# Unselfish Coded Caching Can Yield Unbounded Gains Over Selfish Caching

Federico Brunero and Petros Elia, *Member, IEEE*

***Abstract*— The original coded caching scenario assumes a content library that is of interest to all receiving users. In a realistic scenario though, the users may have diverging interests which may intersect to various degrees. What happens for example if each file is of potential interest to, say, $40\%$ of the users and each user has potential interest in $40\%$ of the library? In this work, we investigate the so-called *symmetrically selfish coded caching* scenario, where each user only makes requests from a subset of the library that defines its own *file demand set (FDS)*, each user caches selfishly only contents from its own FDS, and where the different FDSs symmetrically overlap to some extent. In the context of various traditional prefetching scenarios (prior to the emergence of coded caching), selfish approaches were known to be potentially very effective. On the other hand — with the exception of some notable works — little is known about selfish coded caching. We here present a new information-theoretic converse that proves, in a general setting of symmetric FDS structures, that selfish coded caching, despite enjoying a much larger local caching gain and a much smaller set of possible demands, introduces an unbounded load increase compared to the unselfish case. In particular, in the $K$-user broadcast channel where each user stores a fraction $\gamma$ of the library, where each file (class) is of interest to $\alpha$ users, and where any one specific file is of interest to a fraction $\delta$ of users, the optimal coding gain of symmetrically selfish caching is at least $(K-\alpha)\gamma+1$ times smaller than in the unselfish scenario. This allows us to draw the powerful conclusion that the optimal selfish coding gain is upper bounded by $1/(1-\delta)$, and thus does not scale with $K$. These derived limits are shown to be exact for different types of demands. In the end, this work provides, in a unified manner, the strong conclusion that selfish caching *can* cause unbounded performance deterioration in coded caching systems.**

***Index Terms*— Coded caching, file popularity, index coding, information-theoretic converse, selfish caching.**

## I. Introduction

THE vast increase of network traffic has sparked considerable interest in finding new techniques that reduce the communication load. Toward this, caching has been traditionally used to bring contents closer to their destinations, consequently reducing the volume of the communication problem during peak hours [1]. A key ingredient in using caches has commonly been the exploitation of the fact that some contents/files are more popular than others, and thus are generally to be allocated more cache space [2], [3]. This inevitably introduces the consideration that different users may have different file preferences, which in turn brings to the fore the concept of *selfish caching* where simply users cache independently and selfishly only contents that they are interested in potentially consuming themselves [4]–[7]. In the traditional prefetching scenario where emphasis is based heavily on bringing relevant content closer to each user, this idea of selfish caching brought about performance improvements [8], [9] in the form of higher local caching gains for each user.

A completely different utilization of caching was witnessed with the advent of coded caching [10], whose focus is more on leveraging storage capabilities in order to reduce interference. Depending on the network topology, this coded variant can be a more powerful approach than traditional prefetching, because it employs caching not only to change the *volume* of the communication problem, but also to change the *structure* of the problem itself, simply by changing the interference patterns. Coded caching has been rightfully credited with being able to transform memory into data rates, and has consequently sparked a flurry of research on a variety of topics such as on the interplay between caching and PHY [11]–[21], caching and privacy [22]–[24], on information-theoretic converses [25], [26], on the critical bottleneck of subpacketization [27]–[31], and a variety of other scenarios [32]–[40].

In trying to fuse the traditional caching techniques with coded caching, a variety of works has naturally sought to explore coded caching in the presence of files with different popularity. This is an area of active research that has produced several interesting and insightful results [41]–[51] that focus on the scenario where the file popularity profiles are identical for every user.

### A. Heterogeneous User Profiles and Selfish Coded Caching

On the other hand, we are just beginning to explore the connection between coded caching and selfish caching, where by selfish caching we generally refer to caching schemes in which each user caches only contents that meet its own individual preferences and objectives.

Manuscript received 25 May 2021; revised 12 April 2022; accepted 18 July 2022. Date of publication 1 August 2022; date of current version 22 November 2022. This work was supported by the European Research Council (ERC) through the European Union (EU) Horizon 2020 Research and Innovation Program under Grant 725929 (Project DUALITY). An earlier version of this paper was presented in part at the 2022 IEEE International Symposium on Information Theory (ISIT). *(Corresponding author: Petros Elia.)*

The authors are with the Communication Systems Department, EURECOM, Sophia Antipolis, 06410 Biot, France (e-mail: brunero@eurecom.fr; elia@eurecom.fr).

Communicated by M. A. Maddah-Ali, Associate Editor for Communications.

Recent works have sought to explore this connection. For example, in the context of coded caching with users having heterogeneous content preferences, the recent work in [52] took a game theoretic perspective to analyze the performance of coded caching when it accounts for this heterogeneity. Employing interesting analysis, this work revealed gains from taking this heterogeneity into consideration, where these gains were naturally a function of the structure of the user preferences. Furthermore, the work in [4] analyzed the peak load of three different coded caching schemes that account for the user preferences, and again revealed occasional performance gains that are similarly dependent on the structure of these preferences. Related analysis appears in [5], now for the average load of these same schemes in [4].

On the other hand, the work in [6] focused on finding instances where unselfish coded caching outperforms selfish designs. This work nicely considered the performance of selfish coded caching in the context of heterogeneous file demand sets, cleverly employing bounds to show that, for the case of $K = 2$ users and 3 files, unselfish designs strictly outperform selfish designs in terms of communication load, albeit only by a factor of $14\%$. In addition, the notable work in [7] established the optimal average load — under the assumption of selfish and uncoded prefetching — for the case of $K = 2$ users and a variety of overlaps between the two users' profiles, also providing explicit prefetching schemes of a selfish nature. To the best of our understanding, the above constitutes the extent of works on selfish coded caching.

### *B. An Adversarial Interplay Between Coded Caching and Selfish Caching*

Our motivation to understand the interplay between coded caching and selfish caching comes not only from the fact that coded caching systems may indeed need to operate under some selfish legacy constraints,[1] but also mainly from the fact that there exists an interesting "adversarial" interplay between coded caching and selfish prefetching. To understand this a bit better, we recall that the main idea of coded caching is that it multicasts at any given time a linear combination of different contents desired by different users. This implies that any one receiver associated to a multicast message must be able to find in its cache all the undesired contents (subfiles) of that multicast message. This is achieved in [10] by means of a highly structured and coordinated content placement phase, where each user caches a small fraction of *every* file of a common library. This relationship between undesired and cached contents deteriorates when using selfish caching, simply because each receiver selfishly opts — based on its own preferences — to not cache some of these undesired files. However, these same undesired files may eventually appear as interference at that selfish receiver who will now not be able to "cache-out" this interference. At the same time though, such selfish caching allows for a much more targeted placement of files such that each user can cache more of what it actually wants. Furthermore, such selfish scenario would correspond to a substantially smaller set of possible demands, which could conceivably be exploited to reduce the load.

To better understand the main challenge of the problem, let us consider the following simple scenario with $K = 3$ users and a library $\mathcal{L} = \{A, B, C\}$ of $N = 3$ files. Let us now assume that user 1 is only interested in files from the file demand set $\mathcal{F}_1 = \{A, B\}$, user 2 only from the set $\mathcal{F}_2 = \{A, C\}$, and user 3 only from $\mathcal{F}_3 = \{B, C\}$. We know from [25] that the Maddah-Ali and Niesen (MAN) scheme in [10] guarantees the smallest worst-case load (under uncoded placement) during the delivery phase if each user caches a proper fraction $M/N$ of *each* file, where $M$ represents the memory (in number of files) that each user is equipped with. However, if we know in advance the sets $\mathcal{F}_1$, $\mathcal{F}_2$ and $\mathcal{F}_3$, do we need each user to cache also from files that are not of interest to minimize the number of bits to be transmitted? Is it convenient or disadvantageous to have each user $k$ cache selfishly only from files in $\mathcal{F}_k$? What are the effects of selfish caching on coded caching?

### *C. Main Contributions*

To understand this interplay between coded caching and selfish caching, we first propose a new selfish model which aims to calibrate the selfishness effect, by calibrating the degree of separation between the interests of the different users. Our so-called symmetric file demand set (FDS) structure not only aims to encapsulate this aspect of intersection of interests, but is also designed to reflect and accentuate the aforementioned adversarial relationship between the selfish placement and the ability to encode across users as one would expect in the coded caching setting.

Then, for the aforementioned symmetric FDS structure, we employ index coding arguments to derive an information-theoretic converse (lower bound) on the optimal worst-case communication load under the assumption of uncoded and selfish placement. This bound proves that **unselfish coded caching can far outperform selfish coded caching**. Indeed, the bound makes clear the fact that, while, as noted, selfish caching implies a much smaller set of possible demands (cf. Defintion 2) as well as allows for a much more targeted placement of contents, these benefits will come, in our symmetric setting, at a heavy cost of fewer multicasting opportunities and a substantial loss in coding gain. The main contribution of our work is this information-theoretic converse, which allows us to draw the powerful conclusion that selfish coded caching can be, under some assumptions, rather detrimental.

This same converse offers some interesting insights on coding designs for selfish coded caching. While our converse now reveals that such designs, even if they are optimally constructed, would essentially never be able to provide good performance, these designs do pose an exceptionally interesting and challenging coding problem, which we address partially by providing, for a class of demands, achievable

[1]Here, we can think of a scenario where a server delivers — via a bottleneck link — content to caches, whose purpose is to bring content closer to the end user via dedicated non-interfering links. In such scenario, the delivery *to* the caches would benefit from a coded caching design, while the subsequent delivery *from* the caches would benefit from a selfish placement since the caches may target groups of users with potentially dissimilar interests.

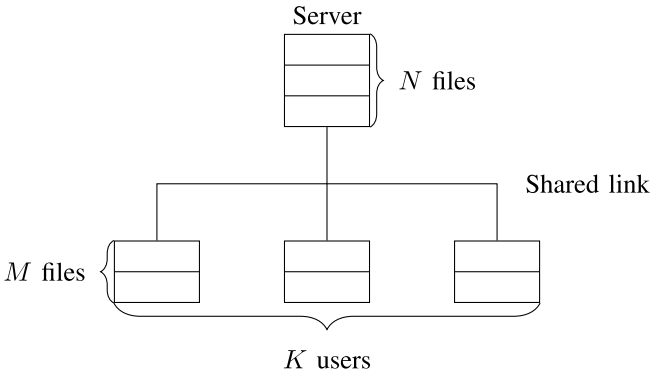


Fig. 1. A server with access to a library of $N$ files is connected through an error-free unit-capacity shared link to $K$ users, each having a cache of size equal to $M$ files.

schemes whose performance matches the expression of the converse.

*Remark 1:* We wish to highlight that the focus of the paper — which is to understand the effects of selfish caching policies — does not capture problems that arise from having a mismatch between the set of cached files and the set of demanded files.[2] Such problems remain, to date, open.

### D. Paper Outline

The rest of the paper is organized as follows. The system model is presented in Section II, where Section II-B offers a small motivating example that can help the reader appreciate the dynamics of selfish coded caching. Then, Section III presents the information-theoretic converse, whose proof in Section IV is followed by a clarifying example. The proposed selfish coded caching placement is presented in Section V and so are the delivery designs for some sets of demands. Additional optimal schemes are presented in Section VI for other sets of demands. Section VII concludes the paper, while some of the proofs are relegated in the appendices.

### E. Notation

We denote by $\mathbb{Z}^+$ the set of positive integers. For $n \in \mathbb{Z}^+$, we define $[n] \coloneqq \{1, 2, \ldots, n\}$. If $a, b \in \mathbb{Z}^+$ such that $a < b$, then $[a : b] \coloneqq \{a, a+1, \ldots, b-1, b\}$. For sets we use calligraphic symbols, whereas for vectors we use bold symbols. Given a finite set $\mathcal{A}$, we denote by $|\mathcal{A}|$ its cardinality. We use $\binom{n}{k}$ to denote the binomial coefficient $\frac{n!}{k!(n-k)!}$ and we let $\binom{n}{k} = 0$ whenever $n < 0$, $k < 0$ or $n < k$. We use the $\oplus$ symbol to denote the bitwise XOR operation. For $\boldsymbol{u} = (u_1, \ldots, u_K)$ being a permutation of the set $[K]$, we use $u\colon [K] \to [K]$ to denote the function which takes as input an element from $[K]$ and outputs its index position in $\boldsymbol{u}$.

## II. System Model

Similarly to the original scenario in [10], we consider the centralized caching scenario (cf. Fig. 1) where one central server has access to a library $\mathcal{L}$ containing $N$ files of $B$ bits each. This server is connected to $K$ users through a shared error-free broadcast channel, and each user is equipped with a cache of size $M$ files or, equivalently, $MB$ bits.

During the placement phase, the server fills the caches of the users according to a caching policy without knowing the future requests. During the delivery phase, when the users simultaneously reveal their demands, the server sends coded messages over the shared link to deliver the missing information to each user. Assuming that in the delivery phase each user demands simultaneously one file, the worst-case communication load $R$ is defined as the total number of transmitted bits, normalized by the file size $B$, that can guarantee delivery of all requested files in the worst-case scenario. The optimal communication load $R^\star$ is then formally defined as

$$R^\star(M) \coloneqq \inf\{R : (M, R) \text{ is achievable}\} \tag{1}$$

where the tuple $(M, R)$ is said to be *achievable* if there exists a caching-and-delivery scheme which guarantees, for any possible demand, a load $R$.

For the original coded caching scenario in [10] — where every file is of potential interest to each user — the load takes the form

$$R_{\text{MAN}}(t) = \frac{\binom{K}{t+1}}{\binom{K}{t}} = \frac{K-t}{t+1} = \frac{K(1-\gamma)}{K\gamma+1}, \quad \forall t \in [0 : K] \tag{2}$$

where $t \coloneqq KM/N = K\gamma$ is the so-called *cache redundancy* and $\gamma \coloneqq M/N$ is the fraction of the library that each user is able to store. This performance was proven in [25] (see also [26]) to be optimal under the assumption of uncoded cache placement. The above reveals a speedup factor of $t+1$ over the case of uncoded delivery. This speedup factor is a result of being able to serve *any* $(t+1)$-tuple of users with a multicast message, which can generally happen if we are able to store bits of each file to *any* possible $t$-tuple of caches. This symmetry will naturally be disrupted once selfish placement is imposed.

### A. The Symmetric $(K, \alpha, f)$ FDS Structure

To capture the interplay between coded caching and selfish caching, we propose an FDS structure that allows us to calibrate the degree of separation between the interests of the different users. To better understand this structure and generally to better understand the concept of an FDS, let us briefly consider a simplified toy example.

*Example 1:* Consider a downlink scenario with $K = 3$ users and a library $\mathcal{L} = \{A, B, C, D, E, F\}$ of $N = 6$ files.[3] Let us now assume that user 1 is only interested in potentially consuming files from the file demand set $\mathcal{F}_1 = \{A, B, C, D\}$, user 2 only from the set $\mathcal{F}_2 = \{A, B, E, F\}$, and user 3 only from $\mathcal{F}_3 = \{C, D, E, F\}$. In this setting, each user is interested in a fraction $2/3$ of the library, so for example user 1 has no interest in ever consuming the files in $\mathcal{L} \setminus \mathcal{F}_1 = \{E, F\}$.

[2] We can have such mismatch when, for example, each user caches only from a subset of files, but then requests content from the entire library, or when each user caches from the entire library but only requests files from a limited subset of this library.

[3] Such files can be movies, different episodes of a TV show, YouTube videos, etc.

Similarly, each file is of interest to the same fraction $2/3$ of users, so for example file $A$ is only of interest to user 1 and user 2.

For such a setting, we wish to understand the performance of selfish coded caching where each user caches only contents from its own FDS. We proceed with the formal definition of the FDS structure. We note that below an FDS will be defined as a collection of file *classes*, rather than just a collection of files. This allows for more generality and we believe it also better reflects how user preferences are often categorized.

*Definition 1 (The Symmetric $(K, \alpha, f)$ FDS Structure):* For $\alpha \in [K]$ and for $f \in \mathbb{Z}^+$, the symmetric $(K, \alpha, f)$ FDS structure assumes an $N$-file library $\mathcal{L} = \{\mathcal{W}_{\mathcal{S}} : \mathcal{S} \subseteq [K], |\mathcal{S}| = \alpha\}$ to be a collection of disjoint file classes $\mathcal{W}_{\mathcal{S}} = \{W_{i,\mathcal{S}} : i \in [f]\}$, with each class $\mathcal{W}_{\mathcal{S}}$ consisting of $f$ different files. In this setting, each user $k \in [K]$ has a file demand set

$$\mathcal{F}_k = \{\mathcal{W}_{\mathcal{S}} : \mathcal{S} \subseteq [K], |\mathcal{S}| = \alpha, k \in \mathcal{S}\} \tag{3}$$

which describes the files this user is potentially interested in.

As the above says, the library is split into $C = \binom{K}{\alpha}$ disjoint classes of files, corresponding to a total of $N = fC = f\binom{K}{\alpha}$ files. The above also says that each user $k$ is interested in its own FDS $\mathcal{F}_k$ of $|\mathcal{F}| = |\mathcal{F}_k| = f\binom{K-1}{\alpha-1}$ files. There are $K$ FDSs, one for each user, and each file class is identified by an $\alpha$-tuple $\mathcal{S}$ that tells us which $\alpha$ users are interested in this class.[4] Finally, we note that $\alpha = 1$ corresponds to the trivial scenario where there is no intersection between the user interests, while $\alpha = K$ corresponds to the traditional unselfish scenario where a common library of $N = f$ files[5] is of interest to every user.

In this context, selfish caching places the constraint that each user $k$ can only cache from its own FDS $\mathcal{F}_k$. Thus, one key aspect of such selfish caching is that it brings about an increase of the effective normalized cache size for each user. Indeed, whereas in the unselfish scenario each user can cache a fraction

$$\gamma = \frac{M}{N} = \frac{t}{K} \tag{4}$$

of each file of possible interest, in the selfish scenario this fraction is elevated to a larger

$$\gamma_\alpha \coloneqq \frac{M}{|\mathcal{F}|} = \frac{t}{\alpha} = \gamma\frac{K}{\alpha} \tag{5}$$

which in turn implies a larger local caching gain.

Deviating from standard notation practices, we will use the double-index notation $W_{f_k,\mathcal{D}_k}$ to denote the file requested by user $k$. Consequently, to describe the entire demand set, we will now be needing two vectors $\boldsymbol{d} = (\mathcal{D}_1, \ldots, \mathcal{D}_K)$ and $\boldsymbol{f} = (f_1, \ldots, f_K)$.

The above structure nicely lets us calibrate the fraction

$$\frac{|\mathcal{F}|}{N} = \frac{f\binom{K-1}{\alpha-1}}{f\binom{K}{\alpha}} = \frac{\alpha}{K} \tag{6}$$

[4]In other words, each file belongs to $\alpha$ FDSs. In particular, each file in class $\mathcal{W}_{\mathcal{S}}$ is of interest to the $\alpha$ users in $\mathcal{S}$. Hence, if $\mathcal{S} \ni k$, then the $f$ files in $\mathcal{W}_{\mathcal{S}}$ are in $\mathcal{F}_k$ and are thus of interest to user $k$. Finally, under our simplifying assumption that each user has its own FDS, $\alpha$ also describes the number of users interested in any one specific file.

[5]In this case we assume $f \geq K$.

TABLE I

IMPORTANT PARAMETERS FOR THE SYMMETRIC $(K, \alpha, f)$ FDS STRUCTURE

| | |
|---|---|
| Total FDSs | $K$ |
| Files per Class | $f$ |
| Total File Classes | $\binom{K}{\alpha}$ |
| Total Files | $f\binom{K}{\alpha}$ |
| File Classes per FDS | $\binom{K-1}{\alpha-1}$ |
| Files per FDS | $f\binom{K-1}{\alpha-1}$ |
| Fraction of Users Interested in a File | $\alpha/K$ |
| Fraction of Files of Interest to a User | $\lvert\mathcal{F}\rvert/N$ |

of the total library that each user is interested in. The imposed symmetry also yields a fraction $\delta \coloneqq \alpha/K$ of users interested in any one specific file.

The following two examples can help familiarize the reader with the notation.

*Example 2 (The Symmetric $(4, 2, 1)$ FDS Structure):* Let us consider the $(K, \alpha, f) = (4, 2, 1)$ structure which has $C = \binom{K}{\alpha} = 6$ file classes $\mathcal{W}_{12}, \mathcal{W}_{13}, \mathcal{W}_{14}, \mathcal{W}_{23}, \mathcal{W}_{24}, \mathcal{W}_{34}$, where[6] each class consists of $f = 1$ file. This corresponds to a library $\mathcal{L} = \{W_{1,12}, W_{1,13}, W_{1,14}, W_{1,23}, W_{1,24}, W_{1,34}\}$ of $N = 6$ files. In the above, $W_{1,12}$ simply represents the first (and, in this case, the only) file in class $\mathcal{W}_{12}$. The $K = 4$ FDSs take the form

$$\mathcal{F}_1 = \{W_{1,12}, W_{1,13}, W_{1,14}\} \tag{7}$$
$$\mathcal{F}_2 = \{W_{1,12}, W_{1,23}, W_{1,24}\} \tag{8}$$
$$\mathcal{F}_3 = \{W_{1,13}, W_{1,23}, W_{1,34}\} \tag{9}$$
$$\mathcal{F}_4 = \{W_{1,14}, W_{1,24}, W_{1,34}\} \tag{10}$$

where we recall that, for each file $W_{1,\mathcal{S}}$, the label $\mathcal{S}$ represents the FDSs the file belongs to. For example, file $W_{1,23}$ belongs to $\mathcal{F}_2$ and $\mathcal{F}_3$, and is thus of interest to user 2 and user 3. Finally we see that each user is interested in a fraction $|\mathcal{F}|/N = 0.5$ of the library, i.e., in $50\%$ of the library, and that each file is of interest to a fraction $\delta = \alpha/K = 0.5$ of the users.

*Example 3 (The Symmetric $(4, 3, 2)$ FDS Structure):* Let us consider the $(K, \alpha, f) = (4, 3, 2)$ structure which has $C = \binom{K}{\alpha} = 4$ classes $\mathcal{W}_{123}, \mathcal{W}_{124}, \mathcal{W}_{134}, \mathcal{W}_{234}$ and $N = 8$ files: $W_{1,123}$ and $W_{2,123}$ from class $\mathcal{W}_{123}$, then $W_{1,124}$ and $W_{2,124}$ from class $\mathcal{W}_{124}$, and so on. The $K$ FDSs take the form

$$\mathcal{F}_1 = \{\mathcal{W}_{123}, \mathcal{W}_{124}, \mathcal{W}_{134}\} \tag{11}$$
$$\mathcal{F}_2 = \{\mathcal{W}_{123}, \mathcal{W}_{124}, \mathcal{W}_{234}\} \tag{12}$$
$$\mathcal{F}_3 = \{\mathcal{W}_{123}, \mathcal{W}_{134}, \mathcal{W}_{234}\} \tag{13}$$
$$\mathcal{F}_4 = \{\mathcal{W}_{124}, \mathcal{W}_{134}, \mathcal{W}_{234}\} \tag{14}$$

where we see that each FDS consists of $2 \times 3 = 6$ files. For example, user 1 is interested in the files contained in the

[6]We will often omit braces and commas when indicating sets, such that for example $W_{\{1,2\}}$ may be written as $W_{12}$.

FDS $\mathcal{F}_1 = \{W_{1,123}, W_{2,123}, W_{1,124}, W_{2,124}, W_{1,134}, W_{2,134}\}$, user 2 is interested in the files contained in the FDS $\mathcal{F}_2 = \{W_{1,123}, W_{2,123}, W_{1,124}, W_{2,124}, W_{1,234}, W_{2,234}\}$, and so on. By calculating $|\mathcal{F}|/N = \alpha/K = 3/4$, we can verify that each user is interested in $75\%$ of the library, and each file is of interest to $75\%$ of the users.

The FDS structure automatically implies restrictions in the set of possible demand vectors. For instance, going back to Example 3, any demand with $\boldsymbol{d} = (234, 123, 123, 234)$ is not valid, because $\{1\} \notin \mathcal{D}_1 = \{2, 3, 4\}$, i.e., because file $W_{f_1,234}$ is not in $\mathcal{F}_1$ and thus would never be demanded by user 1. On the other hand, any demand with $\boldsymbol{d} = (124, 123, 123, 234)$ is valid because $k \in \mathcal{D}_k$ for each $k \in [K]$.

The set of valid demands as well as placement constraints that define selfish coded caching are now stated below.

*Definition 2 (Selfish Coded Caching With Uncoded Placement):* In selfish coded caching, a demand defined by the vectors $\boldsymbol{d} = (\mathcal{D}_1, \ldots, \mathcal{D}_K)$ and $\boldsymbol{f} = (f_1, \ldots, f_K)$ is said to be *valid* if and only if

$$k \in \mathcal{D}_k, \quad \forall k \in [K]. \tag{15}$$

Further, a cache placement is *uncoded* if the bits of the files are simply copied within the caches of the users and is *selfish* when it guarantees that a subfile of $W_{i,\mathcal{S}}$ can be cached at user $k$ only if $k \in \mathcal{S}$.

*Remark 2:* We acknowledge that the aforementioned symmetric FDS structure can be more restrictive than the (very few) existing FDS structures in the literature. For instance, the works in [4], [5] considered the FDS structure where each file is of interest to either groups of users or all users in the system. Nevertheless, our aim is to explore the effect of selfish caching, and what we show is that an important and general instance of selfish caching, embodied by the considered symmetric FDS structure, yields unbounded performance deterioration compared to the unconstrained (unselfish) case. Hence, this instance allows us to provide, in a unified manner, the main contribution of our work, which is the clear conclusion that indeed selfish caching can be unboundedly detrimental. The symmetric FDS structure serves as a proving step toward deriving this conclusion. At the same time, this same chosen FDS structure captures core principles of realistic file demand sets, like for example the amount of intersection between different such sets, where this intersection can be calibrated at will by the parameter $\alpha$.

### *B. Understanding the Dynamics of Selfish Coded Caching With an Example for the $(K, \alpha, f) = (5, 4, 1)$ Structure*

Let us consider a small motivating example that can help the reader appreciate the dynamics of symmetrically selfish coded caching. We will first suggest a selfish cache placement scheme that will be justified in Section V, and we will then present the delivery and decoding process for a class of valid circular demands. The corresponding load that will be achieved here will in fact be matched by the converse of the next section, consequently proving that in our example our delivery is optimal and the converse tight.

We here consider the $(K, \alpha, f) = (5, 4, 1)$ scenario, where each cache is of size $M = 2$ corresponding to the case of $t = 2$. In our scenario there are $C = \binom{K}{\alpha} = 5$ file classes $\mathcal{W}_{1234}, \mathcal{W}_{1235}, \mathcal{W}_{1245}, \mathcal{W}_{1345}, \mathcal{W}_{2345}$, and a total of $N = fC = 5$ library files. For simplicity, we will exploit the fact that $f = 1$ by slightly abusing notation such that, in this early example only, the library of $N = 5$ files will be denoted as $\mathcal{L} = \{W_{1234}, W_{1235}, W_{1245}, W_{1345}, W_{2345}\}$. At this point, the 5 FDSs take the form

$$\mathcal{F}_1 = \{W_{1234}, W_{1235}, W_{1245}, W_{1345}\} \tag{16}$$
$$\mathcal{F}_2 = \{W_{1234}, W_{1235}, W_{1245}, W_{2345}\} \tag{17}$$
$$\mathcal{F}_3 = \{W_{1234}, W_{1235}, W_{1345}, W_{2345}\} \tag{18}$$
$$\mathcal{F}_4 = \{W_{1234}, W_{1245}, W_{1345}, W_{2345}\} \tag{19}$$
$$\mathcal{F}_5 = \{W_{1235}, W_{1245}, W_{1345}, W_{2345}\}. \tag{20}$$

*1) Placement:* The cache placement will follow a selfish adaptation of the MAN scheme. First each file is split into $\binom{\alpha}{t} = \binom{4}{2} = 6$ non-overlapping subfiles as

$$W_{1234} = \{W_{1234,12}, W_{1234,13}, W_{1234,14}, W_{1234,23}, W_{1234,24}, W_{1234,34}\} \tag{21}$$
$$W_{1235} = \{W_{1235,12}, W_{1235,13}, W_{1235,15}, W_{1235,23}, W_{1235,25}, W_{1235,35}\} \tag{22}$$
$$W_{1245} = \{W_{1245,12}, W_{1245,14}, W_{1245,15}, W_{1245,24}, W_{1245,25}, W_{1245,45}\} \tag{23}$$
$$W_{1345} = \{W_{1345,13}, W_{1345,14}, W_{1345,15}, W_{1345,34}, W_{1345,35}, W_{1345,45}\} \tag{24}$$
$$W_{2345} = \{W_{2345,23}, W_{2345,24}, W_{2345,25}, W_{2345,34}, W_{2345,35}, W_{2345,45}\} \tag{25}$$

and then the cache $\mathcal{Z}_k$ of each user $k \in [5]$ is filled as

$$\mathcal{Z}_k = \{W_{\mathcal{S},\mathcal{T}} : \mathcal{S} \subseteq [5], |\mathcal{S}| = 4, \mathcal{T} \subseteq \mathcal{S}, |\mathcal{T}| = 2, k \in \mathcal{T}\}. \tag{26}$$

For example, user 1 would have to cache parts only from files $\{W_{1234}, W_{1235}, W_{1245}, W_{1345}\}$ in order to abide by the selfish constraint, and then, to abide by the cache size constraint, user 1 would cache subfiles labeled by $\{12, 13, 14\}$. Similarly, user 2 would cache only from $\{W_{1234}, W_{1235}, W_{1245}, W_{2345}\}$, and only the subfiles labeled by $\{12, 23, 24\}$, and so on.

*2) Delivery:* The delivery takes place as soon as the requests of the users are revealed. Consider the demand $\boldsymbol{d}_1 = (1234, 2345, 1345, 1245, 1235)$. A schematic of this demand is given by means of the graph in Fig. 2. This graph, which we refer to as the *FDS request graph*, is a directed graph where each vertex is a user and where there is an edge from user $k_1$ to user $k_2$ if $W_{\mathcal{D}_{k_1}} \in \mathcal{F}_{k_2}$. This graph represents at a high level, for each given demand vector $\boldsymbol{d}$, the interplay between the users' interests.

As a consequence of the aforementioned cache placement, each user does not cache (and consequently desires) a total of $\binom{\alpha-1}{t} = \binom{3}{2} = 3$ subfiles for its demanded file. Hence, given the demand $\boldsymbol{d}_1$, the desired subfiles are given as follows.

- User 1 desires the subfiles $W_{1234,23}$, $W_{1234,24}$ and $W_{1234,34}$.
- User 2 desires the subfiles $W_{2345,34}$, $W_{2345,35}$ and $W_{2345,45}$.

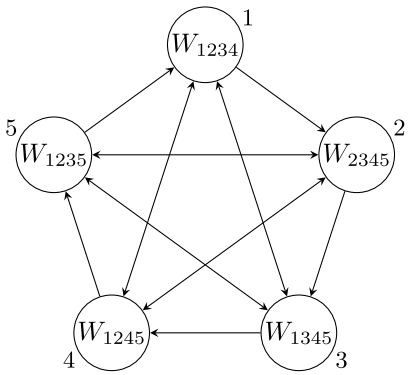


Fig. 2. FDS request graph for the $(K, \alpha, f) = (5, 4, 1)$ FDS structure and the demand $\boldsymbol{d}_1 = (1234, 2345, 1345, 1245, 1235)$.

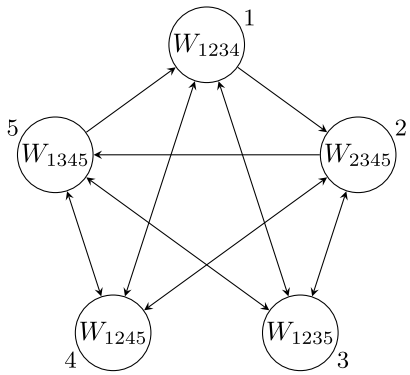


Fig. 3. FDS request graph for the $(K, \alpha, f) = (5, 4, 1)$ FDS structure and the demand $\boldsymbol{d} = (1234, 2345, 1235, 1245, 1345)$.

- User 3 desires the subfiles $W_{1345,14}$, $W_{1345,15}$ and $W_{1345,45}$.
- User 4 desires the subfiles $W_{1245,12}$, $W_{1245,15}$ and $W_{1245,25}$.
- User 5 desires the subfiles $W_{1235,12}$, $W_{1235,13}$ and $W_{1235,23}$.

One key aspect for achieving optimality is the utilization of specifically structured linear combinations of multicast messages, where this structure accepts the following interesting interpretation. These linear combinations effectively allow multicast messages to be used not only to deliver desired content to users, but also to deliver undesired content that can be used as side information to "bridge" the gaps left by the selfish placement. In essence, each transmission now delivers desired content while also disseminating side information that can be used to create *cliques*. To see this, let us consider the following sequence of XORs

$$X_1 = W_{1345,14} \oplus W_{1234,24} \oplus W_{1245,12} \tag{27}$$
$$X_2 = W_{2345,35} \oplus W_{1235,13} \oplus W_{1345,15} \tag{28}$$
$$X_3 = W_{1345,14} \oplus W_{2345,35} \oplus W_{1234,23} \tag{29}$$
$$X_4 = W_{1234,34} \oplus W_{1245,15} \tag{30}$$
$$X_5 = W_{2345,45} \oplus W_{1235,12} \tag{31}$$
$$X_6 = W_{2345,34} \oplus W_{1245,25} \tag{32}$$
$$X_7 = W_{1345,45} \oplus W_{1235,23} \tag{33}$$

transmitted one after the other. Recalling that each file is split into 6 non-overlapping subfiles, we know that each XOR has size $|X_i| = B/6$ for each $i \in [7]$.

By using its own cache, each user can now decode its own desired content as follows.

- User 1 can recover its desired subfiles from $X_1$, $X_2 \oplus X_3$ and $X_4$.
- User 2 can recover its desired subfiles from $X_1 \oplus X_3$, $X_5$ and $X_6$.
- User 3 can recover its desired subfiles from $X_2$, $X_3$ and $X_7$.
- User 4 can recover its desired subfiles from $X_1$, $X_4$ and $X_6$.
- User 5 can recover its desired subfiles from $X_2$, $X_5$ and $X_7$.

For example, in the above, user 1 needs $W_{2345,35}$ to correctly decode its desired $W_{1234,23}$ from $X_3$, whereas user 2 needs $W_{1345,14}$ to correctly decode $W_{2345,35}$ always from $X_3$. The act of "passing" subfiles $W_{2345,35}$ and $W_{1345,14}$ to user 1 and user 2 with $X_2$ and $X_1$, respectively, allows the creation of a clique between user 1, user 2 and user 3. This clique is exploited by creating the XOR $X_3$. The corresponding communication load is equal to $R(t = 2) = |X|/B = 7/6$, which will be met by the converse.

We wish to point out that, intuitively, the set of multicast messages above is constructed around the idea that each transmission serves not only to deliver desired content, but also, at the same time, to form cliques that will be exploited in future transmissions. Indeed, in the above we chose $t+1 = 3$ pivotal users (i.e., user 1, user 2 and user 3) which represent an *almost complete clique* in the side information graph of the induced index coding problem. All the effort consisted then in trying to "deliver" side information[7] to "bridge" the gaps left by the selfish placement, while delivering desired content to users. This was done in $X_1$ and $X_2$, where the subfiles $W_{1345,14}$ and $W_{2345,35}$ were carefully "delivered" to user 2 and user 1, respectively, without interfering with users to which $X_1$ and $X_2$ are delivering desired information. This interpretation related to the creation of cliques is a crucial part of the dynamics of the problem that we are considering.

Now, let us consider the demand vector given by $\boldsymbol{d}_2 = (1234, 2345, 1235, 1245, 1345)$ with its corresponding FDS request graph that is shown in Fig. 3. Since the graphs in Fig. 2 and in Fig. 3 are non-isomorphic,[8] the demand $\boldsymbol{d}_2$ accepts a different delivery solution[9] than that for demand $\boldsymbol{d}_1$. Such phenomenon does not happen in the standard coded caching scenario, where indeed each demand would result in the same

[7]Notice that we use "deliver" in quotes when referring to undesired subfiles because users do not need to actually decode subfiles which are not desired. Indeed, as explained above, it is more a matter of aligning interference, so that users can directly take linear combinations of the multicast messages to cancel out interference terms. Still, the interpretation related to the creation of cliques can give insights on how to construct multicast messages.

[8]This can be concluded by noticing that the graph in Fig. 2 contains 5 bidirectional edges, whereas the graph in Fig. 3 has 6 bidirectional edges.

[9]Having two non-isomorphic problems here implies that the delivery for the second problem cannot be derived from that of the first problem by a simple relabeling of the users.

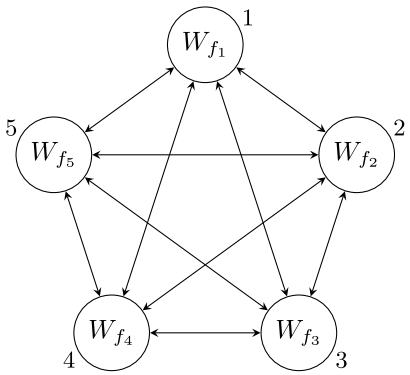


Fig. 4. FDS request graph for any demand in the standard (unselfish) MAN scenario with $K = 5$ users and $N = 5$ files labeled as $W_i$ with $i \in [5]$. In this case the demand is identified by the vector $\boldsymbol{f} = (f_1, f_2, f_3, f_4, f_5)$, where user $k \in [5]$ requests file $W_{f_k}$. This graph is complete. Hence, here the ability to create cliques of subfiles is only limited by $t$, and is not affected at all by the specific demand.

FDS request graph (cf. Fig. 4), which is always complete.[10] In such an unselfish scenario where each file is assumed to be of interest to all users, every user in the FDS request graph is connected to every other user, independently of the requested files. Hence, in the unselfish scenario, having a fixed FDS request graph for every demand allows for an identical delivery procedure for any demand. This seems to be a crucial differentiating aspect between selfish and unselfish coded caching.

## III. Converse Bound for Selfish Coded Caching With Uncoded Placement

Let us recall that each user is interested in its own FDS, and that each FDS only represents a fraction $|\mathcal{F}|/N$ of the library. In the general unselfish scenario, a portion $(1-|\mathcal{F}|/N)$ of each user's cache would be filled with content that would never be requested by that user. Such a non-selfish scheme would relinquish local caching gain for the benefit of being able to encode across all combinations of users. Under the basic clique-based approach in the MAN scheme, we are presented with a trade-off between local caching gain and coding gain, where the latter seems to be more desirable. Are there though other coding techniques that manage to harvest an abundance of coding opportunities, which are usually associated to the standard coded caching approach, exploiting the existence of a more targeted set of demands, while capitalizing on the increased local caching gain brought about by a selfish variant? If not, then what is the amount of coding gain that can be harvested while maintaining selfish caching? These are the questions addressed by our information-theoretic converse that lower bounds the optimal worst-case load assuming uncoded and selfish cache placement.

### A. Theorem Statement

The converse bound employs the index coding techniques of [25] that proved the optimality of the MAN scheme under the constraint of uncoded cache placement. Our main challenge will be to account for the presence of different profiles of interest, adapting consequently the index coding approach to reflect the $(K, \alpha, f)$ FDS structure proposed in the previous section. The converse bound presented here shows that adding the selfish cache placement constraint implies a higher optimal communication load compared to the unselfish scenario. The result is stated in the following theorem. We recall that $\gamma_\alpha = \gamma K/\alpha$ is the effective normalized cache size, and that $t = K\gamma = \alpha\gamma_\alpha$ is the cache redundancy. We also recall that $K\gamma + 1$ is the optimal coding gain for the unselfish scenario.

*Theorem 1 (Converse Bound for Selfish Coded Caching Under Uncoded Prefetching):* Under the assumption of uncoded and selfish cache placement, and given the $(K, \alpha, f)$ FDS structure, the optimal worst-case communication load $R^\star$ is lower bounded by $R_{\text{LB}}$ which is a piece-wise linear curve with corner points

$$(M, R_{\text{LB}}) = \left( t\frac{N}{K}, \frac{\binom{\alpha}{t+1} + (K-\alpha)\binom{\alpha-1}{t}}{\binom{\alpha}{t}} \right), \quad \forall t \in [0:\alpha] \tag{34}$$

corresponding to

$$R_{\text{LB}} = \frac{K(1-\gamma_\alpha)}{K\gamma+1}\Big[(K-\alpha)\gamma + 1\Big]. \tag{35}$$

*Proof:* We provide the proof of the converse in Section IV-A. In Section IV-B we also present an example that aims to help the reader better understand the construction of the outer bound. □

### B. Comments on the Converse Bound

The bound reveals some interesting insights. Before discussing these insights, let us quickly recall that, in our scenario, the integer value $t$ is upper bounded by $\alpha$, since any $t \geq \alpha$ would imply zero communication load.

*1) Comparison With MAN:* The following compares, for any $f$, the optimal load $R^\star(t)$ of selfish coded caching with that of the unselfish (MAN) scenario.[11]

*Corollary 1.1:* Given the symmetric $(K, \alpha, f)$ FDS structure and $\alpha \in [K-1]$, the converse reveals that

$$\frac{R^\star(t)}{R_{\text{MAN}}(t)} \geq 1, \quad \forall t \in [0:\alpha-1] \tag{36}$$

$$\frac{R^\star(t)}{R_{\text{MAN}}(t)} > 1, \quad \forall t \in (0:\alpha-1) \tag{37}$$

which says that, in the non-trivial range $t \in [0:\alpha-1]$, selfish coded caching is not better than unselfish coded caching, which instead, in the non-extremal points of $t$ and under uncoded placement, strictly outperforms any implementation of selfish coded caching. When $\alpha = K$ and $f \geq K$, the converse expression naturally matches that of unselfish coded caching.

*Proof:* The proof can be found in Appendix A, while a graphical comparison can be found in Fig. 5. □

[10] A complete graph is a graph where every node is connected to every other node.

[11] The comparison between the selfish and unselfish scenarios is made easy by the fact that the $t$ values (i.e., the integer points corresponding to the memory-axis of the memory-load trade-off) in the two scenarios coincide.

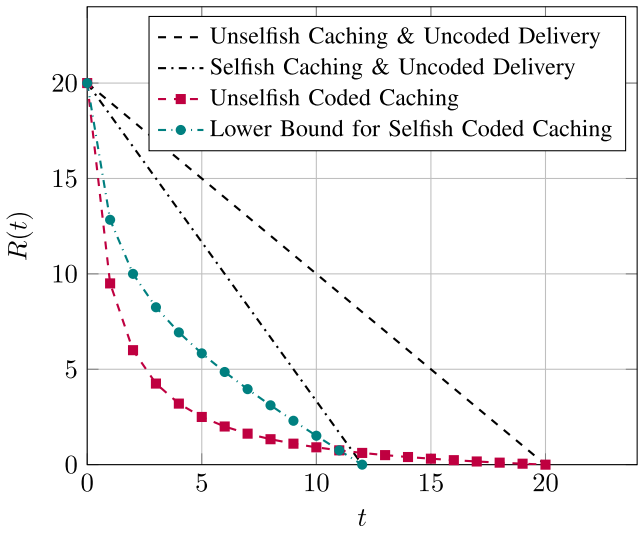


Fig. 5. Comparison between selfish and unselfish caching for the $(20, 12, f)$ FDS structure.

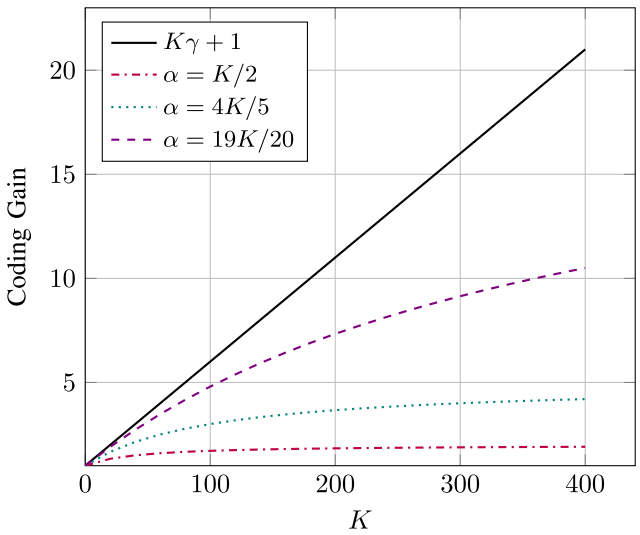


Fig. 6. Plot of different coding gains $G$ for varying values of $K$ and $\alpha$ for the $(K, \alpha, f)$ FDS structure when $\gamma = 1/20$.

*2) Selfish Local Caching Gain and Coding Gain:* We recall that, in the presence of a relatively small $\alpha$, selfish caching implies a sizeable increase in the effective normalized cache size $\gamma_\alpha = \gamma K/\alpha$, which in turn implies a much larger local caching gain.

On the other hand, the converse reveals that a smaller $\alpha$ implies a substantial reduction in the coding gain offered by selfish coded caching. To compare coding gains, we first recall that the coding gain in the original unselfish scenario takes the form

$$\frac{R_{\text{U}}}{R_{\text{MAN}}} = K\gamma + 1 \tag{38}$$

where $R_{\text{U}} = K(1-\gamma)$ is the load for uncoded delivery. As previously stated, this coding gain $K\gamma + 1$ describes the speedup factor over the uncoded case. To reflect this same speedup in the selfish scenario, we must consider that the corresponding load in the uncoded scenario takes the form $R_{\text{U,selfish}} = K(1-\gamma_\alpha)$. With this in place, the converse reveals that the optimal coding gain of selfish coded caching is upper bounded as

$$G^\star \leq \frac{R_{\text{U,selfish}}}{R_{\text{LB}}} = \frac{K\gamma + 1}{(K-\alpha)\gamma + 1} \tag{39}$$

where the value

$$D := (K-\alpha)\gamma + 1 \tag{40}$$

represents the guaranteed deterioration in the coding gain when we choose to cache selfishly. Indeed, if we consider the non-trivial range $\alpha \in [2 : K-1]$, we have $D > 1$ and consequently $G < K\gamma + 1$ for $\gamma > 0$. We can see that — for fixed $K$ and $\gamma$ — this deterioration $D$ increases with decreasing $\alpha$, reflecting the fact that the closer the $(K, \alpha, f)$ FDS structure is to the standard MAN scenario, the smaller this deterioration $D$ is.

An important observation though is that the coding gain of selfish coded caching does not scale with $K$. This is described in the following corollary.

*Corollary 1.2:* For any fixed ratio $\delta = \alpha/K < 1$ the coding gain of selfish caching does not scale as $K$ increases, and it is instead bounded as

$$G^\star < \frac{1}{1-\delta}. \tag{41}$$

*Proof:* The proof can be found in Appendix B. □

We can see in Fig. 6 the comparison between different coding gains for varying values of $K$ and $\alpha$ when the normalized cache size $\gamma$ is fixed. As mentioned, smaller values of $\alpha$ correspond to much smaller coding gains. As stated in Corollary 1.2, each curve is upper bounded[12] by $1/(1-\delta)$.

*Remark 3:* At this point, we ought to point out that our choice of having a fully symmetric FDS structure may indeed be an overly penalizing condition. However, this choice exemplifies the mechanisms and effects that come about when selfishness is considered. This same choice nicely offers a crisp method for calibrating the intersection between the interests of the different users, taking us from a scenario where the intersection is minimal, to scenarios ever closer to the original MAN setting where the interests are identical. As suggested before, this FDS structure is a sufficient proving step for drawing, in a unified manner, the conclusion that indeed selfish coded caching *can* be quite detrimental.

## IV. Proof of the Main Information-Theoretic Converse in Theorem 1

The derivation of the converse makes extensive use of the connection between caching and index coding. This connection was made in [10] and was successfully used in [25] to derive the optimal performance of the unselfish scenario.

We quickly recall that an index coding problem [53]–[56] consists of a server wishing to deliver $N'$ independent messages to $K'$ users via a basic bottleneck link.

[12]When $\alpha = 1$ it naturally holds that $G^\star = 1$, since in such case uncoded delivery is optimal.

Each user $k \in [K']$ has its own *desired message set* $\mathcal{M}_k \subseteq [N']$, and has knowledge of its own *side information set* $\mathcal{A}_k \subseteq [N']$. Let $M_i$ be the message $i$ in the set $[N']$. Then, the index coding problem is typically described by its *side information graph* in the form of a directed graph, where each vertex is a message and where there is an edge from $M_i$ to $M_j$ if $M_i$ is in the side information set of the user requesting $M_j$. The derivation of our converse will use the following well-known result from [57, Corollary 1].

*Lemma 1 ( [57, Corollary 1]):* In an index coding problem with $N'$ messages $M_i$ for $i \in [N']$, the minimum number of transmitted bits $\rho$ is bounded as

$$\rho \geq \sum_{i \in \mathcal{J}} |M_i| \tag{42}$$

for any acyclic subgraph $\mathcal{J}$ of the problem's side information graph.

Before proceeding with the main proof, we also recall that under the $(K, \alpha, f)$ FDS structure we have $\mathcal{L} = \{\mathcal{W}_{\mathcal{S}} : \mathcal{S} \subseteq [K], |\mathcal{S}| = \alpha\}$, where $\mathcal{W}_{\mathcal{S}} = \{W_{i,\mathcal{S}} : i \in [f]\}$ is a class of files. We further recall that there are $C = \binom{K}{\alpha}$ classes of files and $N = fC = f\binom{K}{\alpha}$ files. Additionally, we recall that the FDS of each user $k \in [K]$ is given by

$$\mathcal{F}_k = \{\mathcal{W}_S : \mathcal{S} \subseteq [K], |\mathcal{S}| = \alpha, k \in \mathcal{S}\} \tag{43}$$

that each file has size $B$ bits, and that each user is equipped with a cache of size $MB$ bits. Finally, let us remember that we are interested in the non-trivial range[13] $\alpha \in [2 : K-1]$ and in the range $M \in \left[0 : f\binom{K-1}{\alpha-1}\right]$ simply because having $M = |\mathcal{F}_k| = f\binom{K-1}{\alpha-1}$ implies $R^\star(|\mathcal{F}|) = 0$ as a consequence of each user being able to store the entirety of its FDS.

### A. Main Proof

The first step toward the converse consists of splitting each file in a generic manner into a maximum of $2^{|\mathcal{S}|} = 2^\alpha$ disjoint subfiles as

$$W_{i,\mathcal{S}} = \{W_{i,\mathcal{S},\mathcal{T}} : \mathcal{T} \subseteq \mathcal{S}\},\ \forall \mathcal{S} \subseteq [K] : |\mathcal{S}| = \alpha,\ \forall i \in [f] \tag{44}$$

where $W_{i,\mathcal{S},\mathcal{T}}$ is the subfile of $W_{i,\mathcal{S}}$ cached exactly and only by users in $\mathcal{T}$. As already mentioned in Defintion 2, splitting each file in this way satisfies the selfish cache placement constraint, since $\mathcal{T} \subseteq \mathcal{S}$ and $\mathcal{W}_{\mathcal{S}} \in \mathcal{F}_k$ for each $k \in \mathcal{S}$.

*1) Constructing the Index Coding Problem:* We now proceed by making the connection to index coding. Hence, let us consider the index coding problem with $K' = K$ users and $N' = K2^{\alpha-1}$ messages, such that for any demand, identified by the vectors $\boldsymbol{d} = (\mathcal{D}_1, \ldots, \mathcal{D}_K)$ and $\boldsymbol{f} = (f_1, \ldots, f_K)$, the desired message set and the side information set are respectively given by

$$\mathcal{M}_k = \{W_{f_k,\mathcal{D}_k,\mathcal{T}} : \mathcal{T} \subseteq \mathcal{D}_k, k \notin \mathcal{T}\} \tag{45}$$

$$\mathcal{A}_k = \{W_{i,\mathcal{S},\mathcal{T}} : i \in [f], \mathcal{S} \subseteq [K], |\mathcal{S}| = \alpha, \mathcal{T} \subseteq \mathcal{S}, k \in \mathcal{T}\} \tag{46}$$

for each user $k \in [K]$. For this setting the side information graph takes the form of a directed graph where each subfile represents a vertex, and where there is a connection from (the node corresponding to) $W_{f_{k_1},\mathcal{D}_{k_1},\mathcal{T}_1}$ to $W_{f_{k_2},\mathcal{D}_{k_2},\mathcal{T}_2}$ if and only if $W_{f_{k_1},\mathcal{D}_{k_1},\mathcal{T}_1} \in \mathcal{A}_{k_2}$, i.e., if and only if $k_2 \in \mathcal{T}_1$. To apply Theorem 1, we are interested in acyclic sets of vertices $\mathcal{J}$ in such side information graph. In the spirit of [25], we know that the set

$$\bigcup_{k \in [K]} \bigcup_{\mathcal{T} \subseteq (([K] \setminus \{u_1, \ldots, u_k\}) \cap \mathcal{D}_{u_k})} \left\{W_{f_{u_k},\mathcal{D}_{u_k},\mathcal{T}}\right\} \tag{47}$$

does not contain any directed cycle[14] for any demand $(\boldsymbol{d}, \boldsymbol{f})$ and any vector $\boldsymbol{u}$, where $\boldsymbol{u} = (u_1, \ldots, u_K)$ is a permutation of the users in $[K]$. Consequently, applying Theorem 1 yields the following lower bound

$$BR^\star \geq \sum_{k \in [K]} \sum_{\mathcal{T} \subseteq (([K] \setminus \{u_1, \ldots, u_k\}) \cap \mathcal{D}_{u_k})} \left|W_{f_{u_k},\mathcal{D}_{u_k},\mathcal{T}}\right|. \tag{48}$$

*2) Selection of Distinct Demands:* Now we wish to create several lower bounds as the one in (48) considering different user permutations $\boldsymbol{u}$, and considering a subset of user demands — each determined by the tuple $(\boldsymbol{d}, \boldsymbol{f})$ with $\boldsymbol{d} = (\mathcal{D}_1, \ldots, \mathcal{D}_K)$ and $\boldsymbol{f} = (f_1, \ldots, f_K)$. Our aim is to eventually average these bounds in order to obtain an analytically tractable lower bound on the optimal communication load. For $\mathcal{C}$ being the set of properly selected demands described further below and $\mathcal{U}_{(\boldsymbol{d},\boldsymbol{f})}$ being the set of selected user permutations for each demand $(\boldsymbol{d}, \boldsymbol{f})$ in $\mathcal{C}$, we seek to characterize the expression given by

$$BR^\star \sum_{(\boldsymbol{d},\boldsymbol{f}) \in \mathcal{C}} |\mathcal{U}_{(\boldsymbol{d},\boldsymbol{f})}| \geq \sum_{(\boldsymbol{d},\boldsymbol{f}) \in \mathcal{C}} \sum_{\boldsymbol{u} \in \mathcal{U}_{(\boldsymbol{d},\boldsymbol{f})}} \sum_{k \in [K]} \sum_{\mathcal{T} \subseteq (([K] \setminus \{u_1, \ldots, u_k\}) \cap \mathcal{D}_{u_k})} \left|W_{f_{u_k},\mathcal{D}_{u_k},\mathcal{T}}\right|. \tag{49}$$

Notice that the goal of carefully selecting the demand set $\mathcal{C}$ and the permutation set $\mathcal{U}_{(\boldsymbol{d},\boldsymbol{f})}$ is twofold. The first is to provide the symmetry that will allow us to simplify (49) into a meaningful expression, and the second is to force the bound to be as tight as possible.[15]

Toward this, we proceed to select $\mathcal{C}$ to contain circular demands, as these are defined as follows.

*Definition 3 (Circular Demands):* A demand defined by the vectors $\boldsymbol{d} = (\mathcal{D}_1, \ldots, \mathcal{D}_K)$ and $\boldsymbol{f} = (f_1, \ldots, f_K)$ is said to be a *circular demand* if there exists a permutation

[13] When $\alpha = 1$ the proof is trivial, since for such case we have only two integer points corresponding to $t \in \{0, 1\}$: when $t = 0$ the load is equal to $K$, and when $t = 1$ each user has enough memory to cache entirely its own FDS and the load is equal to 0. Then, the case $\alpha = K$ and $f \geq K$ is equivalent to the standard (unselfish) MAN scenario, which was already considered in [25].

[14] Notice that [25, Lemma 1] considers in fact $\mathcal{T} \subseteq ([K] \setminus \{u_1, \ldots, u_k\})$ and not $\mathcal{T} \subseteq (([K] \setminus \{u_1, \ldots, u_k\}) \cap \mathcal{D}_{u_k})$. However, the latter is a subset of the former, and thus the lemma still holds.

[15] We wish to stress that constructing the converse using the demand set $\mathcal{C}$ and the permutation set $\mathcal{U}_{(\boldsymbol{d},\boldsymbol{f})}$ for each $(\boldsymbol{d}, \boldsymbol{f}) \in \mathcal{C}$ does not mean that the converse is the tightest nor that the demands $\mathcal{C}$ are part of the worst-case demand set.

$\hat{\boldsymbol{u}} = (\hat{u}_1, \ldots, \hat{u}_K)$ of the set of users $[K]$ such that[16]

$$W_{f_{\hat{u}_k}, \mathcal{D}_{\hat{u}_k}} \in \bigcap_{i=k+1}^{k+\alpha-1} \{\mathcal{F}_{\hat{u}_i}\} \tag{50}$$

for each $k \in [K]$. This simply means that the demand reflects a circular pattern if $\cup_{i=k+1}^{k+\alpha-1}\{\hat{u}_i\} = \mathcal{D}_{\hat{u}_k} \setminus \{\hat{u}_k\}$ for each $k \in [K]$.

*Example 4:* Consider the $(6, 4, f)$ FDS structure. An example of circular demand is given by the demand with vectors $\boldsymbol{d} = (1234, 2345, 3456, 1456, 1256, 1236)$ and $\boldsymbol{f} = (f_1, f_2, f_3, f_4, f_5, f_6)$. Indeed, the condition $\cup_{i=k+1}^{k+\alpha-1}\{\hat{u}_i\} = \mathcal{D}_{\hat{u}_k} \setminus \{\hat{u}_k\}$ is satisfied for each $k \in [6]$ with the vector $\hat{\boldsymbol{u}} = (1, 2, 3, 4, 5, 6)$. This same condition is also satisfied by all $K$ circular shifts of $\hat{\boldsymbol{u}} = (1, 2, 3, 4, 5, 6)$.

This new definition allows us to describe the sets $\mathcal{C}$ and $\mathcal{U}_{(\boldsymbol{d},\boldsymbol{f})}$ as

$$\mathcal{C} \coloneqq \{(\boldsymbol{d}, \boldsymbol{f}) : \text{the demand } (\boldsymbol{d}, \boldsymbol{f}) \text{ is circular}\} \tag{51}$$

$$\mathcal{U}_{(\boldsymbol{d},\boldsymbol{f})} \coloneqq \{K \text{ circular shifts of the vector } \hat{\boldsymbol{u}} \text{ associated to a given } (\boldsymbol{d}, \boldsymbol{f}) \in \mathcal{C}\}. \tag{52}$$

These sets will generally yield larger acyclic subgraphs[17] in (47) that can be used to increase the RHS in (48), and to provide a better lower bound on $R^\star$.

*Remark 4:* We wish to point out that the careful selection of the demand set $\mathcal{C}$ and permutation set $\mathcal{U}_{(\boldsymbol{d},\boldsymbol{f})}$ for each $(\boldsymbol{d}, \boldsymbol{f}) \in \mathcal{C}$ is a pivotal difference with respect to the index coding based approaches that followed [25]. Indeed, if one develops a converse bound using all possible valid demands and user permutations following the exact same procedure in [25], then the resulting bound is looser than the one presented here. As a consequence, finding the good set of demands and permutations becomes the key to obtaining a tighter bound while keeping the problem analytically tractable.

*3) Counting the Selected Demands:* Our goal now is to simplify (49) into a more meaningful expression. We start by counting how many circular demands there are. To do so, we observe that there is a one-to-one correspondence between one circular demand $(\boldsymbol{d}, \boldsymbol{f}) \in \mathcal{C}$ and the corresponding set $\mathcal{U}_{(\boldsymbol{d},\boldsymbol{f})}$ of permutations of users. This is easy to see, and the intuition is as follows. A demand is said to be circular if there exists a particular ordering of users such that the property in (50) is satisfied. Such ordering is described by the vector $\hat{\boldsymbol{u}}$ and is clearly preserved under any circular shift of such vector. Consequently evaluating $|\mathcal{C}|$ is equivalent to counting the vectors $\hat{\boldsymbol{u}}$, since each of them corresponds to a distinct circular demand.

Let us focus on user $k \in [K]$. Counting the total number of circular demands where user $k$ requests the file $W_{f_k, \mathcal{D}_k}$ is equivalent to counting the total number of vectors $\hat{\boldsymbol{u}}$ such that $\cup_{i=k+1}^{k+\alpha-1}\{\hat{u}_i\} = \mathcal{D}_k \setminus \{k\}$ and $\cup_{i=k+\alpha}^{K+k-1}\{\hat{u}_i\} = [K] \setminus \mathcal{D}_k$. Recalling that we imply $i \bmod K$ whenever $i > K$, we see that there are $(\alpha-1)!(K-\alpha)!f^{K-1}$ such vectors. Then, if we recall that user $k$ can request a total of $f\binom{K-1}{\alpha-1}$ files, we see that the total number of circular demands is equal to

$$|\mathcal{C}| = f\binom{K-1}{\alpha-1}(\alpha-1)!(K-\alpha)!f^{K-1} = f^K(K-1)!. \tag{53}$$

Furthermore, since $|\mathcal{U}_{(\boldsymbol{d},\boldsymbol{f})}| = K$ for each circular demand, we can see that there is a total of $\sum_{(\boldsymbol{d},\boldsymbol{f})\in\mathcal{C}} |\mathcal{U}_{(\boldsymbol{d},\boldsymbol{f})}| = f^K K!$ lower bounds — as the one in (48) — created for the expression in (49).

*4) Constructing the Optimization Problem:* We will seek to simplify the expression in (49), and then to minimize the new simplified expression, in order to lower bound the optimal worst-case load $R^\star$. Toward simplifying, we first count how many times each subfile $W_{i,\mathcal{S},\mathcal{T}}$ appears in (49), where $i \in [f]$, $\mathcal{S} \subseteq [K]$ with $|\mathcal{S}| = \alpha$, $\mathcal{T} \subseteq \mathcal{S}$ and $|\mathcal{T}| = t$. For this purpose, we make use of the following lemma.

*Lemma 2:* Let $\hat{\boldsymbol{u}} = (\hat{u}_1, \ldots, \hat{u}_K)$ be a permutation of the elements in $[K]$, let $\mathcal{U}$ be the set composed of the $K$ circular shifts of the vector $\hat{\boldsymbol{u}}$, and let $k_1, k_2 \in [K]$ such that $k_1 \neq k_2$. Consider

$$\ell \coloneqq \left| \bigcup_{i=\hat{u}(k_1)+1}^{\hat{u}(k_2)} \{\hat{u}_i\} \right| \tag{54}$$

where we assume $i \bmod K$ whenever $i > K$. Then, there is a total of $(K - \ell)$ vectors $\boldsymbol{u} \in \mathcal{U}$ such that $k_1$ appears before $k_2$ in the vector $\boldsymbol{u}$.

*Proof:* The proof is reported in Appendix C. □

Let us focus on subfile $W_{i,\mathcal{S},\mathcal{T}}$. We start by considering all circular demands $(\boldsymbol{d}, \boldsymbol{f}) \in \mathcal{C}$ such that $f_{k_1} = i$ and $\mathcal{D}_{k_1} = \mathcal{S}$ for some $k_1 \in \mathcal{S}$ and $\mathcal{T} \subseteq (\mathcal{S} \setminus \{k_1\})$ with $|\mathcal{T}| = t$. For each of these circular demands we have a vector $\hat{\boldsymbol{u}}$ of ordered users and we select as user permutations the vectors in the set $\mathcal{U}_{(\boldsymbol{d},\boldsymbol{f})}$. Considering how the acyclic set of vertices in (47) is built, it is clear that $W_{i,\mathcal{S},\mathcal{T}}$ appears in (49) whenever *all* the elements in $\mathcal{T}$ appear after $k_1$ in $\boldsymbol{u} \in \mathcal{U}_{(\boldsymbol{d},\boldsymbol{f})}$. By Lemma 2, we know that this happens a total of $(K - \ell)$ times, where

$$\ell = \max_{j\in\mathcal{T}} \left| \bigcup_{i=\hat{u}(k_1)+1}^{\hat{u}(j)} \{\hat{u}_i\} \right|. \tag{55}$$

Notice that the above maximization is required since our aim is to count — for any given $\hat{\boldsymbol{u}}$, and thus for a given circular demand — how many times *all* the elements in $\mathcal{T}$ appear after[18] $k_1$ when considering all the user permutations in $\mathcal{U}_{(\boldsymbol{d},\boldsymbol{f})}$.

Recalling that $|\mathcal{T}| = t$, we observe that $\ell \in [t : \alpha - 1]$. To see this, we note that the minimum value of $\ell$ is $t$ when $k_1$ and all the elements in $\mathcal{T}$ are in consecutive positions in the vector $\hat{\boldsymbol{u}}$. Additionally, we also note that the maximum value of $\ell$ is $(\alpha - 1)$, because $\mathcal{T} \subseteq (\mathcal{S} \setminus \{k_1\})$ and all the $(\alpha - 1)$ elements in $\mathcal{S} \setminus \{k_1\}$ are immediately after $k_1$ in $\hat{\boldsymbol{u}}$ (and any of its circular shifts in $\mathcal{U}_{(\boldsymbol{d},\boldsymbol{f})}$). This is because we are considering circular demands where user $k_1$ requests for the file class $\mathcal{D}_{k_1} = \mathcal{S}$. Hence, when we consider all possible values of $\ell$,

[16] We imply $i \bmod K$ whenever $i > K$.

[17] This is based on the following observation. Throughout various examples, such demands generally yielded the largest bounds compared to other classes of demands.

[18] Indeed, recalling that we build acyclic subgraphs as in (47), the subfile $W_{f_{k_1}, \mathcal{D}_{k_1}, \mathcal{T}}$ appears in (49) only when *all* elements in $\mathcal{T}$ are after $k_1$ in the vector $\boldsymbol{u} \in \mathcal{U}_{(\boldsymbol{d},\boldsymbol{f})}$ and the maximization is needed to count in how many of such user permutations it happens to have *all* the elements in $\mathcal{T}$ after $k_1$.

the subfile $W_{i,\mathcal{S},\mathcal{T}}$ is counted a total of $\sum_{\ell=t}^{\alpha-1} a_\ell(K-\ell)$ times in (49) when we focus on circular demands with $f_{k_1}=i$ and $\mathcal{D}_{k_1}=\mathcal{S}$. The term

$$a_\ell \coloneqq t!(\alpha-1-t)!(K-\alpha)!\binom{\ell-1}{t-1}f^{K-1} \tag{56}$$

counts the total number of vectors $\hat{\boldsymbol{u}}$ (and consequently of circular demands) for which $f_{k_1}=i$, $\mathcal{D}_{k_1}=\mathcal{S}$ and $\max_{j\in\mathcal{T}}\left|\bigcup_{i=\hat{u}(k_1)+1}^{\hat{u}(j)}\{\hat{u}_i\}\right|=\ell$. Since the same reasoning applies whenever the file $W_{i,\mathcal{S},\mathcal{T}}$ is requested by any of the other $(\alpha-1-t)$ users in $\mathcal{S}\setminus\{\mathcal{T},k_1\}$, namely, when $f_k=i$ and $\mathcal{D}_k$ for every $k\in(\mathcal{S}\setminus\{\mathcal{T},k_1\})$, we can conclude that the subfile $W_{i,\mathcal{S},\mathcal{T}}$ appears a total of $(\alpha-t)\sum_{\ell=t}^{\alpha-1}a_\ell(K-\ell)$ times in (49) when we consider all circular demands in $\mathcal{C}$. Moreover, the same reasoning applies to any other subfile. Hence, the expression in (49) simplifies as

$$\begin{aligned}
R^\star &\geq \frac{1}{B\sum_{(\boldsymbol{d},\boldsymbol{f})\in\mathcal{C}}|\mathcal{U}_{(\boldsymbol{d},\boldsymbol{f})}|}\sum_{(\boldsymbol{d},\boldsymbol{f})\in\mathcal{C}}\sum_{\boldsymbol{u}\in\mathcal{U}_{(\boldsymbol{d},\boldsymbol{f})}}\sum_{k\in[K]}\\
&\qquad\sum_{\mathcal{T}\subseteq(([K]\setminus\{u_1,\ldots,u_k\})\cap\mathcal{D}_{u_k})}\left|W_{f_{u_k},\mathcal{D}_{u_k},\mathcal{T}}\right| &(57)\\
&= \frac{1}{Bf^KK!}\sum_{(\boldsymbol{d},\boldsymbol{f})\in\mathcal{C}}\sum_{\boldsymbol{u}\in\mathcal{U}_{(\boldsymbol{d},\boldsymbol{f})}}\sum_{k\in[K]}\\
&\qquad\sum_{\mathcal{T}\subseteq(([K]\setminus\{u_1,\ldots,u_k\})\cap\mathcal{D}_{u_k})}\left|W_{f_{u_k},\mathcal{D}_{u_k},\mathcal{T}}\right| &(58)\\
&= \sum_{t=0}^{\alpha} f(t)x_t &(59)
\end{aligned}$$

where we define

$$c_t \coloneqq \frac{(\alpha-t)}{f^KK!}\sum_{\ell=t}^{\alpha-1}a_\ell(K-\ell) \tag{60}$$

$$f(t) \coloneqq Nc_t \tag{61}$$

$$0\leq x_t \coloneqq \sum_{\mathcal{S}\subseteq[K]:|\mathcal{S}|=\alpha}\sum_{\mathcal{T}\subseteq\mathcal{S}:|\mathcal{T}|=t}\sum_{i\in[f]}\frac{|W_{i,\mathcal{S},\mathcal{T}}|}{NB}. \tag{62}$$

At this point, we seek to lower bound the minimum worst-case load $R^\star(t)$ by lower bounding the solution to the following optimization problem

$$\begin{aligned}
\min_{\boldsymbol{x}} \quad & \sum_{t=0}^{\alpha} f(t)x_t &(63a)\\
\text{subject to} \quad & \sum_{t=0}^{\alpha} x_t = 1 &(63b)\\
& \sum_{t=0}^{\alpha} tx_t \leq \frac{KM}{N} &(63c)
\end{aligned}$$

where (63b) and (63c) correspond to the file size constraint and the cumulative cache size constraint, respectively.

*5) Simplifying and Lower Bounding the Solution to the Optimization Problem:* Before proceeding to solve the optimization problem, we wish to further simplify the coefficients $c_t$ and thus also $f(t)$. Indeed, this coefficient $c_t$ can be rewritten as

$$c_t = \frac{1}{N\binom{\alpha}{t}}\sum_{\ell=t}^{\alpha-1}\binom{\ell-1}{t-1}(K-\ell) \tag{64}$$

recalling that $N=f\binom{K}{\alpha}$. Then, we can write

$$\begin{aligned}
c_t &= \frac{1}{N\binom{\alpha}{t}}\sum_{\ell=t}^{\alpha-1}\binom{\ell-1}{t-1}(K-\ell) &(65)\\
&= \frac{1}{N\binom{\alpha}{t}}\left(K\sum_{\ell=t}^{\alpha-1}\binom{\ell-1}{t-1}-\sum_{\ell=t}^{\alpha-1}\ell\binom{\ell-1}{t-1}\right) &(66)\\
&= \frac{1}{N\binom{\alpha}{t}}\left(K\sum_{\ell=t-1}^{\alpha-2}\binom{\ell}{t-1}-t\sum_{\ell=t}^{\alpha-1}\binom{\ell}{t}\right) &(67)\\
&= \frac{K\binom{\alpha-1}{t}-t\binom{\alpha}{t+1}}{N\binom{\alpha}{t}} &(68)\\
&= \frac{\binom{\alpha}{t+1}+(K-\alpha)\binom{\alpha-1}{t}}{N\binom{\alpha}{t}} &(69)
\end{aligned}$$

where (68) uses the well-known hockey-stick identity which states that

$$\sum_{i=k}^{n}\binom{i}{k}=\binom{n+1}{k+1},\quad \forall n,k\in\mathbb{N},\quad n\geq k. \tag{70}$$

At this point, we can rewrite $f(t)$ as

$$f(t)=\frac{\binom{\alpha}{t+1}+(K-\alpha)\binom{\alpha-1}{t}}{\binom{\alpha}{t}}. \tag{71}$$

Now, since the auxiliary variable $x_t$ can be considered as a probability mass function, the optimization problem in (63) can be seen as the minimization of $\mathbb{E}[f(t)]$. Moreover, the following holds.

*Lemma 3:* The function $f(t)$ is convex and is strictly decreasing for increasing values of $t$.

*Proof:* The proof is reported in Appendix D. □

Taking advantage of Lemma 3, we can write $\mathbb{E}[f(t)]\geq f(\mathbb{E}[t])$ using Jensen's inequality. Then, since $f(t)$ is strictly decreasing with increasing $t\in[0:\alpha]$, we can further write $f(\mathbb{E}[t])\geq f(KM/N)$ taking advantage of the fact that $\mathbb{E}[t]$ is upper bounded as in (63c). Consequently, $\mathbb{E}[f(t)]\geq f(KM/N)$, and thus for $t=KM/N$ the converse bound is a piece-wise linear curve with corner points

$$(M,R_{\text{LB}})=\left(t\frac{N}{K},\frac{\binom{\alpha}{t+1}+(K-\alpha)\binom{\alpha-1}{t}}{\binom{\alpha}{t}}\right),\quad\forall t\in[0:\alpha]. \tag{72}$$

Thus, $R_{\text{LB}}$ takes the form

$$\begin{aligned}
R_{\text{LB}} &= \frac{\binom{\alpha}{t+1}+(K-\alpha)\binom{\alpha-1}{t}}{\binom{\alpha}{t}} &(73)\\
&= \frac{\alpha-t}{1+t}+(K-\alpha)\left(1-\frac{t}{\alpha}\right) &(74)\\
&= \frac{\alpha(1-\gamma_\alpha)}{1+K\gamma}+K(1-\gamma_\alpha)-\alpha(1-\gamma_\alpha) &(75)\\
&= K(1-\gamma_\alpha)\left[1+\frac{\alpha}{K(1+K\gamma)}-\frac{\alpha}{K}\right] &(76)\\
&= \frac{K(1-\gamma_\alpha)}{1+K\gamma}\left[1+K\gamma+\frac{\alpha}{K}-\frac{\alpha}{K}(1+K\gamma)\right] &(77)\\
&= \frac{K(1-\gamma_\alpha)}{1+K\gamma}\left[(K-\alpha)\gamma+1\right] &(78)
\end{aligned}$$

which completes the proof. □

### B. A Detailed Example for the Converse Bound

We present here in detail an example that can help the reader better understand the construction of the converse bound.

Let us consider the $(6, 4, 1)$ FDS structure which involves a file library $\mathcal{L} = \{W_{1,\mathcal{S}} : \mathcal{S} \subseteq [6], |\mathcal{S}| = 4\}$ consisting of $C = \binom{K}{\alpha} = \binom{6}{4} = 15$ classes of files. Since there is only $f = 1$ file per class, there is a total of $N = fC = 15$ files, so in this example we make no distinction between files and classes of files. Thus, for simplicity, we will here refer to file $W_{1,\mathcal{S}}$ directly as $W_{\mathcal{S}}$, which means that each file is entirely described by a 4-tuple $\mathcal{S} \subseteq [6]$, and each demand instance is entirely defined by the $\boldsymbol{d} = (\mathcal{D}_1, \ldots, \mathcal{D}_K)$ vector only. The FDS of each user $k \in [6]$ is given by

$$\mathcal{F}_k = \{W_S : \mathcal{S} \subseteq [6], |\mathcal{S}| = 4, k \in \mathcal{S}\} \tag{79}$$

and it consists of $f\binom{K-1}{\alpha-1} = 10$ files. Hence, this example considers $M \in [0 : 10]$, simply because having $M \geq 10$ implies that each user can preemptively cache the entirety of its FDS, which in turn implies $R^\star(|\mathcal{F}|) = 0$.

We start by assuming the most general uncoded and selfish cache placement where each file $W_{\mathcal{S}}$ is split into a total of $2^{|\mathcal{S}|} = 2^\alpha = 16$ disjoint subfiles as

$$W_{\mathcal{S}} = \{W_{\mathcal{S},\mathcal{T}} : \mathcal{T} \subseteq \mathcal{S}\}, \quad \forall \mathcal{S} \subseteq [6] : |\mathcal{S}| = 4 \tag{80}$$

where we recall that $W_{\mathcal{S},\mathcal{T}}$ is the subfile of $W_{\mathcal{S}}$ cached by the users in $\mathcal{T}$, and where we recall that the selfishness condition is guaranteed by forcing $\mathcal{T} \subseteq \mathcal{S}$.

*1) Constructing the Index Coding Problem:* For any given demand $\boldsymbol{d} = (\mathcal{D}_1, \ldots, \mathcal{D}_K)$ where user $k$ asks for $W_{\mathcal{D}_k}$, we consider the index coding problem with $K' = K = 6$ users and $N' = K2^{\alpha-1} = 48$ messages, where each user $k \in [6]$ has a desired message set

$$\mathcal{M}_k = \{W_{\mathcal{D}_k,\mathcal{T}} : \mathcal{T} \subseteq \mathcal{D}_k, k \notin \mathcal{T}\} \tag{81}$$

and a side information set

$$\mathcal{A}_k = \{W_{\mathcal{S},\mathcal{T}} : \mathcal{S} \subseteq [K], |\mathcal{S}| = \alpha, \mathcal{T} \subseteq \mathcal{S}, k \in \mathcal{T}\}. \tag{82}$$

The side information graph of this index coding problem is the directed graph where each desired subfile represents a graph vertex and where a connection exists from vertex $W_{\mathcal{D}_{k_1},\mathcal{T}_1}$ to $W_{\mathcal{D}_{k_2},\mathcal{T}_2}$ if and only if $W_{\mathcal{D}_{k_1},\mathcal{T}_1} \in \mathcal{A}_{k_2}$. To now create an acyclic subgraph of the above graph, we recall from [25, Lemma 1] that the set

$$\bigcup_{k\in[K]} \bigcup_{\mathcal{T}\subseteq(([K]\setminus\{u_1,\ldots,u_k\})\cap\mathcal{D}_{u_k})} \left\{W_{\mathcal{D}_{u_k},\mathcal{T}}\right\} \tag{83}$$

does not contain any directed cycle for any demand $\boldsymbol{d}$ and user permutation $\boldsymbol{u}$. This, together with Theorem 1, implies that

$$BR^\star \geq \sum_{k\in[K]} \sum_{\mathcal{T}\subseteq(([K]\setminus\{u_1,\ldots,u_k\})\cap\mathcal{D}_{u_k})} \left|W_{\mathcal{D}_{u_k},\mathcal{T}}\right|. \tag{84}$$

*2) Selection of Distinct Demands:* To render the above bound meaningful, we need to carefully select a set of demands that will eventually help us symmetrize the problem as well as render the bound tighter. Toward this, we create several lower bounds as the one in (84), one for each chosen demand and user permutation. Our desired symmetry is achieved by considering only the set of circular demands $\mathcal{C}$ and, for each $\boldsymbol{d} \in \mathcal{C}$, the user permutations in $\mathcal{U}_{\boldsymbol{d}}$, where this last set simply contains the $K$ circular shifts of the vector $\hat{\boldsymbol{u}}$ associated to each circular demand. For example, in our $(6, 4, 1)$ FDS scenario, one such circular demand is $\boldsymbol{d} = (1234, 2345, 3456, 1456, 1256, 1236)$, because it satisfies the condition $\cup_{i=k+1}^{k+\alpha-1}\{\hat{u}_i\} = \mathcal{D}_{\hat{u}_k} \setminus \{\hat{u}_k\}$ for each $k \in [6]$ with the vector $\hat{\boldsymbol{u}} = (1, 2, 3, 4, 5, 6)$. This same condition is also satisfied by all $K$ circular shifts of $\hat{\boldsymbol{u}} = (1, 2, 3, 4, 5, 6)$. For this demand $\boldsymbol{d} = (1234, 2345, 3456, 1456, 1256, 1236)$, we will consequently construct 6 bounds as in (84).

By averaging all such bounds, a new lower bound on $R^\star$ appears in the following form

$$BR^\star \sum_{\boldsymbol{d}\in\mathcal{C}} |\mathcal{U}_{\boldsymbol{d}}| \geq \sum_{\boldsymbol{d}\in\mathcal{C}} \sum_{\boldsymbol{u}\in\mathcal{U}_{\boldsymbol{d}}} \sum_{k\in[K]} \sum_{\mathcal{T}\subseteq(([K]\setminus\{u_1,\ldots,u_k\})\cap\mathcal{D}_{u_k})} \left|W_{\mathcal{D}_{u_k},\mathcal{T}}\right|. \tag{85}$$

*3) Counting the Selected Demands:* To simplify the above, we proceed to count the total number of circular demands. Let us focus without loss of generality on user 1 and also on those circular demands where user 1 requests the file $W_{\mathcal{S}}$ such that $\mathcal{S} = \mathcal{D}_1 = \{1, 2, 3, 4\}$. We can see that there exists a total of $(\alpha - 1)!(K - \alpha)! = 12$ circular demands with this $\mathcal{D}_1$. If we consider $\hat{u}_1 = 1$ without loss of generality, such demands are all those associated to a vector $\hat{\boldsymbol{u}}$ where $\cup_{i=2}^{\alpha}\{\hat{u}_i\} = \mathcal{D}_1 \setminus \{1\} = \{2, 3, 4\}$ and $\cup_{i=\alpha+1}^{K}\{\hat{u}_i\} = [K] \setminus \mathcal{D}_1 = \{5, 6\}$. Given that user 1 can ask for a file among a total of $|\mathcal{F}_1| = \binom{K-1}{\alpha-1} = 10$ files, we can conclude that the total number of such circular demands is equal to $|\mathcal{C}| = |\mathcal{F}_1|(\alpha - 1)!(K - 1)! = 120$. For each such demand we can build $K$ index coding bounds, one for each of the $K$ circular shifts corresponding to the vector $\hat{\boldsymbol{u}}$, consequently resulting in a total of $\sum_{\boldsymbol{d}\in\mathcal{C}} |\mathcal{U}_{\boldsymbol{d}}| = 720$ lower bounds — as the one in (84) — being used for the expression in (85).

*4) Constructing the Optimization Problem:* Let us keep our focus on user 1 and again on those circular demands where user 1 requests the file $W_{\mathcal{S}}$ with $\mathcal{S} = \{1, 2, 3, 4\} = \mathcal{D}_1$. Since there is a one-to-one correspondence between the $(\alpha-1)!(K-\alpha)! = 12$ circular demands and the set of $K$ circular shifts of the ordered vector of users $\hat{\boldsymbol{u}}$, then — assuming without loss of generality that $\hat{u}_1 = 1$ — the vectors $\hat{\boldsymbol{u}}$ for all the 12 demands take the form

$$\hat{\boldsymbol{u}}_1 = (1, 2, 3, 4, 5, 6) \qquad \hat{\boldsymbol{u}}_7 = (1, 2, 3, 4, 6, 5) \tag{86}$$
$$\hat{\boldsymbol{u}}_2 = (1, 3, 2, 4, 5, 6) \qquad \hat{\boldsymbol{u}}_8 = (1, 3, 2, 4, 6, 5) \tag{87}$$
$$\hat{\boldsymbol{u}}_3 = (1, 3, 4, 2, 5, 6) \qquad \hat{\boldsymbol{u}}_9 = (1, 3, 4, 2, 6, 5) \tag{88}$$
$$\hat{\boldsymbol{u}}_4 = (1, 4, 3, 2, 5, 6) \qquad \hat{\boldsymbol{u}}_{10} = (1, 4, 3, 2, 6, 5) \tag{89}$$
$$\hat{\boldsymbol{u}}_5 = (1, 4, 2, 3, 5, 6) \qquad \hat{\boldsymbol{u}}_{11} = (1, 4, 2, 3, 6, 5) \tag{90}$$
$$\hat{\boldsymbol{u}}_6 = (1, 2, 4, 3, 5, 6) \qquad \hat{\boldsymbol{u}}_{12} = (1, 2, 4, 3, 6, 5). \tag{91}$$

The above one-to-one correspondence means that each vector $\hat{\boldsymbol{u}}$ above corresponds to a circular demand with $\mathcal{D}_1 = \{1,2,3,4\}$. For example, vector $\hat{\boldsymbol{u}}_1$ corresponds to the demand $\boldsymbol{d}_1 = (1234, 2345, 3456, 1456, 1256, 1236)$. In addition, for each vector $\hat{\boldsymbol{u}}$, we consider as user permutations the $K$ circular shifts of $\hat{\boldsymbol{u}}$. For instance, when we consider the circular demand associated to the vector $\hat{\boldsymbol{u}}_1$, the user permutations are given by the following set

$$\begin{aligned}\mathcal{U}_{\boldsymbol{d}_1} = \{&(1,2,3,4,5,6),(2,3,4,5,6,1),(3,4,5,6,1,2),\\ &(4,5,6,1,2,3),(5,6,1,2,3,4),(6,1,2,3,4,5)\}.\end{aligned} \tag{92}$$

Consider the subfile $W_{\mathcal{S},\mathcal{T}}$ with $\mathcal{T} \subseteq (\mathcal{S} \setminus \{1\})$ and $|\mathcal{T}| = t$. Following the same line of reasoning as in Section IV-A, and taking advantage of Lemma 2 as well as focusing on circular demands with $\mathcal{D}_1 = \mathcal{S} = \{1,2,3,4\}$, we see that this subfile is counted $(K-\ell)$ times when we consider the $K$ circular shifts of each $\hat{\boldsymbol{u}}$, where

$$\ell = \max_{j\in\mathcal{T}} \left| \bigcup_{i=2}^{\hat{u}(j)} \{\hat{u}_i\} \right| \tag{93}$$

and where we again used that $\hat{u}(1) = 1$. For example, consider $\hat{\boldsymbol{u}}_4 = (1,4,3,2,5,6)$ and the subfile $W_{\mathcal{S},24}$. Since $\ell = \max_{j\in\mathcal{T}} \left|\bigcup_{i=2}^{\hat{u}(j)} \{\hat{u}_i\}\right| = \hat{u}(2) - \hat{u}(1) = 3$, the subfile $W_{\mathcal{S},24}$ is counted a total of $(K-\ell) = 3$ times across the $K$ circular shifts of $\hat{\boldsymbol{u}}_4$.

Considering that $\ell \in [t : \alpha - 1] = [t : 3]$, as already explained in the general description of the main proof of the converse, we see that each subfile $W_{\mathcal{S},\mathcal{T}}$ with $|\mathcal{T}| = t$ and $\mathcal{T} \subseteq (\mathcal{S} \setminus \{1\})$ is counted a total of $\sum_{\ell=t}^{3} a_\ell (K-\ell)$ times in (85) when going over the circular demands having $\mathcal{D}_1 = \mathcal{S} = \{1,2,3,4\}$. The term

$$a_\ell = t!(\alpha-1-t)!(K-\alpha)!\binom{\ell-1}{t-1} = t!(3-t)!2!\binom{\ell-1}{t-1} \tag{94}$$

counts the total number of vectors $\hat{\boldsymbol{u}}$ (and consequently the number of circular demands $\boldsymbol{d} \in \mathcal{C}$) corresponding to $\mathcal{D}_1 = \{1,2,3,4\}$ and $\max_{j\in\mathcal{T}} \left|\bigcup_{i=2}^{\hat{u}(j)} \{\hat{u}_i\}\right| = \ell$. Since the same reasoning applies whenever the file $W_{\mathcal{S}}$ is requested by any other of the $(\alpha-1-t)$ users in $\mathcal{S}\setminus\{\mathcal{T},1\}$, we can see that each subfile $W_{\mathcal{S},\mathcal{T}}$ appears $(4-t)\sum_{\ell=t}^{3} a_\ell(K-\ell)$ times in (85) when we consider all circular demands in $\mathcal{C}$. Similarly, the same reasoning applies to any other file in $\mathcal{L}$. Consequently, the expression in (85) simplifies as

$$\begin{aligned} R^\star &\geq \frac{1}{B\sum_{\boldsymbol{d}\in\mathcal{C}} |\mathcal{U}_{\boldsymbol{d}}|} \sum_{\boldsymbol{d}\in\mathcal{C}} \sum_{\boldsymbol{u}\in\mathcal{U}_{\boldsymbol{d}}} \sum_{k\in[K]} \\ &\qquad \sum_{\mathcal{T}\subseteq(([K]\setminus\{u_1,\ldots,u_k\})\cap\mathcal{D}_{u_k})} \left| W_{\mathcal{D}_{u_k},\mathcal{T}} \right| \qquad (95)\\ &= \frac{1}{720B} \sum_{\boldsymbol{d}\in\mathcal{C}} \sum_{\boldsymbol{u}\in\mathcal{U}_{\boldsymbol{d}}} \sum_{k\in[K]} \\ &\qquad \sum_{\mathcal{T}\subseteq(([K]\setminus\{u_1,\ldots,u_k\})\cap\mathcal{D}_{u_k})} \left| W_{\mathcal{D}_{u_k},\mathcal{T}} \right| \qquad (96)\\ &= \sum_{t=0}^{\alpha} f(t) x_t \qquad (97)\end{aligned}$$

where we define

$$c_t \coloneqq \frac{(4-t)}{720} \sum_{\ell=t}^{\alpha-1} a_\ell (K-\ell) \tag{98}$$

$$f(t) \coloneqq N c_t \tag{99}$$

$$0 \leq x_t \coloneqq \sum_{\mathcal{S}\subseteq[K]:|\mathcal{S}|=\alpha} \sum_{\mathcal{T}\subseteq\mathcal{S}:|\mathcal{T}|=t} \frac{|W_{\mathcal{S},\mathcal{T}}|}{NB}. \tag{100}$$

At this point we can formulate the optimization problem as in (63) and lower bound its solution to obtain the converse.

*5) Lower Bounding the Solution to the Optimization Problem:* Since the variable $x_t$ can be interpreted as a probability mass function and since — as we recall from Lemma 3 — the coefficients $f(t)$ represent a strictly decreasing convex sequence, we can conclude that the optimization problem can be easily lower bounded by using Jensen's inequality and the cumulative cache size constraint. As shown in Section IV-A, the coefficients can be rewritten in the following form

$$f(t) = \frac{\binom{\alpha}{t+1} + (K-\alpha)\binom{\alpha-1}{t}}{\binom{\alpha}{t}} \tag{101}$$

so we can obtain that the converse bound is a piece-wise linear curve with corner points $\left(t\frac{5}{2}, \frac{\binom{4}{t+1}+2\binom{3}{t}}{\binom{4}{t}}\right)$ for every $t \in [0:4]$.

## V. The Exact Memory-Load Trade-Off for $\alpha$-Demands

We will here draw insights from the converse to establish a general cache placement policy, and then a delivery scheme that applies to a specific set of so-called $\alpha$-Demands. For these demands and for the specific placement policy, the scheme will be proven optimal with the use of an additional converse.

We start by noticing that the converse in Theorem 1 can be decomposed as

$$R_{\text{LB}}(t) = \frac{\binom{\alpha}{t+1}}{\binom{\alpha}{t}} + \frac{(K-\alpha)\binom{\alpha-1}{t}}{\binom{\alpha}{t}} \tag{102}$$

with the first term $R_1(t) \coloneqq \frac{\binom{\alpha}{t+1}}{\binom{\alpha}{t}} = \frac{\alpha(1-\gamma_\alpha)}{1+\alpha\gamma_\alpha}$ bringing to mind a smaller MAN placement-and-delivery (unselfish) problem with $\alpha$ users, a common library, and normalized cache size $\gamma_\alpha$, and with the second term $R_2(t) \coloneqq \frac{(K-\alpha)\binom{\alpha-1}{t}}{\binom{\alpha}{t}} = (K-\alpha)(1-\gamma_\alpha)$ bringing to mind uncoded delivery to $(K-\alpha)$ users. Let us exploit this observation to suggest a placement.

### A. Cache Placement

As noted, we can think of the $R_1(t)$ term as representing the optimal load in a "smaller" $\alpha$-MAN problem with $\alpha$ users that *are known in advance* to be interested in a common class of files and thus benefit from the corresponding $\alpha$-user MAN placement. If each user — as is the case in our setting — can allocate a fraction $\gamma_\alpha$ for each file of potential interest, then a MAN placement implies that each user stores a total of

$f\gamma_\alpha$ files.[19] Here, in our effort to provide a placement method, we must account for the fact that there is a total of $\binom{K}{\alpha}$ such "smaller" MAN problems, because there are $C = \binom{K}{\alpha}$ file classes. Let us now recall that each user appears in a total of $\binom{K-1}{\alpha-1}$ such smaller problems, since there are $\binom{K-1}{\alpha-1}$ file classes that each user is interested in. Our placement must account for the possibility of each user participating in any such smaller problem. This requires each user to store $f\gamma_\alpha$ files per class of interest, and thus requires a total storage capacity of $\gamma_\alpha f\binom{K-1}{\alpha-1} = M$, which, as we see, nicely satisfies the cache size constraint. This reasoning justifies the cache placement procedure that we present below.

In our proposed uncoded and selfish cache placement method, based on the same combinatorial argument of the MAN scheme, each user $k$ proceeds to cache only from $\mathcal{F}_k$. The placement begins by splitting each file into $\binom{\alpha}{t}$ non-overlapping subfiles as

$$W_{i,\mathcal{S}} = \{W_{i,\mathcal{S},\mathcal{T}} : \mathcal{T} \subseteq \mathcal{S}, |\mathcal{T}| = t\} \tag{103}$$

for each $\mathcal{S} \subseteq [K]$ such that $|\mathcal{S}| = \alpha$ and for each $i \in [f]$, and then is completed by filling the cache $\mathcal{Z}_k$ of each user $k \in [K]$ as

$$\mathcal{Z}_k = \{W_{i,\mathcal{S},\mathcal{T}} : i \in [f], \mathcal{S} \subseteq [K], |\mathcal{S}| = \alpha, \mathcal{T} \subseteq \mathcal{S}, \\ |\mathcal{T}| = t, k \in \mathcal{T}\}. \tag{104}$$

Each cache stores $\binom{\alpha-1}{t-1}$ subfiles for each file in its FDS, so abiding by the cache size constraint

$$|\mathcal{F}|\binom{\alpha-1}{t-1}\frac{B}{\binom{\alpha}{t}} = f\binom{K-1}{\alpha-1}\frac{t}{\alpha}B = MB. \tag{105}$$

*Remark 5:* Unfortunately, the above reasoning does not immediately reflect — at least not to us — a universal delivery solution for any set of demands. To the best of our understanding, our cache placement introduces the need to resolve a large number of non-isomorphic index coding problems. Nevertheless, the careful interpretation of the converse allowed us to suggest a general selfish cache placement policy. Similarly, it also allows us to identify a well-defined class of demands, as we can see below.

### B. Delivery Scheme for the Set of $\alpha$-Demands

We now present the delivery method for the following class of demands.

*Definition 4 ($\alpha$-Demands):* Considering the $(K, \alpha, f)$ FDS structure with $f \geq \alpha$, the demand defined by the vectors $\boldsymbol{d} = (\mathcal{D}_1, \ldots, \mathcal{D}_K)$ and $\boldsymbol{f} = (f_1, \ldots, f_K)$ is an $\alpha$-demand if and only if there exists at least one set of users $\mathcal{K} \subseteq [K]$ such that $|\mathcal{K}| = \alpha$, $f_{k_1} \neq f_{k_2}$ for any $k_1 \neq k_2$ with $k_1, k_2 \in \mathcal{K}$ and $\mathcal{D}_k = \mathcal{K}$ for all $k \in \mathcal{K}$.

Such demands can exist only if $f \geq \alpha$. Indeed, if we have at least $\alpha$ files per class, then we can have distinct demands where there exists at least one set of $\alpha$ users requesting distinct files, all belonging to the same file class.

Let $R_{\alpha,\mathrm{c}}$ denote the worst-case load when only $\alpha$-demands are considered, and when the cache placement in Section V-A is adopted. We are now ready to provide the exact characterization of optimal such load $R^\star_{\alpha,\mathrm{c}}$.

*Proposition 1 (The Exact Memory-Load Trade-Off for $\alpha$-Demands Under the Presented Symmetric Placement):* Assuming the selfish and uncoded cache placement presented in Section V-A, the optimal worst-case communication load $R^\star_{\alpha,\mathrm{c}}$ for the $(K, \alpha, f)$ FDS structure and $\alpha$-Demands is a piece-wise linear curve with corner points

$$(M, R^\star_{\alpha,\mathrm{c}}) = \left(t\frac{N}{K}, \frac{\binom{\alpha}{t+1} + (K-\alpha)\binom{\alpha-1}{t}}{\binom{\alpha}{t}}\right), \quad \forall t \in [0:\alpha] \tag{106}$$

again corresponding to

$$R^\star_{\alpha,\mathrm{c}} = \frac{K(1-\gamma_\alpha)}{K\gamma + 1}\Big[(K-\alpha)\gamma + 1\Big]. \tag{107}$$

*Proof:* The proof of the converse is reported in Appendix E, whereas the proof of the achievability is reported below in Section V-C. □

### C. Achievability Proof of Proposition 1

By definition, any $\alpha$-demand has at least one set of $\alpha$ users requesting distinct files from the same file class. If we denote by $\mathcal{K}$ one of such sets, it holds that $\mathcal{D}_k = \mathcal{K}$ for all $k \in \mathcal{K}$ and $f_{k_1} \neq f_{k_2}$ for all $k_1 \neq k_2$ with $k_1, k_2 \in \mathcal{K}$. Consider user $k \in \mathcal{K}$. According to the cache placement procedure in Section V-A, this user does not have in its cache any subfile $W_{f_k,\mathcal{K},\mathcal{T}}$ where $\mathcal{T} \subseteq \mathcal{K}$, $|\mathcal{T}| = t$ and $k \notin \mathcal{T}$. If we focus on this set $\mathcal{K}$ of $\alpha$ users only, we can automatically construct the following sequence of multicast messages

$$X_{\mathcal{K}} = \left(\bigoplus_{k\in\mathcal{S}} W_{f_k,\mathcal{K},\mathcal{S}\setminus\{k\}} : \mathcal{S} \subseteq \mathcal{K}, |\mathcal{S}| = t+1\right). \tag{108}$$

For the remaining $(K-\alpha)$ users in $[K] \setminus \mathcal{K}$, we consider the following sequence

$$X_{[K]\setminus\mathcal{K}} = (W_{f_k,\mathcal{D}_k,\mathcal{T}} : k \in ([K] \setminus \mathcal{K}), k \notin \mathcal{T}) \tag{109}$$

of uncoded transmissions. Then, the transmitter delivers the concatenated $X = (X_{\mathcal{K}}, X_{[K]\setminus\mathcal{K}})$, inducing a load

$$\frac{|X|}{B} = \frac{|X_{\mathcal{K}}| + |X_{[K]\setminus\mathcal{K}}|}{B} = \frac{\binom{\alpha}{t+1} + (K-\alpha)\binom{\alpha-1}{t}}{\binom{\alpha}{t}} \tag{110}$$

which implies that $R^\star_{\alpha,\mathrm{c}}(t) \leq \frac{\binom{\alpha}{t+1}+(K-\alpha)\binom{\alpha-1}{t}}{\binom{\alpha}{t}}$ for all $t \in [0:\alpha]$.

*Remark 6:* As a testament to the additional usefulness of the converse developed here, we point out that the above achievable scheme is developed as a direct outcome of a meticulous inspection of the expression of the bound itself. □

### D. Example of the Achievable Scheme

Consider the $(K, \alpha, f) = (5, 3, 3)$ FDS structure. We have $C = \binom{K}{\alpha} = \binom{5}{3} = 10$ classes of files with a total of $N = fC = 30$ files. The FDS structure is given by

$$\mathcal{F}_1 = \{\mathcal{W}_{123}, \mathcal{W}_{124}, \mathcal{W}_{125}, \mathcal{W}_{134}, \mathcal{W}_{135}, \mathcal{W}_{145}\} \tag{111}$$

[19] Recall that $f$ is the total number of files in this common class of files.

$$\mathcal{F}_2 = \{\mathcal{W}_{123}, \mathcal{W}_{124}, \mathcal{W}_{125}, \mathcal{W}_{234}, \mathcal{W}_{235}, \mathcal{W}_{245}\} \quad (112)$$
$$\mathcal{F}_3 = \{\mathcal{W}_{123}, \mathcal{W}_{134}, \mathcal{W}_{135}, \mathcal{W}_{234}, \mathcal{W}_{235}, \mathcal{W}_{345}\} \quad (113)$$
$$\mathcal{F}_4 = \{\mathcal{W}_{124}, \mathcal{W}_{134}, \mathcal{W}_{145}, \mathcal{W}_{234}, \mathcal{W}_{245}, \mathcal{W}_{345}\} \quad (114)$$
$$\mathcal{F}_5 = \{\mathcal{W}_{125}, \mathcal{W}_{135}, \mathcal{W}_{145}, \mathcal{W}_{235}, \mathcal{W}_{245}, \mathcal{W}_{345}\} \quad (115)$$

where $\mathcal{W}_{\mathcal{S}} = \{W_{1,\mathcal{S}}, W_{2,\mathcal{S}}, W_{3,\mathcal{S}}\}$ for each triplet $\mathcal{S} \subseteq [5]$. Let us consider the scenario of $t = 2$. In this case, each file is split as

$$W_{i,123} = \{W_{i,123,12}, W_{i,123,13}, W_{i,123,23}\} \quad (116)$$
$$W_{i,124} = \{W_{i,124,12}, W_{i,124,14}, W_{i,124,24}\} \quad (117)$$
$$W_{i,125} = \{W_{i,125,12}, W_{i,125,15}, W_{i,125,25}\} \quad (118)$$
$$W_{i,134} = \{W_{i,134,13}, W_{i,134,14}, W_{i,134,34}\} \quad (119)$$
$$W_{i,135} = \{W_{i,135,13}, W_{i,135,15}, W_{i,135,35}\} \quad (120)$$
$$W_{i,145} = \{W_{i,145,14}, W_{i,145,15}, W_{i,145,45}\} \quad (121)$$
$$W_{i,234} = \{W_{i,234,23}, W_{i,234,24}, W_{i,234,34}\} \quad (122)$$
$$W_{i,235} = \{W_{i,235,23}, W_{i,235,25}, W_{i,235,35}\} \quad (123)$$
$$W_{i,245} = \{W_{i,245,24}, W_{i,245,25}, W_{i,245,45}\} \quad (124)$$
$$W_{i,345} = \{W_{i,345,34}, W_{i,345,35}, W_{i,345,45}\} \quad (125)$$

for all $i \in [3]$.

Consider the demand defined by $\boldsymbol{f} = (1, 2, 3, 1, 1)$ and $\boldsymbol{d} = (123, 123, 123, 124, 125)$. This is an $\alpha$-demand because there exists a set $\mathcal{K}$ of $\alpha$ users all requesting distinct files belonging to the same file class $\mathcal{K}$. Here, this set is $\mathcal{K} = \{1, 2, 3\}$.

In accordance to the described selfish and uncoded cache placement, each user desires a total of $\binom{\alpha-1}{t}$ subfiles which are not in its cache. In this case, each user simply desires $\binom{2}{2} = 1$ subfile. The delivery of these subfiles involves the following MAN XOR

$$X_{123} = \left( \bigoplus_{k \in \mathcal{S}} W_{f_k, \mathcal{K}, \mathcal{S}\setminus\{k\}} : \mathcal{S} \subseteq \mathcal{K}, |\mathcal{S}| = t+1 \right) \quad (126)$$
$$= (W_{1,123,23} \oplus W_{2,123,13} \oplus W_{3,123,12}) \quad (127)$$

and then the following two uncoded transmissions

$$X_{45} = (W_{1,124,12}, W_{1,125,12}) \quad (128)$$

that serve the users outside $\mathcal{K}$. Given that the subpacketization is $\binom{\alpha}{t} = \binom{3}{2} = 3$, the transmitted signal $X = (X_{123}, X_{45})$ induces a communication load of $R_{\alpha,\mathrm{c}}(2) = |X|/B = 1$ which matches the corresponding optimal $R^{\star}_{\alpha,\mathrm{c}}(2)$ from Theorem 1.

## VI. Additional Optimal Schemes for Circular Demands

We here present schemes that optimally deliver circular demands. We will do so for the $(5, 4, f)$ FDS structure with $t \in \{2, 3\}$, and for the $(6, 5, f)$ FDS structure with $t = 3$. The optimal schemes assume the selfish and uncoded cache placement described in Section V-A. We prove optimality simply by showing that the load provided by the proposed achievable schemes matches the converse bound in Theorem 1. This suffices because, as we might recall, the construction of the converse employed only circular demands.[20]

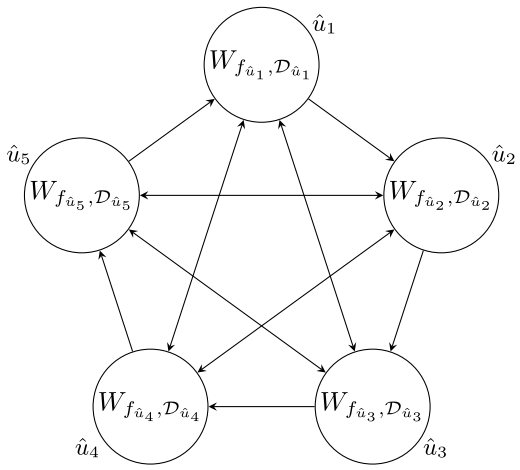


Fig. 7. FDS request graph for a generic circular demand identified by the vector $\hat{\boldsymbol{u}} = (\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4, \hat{u}_5)$ and the $(K, \alpha, f) = (5, 4, f)$ FDS structure.

### A. Circular Demands and the $(5, 4, f)$ FDS Structure

The scheme presented here is a generalization, for any circular demand, of the example in Section II-B. For the considered $(K, \alpha, f) = (5, 4, f)$ structure, we know that there are $C = \binom{K}{\alpha} = 5$ file classes $\mathcal{W}_{1234}, \mathcal{W}_{1235}, \mathcal{W}_{1245}, \mathcal{W}_{1345}, \mathcal{W}_{2345}$, corresponding to $N = fC = 5f$ files. The 5 FDSs take the form

$$\mathcal{F}_1 = \{\mathcal{W}_{1234}, \mathcal{W}_{1235}, \mathcal{W}_{1245}, \mathcal{W}_{1345}\} \quad (129)$$
$$\mathcal{F}_2 = \{\mathcal{W}_{1234}, \mathcal{W}_{1235}, \mathcal{W}_{1245}, \mathcal{W}_{2345}\} \quad (130)$$
$$\mathcal{F}_3 = \{\mathcal{W}_{1234}, \mathcal{W}_{1235}, \mathcal{W}_{1345}, \mathcal{W}_{2345}\} \quad (131)$$
$$\mathcal{F}_4 = \{\mathcal{W}_{1234}, \mathcal{W}_{1245}, \mathcal{W}_{1345}, \mathcal{W}_{2345}\} \quad (132)$$
$$\mathcal{F}_5 = \{\mathcal{W}_{1235}, \mathcal{W}_{1245}, \mathcal{W}_{1345}, \mathcal{W}_{2345}\} \quad (133)$$

where we recall that $\mathcal{W}_{\mathcal{S}} = \{W_{i,\mathcal{S}} : i \in [f]\}$. The scheme works for any value of $f \in \mathbb{Z}^+$, and for any circular demand. Such general circular demand is identified by the vector $\hat{\boldsymbol{u}} = (\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4, \hat{u}_5)$ that must satisfy the property (cf. Defintion 3) that $\cup_{i=k+1}^{k+3} \{\hat{u}_i\} = \mathcal{D}_{\hat{u}_k} \setminus \{\hat{u}_k\}$ for each $k \in [5]$. The FDS request graph of such generic circular demand is shown in Fig. 7. Notice that, as expected, this is not a complete graph.

Before presenting the schemes, we wish to point out the key novelty and the general idea. As already mentioned in the example in Section II-B, the crucial aspect consists of designing multicast messages that not only deliver messages themselves, but where also their linear combinations can be used by the users to correctly decode requested subfiles. Hence, the idea here is to find an efficient way to align interference, such that users can cache-out interference terms

[20] We wish to point out that the fact that the converse is matched when considering some instances of circular demands does not necessarily imply that the converse is exactly matched in general nor that circular demands are worst-case demands.

by simply XORing some of the received messages. The pivotal aspect of this approach is the ability of choosing the users that will have to take linear combinations of messages to cancel interference and correctly decode the requested subfiles.

*1) The Case of $t = 2$:* According to the cache placement in Section V-A, each file is split into $\binom{\alpha}{t} = \binom{4}{2} = 6$ non-overlapping subfiles as

$$W_{i,1234} = \{W_{i,1234,12}, W_{i,1234,13}, W_{i,1234,14}, W_{i,1234,23}, W_{i,1234,24}, W_{i,1234,34}\} \quad (134)$$

$$W_{i,1235} = \{W_{i,1235,12}, W_{i,1235,13}, W_{i,1235,15}, W_{i,1235,23}, W_{i,1235,25}, W_{i,1235,35}\} \quad (135)$$

$$W_{i,1245} = \{W_{i,1245,12}, W_{i,1245,14}, W_{i,1245,15}, W_{i,1245,24}, W_{i,1245,25}, W_{i,1245,45}\} \quad (136)$$

$$W_{i,1345} = \{W_{i,1345,13}, W_{i,1345,14}, W_{i,1345,15}, W_{i,1345,34}, W_{i,1345,35}, W_{i,1345,45}\} \quad (137)$$

$$W_{i,2345} = \{W_{i,2345,23}, W_{i,2345,24}, W_{i,2345,25}, W_{i,2345,34}, W_{i,2345,35}, W_{i,2345,45}\} \quad (138)$$

for each $i \in [f]$. Considering then that the cache content of each user $k \in [5]$ is filled as

$$\mathcal{Z}_k = \{W_{i,\mathcal{S},\mathcal{T}} : i \in [f], \mathcal{S} \subseteq [5], |\mathcal{S}| = 4, \mathcal{T} \subseteq \mathcal{S}, |\mathcal{T}| = 2, k \in \mathcal{T}\} \quad (139)$$

it can be easily seen that each user desires a total of $\binom{\alpha-1}{t} = \binom{3}{2} = 3$ subfiles, each of size $B/6$ bits. Recalling that the vector of ordered users $\hat{\boldsymbol{u}} = (\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4, \hat{u}_5)$ satisfies $\mathcal{D}_{\hat{u}_k} = \cup_{i=k}^{k+3}\{\hat{u}_i\}$ for each $k \in [5]$, we conclude that the subfiles desired by each user are the following.

- User $\hat{u}_1$. The subfiles desired are $W_{f_{\hat{u}_1},\mathcal{D}_{\hat{u}_1},\{\hat{u}_2,\hat{u}_3\}}$, $W_{f_{\hat{u}_1},\mathcal{D}_{\hat{u}_1},\{\hat{u}_2,\hat{u}_4\}}$ and $W_{f_{\hat{u}_1},\mathcal{D}_{\hat{u}_1},\{\hat{u}_3,\hat{u}_4\}}$.
- User $\hat{u}_2$. The subfiles desired are $W_{f_{\hat{u}_2},\mathcal{D}_{\hat{u}_2},\{\hat{u}_3,\hat{u}_4\}}$, $W_{f_{\hat{u}_2},\mathcal{D}_{\hat{u}_2},\{\hat{u}_3,\hat{u}_5\}}$ and $W_{f_{\hat{u}_2},\mathcal{D}_{\hat{u}_2},\{\hat{u}_4,\hat{u}_5\}}$.
- User $\hat{u}_3$. The subfiles desired are $W_{f_{\hat{u}_3},\mathcal{D}_{\hat{u}_3},\{\hat{u}_1,\hat{u}_4\}}$, $W_{f_{\hat{u}_3},\mathcal{D}_{\hat{u}_3},\{\hat{u}_1,\hat{u}_5\}}$ and $W_{f_{\hat{u}_3},\mathcal{D}_{\hat{u}_3},\{\hat{u}_4,\hat{u}_5\}}$.
- User $\hat{u}_4$. The subfiles desired are $W_{f_{\hat{u}_4},\mathcal{D}_{\hat{u}_4},\{\hat{u}_1,\hat{u}_2\}}$, $W_{f_{\hat{u}_4},\mathcal{D}_{\hat{u}_4},\{\hat{u}_1,\hat{u}_5\}}$ and $W_{f_{\hat{u}_4},\mathcal{D}_{\hat{u}_4},\{\hat{u}_2,\hat{u}_5\}}$.
- User $\hat{u}_5$. The subfiles desired are $W_{f_{\hat{u}_5},\mathcal{D}_{\hat{u}_5},\{\hat{u}_1,\hat{u}_2\}}$, $W_{f_{\hat{u}_5},\mathcal{D}_{\hat{u}_5},\{\hat{u}_1,\hat{u}_3\}}$ and $W_{f_{\hat{u}_5},\mathcal{D}_{\hat{u}_5},\{\hat{u}_2,\hat{u}_3\}}$.

These subfiles are delivered by the following sequence of XORs

$$X_1 = W_{f_{\hat{u}_1},\mathcal{D}_{\hat{u}_1},\{\hat{u}_2,\hat{u}_3\}} \oplus W_{f_{\hat{u}_2},\mathcal{D}_{\hat{u}_2},\{\hat{u}_3,\hat{u}_5\}} \oplus W_{f_{\hat{u}_3},\mathcal{D}_{\hat{u}_3},\{\hat{u}_1,\hat{u}_4\}} \quad (140)$$

$$X_2 = W_{f_{\hat{u}_3},\mathcal{D}_{\hat{u}_3},\{\hat{u}_1,\hat{u}_4\}} \oplus W_{f_{\hat{u}_1},\mathcal{D}_{\hat{u}_1},\{\hat{u}_2,\hat{u}_4\}} \oplus W_{f_{\hat{u}_4},\mathcal{D}_{\hat{u}_4},\{\hat{u}_1,\hat{u}_2\}} \quad (141)$$

$$X_3 = W_{f_{\hat{u}_2},\mathcal{D}_{\hat{u}_2},\{\hat{u}_3,\hat{u}_5\}} \oplus W_{f_{\hat{u}_5},\mathcal{D}_{\hat{u}_5},\{\hat{u}_1,\hat{u}_3\}} \oplus W_{f_{\hat{u}_3},\mathcal{D}_{\hat{u}_3},\{\hat{u}_1,\hat{u}_5\}} \quad (142)$$

$$X_4 = W_{f_{\hat{u}_1},\mathcal{D}_{\hat{u}_1},\{\hat{u}_3,\hat{u}_4\}} \oplus W_{f_{\hat{u}_4},\mathcal{D}_{\hat{u}_4},\{\hat{u}_1,\hat{u}_5\}} \quad (143)$$

$$X_5 = W_{f_{\hat{u}_2},\mathcal{D}_{\hat{u}_2},\{\hat{u}_3,\hat{u}_4\}} \oplus W_{f_{\hat{u}_4},\mathcal{D}_{\hat{u}_4},\{\hat{u}_2,\hat{u}_5\}} \quad (144)$$

$$X_6 = W_{f_{\hat{u}_2},\mathcal{D}_{\hat{u}_2},\{\hat{u}_4,\hat{u}_5\}} \oplus W_{f_{\hat{u}_5},\mathcal{D}_{\hat{u}_5},\{\hat{u}_1,\hat{u}_2\}} \quad (145)$$

$$X_7 = W_{f_{\hat{u}_3},\mathcal{D}_{\hat{u}_3},\{\hat{u}_4,\hat{u}_5\}} \oplus W_{f_{\hat{u}_5},\mathcal{D}_{\hat{u}_5},\{\hat{u}_2,\hat{u}_3\}} \quad (146)$$

after which each user $k \in [5]$ can employ its own cache content $\mathcal{Z}_k$ to decode as follows.

- User $\hat{u}_1$ recovers its desired subfiles from $X_1 \oplus X_3$, $X_2$ and $X_4$.
- User $\hat{u}_2$ recovers its desired subfiles from $X_1 \oplus X_2$, $X_5$ and $X_6$.
- User $\hat{u}_3$ recovers its desired subfiles from $X_1$, $X_3$ and $X_7$.
- User $\hat{u}_4$ recovers its desired subfiles from $X_2$, $X_4$ and $X_5$.
- User $\hat{u}_5$ recovers its desired subfiles from $X_3$, $X_6$ and $X_7$.

Given that $|X_i| = B/6$ for each $i \in [7]$, and given that there are 7 transmissions, we have a load of $R(2) = |X|/B = 7/6$. Since then $R_{\text{LB}}(2) = 7/6$, we can conclude that the converse is tight.

*2) The Case of $t = 3$:* In this case each file is split into $\binom{4}{3} = 4$ non-overlapping subfiles as

$$W_{i,1234} = \{W_{i,1234,123}, W_{i,1234,124}, W_{i,1234,134}, W_{i,1234,234}\} \quad (147)$$

$$W_{i,1235} = \{W_{i,1235,123}, W_{i,1235,125}, W_{i,1235,135}, W_{i,1235,235}\} \quad (148)$$

$$W_{i,1245} = \{W_{i,1245,124}, W_{i,1245,125}, W_{i,1245,145}, W_{i,1245,245}\} \quad (149)$$

$$W_{i,1345} = \{W_{i,1345,134}, W_{i,1345,135}, W_{i,1345,145}, W_{i,1345,345}\} \quad (150)$$

$$W_{i,2345} = \{W_{i,2345,234}, W_{i,2345,235}, W_{i,2345,245}, W_{i,2345,345}\} \quad (151)$$

for each $i \in [f]$. Each user then desires $\binom{3}{3} = 1$ subfile of size $B/4$. More precisely, always considering the general circular demands identified by the vector $\hat{\boldsymbol{u}}$, the desired subfiles are given as follows.

- User $\hat{u}_1$ desires $W_{f_{\hat{u}_1},\mathcal{D}_{\hat{u}_1},\{\hat{u}_2,\hat{u}_3,\hat{u}_4\}}$.
- User $\hat{u}_2$ desires $W_{f_{\hat{u}_2},\mathcal{D}_{\hat{u}_2},\{\hat{u}_3,\hat{u}_4,\hat{u}_5\}}$.
- User $\hat{u}_3$ desires $W_{f_{\hat{u}_3},\mathcal{D}_{\hat{u}_3},\{\hat{u}_1,\hat{u}_4,\hat{u}_5\}}$.
- User $\hat{u}_4$ desires $W_{f_{\hat{u}_4},\mathcal{D}_{\hat{u}_4},\{\hat{u}_1,\hat{u}_2,\hat{u}_5\}}$.
- User $\hat{u}_5$ desires $W_{f_{\hat{u}_5},\mathcal{D}_{\hat{u}_5},\{\hat{u}_1,\hat{u}_2,\hat{u}_3\}}$.

After transmitting the following two XORs

$$X_1 = W_{f_{\hat{u}_1},\mathcal{D}_{\hat{u}_1},\{\hat{u}_2,\hat{u}_3,\hat{u}_4\}} \oplus W_{f_{\hat{u}_2},\mathcal{D}_{\hat{u}_2},\{\hat{u}_3,\hat{u}_4,\hat{u}_5\}} \oplus W_{f_{\hat{u}_4},\mathcal{D}_{\hat{u}_4},\{\hat{u}_1,\hat{u}_2,\hat{u}_5\}} \quad (152)$$

$$X_2 = W_{f_{\hat{u}_1},\mathcal{D}_{\hat{u}_1},\{\hat{u}_2,\hat{u}_3,\hat{u}_4\}} \oplus W_{f_{\hat{u}_3},\mathcal{D}_{\hat{u}_3},\{\hat{u}_1,\hat{u}_4,\hat{u}_5\}} \oplus W_{f_{\hat{u}_5},\mathcal{D}_{\hat{u}_5},\{\hat{u}_1,\hat{u}_2,\hat{u}_3\}} \quad (153)$$

each user can decode as follows.

- User $\hat{u}_1$ and user $\hat{u}_3$ recover their desired subfiles from $X_2$.
- User $\hat{u}_2$ and user $\hat{u}_4$ recover their desired subfiles from $X_1$.
- User $\hat{u}_5$ recovers its desired subfile from $X_1 \oplus X_2$.

Recalling that $|X_1| = |X_2| = B/4$, the 2 transmissions correspond to a communication load $R(3) = |X|/B = 1/2$ which matches the converse $R_{\text{LB}}(3) = 1/2$. This means that the scheme is optimal among all the caching-and-delivery schemes that deliver circular demands.

### *B. Circular Demands and the $(6, 5, f)$ FDS Structure*

In this setting we consider the $(K, \alpha, f) = (6, 5, f)$ structure, where there is a total of $C = 6$ classes of files

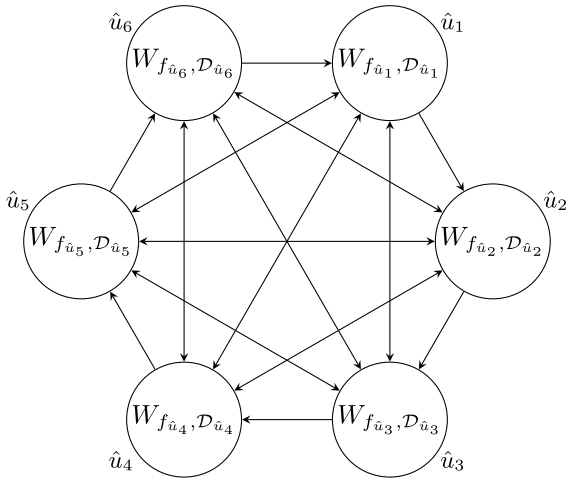


Fig. 8. FDS request graph for a generic circular demand identified by the vector $\hat{\boldsymbol{u}} = (\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4, \hat{u}_5, \hat{u}_6)$ and the $(K, \alpha, f) = (6, 5, f)$ FDS structure.

$\mathcal{W}_{12345}, \mathcal{W}_{12346}, \mathcal{W}_{12356}, \mathcal{W}_{12456}, \mathcal{W}_{13456}, \mathcal{W}_{23456}$ and there are $N = fC = 6f$ files. The 6 FDSs take the form

$$\mathcal{F}_1 = \{\mathcal{W}_{12345}, \mathcal{W}_{12346}, \mathcal{W}_{12356}, \mathcal{W}_{12456}, \mathcal{W}_{13456}\} \quad (154)$$
$$\mathcal{F}_2 = \{\mathcal{W}_{12345}, \mathcal{W}_{12346}, \mathcal{W}_{12356}, \mathcal{W}_{12456}, \mathcal{W}_{23456}\} \quad (155)$$
$$\mathcal{F}_3 = \{\mathcal{W}_{12345}, \mathcal{W}_{12346}, \mathcal{W}_{12356}, \mathcal{W}_{13456}, \mathcal{W}_{23456}\} \quad (156)$$
$$\mathcal{F}_4 = \{\mathcal{W}_{12345}, \mathcal{W}_{12346}, \mathcal{W}_{12456}, \mathcal{W}_{13456}, \mathcal{W}_{23456}\} \quad (157)$$
$$\mathcal{F}_5 = \{\mathcal{W}_{12345}, \mathcal{W}_{12356}, \mathcal{W}_{12456}, \mathcal{W}_{13456}, \mathcal{W}_{23456}\} \quad (158)$$
$$\mathcal{F}_6 = \{\mathcal{W}_{12346}, \mathcal{W}_{12356}, \mathcal{W}_{12456}, \mathcal{W}_{13456}, \mathcal{W}_{23456}\} \quad (159)$$

where $\mathcal{W}_{\mathcal{S}} = \{W_{i,\mathcal{S}} : i \in [f]\}$. As in the previous case, we here provide a scheme for any $f \in \mathbb{Z}^+$ and any circular demand. Each such circular demand is identified by a vector $\hat{\boldsymbol{u}} = (\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4, \hat{u}_5, \hat{u}_6)$, and it induces the FDS request graph in Fig. 8. The optimal scheme is provided for the case $t = 3$.

According to the cache placement in Section V-A, each file is split into $\binom{\alpha}{t} = \binom{5}{3} = 10$ non-overlapping subfiles as

$$W_{i,\mathcal{S}} = \{W_{i,\mathcal{S},\mathcal{T}} : \mathcal{T} \subseteq \mathcal{S}, |\mathcal{T}| = 3\} \quad (160)$$

for each $\mathcal{S} \subseteq [6]$ such that $|\mathcal{S}| = 5$ and for each $i \in [f]$, where each subfile has size $B/10$. For example, the file $W_{i,12345}$ is split into 10 non-overlapping subfiles labeled as $W_{i,12345,\mathcal{T}}$ for each $\mathcal{T} \in \{123, 124, 125, 134, 135, 145, 234, 235, 245, 345\}$. We recall that the set $\mathcal{T}$ represents the users which the subfile $W_{i,12345,\mathcal{T}}$ is exactly and uniquely cached at. If we consider a generic circular demand, each user misses $\binom{\alpha-1}{t} = \binom{4}{3} = 4$ subfiles given by the following.

- The subfiles $W_{f_{\hat{u}_1},\mathcal{D}_{\hat{u}_1},\{\hat{u}_2,\hat{u}_3,\hat{u}_4\}}$, $W_{f_{\hat{u}_1},\mathcal{D}_{\hat{u}_1},\{\hat{u}_2,\hat{u}_3,\hat{u}_5\}}$, $W_{f_{\hat{u}_1},\mathcal{D}_{\hat{u}_1},\{\hat{u}_2,\hat{u}_4,\hat{u}_5\}}$ and $W_{f_{\hat{u}_1},\mathcal{D}_{\hat{u}_1},\{\hat{u}_3,\hat{u}_4,\hat{u}_5\}}$ are desired by user $\hat{u}_1$.
- The subfiles $W_{f_{\hat{u}_2},\mathcal{D}_{\hat{u}_2},\{\hat{u}_3,\hat{u}_4,\hat{u}_5\}}$, $W_{f_{\hat{u}_2},\mathcal{D}_{\hat{u}_2},\{\hat{u}_3,\hat{u}_4,\hat{u}_6\}}$, $W_{f_{\hat{u}_2},\mathcal{D}_{\hat{u}_2},\{\hat{u}_3,\hat{u}_5,\hat{u}_6\}}$ and $W_{f_{\hat{u}_2},\mathcal{D}_{\hat{u}_2},\{\hat{u}_4,\hat{u}_5,\hat{u}_6\}}$ are desired by user $\hat{u}_2$
- The subfiles $W_{f_{\hat{u}_3},\mathcal{D}_{\hat{u}_3},\{\hat{u}_1,\hat{u}_4,\hat{u}_5\}}$, $W_{f_{\hat{u}_3},\mathcal{D}_{\hat{u}_3},\{\hat{u}_1,\hat{u}_4,\hat{u}_6\}}$, $W_{f_{\hat{u}_3},\mathcal{D}_{\hat{u}_3},\{\hat{u}_1,\hat{u}_5,\hat{u}_6\}}$ and $W_{f_{\hat{u}_3},\mathcal{D}_{\hat{u}_3},\{\hat{u}_4,\hat{u}_5,\hat{u}_6\}}$ are desired by user $\hat{u}_3$.
- The subfiles $W_{f_{\hat{u}_4},\mathcal{D}_{\hat{u}_4},\{\hat{u}_1,\hat{u}_2,\hat{u}_5\}}$, $W_{f_{\hat{u}_4},\mathcal{D}_{\hat{u}_4},\{\hat{u}_1,\hat{u}_2,\hat{u}_6\}}$, $W_{f_{\hat{u}_4},\mathcal{D}_{\hat{u}_4},\{\hat{u}_1,\hat{u}_5,\hat{u}_6\}}$ and $W_{f_{\hat{u}_4},\mathcal{D}_{\hat{u}_4},\{\hat{u}_2,\hat{u}_5,\hat{u}_6\}}$ are desired by user $\hat{u}_4$.
- The subfiles $W_{f_{\hat{u}_5},\mathcal{D}_{\hat{u}_5},\{\hat{u}_1,\hat{u}_2,\hat{u}_3\}}$, $W_{f_{\hat{u}_5},\mathcal{D}_{\hat{u}_5},\{\hat{u}_1,\hat{u}_2,\hat{u}_6\}}$, $W_{f_{\hat{u}_5},\mathcal{D}_{\hat{u}_5},\{\hat{u}_1,\hat{u}_3,\hat{u}_6\}}$ and $W_{f_{\hat{u}_5},\mathcal{D}_{\hat{u}_5},\{\hat{u}_2,\hat{u}_3,\hat{u}_6\}}$ are desired by user $\hat{u}_5$.
- The subfiles $W_{f_{\hat{u}_6},\mathcal{D}_{\hat{u}_6},\{\hat{u}_1,\hat{u}_2,\hat{u}_3\}}$, $W_{f_{\hat{u}_6},\mathcal{D}_{\hat{u}_6},\{\hat{u}_1,\hat{u}_2,\hat{u}_4\}}$, $W_{f_{\hat{u}_6},\mathcal{D}_{\hat{u}_6},\{\hat{u}_1,\hat{u}_3,\hat{u}_4\}}$ and $W_{f_{\hat{u}_6},\mathcal{D}_{\hat{u}_6},\{\hat{u}_2,\hat{u}_3,\hat{u}_4\}}$ are desired by user $\hat{u}_6$.

If we consider the following linear combinations of subfiles

$$X_1 = W_{f_{\hat{u}_1},\mathcal{D}_{\hat{u}_1},\{\hat{u}_2,\hat{u}_3,\hat{u}_4\}} \oplus W_{f_{\hat{u}_2},\mathcal{D}_{\hat{u}_2},\{\hat{u}_4,\hat{u}_5,\hat{u}_6\}} \oplus W_{f_{\hat{u}_4},\mathcal{D}_{\hat{u}_4},\{\hat{u}_2,\hat{u}_5,\hat{u}_6\}} \quad (161)$$
$$X_2 = W_{f_{\hat{u}_4},\mathcal{D}_{\hat{u}_4},\{\hat{u}_1,\hat{u}_5,\hat{u}_6\}} \oplus W_{f_{\hat{u}_1},\mathcal{D}_{\hat{u}_1},\{\hat{u}_2,\hat{u}_3,\hat{u}_5\}} \oplus W_{f_{\hat{u}_5},\mathcal{D}_{\hat{u}_5},\{\hat{u}_1,\hat{u}_2,\hat{u}_3\}} \quad (162)$$
$$X_3 = W_{f_{\hat{u}_1},\mathcal{D}_{\hat{u}_1},\{\hat{u}_2,\hat{u}_3,\hat{u}_4\}} \oplus W_{f_{\hat{u}_4},\mathcal{D}_{\hat{u}_4},\{\hat{u}_1,\hat{u}_5,\hat{u}_6\}} \oplus W_{f_{\hat{u}_6},\mathcal{D}_{\hat{u}_6},\{\hat{u}_1,\hat{u}_3,\hat{u}_4\}} \oplus W_{f_{\hat{u}_3},\mathcal{D}_{\hat{u}_3},\{\hat{u}_1,\hat{u}_4,\hat{u}_6\}} \quad (163)$$
$$X_4 = W_{f_{\hat{u}_2},\mathcal{D}_{\hat{u}_2},\{\hat{u}_3,\hat{u}_4,\hat{u}_5\}} \oplus W_{f_{\hat{u}_3},\mathcal{D}_{\hat{u}_3},\{\hat{u}_1,\hat{u}_5,\hat{u}_6\}} \oplus W_{f_{\hat{u}_5},\mathcal{D}_{\hat{u}_5},\{\hat{u}_1,\hat{u}_3,\hat{u}_6\}} \quad (164)$$
$$X_5 = W_{f_{\hat{u}_5},\mathcal{D}_{\hat{u}_5},\{\hat{u}_1,\hat{u}_2,\hat{u}_6\}} \oplus W_{f_{\hat{u}_2},\mathcal{D}_{\hat{u}_2},\{\hat{u}_3,\hat{u}_4,\hat{u}_6\}} \oplus W_{f_{\hat{u}_6},\mathcal{D}_{\hat{u}_6},\{\hat{u}_2,\hat{u}_3,\hat{u}_4\}} \quad (165)$$
$$X_6 = W_{f_{\hat{u}_2},\mathcal{D}_{\hat{u}_2},\{\hat{u}_3,\hat{u}_4,\hat{u}_5\}} \oplus W_{f_{\hat{u}_5},\mathcal{D}_{\hat{u}_5},\{\hat{u}_1,\hat{u}_2,\hat{u}_6\}} \oplus W_{f_{\hat{u}_1},\mathcal{D}_{\hat{u}_1},\{\hat{u}_2,\hat{u}_4,\hat{u}_5\}} \oplus W_{f_{\hat{u}_4},\mathcal{D}_{\hat{u}_4},\{\hat{u}_1,\hat{u}_2,\hat{u}_5\}} \quad (166)$$
$$X_7 = W_{f_{\hat{u}_3},\mathcal{D}_{\hat{u}_3},\{\hat{u}_4,\hat{u}_5,\hat{u}_6\}} \oplus W_{f_{\hat{u}_4},\mathcal{D}_{\hat{u}_4},\{\hat{u}_1,\hat{u}_2,\hat{u}_6\}} \oplus W_{f_{\hat{u}_6},\mathcal{D}_{\hat{u}_6},\{\hat{u}_1,\hat{u}_2,\hat{u}_4\}} \quad (167)$$
$$X_8 = W_{f_{\hat{u}_6},\mathcal{D}_{\hat{u}_6},\{\hat{u}_1,\hat{u}_2,\hat{u}_3\}} \oplus W_{f_{\hat{u}_3},\mathcal{D}_{\hat{u}_3},\{\hat{u}_1,\hat{u}_4,\hat{u}_5\}} \oplus W_{f_{\hat{u}_1},\mathcal{D}_{\hat{u}_1},\{\hat{u}_3,\hat{u}_4,\hat{u}_5\}} \quad (168)$$
$$X_9 = W_{f_{\hat{u}_3},\mathcal{D}_{\hat{u}_3},\{\hat{u}_4,\hat{u}_5,\hat{u}_6\}} \oplus W_{f_{\hat{u}_6},\mathcal{D}_{\hat{u}_6},\{\hat{u}_1,\hat{u}_2,\hat{u}_3\}} \oplus W_{f_{\hat{u}_2},\mathcal{D}_{\hat{u}_2},\{\hat{u}_3,\hat{u}_5,\hat{u}_6\}} \oplus W_{f_{\hat{u}_5},\mathcal{D}_{\hat{u}_5},\{\hat{u}_2,\hat{u}_3,\hat{u}_6\}} \quad (169)$$

and we denote by $X = (X_i : i \in [9])$ the concatenated message sent by the central server, then each user can correctly decode its desired subfiles as follows.

- User $\hat{u}_1$ recovers its desired subfiles from $X_2$, $X_3$, $X_4 \oplus X_6$ and $X_8$.
- User $\hat{u}_2$ recovers its desired subfiles from $X_1$, $X_5$, $X_6$ and $X_7 \oplus X_9$.
- User $\hat{u}_3$ recovers its desired subfiles from $X_2 \oplus X_3$, $X_4$, $X_8$ and $X_9$.
- User $\hat{u}_4$ recovers its desired subfiles from $X_1$, $X_3$, $X_5 \oplus X_6$ and $X_7$.
- User $\hat{u}_5$ recovers its desired subfiles from $X_2$, $X_4$, $X_6$ and $X_8 \oplus X_9$.
- User $\hat{u}_6$ recovers its desired subfiles from $X_1 \oplus X_3$, $X_5$, $X_7$ and $X_9$.

The delivery procedure is slightly more involved with respect to the previous FDS structure, but is based as before on the idea of carefully aligning interference. Indeed, the messages $X_i$ are carefully designed in such a way that also their linear combinations can be useful to some users. An equivalent interpretation of this fact is related to the previously mentioned creation of cliques. Consider for example the XOR $X_3$. User 1 and user 4 can directly cache-out interference to correctly decode their desired subfiles, while user 3 and user 6 miss in their cache — due to the selfish cache

placement — some interfering messages appearing in the XOR $X_3$. Such interfering (and consequently undesired) messages are "delivered" to[21] both user 3 and user 6 by means of XORs $X_2$ and $X_1$, so allowing them to decode the desired subfiles from $X_3$. We can see here that with $X_3$ we are able serve the clique composed by user 1, user 4, user 3 and user 6, by carefully "passing" some undesired (and not cached) information to the last two users. A similar reasoning applies to the XORs $X_6$ and $X_9$, both of which are useful to 4 users simultaneously.

The communication load is equal to $R(3) = |X|/B = 9/10$ and it matches the converse $R_{\text{LB}}(3) = 9/10$. Hence, the converse here is tight.

## VII. Conclusion

In this work, we investigated the effects that selfish caching can have on the optimal worst-case communication load in the coded caching framework. The proposed FDS structure seeks to capture the degree of intersection between the interests of the different users. While somewhat restrictive, the proposed structure was designed to bring to the fore and accentuate the adversarial relationship between coded caching and selfish caching, and by doing so, to allow us to provide insight on the nature of this adversarial relationship.

This insight is provided with the introduction here of a new information-theoretic converse on the minimum worst-case communication load by means of index coding arguments. For the specific proposed broad FDS structure, the converse bound definitively resolves the question of whether selfish caching is beneficial or not. Indeed, the converse reveals that any non-zero load brought about by symmetrically selfish caching is always (with the exception of the extreme points of $t$) strictly worse than the optimal load guaranteed in the unselfish scenario. The rationale behind this is that, despite the sizeable increase of local caching gain brought about by the very targeted placement of selfish caching, and despite a very restricted set of demands, the loss in multicasting opportunities is too severe. In fact, what the converse shows is that this damage is so prominent that — for any fixed (or decreasing) ratio $\delta = \alpha/K < 1$ — the coding gain does not scale with $K$, and is in fact bounded above by $1/(1-\delta)$. In other words, even if there is, for example, a $99\%$ symmetric intersection between the interests of the users (meaning, even if the users have interest in almost the same content), the coding gain will not scale as $K$ increases. The above solidifies our conclusion that, for some powerful instances of FDS structures, selfish caching can be indeed very detrimental for the overall performance. While our FDS structure may be somewhat restrictive, the fact that this structure captures crucial elements of the selfish problem allows us to draw the conclusion that one must be cautious when considering selfish policies.

In light of the above, there are several interesting research directions. Firstly, one possibility is to study new, less restrictive FDS structures. One of the very few examples of FDS structures that can be defined for any number of users $K$ can be found in [4], [58]. Secondly, it would be interesting to explore the performance effect[22] of FDS symmetry in selfish approaches, and to search for FDS structures that benefit well from selfish coded schemes. This brings about the natural question of understanding when FDS structures can gain the most from selfish schemes. Clearly, a remaining challenge is to provide optimal schemes for any set of demands, either for the proposed or for another FDS structure.

[21] We recall that we use "deliver" in quotes when referring to undesired subfiles because users do not actually decode undesired subfiles, but rather they take linear combinations of the multicast messages to cancel out interference terms.

## Appendix A
## Proof of Corollary 1.1

Recalling that $R_{\text{MAN}}(t) = \binom{K}{t+1}/\binom{K}{t}$ as well as recalling the load expression for $R_{\text{LB}}(t)$ in Theorem 1, we have that

$$\frac{R^\star(t)}{R_{\text{MAN}}(t)} \geq \frac{R_{\text{LB}}(t)}{R_{\text{MAN}}(t)} \tag{170}$$

$$= \frac{\binom{\alpha}{t+1} + (K-\alpha)\binom{\alpha-1}{t}}{\binom{\alpha}{t}} \frac{\binom{K}{t}}{\binom{K}{t+1}} \tag{171}$$

$$= \frac{\binom{\alpha}{t+1}}{\binom{\alpha}{t}} \left(1 + (K-\alpha)\frac{t+1}{\alpha}\right) \frac{\binom{K}{t}}{\binom{K}{t+1}} \tag{172}$$

$$= \frac{\alpha - t}{K - t} \left(\frac{K(1+t) - \alpha t}{\alpha}\right) \tag{173}$$

$$= 1 + \underbrace{\frac{t(K-\alpha)(\alpha - 1 - t)}{\alpha(K-t)}}_{\geq 0} \tag{174}$$

where the second term in the last expression is equal to 0 either when $\alpha \in [K-1]$ for $t \in \{0, \alpha - 1\}$, or when $\alpha = K$ and $f \geq K$ for any $t$. This concludes the proof. □

## Appendix B
## Proof of Corollary 1.2

We know that the optimal coding gain is upper bounded as

$$G^\star \leq \frac{t+1}{1 + t(K-\alpha)/K} = \bar{G}(t). \tag{175}$$

It can be easily verified that $\bar{G}''(t) < 0$ for $t \geq 0$, which means that $\bar{G}(t)$ is concave for positive values of $t$. Then, we can see that $\bar{G}(0) = 1$, whereas

$$\lim_{t \to \infty} \bar{G}(t) = \frac{K}{K - \alpha} > 1 \tag{176}$$

for any $\alpha \in [K-1]$. Consequently, $\bar{G}(t) < K/(K-\alpha)$, which means that $G^\star < 1/(1-\delta)$ for any $\delta = \alpha/K$. This concludes the proof. □

[22] Considering for example the work in [4], the authors proposed therein a more lenient FDS structure, where each file is of interest to either groups of users or all users in the system. First of all, such setting represents an example of how the degree of separation among the users' interests is not concentrated in one single $\alpha$ value, since the model in [4] considers somehow two values $\alpha_1$ and $\alpha_2$, where $\alpha_1 = K$ relates to the part of library which is of interest to all users, while $\alpha_2 = 1$ relates to the subset of files which are of interest to disjoint (groups of) users. Second, for the FDS structure in [4] the authors proposed a selfish coded scheme (there referred to as Scheme 2) which can outperform the standard MAN scheme for some non-trivial memory points under some system parameters. Similarly, selfish policies were shown to be quite effective also for some instances of the system model in [58]. Hence, there exist FDS structures which can benefit from selfish policies when coded caching is adopted.

## Appendix C
## Proof of Lemma 2

Let $\hat{\boldsymbol{u}} = (\hat{u}_1, \dots, \hat{u}_K)$ be a permutation of the elements in $[K]$. Consider $k_1, k_2 \in [K]$ such that $k_1 \neq k_2$. Assume without loss of generality that $\hat{u}(k_1) < \hat{u}(k_2) \leq K$, in which case $\ell = \hat{u}(k_2) - \hat{u}(k_1)$. Denoting by $\mathcal{U}$ the set containing the $K$ circular shifts of the vector $\hat{\boldsymbol{u}}$, we see that there are $\ell$ vectors $\boldsymbol{u} \in \mathcal{U}$ such that $k_2$ appears before $k_1$ in $\boldsymbol{u}$. These cases correspond to the vectors $\boldsymbol{u} \in \mathcal{U}$ where we have in the first position of $\boldsymbol{u}$ either the element $k_2$, or any one of the $(\ell - 1)$ elements between $\hat{u}(k_1)$ and $\hat{u}(k_2)$ in the vector $\hat{\boldsymbol{u}}$. As a consequence, the total number of vectors $\boldsymbol{u} \in \mathcal{U}$ such that $k_1$ appears before $k_2$ is equal to $(K - \ell)$. This concludes the proof. □

## Appendix D
## Proof of Lemma 3

The convexity of $f(t)$ can be easily shown by verifying that the second derivative $f''(t)$ with respect to $t$ is strictly positive for $t \geq 0$. Indeed, we have

$$f(t) = \frac{\binom{\alpha}{t+1} + (K-\alpha)\binom{\alpha-1}{t}}{\binom{\alpha}{t}} \tag{177}$$

$$= \frac{\alpha - t}{1+t} + (K-\alpha)\left(1 - \frac{t}{\alpha}\right) \tag{178}$$

$$f'(t) = -\frac{1+\alpha}{(1+t)^2} - \frac{(K-\alpha)}{\alpha} \tag{179}$$

$$f''(t) = \frac{2(1+\alpha)}{(1+t)^3} > 0 \tag{180}$$

where $f'(t)$ denotes the first derivative. Then, since $t \in [0 : \alpha]$, we can evaluate $f(0) = K$ and $f(\alpha) = 0$, showing that $f(0) > f(\alpha)$. Hence, since $f(t)$ is convex, it has to be also strictly decreasing for $t \in [0 : \alpha]$, otherwise the convexity property would be violated. This concludes the proof. □

## Appendix E
## Converse Proof of Proposition 1

While the achievable expression matches exactly the converse expression $R_{\text{LB}}$, this latter converse cannot be used to prove the optimality of $R^{\star}_{\alpha,\text{c}}$, because $R_{\text{LB}}$ bounds the optimal *worst-case* communication load. As there is no a priori guarantee that the $\alpha$-demands are part of the worst-case demands, we will here derive another bound that focuses on $\alpha$-demands to prove that the achievable performance is indeed optimal.

Following the same line of reasoning as in Section IV-A, we apply again the index coding lower bound in Theorem 1, with the only difference being that now the cache placement is fixed. The corresponding index coding problem has $K' = K$ users and $N' = K\binom{\alpha-1}{t}$ messages, where $\binom{\alpha-1}{t}$ is the total number of subfiles desired by each user for a fixed value of $t \in [0 : \alpha]$. The desired message set and the side information set are respectively given by

$$\mathcal{M}_k = \{W_{f_k,\mathcal{D}_k,\mathcal{T}} : \mathcal{T} \subseteq \mathcal{D}_k, |\mathcal{T}| = t, k \notin \mathcal{T}\} \tag{181}$$

$$\mathcal{A}_k = \{W_{i,\mathcal{S},\mathcal{T}} : i \in [f], \mathcal{S} \subseteq [K], |\mathcal{S}| = \alpha, \mathcal{T} \subseteq \mathcal{S}, |\mathcal{T}| = t, k \in \mathcal{T}\} \tag{182}$$

for all $k \in [K]$. In the corresponding side information graph, an edge exists from $W_{f_{k_1},\mathcal{D}_{k_1},\mathcal{T}_1}$ to $W_{f_{k_2},\mathcal{D}_{k_2},\mathcal{T}_2}$ if and only if $W_{f_{k_1},\mathcal{D}_{k_1},\mathcal{T}_1} \in \mathcal{A}_{k_2}$.

Since we are considering a converse bound on the optimal communication load under a specific cache placement and under a specific set of demands, it suffices to find a single $\alpha$-demand such that $R^{\star}_{\alpha,\text{c}}(t) \geq \frac{\binom{\alpha}{t+1} + (K-\alpha)\binom{\alpha-1}{t}}{\binom{\alpha}{t}}$. Toward this, consider the $\alpha$-demand where $\mathcal{K}$ is a set of $\alpha$ users that request distinct files from the file class $\mathcal{K}$, and where the remaining users in $[K] \setminus \mathcal{K}$ request distinct files so that the set of vertices

$$\mathcal{J}_1 = \bigcup_{k \in ([K]\setminus\mathcal{K})} \bigcup_{\mathcal{T} \subseteq (\mathcal{D}_k \setminus \{k\}) : |\mathcal{T}| = t} \{W_{f_k,\mathcal{D}_k,\mathcal{T}}\} \tag{183}$$

is acyclic. Then, such set contains a total of $(K-\alpha)\binom{\alpha-1}{t}$ subfiles, which means that $|\mathcal{J}_1| = (K-\alpha)\binom{\alpha-1}{t}\frac{B}{\binom{\alpha}{t}}$. Indeed, we can see that there exist $\alpha$-demands for which $\mathcal{J}_1$ is acyclic. For instance, if we assume $\mathcal{D}_k = \mathcal{S} \cup \{k\}$ for every $k \in ([K] \setminus \mathcal{K})$ for some $\mathcal{S} \subseteq \mathcal{K}$ such that $|\mathcal{S}| = \alpha - 1$, then the set $\mathcal{J}_1$ is acyclic.

Consider now the set of users in $\mathcal{K}$. Take any permutation of users $\boldsymbol{u} = (u_1, \dots, u_\alpha)$ with $u_k \in \mathcal{K}$ for all $k \in [\alpha]$. Then, since $\mathcal{D}_k = \mathcal{K}$ for all $k \in \mathcal{K}$, the set of vertices

$$\mathcal{J}_2 = \bigcup_{k \in [\alpha]} \bigcup_{\mathcal{T} \subseteq (\mathcal{K} \setminus \{u_1,\dots,u_k\}) : |\mathcal{T}| = t} \{W_{f_{u_k},\mathcal{K},\mathcal{T}}\} \tag{184}$$

is acyclic for any permutation $\boldsymbol{u}$ (see [25, Lemma 1]). It can be easily seen that such set contains a total of $\binom{\alpha-1}{t} + \binom{\alpha-2}{t} + \cdots + \binom{t}{t} = \binom{\alpha}{t+1}$ subfiles, so $|\mathcal{J}_2| = \binom{\alpha}{t+1}\frac{B}{\binom{\alpha}{t}}$.

Due to the fact that $\mathcal{D}_k = \mathcal{K}$ for all $k \in \mathcal{K}$, there is no edge connecting any vertex in $\mathcal{J}_2$ to any vertex in $\mathcal{J}_1$, and thus also the set $\mathcal{J}_1 \cup \mathcal{J}_2$ is acyclic. At this point, applying Theorem 1 with respect to the acyclic set $\mathcal{J}_1 \cup \mathcal{J}_2$, we get

$$BR^{\star}_{\alpha,\text{c}} \geq \sum_{k \in ([K]\setminus\mathcal{K})} \sum_{\mathcal{T} \subseteq (\mathcal{D}_k \setminus \{k\}) : |\mathcal{T}| = t} |W_{f_k,\mathcal{D}_k,\mathcal{T}}| + \sum_{k \in [\alpha]} \sum_{\mathcal{T} \subseteq (\mathcal{K} \setminus \{u_1,\dots,u_k\}) : |\mathcal{T}| = t} \left|W_{f_{u_k},\mathcal{K},\mathcal{T}}\right| \tag{185}$$

$$= |\mathcal{J}_1| + |\mathcal{J}_2| \tag{186}$$

$$= (K-\alpha)\binom{\alpha-1}{t}\frac{B}{\binom{\alpha}{t}} + \binom{\alpha}{t+1}\frac{B}{\binom{\alpha}{t}} \tag{187}$$

which means that $R^{\star}_{\alpha,\text{c}}(t) \geq \frac{\binom{\alpha}{t+1} + (K-\alpha)\binom{\alpha-1}{t}}{\binom{\alpha}{t}}$. This concludes the proof. □

## References


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

[2] G. Zhang, Y. Li, and T. Lin, "Caching in information centric networking: A survey," *Comput. Netw.*, vol. 57, no. 16, pp. 3128–3141, 2013.

[3] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "FemtoCaching: Wireless content delivery through distributed caching helpers," *IEEE Trans. Inf. Theory*, vol. 59, no. 12, pp. 8402–8413, Dec. 2013.

[4] S. Wang and B. Peleato, "Coded caching with heterogeneous user profiles," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2019, pp. 2619–2623.

[5] C. Zhang and B. Peleato, "On the average rate for coded caching with heterogeneous user profiles," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2020, pp. 1–6.

[6] C.-H. Chang and C.-C. Wang, "Coded caching with heterogeneous file demand sets—The insufficiency of selfish coded caching," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2019, pp. 1–5.

[7] C.-H. Chang, C.-C. Wang, and B. Peleato, "On coded caching for two users with overlapping demand sets," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2020, pp. 1–6.

[8] D. Karamshuk, N. Sastry, M. Al-Bassam, A. Secker, and J. Chandaria, "Take-away TV: Recharging work commutes with predictive preloading of catch-up TV content," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 8, pp. 2091–2101, Aug. 2016.

[9] M.-C. Lee and A. F. Molisch, "Individual preference aware caching policy design in wireless D2D networks," *IEEE Trans. Wireless Commun.*, vol. 19, no. 8, pp. 5589–5604, Aug. 2020.

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

[11] S. P. Shariatpanahi, S. A. Motahari, and B. H. Khalaj, "Multi-server coded caching," *IEEE Trans. Inf. Theory*, vol. 62, no. 12, pp. 7253–7271, Dec. 2016.

[12] A. Tolli, S. P. Shariatpanahi, J. Kaleva, and B. H. Khalaj, "Multi-antenna interference management for coded caching," *IEEE Trans. Wireless Commun.*, vol. 19, no. 3, pp. 2091–2106, Mar. 2020.

[13] S. P. Shariatpanahi and B. H. Khalaj, "On multi-server coded caching in the low memory regime," 2018, *arXiv:1803.07655*.

[14] J. Zhang and P. Elia, "Fundamental limits of cache-aided wireless BC: Interplay of coded-caching and CSIT feedback," *IEEE Trans. Inf. Theory*, vol. 63, no. 5, pp. 3142–3160, May 2017.

[15] E. Lampiris, J. Zhang, and P. Elia, "Cache-aided cooperation with no CSIT," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2017, pp. 2960–2964.

[16] M. M. Amiri and D. Gündüz, "Cache-aided content delivery over erasure broadcast channels," *IEEE Trans. Commun.*, vol. 66, no. 1, pp. 370–381, Jan. 2018.

[17] S. Mohajer and I. Bergel, "MISO cache-aided communication with reduced subpacketization," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2020, pp. 1–6.

[18] I. Bergel and S. Mohajer, "Cache-aided communications with multiple antennas at finite SNR," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 8, pp. 1682–1691, Aug. 2018.

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

[20] S. P. Shariatpanahi, G. Caire, and B. H. Khalaj, "Physical-layer schemes for wireless coded caching," *IEEE Trans. Inf. Theory*, vol. 65, no. 5, pp. 2792–2807, May 2019.

[21] Y. Cao and M. Tao, "Degrees of freedom of cache-aided wireless cellular networks," *IEEE Trans. Commun.*, vol. 68, no. 5, pp. 2777–2792, May 2020.

[22] F. Engelmann and P. Elia, "A content-delivery protocol, exploiting the privacy benefits of coded caching," in *Proc. 15th Int. Symp. Modeling Optim. Mobile, Ad Hoc, Wireless Netw.*, May 2017, pp. 1–6.

[23] Q. Yan and D. Tuninetti, "Fundamental limits of caching for demand privacy against colluding users," *IEEE J. Sel. Areas Inf. Theory*, vol. 2, no. 1, pp. 192–207, Mar. 2021.

[24] K. Wan and G. Caire, "On coded caching with private demands," *IEEE Trans. Inf. Theory*, vol. 67, no. 1, pp. 358–372, Jan. 2021.

[25] K. Wan, D. Tuninetti, and P. Piantanida, "An index coding approach to caching with uncoded cache placement," *IEEE Trans. Inf. Theory*, vol. 66, no. 3, pp. 1318–1332, Jan. 2020.

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

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

[28] L. Tang and A. Ramamoorthy, "Coded caching schemes with reduced subpacketization from linear block codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 4, pp. 3099–3120, Apr. 2018.

[29] P. Krishnan, "Coded caching via line graphs of bipartite graphs," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Nov. 2018, pp. 1–5.

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

[31] E. Lampiris and P. Elia, "Adding transmitters dramatically boosts coded-caching gains for finite file sizes," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 6, pp. 1176–1188, Jun. 2018.

[32] A. M. Ibrahim, A. A. Zewail, and A. Yener, "Device-to-device coded-caching with distinct cache sizes," *IEEE Trans. Commun.*, vol. 68, no. 5, pp. 2748–2762, May 2020.

[33] E. Parrinello, A. Unsal, and P. Elia, "Fundamental limits of coded caching with multiple antennas, shared caches and uncoded prefetching," *IEEE Trans. Inf. Theory*, vol. 66, no. 4, pp. 2252–2268, Apr. 2020.

[34] E. Lampiris and P. Elia, "Full coded caching gains for cache-less users," *IEEE Trans. Inf. Theory*, vol. 66, no. 12, pp. 7635–7651, Dec. 2020.

[35] K. S. Reddy and N. Karamchandani, "Rate-memory trade-off for multi-access coded caching with uncoded placement," *IEEE Trans. Commun.*, vol. 68, no. 6, pp. 3261–3274, Jun. 2020.

[36] Y.-P. Wei and S. Ulukus, "Novel decentralized coded caching through coded prefetching," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Nov. 2017, pp. 1–5.

[37] D. Katyal, P. N. Muralidhar, and B. S. Rajan, "Multi-access coded caching schemes from cross resolvable designs," *IEEE Trans. Commun.*, vol. 69, no. 5, pp. 2997–3010, May 2021.

[38] B. Serbetci, E. Parrinello, and P. Elia, "Multi-access coded caching: Gains beyond cache-redundancy," in *Proc. IEEE Inf. Theory Workshop*, Aug. 2019, pp. 1–5.

[39] E. Lampiris, A. Bazco-Nogueras, and P. Elia, "Resolving the feedback bottleneck of multi-antenna coded caching," *IEEE Trans. Inf. Theory*, vol. 68, no. 4, pp. 2331–2348, Apr. 2022.

[40] H. Zhao, A. Bazco-Nogueras, and P. Elia, "Vector coded caching multiplicatively boosts the throughput of realistic downlink systems," 2022, *arXiv:2202.07047*.

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

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

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

[44] P. Quinton, S. Sahraei, and M. Gastpar, "A novel centralized strategy for coded caching with non-uniform demands," Jan. 2018, *arXiv:1801.10563*.

[45] E. Ozfatura and D. Gündüz, "Uncoded caching and cross-level coded delivery for non-uniform file popularity," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6.

[46] H. Ding and L. Ong, "An improved caching scheme for nonuniform demands and its optimal allocation," in *Proc. 3rd IEEE Int. Conf. Comput. Commun. (ICCC)*, Dec. 2017, pp. 389–393.

[47] S. A. Saberali, L. Lampe, and I. F. Blake, "Full characterization of optimal uncoded placement for the structured clique cover delivery of nonuniform demands," *IEEE Trans. Inf. Theory*, vol. 66, no. 1, pp. 633–648, Jan. 2020.

[48] H. Al-Lawati, N. Ferdinandy, and S. C. Draperz, "Coded caching with non-identical user demands," in *Proc. 15th Can. Workshop Inf. Theory (CWIT)*, Jun. 2017, pp. 1–5.

[49] Y. Deng and M. Dong, "Fundamental structure of optimal cache placement for coded caching with nonuniform demands," *IEEE Trans. Inf. Theory*, early access, Dec. 31, 2022, doi: 10.1109/TIT.2022.3179266.

[50] J. Hachem, N. Karamchandani, and S. N. Diggavi, "Coded caching for multi-level popularity and access," *IEEE Trans. Inf. Theory*, vol. 63, no. 5, pp. 3108–3141, May 2017.

[51] Y. Deng and M. Dong, "Memory-rate tradeoff for caching with uncoded placement under nonuniform random demands," 2021, *arXiv:2103.09925*.

[52] Y. Lu, C. Li, W. Chen, and H. V. Poor, "On the effective throughput of coded caching with heterogeneous user preferences: A game theoretic perspective," *IEEE Trans. Commun.*, vol. 69, no. 3, pp. 1387–1402, Mar. 2021.

[53] Z. Bar-Yossef, Y. Birk, T. S. Jayram, and T. Kol, "Index coding with side information," *IEEE Trans. Inf. Theory*, vol. 57, no. 3, pp. 1479–1494, Mar. 2011.

[54] F. Arbabjolfaei and Y.-H. Kim, "Fundamentals of index coding," *Found. Trends Commun. Inf. Theory*, vol. 14, nos. 3–4, pp. 163–346, 2018.

[55] C. Thapa, L. Ong, and S. J. Johnson, "Interlinked cycles for index coding: Generalizing cycles and cliques," *IEEE Trans. Inf. Theory*, vol. 63, no. 6, pp. 3692–3711, Jun. 2017.

[56] M. B. Vaddi and B. S. Rajan, "Optimal index codes for a new class of interlinked cycle structure," *IEEE Commun. Lett.*, vol. 22, no. 4, pp. 684–687, Apr. 2018.

[57] F. Arbabjolfaei, B. Bandemer, Y.-H. Kim, E. Sasoglu, and L. Wang, "On the capacity region for index coding," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2013, pp. 962–966.

[58] K. Wan, M. Cheng, M. Kobayashi, and G. Caire, "On the optimal memory-load tradeoff of coded caching for location-based content," *IEEE Trans. Commun.*, vol. 70, no. 5, pp. 3047–3062, May 2022.

**Federico Brunero** was born in Torino, Italy, in 1995. He received the B.Sc. degree in telecommunications engineering and the M.Sc. degree in communications and computer networks engineering from the Politecnico di Torino in 2017 and 2019, respectively, the M.Sc. degree in electrical and computer engineering from the University of Illinois Chicago (UIC) in 2019, and the M.Sc. degree in data science and engineering from the EURECOM, Institut Mines-Télécom, in 2021. He is currently pursuing the Ph.D. degree with the Communication Systems Department, EURECOM, Sorbonne Université. His research interests include information theory, coding theory, and machine learning.

**Petros Elia** (Member, IEEE) received the B.Sc. degree from the Illinois Institute of Technology and the M.Sc. and Ph.D. degrees in electrical engineering from the University of Southern California (USC), Los Angeles, in 2001 and 2006, respectively. He is currently a Professor with the Department of Communication Systems, EURECOM, Sophia Antipolis, France. His latest research deals with distributed computing and with the intersection of caching and communications in multiuser settings. He has also worked in the area of complexity-constrained communications, MIMO, queueing theory and cross-layer design, coding theory, information-theoretic limits in cooperative communications, and surveillance networks. He is a fulbright scholar, a co-recipient of the NEWCOM++ Distinguished Achievement Award from 2008 to 2011 for a sequence of publications on the topic of complexity in wireless communications, and a recipient of the ERC Consolidator Grant from 2017 to 2022 on cache-aided wireless communications.