do not have to transmit all the 15 signals. Instead, we can take one of the users requesting file A and one of the users requesting file B and send only the transmissions that involve these two users. We call these two users, let's say user 1 and 4, leaders. All the rest non-transmitted signals required by the non-leaders, can be derived from the transmitted ones using the fact that these users request the same file as their respective leader. In our example, that means that from the 15 transmissions, only 14 need to take place. These are

$$\begin{aligned}
Y_{\{1,2,3,4\}} &= A_{\{2,3,4\}} \oplus A_{\{1,3,4\}} \oplus A_{\{1,2,4\}} \oplus B_{\{1,2,3\}},\\
Y_{\{1,2,3,5\}} &= A_{\{2,3,5\}} \oplus A_{\{1,3,5\}} \oplus A_{\{1,2,5\}} \oplus B_{\{1,2,3\}},\\
Y_{\{1,2,3,6\}} &= A_{\{2,3,6\}} \oplus A_{\{1,3,6\}} \oplus A_{\{1,2,6\}} \oplus B_{\{1,2,3\}},\\
Y_{\{1,2,4,5\}} &= A_{\{2,5,6\}} \oplus A_{\{1,4,5\}} \oplus A_{\{1,2,5\}} \oplus B_{\{1,2,4\}},\\
Y_{\{1,2,4,6\}} &= A_{\{2,4,6\}} \oplus A_{\{1,4,6\}} \oplus B_{\{1,2,6\}} \oplus B_{\{1,2,4\}},\\
Y_{\{1,2,5,6\}} &= A_{\{2,5,6\}} \oplus A_{\{1,5,6\}} \oplus B_{\{1,2,6\}} \oplus B_{\{1,2,5\}},
\end{aligned}$$

$$\begin{aligned}
Y_{\{1,3,4,5\}} &= A_{\{3,4,5\}} \oplus A_{\{1,4,5\}} \oplus B_{\{1,3,5\}} \oplus B_{\{1,3,4\}},\\
Y_{\{1,3,4,6\}} &= A_{\{3,4,6\}} \oplus A_{\{1,4,6\}} \oplus B_{\{1,3,6\}} \oplus B_{\{1,3,4\}},\\
Y_{\{1,3,5,6\}} &= A_{\{3,5,6\}} \oplus A_{\{1,5,6\}} \oplus B_{\{1,3,6\}} \oplus B_{\{1,3,5\}},\\
Y_{\{1,4,5,6\}} &= A_{\{4,5,6\}} \oplus B_{\{1,5,6\}} \oplus B_{\{1,4,6\}} \oplus B_{\{1,4,5\}},\\
Y_{\{2,3,4,5\}} &= A_{\{3,4,5\}} \oplus A_{\{2,4,5\}} \oplus B_{\{2,3,5\}} \oplus B_{\{2,3,4\}},\\
Y_{\{2,3,4,6\}} &= A_{\{3,4,6\}} \oplus A_{\{2,4,6\}} \oplus B_{\{2,3,6\}} \oplus B_{\{2,3,4\}},\\
Y_{\{2,4,5,6\}} &= A_{\{4,5,6\}} \oplus B_{\{2,5,6\}} \oplus B_{\{2,4,6\}} \oplus B_{\{2,4,5\}},\\
Y_{\{3,4,5,6\}} &= A_{\{4,5,6\}} \oplus B_{\{3,5,6\}} \oplus B_{\{3,4,6\}} \oplus B_{\{3,4,5\}}.
\end{aligned}$$

We can observe now that, the non-transmitted signal $Y_{\{2,3,5,6\}}$ can be computed from the transmitted ones using the expression

$$\begin{aligned}
Y_{\{2,3,5,6\}} =& Y_{\{1,3,5,6\}} \oplus Y_{\{1,2,5,6\}} \oplus Y_{\{2,3,4,6\}} \oplus Y_{\{2,3,4,5\}} \oplus\\
& Y_{\{1,3,4,6\}} \oplus Y_{\{1,3,4,5\}} \oplus Y_{\{1,2,4,6\}} \oplus Y_{\{1,2,4,5\}}\\
=& A_{\{3,5,6\}} \oplus A_{\{2,5,6\}} \oplus B_{\{2,3,6\}} \oplus B_{\{2,3,5\}}.
\end{aligned} \tag{1}$$

After each of the users $2, 3, 5$ and 6 has derived this signal, they can proceed with decoding their requested subfile by XOR-ing with the subfiles they already possess. For example, user 2 can recover $A_{\{3,5,6\}}$ by doing

$$A_{\{3,5,6\}} = Y_{\{2,3,5,6\}} \oplus A_{\{2,5,6\}} \oplus B_{\{2,3,6\}} \oplus B_{\{2,3,5\}} \tag{2}$$

What we observe here is that each non-leader has to perform some additional computational steps, compared to their leader, in order to derive all the non-transmitted signals and then decode for their requested subfile. If they were a leader, they would only have to perform the decoding computation for each subfile instead. This introduces a computational imbalance between the leaders and the non-leaders that in some cases can become quite significant. The authors of [47] realized in their motivating example that the non-leaders could do better than this and they ask whether their special case observation can be formalized and generalized to the arbitrary system, mitigating this computational imbalance.

In this work, we provide an answer to this request by claiming that it is possible to achieve this improvement in the general case. This can be achieved if in derivation (1) a user replaces the signals that they participate in their index with the subfile corresponding to the rest of the index. For example, user 2 can directly recover their subfile $A_{\{3,5,6\}}$, by performing the calculation

$$\begin{aligned}
A_{\{3,5,6\}} =& Y_{\{1,3,5,6\}} \oplus Y_{\{1,3,4,6\}} \oplus Y_{\{1,3,4,5\}} \oplus A_{\{1,5,6\}} \oplus\\
& A_{\{3,4,6\}} \oplus A_{\{3,4,5\}} \oplus A_{\{1,4,6\}} \oplus A_{\{1,4,5\}}.
\end{aligned} \tag{3}$$

In essence, what we see is that user 2 is able to directly recover their requested subfile, by avoiding the computational steps of the previous final decoding phase. An important observation is that the subfiles $A_{\{1,5,6\}}$, $A_{\{3,4,6\}}$, $A_{\{3,4,5\}}$, $A_{\{1,4,6\}}$ and $A_{\{1,4,5\}}$ do not come for user's 2 cache but rather from the previously received signals $Y_{\{1,2,5,6\}}$, $Y_{\{2,3,4,6\}}$, $Y_{\{2,3,4,5\}}$, $Y_{\{1,2,4,6\}}$ and $Y_{\{1,2,4,5\}}$, respectively. Similarly, all other non-leaders can perform the same procedure in order to directly acquire their required subfiles.

B. Centralized Caching

The system under investigation is the centralized coded caching system presented in [47]. This system comprises of a server and K connected users through a shared, error-free channel. The server contains a library of N files $W_1, W_2, \ldots, W_N$ each of size F bits. Also, each user has an amount of cache memory equal to MF bits.

The system operates in two phases, a placement and a delivery phase. During the placement phase, the users have free access to the library in order to fill their caches, without performing any coding on the content. During the delivery phase, each user makes a demand for a specific file to the server. Then, the server being the only one having access to the library, must deliver the requested content as efficiently as possible by utilizing the shared nature of the channel.

For the delivery phase, also called the prefetching phase, the authors in [47] utilize a scheme called symmetric batch prefetching. In the same paper, it is proved that this prefetching scheme, along with the proposed delivery method therein, constitutes an information-theoretically optimal way of delivering the content. Thus, for the rest of this paper, the method presented in [47] will be called the *Information Theoretic Optimal Delivery Method* or *ITODM* for short. We should note that ITODM is a complete algorithmic procedure, with a full specification of the way prefecthing and decoding are performed, involving specific computational operations. The characterization of those operations is performed in the computational analysis presented in Section III.

The key parameter of this algorithm is the quantity t equal to the ratio of the total cache memory among the users over the size of the library:

$$t = \frac{MK}{N}. \tag{4}$$

When this parameter is an integer, $t \in \{0, 1, \ldots, K\}$, the symmetric batch prefetching considers all the sets comprised of t users:

$$\mathcal{A}_t = \{A_t \in 2^{[K]} : |A_t| = t\}. \tag{5}$$

Here, $[K] = \{1, 2, ..., K\}$ is the user set, with each user represented by its unique index, $2^{[K]}$ is the user powerset, that is, all the possible user subsets and $|\cdot|$ is the cardinality of a set. The case where t is not an integer is typically handled through memory sharing.

If we call the contents of $\mathcal{A}_t$ "t-subsets", is it easy to see that there are

$$|\mathcal{A}_t| = \binom{K}{t} \tag{6}$$

t-subsets, equal to the number of ways we can choose t users out of K.

Then, symmetric batch prefetching breaks up each file in $|\mathcal{A}_t|$ subfiles of size $F/\binom{K}{t}$ bit, with each subfile corresponding to a different t-subset, and sends each subfile to the users in that corresponding t-subset. For simplicity, it is assumed that F is divisible by $\binom{K}{t}$, otherwise, the end of each file is virtually expanded by appending a suitable number of zeros. By expressing the library as an $N \times F$ bit matrix, the matrix is broken into $|\mathcal{A}_t|$ columns of size $FN/\binom{K}{t}$ bit, each corresponding to at different t-subset $A_t \in \mathcal{A}_t$. Each column is then sent to all the users contained in its t-subset. Since there are $\binom{K-1}{t-1}$ subsets containing a user, each user receives $\binom{K-1}{t-1}/\binom{K}{t}FN = MF$ bits fully filling up their cache.

During the delivery phase, each user makes a file request $d_i \in [N]$, where $[N] = \{1, 2, \ldots, N\}$. These requests form the demand vector $\mathbf{d} = (d_1, d_2, \ldots d_K) \in [N]^K$, on which the delivery algorithm operates. The delivery algorithm is based on two key observations. The first is that during the placement phase, all t-subsets $A_t \in \mathcal{A}_t$ receive a particular column of the library bit matrix. That means that for any $(t+1)$-subset $A_{t+1} \in \mathcal{A}_{t+1}$, each user $u \in A_{t+1}$ is requesting a particular line (i.e. a particular subfile) from the column, which has been received from the rest of the users in $A_{t+1}\backslash\{u\}$ during the placement phase. We can name this subfile $W_{d_u, A_{t+1}\backslash\{u\}}$. So, all user subfile demands that correspond to a particular $(t+1)$-subset can be served at once by XOR-ing the corresponding subfiles and transmitting the result:

$$Y_{A_{t+1}} = \bigoplus_{u \in A_{t+1}} W_{d_u, A_{t+1}\backslash\{u\}}. \tag{7}$$

Then, each user in A_{t+1} can recover their requested subfile by just XOR-ing this transmission with the subfiles they already have stored in their cache:

$$W_{d_u, A_{t+1}\backslash\{u\}} = Y_{A_{t+1}} \bigoplus_{u' \in A_{t+1}\backslash\{u\}} W_{d_{u'}, A_{t+1}\backslash\{u'\}}. \tag{8}$$

The second observation is that when the number N_e of distinct files requested in $\mathbf{d}$ is less than the number K of users, not all transmissions $Y_{A_{t+1}}$ for all $(t+1)$-subsets $A_{t+1} \in \mathcal{A}_{t+1}$ need to take place. In particular, it is the realization that by arbitrarily selecting N_e users $\mathcal{U} \subset [K]$, each with a distinct request and calling them leaders, any transmission $Y_{A_{t+1}}$ corresponding to a $(t+1)$-subset A_{t+1} comprised solely of non-leaders is redundant. If we call $\mathcal{A}^{nl}_{t+1}$ the family of all $(t+1)$-subsets comprised solely of non-leaders, the above means that, in order to have some profit from the commonality among the user requests, there must be at least one $(t+1)$-subset in $\mathcal{A}^{nl}_{t+1}$. In other words, we must have:

$$K - N_e \geqslant t + 1. \tag{9}$$

The converse proof given in [47] shows that this condition is fundamental. Combining these two observations, one reaches the conclusion that among all $\binom{K}{t+1}$ $(t+1)$-subsets, only $\binom{K}{t+1} - \binom{K-N_e}{t+1}$ correspond to actual transmissions. Since each transmission is $F/\binom{K}{t}$ bits long, a rate of

$$R = \frac{\binom{K}{t+1} - \binom{K-N_e}{t+1}}{\binom{K}{t}} \tag{10}$$

is achievable. Notice that F is not included in (10), since the rate corresponds to transmitted bits per file bit. Also, we should note that although R is called rate in the literature, it is, in fact, the transmission load on the channel during the delivery phase. In other words, the channel should be able to support a transmission with the rate implied by R or above. In this sense, we are interested in devising delivery schemes that make this rate as small as possible. It has been proven [47] that (10) represents the best possible rate for a centralized coded caching system with uncoded prefetching. The term centralized means that during the placement phase, we know the K users that will be requesting a file.

In the delivery method presented in [47], each omitted transmission $Y_{A_{t+1}}$ corresponding to a non-leader $(t+1)$-subset $A_{t+1} \in \mathcal{A}^{nl}_{t+1}$ is computed from the transmitted signals. This computation involves two constructions. First, the set $\mathcal{B}$ formed by the union of the leader set $\mathcal{U}$ and the non-leaders in A_{t+1}:

$$\mathcal{B} = \mathcal{U} \cup A_{t+1}. \tag{11}$$

Second, the family $\mathcal{V}^F$ of $\mathcal{V}$-subsets of $\mathcal{B}$ is formed, where each $\mathcal{V}$-set contains N_e users of $\mathcal{B}$ with distinct demands.

$$\mathcal{V}^F = \left\{ \mathcal{V} \in 2^{\mathcal{B}} : \begin{array}{l} |\mathcal{V}| = N_e, \\ \forall u_1 \neq u_2 \in \mathcal{V} \quad d_{u_1} \neq d_{u_2} \end{array} \right\}, \tag{12}$$

where $2^{\mathcal{B}}$ is the powerset of $\mathcal{B}$. These $\mathcal{V}$-sets can be generated by starting with $\mathcal{U}$ and then replacing one or more leaders by one of their non-leaders (having the same request) in $\mathcal{B}$. Having these two constructions, the authors of [47] prove that the signal $Y_{A_{t+1}}$ can be computed as

$$Y_{A_{t+1}} = \bigoplus_{\mathcal{V} \in \mathcal{V}^F \backslash\{\mathcal{U}\}} Y_{\mathcal{B}\backslash\mathcal{V}}. \tag{13}$$

This computation is the main focus of this work. In particular, we propose an alternative expression of equal computational cost that instead of generating $Y_{A_{t+1}}$, it directly gives the file requested by each user in A_{t+1}.

C. Decentralized Caching

The decentralized caching, represents the more realistic scenario where the number of users that will be requesting a file during the delivery phase is not known during the placement phase. As it is shown in [47], assuming a system with a library of N files, each of size F bit, a random caching of MF/N bits from each of the N files by each user during the placement phase is enough to guarantee optimality during the delivery phase. We do not mention the total number of users since this does not matter during this phase.

Suppose that during the delivery phase only K users actually make a request $\mathbf{d} = (d_1, \ldots, d_K)$. We can denote the set of these active users as $[K] = \{1, 2, ..., K\}$, representing each one of them by a suitable index. Calling the library bits $\mathcal{L}$, if we examine the way they were transferred to these users during the random prefetching, we will see that they can be organized (partitioned) into the following classes

$$\mathcal{L}_j = \left\{ b \in \mathcal{L} : \begin{array}{l} b \text{ is cached by exactly} \\ j \text{ users among } [K] \end{array} \right\}, \tag{14}$$

for $j = 0, 1, \ldots, K$. This collection forms a partition since any bit in $\mathcal{L}$ can be cached either by no user or exactly 1 user or exactly 2 users, etc. up to exactly K users. Each of these classes can be further partitioned if we ask the question which are the particular j users caching a bit of $\mathcal{L}_j$. If $A_j \in \mathcal{A}_j$ is any j-subset of the K users, this leads to the following partitioning within each $\mathcal{L}_j$:

$$\mathcal{L}^j_{A_j} = \{ b \in \mathcal{L}_j : \text{bit } b \text{ is shared among all users in } A_j \}. \tag{15}$$

We can see now that for each $j \in \{0, \dots, K\}$, the situation is analogous to the case of centralized caching with a file library $\mathcal{L}_j$ being broken to $\binom{K}{j}$ parts, each corresponding to a distinct j-subset and being shared among the j users therein. That means that for any $(j+1)$-subset $A_{j+1} \in \mathcal{A}_{j+1}$ each user $u \in A_{j+1}$ will be requesting the bits of file W_{d_u} shared among the other $u' \in A_{j+1} \backslash \{u\}$ users.

So, for each such $(j+1)$-subset $A_{j+1} \in \mathcal{A}_{j+1}$, the server can transmit the signal

$$Y_{A_{j+1}} = \bigoplus_{u \in A_{j+1}} V_{d_u, A_{t+1} \backslash \{u\}} \tag{16}$$

to accommodate the users in A_{j+1}. The quantity $V_{d_u, A_{j+1} \backslash \{u\}}$ contains the bits of file W_{d_u} shared exactly by the j users in $A_{t+1} \backslash \{u\}$. A difference from centralized caching is that the size of each $V_{d_u, A_{j+1} \backslash \{u\}}$ in (16) will be different due to the prefetching being random and so a padding of the smaller terms with zeros can make the expression computable. Also, we should note that both the server and each user should be aware of the bit positions being cached by the other users in order for the signal in (16) to be computable and decodable. Thus, some kind of synchronization between the users and the server should take place. As in centralized caching, if the number of distinct demands $N_e \leqslant K - (j+1)$, the K users can be split to "leaders" and "non-leaders" and any of the $\binom{K-N_e}{j+1}$ signals $Y_{A_{j+1}}$ corresponding to the non-leader $(j+1)$-subsets $A_{j+1} \in A_{j+1}^{nl}$ can be omitted from transmission, being derivable from the ones transmitted.

As explained in [52] the number $|\mathcal{L}_{A_j}^j(W)|$ of bits being placed in $\mathcal{L}_{A_j}^j$ and the number of bits $|\mathcal{L}_j(W)|$ being placed in $\mathcal{L}_j$ will asymptotically almost surely (a.a.s) be:

$$\begin{aligned} |\mathcal{L}_j(W)| &= \binom{K}{j} \left(\frac{M}{N}\right)^j \left(1 - \frac{M}{N}\right)^{K-j} F + o(F), \\ |\mathcal{L}_{A_j}^j(W)| &= \left(\frac{M}{N}\right)^j \left(1 - \frac{M}{N}\right)^{K-j} F + o(F), \end{aligned} \tag{17}$$

which are the expressions presented in [47] with the first being $\binom{K}{j}$ times bigger than the second, as expected. So, we see that asymptotically, for any j, the delivery phase of decentralized caching can be implemented by the same techniques used in centralized caching. We will use these expressions in Appendix A-C to calculate the computational cost of decentralized caching.

Since the bits comprising $V_{d_u, A_{t+1} \backslash \{u\}}$ in (16) are the ones contained in $\mathcal{L}_{A_{j+1} \backslash \{u\}}^j(W_{d_u})$, each transmission $Y_{A_{j+1}}$ will be $|\mathcal{L}_{A_j}^j(W)|$ bits long (a.a.s.) as given by (17). Multiplying this with the number $\binom{K}{j+1} - \binom{K-N_e}{j+1}$ of actual transmissions, summing over all j and dividing by F gives a rate of

$$R = \frac{N-M}{N} \left(1 - \left(1 - \frac{M}{N}\right)^{N_e}\right). \tag{18}$$

So, we observe that, even though the prefetching is random and the delivery phase transmissions are of varying size, in the limit of large file sizes F the statistic behavior becomes much more tractable. This allows us to explicitly compute the transmission load per file bit. We will utilize this emerging behavior to compute the computational benefits of our proposed decoding method for decentralized caching.

III. Computational Analysis of ITODM

A. Centralized Caching

In this section, we provide a computational analysis of ITODM presented in [47]. We start by defining the term *compuational cost* that we are using throughout this paper.

Definition 1: We define as *computational cost* of a calculation, the number of XOR operations performed during this calculation.

We are interested in deriving the computational cost of a leader C_l, the computational cost for an non-leader in order to compute the non-transmitted subfiles from the transmitted ones C_{nl}^{nt} as well as the total computational cost for a non-leader C_{nl}. We use the symbols $|\cdot|_c$ and $|\cdot|_s$ to refer to the computational cost and the number of bits, respectively. So, for example, if A, B and C are three bit blocks of size $|A|_s = |B|_s = |C|_s = n$ bit, then XOR-ing these three blocks, will require the execution of $|A \oplus B \oplus C|_c = 2n$ instructions (or XORs at the bit level).

We would like to clarify here that by *computational cost*, we do not mean the computational complexity of the corresponding task, in the sense of the theoretically optimal algorithm to perform that task, neither we mean the complexity class of that tasks. We merely mean, the number of XOR operations a computer would perform in order to complete the calculation by following the steps laid out by the corresponding algorithm, which is typically higher than the theoretical minimum. So, for example, when studying the XOR operation among m bit blocks of size n, $A_1 \oplus ... \oplus A_k$, there may be the case that there are some symmetries or common features among them that may allow for the operation's result to be completed in fewer that $(m-1)n$ XORs. However, if such provision is not stated in the algorithm or recognized somehow during its execution, as typically is the case for this work and the broader coded caching literature, we consider the computational cost to be $(m-1)n$.

We would further like to explain that both in this section as well as the rest of the paper, we are interested in the computational analysis of the decoding algorithms, as presented in the literature. In particular, we do not take into account any additional implementation details like the cost for reading from system inputs or writing to outputs. We also do not take into account the process of merging the individual subfiles into a single file once their decoding phase is completed as this falls outside the scope of the decoding algorithm itself. Any such merging process should be able to complete within F steps. Should one wishes to account for such a process, it shouldn't alter our results by more than an additive term of F.

Theorem 1: In a centralized coded caching system with symmetric batch prefetching, the total computational cost for a leader under ITODM is given by

$$C_l = \binom{K-1}{t} \frac{Ft}{\binom{K}{t}}, \tag{19}$$

or equivalently

$$C_l = \left(1 - \frac{M}{N}\right)\frac{MKF}{N}. \tag{20}$$

Proof: Under ITODM, all transmissions relevant to a leader $u \in \mathcal{U}$ take place. The number of those transmissions equals the number of $(t+1)$-subsets $A_{t+1} \in \mathcal{A}_{t+1}$ containing the user u. But this is the number of t-subsets $A_t \in \mathcal{A}_t$ that do not contain the user u. So there will be

$$\begin{aligned}|\{A_{t+1} \in \mathcal{A}_{t+1} : u \in A_{t+1}\}| &= |\{A_t \in \mathcal{A}_t : u \notin A_t\}| \\ &= \binom{K-1}{t}\end{aligned} \tag{21}$$

transmissions relative to leader u.

Let $Y_{A_{t+1}}$ be one such transmission. In order for u to recover the subfile $W_{d_u, A_{t+1}\backslash\{u\}}$, $Y_{A_{t+1}}$ must be XORed with all the subfiles $W_{d_{u'}, A_{t+1}\backslash\{u'\}}$ requested by the other users $u' \in A_{t+1}\backslash\{u\}$ that are already present in u's cache, as illustrated in (8). There are $|A_{t+1}\backslash\{u\}| = t$ such subfiles so there will be t block-XORs with each block having a size of $|W_{d_u, A_{t+1}\backslash\{u\}}| = F/\binom{K}{t}$ bit. So the computational cost of decoding the subfile $W_{d_u, A_{t+1}\backslash\{u\}}$ will be

$$\begin{aligned}C(W_{d_u, A_{t+1}\backslash\{u\}}) &= \left| Y_{A_{t+1}} \bigoplus_{u' \in A_{t+1}\backslash\{u\}} W_{d_{u'}, A_{t+1}\backslash\{u'\}} \right|_c \\ &= \frac{Ft}{\binom{K}{t}}.\end{aligned} \tag{22}$$

Since there are $\binom{K-1}{t}$ such subfiles that need decoding, the total computational cost for a leader will be expressed by (19) or, substituting the binomial coefficients, by (20). ■

Finding an expression for the total computational cost of a non-leader $u^{nl} \in [K]\backslash\{\mathcal{U}\}$ is not so straightforward since for different non-leader subsets $A_{t+1} \in \mathcal{A}_{t+1}^{nl}$, the computation of $Y_{A_{t+1}}$ according to (13) has different computational costs. Nevertheless, due to the importance of such a quantity, we make an attempt to derive an expression for it starting from the cost of calculating the subfile $W_{d_{u^{nl}}, A_{t+1}\backslash\{u^{nl}\}}$ that is shared among the non-leaders in $A_{t+1}\backslash\{u^{nl}\}$.

In order to give an exact expression for this computational cost, we will use some additional terminology introduced in [47]. If A_{t+1} is a non-leader set, then the corresponding $\mathcal{B} = \mathcal{U} \cup A_{t+1}$ is partitioned into sets of users having the same demand. We will call these sets the "tail" of the corresponding leader.

Definition 2: For any non-leader set A_{t+1} and any leader $u \in \mathcal{B} = \mathcal{U} \cup A_{t+1}$ we define the *tail* of this leader as

$$\mathcal{B}_u = \{x \in \mathcal{B} : d_u = d_x\}. \tag{23}$$

Using this definition, it is easy to see that each $\mathcal{V}$-set in (12) is, in fact, a selection of users among the different tails, taking one from each tail. Thus, we have that

$$\mathcal{V}^F \simeq \mathcal{B}_{u_1} \times \mathcal{B}_{u_2} \times \cdots \times \mathcal{B}_{u_{N_e}}. \tag{24}$$

The symbol "$\simeq$" is used to show that the two sets are not actually equal, but can be put in an one-to-one correspondence. As a matter of fact, if we change the tuple structure to that of a subset (by dropping the ordering), then the two become equal. We can now state the following theorem.

Theorem 2: In a centralized coded caching system with symmetric batch prefetching, let A_{t+1} be a set comprised of non-leaders. The computational cost for a non-leader $u^{nl} \in [K]\backslash\{\mathcal{U}\}$ under ITODM for decoding a specific subfile $W_{d_{u^{nl}}, A_{t+1}\backslash\{u^{nl}\}}$ shared among the other non-leaders $u' \in A_{t+1\backslash\{u\}}$ is

$$C_{nl}\left(W_{d_{u^{nl}}, A_{t+1}\backslash\{u^{nl}\}}\right) = \left(|\mathcal{V}^F| - 2\right)\frac{F}{\binom{K}{t}} + \frac{Ft}{\binom{K}{t}}, \tag{25}$$

where $|\mathcal{V}^F| = \prod_{u \in \mathcal{U}} |\mathcal{B}_u|$.

Proof: According to ITODM, the computation of $W_{d_{u^{nl}}, A_{t+1}\backslash\{u^{nl}\}}$ is done in two parts. The first part is computing the signal $Y_{A_{t+1}}$ from the actual transmissions using (13). This computation involves $|\mathcal{V}^F\backslash\{U\}| = |\mathcal{V}^F| - 1$ signals and thus needs $|\mathcal{V}^F| - 2$ block-XORs. Each block has size $|Y_{A_{t+1}}|_s = F/\binom{K}{t}$ bits which leads to the first summand of (25).

The second step has to do with decoding the actual subfile $W_{d_{u^{nl}}, A_{t+1}\backslash\{u^{nl}\}}$ from the computed signal $Y_{A_{t+1}}$. This step is the same as the corresponding one for a leader, so its computational cost is given directly from (22), leading to the second summand of (25). The expression for $|\mathcal{V}^F|$ comes directly from (24). ■

We can now proceed with our attempt for a total characterization of the computational cost of a non-leader for a particular demand vector. For this, we will use the following definitions.

Definition 3: Let $\mathbf{d} \in [N]^K$ a demand vector with N_e distinct file requests and $\mathcal{U}$ the leader set. We call the set Q_{u_i} of the non-leaders requesting the same file as leader $u_i \in \mathcal{U}$ the *pure tail* of this leader.

$$Q_{u_i} = \{u \in [K]\backslash\mathcal{U} : l(u) = u_i\}, \tag{26}$$

where $l(u)$ signifies the leader of u.

The difference between a tail B_{u_i} and a pure tail Q_{u_i} is that the former contains the leader u_i while the latter contains just the non-leaders. In other words $B_{u_i} = \{u_i\} \cup Q_{u_i}$. For the rest of this section, without loss of generality (w.l.o.g.) we assume that the non-leader we are interested in belongs in Q_{u_1}, or equivalently that $l(u^{nl}) = u_1$. This will simplify our expressions while still covering the general case with a simple renaming of the users. It also means that while the cardinality of the other pure tails can be between 0 and $K - N_e - 1$, the cardinality of Q_{u_1} is between 1 and $K - N_e$.

Definition 4: For demand vector $\mathbf{d} \in [N]^K$ and leader set $\mathcal{U}$, we call $\mathcal{W}(u^{nl})$ the set of all non-leader $(t+1)$-sets that contain user u^{nl}.

$$\mathcal{W}(u^{nl}) = \{A_{t+1} \in \mathcal{A}_{t+1}^{nl} : u^{nl} \in A_{t+1}\}. \tag{27}$$

It is easy to see that $|\mathcal{W}(u^{nl})| = \binom{K-N_e-1}{t}$. Since user u^{nl} is always selected by definition, when forming a set $A_{t+1} \in \mathcal{W}(u^{nl})$ we get to choose only t among $K - N_e - 1$ non-leaders. We can understand that after transmission has taken place, the non-leader u^{nl} is missing exactly the subfiles that correspond to the sets in $\mathcal{W}(u^{nl})$.

Definition 5: Let $A_{t+1} \in \mathcal{W}(u^{nl})$. We call the vector $(|\mathcal{B}_{u_1}|, \dots, |\mathcal{B}_{u_{N_e}}|)$ the *profile* of A_{t+1}.

Since $l(u^{nl}) = 1$, we have $|\mathcal{B}_{u_1}| \in \{2, \dots, |Q_{u_1}| + 1\}$ while for the rest we have $|\mathcal{B}_{u_i}| \in \{1, \dots, |Q_{u_i}| + 1\}$.

All subfiles $W_{d_{u^{nl}}, A_{t+1}\setminus\{u^{nl}\}}$ coming from sets A_{t+1} that have the same profile have the same computational costs given by (25). The following theorem gives the number of non-leader sets $A_{t+1} \in \mathcal{W}(u^{nl})$ that have a particular profile.

Theorem 3: Let

$$(k_1, \dots, k_{N_e}) \in \{0, \dots, |Q_{u_1}| - 1\} \bigtimes_{i=2}^{N_e} \{0, \dots, |Q_{u_i}|\}$$

such that $k_1 + \cdots + k_{N_e} = t$. The number of $A_{t+1} \in \mathcal{W}(u^{nl})$ having profile $(k_1 + 2, k_2 + 1, \dots, k_{N_e} + 1)$ is

$$\binom{|Q_{u_1}| - 1}{k_1}\binom{|Q_{u_2}|}{k_2}\cdots\binom{|Q_{u_{N_e}}|}{k_{N_e}}. \tag{28}$$

Proof: For $i \neq 1$, since the pure tail of leader u_i has $|Q_{u_i}|$ elements, there are $\binom{|Q_{u_i}|}{k_i}$ ways to choose k_i of them. For $i = 1$, user u^{nl} is always chosen by definition. That leaves the rest $|Q_{u_1}| - 1$ non-leaders among which k_1 are chosen which can be done in $\binom{|Q_{u_1}|-1}{k_1}$ different ways. Thus, the total number of non-leader sets $A_{t+1} \in \mathcal{W}(u^{nl})$ that we can form with the particular profile is given by the product of the above binomial coefficients. ■

Now we are in a position to calculate the total computational cost for the non-leader u^{nl} needed for deriving the non-transmitted subfiles from the transmitted ones by summing (25) over all $A_{t+1} \in \mathcal{W}(u^{nl})$, in effect proving the following theorem.

Theorem 4: Suppose a centralized caching scheme with symmetric batch prefetching and a demand vector $\mathbf{d}$ with N_e leaders whose pure tail sizes are $|Q_{u_i}|$, $i \in \{1, \dots, N_e\}$. Under ITODM, the total computational cost of a non-leader u^{nl}, whose (w.l.o.g.) leader is u_1, for computing the non-transmitted subfiles from the transmitted ones is

$$C_{nl}^{nt} = S\frac{F}{\binom{K}{t}} + \binom{K - N_e - 1}{t}\frac{F(t-2)}{\binom{K}{t}}, \tag{29}$$

where

$$S = \sum_{\substack{(k_1,\dots,k_{N_e})\in\mathcal{R} \\ k_1+\cdots+k_{N_e}=t}} \binom{|Q_{u_1}| - 1}{k_1}\binom{|Q_{u_2}|}{k_2}\cdots\binom{|Q_{u_{N_e}}|}{k_{N_e}} \times (k_1 + 2)(k_2 + 1)\dots(k_{N_e} + 1), \tag{30}$$

or equivalently

$$S = \sum_{l=0}^{\min(t,N_e)} \binom{K - N_e - 1 - l}{t - l} \times \Bigg(\sum_{1<i_2<\cdots<i_l\leq N_e} (|Q_{u_1}| - 1)\,|Q_{u_{i_2}}|\dots|Q_{u_{i_l}}| + 2\sum_{1<i_1<\cdots<i_l\leq N_e} |Q_{u_{i_1}}|\dots|Q_{u_{i_l}}| \Bigg), \tag{31}$$

and

$$\mathcal{R} = \{0, \dots, |Q_{u_1}| - 1\} \bigtimes_{i=2}^{N_e} \{0, \dots, |Q_{u_i}|\}. \tag{32}$$

Furthermore, the total computational cost for this non-leader in order to fully recover their requested subfiles is

$$C_{nl} = \binom{K-1}{t}\frac{Ft}{\binom{K}{t}} + \left[S - 2\binom{K - N_e - 1}{t}\right]\frac{F}{\binom{K}{t}}. \tag{33}$$

As we expected, the expression for this computational cost is quite complicated due to the many factors affecting the end result. These are K, t, N_e and the sizes of all pure tails $|Q_{u_i}|$ for $u_i \in \mathcal{U}$, with N_e and the pure tail sizes ultimately depending on the demand vector $\mathbf{d}$. The main difficulty lies in calculating the factor S appearing in the above theorem in which the effect of the pure tail sizes $|Q_{u_i}|$ is taken into account. A further study of this term, simplifying (30) to (31) is performed in Appendix A-A.

Finally, the expression (33) for the total computational cost of a non-leader can be easily derived from (29) but taking into account the fact that the cost for decoding each transmitted subfile is given by (22) as well as the fact that the number of the total transmitted subfiles requested by u^{nl} are $\binom{K-1}{t} - \binom{K-N_e-1}{t}$.

B. Decentralized Caching

For simplicity, we will assume that the choice of leaders and non-leaders is the same among all centralized instances of decentralized caching. In other words, the users selected as leaders remain constant throughout the whole transmission process. This is an assumption that places a big computational burden on the users chosen as non-leaders. Alternate scenarios that seek to distribute the extra computational burden of being a non-leader among the users may offer a more balanced computational load between the users. However, this assumption is the simplest one can make regarding this issue and provides a solid starting point for such further analyses, which makes it quite important to study.

Extending the previous section analysis to decentralized caching, the exact computational cost for this scheme is given in the two following theorems, that correspond to theorems (1) and (4) of the previous section.

Theorem 5: In a decentralized coded caching system with symmetric batch prefetching, the total computational cost for a leader under ITODM is

$$C_l^{dec} = (K-1)\left(1 - \frac{M}{N}\right)\frac{MF}{N} + o(F). \tag{34}$$

Proof: Since decentralized caching is comprised of multiple instances of centralized caching, we can sum up the computational cost of a leader over all these instances. The computational cost of a single instance of centralized caching is given by (20) after we substitute F with the equivalent file size given by (17).

So the total computational cost for a leader will be

$$\begin{aligned} C_l^{dec} &= \sum_{j=0}^{K-1} \binom{K-1}{j} \frac{|\mathcal{L}_j(W)|j}{\binom{K}{j}} \\ &= \sum_{j=0}^{K-1} \binom{K-1}{j} \left(\frac{M}{N}\right)^j \left(1-\frac{M}{N}\right)^{K-j} jF. \end{aligned} \tag{35}$$

Taking the term $1-\frac{M}{N}$ out of the sum and using the formula [63] for the expected value the binomial distribution $\mathcal{B}(K-1, M/N)$ we reach (34) after plugging in the term $o(F)$ that we omitted for simplicity. ■

Theorem 6: Suppose a decentralized caching scheme and a demand vector $\mathbf{d}$ with N_e leaders whose pure tail sizes are $|Q_{u_i}|$, $i \in \{1, \dots, N_e\}$. Under ITODM, the total computational cost of a non-leader u^{nl}, whose (w.l.o.g.) leader is u_1, for deriving the non-transmitted subfiles from the transmitted ones is

$$\begin{aligned} C_{nl}^{dec,nt} &= \left(1-\frac{M}{N}\right)^{N_e+1} F \\ &\times \left(\left(\frac{M}{N}(|Q_{u_1}|-1)+2\right) \prod_{i=2}^{N_e} \left(\frac{M}{N}|Q_{u_i}|+1\right) \right. \\ &\left. + \frac{M}{N}(K-N_e-1)-2 \right) + o(F). \end{aligned} \tag{36}$$

Furthermore, the total computational cost for this non-leader in order to fully recover their requested subfiles is

$$\begin{aligned} C_{nl}^{dec} &= \left(1-\frac{M}{N}\right)^{N_e+1} F \\ &\times \left(\left(\frac{M}{N}(|Q_{u_1}|-1)+2\right) \prod_{i=2}^{N_e} \left(\frac{M}{N}|Q_{u_i}|+1\right) -2 \right) \\ &+ (K-1)\left(1-\frac{M}{N}\right) \frac{MF}{N} + o(F). \end{aligned} \tag{37}$$

Proof: See Appendix A-B for (36) and Appendix A-C for (37). ■

As before, these results depend on K, N_e, M, N, F and the pure tail sizes $|Q_{u_i}|$ for $u_i \in \mathcal{U}$, with N_e and pure tail sizes ultimately depending on the demand vector $\mathbf{d}$. Interestingly, although these results depend on a big number of parameters, they are quite compact. This becomes more evident when they are compared to the corresponding expressions (29) and (33) for the computational cost in the centralized caching scheme, which are quite more complicated, even if we use the simplified form of (31) for the term S. This fact bares the question of whether it could be possible to devise a different but still information-theoretically equivalent scheme for decentralized caching that could be more suited as the basic paradigm of coded caching with uncoded prefetching instead of the centralized one.

C. Performance Metrics

1) Centralized Caching: The problem with (29) and (33) is that, although they fully capture the corresponding computational costs involved with a particular demand vector, they implicate quite a big number of parameters, namely K, t, N_e and the sizes of all pure tails $|Q_{u_i}|$ for $u_i \in \mathcal{U}$. This makes the use of these expressions for any comparison or general characterization purposes overly specific and quite subjective (pending on the parameter choices one opts for) to bare any particular significance. Thus, in order to create suitable performance metrics for centralized caching we would like to have a more objective criterion. We would also like to have some expressions that would be more meaningful and representative of the big number of different values S can take.

One approach to do this would be to average all the pure tail sizes out. In other words, we can take into account all the different choices we have for the parameters $|Q_{u_i}|$ for a given number N_e of different requests and then calculate the average value of S. This is what we try to do in Appendix B-A and, after some manipulations and the use of a mathematical formula proven in Appendix C, we reach the following expression.

$$\bar{S} = \frac{\binom{K-2}{N_e-1+t}}{\binom{K-2}{N_e-1}} \left[\binom{2N_e-1+t}{t} + \binom{2N_e-2+t}{t} \right] \tag{38}$$

Note that N_e no longer depends on a particular demand vector $\mathbf{d}$ as it has become itself a free parameter of the expression.

Substituting this in (29) yields the averaged computational cost of a non-leader in order to compute the non-transmitted files from the transmitted ones.

$$\bar{C}_{nl}^{nt} = \bar{S}\frac{F}{\binom{K}{t}} + \binom{K-N_e-1}{t} \frac{F(t-2)}{\binom{K}{t}}, \tag{39}$$

.

Further, substituting $\bar{S}$ in (33) gives us the averaged computational cost for a non-leader.

$$\bar{C}_{nl} = \binom{K-1}{t}\frac{Ft}{\binom{K}{t}} + \left[\bar{S} - 2\binom{K-N_e-1}{t}\right] \frac{F}{\binom{K}{t}}, \tag{40}$$

Both of these expression suit our stated requirements for a meaningful metric of the corresponding computational costs.

We must stress here that these expressions are the average over all values of S for the different choices of $|Q_{u_i}|$ given N_e and express the corresponding average of the computational cost of a non-leader with respect to these values. They does not express the expected computational cost of a non-leader. The reason for this is that in deriving (38) we did not account for the multiple different requests leading to the same pure tail sizes. Also, such an expected value would require the assumption of a particular scheme by which leaders and non-leaders are assigned during different transmissions. Only then would one be able to compute the expectation with respect to that particular scheme. We see that seeking for an expected computational cost characterization not only makes the analysis more complicated but also re-introduces a particular level of subjectivity (the choice of the assignment scheme) that we are trying to avoid.

2) Decentralized Caching: Like we did for centralized caching, we would like a more representative expression of the different values C_{nl}^{dec} and $C_{nl}^{dec,nt}$ can take when the pure tail sizes $|Q_{u_i}|$, with $u_i \in \mathcal{U}$, range among their possible choices for specific K, N_e, M, N and F. As we discussed in the previous section, one way to do it would be to average out the pure tail sizes in the term

$$S_{dec} = \left(\frac{M}{N}(|Q_{u_1}| - 1) + 2\right) \prod_{i=2}^{N_e} \left(\frac{M}{N}|Q_{u_i}| + 1\right). \quad (41)$$

Again, we have removed the dependence of N_e on the particular demand vector $\mathbf{d}$ as we are interested in averaging out this expression with respect to the pure tail sizes, which is dependent on a specific demand vector.

Doing so will give us the following result:

$$\begin{aligned}\bar{S}_{dec} = {}_2F_1 &\left(\begin{matrix}-N_e, -K + N_e + 1 \\ N_e\end{matrix}; \frac{M}{N}\right) \\ &+ {}_2F_1 \left(\begin{matrix}-N_e + 1, -K + N_e + 1 \\ N_e\end{matrix}; \frac{M}{N}\right).\end{aligned} \quad (42)$$

In this expression, ${}_2F_1$ is the Gaussian hypergeometric function [64]. The derivation of the above formula is performed in Appendix B-B.

Substituting this in (36) and (37) we reach the following performance metrics corresponding to the averaged computational cost of a non-leader for computing the non-transmitted subfiles from the transmitted ones

$$\begin{aligned}\bar{C}_{nl}^{dec,nt} &= \left(1 - \frac{M}{N}\right)^{N_e+1} F \\ &\times \left(\bar{S}_{dec} + \frac{M}{N}(K - N_e - 1) - 2\right) + o(F),\end{aligned} \quad (43)$$

and the averaged computational cost of a non-leader $\bar{C}_{nl}^{dec}$ for the whole decoding process

$$\begin{aligned}\bar{C}_{nl}^{dec} &= \left(1 - \frac{M}{N}\right)^{N_e+1} F\left(\bar{S}_{dec} - 2\right) \\ &+ (K-1)\left(1 - \frac{M}{N}\right)\frac{MF}{N} + o(F).\end{aligned} \quad (44)$$

IV. Proposed Decoding Method

A. Method Description for Centralized Caching

In this section, we propose an alternative way for a non-leader $u^{nl} \in [K]\backslash\{\mathcal{U}\}$ to decode the subfile of interest $W_{d_{u^{nl}}, A_{t+1}\backslash\{u^{nl}\}}$ for any $(t+1)$-subset A_{t+1} comprised of non-leaders. Using this method, the non-leader will perform a direct computation of the subfile from previous leader-related transmissions without having to compute the signal $Y_{A_{t+1}}$ as it is normally done in [47]. Moreover, we show that the computational cost of this new method is equal to the cost of computing $Y_{A_{t+1}}$, thus saving the last step of figuring out $W_{d_{u^{nl}}, A_{t+1}\backslash\{u^{nl}\}}$ from $Y_{A_{t+1}}$. We first show the following lemma.

Lemma 1: Assume a coded caching system with N files of size F, K users and MF cache size per user where symmetric batch prefetching is used and $t = \frac{MK}{N} \in \{0, 1, \ldots K\}$. For any demand vector $\mathbf{d} = (d_1, \ldots, d_K)$ with $N_e \leqslant K - t - 1$, any leader set $\mathcal{U} = \{u_1, \ldots, u_{N_e}\}$, any $(t+1)$-non-leader subset $A_{t+1} \in \mathcal{A}_{t+1}^{nl}$ and any non-leader $u^{nl} \in A_{t+1}$ we have

$$\bigoplus_{\substack{\mathcal{V}\in\mathcal{V}^F \\ u^{nl}\in\mathcal{V}}} Y_{\mathcal{B}\backslash\mathcal{V}} \bigoplus_{\substack{\mathcal{V}\in\mathcal{V}^F \\ u^{nl}\notin\mathcal{V}}} W_{d_{u^{nl}}, \mathcal{B}\backslash(\mathcal{V}\cup\{u^{nl}\})} = 0, \quad (45)$$

where $\mathcal{B} = \mathcal{U} \cup A_{t+1}$ and $\mathcal{V}^F$ is the family of $\mathcal{V}$-sets of $\mathcal{B}$, each containing N_e users from $\mathcal{B}$ with each one requesting a different file.

Proof: We shall begin the proof by examining the very important Lemma 1 of [47] stating that

$$\bigoplus_{\mathcal{V}\in\mathcal{V}^F} Y_{\mathcal{B}\backslash\mathcal{V}} = 0. \quad (46)$$

This expression is derived by showing that every subfile participating in the above XOR appears exactly twice. We can separate the XOR operations according to whether the user u^{nl} is contained in the sets $\mathcal{V} \in \mathcal{V}^F$ or not, i.e.,

$$\bigoplus_{\substack{\mathcal{V}\in\mathcal{V}^F \\ u^{nl}\in\mathcal{V}}} Y_{\mathcal{B}\backslash\mathcal{V}} \bigoplus_{\substack{\mathcal{V}\in\mathcal{V}^F \\ u^{nl}\notin\mathcal{V}}} Y_{\mathcal{B}\backslash\mathcal{V}} = 0. \quad (47)$$

Further, the rightmost group of XOR can be re-written as

$$\begin{aligned}\bigoplus_{\substack{\mathcal{V}\in\mathcal{V}^F \\ u^{nl}\notin\mathcal{V}}} Y_{\mathcal{B}\backslash\mathcal{V}} &= \bigoplus_{\substack{\mathcal{V}\in\mathcal{V}^F \\ u^{nl}\notin\mathcal{V}}} \bigoplus_{x\in\mathcal{B}\backslash\mathcal{V}} W_{d_x, \mathcal{B}\backslash(\mathcal{V}\cup\{x\})} \\ &= \bigoplus_{\substack{\mathcal{V}\in\mathcal{V}^F \\ u^{nl}\notin\mathcal{V}}} \bigoplus_{\substack{x\in\mathcal{B}\backslash\mathcal{V} \\ x\neq u^{nl}}} W_{d_x, \mathcal{B}\backslash(\mathcal{V}\cup\{x\})} \oplus W_{d_{u^{nl}}, \mathcal{B}\backslash(\mathcal{V}\cup\{u^{nl}\})}.\end{aligned} \quad (48)$$

In the above expression, the subfiles $W_{d_x, \mathcal{B}\backslash(\mathcal{V}\cup\{x\})}$ are exactly the subfiles cached by user u^{nl}. Since, these files only appear here and the total XOR in (47) equals 0, each of these must appear exactly twice. This means we can remove them from the XOR and get

$$\bigoplus_{\substack{\mathcal{V}\in\mathcal{V}^F \\ u^{nl}\notin\mathcal{V}}} Y_{\mathcal{B}\backslash\mathcal{V}} = \bigoplus_{\substack{\mathcal{V}\in\mathcal{V}^F \\ u^{nl}\notin\mathcal{V}}} W_{d_{u^{nl}}, \mathcal{B}\backslash(\mathcal{V}\cup\{u^{nl}\})}. \quad (49)$$

We can see this rigorously by studying the term

$$A = \bigoplus_{\substack{\mathcal{V}\in\mathcal{V}^F \\ u^{nl}\notin\mathcal{V}}} \bigoplus_{\substack{x\in\mathcal{B}\backslash\mathcal{V} \\ x\neq u^{nl}}} W_{d_x, \mathcal{B}\backslash(\mathcal{V}\cup\{x\})} \quad (50)$$

and forming an argument similar to proof for Lemma 1 in [47]. In particular, the tails $B_u = \{x \in \mathcal{B} : d_x = d_u\}$ of the leaders $u \in \mathcal{U}$ partition the set $\mathcal{B}$ and the sets $\mathcal{V}^F$ can be decomposed as

$$\mathcal{V}^F = \left\{\{y\} \cup \mathcal{V}' \mid y \in \mathcal{B}_u, \mathcal{V}' \in \mathcal{V}_u\right\}. \quad (51)$$

In (51), $\mathcal{V}_u^F$ is the family of the sets formed by selecting one user from each tail in $\mathcal{B}$ except $\mathcal{B}_u$, i.e.,

$$\mathcal{V}_u^F = \left\{\mathcal{V} \in 2^{\mathcal{B}\backslash\mathcal{B}_u} : \begin{array}{l}|\mathcal{V}| = N_e \text{ - } 1, \\ \forall u_1 \neq u_2 \in \mathcal{V}\ d_{u_1} \neq d_{u_2}\end{array}\right\}. \quad (52)$$

These facts, allow us to perform the following manipulations

$$\begin{aligned}A &= \bigoplus_{u\in\mathcal{U}} \bigoplus_{\substack{\mathcal{V}\in\mathcal{V}^F \\ u^{nl}\notin\mathcal{V}}} \bigoplus_{\substack{x\in\mathcal{B}\cap\mathcal{B}_u\backslash\mathcal{V} \\ x\neq u^{nl}}} W_{d_x, \mathcal{B}\backslash(\mathcal{V}\cup\{x\})} \\ &= \bigoplus_{u\in\mathcal{U}} \bigoplus_{\substack{\mathcal{V}'\in\mathcal{V}_u^F \\ u^{nl}\notin\mathcal{V}'}} \bigoplus_{\substack{y\in\mathcal{B}_u \\ y\neq u^{nl}}} \bigoplus_{x\in\mathcal{B}_u\backslash\{y\}} W_{d_x, \mathcal{B}\backslash(\mathcal{V}'\cup\{x,y\})} \\ &= \bigoplus_{u\in\mathcal{U}} \bigoplus_{\substack{\mathcal{V}'\in\mathcal{V}_u^F \\ u^{nl}\notin\mathcal{V}'}} \bigoplus_{\substack{(x,y)\in\mathcal{B}_u^2 \\ x\neq y\neq u^{nl}}} \oplus W_{d_x, \mathcal{B}\backslash(\mathcal{V}'\cup\{x,y\})} = 0,\end{aligned} \quad (53)$$

completing the proof. ∎

We can now state the following theorem that provides the means of directly calculating the subfile $W_{d_{u^{nl}},A_{t+1}}$.

Theorem 7: Assume a coded caching system with N files of size F, K users and MF cache size per user where symmetric batch prefetching is used and $t = \frac{MK}{N} \in \{0, 1, \ldots K\}$. For any demand vector $\mathbf{d} = (d_1, \ldots, d_K)$ with $N_e \leqslant K - t - 1$, any leader set $\mathcal{U} = \{u_1, \ldots, u_{N_e}\}$, any $(t+1)$-non-leader subset $A_{t+1} \in \mathcal{A}^{nl}_{t+1}$ and any non-leader $u^{nl} \in A_{t+1}$ the requested subfile $W_{d_{u^{nl}},A_{t+1}}$ is given by

$$W_{d_{u^{nl}},A_{t+1}\backslash\{u^{nl}\}} = \bigoplus_{\substack{\mathcal{V}\in\mathcal{V}^F \\ u^{nl}\in\mathcal{V}}} Y_{\mathcal{B}\backslash\mathcal{V}} \bigoplus_{\substack{\mathcal{V}\in\mathcal{V}^F\backslash\{\mathcal{U}\} \\ u^{nl}\notin\mathcal{V}}} W_{d_{u^{nl}},\mathcal{B}\backslash(\mathcal{V}\cup\{u^{nl}\})}. \tag{54}$$

Proof: We note that

$$W_{d_{u^{nl}},A_{t+1}\backslash\{u^{nl}\}} = W_{d_{u^{nl}},\mathcal{B}\backslash(\mathcal{U}\cup\{u^{nl}\})}, \tag{55}$$

which means that the subfile $W_{d_{u^{nl}},A_{t+1}\backslash\{u^{nl}\}}$ is already present in (45). Hence, from (45), by isolating the term (55) we get (54). ∎

We should note that all the terms in the right hand side of (54) are either transmitted signals or subfiles requested by u^{nl} that are directly computable from the transmitted signals. This allows the non-leader u^{nl} to compute any subfile still missing from their requested file after all the leader-related transmissions have taken place.

B. Computational Analysis

In this section, we analyze the computational cost of our proposed decoding method. First, we should note that both our proposed method and ITODM in [47] coincide when decoding the subfiles $W_{d_{u^{nl}},A_{t+1}\backslash\{u^{nl}\}}$ come from actual transmissions $Y_{A_{t+1}}$ where there is at least one leader in A_{t+1}. So the computational cost for these files is the same and is given by (22). For the rest of the subfiles requested by u^{nl} the cost of computing them from previous transmissions and the subfiles collected from these transmissions is given by the following theorem.

Theorem 8: For any non-leader $u^{nl} \in [K]\backslash\{\mathcal{U}\}$ and any non-leader $(t+1)$-subset A_{t+1}, the cost of computing the subfile $W_{d_{u^{nl}},A_{t+1}\backslash\{u^{nl}\}}$ is given by

$$C_{nl}\left(W_{d_{u^{nl}},A_{t+1}\backslash\{u^{nl}\}}\right) = \left(|\mathcal{V}^F| - 2\right)\frac{F}{\binom{K}{t}}. \tag{56}$$

Proof: In order to find the cost of computing $W_{d_{u^{nl}},A_{t+1}\backslash\{u^{nl}\}}$ we count the number of XORs needed to compute (54). From (24) we have that

$$\begin{aligned}\left|\left\{\mathcal{V} \in \mathcal{V}^F : u^{nl} \in \mathcal{V}\right\}\right| &= \left|\left\{\mathcal{V}' \in \mathcal{V}^F_{u^{nl}} : u^{nl} \in \mathcal{V}\right\}\right| \\ &= \frac{|\mathcal{V}^F|}{|\mathcal{B}_{u_0}|}\end{aligned}, \tag{57}$$

$$\begin{aligned}\left|\left\{\mathcal{V} \in \mathcal{V}^F : u^{nl} \notin \mathcal{V}\right\}\right| &= |\mathcal{V}^F| - \left|\left\{\mathcal{V}' \in \mathcal{V}^F_{u^{nl}} : u^{nl} \in \mathcal{V}\right\}\right| \\ &= \frac{|\mathcal{B}_{u_0}| - 1}{|\mathcal{B}_{u_0}|}|\mathcal{V}^F|.\end{aligned} \tag{58}$$

So the number of XORs inside each of the two block-XORs of (54) will be

$$\left|\bigoplus_{\substack{\mathcal{V}\in\mathcal{V}^F \\ u^{nl}\in\mathcal{V}}} Y_{\mathcal{B}\backslash\mathcal{V}}\right|_c = \left(\frac{|\mathcal{V}^F|}{|\mathcal{B}_{u_0}|} - 1\right)\frac{F}{\binom{K}{t}}, \tag{59}$$

and

$$\left|\bigoplus_{\substack{\mathcal{V}\in\mathcal{V}^F\backslash\{\mathcal{U}\} \\ u^{nl}\notin\mathcal{V}}} W_{d_{u^{nl}},\mathcal{B}\backslash\mathcal{V}\cup\{u^{nl}\}}\right|_c = \left(\frac{|\mathcal{B}_{u_0}| - 1}{|\mathcal{B}_{u_0}|}|\mathcal{V}^F| - 2\right)\frac{F}{\binom{K}{t}}. \tag{60}$$

Summing those up, and taking into account that we have one more block-XOR between these two parts of (54), introducing $F/\binom{K}{t}$ additional bit XORs, we reach (56). ∎

Comparing (56) to (25) we can see that the proposed decoding method requires $Ft/\binom{K}{t}$ fewer XORs than ITODM per non-leader $(t+1)$-subset. Given that the number of non-leader $(t+1)$-subsets $A_{t+1} \in \mathcal{A}^{nl}_{t+1}$ containing a particular non-leader u^{nl} is

$$\begin{aligned}\left|\left\{A_{t+1} \in \mathcal{A}^{nl}_{t+1} : u^{nl} \in A_{t+1}\right\}\right| &= \left|\left\{A_t \in \mathcal{A}^{nl}_t : u^{nl} \notin A_t\right\}\right| \\ &= \binom{K - N_e) - 1}{t}\end{aligned} \tag{61}$$

we can state the following corollary.

Corollary 1: A non-leader using (54) to compute the requested subfiles $W_{d_{u^{nl}},A_{t+1}\backslash\{u^{nl}\}}$ corresponding to all $A_{t+1} \in \mathcal{A}^{nl}_{t+1}$ will have a total saving of

$$S_{nl} = \binom{K - N_e - 1}{t}\frac{Ft}{\binom{K}{t}} \tag{62}$$

XORs compared to ITODM.

Since there are $K - N_e$ non-leaders, we also get that the total computational savings throughout the whole system will be given by the following corollary.

Corollary 2: The total computational savings of using (54) throughout the system will be

$$\begin{aligned}S &= (K - N_e)\binom{K - N_e - 1}{t}\frac{Ft}{\binom{K}{t}} \\ &= \binom{K - N_e}{t+1}\frac{Ft(t+1)}{\binom{K}{t}}\end{aligned} \tag{63}$$

XORs compared to ITODM.

C. Extension to decentralized caching

The proposed decoding method finds applicability in any scenario that the centralized coded caching with uncoded prefetching appears in some form. In this section, we show how the previous results extend to the case of decentralized caching with uncoded prefetching and derive the corresponding computational savings per user as well as throughout the system.

In subsection II-C we have seen that the delivery phase of a decentralized system consists of multiple delivery phases. Each phase is equivalent to the delivery phase of a centralized caching system with file size $|\mathcal{L}^j_{A_j}(W)|$ (a.a.s.) for

$j \in \{0, 1, \ldots, K\}$. Also, for each $j \in \{1, 2, \ldots, K - N_e - 1\}$ there will be non-leader sets A_{j+1} whose corresponding transmissions will not take place but instead will be computed from the ones that do take place. For each such non-leader set, a non-leader could apply our proposed decoding method using (54) and save the number of computational operations given by Corollary 1. Summing up these computational savings for all $j \in \{1, \ldots, K - N_e - 1\}$ will yield the total computational savings of a non-leader u^{nl} for the decentralized caching system.

Theorem 9: In a decentralized caching scheme with uncoded prefetching, a non-leader using (54) to compute the requested subfiles $V_{d_{u^{nl}}, A_{t+1} \backslash \{u^{nl}\}}$ corresponding to all non-leader $(j+1)$-subsets $A_{j+1} \in \mathcal{A}^{nl}_{j+1}$ for all $j \in \{1, 2, \ldots, K - N_e - 1\}$, will have a total computational saving of

$$S^{dec}_{nl} = (K - N_e - 1)\left(1 - \frac{M}{N}\right)^{N_e+1} \frac{MF}{N} + o(F) \quad (64)$$

XOR operations (a.a.s.).

Proof: This quantity equals the term S_2 in the proof of (36) and is given by (90) after repositioning the term $o(F)$ that was stripped away for simplicity. ■

Again, since there are $K - N_e$ non-leaders, we have the following corollary.

Corollary 3: The (a.a.s.) total computational savings of using (54) throughout a decentralized caching system with uncoded prefetching will be

$$\begin{aligned} S^{dec} &= (K - N_e)(K - N_e - 1) \\ &\quad \times \left(1 - \frac{M}{N}\right)^{N_e+1} \frac{MF}{N} + o(F) \end{aligned} \quad (65)$$

XOR operations relative to ITODM.

V. Comparison

A. Centralized Caching

In this section, we would like to compare the efficiency of the centrilized ITODM scheme with the one we propose in this paper.

In order to compare ITODM with our proposed method, we calculate the relative computational improvement, defined as

$$a = \frac{S_{nl}}{\bar{C}_{nl}}. \quad (66)$$

The numerator is the total computational savings coming from using our proposed method, given in (62). The denominator is the average computational cost of the ITODM for the particular K, t and N_e given by (40). An important point that we have to clarify for this and the next sections is that although the value of a (and the other relative improvements we will examine) seems to be dependent on a particular demand vector $\mathbf{d}$, it is in fact dependent only on the value N_e of the demand vector, making each value characteristic of the broad class of different demand vectors having the same N_e. That is why we dropped the dependence of the numerator on $\mathbf{d}$ to avoid giving the wrong impression that the results are specific to a single demand vector $\mathbf{d}$. Note, also that the numerator depends only on K, t and N_e, making it suitable for our comparison. We do not mention the dependence with respect to F, as it is canceled out.

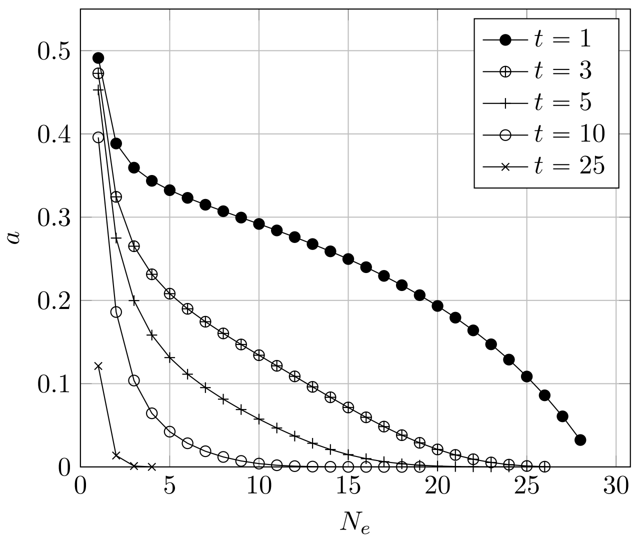
First, we will examine a small user case. Supposing we have $K = 30$ users and plotting a against N_e for various values of t, we get the results displayed in Fig. 3a. Note that the different plots have different endpoints. That is because, for a specific K and a specific t, N_e is allowed to vary up to $K - t - 1$ that is the highest value for which there are non-transmitted subsets and the ITODM can profit from our proposed method. We observe here a significant computational improvement for most kinds of requests. As the number of users t sharing the same information gets higher and as the number of distinct file requests N_e increases, this improvement becomes lower. Overall, we can see that for the small user case the proposed method can contribute significantly to the reduction of the computational cost of decoding the requested subfiles.

Next, we will examine the other end, which is a large user case. If we set $K = 300$ and plot a against N_e for various values of t we will get Fig. 3b. The first impression here is similar to the small user case. We have high gains for the smaller values of t and N_e that decrease as these parameters get higher. The important difference is that while in the small user case, the parameter t could get as high as $K - 2 = 28$ (otherwise there are no non-transmitted subfiles), in the large user case, the parameter t can get as high as $K - 2 = 298$. That means that only for a small region of arrangements, we actually have significant computational gains. This region contains the cases where the total cache memory of the users (KMF) is not much larger than the library size (NF). Thus, we can conclude that even for a large user set, the proposed computational method yields significant computational gains as long as the total cache memory of the system remains close in scale with the total library size (keeping the parameter t below 10). For larger total user cache sizes, the sheer amount of the transmissions that take place in the derivation of the non-transmitted subfiles is so big, that overshadows any computational benefit their direct computation offered by our method has, compared to ITODM.

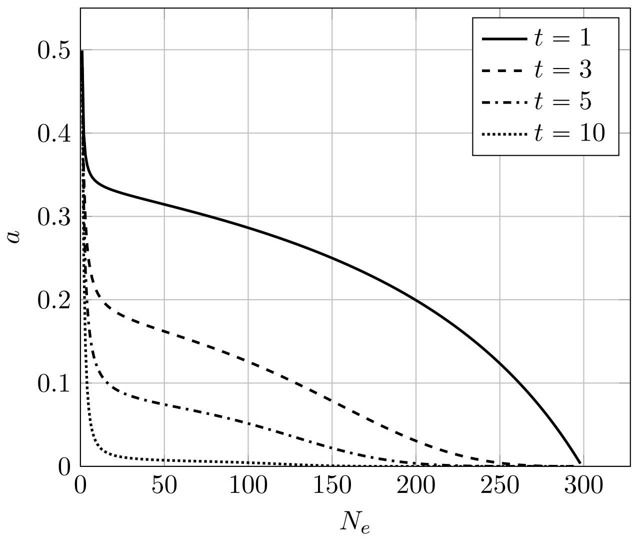
Another interesting comparison we can make is to study the improvement in the computational cost related to the derivation of the non-transmitted files from the transmitted ones. The difference is that this quantity does not include the cost for decoding the transmitted subfiles and can give us a more direct glimpse of the improvement in the actual calculation that takes place with respect to this derivation. Also, for the same reason, this quantity can be thought of as an upper bound for a. We can define this relative improvement as

$$a^{nt} = \frac{S_{nl}}{\bar{C}^{nt}_{nl}}. \quad (67)$$

Again, the numerator is given by (62) and represents the total computational gains of the proposed method. Also, the denominator is the average of the computational cost of deriving the non-transmitted subfiles from the transmitted ones for specific K, t and N_e given by (39). Plotting a^{nt} for the small and large user cases we had before, gives us the results presented in Fig. 4a and 4b. The main characteristics are similar to the previous plots of a and the comments we did there apply here

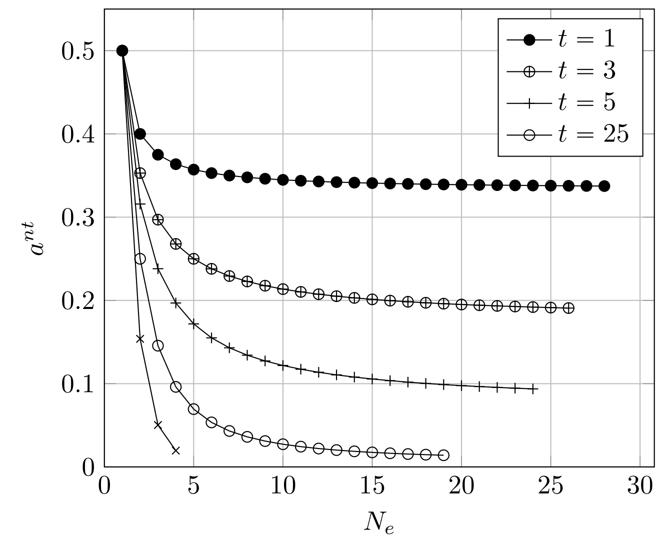


(a) Small User Case ($K = 30$).

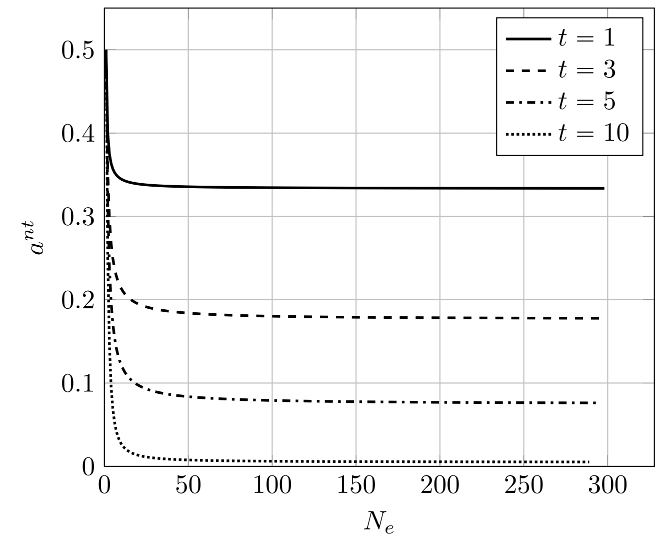


(b) Large User Case ($K = 300$).

Fig. 3: Relative computational improvement of a non-leader when fully decoding their requested file. For the small user case, a significant improvement is observed for most kinds of requests. For the large user case, we have a significant improvement in a region where the aggregate cache memory of the users is close in scale (less than 10 times) to the size of the library.



(a) Small User Case ($K = 30$).



(b) Large User Case ($K = 30$).

Fig. 4: Relative computational improvement of a non-leader when deriving their untransmitted subfiles from the transmitted ones. The major new characteristic we observe here is the presence of an asymptotic behavior having an exponential drop with respect to the parameter t.

as well. However, a major effect that we did not observe in a is the presence of an asymptotic behavior. What these graphs show is that as K gets larger, increasing the number of distinct requests N_e leads to a steady, non-vanishing improvement in the computational cost of deriving the non-transmitted subfiles. We can find an expression for this asymptotic value by letting K go to infinity while replacing N_e with its end value $K-t-1$. Doing so and using's Stirling's approximation [65] for the factorial leads us, after some straight forward manipulations, to the following expression

$$a_l^{nt} = \frac{t}{2^{t+1}}. \tag{68}$$

This expression corresponds to the exponential drop in the relative computational improvement we see in these figures and contributes to our understanding of the rapid decrease of the total computational improvement we observed in Fig. 3b for the large user case where this asymptotic behavior gets a chance to be fully expressed.

The fact that XOR operations are readily translatable to energy demands allows us to directly translate all the previous computational improvements to corresponding improvements in the system's energy profile. In other words, we can view a

and a^{nt} either as relative computational improvements or as relative improvements in energy efficiency. We would like to close this subsection by performing one more comparison that will allow us to appreciate the impact of the proposed method on the energy efficiency of the whole system.

In particular, we will examine the following relative improvement

$$r = \frac{S}{\bar{C}_t}. \tag{69}$$

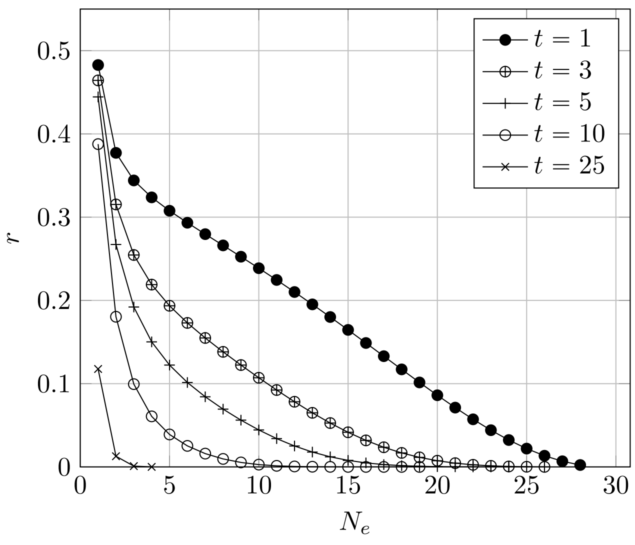
The numerator is the total computational savings coming from the proposed method throughout the system, given by (63). The denominator is the total computational cost among all leaders and non-leaders, and is given by

$$\bar{C}_t = N_e C_l + (K - N_e)\bar{C}_{nl}, \tag{70}$$

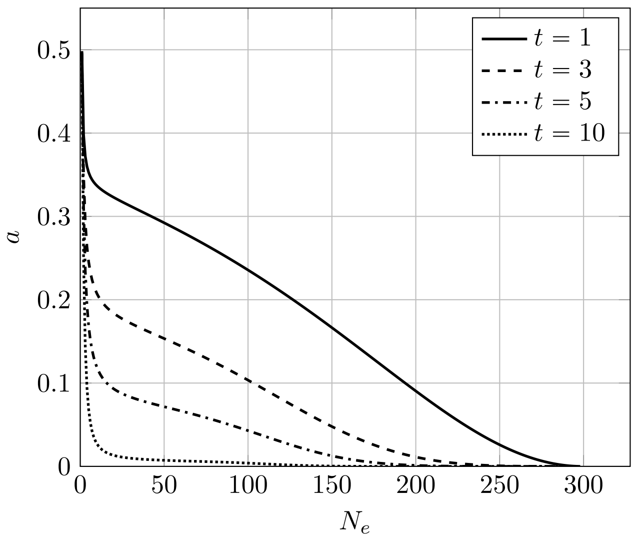
where C_l is given by (19) or (20) and $\bar{C}_{n,nl}$ by (33) if we replace the S term with its average $\bar{S}$ given in (38). Note that, as we did before, the computational cost for a non-leader is averaged out with respect to the pure tail sizes. This relative improvement is not very interesting from a computational point of view, as it expresses an improvement among computations that happen in parallel among the different users. However, it is very interesting from the energy consumption point of view, as is expresses the relative improvement in the energy consumed by the users as they perform their decoding task, which is an important figure of merit in the system's energy profile.

Plotting r against N_e for the small (K=30) and large (K=300) user cases, gives us the results displayed in Fig. 5a and 5b. We observe that r displays a similar behavior as a. In particular, in the small user case, the relative energy consumption has a significant improvement for most demands. It becomes lower as the number of users t sharing the same information gets higher and as the number of distinct requests N_e increases. In the large user case, the improvement is more pronounced in the region of small t corresponding to the cases where the total cache memory of the users (KMF) is not much larger than the library size (NF).

This similarity in behavior between r and a was expected since the system is naturally expected to benefit more when the separate non-leader users benefit more and vice versa, which is another expression of the imbalance in the computational burden between the leaders and the non-leaders.



(a) Small User Case ($K = 30$).



(b) Large User Case ($K = 300$).

Fig. 5: Relative improvement in the energy consumed by the users during the delivery phase. The behavior here is similar to a. For the small user case, we have a significant improvement for most kinds of request while for the large user case, we have a significant improvement in a region where the aggregate cache memory of the users is close in scale (less than 10 times) to the size of the library.

B. Decentralized Caching

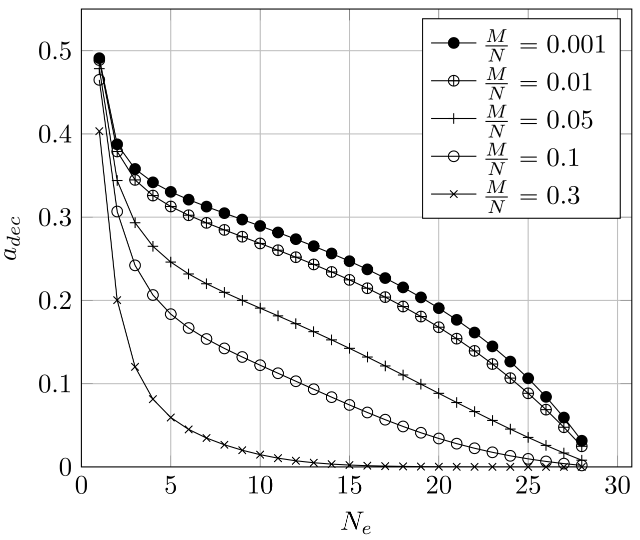
We proceed now to examine the relative computational improvement as we did for centralized caching. Again, we can define this quantity to be

$$a_{dec} = \frac{S_{nl}^{dec}}{\bar{C}_{nl}^{dec}}. \tag{71}$$

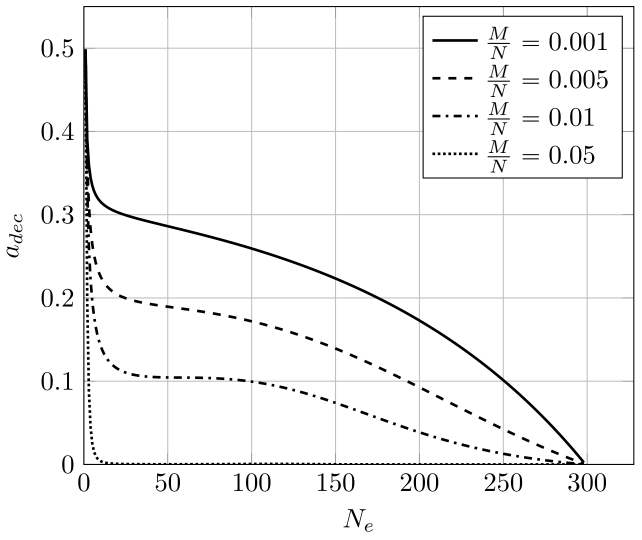
Here, S_{nl}^{dec} is the computational improvement of our proposed method for the decentralized caching, given by (64) and $\bar{C}_{nl}^{dec}$ the average computational cost of the decentralized ITODM for the particular K given by (44). As before, we do not take the dependence on F into account, as for adequately large values, it is practically canceled out.

Examining first the small user case ($K = 30$), plotting a_{dec} against N_e for different user-over-libary (M/N) ratios, we get the results displayed in Fig. 6a, where we see some quite different behavior from what we had in centralized caching. What we observe here is that as long as the individual user caches MF are small compared to the total library size NF, our proposed method yields significant computational benefits compared to the decentralized ITODM for all kinds of demands.

In Fig. 6b we plot the a_{dec} for the large user case ($K = 300$) and we observe the same behavior. As long as the user cache



(a) Small User Case ($K = 30$).



(b) Large User Case ($K = 300$).

Fig. 6: Relative computational improvement of a non-leader when fully decoding their requested file. We observe that as long as the individual user caches remain small compared to the total library size, there is a significant computational improvement for practically the whole range of demands.

size MF remains low compared to the total library size NF, our proposed method yields significant computational gains for all kinds of demands. What these results further illustrate is that as the user count K becomes higher, the computational gains slowly decrease. In other words, the range of values for M/N for which our method provides significant computational gains becomes smaller as the number of users increases. However, given that in most typical scenarios, the user caches are quite smaller compared to the total library size of the system, our proposed method still provides significant computational gains practically for the whole range of N_e.

As we did in the previous section, we can also compare the computational improvement with respect to the computational cost of deriving the untransmitted subfiles from the transmitted ones. We can define this relative computational improvement as

$$a_{dec}^{nt} = \frac{S_{nl}^{dec}}{\bar{C}_{nl}^{dec,nt}}. \tag{72}$$

The numerator here is the same as that in (71) and the denominator is the average computational cost for deriving the transmitted from the untransmitted subfiles for specific values of K, M, N and N_e given by (43). Plotting a_{dec}^{nt} for the small user case ($K = 30$) against N_e for different values of the user-to-library (M/N) ratio we get Fig. 7a. Again, we observe that we have significant computational gains in the whole range for N_e and we see an important difference. As N_e increased, the relative computational improvement a_{dec}^{nt} starts from value 0.5, decreases and the increases returning to value (asymptotically) equal to $1/3$. So, if we are interested in this kind of computational improvement, we can say that we have non-vanishing computational gains, as long as the user caches size MF remains small with respect to the total library size NF.

As we can see in Fig. 7b, where we plot the same quantity for the large user case ($K = 300$), this significant gain remains for all kinds of user demand, albeit for a somewhat smaller but still quite larger than the typical range of M/N ratio values.

Finally, we would like to examine the relative improvement in energy consumption for the system as a whole, as we did in the previous section for centralized caching. The corresponding relative improvement for decentralized caching will be

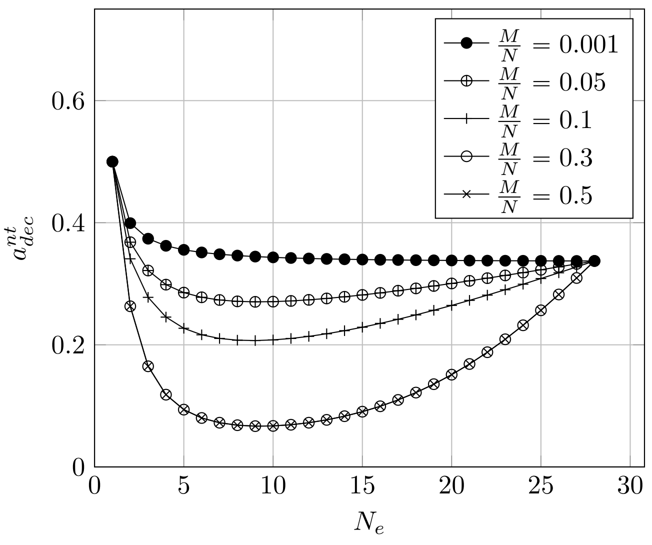
$$r_{dec} = \frac{S^{dec}}{\bar{C}_t^{dec}}. \tag{73}$$

The numerator is given by (65) and represents the total computational savings the proposed method provides to the system. The denominator is the total computational cost among all users and is given by

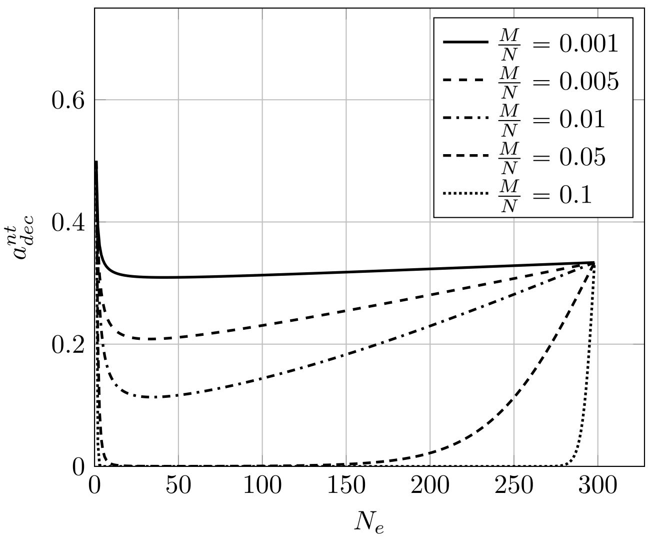
$$\bar{C}_t^{dec} = N_e C_l^{dec} + (K - N_e)\bar{C}_{nl}^{dec}, \tag{74}$$

where C_l^{dec} is given by (34) and $\bar{C}_{nl}^{dec}$ is given by (37) by replacing the term S_{dec}, as given by (41), with its average $\bar{S}_{dec}$ given in (42). We should note again here that the computational cost for a non-leader is averaged out with respect to the pure tail sizes. Examining the small ($K = 30$) and large ($K = 300$) user cases by plotting r_{dec} against N_e for various user-over-library (M/N) ratios, we get the results displayed in Fig. 8a and 8b. As with centralized caching, the behavior of r_{dec} closely resembles that of a_{dec}. In particular, we observe significant improvements in energy consumption among the users for almost all kinds of requests, both in the small as well as the large user case. The only condition is that the user-over-library ratio remains small, with the actual range becoming smaller as the number of users increases.

What the analysis of this section shows is that the proposed method provides significant computational and energy-related advantages over the decentralized ITODM for all kinds of system arrangements and demands, as long as the individual user cache size MF is small when compared to the total library size NF. This is a natural condition expected to hold

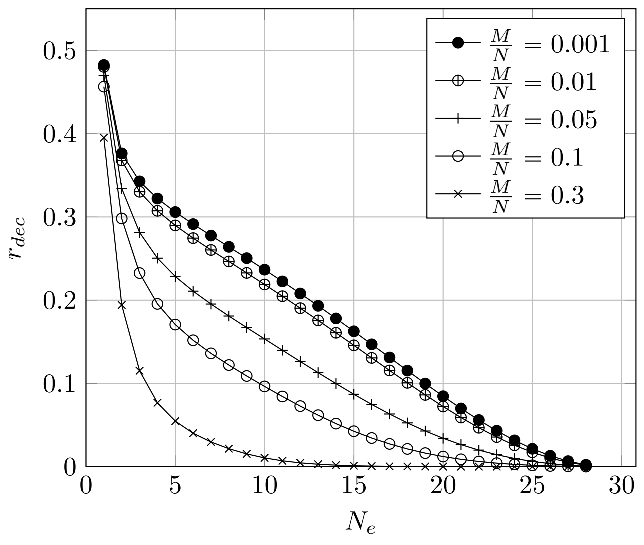


(a) Small User Case ($K = 30$).

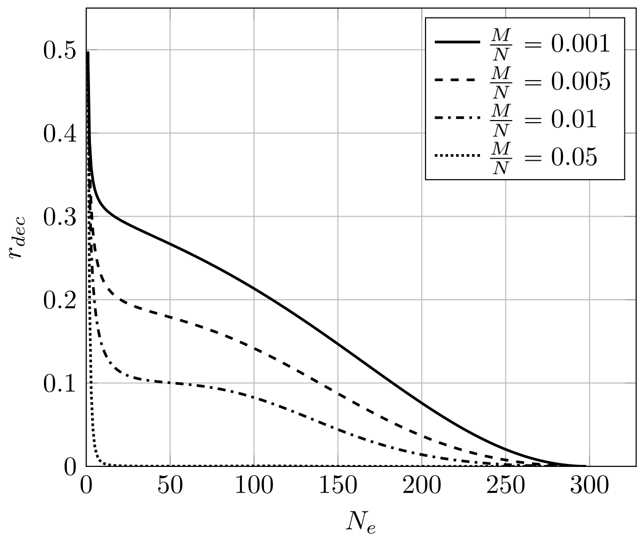


(b) Large User Case ($K = 300$).

Fig. 7: Relative computational improvement of a non-leader when deriving their untransmitted subfiles from the transmitted ones. We observe that as long as the individual user caches remain small compared to the library size, we have a significant non-vanishing computational gain that asymptotically converges to the same value.



(a) Small User Case ($K = 30$).



(b) Large User Case ($K = 300$).

Fig. 8: Relative improvement in the energy consumed by the users during the delivery phase. The behavior of r_{dec} closely resembles that of a_{dec} having significant improvements in energy consumption among the users, as long as the individual cache size of the users remains small in comparison to the library size.

for almost all caching systems, as the available memory in a user device (UE) is typically much smaller than the memory in a content delivery server or in a base station (BS).

VI. Conclusions

In this work, we have performed a complete computational analysis of information-theoretic optimal delivery method (ITODM) for centralized and decentralized caching. Both methods take advantage of the commonality in the file requests among the users to reduce the telecommunication load down to the information-theoretic optimal level. However, this is done at an exponentially increasing computational cost. Thus, our analysis allowed us to specify the exact amount of this computational cost down to the number of XOR operations required. This is an important figure of merit not only because it gives us an exact expression for the computational operations but is also readily translatable to energy demands both for the individual users and the overall system in general.

Furthermore, we have developed an alternative method for the delivery stage of centralized and decentralized caching that provide significant computational and energy consumption improvements over ITODM. This is achieved by introducing a computational shortcut in the derivation phase of the untransmitted subfiles from the transmitted ones. For centralized caching, the improvements are more pronounced when the number of users is small, or the total cache size among the users is comparable in scale to the total library size. For the more realistic case of decentralized caching, however, we observed significant computational improvements for all kinds of scenarios, as long as the individual user cache size remains small in comparison to the total library size, a condition that naturally happens in such systems.

Due to the principal position of ITODM in the domain of coded caching, any improvement or new results regarding it immediately reverberate outwards to all other kinds of coded caching systems. Thus, future research could extend the results of this paper to other coded caching systems and could examine other ways and different aspects of characterizing the computational costs involved. Minimizing the computational burden coming from the utilization of commonality and finding alternative schemes to balance out this cost among the non-leaders is still an open question and we aspire that this work will push the discussion forward.

Acknowledgement

The authors would like to express their gratitude towards the editor Mari Kobayasi as well as the reviewers, and especially reviewer 1, whose insightful suggestions enabled improved both the content and the presentation of the paper and lead to a more direct proof of the main expression we use in our proposed method than the one we had originally developed.

Appendix A

A. Simplification of the S term in Theorem 4

In this section, we prove (31). To do so, we start from (30) that we repeat here for convenience.

$$S = \sum_{\substack{(k1,\ldots,k_{N_e})\in R \\ k_1+\cdots+k_{N_e}=t}} \binom{|Q_{u_1}|-1}{k_1}\binom{|Q_{u_2}|}{k_2}\cdots\binom{|Q_{u_{N_e}}|}{k_{N_e}} \times (k_1+2)(k_2+1)\ldots(k_{N_e}+1), \tag{75}$$

where

$$R = \{0,\ldots,|Q_{u_1}|-1\} \times \{0,\ldots,|Q_{u_i}|\}^{N_e-1}. \tag{76}$$

Doing the multiplications among the different k_i gives the following result

$$S = \sum_{l=0}^{N_e} \left(\sum_{1<i_2<\cdots<i_l\leqslant N_e} \sum_{\substack{(k1,\ldots,k_{N_e})\in R \\ k_1+\cdots+k_{N_e}=t}} \binom{|Q_{u_1}|-1}{k_1}\cdots\binom{|Q_{u_{N_e}}|}{k_{N_e}} k_1 k_{i_1}\ldots k_{i_l} + 2 \sum_{1<i_1<\cdots<i_l\leqslant N_e} \sum_{\substack{(k1,\ldots,k_{N_e})\in R \\ k_1+\cdots+k_{N_e}=t}} \binom{|Q_{u_1}|-1}{k_1}\cdots\binom{|Q_{u_{N_e}}|}{k_{N_e}} k_{i_1}\ldots k_{i_l} \right). \tag{77}$$

We should note that the set R in this expression can be generalized to the whole $\mathbb{N}^{N_e}$ since there are not any additional tuples in $\mathbb{N}^{N_e}$ that lead to a non-zero summand. Now, using the binomial coefficient property $\binom{n}{k}k = n\binom{n-1}{k-1}$ and the generalized Vandermonde's identity [63], which reads

$$\sum_{\substack{(k_1,\ldots,k_p)\in\mathbb{N}^p \\ k_1+\cdots+k_p=m}} \binom{n_1}{k_1}\cdots\binom{n_p}{k_p} = \binom{n_1+\cdots+n_p}{m}, \tag{78}$$

as well as the fact that the sum of the pure tail sizes is equal to the number on non-leaders, we can reach the following form

$$S = \sum_{l=0}^{N_e} \binom{K-N_e-1-l}{t-l} \times \left(\sum_{1<i_2<\cdots<i_l\leqslant N_e} (|Q_{u_1}|-1)\,|Q_{u_{i_2}}|\ldots|Q_{u_{i_l}}| + 2 \sum_{1<i_1<\cdots<i_l\leqslant N_e} |Q_{u_{i_1}}|\ldots|Q_{u_{i_l}}| \right). \tag{79}$$

This form highlights the fact that for $N_e > t$ the summation ends at $l = t$. We could have seen this in the previous form of (77) if we had noticed that when $N_e > t$ the condition $k_1 + \ldots + k_{N_e} = t$ allows only for up to t terms to be non-zero at the same time. However, this is easier to see in (79). Also, since $(|Q_{u_1-1}) + |Q_{u_2}| + \cdots + |Q_{u_{N_e}}| = K - N_e - 1$, only up to $K - N_e - 1$ terms of the above sum can be non-zero at the same time. That means that if $N_e > K - N_e - 1$, the summation stops at $l = K - N_e - 1$.

This allows us to replace N_e in the sum with $\min(t, N_e)$ reaching (31) and completing the proof.

B. Proof of (36) in Theorem 6

In this section, we perform the proof of expression (36) in theorem 6. We recall that the decentralized caching is, in fact, broken down to multiple instances of centralized caching, with each instance dealing with the bits shared among j users, with $j \in \{0, 1, \ldots, K\}$. Since for $j \geqslant K-N_e$, there are no $(j+1)$-sets composed of solely non-leaders, all subfile transmissions take place and there is no computational cost involved with deriving any untransmitted subfiles. Thus, in order to find out the total cost of a non-leader related to the computation of the non-transmitted subfiles, we just have to add the individual computational costs for j up to $K-N_e-1$, as given by (29). If $C_{nl}^{nt}(j)$ is the computational cost of u^{nl} in the j-th instance of centralized caching, then the total cost for non-leader u^{nl} will be

$$\begin{aligned}C_{nl}^{dec,nt} = \sum_{j=0}^{K-N_e-1} C_{nl}^{nt}(j) = \underbrace{\sum_{j=0}^{K-N_e-1} S(j)\frac{|\mathcal{L}_j(W)|}{\binom{K}{j}}}_{S_1} \\ + \underbrace{\sum_{j=0}^{K-N_e-1}\binom{K-N_e-1}{j}\frac{|\mathcal{L}_j(W)|j}{\binom{K}{j}}}_{S_2} \\ - \underbrace{\sum_{j=0}^{K-N_e-1}\binom{K-N_e-1}{j}\frac{|\mathcal{L}_j(W)|2}{\binom{K}{j}}}_{S_3}.\end{aligned} \tag{80}$$

The above expression comes from (33) by replacing the file size F with the equivalent file size $|\mathcal{L}_j(W)|$ of the j-th instance of centralized caching given by (17) and instead of S using $S(j)$, which is the value of (30) for $t=j$. We should note that for $j=0$ the computational cost is actually zero since there is no derivation happening. The information in the corresponding Y_{A_1} transmissions that take place is readily available to any non-leader that is interested in it. However, we let j in the above expression start from zero because it will help us in our manipulations.

We will now examine each of the three summands by itself. We should note that for simplicity, in substituting $|\mathcal{L}_j(W)|$, we will omit the term $o(F)$ of (17), which nevertheless is always present. Starting with S_1 and using (31) for $S(j)$ gives us

$$\begin{aligned}S_1 &= \sum_{j=0}^{K-N_e-1}\sum_{l=0}^{\min(j,N_e)}\binom{K-N_e-1-l}{j-l}\mathcal{Q}(l) \\ &\qquad \times\left(\frac{M}{N}\right)^j\left(1-\frac{M}{N}\right)^{K-j}F \\ &= \sum_{\{j,l\}\in\mathcal{R}}\binom{K-N_e-1-l}{j-l}\mathcal{Q}(l) \\ &\qquad \times\left(\frac{M}{N}\right)^j\left(1-\frac{M}{N}\right)^{K-j}F,\end{aligned} \tag{81}$$

where

$$\begin{aligned}\mathcal{Q}(l) = &\sum_{1<i_2<\dots<i_l\leqslant N_e}(|Q_{u_1}|-1)\,|Q_{u_{i_2}}|\dots|Q_{u_{i_l}}| \\ &+ 2\sum_{1<i_1<\dots<i_l\leqslant N_e}|Q_{u_{i_1}}|\dots|Q_{u_{i_l}}|,\end{aligned} \tag{82}$$

and

$$\mathcal{R} = \left\{(j,l)\in\mathbb{N}^2 : \begin{array}{l} j\in\{0,\dots,K-N_e-1\}, \\ l\in\{0,\dots,\min(j,N_e)\}\end{array}\right\}. \tag{83}$$

Region $\mathcal{R}$ can be equivalently described as

$$\mathcal{R} = \left\{(j,l)\in\mathbb{N}^2 : \begin{array}{l} l\in\{0,\dots,m\} \\ j\in\{1,\dots,K-N_e-1\}\end{array}\right\}, \tag{84}$$

where

$$m = \min(K-N_e-1, N_e). \tag{85}$$

This allows us to exchange the order of summation in (81) and get

$$\begin{aligned}S_1 &= \sum_{l=0}^{m}\sum_{j=l}^{K-N_e-1}\binom{K-N_e-1-l}{j-l}\mathcal{Q}(l) \\ &\qquad \times\left(\frac{M}{N}\right)^j\left(1-\frac{M}{N}\right)^{K-j}F \\ &\overset{(a)}{=} \sum_{l=0}^{m}\mathcal{Q}(l)\left(\frac{M}{N}\right)^l\left(1-\frac{M}{N}\right)^{N_e+1}F \\ &\times\Bigg\{\sum_{j'=0}^{K-N_e-1-l}\binom{K-N_e-1-l}{j'} \\ &\times\left(\frac{M}{N}\right)^{j'}\left(1-\frac{M}{N}\right)^{K-N_e-1-l-j'}\Bigg\} \\ &\overset{(b)}{=} \sum_{l=0}^{m}\mathcal{Q}(l)\left(\frac{M}{N}\right)^l\left(1-\frac{M}{N}\right)^{N_e+1}F.\end{aligned} \tag{86}$$

In (a) we make the change $j'=j-l$ and tidy up the term a bit and in (b) we apply the binomial theorem [63]. Plugging in $\mathcal{Q}(l)$ from (82), the expression for S_1 becomes

$$\begin{aligned}S_1 = &\left(1-\frac{M}{N}\right)^{N_e+1}F \\ &\times\sum_{l=0}^{m}\Bigg(\sum_{1<i_2<\dots<i_l\leqslant N_e}\frac{M}{N}(|Q_{u_1}|-1) \\ &\qquad \times\frac{M}{N}|Q_{u_{i_2}}|\dots\frac{M}{N}|Q_{u_{i_l}}| \\ &\quad + 2\sum_{1<i_1<\dots<i_l\leqslant N_e}\frac{M}{N}|Q_{u_{i_1}}|\dots\frac{M}{N}|Q_{u_{i_l}}|\Bigg).\end{aligned} \tag{87}$$

Recovering now the product terms from the sums of this expression gives us

$$\begin{aligned}S_1 = &\left(1-\frac{M}{N}\right)^{N_e+1}F \\ &\times\left(\frac{M}{N}(|Q_{u_1}|-1)+2\right)\prod_{i=2}^{N_e}\left(\frac{M}{N}|Q_{u_i}|+1\right).\end{aligned} \tag{88}$$

The above derivation is straightforward if $m=N_e$. If $m=K-N_e-1$ then from

$$(|Q_{u_1}|-1)+|Q_{u_2}|+\dots+|Q_{u_{N_e}}| = K-N_e-1 \tag{89}$$

we have that only up to $K-N_e-1$ summands can be concurrently non-zero. Thus we can extend the sum of (87) up to N_e by including the zero valued terms and get (88).

The terms S_2 and S_3 are easier to handle. In particular

$$\begin{aligned} S_2 &= \sum_{j=0}^{K-N_e-1} \binom{K-N_e-1}{j} \left(\frac{M}{N}\right)^j \left(1-\frac{M}{N}\right)^{K-j} jF \\ &= \left(1-\frac{M}{N}\right)^{N_e+1} F \\ &\times \sum_{j=0}^{K-N_e-1} \binom{K-N_e-1}{j} \\ &\qquad \times \left(\frac{M}{N}\right)^j \left(1-\frac{M}{N}\right)^{K-N_e-1-j} j \\ &= \left(1-\frac{M}{N}\right)^{N_e+1} (K-N_e-1)\frac{MF}{N}. \end{aligned} \tag{90}$$

In this expression, we have used the formula [63] for the expected value of the binomial distribution $\mathcal{B}(K-N_e-1, M/N)$.

Finally, for the third term, using the binomial theorem, we can easily get

$$\begin{aligned} S_3 &= 2\sum_{j=0}^{K-N_e-1} \binom{K-N_e-1}{j} \left(\frac{M}{N}\right)^j \left(1-\frac{M}{N}\right)^{K-j} F \\ &= 2\left(1-\frac{M}{N}\right)^{N_e+1} F \\ &\times \sum_{j=0}^{K-N_e-1} \binom{K-N_e-1}{j} \\ &\qquad \times \left(\frac{M}{N}\right)^j \left(1-\frac{M}{N}\right)^{K-N_e-1-j} \\ &= 2\left(1-\frac{M}{N}\right)^{N_e+1} F. \end{aligned} \tag{91}$$

Plugging back all these terms in (80) and re-including the term $o(F)$ that we omitted before, we reach (36) that we are trying to prove.

C. *Proof of (37) in Theorem 6*

We now proceed to prove expression (37) in theorem 6. In order to derive (37) we just have to make some simple observations on (80) in order to extend it to the total computational cost of a non-leader. This expression gives the computational cost of u^{nl} in order to extract all the non-transmitted subfiles from the transmitted ones. So in order to get the total computational cost we have to add to this expression the cost of decoding the transmitted subfiles.

For each $j \in \{0, \dots, K-N_e-1\}$ the additional computational cost for decoding the transmitted subfiles is

$$\underbrace{\left[\binom{K-j}{j} - \binom{K-N_e-1}{j}\right]}_{\text{number of transmitted subfiles}} \underbrace{\frac{|\mathcal{L}_j(W)|j}{\binom{K}{j}}}_{\substack{\text{cost per}\\ \text{subfile}}}. \tag{92}$$

Also for each $j \in \{K-N_e, \dots, K-1\}$ the only computational cost is that of decoding the transmitting subfiles, since there are no $(j+1)$-subsets comprised solely of non-leaders. This additional cost is

$$\underbrace{\binom{K-j}{j}}_{\substack{(j+1)-\text{subsets}\\ \text{involving } u^{nl}}} \underbrace{\frac{|\mathcal{L}_j(W)|j}{\binom{K}{j}}}_{\substack{\text{cost per}\\ \text{subfile}}}. \tag{93}$$

We should note that for $j = K$, the corresponding bits are present in the caches of all users, and thus there is no computation (or even transmission) taking place. Adding up these additional computational costs in (80) we can realize that the total computational cost of u^{nl} will be given by this equation, if we replace the sum S_2 with

$$\begin{aligned} S_2' &= \sum_{j=0}^{K-1} \binom{K-1}{j} \frac{|\mathcal{L}_j(W)|j}{\binom{K}{j}} \\ &= \sum_{j=0}^{K-1} \binom{K-1}{j} \left(\frac{M}{N}\right)^j \left(1-\frac{M}{N}\right)^{K-j} jF \\ &= (K-1)\left(1-\frac{M}{N}\right)\frac{MF}{N}. \end{aligned} \tag{94}$$

Using this result and following the same operations for the other terms of (80) as well as re-positioning the term $o(F)$ that we omitted for simplicity, yields (37).

Appendix B

A. *Averaging of the S term in Theorem 4*

In this section, we perform an averaging over the pure tail sizes of (31) reaching (38). First of all, we have to remark that with respect to the averaging we are trying to perform N_e will be handled as a free parameter which is not fixed on a specific demand vector. We should further note that the number of ways we can choose the pure tail sizes equals the number of ways we can distribute the $K-N_e-1$ non-leaders into N_e sets. In other words, it is the number of N_e-tuples of non-negative integers whose sum is $K-N_e-1$. This is a well know combinatorics problem and the answer can be proven [66] to be $\binom{K-2}{N_e}$.

Then we have to find the value of the sum of each product term in (31) as the pure tail sizes move over their entire range. In particular, we have to find the following sums

$$\begin{aligned} &\sum_{\substack{(|Q_{u_1}|-1, |Q_{u_2}|, \dots, |Q_{u_{N_e}}|) \in R_q^{N_e} \\ |Q_{u_1}|+|Q_{u_2}|+\dots+|Q_{u_{N_e}}| = K-N_e}} (|Q_{u_1}|-1)\,|Q_{u_{i_2}}| \dots |Q_{u_{i_l}}| \\ &= \sum_{\substack{(q_1, \dots, q_{N_e}) \in \mathbb{N}^{N_e} \\ q_1+\dots+q_{N_e} = K-N_e-1}} q_1 \dots q_l, \end{aligned} \tag{95}$$

$$\begin{aligned} &\sum_{\substack{(|Q_{u_1}|, \dots, |Q_{u_{N_e}}|) \in R_q^{N_e} \\ |Q_{u_1}|+\dots+|Q_{N_e}| = K-N_e-1}} |Q_{u_{i_1}}| \dots |Q_{u_{i_l}}| \\ &= \sum_{\substack{(q_1, \dots, q_{N_e}) \in \mathbb{N}^{N_e} \\ q_1+\dots+q_{N_e} = K-N_e-1}} q_1 \dots q_l, \end{aligned} \tag{96}$$

where

$$R_q = \{0, \dots, K - N_e - 1\}. \tag{97}$$

In these two sums we make the substitutions $q_i = |Q_{u_{i_i}}|$ with the only exception of $q_1 = |Q_{u_1}| - 1$ for (95). The generalization from R_q to $\mathbb{N}$ is valid since it does not introduce any additional non-zero summands. So we actually see that for the same l these sums are in effect equal. Their value is given by (111) that we prove in the Appendix C and is equal to

$$\sum_{\substack{(q_1,\dots,q_{N_e})\in\mathbb{N}^{N_e} \\ q_1+\dots+q_{N_e}=K-N_e-1}} q_1 \dots q_l = \binom{K-2}{N_e-1+l} \tag{98}$$

Thus the only difference between the two sums in (31), apart from the multiplication with two in the second sum, is the number of product terms that are being summed up. The first, is the sum of products of l terms where the first term is always $|Q_{u_1}| - 1$. Thus the number of these product terms is equal to the ways we can choose $l-1$ things out of $N_e - 1$, or $\binom{N_e-1}{l-1}$. Similarly, for the second sum, since it is the sum of products of l terms whose selection excludes $|Q_{u_1}|$ their number will be equal to the ways we can select l things out of $N_e - 1$ or $\binom{N_e-1}{l}$.

So taking the average and replacing the above while doing some manipulations we get

$$\begin{aligned}\bar{S} &= \sum_{l=0}^{\min(t,N_e)} \binom{K-N_e-1-l}{t-l} \frac{\binom{K-2}{N_e-1+l}}{\binom{K-2}{N_e-1}} \\ &\quad \times \left(\binom{N_e-1}{l-1} + 2\binom{N_e-1}{l}\right) \\ &= \frac{(N_e-1)!(K-N_e-1)!}{(K-N_e-1-t)!} \\ &\quad \times \sum_{l=0}^{N_e} \frac{\binom{N_e}{l} + \binom{N_e-1}{l}}{(t-l)!(N_e-1+l)!}.\end{aligned} \tag{99}$$

Note that in this expression, we choose to use N_e as the upper limit of the sum, instead of $\min(t, N_e)$, because it will make our derivations more natural. This does not change the end result, as long as the "out of bounds" terms are taken to be zero, as they should. Also, in the above manipulations we have used the property

$$\binom{N_e-1}{l-1} + \binom{N_e-1}{l} = \binom{N_e}{l}. \tag{100}$$

In this handling, some extra care is warranted towards the first and the last terms of the sum to make sure that the zero-valued "out of bounds" terms appearing do not lead to different results.

We now focus on the following sums

$$\bar{S}_1 = \sum_{l=0}^{N_e} \frac{\binom{N_e}{l}}{(t-l)!(N_e-1+l)!}, \tag{101}$$

$$\bar{S}_2 = \sum_{l=0}^{N_e} \frac{\binom{N_e-1}{l}}{(t-l)!(N_e-1+l)!}. \tag{102}$$

Expanding the factorials and rearranging the multiplications, we can see that after a suitable pairing of the resulting terms, these sums can be expressed as

$$\begin{aligned}\bar{S}_1 &= \frac{N_e!}{t!(2N_e-1)!} \sum_{l=0}^{N_e} \binom{t}{l}\binom{2N_e-1}{N_e-l} \\ &= \frac{N_e!}{t!(2N_e-1)!} \binom{2N_e-1+t}{N_e} \\ &= \frac{1}{(N_e-1+t)!} \binom{2N_e-1+t}{t},\end{aligned} \tag{103}$$

$$\begin{aligned}\bar{S}_2 &= \frac{(N_e-1)!}{t!(2N_e-2)!} \sum_{l=0}^{N_e} \binom{t}{l}\binom{2N_e-2}{N_e-1-l} \\ &= \frac{(N_e-1)!}{t!(2N_e-2)!} \binom{2N_e-2+t}{N_e-1} \\ &= \frac{1}{(N_e-1+t)!} \binom{2N_e-2+t}{t}.\end{aligned} \tag{104}$$

In the above, we have used Vandermonde's identity as well as a proper pairing of the terms appearing in the binomial coefficients. Plugging back these results to (99) and properly rearranging the factorials yields (38), which concludes the proof.

B. Averaging of the term S_{dec} in (41)

In this section, we perform an averaging of (41) with respect to the possible values of $|Q_{u_i}|$ and derive (42). To derive this average, first of all, we unfold the product terms by performing the multiplications in (41) and get

$$S_{dec} = \sum_{l=0}^{m} \mathcal{Q}(l) \left(\frac{M}{N}\right)^l, \tag{105}$$

where $\mathcal{Q}(l)$ is given by (82). We can repeat now the reasoning of Appendix B-A to acquire the average of each $\mathcal{Q}(l)$. Doing so and plugging the result back to (105) we get

$$\begin{aligned}\bar{S}_{dec} &= \underbrace{\sum_{l=0}^{m} \left(\frac{M}{N}\right)^l \frac{\binom{K-2}{N_e-1+l}}{\binom{K-2}{N_e-1}} \binom{N_e}{l}}_{S_a} \\ &\quad + \underbrace{\sum_{l=0}^{m} \left(\frac{M}{N}\right)^l \frac{\binom{K-2}{N_e-1+l}}{\binom{K-2}{N_e-1}} \binom{N_e-1}{l}}_{S_b},\end{aligned} \tag{106}$$

We can now write S_{1a} as

$$S_a = \sum_{l}^{m} \left(\frac{M}{N}\right)^l \frac{\binom{N_e}{l}\binom{K-2}{K-N_e-1-l}}{\binom{K-2}{N_e-1}}, \tag{107}$$

and recognize the expression as the sum of the probability generating function for the hypergeometric distribution [64], after we substitute M/N by z. Thus we get

$$S_a = {}_2F_1\left(\begin{matrix}-N_e, -K+N_e+1 \\ N_e\end{matrix}; \frac{M}{N}\right). \tag{108}$$

Similarly, we can do the same for S_{1b} after we write it in the form

$$S_b = \sum_{l}^{m} \left(\frac{M}{N}\right)^l \frac{\binom{N_e-1}{l}\binom{K-2}{K-N_e-1-l}}{\binom{K-2}{N_e-1}}, \tag{109}$$

and get

$$S_b = {}_2F_1\left(\begin{matrix}-N_e+1, -K+N_e+1\\ N_e\end{matrix}; \frac{M}{N}\right). \tag{110}$$

Using these results in (106) we reach (42) that we are trying to prove.

Appendix C

In this appendix we prove the following very useful formula

$$\mathcal{S} = \sum_{\substack{(q_1,\ldots,q_n)\in\mathbb{N}^n\\ q_1+\cdots+q_n=N}} q_1\ldots q_k = \binom{N+n-1}{n-1+k} \qquad ,k \leqslant n. \tag{111}$$

Let us start with two sets. The first is the set of n-tuples of non-negative integers whose sum is N that lead to non-zero summands in (111)

$$A = \left\{(q_1,\ldots,q_n)\in\mathbb{N}^n : \begin{matrix} q_1+\cdots+q_n = N,\\ q_1 \geqslant 1,\ldots,q_k \geqslant 1\end{matrix}\right\}. \tag{112}$$

The second is the set of $(n+k)$-tuples of non-negative integers whose sum is $N-k$.

$$B = \left\{(q_1,\ldots,q_{n+k})\in\mathbb{N}^{n+k} : q_1+\ldots q_{n+k} = N-k\right\}. \tag{113}$$

We should note that in order for the sum of (111) to be non-zero, there must be at least one term $q_{1,0}\ldots q_{k,0} \neq 0$. That means $q_{1,0} \geqslant 1,\ldots,q_{k,0} \geqslant 1$. Since

$$\underbrace{q_{1,0}+\cdots+q_{k,0}}_{\geqslant k} + \underbrace{q_{k+1,0}+\cdots+q_{n,0}}_{\geqslant 0} = N,$$

we get $k \leqslant N$, which shows that the above definition of set B is always meaningful (in the sense of it always being non-empty).

Now we can define a function $f : B \to A$ such that if

$$b = (q_1,\ldots,q_n,q_{n+1},\ldots,q_{n+k}) \in B,$$

then

$$f(b) = (q_1+q_{n+1}+1,\ldots,q_k+q_{n+k}+1,q_{k+1},\ldots,q_n) \in A.$$

It is easy to see that the image $f(b)$ lies within A by a simple addition of its components giving N.

We further show that function f is a surjection. Let

$$a = (q_1,\ldots,q_k,q_{k+1},\ldots,q_n) \in A.$$

Then, if we take

$$b = (q_1-1,\ldots,q_k-1,q_k,\ldots,q_n,0,\ldots,0) \in B,$$

it is easy to see that $f(b) = a$. Because $q_i \geqslant 1$ for $i \in \{1,\ldots,k\}$ b is guaranteed to belong to B.

Since f is a surjection, we know that the sets

$$f^{-1}(a) = \{b \in B : f(b) = a\} \tag{114}$$

are equivalence classes of B. In particular, they form the quotient set with respect to the equivalence relation $b_1 \sim b_2 \Leftrightarrow f(b_1) = f(b_2)$.

Suppose now that

$$a = (q_1^a,\ldots,q_k,^a q_{k+1}^a,\ldots,q_n^a) \in A,$$

and we would like to characterize all the $b \in B$ that belong to $f^{-1}(a)$. The general form of a $b \in B$ is

$$b = (q_1,\ldots,q_k,q_{k+1},\ldots,q_n,i_1,\ldots,i_k).$$

The expression $f(b) = a$ imposes the following restrictions

$$\begin{cases} q_1 = q_1^a - 1 - i_1\\ \vdots\\ q_k = q_k^a - 1 - i_k\\ q_{k+1} = q_{k+1}^a\\ \vdots\\ q_n = q_n^a. \end{cases} \tag{115}$$

Here we can make three observations. First, that these conditions show that only the quantities i_j for $j \in \{1,\ldots,k\}$ are actually variable. Furthermore, the value of each such i_j can be selected independently from the others from the range $i_j \in \{0,1,\ldots,q_j^a-1\}$. The lower end comes from the fact that $i_j \geqslant 0$ and the upper end from $q_j \geqslant 0$. Secondly, that different choices lead to different $b \in f^{-1}(a)$ and thirdly that for any $b \in f^{-1}(a)$ there is a unique choice of $i_1,\ldots,i_k$ giving the above form.

These three observations show that there are $|f^{-1}(a)| = q_1^a\ldots q_k^a$ elements in $f^{-1}(a)$, which are the degrees of freedom in the above system of equations.

Now, we can go back to the sum in (111) where we can limit the range to the non-zero terms and write

$$\mathcal{S} = \sum_{a=(q_1,\ldots,q_n)\in A} q_1\ldots q_k = \sum_{a\in A} |f^{-1}(a)| = |B|. \tag{116}$$

But, since B is the set of all $(n+k)$-tuples of non-negative integers whose sum is $N-k$ its size is given by [66]

$$|B| = \binom{N+n-1}{n-1+k}. \tag{117}$$

completing the proof.

References

[1] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Commun. ACM*, vol. 28, no. 2, pp. 202–208, Feb. 1985.

[2] L. W. Dowdy and D. V. Foster, "Comparative models of the file assignment problem," *ACM Comput. Surv.*, vol. 14, no. 2, pp. 287–313, Jun. 1982.

[3] K. C. Almeroth and M. H. Ammar, "The use of multicast delivery to provide a scalable and interactive video-on-demand service," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 6, pp. 1110–1122, Aug 1996.

[4] A. Dan, D. Sitaram, and P. Shahabuddin, "Dynamic batching policies for an on-demand video server," *Multimedia Systems*, vol. 4, no. 3, pp. 112–121, Jun 1996.

[5] M. R. Korupolu, C. Plaxton, and R. Rajaraman, "Placement algorithms for hierarchical cooperative caching," *Journal of Algorithms*, vol. 38, no. 1, pp. 260 – 302, 2001.

[6] A. Meyerson, K. Munagala, and S. Plotkin, "Web caching using access statistics," in *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '01. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2001, pp. 354–363.

[7] I. Baev, R. Rajaraman, and C. Swamy, "Approximation algorithms for data placement problems," *SIAM J. Comput.*, vol. 38, no. 4, pp. 1411–1429, Aug. 2008.

[8] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *2010 Proceedings IEEE INFOCOM*, March 2010, pp. 1–9.

[9] Y. Birk and T. Kol, "Coding on demand by an informed source (iscod) for efficient broadcast of different supplemental data to caching clients," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2825–2830, June 2006.

[10] Z. Bar-Yossef, Y. Birk, T. S. Jayram, and T. Kol, "Index coding with side information," *IEEE Transactions on Information Theory*, vol. 57, no. 3, pp. 1479–1494, March 2011.

[11] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Transactions on Information Theory*, vol. 60, no. 5, pp. 2856–2867, May 2014.

[12] Z. Chen, P. Fan, and K. B. Letaief, "Fundamental limits of caching: improved bounds for users with small buffers," *IET Communications*, vol. 10, no. 17, pp. 2315–2318, 2016.

[13] K. Wan, D. Tuninetti, and P. Piantanida, "On caching with more users than files," in *2016 IEEE International Symposium on Information Theory (ISIT)*, July 2016, pp. 135–139.

[14] S. Sahraei and M. Gastpar, "K users caching two files: An improved achievable rate," in *2016 Annual Conference on Information Science and Systems (CISS)*, March 2016, pp. 620–624.

[15] C. Tian and J. Chen, "Caching and delivery via interference elimination," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1548–1560, March 2018.

[16] M. Mohammadi Amiri and D. Gündüz, "Fundamental limits of coded caching: Improved delivery rate-cache capacity tradeoff," *IEEE Transactions on Communications*, vol. 65, no. 2, pp. 806–815, Feb 2017.

[17] M. M. Amiri, Q. Yang, and D. Gündüz, "Coded caching for a large number of users," in *2016 IEEE Information Theory Workshop (ITW)*, Sep. 2016, pp. 171–175.

[18] M. A. Maddah-Ali and U. Niesen, "Decentralized coded caching attains order-optimal memory-rate tradeoff," *IEEE/ACM Transactions on Networking*, vol. 23, no. 4, pp. 1029–1040, Aug 2015.

[19] R. Pedarsani, M. A. Maddah-Ali, and U. Niesen, "Online coded caching," *IEEE/ACM Transactions on Networking*, vol. 24, no. 2, pp. 836–845, April 2016.

[20] J. Hachem, N. Karamchandani, and S. Diggavi, "Multi-level coded caching," in *2014 IEEE International Symposium on Information Theory*, June 2014, pp. 56–60.

[21] N. Karamchandani, U. Niesen, M. A. Maddah-Ali, and S. N. Diggavi, "Hierarchical coded caching," *IEEE Transactions on Information Theory*, vol. 62, no. 6, pp. 3212–3229, June 2016.

[22] J. Hachem, N. Karamchandani, and S. Diggavi, "Effect of number of users in multi-level coded caching," in *2015 IEEE International Symposium on Information Theory (ISIT)*, June 2015, pp. 1701–1705.

[23] M. Ji, G. Caire, and A. F. Molisch, "Fundamental limits of caching in wireless d2d networks," *IEEE Transactions on Information Theory*, vol. 62, no. 2, pp. 849–869, Feb 2016.

[24] U. Niesen and M. A. Maddah-Ali, "Coded caching with nonuniform demands," *IEEE Transactions on Information Theory*, vol. 63, no. 2, pp. 1146–1158, Feb 2017.

[25] J. Zhang, X. Lin, and X. Wang, "Coded caching under arbitrary popularity distributions," *IEEE Transactions on Information Theory*, vol. 64, no. 1, pp. 349–366, Jan 2018.

[26] M. Ji, A. M. Tulino, J. Llorca, and G. Caire, "Order-optimal rate of caching and coded multicasting with random demands," *IEEE Transactions on Information Theory*, vol. 63, no. 6, pp. 3923–3949, June 2017.

[27] A. Ramakrishnan, C. Westphal, and A. Markopoulou, "An efficient delivery scheme for coded caching," in *2015 27th International Teletraffic Congress*, Sep. 2015, pp. 46–54.

[28] M. A. Maddah-Ali and U. Niesen, "Cache-aided interference channels," in *2015 IEEE International Symposium on Information Theory (ISIT)*, June 2015, pp. 809–813.

[29] N. Naderializadeh, M. A. Maddah-Ali, and A. S. Avestimehr, "Fundamental limits of cache-aided interference management," *IEEE Transactions on Information Theory*, vol. 63, no. 5, pp. 3092–3107, May 2017.

[30] J. Hachem, U. Niesen, and S. Diggavi, "A layered caching architecture for the interference channel," in *2016 IEEE International Symposium on Information Theory (ISIT)*, July 2016, pp. 415–419.

[31] J. Hachem, U. Niesen, and S. N. Diggavi, "Degrees of freedom of cache-aided wireless interference networks," *IEEE Transactions on Information Theory*, vol. 64, no. 7, pp. 5359–5380, July 2018.

[32] C. Wang, S. H. Lim, and M. Gastpar, "Information-theoretic caching," in *2015 IEEE International Symposium on Information Theory (ISIT)*, June 2015, pp. 1776–1780.

[33] ——, "Information-theoretic caching: Sequential coding for computing," *IEEE Transactions on Information Theory*, vol. 62, no. 11, pp. 6393–6406, Nov 2016.

[34] S. H. Lim, C. Wang, and M. Gastpar, "Information-theoretic caching: The multi-user case," *IEEE Transactions on Information Theory*, vol. 63, no. 11, pp. 7018–7037, Nov 2017.

[35] R. Timo and M. Wigger, "Joint cache-channel coding over erasure broadcast channels," in *2015 International Symposium on Wireless Communication Systems (ISWCS)*, Aug 2015, pp. 201–205.

[36] S. S. Bidokhti, M. Wigger, and R. Timo, "Erasure broadcast networks with receiver caching," in *2016 IEEE International Symposium on Information Theory (ISIT)*, July 2016, pp. 1819–1823.

[37] S. Saeedi Bidokhti, M. Wigger, and R. Timo, "Noisy broadcast networks with receiver caching," *IEEE Transactions on Information Theory*, vol. 64, no. 11, pp. 6996–7016, Nov 2018.

[38] S. S. Bidokhti, M. Wigger, and R. Timo, "An upper bound on the capacity-memory tradeoff of degraded broadcast channels," in *2016 9th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, Sep. 2016, pp. 350–354.

[39] J. Zhang and P. Elia, "Fundamental limits of cache-aided wireless bc: Interplay of coded-caching and csit feedback," *IEEE Transactions on Information Theory*, vol. 63, no. 5, pp. 3142–3160, May 2017.

[40] J. Zhang, F. Engelmann, and P. Elia, "Coded caching for reducing csit-feedback in wireless communications," in *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Sep. 2015, pp. 1099–1105.

[41] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing," *IEEE Transactions on Information Theory*, vol. 64, no. 1, pp. 109–128, Jan 2018.

[42] S. Li, Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Edge-facilitated wireless distributed computing," in *2016 IEEE Global Communications Conference (GLOBECOM)*, Dec 2016, pp. 1–7.

[43] ——, "A scalable framework for wireless distributed computing," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2643–2654, Oct 2017.

[44] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coding for distributed fog computing," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 34–40, April 2017.

[45] A. Sengupta, R. Tandon, and T. C. Clancy, "Improved approximation of storage-rate tradeoff for caching via new outer bounds," in *2015 IEEE International Symposium on Information Theory (ISIT)*, June 2015, pp. 1691–1695.

[46] C. Wang, S. H. Lim, and M. Gastpar, "A new converse bound for coded caching," in *2016 Information Theory and Applications Workshop (ITA)*, Jan 2016, pp. 1–6.

[47] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "The exact rate-memory tradeoff for caching with uncoded prefetching," *IEEE Transactions on Information Theory*, vol. 64, no. 2, pp. 1281–1296, Feb 2018.

[48] C. H. H. Suthan, I. Chugh, and P. Krishnan, "An improved secretive coded caching scheme exploiting common demands," in *2017 IEEE Information Theory Workshop (ITW)*, Nov 2017, pp. 66–70.

[49] Kai Wan, D. Tuninetti, and P. Piantanida, "Novel delivery schemes for decentralized coded caching in the finite file size regime," in *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, May 2017, pp. 1183–1188.

[50] H. Ghasemi and A. Ramamoorthy, "Improved lower bounds for coded caching," *IEEE Transactions on Information Theory*, vol. 63, no. 7, pp. 4388–4413, July 2017.

[51] J. Gómez-Vilardebó, "Fundamental limits of caching: Improved rate-memory tradeoff with coded prefetching," *IEEE Transactions on Communications*, vol. 66, no. 10, pp. 4488–4497, Oct 2018.

[52] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Characterizing the rate-memory tradeoff in cache networks within a factor of 2," *IEEE Transactions on Information Theory*, vol. 65, no. 1, pp. 647–663, Jan 2019.

[53] K. Shanmugam, M. Ji, A. M. Tulino, J. Llorca, and A. G. Dimakis., "Finite-length analysis of caching-aided coded multicasting," *IEEE Transactions on Information Theory*, vol. 62, no. 10, pp. 5524–5537, Oct 2016.

[54] L. Tang and A. Ramamoorthy, "Coded caching schemes with reduced subpacketization from linear block codes," *IEEE Transactions on Information Theory*, vol. 64, no. 4, pp. 3099–3120, April 2018.

[55] Q. Yan, M. Cheng, X. Tang, and Q. Chen, "On the placement delivery array design for centralized coded caching scheme," *IEEE Transactions on Information Theory*, vol. 63, no. 9, pp. 5821–5833, Sep. 2017.

[56] C. Shangguan, Y. Zhang, and G. Ge, "Centralized coded caching schemes: A hypergraph theoretical approach," *IEEE Transactions on Information Theory*, vol. 64, no. 8, pp. 5755–5766, Aug 2018.

[57] K. Shanmugam, A. M. Tulino, and A. G. Dimakis, "Coded caching with linear subpacketization is possible using ruzsa-szemḗredi graphs," in *2017 IEEE International Symposium on Information Theory (ISIT)*, June 2017, pp. 1237–1241.

[58] G. Vettigli, M. Ji, A. M. Tulino, J. Llorca, and P. Festa, "An efficient coded multicasting scheme preserving the multiplicative caching gain," in *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2015, pp. 251–256.

[59] M. Ji, K. Shanmugam, G. Vettigli, J. Llorca, A. M. Tulino, and G. Caire, "An efficient multiple-groupcast coded multicasting scheme for finite fractional caching," in *2015 IEEE International Conference on Communications (ICC)*, June 2015, pp. 3801–3806.

[60] S. Jin, Y. Cui, H. Liu, and G. Caire, "A new order-optimal decentralized coded caching scheme with good performance in the finite file size regime," *IEEE Transactions on Communications*, vol. 67, no. 8, pp. 5297–5310, Aug 2019.

[61] S. M. Asghari, Y. Ouyang, A. Nayyar, and A. S. Avestimehr, "Optimal coded multicast in cache networks with arbitrary content placement," in *2018 IEEE International Conference on Communications (ICC)*, May 2018, pp. 1–6.

[62] G. Vettigli, M. Ji, K. Shanmugam, J. Llorca, A. M. Tulino, and G. Caire, "Efficient algorithms for coded multicasting in heterogeneous caching networks," *Entropy*, vol. 21, no. 3, 2019.

[63] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1994.

[64] C. Walck *et al.*, "Hand-book on statistical distributions for experimentalists," *University of Stockholm*, vol. 10, pp. 96–01, 2007.

[65] A. Papoulis and S. U. Pillai, *Probability, Random Variables, and Stochastic Processes*, 4th ed. McGraw Hill, 2002.

[66] R. P. Stanley, *Enumerative Combinatorics: Volume 1*, 2nd ed. New York, NY, USA: Cambridge University Press, 2011.

Sotirios K. Michos has received his Diploma and PhD from the Department of Electrical and Computer Engineering, Aristotle University of Thessaloniki (AUTH). During his PhD studies, he was part of the Wireless Communications & Information Processing (WCIP) Group at AUTH and worked in the broader area of Big Data and Network Science, with a special emphasis on their Information Theoretic aspects. His research interests span a range of subject areas including computer science, information theory, complexity, machine learning, telecommunications, control theory with a focus on their relationship to pure mathematics.

Panagiotis D. Diamantoulakis (M'13, SM'18) received the Diploma (five years) and PhD from the Department of Electrical and Computer Engineering, Aristotle University of Thessaloniki (AUTH), Greece, in 2012 and 2017, respectively. Since 2017, he works as a Post-doctoral Fellow in Wireless Communications & Information Processing (WCIP) Group at AUTH and, since 2021, he is also a visiting Assistant Professor in the Key Lab of Information Coding and Transmission at Southwest Jiaotong University (SWJTU), China. From 2018 to 2020, he also worked as visiting Post-doctoral Researcher in the Key Lab of Information Coding and Transmission at SWJTU and in the Institute for Digital Communications (IDC) of the Telecommunications Laboratory (LNT) at Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany. His current research interests include optimization theory and applications in integrated communication and computing networks, game theory, wireless power transfer, and optical wireless communications. He is a Working Group Member in the Newfocus COST Action "European Network on Future Generation Optical Wireless Communication Technologies". He serves as an Associate Editor for IEEE Wireless Communications Letters, IEEE Open Journal of the Communications Society, Physical Communications (Elsevier), and Frontiers in Communications and Networks. He was also an Exemplary Editor of IEEE Wireless Communications Letters in 2020, an Exemplary Reviewer of IEEE Communications Letters in 2014 and IEEE Transactions on Communications in 2017 and 2019 (top 3% of reviewers).

Leonidas Georgiadis (S'76-M'78-SM'96) received the Diploma degree in electrical engineering from Aristotle University, Thessaloniki, Greece, in 1979, and his M.S. and Ph.D degree both in electrical engineering from the University of Connecticut, in 1981 and 1986 respectively. From 1981 to 1983 he was with the Greek army. From 1986 to 1987 he was Research Assistant Professor at the University of Virginia, Charlottesville. In 1987 he joined IBM T. J. Watson Research Center, Yorktown Heights as a Research Staff Member. Since October 1995, he has been with the Telecommunications Department of Aristotle University, Thessaloniki, Greece. His interests are in the area of wireless networks, information theory and coding, scheduling and congestion control, energy efficient communications, distributed systems, routing, modeling and performance analysis. In 1992 he received the IBM Outstanding Innovation Award for his work on Goal-Oriented Workload Management for Multi-class systems and in 1994 the IBM Research Division Award for work on the architecture and design of Broadband Networking Systems.

George K. Karagiannidis (M'96-SM'03-F'14) was born in Pithagorion, Samos Island, Greece. He received the University Diploma (5 years) and PhD degree, both in electrical and computer engineering from the University of Patras, in 1987 and 1999, respectively. From 2000 to 2004, he was a Senior Researcher at the Institute for Space Applications and Remote Sensing, National Observatory of Athens, Greece. In June 2004, he joined the faculty of Aristotle University of Thessaloniki, Greece where he is currently Professor in the Electrical & Computer Engineering Dept. and Head of Wireless Communications & Information Processing (WCIP) Group. He is also Honorary Professor at South West Jiaotong University, Chengdu, China. His research interests are in the broad area of Digital Communications Systems and Signal processing, with emphasis on Wireless Communications, Optical Wireless Communications, Wireless Power Transfer and Applications and Communications & Signal Processing for Biomedical Engineering. Dr. Karagiannidis has been involved as General Chair, Technical Program Chair and member of Technical Program Committees in several IEEE and non-IEEE conferences. In the past, he was Editor in several IEEE journals and from 2012 to 2015 he was the Editor-in Chief of IEEE Communications Letters. Currently, he serves as Associate Editor-in Chief of IEEE Open Journal of Communications Society. Dr. Karagiannidis is one of the highly-cited authors across all areas of Electrical Engineering, recognized from Clarivate Analytics as Web-of-Science Highly-Cited Researcher during 2015-2021. Prof. Karagiannidis received the 2021 IEEE Communications Society Radio Communications Committee Technical Recognition Award and the 2018 Signal Processing and Communications Electronics Technical Recognition Award of the IEEE Communications Society.