

Decentralized Coded Caching Attains Order-Optimal Memory-Rate Tradeoff

Mohammad Ali Maddah-Ali and Urs Niesen

Abstract

Replicating or caching popular content in memories distributed across the network is a technique to reduce peak network loads. Conventionally, the main performance gain of this caching was thought to result from making part of the requested data available closer to end users. Instead, we recently showed that a much more significant gain can be achieved by using caches to create coded-multicasting opportunities, even for users with different demands, through coding across data streams. These coded-multicasting opportunities are enabled by careful content overlap at the various caches in the network, created by a central coordinating server.

In many scenarios, such a central coordinating server may not be available, raising the question if this multicasting gain can still be achieved in a more decentralized setting. In this paper, we propose an efficient caching scheme, in which the content placement is performed in a decentralized manner. In other words, no coordination is required for the content placement. Despite this lack of coordination, the proposed scheme is nevertheless able to create coded-multicasting opportunities and achieves a rate close to the optimal centralized scheme.

I. INTRODUCTION

Traffic in content delivery networks exhibits strong temporal variability, resulting in congestion during peak hours and resource underutilization during off-peak hours. It is therefore desirable to try to “shift” some of the traffic from peak to off-peak hours. One approach to achieve this is to exploit idle network resources to duplicate some of the content in memories distributed across the network. This duplication of content is called content placement or caching. The duplicated content can then be used during peak hours to reduce network congestion.

From the above description, it is apparent that the network operates in two different phases: a content placement phase and a content delivery phase. In the placement phase, the network is not congested, and the system is constrained mainly by the size of the cache memories. In the delivery phase, the network is congested, and the system is constrained mainly by the rate required to serve the content requested by the users. The goal is thus to design the placement phase such that the rate in the delivery phase is minimized.

There are two fundamentally different approaches, based on two distinct understandings of the role of caching, for how the placement and the delivery phases are performed.

- *Providing Content Locally:* In the first, conventional, caching approach, replication is used to make part of the requested content available close to the end users. If a user finds part of a requested file in a close-by cache memory, that part can be served locally. The central content server only sends the remaining file parts using simple orthogonal unicast transmissions. If more than one user requests the same file, then the server has the option to multicast a single stream to those users. Extensive research has been done on this conventional caching approach, mainly on how to exploit differing file popularities to maximize the caching gain [1]–[7]. The gain of this approach is proportional to the fraction of the popular content that can be stored locally. As a result, this conventional caching approach is effective whenever the local cache memory is large enough to store a significant fraction of the total popular content.

The authors are with Bell Labs, Alcatel-Lucent. Emails: {mohammadali.maddah-ali, urs.niesen}@alcatel-lucent.com

The material in this paper has been presented in part at the 51st Annual Allerton Conference on Communication, Control, and Computing, Oct. 2013.

- *Creating Simultaneous Coded-Multicasting Opportunities:* In this approach, which we recently proposed in [8], content is placed in order to allow the central server to satisfy the requests of several users with *different* demands with a *single* multicast stream. The multicast streams are generated by coding across the different files requested by the users. Each user exploits the content stored in the local cache memory to enable decoding of its requested file from these data streams. Since the content placement is performed before the actual user demands are known, it has to be designed carefully such that these coded-multicasting opportunities are available simultaneously for all possible requests. In [8], we show that this simultaneous coded-multicasting gain can significantly reduce network congestion. Moreover, for many situations, this approach results in a much larger caching gain than the one obtained from the conventional caching approach discussed above. Unlike the conventional approach, the simultaneous coded-multicast approach is effective whenever the aggregate *global* cache size (i.e., the cumulative cache available at all users) is large enough compared to the total amount of popular content, even though there is no cooperation among the caches.

As mentioned above, the scheme proposed in [8], relies on a carefully designed placement phase in order to create coded-multicasting opportunities among users with different demands. A central server arranges the caches such that every subset of the cache memories shares a specific part of the content. It is this carefully arranged overlap among the cache memories that guarantees the coded-multicasting opportunities simultaneously for all possible user demands.

While the assumption of a centrally coordinated placement phase was helpful to establish the new caching approach in [8], it limits its applicability. For example, the identity or even just the number of active users in the delivery phase may not be known several hours in advance during the placement phase. As another example, in some cases the placement phase could be performed in one network, say a WiFi network, to reduce congestion in the delivery phase in another network, say a cellular network. In either case, coordination in the placement phase may not be possible.

This raises the important question whether lack of coordination in the placement phase eliminates the significant rate reduction promised by the simultaneous coded-multicast approach proposed in [8]. Put differently, the question is if simultaneous coded-multicasting opportunities can still be created without a centrally coordinated placement phase.

In this paper, we answer this question in the positive by developing a caching algorithm that creates simultaneous coded-multicasting opportunities without coordination in the placement phase. More precisely, the proposed algorithm is able to operate in the placement phase with an unknown number of users situated in isolated networks and acting independently from each other. Thus, the placement phase of the proposed algorithm is *decentralized*. In the delivery phase, some of these users are connected to a server through a shared bottleneck link. In this phase, the server is first informed about the set of active users, their cache contents, and their requests. The proposed algorithm efficiently exploits the multicasting opportunities created during the placement phase in order to minimize the rate over the shared bottleneck link. We show that our proposed decentralized algorithm can significantly improve upon the conventional uncoded scheme. Moreover, we show that the performance of the proposed decentralized coded caching scheme is close to the performance of the centralized coded scheme of [8].

These two claims are illustrated in Fig. 1 for a system with 20 users and 100 pieces of content. For example, when each user is able to cache 25 of the files, the peak rate of the conventional uncoded scheme is equivalent to transmitting 15 files. However, in the proposed decentralized coded scheme, the peak rate is equivalent to transmitting only about 3 files. By comparing this to the performance of the centralized coded scheme, we can see that the rate penalty for decentralization of the placement phase of the caching system is modest.

The remainder of this paper is organized as follows. Section II formally introduces the problem setting. Section III presents the proposed algorithm. In Section IV, the performance of the proposed algorithm is evaluated and compared with the uncoded and the centralized coded caching schemes. In Section V, the results are extended to other topologies of practical interest. Section VI discusses various implications of the results.

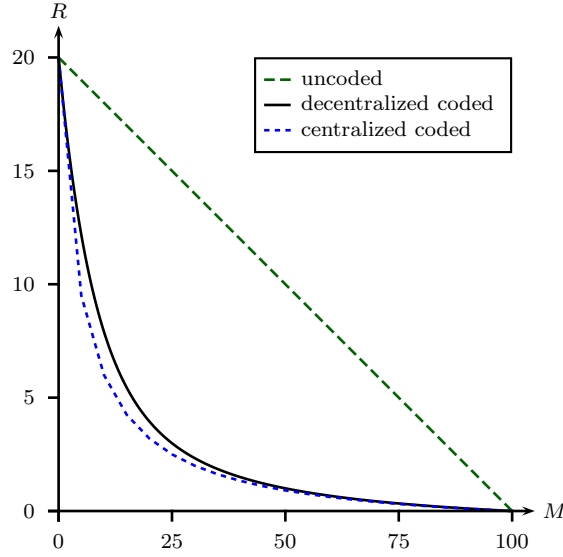


Fig. 1. Performance of different caching schemes for a system with 20 users connected to a server storing 100 files through a shared bottleneck link. The horizontal axis is the size of the cache memory (normalized by the file size) at each user; the vertical axis shows the peak rate (again normalized by the file size) over the shared link in the delivery phase. The dashed green curve depicts the rate achieved by the conventional uncoded caching scheme advocated in the prior literature. The solid black curve depicts the rate achieved by the decentralized coded caching scheme proposed in this paper. The dashed blue curve depicts the rate achieved by the centralized coded caching algorithm from the recent paper [8].

II. PROBLEM SETTING

To gain insight into how to optimally operate content-distribution systems, we introduce here a basic model for such systems capturing the fundamental challenges, tensions, and tradeoffs in the caching problem. For the sake of clarity, we initially study the problem under some simplifying assumptions, which will be relaxed later, as is discussed in detail in Sections V and VI.

We consider a content-distribution system consisting of a server connected through an error-free¹ shared (bottleneck) link to K users. The server stores N files each of size F bits. The users each have access to a cache able to store MF bits for $M \in [0, N]$. This scenario is illustrated in Fig. 2.

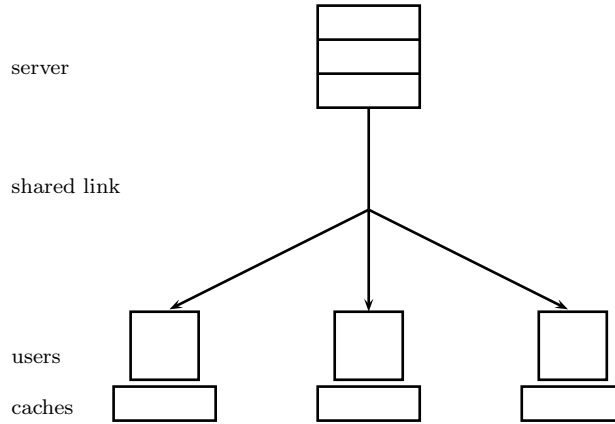


Fig. 2. Caching system considered in this paper. A server containing N files of size F bits each is connected through a shared link to K users each with a cache of size MF bits. In the figure, $N = K = 3$ and $M = 1$.

The system operates in two phases: a *placement* phase and a *delivery* phase. The placement phase

¹Any errors in this link have presumably been already taken care of using error correction coding.

occurs when the network load is low. During this time, the shared link can be utilized to fill the caches of the users. The main constraint in this phase is the size of the cache memory at each user. The delivery phase occurs after the placement phase when the network load is high. At this time, each user requests one file from the server, which proceeds to transmit its response over the shared link. Given the output of the shared link (observed by all users) and its cache content, each user should be able to recover its requested file. The main constraint in this phase is the load of the shared link. The objective is to minimize the worst-case (over all possible requests) load of the shared link in the delivery phase.

We now formalize this problem description. In the placement phase, each user is able to fill its cache as an arbitrary function (linear, nonlinear, ...) of the N files subject only to its memory constraint of MF bits with $M \in [0, N]$. We emphasize that the requests of the users are not known during the placement phase, and hence the caching function is not allowed to depend on them.

In the delivery phase, each of the K users requests one of the N files and communicates this request to the server. Let $d_k \in \{1, \dots, N\}$ be the request of user $k \in \{1, \dots, K\}$. The server replies to these requests by sending a message over the shared link, which is observed by all the K users. Let $R^{(d_1, \dots, d_K)} F$ be the number of bits in the message sent by the server. We impose that each user is able to recover its requested file from the content of its cache and the message received over the shared link with probability arbitrary close to one for large enough file size F . Denote by

$$R \triangleq \max_{d_1, \dots, d_K} R^{(d_1, \dots, d_K)}$$

the worst-case normalized rate for a caching scheme.

Our objective is to minimize the rate R in order to minimize the worst-case network load RF during the delivery phase. Clearly, R is a function of the cache size MF . In order to emphasize this dependence, we will usually write the rate as $R(M)$. The function $R(M)$ expresses the *memory-rate tradeoff* of the content-distribution system.

The following example illustrates the definitions and notations and introduces the uncoded caching approach advocated in most of the prior literature. This uncoded caching scheme will be used as a benchmark throughout the paper.

Example 1 (Uncoded Caching). Consider the caching problem with $N = 2$ files and $K = 2$ users each with a cache of size $M = 1$. In the uncoded caching scheme, each of the two files A and B are split into two parts of equal size, namely $A = (A_1, A_2)$ and $B = (B_1, B_2)$. In the placement phase, both users cache (A_1, B_1) , i.e., the first part of each file. Since each of these parts has size $F/2$, this satisfies the memory constraint of $MF = F$ bits.

Consider now the delivery phase of the system. Assume that each user requests the same file A , i.e., $d_1 = d_2 = 1$. The server responds by sending the file part A_2 of size $F/2$ bits. Clearly, from their cache content and the message received over the shared link, each user can recover the requested file $A = (A_1, A_2)$. The (normalized) rate in the delivery phase is $R^{(1,1)} = 1/2$.

Assume instead that user one requests file A and user two requests file B , i.e., $d_1 = 1$ and $d_2 = 2$. The server needs to transmit (A_2, B_2) of size F bits to satisfy these requests, resulting in a rate in the delivery phase of $R^{(1,2)} = 1$. It is easy to see that this is the worst-case request, and hence $R = 1$ for this scheme.

For general N , K , and M , the uncoded scheme caches the first M/N fraction of each of the N files. Therefore, in the delivery phase, the server has to send the remaining $1 - M/N$ fraction of the requested files. The resulting rate in the delivery phase, denoted by $R_U(M)$ for future reference, is

$$R_U(M) \triangleq K \cdot (1 - M/N) \cdot \min\{1, N/K\}.$$

For $N = K = 2$ and $M = 1$, this yields $R_U(1) = 1$, as before.

As we will see, this conventional caching scheme can be significantly improved upon. In particular, see Example 2 in Section III. \diamond

One important feature of the uncoded scheme introduced in Example 1 is that it has a *decentralized* placement phase. By that we mean that the cache of each user is filled independently of other users. In

particular, the placement operation of a given user neither depends on the identity nor the number of other users in the system. As a result, the users could, in fact, contact different servers at different times for the placement phase. Having a decentralized placement phase is thus an important robustness property for a caching system. This is discussed further in Sections III and V.

As was mentioned earlier, the system description introduced in this section makes certain simplifying assumptions. In particular, we assume a system having a single shared broadcast link, with a cache at each user, and we focus on worst-case demands, with synchronized user requests in the delivery phase. All these assumptions can be relaxed, as is discussed in Sections V and VI.

III. A DECENTRALIZED CODED CACHING ALGORITHM

We now present a new algorithm (referred to as decentralized coded caching in the following) for the caching problem. In the statement of the algorithm, we use the notation $V_{k,S}$ to denote the bits of the file d_k requested by user k cached exclusively at users in S . In other words, a bit of file d_k is in $V_{k,S}$ if it is present in the cache of every user in S and if it is absent from the cache of every user outside S . We also use the notation $[K] \triangleq \{1, 2, \dots, K\}$ and $[N] \triangleq \{1, 2, \dots, N\}$.

The proposed algorithm consists of a placement procedure and two delivery procedures. In the placement phase, we always use the same placement procedure. In the delivery phase, the server chooses the delivery procedure minimizing the resulting rate over the shared link.

Algorithm 1 Decentralized Coded Caching

```

1: procedure PLACEMENT
2:   for  $k \in [K], n \in [N]$  do
3:     user  $k$  independently caches a subset of  $\frac{MF}{N}$  bits of file  $n$ , chosen uniformly at random
4:   end for
5: end procedure

6: procedure DELIVERY( $d_1, \dots, d_K$ )
7:   for  $s = K, K-1, \dots, 1$  do
8:     for  $S \subset [K] : |S| = s$  do
9:       server sends  $\bigoplus_{k \in S} V_{k, S \setminus \{k\}}$ 
10:    end for
11:  end for
12: end procedure

13: procedure DELIVERY'( $d_1, \dots, d_K$ )
14:   for  $n \in [N]$  do
15:     server sends enough random linear combinations of bits in file  $n$  for all users requesting it to decode
16:   end for
17: end procedure

```

Remark 1: The \oplus operation in Line 9 of Algorithm 1 represents the bit-wise XOR operation. All elements $V_{k, S \setminus \{k\}}$ are assumed to be zero padded to the length of the longest element.

We illustrate the Algorithm 1 with a small example.

Example 2 (Decentralized Coded Caching). Consider the caching problem with $N = 2$ files A and B , and $K = 2$ users each with a cache of size $M \in [0, 2]$. In the placement phase of Algorithm 1, each user caches a subset of $MF/2$ bits of each file independently at random, satisfying the memory constraint. As a result, each bit of a file is cached by a specific user with probability $M/2$.

Let us focus on file A . The actions of the placement procedure effectively partition file A into 4 subfiles,

$$A = (A_\emptyset, A_1, A_2, A_{1,2}),$$

where, for $\mathcal{S} \subset \{1, 2\}$, $A_{\mathcal{S}}$ denotes the bits of file A that are stored in the cache memories of users in \mathcal{S} . For example, $A_{1,2}$ are the bits of A available in the cache memories of users one and two, whereas A_1 are the bits of A available exclusively in the cache memory of user one.²

By the law of large numbers,

$$|A_{\mathcal{S}}| \approx (M/2)^{|\mathcal{S}|}(1 - M/2)^{2-|\mathcal{S}|}F$$

with probability approaching one for large enough file size F . Therefore, we have with high probability:

- $|A_\emptyset|/F$ is approximately $(1 - M/2)^2$.
- $|A_1|/F$ and $|A_2|/F$ are approximately $(M/2)(1 - M/2)$.
- $|A_{1,2}|/F$ is approximately $(M/2)^2$.

The same analysis holds for file B .

We now consider the delivery phase in Algorithm 1. As we will see later (see Remark 6 below), for the scenario at hand the first delivery procedure will be used. Assume that user one requests file A and user two requests file B .

The iteration in Line 7 of Algorithm 1 starts with $s = 2$. By Line 8, this implies that we consider the set $\mathcal{S} = \{1, 2\}$. Observe that:

- The cache of user two contains A_2 , which is needed by user one. Hence, $V_{1,2} = A_2$.
- The cache of users one contains B_1 , which is needed by user two. Hence, $V_{2,1} = B_1$.

As a result, in Line 9 of Algorithm 1, the server transmits $A_2 \oplus B_1$ over the shared link. User one can solve for A_2 from the received message $A_2 \oplus B_1$ and the cached subfile B_1 . User two can solve for B_1 from the message $A_2 \oplus B_1$ and the cached subfile A_2 . Therefore, $A_2 \oplus B_1$ is simultaneously useful for $s = 2$ users. Thus, even though the two users request different files, the server can successfully multicast useful information to both of them. We note that the normalized (by F) size of $A_2 \oplus B_1$ is $(M/2)(1 - M/2)$.

The second iteration in Line 7 is for $s = 1$. In this iteration, the server simply sends $V_{1,\emptyset} = A_\emptyset$ and $V_{2,\emptyset} = B_\emptyset$ in Line 8. Each of these transmissions is useful for $s = 1$ user and has normalized size $(1 - M/2)^2$.

From A_2 computed in iteration one, A_\emptyset received in iteration two, and its cache content $(A_1, A_{1,2})$, user one can recover the requested file $A = (A_\emptyset, A_1, A_2, A_{1,2})$. Similarly, user two can recover the requested file B .

Summing up the contributions for $s = 2$ and $s = 1$, the aggregate size (normalized by F) of the messages sent by the server is

$$(M/2)(1 - M/2) + 2(1 - M/2)^2.$$

This can be rewritten as

$$2 \cdot (1 - M/2) \cdot \frac{1}{M}(1 - (1 - M/2)^2).$$

In particular, for $M = 1$, the rate of Algorithm 1 is $3/4$.

This compares to a rate of $R_U(1) = 1$ achieved by the uncoded caching scheme described in Example 1 in Section II. While the improvement in this scenario is relatively small, as we will see shortly, for larger values of N and K , this improvement over the uncoded scheme can be large. \diamond

Remark 2 (Unknown Number of Users during Placement Phase): The placement procedure of Algorithm 1 is *decentralized*, in the sense that the user's caches are filled independently of each other. This implies that neither the identity nor even the number of users sharing the same bottleneck link during the delivery phase need to be known during the earlier placement phase.

²To avoid heavy notation, we write $A_{1,2}$ as shorthand for $A_{\{1,2\}}$. Similarly, we write $V_{1,2}$ for $V_{1,\{2\}}$.

This decentralization of the placement phase enables the content-distribution system to be much more flexible than a centralized placement phase. This flexibility is essential. For example, in wireline networks, some of the users may not request any file in the delivery phase. In wireless networks, users may move from one network or cell to another, and hence might not even be connected to the same server in the two phases. In either case, the result is that the precise number and identity of users in the delivery phase is unknown in the placement phase. One of the salient features of the decentralized algorithm proposed in this paper is that it can easily deal with these situations.

This flexibility is also crucial to deal with asynchronous user requests, as is explained in detail in Section V-C. It is also a key ingredient to extending the coded-caching approach to scenarios with nonuniform demands or with online cache updates, as is discussed further in Section VI and in the follow-up works [9] and [10].

Remark 3 (Greedy Coding Strategy): The first delivery procedure in Algorithm 1 follows a *greedy* strategy. It first identifies and forms coded messages that are useful for all $s = K$ users. In the next iteration, it forms coded messages that are useful for subsets of $s = K - 1$ users. The iteration continues until it identifies messages that are useful for only $s = 1$ user.

Remark 4 (Simplified Decision Rule): Algorithm 1 provides two delivery procedures. The general rule is to choose the procedure which minimizes the resulting rate over the shared link. A simple alternative rule to decide between these two procedures is as follows: if $M > 1$, employ the first procedure; otherwise, employ the second procedure. The performance loss due to this simpler rule can be shown to be small.³

Remark 5 (Knowledge of Cache Contents): The delivery procedure in Algorithm 1 assumes that the server knows which bits are cached at each user. In practice, each user will choose which bits to cache using a random number generator. By communicating only the seed value of this random number generator from the user back to the server, the server can reconstruct the cache contents of the user.

IV. PERFORMANCE ANALYSIS

We now analyze the performance of the proposed decentralized coded caching scheme given by Algorithm 1. Section IV-A provides an analytic expression for the rate of Algorithm 1. Section IV-B compares the proposed decentralized coded caching scheme with the decentralized uncoded caching scheme from Example 1 (the best previously known decentralized caching scheme). Section IV-C compares the proposed decentralized coded caching scheme with the optimal centralized caching scheme and the caching scheme from [8] (the best known centralized caching scheme).

A. Rate of Decentralized Coded Caching Scheme

The performance of decentralized coded caching is analyzed in the next theorem, whose proof can be found in Appendix A.

Theorem 1. *Consider the caching problem with N files each of size F bits and with K users each having access to a cache of size MF bits with $M \in (0, N]$. Algorithm 1 is correct and, for F large enough, achieves rate arbitrarily close to*

$$R_D(M) \triangleq K \cdot (1 - M/N) \cdot \min \left\{ \frac{N}{KM} (1 - (1 - M/N)^K), \frac{N}{K} \right\}.$$

Remark 6: We note that if $N \geq K$ or $M \geq 1$, then the minimum in $R_D(M)$ is achieved by the first term so that

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

³In fact, the achievable rate with this simpler rule is still within a constant factor of the optimal centralized memory-rate tradeoff. This follows from the proof of Theorem 2 with some minor modifications.

This is the rate of the first delivery procedure in Algorithm 1. Since $N \geq K$ or $M \geq 1$ is the regime of most interest, the majority of the discussion in the following focuses on this case.

Remark 7: Theorem 1 is only stated for $M > 0$. For $M = 0$ Algorithm 1 is easily seen to achieve a rate of

$$R_D(0) \triangleq \min\{N, K\}.$$

We see that $R_D(0)$ is the continuous extension of $R_D(M)$ for $M > 0$. To simplify the exposition, we will not treat the case $M = 0$ separately in the following.

The rate $R_D(M)$ of Algorithm 1 consists of three distinct factors. The first factor is K ; this is the rate without caching. The second factor is $(1 - M/N)$; this is a *local* caching gain that results from having part of the requested file already available in the local cache. The third factor is a *global* gain that arises from using the caches to create simultaneous coded-multicasting opportunities. See Example 2 in Section III for an illustration of the operational meaning of these three factors.

B. Comparison with Decentralized Uncoded Caching Scheme

It is instructive to examine the performance of the proposed decentralized coded caching scheme (Algorithm 1) for large and small values of cache size M . For simplicity, we focus here on the most relevant case $N \geq K$, i.e., the number of files is at least as large as the number of users, so that the rate $R_D(M)$ of Algorithm 1 is given by Remark 6.

As a baseline, we compare the result with the uncoded caching scheme, introduced in Example 1 in Section II. This is the best previously known algorithm with a decentralized placement phase. For $N \geq K$, the uncoded scheme achieves the rate

$$R_U(M) = K \cdot (1 - M/N), \quad (1)$$

which is linear with slope of $-K/N$ throughout the entire range of M .

1) *Small M :* For small cache size $M \in [0, N/K]$, the rate achieved by Algorithm 1 behaves approximately⁴ as

$$R_D(M) \approx K \cdot \left(1 - \frac{KM}{2N}\right). \quad (2)$$

In this regime, $R_D(M)$ scales approximately linearly with the memory size M with slope $-K^2/(2N)$: increasing M by one decreases the rate by approximately $K^2/(2N)$. This is illustrated in Fig. 3.

Comparing (1) and (2), we have the following observations:

- *Order- K Improvement in Slope:* The slope of $R_D(M)$ around $M = 0$ is approximately $K/2$ times steeper than the slope of $R_U(M)$. Thus, the reduction in rate as a function of M is approximately $K/2$ times faster for Algorithm 1 than for the uncoded scheme. In other words, for small M the scheme proposed here makes approximately $K/2$ times better use of the cache resources: an improvement on the order of the number of users in the system. This behavior is clearly visible in Fig. 1 in Section I.
- *Virtual Shared Cache:* Consider a hypothetical setting in which the K cache memories are collocated and shared among all K users. In this hypothetical system, arising from allowing complete cooperation among the K users, it is easy to see that the optimal rate is $K \cdot (1 - KM/N)$. Comparing this to (2), we see that, up to a factor 2, the proposed decentralized coded caching scheme achieves the same order behavior. Therefore, this scheme is essentially able to create a single virtually shared cache, even though the caches are isolated without any cooperation between them.

⁴More precisely, $R_D(M) = K - \frac{K(K+1)}{2N}M + O(M^2)$, and by analyzing the constant in the $O(M^2)$ expression it can be shown that this is a good approximation in the regime $M \in [0, N/K]$.

2) *Large M* : On the other hand, for $M \in [N/K, N]$ we can approximate⁵

$$R_D(M) \approx N/M - 1. \quad (3)$$

In this regime, the rate achieved by Algorithm 1 scales approximately inversely with the memory size: doubling M approximately halves the rate. This is again illustrated in Fig. 3.

Comparing (1) and (3), we have the following observation:

- *Order- K Improvement in Rate*: In this regime, the rate of the proposed decentralized coded scheme can be up to a factor K smaller than the uncoded scheme: again an improvement on the order of the number of users in the system. This behavior is again clearly visible in Fig. 1 in Section I.

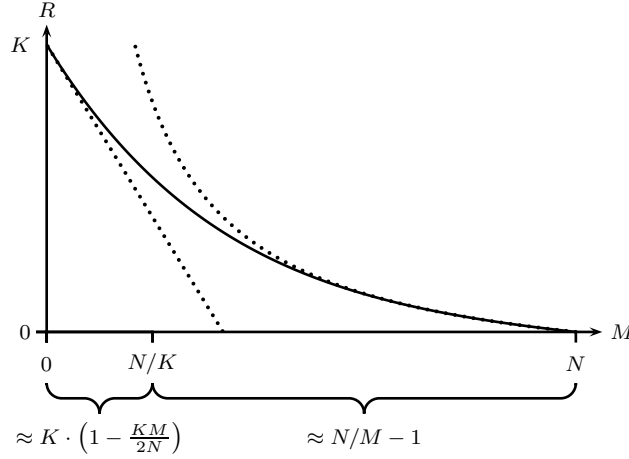


Fig. 3. Memory-rate tradeoff $R_D(M)$ achieved by Algorithm 1 for $N = 100$ files and $K = 5$ users (see Theorem 1). The function $R_D(M)$ behaves approximately linearly for $M \in [0, N/K]$ and behaves approximately as $N/M - 1$ for $M \in [N/K, N]$ (both approximations are indicated by dotted curves).

C. Comparison with Centralized Coded Caching Scheme

We have compared the performance of the proposed decentralized coded caching scheme to the uncoded caching scheme, which is the best previously known *decentralized* algorithm for this setting. We now compare the decentralized coded caching scheme to the rate achievable by *centralized* caching schemes. We start with an information-theoretic lower bound on the rate of *any* centralized caching scheme. We then consider the rate of the best known centralized caching scheme recently introduced in [8].

Theorem 2. Let $R_D(M)$ be the rate of the decentralized coded caching scheme given in Algorithm 1, and let $R^*(M)$ be the rate of the optimal centralized caching scheme. For any number of files N and number of users K and for any $M \in [0, N]$, we have

$$\frac{R_D(M)}{R^*(M)} \leq 12.$$

The proof of Theorem 2, presented in Appendix B, uses an information-theoretic argument to lower bound the rate of the optimal scheme $R^*(M)$. As a result, Theorem 2 implies that *no* scheme (centralized, decentralized, with linear caching, nonlinear caching, ...) regardless of its computational complexity can improve by more than a constant factor upon the efficient decentralized caching scheme given by Algorithm 1 presented in this paper.

⁵Since $(1 - M/N)^K \leq (1 - 1/K)^K \leq 1/e$, we have $R_D(M) = \Theta(N/M - 1)$ in the regime $M \in [N/K, N]$, and the pre-constant in the order notation converges to 1 as $M \rightarrow N$.

Remark 8 (Optimality of Uncoded Prefetching and Linearly-Coded Delivery): Theorem 2 implies that uncoded caching in the placement phase combined with greedy linear coding in the delivery phase is sufficient to achieve a rate within a constant factor of the optimum.

We now compare the rate $R_D(M)$ of decentralized coded caching to the rate $R_C(M)$ of the best known centralized coded caching scheme. In [8, Theorem 2], $R_C(M)$ is given by

$$R_C(M) \triangleq K \cdot (1 - M/N) \cdot \min \left\{ \frac{1}{1 + KM/N}, \frac{N}{K} \right\}$$

for $M \in \frac{N}{K}\{0, 1, \dots, K\}$, and the lower convex envelope of these points for all remaining values of $M \in [0, N]$. Fig. 1 in Section I compares the performance $R_C(M)$ of this centralized coded caching scheme to the performance $R_D(M)$ of the decentralized coded caching scheme proposed here. As can be seen from the figure, the centralized and decentralized caching algorithms are very close in performance. Thus, there is only a small price to be paid for decentralization. Indeed, we have the following corollary to Theorem 2.

Corollary 3. *Let $R_D(M)$ be the rate of the decentralized coded caching scheme given in Algorithm 1, and let $R_C(M)$ be the rate of the centralized coded caching scheme from [8]. For any number of files N and number of users K and for any $M \in [0, N]$, we have*

$$\frac{R_D(M)}{R_C(M)} \leq 12.$$

Corollary 3 shows that the rate achieved by the decentralized coded caching scheme given by Algorithm 1 is at most a factor 12 worse than the one of the best known centralized algorithm from [8]. This bound can be tightened numerically to

$$\frac{R_D(M)}{R_C(M)} \leq 1.6$$

for all values of K , N , and M . Hence, the rate of the decentralized caching scheme proposed here is indeed quite close to the rate of the best known centralized caching scheme.

It is instructive to understand why the decentralized scheme performs close to the centralized one. In the centralized scheme, content is placed in the placement phase such that in the delivery phase every message is useful for exactly $1 + KM/N$ users. In the decentralized scheme, we cannot control the placement phase as accurately. However, perhaps surprisingly, the number of messages that are useful for about $1 + KM/N$ users is nevertheless the dominant term in the overall rate of the decentralized scheme.

More precisely, from the proof of Theorem 1 in Appendix A, we can write the rate $R_D(M)$ of the decentralized coded scheme as a convex combination of the rate $R_C(M)$ of the centralized coded scheme:

$$R_D(M) = \sum_{s=0}^K R_C(sN/K) p(s) \quad (4)$$

with $p(s) \geq 0$ and $\sum_{s=0}^K p(s) = 1$. The dominant term in this sum is $R_C(M)$ occurring at $s = KM/N$, as is illustrated in Fig. 4. This observation explains why the centralized and the decentralized schemes have approximately the same rate.

V. EXTENSIONS

In this section, we extend the results presented so far to some important cases arising in practical systems. In particular, we show how to handle networks with tree topologies in Section V-A, caches shared by several users in Section V-B, and users with asynchronous requests in Section V-C.

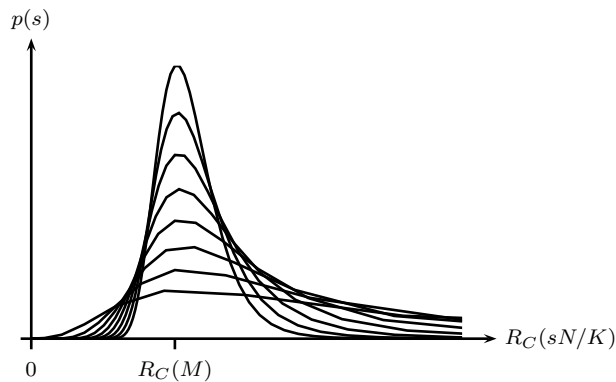


Fig. 4. Concentration of the rate terms in the convex combination (4) expressing the rate of the decentralized coded caching scheme $R_D(M)$ around the rate $R_C(M)$ of the centralized coded caching scheme. The curves are for different values of $N \in \{2^3, 2^4, \dots, 2^{10}\}$ with $K = N$ and $M = \sqrt{N}$. Each curve depicts $p(s)$ versus $R_C(sN/K)$ parametrized by $s \in \{0, 1, \dots, K\}$.

A. Tree Networks

The basic problem setting considered so far considers users connected to the server through a single shared bottleneck link. We showed that the rate of our proposed algorithm over the shared link is within a constant factor of the optimum. Here we extend this result to more general networks with tree structure (see Fig. 5).

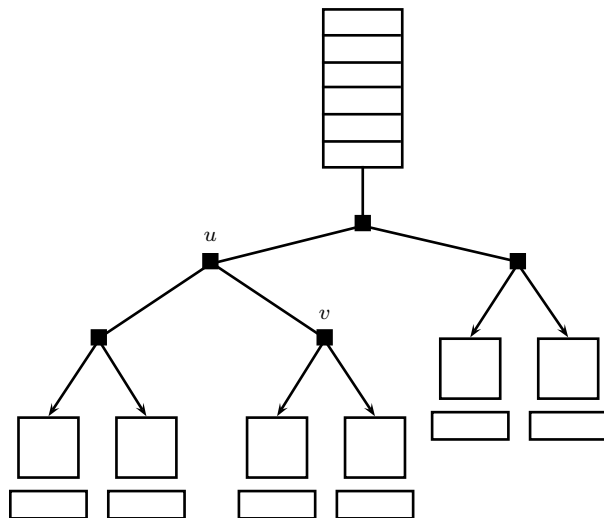


Fig. 5. Network with tree structure. A server containing N files of size F bits each is connected through a tree-structured network to K users each with a cache of size MF bits. Internal nodes of the tree represent routers. In this figure, $N = K = 6$, and $M = 1$. The proposed placement and delivery procedures together with a routing algorithm achieves the order-optimal rate over every link (u, v) of the network.

Consider a directed tree network, oriented from the root to the leaves. The server is located at the root of the tree, and users with their caches are located at the leaves. Each internal node of the network represents a router. The router decides what to transmit over each of its outgoing links as a function of what it received over its single incoming link from its parent.

We again assume that the system operates in two phases. In the placement phase, the caches are populated without knowledge of users' future demands. In the delivery phase, the users reveal their requests, and the server has to satisfy these demands exploiting the cached content.

For this network, we propose the following caching and routing procedures. For the placement phase, we use the same placement procedure as in Algorithm 1. For the delivery phase, we use the two delivery procedures detailed in Algorithm 1, but with the simplified decision rule explained in Remark 4. In other

words, if $M > 1$, the server creates coded packets according to the first delivery procedure. If $M \leq 1$, it uses the second delivery procedure and the server creates linear combinations of the bits of each file without coding across different files.

It remains to describe the operations of the routers at the internal nodes of the tree. Each router operates according to the following simple rule. The router at node u forwards a coded message over link (u, v) if and only if that coded message is directly useful to at least one of the descendant leaves of node v . To be precise, let us assume that $M > 1$ so that the server uses the first delivery procedure. Thus for each subset $\mathcal{S} \subset [K]$ of users, the server creates the coded message $\oplus_{k \in \mathcal{S}} V_{k, \mathcal{S} \setminus \{k\}}$. This coded message is useful for all users in \mathcal{S} and is completely useless for the remaining users. The router located at node u forwards this coded message $\oplus_{k \in \mathcal{S}} V_{k, \mathcal{S} \setminus \{k\}}$ over the link (u, v) , if at least one of the descendants of v (including v itself if it is a leaf) is in the set \mathcal{S} . A similar routing procedure is used for $M \leq 1$.

The performance of this scheme is analyzed in Appendix C. We show there that, for $M > 1$, the rate of this scheme over the link (u, v) is equal to

$$K_v \cdot (1 - M/N) \cdot \frac{N}{K_v M} (1 - (1 - M/N)^{K_v}), \quad (5)$$

where K_v is the number of descendant leaves of node v . For $M \leq 1$, it is easy to see that the rate over the link (u, v) is equal to

$$K_v \cdot (1 - M/N) \cdot \min\{1, N/K_v\}. \quad (6)$$

The rate over every link in the tree network can be shown to be within a constant factor of optimal. To prove approximate optimality for edge (u, v) , we consider the subtree rooted at v together with u and the edge (u, v) . We can then use the same bound used in Theorem 2 over this subtree, treating (u, v) as the shared bottleneck link.

Remark 9 (Universality and Separation of Caching and Routing): This result shows that, for tree-structured networks, caching and routing can be performed separately with at most a constant factor loss in performance compared to the optimal joint scheme. This means that the proposed placement and delivery procedures are universal in the sense that they do not depend on the topology of the tree network connecting the server to the caches at the leaves.

Example 3 (Universality). Consider K users connected to a server through K orthogonal links (i.e., no shared links). For this topology the optimal rate over each link can be achieved without coding. However, it is easy to see that the proposed universal scheme achieves the same optimal rate. Thus, depending on the network topology, we may be able to develop simpler schemes, but the performance will be the same up to a constant factor as the proposed universal scheme. \diamond

Example 4 (Rate over Private Links). Consider the original scenario of users sharing a single bottleneck link. As an example, assume we have $N = 2$ files and $K = 2$ users as described in Example 2 in Section III. Observe that a user does not need all messages sent by the server over the shared link in order to recover its requested file. For example, in order to recover file A , user one only needs $A_2 \oplus B_1$ and A_0 . Thus, if a router is located right where the shared link splits into the two private links, it can forward only these two messages over the private link to user one. By the analysis in this section, the resulting normalized rate over the private link to user one is then

$$(M/2)(1 - M/2) + (1 - M/2)^2 = 1 - M/N.$$

We note that in the uncoded scheme the rate over each private link is also $1 - M/N$. Hence, we see that by proper routing the rate over the private links for both the coded as well as the uncoded schemes are the same. The reduction of rate over the shared link achieved by coding does therefore not result in an increase of rate over the private links. This conclusion holds also for general values of N , K , and M . \diamond

In this section, we have only considered tree networks with caches at the leaves. The general scenario, in which caches are also present at internal nodes of the tree, is more challenging and is analyzed in follow-up work [11].

B. Shared Caches

The problem setting considered throughout this paper assumes that each user has access to a private cache. In this example, we evaluate the gain of shared caches. This situation arises when the cache memory is located close to but not directly at the users.

We consider a system with K users partitioned into subsets, where users within the same subset share a common cache (see Fig. 6). For simplicity, we assume that these subsets have equal size of L users, where L is a positive integer dividing K . We also assume that the number of files N is greater than the number of users K . To keep the total amount of cache memory in the system constant, we assume that each of the shared caches has size LMF bits.

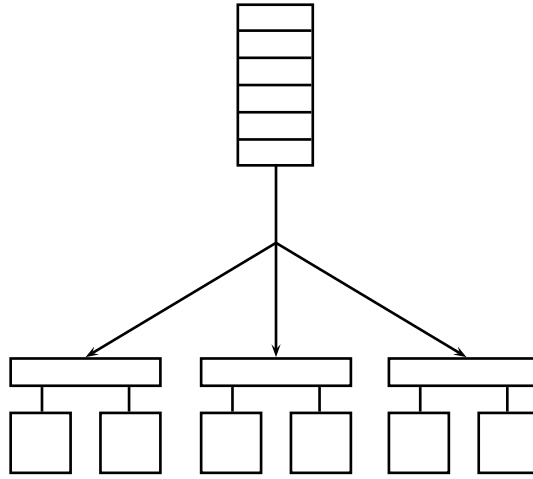


Fig. 6. Users with shared caches. A server containing N files of size F bits each is connected through a shared link to K/L caches each of size LMF bits. Each cache is shared among L users. In the figure, $K = N = 6$ and $L = 2$.

We can operate this system as follows. Define K/L super users, one for each subset of L users. Run the placement procedure of Algorithm 1 for these K/L users with cache size LMF . In the delivery phase, treat the (up to) L files requested by the users in the same subset as a single super file of size LF . Applying Theorem 1 to this setting yields an achievable rate of⁶

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

Let us again consider the regimes of small and large values of M of $R_D^L(M)$. For $M \in [0, N/K]$, we have

$$R_D^L(M) \approx K \cdot \left(1 - \frac{K+L}{2N} M\right).$$

Comparing this to the small- M approximation (2) of $R_D(M)$ (for a system with private caches), we see that

$$R_D^L(M) \approx R_D(M),$$

i.e., there is only a small effect on the achievable rate from sharing a cache. This should not come as a surprise, since we have already seen in Section IV-B that, for small M , $R_D(M)$ behaves almost like a

⁶This can be derived from $R_D(M)$ in Theorem 1 by replacing M by LM (since each cache has now size LMF instead of MF), replacing K by K/L (since there are K/L super users), and multiplying the result by an extra factor of L (since each super file is L times the size of a normal file).

system in which all K caches are combined. Hence, there is no sizable gain to be achieved by having collaboration among caches in this regime.

Consider then the regime $M > N/K$. Here, we have

$$R_D^L(M) \approx N/M - L = K \cdot (1 - LM/N) \cdot \frac{N}{KM}$$

and from (3)

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

The difference between the two approximations is only in the second factor. We recall that this second factor represents the caching gain due to making part of the files available locally. Quite naturally, this part of the caching gain improves through cache sharing, as a larger fraction of each file can be stored locally.

C. Asynchronous User Requests

Up to this point, we have assumed that in the delivery phase all users reveal their requests simultaneously, i.e., that the users are perfectly synchronized. In practice, however, users reveal their requests at different times. In this example, we show that the proposed algorithm can be modified to handle such asynchronous user requests.

We explain the main idea with an example. Consider a system with $N = 3$ files A, B, C , and $K = 3$ users. We split each file into J consecutive segments, e.g., $A = (A^{(1)}, \dots, A^{(J)})$ and similarly for B and C . Here J is a positive integer selected depending on the maximum tolerable delay, as will be explained later. To be specific, we choose $J = 4$ in this example.

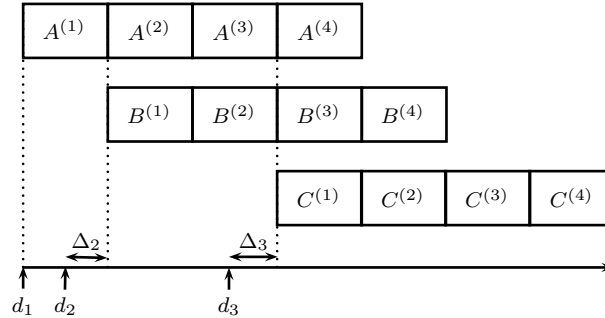


Fig. 7. The proposed scheme over segmented files can be used to handle asynchronous user requests. In this example, each file is split into four segments. Users two and three are served with a delay of Δ_2 and Δ_3 , respectively.

In the placement phase, we simply treat each segment as a file. We apply the placement procedure of Algorithm 1. For the delivery phase, consider an initial request d_1 from user one, say for file A . The server responds by starting delivery of the first segment $A^{(1)}$ of file A . Meanwhile, assume that user two requests file d_2 , say B , as shown in Fig. 7. The server puts the request of user two on hold, and completes the delivery of $A^{(1)}$ for user one. It then starts to deliver the second segment $A^{(2)}$ of A and the first segment $B^{(1)}$ of B using the delivery procedure of Algorithm 1 for two users. Delivery of the next segments $A^{(3)}$ and $B^{(2)}$ is handled similarly. Assume that at this point user three requests file d_3 , say C , as shown in Fig. 7. After completing the current delivery phase, the server reacts to this request by delivering $A^{(4)}$, $B^{(3)}$, and $C^{(1)}$ to users one, two, and three, respectively, using the delivery procedure of Algorithm 1 for three users. The process continues in the same manner as depicted in Fig. 7.

We note that users two and three experience delays of Δ_2 and Δ_3 as shown in the figure. The maximum delay depends on the size of the segments. Therefore, segment size, or equivalently the value of J , can be adjusted to ensure that this delay is tolerable (while keeping the segment size large enough to ensure that the law of large number applies as discussed in Example 2 in Section III).

We point out that the number of effective users in the system varies throughout the delivery phase. Due to its decentralized nature, the proposed caching algorithm is close to optimal for any value of users as discussed in Remark 2. This is instrumental for the segmentation approach just discussed to be efficient.

VI. DISCUSSION

A. Connection to Index and Network Coding

The caching problem considered in this paper is connected to the index coding problem [12], [13] (or, equivalently [14], to the network coding problem [15]). In the index coding problem, we are given a set of K users and a set of N files. Each of the users has access to a fixed subset of those files and requests another fixed subset of the files. The goal is to broadcast the minimum number of bits to the K users in order to satisfy all user requests.

From the above description, we see that for *fixed* content placement and for *fixed* user demands, the caching problem considered in this paper induces an index coding problem in the delivery phase. Since there are N^K possible user demands, the delivery phase of the caching problem can thus be interpreted as an exponential number of parallel index coding subproblems. To complicate matters, the index coding problem itself does not admit a closed-form solution and is known to be computationally intractable in general [16].

One contribution of this paper is thus the design of the placement phase such that all these exponentially many index coding problems simultaneously have an efficient and closed-form solution.

B. Caching Random Linear Combinations is Inefficient

Caching random linear combinations of file segments is a popular prefetching scheme. In this example, we argue that in some scenarios this form of caching can be quite inefficient.

To be precise, let us focus on a specific scenario with K users and $N = K$ files, where each user has sufficient cache memory to store half of the files, i.e. $M = N/2 = K/2$. According to Theorem 1, Algorithm 1 achieves a rate of less than one, i.e., $R_D(M) \leq 1$.

On the other hand, the rate achieved by caching of random linear combinations can be shown to be at least $K/4$, which is significantly larger than $R_D(M)$ for large number of users K . Indeed, assume that user one requests file A . Recall that each user has cached $F/2$ random linear combinations of the bits of file A . With high probability, these random linear combinations span a $F/2$ -dimensional space at each user and the subspaces of different users do not overlap. For example, consider users two and three. As a consequence of this lack of overlap, these two users do not have access to a shared part of the file A . This implies that, in the delivery phase, the server cannot form a linear combination that is simultaneously useful for three users. In other words, the server can form messages that are at most useful simultaneously for up to two users. A short calculation reveals that then the server has to send at least $FK/4$ bits over the shared link.

This inefficiency of caching random linear combinations can be interpreted as follows. The placement phase follows two competing objectives: The first objective is to spread the available content as much as possible over the different caches. The second objective, is to ensure maximum overlap among different caches. The system performance is optimized if the right balance between these two objectives is struck. Caching random linear combinations maximizes the spreading of content over the available caches, but provides minimal overlap among them. At the other extreme, the uncoded caching scheme maximizes the overlap, but provides only minimal spreading of the content. As a consequence, both of these schemes can be highly suboptimal.

C. Worst-Case Demands

Our problem formulation focuses on worst-case requests. In some situation, this is the correct figure of merit. For example, in a wireless scenario, whenever the delivery rate required for a request exceeds

the available link bandwidth, the system will be in outage, degrading user experience. In other situations, for example a wireline scenario, excess rates might only incur a small additional cost and hence might be acceptable. In such cases, the average rate is the right metric, especially when files have different popularities. This is discussed further in [9].

D. Online Coded Caching

In practical scenarios, the set of popular files is time varying. To keep the caching algorithm efficient, the cache contents have to be dynamically updated to track this variation. A popular scheme to update the caches is to evict the least-recently used (LRU) file from the caches and replace it with a newly requested one. This LRU eviction scheme is known to be approximately optimal for systems with a single cache [17]. However, it is not efficient for networks with multiple caches as considered here. This problem of online caching with several caches is investigated in [10]. The decentralized Algorithm 1 presented in this paper turns out to be a crucial ingredient of the suggested online coded caching algorithm in [10].

APPENDIX A PROOF OF THEOREM 1

We first prove correctness. Note that, since there are a total of N files, the operations in Line 3 of Algorithm 1 satisfies the memory constraint of MF bits at each user. Hence the placement phase of Algorithm 1 is correct.

For the delivery phase, assume the server uses the first delivery procedure, and consider a bit in the file requested by user k . If this bit is already cached at user k , it does not need to be sent by the server. Assume then that it is cached at some (possibly empty) set \mathcal{T} of users with $k \notin \mathcal{T}$. Consider the set $\mathcal{S} = \mathcal{T} \cup \{k\}$ in Line 8. By definition, the bit under consideration is contained in $V_{k, \mathcal{S} \setminus \{k\}}$, and as a consequence, it is included in the sum sent by the server in Line 9. Since $k \in \mathcal{S} \setminus \{\tilde{k}\}$ for every other $\tilde{k} \in \mathcal{S}$, user k has access to all bits in $V_{\tilde{k}, \mathcal{S} \setminus \{\tilde{k}\}}$ from its own cache. Hence, user k is able to recover the requested bit from this sum. This shows that the first delivery procedure is correct.

The second delivery procedure is correct as well since the server sends in Line 15 enough linear combinations of every file for all users to successfully decode. This shows that the delivery phase of Algorithm 1 is correct.

It remains to compute the rate. We start with the analysis of the second delivery procedure. If $N \leq K$, then in the worst case there is at least one user requesting every file. Consider then all users requesting file n . Recall that each user requesting this file already has FM/N of its bits cached locally by the operation of the placement phase. An elementary analysis reveals that with high probability for F large enough at most

$$F(1 - M/N) + o(F)$$

random linear combinations need to be sent in Line 9 for all those users to be able to decode. We will assume that the file size F is large and ignore the $o(F)$ term in the following. Since this needs to be done for all N files, the normalized rate in the delivery phase is

$$(1 - M/N)N.$$

If $N > K$, then there are at most K different files that are requested. The same analysis yields a normalized rate of

$$(1 - M/N)K.$$

Thus, the second procedure has a normalized rate of

$$\begin{aligned} R(M) &= (1 - M/N) \min\{K, N\} \\ &= K \cdot (1 - M/N) \cdot \min\{1, N/K\} \end{aligned} \tag{7}$$

for $M \in (0, N]$.

We continue with the analysis of the first delivery procedure. Consider a particular bit in one of the files, say file n . Since the choice of subsets is uniform, by symmetry this bit has probability

$$q \triangleq M/N \in (0, 1]$$

of being in the cache of any fixed user. Consider now a fixed subset of t out of the K users. The probability that this bit is cached at exactly those t users is

$$q^t(1 - q)^{K-t}.$$

Hence the expected number of bits of file n that are cached at exactly those t users is

$$Fq^t(1 - q)^{K-t}.$$

In particular, the expected size of $V_{k, \mathcal{S} \setminus \{k\}}$ with $|\mathcal{S}| = s$ is

$$Fq^{s-1}(1 - q)^{K-s+1}.$$

Moreover, for F large enough the actual realization of the random number of bits in $V_{k, \mathcal{S} \setminus \{k\}}$ is in the interval

$$Fq^{s-1}(1 - q)^{K-s+1} \pm o(F)$$

with high probability. For ease of exposition, we will again ignore the $o(F)$ term in the following.

Consider a fixed value of s in Line 7 and a fixed subset \mathcal{S} of cardinality s in Line 8. In Line 9, the server sends

$$\max_{k \in \mathcal{S}} |V_{k, \mathcal{S} \setminus \{k\}}| = Fq^{s-1}(1 - q)^{K-s+1}$$

bits. Since there are $\binom{K}{s}$ subsets \mathcal{S} of cardinality s , the loop starting in Line 8 generates

$$\binom{K}{s} Fq^{s-1}(1 - q)^{K-s+1}$$

bits. Summing over all values of s yields a total of

$$\begin{aligned} R_D(M)F &= F \sum_{s=1}^K \binom{K}{s} q^{s-1}(1 - q)^{K-s+1} \\ &= F \frac{1 - q}{q} \left(\sum_{s=0}^K \binom{K}{s} q^s(1 - q)^{K-s} - (1 - q)^K \right) \\ &= F \frac{1 - q}{q} (1 - (1 - q)^K) \end{aligned}$$

bits being sent over the shared link. Substituting the definition of $q = M/N$ yields a rate of the first delivery procedure of

$$\begin{aligned} R_D(M) &= (N/M - 1)(1 - (1 - M/N)^K) \\ &= K \cdot (1 - M/N) \cdot \frac{N}{KM} (1 - (1 - M/N)^K) \end{aligned} \tag{8}$$

for $M \in (0, N]$.

Since the server uses the better of the two delivery procedures, (7) and (8) show that Algorithm 1 achieves a rate of

$$R_D(M) = K \cdot (1 - M/N) \cdot \min \left\{ \frac{N}{KM} (1 - (1 - M/N)^K), 1, \frac{N}{K} \right\}.$$

Using that

$$(1 - M/N)^K \geq 1 - KM/N,$$

this can be simplified to

$$K \cdot (1 - M/N) \cdot \min \left\{ \frac{N}{KM} (1 - (1 - M/N)^K), \frac{N}{K} \right\},$$

concluding the proof.

APPENDIX B PROOF OF THEOREM 2

Recall from Theorem 1 that

$$R_D(M) = K \cdot (1 - M/N) \cdot \min \left\{ \frac{N}{KM} (1 - (1 - M/N)^K), \frac{N}{K} \right\}.$$

Using that

$$(1 - M/N)^K \geq 0$$

and

$$(1 - M/N)^K \geq 1 - KM/N,$$

this can be upper bounded as

$$R_D(M) \leq \min \{ N/M - 1, K(1 - M/N), N(1 - M/N) \} \quad (9)$$

for all $M \in [0, N]$. Moreover, we have from [8, Theorem 2]

$$R^*(M) \geq \max_{s \in \{1, \dots, \min\{N, K\}\}} \left(s - \frac{s}{\lfloor N/s \rfloor} M \right). \quad (10)$$

We will treat the cases $0 \leq \min\{N, K\} \leq 12$ and $\min\{N, K\} \geq 13$ separately. Assume first that $0 \leq \min\{N, K\} \leq 12$. By (9),

$$R_D(M) \leq \min\{N, K\}(1 - M/N) \leq 12(1 - M/N),$$

and by (10) with $s = 1$,

$$R^*(M) \geq 1 - M/N.$$

Hence

$$\frac{R_D(M)}{R^*(M)} \leq 12 \quad (11)$$

for $0 \leq \min\{N, K\} \leq 12$.

Assume in the following that $\min\{N, K\} \geq 13$. We consider the cases

$$M \in \begin{cases} [0, \max\{1, N/K\}], \\ (\max\{1, N/K\}, N/12], \\ (N/12, N], \end{cases}$$

separately. Assume first that $0 \leq M \leq \max\{1, N/K\}$. By (9),

$$R_D(M) \leq \min\{N, K\}(1 - M/N) \leq \min\{N, K\}.$$

On the other hand, by (10) with $s = \lfloor \min\{N, K\}/4 \rfloor$,

$$\begin{aligned} \mathbb{R}^*(M) &\geq s - \frac{s^2}{1 - s/N} \frac{M}{N} \\ &\geq \min\{N, K\}/4 - 1 - \frac{(\min\{N, K\})^2/16}{1 - \min\{N, K\}/(4N)} \frac{M}{N} \\ &\stackrel{(a)}{\geq} \min\{N, K\} \left(1/4 - 1/13 - \frac{1/16}{1 - 1/4} \right) \\ &\geq \min\{N, K\}/12, \end{aligned}$$

where in (a) we have used that $\min\{N, K\} \geq 13$ and $M \leq \max\{1, N/K\}$. Hence

$$\frac{R_D(M)}{R^*(M)} \leq 12 \quad (12)$$

for $\min N, K \geq 13$ and $0 \leq M \leq \max\{1, N/K\}$.

Assume then that $\max\{1, N/K\} < M \leq N/12$. By (9),

$$R_D(M) \leq N/M - 1 \leq N/M.$$

On the other hand, by (10) with $s = \lfloor N/(4M) \rfloor$,

$$\begin{aligned} R^*(M) &\geq s - \frac{s^2}{1 - s/N} \frac{M}{N} \\ &\geq N/(4M) - 1 - \frac{N^2/(16M^2)}{1 - 1/(4M)} \frac{M}{N} \\ &= \frac{N}{M} \left(1/4 - M/N - \frac{1/16}{1 - 1/(4M)} \right) \\ &\stackrel{(a)}{\geq} \frac{N}{M} \left(1/4 - 1/12 - \frac{1/16}{1 - 1/4} \right) \\ &= N/(12M), \end{aligned}$$

where in (a) we have used that $M/N \leq 1/12$ and that $M > \max\{1, N/K\} \geq 1$. Hence

$$\frac{R_D(M)}{R^*(M)} \leq 12 \quad (13)$$

for $\min\{N, K\} \geq 13$ and $\max\{1, N/K\} < M \leq N/12$.

Finally, assume that $N/12 < M \leq N$. By (9),

$$R_D(M) \leq N/M - 1.$$

On the other hand, by (10) with $s = 1$,

$$R^*(M) \geq 1 - M/N.$$

Hence,

$$\begin{aligned} \frac{R_D(M)}{R^*(M)} &\leq \frac{N/M - 1}{1 - M/N} \\ &= N/M \\ &\leq 12 \end{aligned} \quad (14)$$

for $\min\{N, K\} \geq 13$ and $N/12 < M \leq N$.

Combining (11), (12), (13), and (14) yields that

$$\frac{R_D(M)}{R^*(M)} \leq 12$$

for all N, K , and $0 \leq M \leq N$.

APPENDIX C

PROOF OF (5) IN SECTION V-A

As is shown in Appendix A, for F large enough the actual realization of the random number of bits in $V_{k, \mathcal{S} \setminus \{k\}}$ is in the interval

$$Fq^{s-1}(1-q)^{K-s+1} \pm o(F)$$

with high probability, and where $q = M/N$. As before, we will again ignore the $o(F)$ term in the following.

Recall that only a subset of coded messages generated in Line 8 in Algorithm 1 pass through link (u, v) , namely only those $\oplus_{k \in \mathcal{S}} V_{k, \mathcal{S} \setminus \{k\}}$ for which the subset \mathcal{S} has at least one element among leave descendants of node v . We split \mathcal{S} into \mathcal{S}_1 and \mathcal{S}_2 where \mathcal{S}_1 is the subset of descendant leaves of node v , and $\mathcal{S}_2 \triangleq \mathcal{S} \setminus \mathcal{S}_1$. Denote the cardinalities of \mathcal{S}_1 and \mathcal{S}_2 by s_1 and s_2 , respectively, so that $s = s_1 + s_2$. With this, only coded messages with $s_1 \geq 1$ are forwarded over link (u, v) . The number of bits sent over this link is then equal to

$$\begin{aligned} & F \sum_{s_1=1}^{K_v} \sum_{s_2=0}^{K-K_v} \binom{K_v}{s_1} \binom{K-K_v}{s_2} q^{s_1+s_2-1} (1-q)^{K-s_1-s_2+1} \\ &= F \sum_{s_1=1}^{K_v} \binom{K_v}{s_1} q^{s_1-1} (1-q)^{K_v-s_1+1} \sum_{s_2=0}^{K-K_v} \binom{K-K_v}{s_2} q^{s_2} (1-q)^{K-K_v-s_2} \\ &= F \sum_{s_1=1}^{K_v} \binom{K_v}{s_1} q^{s_1-1} (1-q)^{K_v-s_1+1} \\ &= F \frac{1-q}{q} (1 - (1-q)^{K_v}). \end{aligned}$$

Substituting $q = M/N$ yields the desired result.

REFERENCES

- [1] L. W. Dowdy and D. V. Foster, "Comparative models of the file assignment problem," *ACM Comput. Surv.*, vol. 14, pp. 287–313, June 1982.
- [2] K. C. Almeroth and M. H. Ammar, "The use of multicast delivery to provide a scalable and interactive video-on-demand service," *IEEE J. Sel. Areas Commun.*, vol. 14, pp. 1110–1122, Aug. 1996.
- [3] A. Dan, D. Sitaram, and P. Shahabuddin, "Dynamic batching policies for an on-demand video server," *Multimedia Syst.*, vol. 4, pp. 112–121, June 1996.
- [4] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, "Placement algorithms for hierarchical cooperative caching," in *Proc. ACM-SIAM SODA*, pp. 586–595, Jan. 1999.
- [5] A. Meyerson, K. Munagala, and S. Plotkin, "Web caching using access statistics," in *Proc. ACM-SIAM SODA*, pp. 354–363, 2001.
- [6] I. Baev, R. Rajaraman, and C. Swamy, "Approximation algorithms for data placement problems," *SIAM J. Comput.*, vol. 38, pp. 1411–1429, July 2008.
- [7] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *Proc. IEEE INFOCOM*, pp. 1–9, Mar. 2010.
- [8] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *arXiv:1209.5807 [cs.IT]*, Sept. 2012. To appear in *IEEE Trans. Inf. Theory*.
- [9] U. Niesen and M. A. Maddah-Ali, "Coded caching with nonuniform demands," *arXiv:1308.0178 [cs.IT]*, Aug. 2013.
- [10] R. Pedarsani, M. A. Maddah-Ali, and U. Niesen, "Online coded caching," *arXiv:1311.3646 [cs.IT]*, Nov. 2013.
- [11] N. Karamchandani, U. Niesen, M. A. Maddah-Ali, and S. Diggavi, "Hierarchical coded caching," *arXiv:1403.7007 [cs.IT]*, Mar. 2014.
- [12] Y. Birk and T. Kol, "Coding on demand by an informed source (ISCOD) for efficient broadcast of different supplemental data to caching clients," *IEEE Trans. Inf. Theory*, vol. 52, pp. 2825–2830, June 2006.
- [13] Z. Bar-Yossef, Y. Birk, T. S. Jayram, and T. Kol, "Index coding with side information," *IEEE Trans. Inf. Theory*, vol. 57, pp. 1479–1494, Mar. 2011.
- [14] M. Effros, S. El Rouayheb, and M. Langberg, "An equivalence between network coding and index coding," *arXiv:1211.6660 [cs.IT]*, Nov. 2012.
- [15] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, pp. 1204–1216, Apr. 2000.
- [16] M. Langberg and A. Sprintson, "On the hardness of approximating the network coding capacity," *IEEE Trans. Inf. Theory*, vol. 57, pp. 1008–1014, Feb. 2011.

- [17] D. D. Sleator and R. E. Tarjan, “Amortized efficiency of list update and paging rules,” *Communications ACM*, vol. 28, pp. 202–208, Feb. 1985.