



**HAL**  
open science

## Cache-Aided Polar Coding: From Theory to Implementation

Yasser Fadlallah, Othmane Oubejja, Sarah Kamel, Philippe Ciblat, Michele Wigger, Jean-Marie S Gorce

► **To cite this version:**

Yasser Fadlallah, Othmane Oubejja, Sarah Kamel, Philippe Ciblat, Michele Wigger, et al.. Cache-Aided Polar Coding: From Theory to Implementation. *IEEE Journal on Selected Areas in Information Theory*, 2021, 2 (4), pp.1206 - 1223. 10.1109/JSAIT.2021.3128232 . hal-03482281

**HAL Id: hal-03482281**

**<https://inria.hal.science/hal-03482281>**

Submitted on 18 Jun 2024

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Cache-Aided Polar Coding: From Theory to Implementation

Yasser Fadlallah<sup>1</sup>, *Member, IEEE*, Othmane Oubejja<sup>2</sup>, Sarah Kamel, Philippe Ciblat<sup>3</sup>,  
Michèle Wigger<sup>4</sup>, *Senior Member, IEEE*, and Jean-Marie Gorce, *Senior Member, IEEE*

**Abstract**—This paper proposes an extended coded caching scheme based on piggyback coding for single-server multi-user networks with decentralized caching. The proposed scheme is obtained by adapting Polar codes and extending the original coded caching scheme, which is based on index coding and a data assignment that can be implemented via minimum graph-colouring. Polar codes are adapted so that users can apply parts of their cache contents as the “frozen bits” for Polar decoding, and the coded caching is adapted so as to account for different user coding rates and to combine transmissions to cache-aided and cache-free users. Numerical simulations prove that our piggyback-coding based scheme achieves higher rates than previous schemes also in the finite block-length regime. Real testbed measurements are presented, which validate the practical implementation.

**Index Terms**—Caching, Polar codes, graph colouring, testbed, fading channels,

## I. INTRODUCTION

PLACING contents in cache memories during periods of low network congestions is a powerful tool to reduce network load during peak-traffic times. The fundamental works [1], [2], see also [3]–[15], showed that smart placement and coded multicasting strategies allow to further decrease the network load by a multiplicative factor. These works modeled the communication link from the server to the users as a noise-free shared link. Subsequently, [16], [17] showed that further reductions in network load are possible in asymmetric or fading scenarios, see also [18]–[29]. Specifically, [16], [17] proposed a new coding scheme, termed *piggyback coding*, which exploits the noisy nature of the channel through a joint design of the multicast transmissions and channel coding,

Manuscript received May 15, 2021; revised September 10, 2021; accepted November 3, 2021. Date of publication November 22, 2021; date of current version December 23, 2021. This work was supported by the European Research Council (ERC) through the European Union’s Horizon 2020 under Grant 715111. (*Corresponding author: Michèle Wigger.*)

Yasser Fadlallah is with the CS Department, University of Sciences and Arts in Lebanon, Ghobeiry 47-25, Lebanon (e-mail: y.fadlallah@usal.edu.lb).

Othmane Oubejja was with the LTCI, Telecom Paris, IP Paris, 91120 Palaiseau, France. He is now with ST Microelectronics, 38000 Grenoble, France (e-mail: oubejaa@gmail.com).

Sarah Kamel was with the LTCI, Telecom Paris, IP Paris, 91120 Palaiseau, France. She is now with the Optical Communication Technology Lab, Huawei Technologies Paris Research Center, 92100 Boulogne-Billancourt, France (e-mail: sarah.e.kamel@gmail.com).

Philippe Ciblat and Michèle Wigger are with the LTCI, IP Paris, Telecom Paris, 91120 Palaiseau, France (e-mail: philippe.ciblat@telecom-paris.fr; michele.wigger@telecom-paris.fr).

Jean-Marie Gorce is with Univ Lyon, INSA Lyon, Inria, CITI, 69621 Villeurbanne, France (e-mail: jean-marie.gorce@inria.fr).

Digital Object Identifier 10.1109/JSAT.2021.3128232

and proved that the piggyback coding achieves higher reliable communication rates in an information-theoretic framework where the blocklength tends to infinity.

The goal of this paper is three-fold: 1) Present an extended coded caching scheme based on the piggyback coding schemes in [16], [17], [19] for the single-server multi-user broadcast network; 2) Provide a prototype implementation of the proposed scheme based on Polar codes and show through numerical simulations that it improves performance over classic coded caching schemes also in the finite blocklength regime; and 3) Validate the proposed implementation on a practical testbed.

We consider a setup with a single server, which stores an entire library of  $N$  files, and  $K$  users, each wishing to download one of these files. Some of the users (but not necessarily all) are equipped with cache memories where they can prefetch some of the contents in the library. We consider a *decentralized caching scenario* where each cache-aided user randomly prefetches information from the library to its cache memory, irrespective of the contents prefetched by other users. During this *cache prefetching phase*, also called *placement phase*, the users have not yet chosen which specific file they wish to download from the library. The cache-prefetching should thus follow a universal algorithm as in [1]. Following this placement phase, all users select one of the files for download, and the server will deliver the missing information (the information that was not placed in their cache memories) during the subsequent *delivery phase*. In this article, we model the network during the delivery phase either as a Gaussian or a frequency-selective fading broadcast channel.

The coded-caching delivery scheme in [1], [2] proposed that the server successively multicasts the XOR of contents for different groups of users. Each user that is served by this multicast transmission can recover its desired contents if it has stored all other XORed contents in its cache memory. Piggyback coding combines this idea with channel coding, so as to allow different users in each group to be served at their corresponding capacities. Furthermore, piggyback coding allows to add contents to cache-free users to the various multicast transmissions. We explain the principle of piggyback coding directly on the Polar-codes implementation proposed in this paper. For each Polar codeword, the server XORs contents for a specific group of cache-aided users, where it zero-pads the contents to the various users to the same length if required. Then, it appends further contents for cache-free users to this XORed bit string, and feeds the concatenated string to a Polar encoder. In this last encoding step it is important that *the first*

bits of the string (the XOR-bits) are mapped to the most reliable bits of the Polar code and the latter to less reliable bits. A cache-free user simply decodes all the information bits from the Polar code, and should thus have good channel conditions, unless the XORed string is short. A cache-aided user should be able to retrieve the bits intended for the cache-free users and the high-order bits of the XOR string from its cache memory and use them as “frozen bits” in a standard Polar decoding algorithm, e.g., [30], [31], so as to be able to decode the part of the XOR string containing its intended contents. Finally, it also uses its cache contents to retrieve its intended contents from the truncated XOR bits as in coded caching. In this sense, in our piggyback coding implementation the cache contents not only allow the users to cancel interference in the XORs, but by serving as frozen bits the cache contents also allow the users to decrease their decoding error probabilities.

A particularity of our piggyback-coding usage of Polar codes is that different users are served by information bits of different reliabilities. In fact, the most reliable bits of the Polar codes are dedicated to the transmission of XORs, and thus to the transmission of information to cache-aided users. Cache-free users are typically served in positions with lower reliabilities. In an information-theoretic setting this distinction is not relevant because the probability of error tends to 0 on all the Polar code information bits. Our numerical simulation results show that also in the finite block-length regime studied in this paper, cache-free users are served at similar bit-error rates (BER) as if they were cache-aided users. This indicates that the Polar code crystallizes sufficiently fast so that the various information-bits have similar BERs and our piggyback coding does not significantly penalize cache-free users by serving them only on information bits with lower reliabilities.

Both the original coded-caching delivery scheme, as well as the extended piggyback coding schemes require a careful assignment of the various contents to the different multicast transmissions. The optimal assignment for coded-caching delivery can be phrased as an Index Coding problem [32] and solved by applying a minimum graph colouring algorithm to a *conflict graph* representing the contents that have to be delivered [14]. (Each colour indicates a group of contents that should be sent in a single multicast transmission.) Since minimum graph-colouring is known to be NP-hard, [33]–[35] proposed polynomial-time algorithms to obtain graph colourings yielding large throughputs. Finding the assignment for our piggyback coding delivery is even more complicated because different users should be served at different rates and because contents for cache-free users also have to be assigned. In this paper, we solve this problem by iterative applications of the *Greedy Randomized Adaptive Search Procedure (GRASP)* for graph colouring in [33] to varying conflict graphs, which allows to account for different rates at the various users. In a second step, a maximum bipartite matching algorithm is used to assign contents of cache-free users to the multicast groups obtained in the first step.

We conclude the introduction with a list of the main contributions of this article:

- 1) A practical scheme based on the piggyback coding idea in [17]–[19] is proposed under decentralized caching. The implementation is based on *Polar codes* [30] and an

extended data assignment algorithm. Polar codes offer a natural tool to implement piggyback coding schemes through smart selection of the frozen bits during the users’ decoding steps. The data assignment is extended to account for different rates at the various users and to include also contents intended for cache-free users.

- 2) An implementation prototype of the proposed Polar piggyback coding scheme is presented, and through numerical simulations an improved throughput of our implementation in the finite block-length regime is shown for Gaussian and frequency-selective fading channels. The gains compared to the classical coded caching scheme prove the utility of piggyback coding also for finite blocklengths. Previous works [17]–[19], [21] had focused on the asymptotic infinite blocklength regime. The gains of our practical implementation compared to a system without piggyback coding nor coded caching prove the effectiveness of the proposed scheme. In particular, they show that the proposed extended data assignment algorithm efficiently assigns contents to both cache-aided and cache-free users.
- 3) The practical implementation of the overall cache-aided Polar coding scheme is validated with real testbed measurements on FIT/CorteXlab [36]. The present paper thus significantly extends and improves over the previous testbed implementation in [37], which implemented the original coded caching scheme using the GRASP minimum graph-colouring algorithm. In our implementation we are able to serve cache-aided users at different rates and can also include transmissions to cache-free users.

## II. DETAILED MODEL DESCRIPTION

Consider a communication network with a single server and users  $1, \dots, K$ . The server has access to a library of  $N$  files:

$$W_1, \dots, W_N, \quad (1)$$

where each file consists of  $F$  independent and identically distributed (i.i.d.) bits. The popularity of a file is defined as the probability to be requested by a user  $k$ , i.e., user  $k$  requests the file  $W_n$  with probability  $p_n$ . This popularity  $p_n$  is computed according to a Zipf-distribution [38] with parameter  $s$  as follows

$$p_n = \frac{1/n^s}{\sum_{k=1}^N 1/k^s}. \quad (2)$$

Consequently, the popularity of  $W_n$  is a decreasing sequence.

Users are divided into two categories: i) users equipped with a cache memory, where parts of files can be prefetched, and ii) users without cache memories. Without loss of generality, we can assume that users in  $\mathcal{K}_{\text{cache}} := \{1, \dots, K_0\}$  have a cache memory, for some given positive integer  $K_0 > 0$ , and users in  $\mathcal{K}_{\text{nocache}} := \{K_0 + 1, \dots, K\}$  don’t.

Communication takes place in two phases as explained next.

### A. Decentralized Placement Phase

During the *the placement phase*, each cache-aided user  $k \in \mathcal{K}_{\text{cache}}$  downloads bits from the server according to a common randomized policy. Specifically, each file is split into

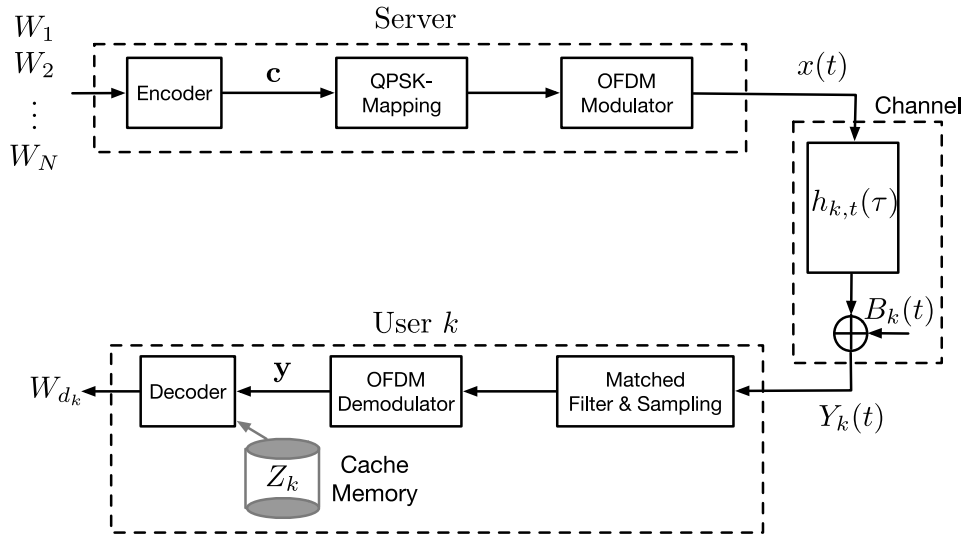


Fig. 1. System for Delivery Phase Transmission.

$N_{\text{chunks}}$  chunks of equal size  $F_{\text{chunk}}$ , and each cache-aided user independently and randomly downloads chunks from the various files in a way that it totally caches a fraction  $\alpha \in (0, 1)$  of the entire library (i.e.,  $\alpha NF$  bits in total) and the number of bits cached from file  $n$  is proportional to its file popularity  $p_n$ . This caching policy is achieved, if each cache-aided user  $k \in \mathcal{K}_{\text{cache}}$  stores a fraction<sup>1</sup>  $\alpha p_n N$  of the chunks associated with file  $W_n$  in its cache memory  $Z_k$ , which results in a total cache size per user

$$\sum_{n=1}^N \alpha p_n N \underbrace{N_{\text{chunks}} F_{\text{chunk}}}_{=F} = \alpha N \underbrace{\sum_{n=1}^N p_n}_{=1} F = \alpha NF. \quad (3)$$

### B. Delivery Phase Over a Wireless Channel

In the subsequent *delivery phase*, each user  $k$  makes a demand  $d_k \in \{1, \dots, N\}$  and sends it to the server and all other receivers. This communication roughly consumes  $2K \log_2(N)$  bits, which can be neglected when  $\log_2(N) \ll F$ . This assumption is satisfied, for instance, in a video delivery system containing only the most popular files.

The task of the server is to send a signal so that each user can reconstruct its demanded file from the received delivery signal and its local cache content  $Z_k$ . The delivery system is illustrated in Figure 1 and explained in detail in the following.

1) *Fading Channel and Modulation System*: Communication during the delivery phase takes place over a noisy broadcast channel from the server to the  $K$  users. For our numerical simulations we either assume a Gaussian broadcast channel or a frequency-selective fading channel. In the following we describe the modulation system based on a fading channel. For the Gaussian channel, the system is similar.

Communication takes place around the central frequency  $f_0$  and for a given baseband input signal  $x(\cdot)$ , the baseband output

signal at user  $k$  is:

$$Y_k(t) = \sum_{m=1}^L a_{k,m}(t)x(t - \tau_{k,m}) + B_k(t), \quad t \in \mathbb{R}, \quad (4)$$

where

- $L$  stands for the number of transmission paths (which is assumed finite at all times and for all users);
- $\tau_{k,m}$  corresponds to the delay of the  $m$ -th path for user  $k$ . The delays are assumed to be fixed over the communication duration.
- $a_{k,m}(t)$  corresponds to the *random* amplitude of the  $m$ -th path for user  $k$  at time  $t$ . The exact model used in our simulations is detailed in Section VI-B.
- $B_k(t)$  is an additive circularly-symmetric zero-mean white Gaussian noise process.

The baseband input signal  $x(\cdot)$  corresponds to the output of an OFDM modulator with  $N_c$  subcarriers and a cyclic prefix of length  $N_{cp}$ . More precisely, the server has a bit stream  $\mathbf{c}$  produced by the proposed encoder (see Fig. 1). This stream is then QPSK-modulated and passed to the OFDM modulator. The length of each OFDM symbol is given by  $T = (N_c + N_{cp})T_s$  where  $T_s$  is the sample period.

At the receiver-side, any user  $k$  passes its baseband receive signal  $Y_k(t)$  through a low-pass filter, a sampler with sampling period  $T_s$ , and an OFDM demodulator. Parameters  $N_c$  and  $N_{cp}$  are chosen<sup>2</sup> such that for each subcarrier  $f \in \{1, \dots, N_c\}$  and OFDM symbol  $j = 1, 2, \dots$ , the relation between the corresponding QPSK symbol  $x(f, j)$  and output  $\tilde{y}_k(f, j)$  can be written as

$$\tilde{y}_k(f, j) = h_k(f, j) \cdot x(f, j) + b_k(f, j), \quad j = 1, 2, \dots, \quad (5)$$

where  $b_k(f, j)$  is a memoryless Gaussian noise and  $h_k(f, j)$  stands for an appropriate channel coefficient.

<sup>2</sup>We assume  $N_{cp}T_s$  smaller than the dispersion time of the channel in (4), and  $T$  much smaller than the coherence time of this channel. Consequently, the amplitude  $a_{k,m}(t)$  remains unchanged during the duration of a single OFDM symbol.

<sup>1</sup>Note that  $\alpha p_n N$  has to be less than 1. If not, we replace its value by 1. When  $s = 0$  (uniform distribution),  $p_n = 1/N$  for any  $n$  leading to  $\alpha p_n N = \alpha < 1$ .

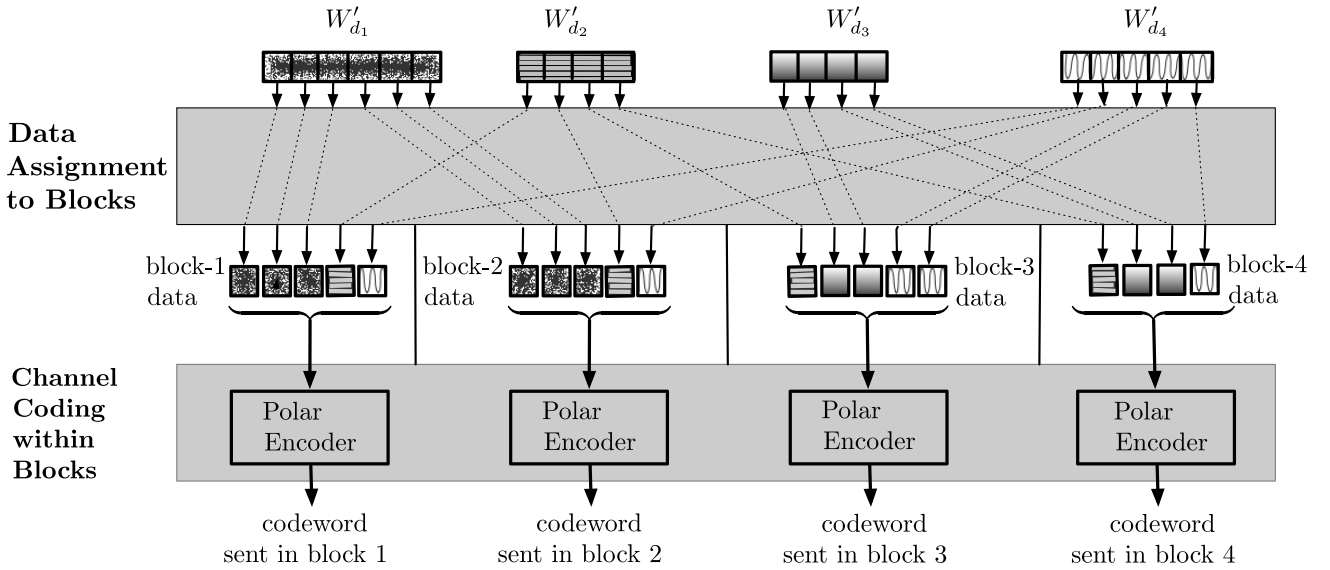


Fig. 2. General structure of the delivery encoding. A *Data Assignment* procedure assigns the non-cached data into the various transmission blocks (4 blocks in the example). A *Channel Coding* procedure encodes the data for a given block into an independent (Polar)codeword that is sent over the channel.

The stream  $\mathbf{y}$  (see Fig. 1) consists of log-likelihood ratios (LLR) constructed from  $\{\tilde{y}_k(f, j) : f = 1, \dots, N_c, j = 1, 2, \dots\}$  with respect to the QPSK mapping.

2) *Encoding and Decoding*: The server constructs the input signal bit-stream  $\mathbf{c}$  in function of all messages  $W_1, \dots, W_N$ , the users' demands  $d_1, \dots, d_K$  and their cache contents  $Z_1, \dots, Z_{K_0}$ . In the simplest case, the bit-stream  $\mathbf{c}$  is obtained by applying a traditional channel encoder to the bits of the demanded file  $W_{d_k}$  that are not cached at user  $k$ , for any user  $k \in \mathcal{K}$ . In this article (Sections III–V) we propose a new design of this encoder (and the decoder).

The LLR-stream  $\mathbf{y}$  is passed to a decoder, which uses also its cache-content  $Z_k$  to reconstruct its demanded message  $W_{d_k}$ . In the simplest case, this decoder applies a traditional channel soft decoder to  $\mathbf{y}$  and then recover all the missing bits of its demanded message  $W_{d_k}$  from the decoded message. In Sections III and V, we propose a more sophisticated decoding algorithm that also uses the cache-contents to improve the performance of standard Polar decoders.

### III. DELIVERY SCHEME: MAIN STRUCTURE AND EXAMPLE

#### A. General Structure

For each user  $k$  let  $W'_{d_k}$  denote the set of the chunks of file  $W_{d_k}$  that are not cached at this user  $k$ . All the chunks of  $W'_{d_k}$  have to be sent to this user during the delivery phase.

The general two-stage structure of the delivery encoding is illustrated in Figure 2 for the case of  $K = 4$  users and transmission taking place over 4 blocklengths of a given Polar code.

In a first step, called *Data Assignment*, the server assigns for each user  $k$  all the bits of  $W'_{d_k}$  to the various transmission blocks  $b = 1, 2, \dots$ . In this step, the chunks can be split into subchunks that are possibly assigned to different blocks. In a second step, called *Channel Coding*, the server combines

the various bits assigned to a given block  $b$  in a sophisticated way and sends the obtained strings using a Polar code during transmission block  $b$ . An example is given in Section III-B. The following Sections IV–V are devoted to our choices of the data-assignment procedure and the channel coding procedure.

As we will see, *both procedures take into account the cache contents at the users*. Moreover, for the overall system to perform well, the data-assignment procedure also has to adapt to the decoding capabilities (i.e., the channel conditions) of the various users. More specifically, the data-assignment is performed in a way that:

- any (sub)chunk sent to a cache-aided user in a given block has a rate below the capacity to this user and is stored at all other cache-aided users served during this block;
- the (sub)chunks sent to cache-free users in a given block are stored at all cache-aided users served in the same block;
- the sum-rate to all cache-free users plus the maximum rate to a cache-aided user is below the capacity of any cache-free user.

In the following subsection, we provide an illustrative example for the two procedures.

#### B. An Example

Consider a setup with four users  $K = 4$ , where users 1–3 have cache memories but user 4 does not, i.e.,  $\mathcal{K}_{\text{cache}} = \{1, 2, 3\}$  and  $\mathcal{K}_{\text{nocache}} = \{4\}$ . The example is designed for a network where users 1 and 2 have capacity at least  $1/2$ , user 3 has capacity at least  $1/6$ , and user 4 capacity at least  $2/3$ .

For ease of exposition, we assume that each file consists of  $N_{\text{chunk}} = 3$  chunks only. The channel coding that we will describe shortly is based on a Polar code of blocklength  $n$  that equals twice the size of a chunk, i.e.,  $n = 2F_{\text{chunk}}$ , and thus each chunk is of rate  $1/2$  with respect to this blocklength. We further simplify notation by naming the chunks of the

requested files as:

$$W_{d_1} = (a, \tilde{a}, \hat{a}); \quad W_{d_2} = (b, \tilde{b}, \hat{b}); \quad (6)$$

$$W_{d_3} = (c, \tilde{c}, \hat{c}); \quad W_{d_4} = (e, \tilde{e}, \hat{e}). \quad (7)$$

For the purpose of the example, chunks  $c$  and  $e$  are divided into 3 subchunks  $c = (c_1, c_2, c_3)$ , and  $e = (e_1, e_2, e_3)$  each consisting of  $F_{\text{chunk}}/3$  bits.

Assume that after the placement phase as described in Section II-A, Users 1–3 have the following cache contents:

$$Z_1 = (\hat{a}, b, c, e) \quad (8)$$

$$Z_2 = (a, \tilde{b}, \hat{b}, c, e, \tilde{e}) \quad (9)$$

$$Z_3 = (a, \tilde{a}, b, \tilde{c}, \hat{c}, e, \tilde{e}) \quad (10)$$

(We are not showing cache contents pertaining to files that are not requested during the delivery phase.) The four users thus have to learn the following (sub)chunks during the delivery phase

$$W'_{d_1} = (a, \tilde{a}) \quad (11a)$$

$$W'_{d_2} = b \quad (11b)$$

$$W'_{d_3} = (c_1, c_2, c_3) \quad (11c)$$

$$W'_{d_4} = (e_1, e_2, e_3, \tilde{e}, \hat{e}). \quad (11d)$$

*Data Assignment:* The missing (sub)chunks in (11) are assigned to the channel coding blocks in the following way:

- 1) subchunks  $(a, b, c_1, e_1)$  are assigned to block 1, which will serve the group of users  $\mathcal{G}^{(1)} = \{1, 2, 3, 4\}$ ;
- 2) subchunks  $(\tilde{a}, c_2, e_2)$  are assigned to block 2, which will serve the group of users  $\mathcal{G}^{(2)} = \{1, 3, 4\}$ ;
- 3) subchunks  $(c_3, \tilde{e})$  are assigned to block 3, which will serve the group of users  $\mathcal{G}^{(3)} = \{3, 4\}$ ;
- 4) subchunks  $(e_3, \hat{e})$  are assigned to block 4, which will serve the group of users  $\mathcal{G}^{(4)} = \{4\}$ .

Notice that in each block  $b$ :

- Each (sub)chunk intended for a given cache-aided user in  $\mathcal{G}^{(b)}$  is stored in the cache memories of all other cache-aided users of  $\mathcal{G}^{(b)}$ .
- Each (sub)chunk intended for a cache-free user (here always user 4) is stored in the cache memories of *all* cache-aided users of  $\mathcal{G}^{(b)}$ .
- The cache-aided users 1, 2, 3 are served at data rates  $r_1 = 1/2$ ,  $r_2 = 1/2$ , and  $r_3 = 1/6$ , respectively, if they are in  $\mathcal{G}^{(b)}$ .
- The *data rate* to the cache-free user 4 is  $r_4^{(1)} = r_4^{(2)} = 1/6$  during the first two blocks,  $r_4^{(3)} = 1/2$  during block 3, and  $r_4^{(4)} = 2/3$  during the last block 4. In general it is chosen as  $r_4^{(b)} := r_{\text{max}} - r_{\text{cache,max}}^{(b)}$ , where

$$r_{\text{max}} := 2/3 \quad (12)$$

$$r_{\text{cache,max}}^{(b)} := \max_{k \in \mathcal{G}_{\text{cache}}^{(b)}} r_k. \quad (13)$$

(i.e.,  $r_{\text{cache,max}}^{(b)}$  denotes the largest data-rate of all *cache-aided* users served in block  $b$ .)

The channel codings that we describe in the following enable reliable communication if users 1 and 2 can reliably decode

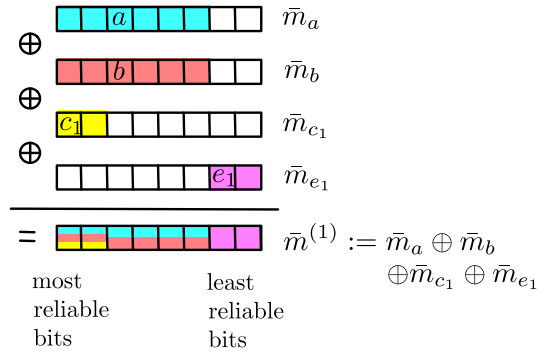


Fig. 3. Creation of “information bits” for the Polar encoder used in block 1.

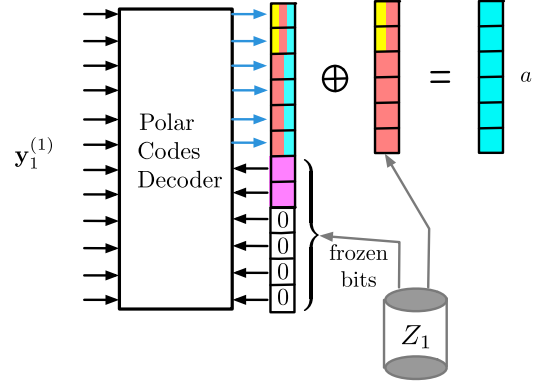


Fig. 4. Decoding procedure at the cache-aided User 1.

at rates  $r_1 = r_2 = 1/2$ , user 3 at rate  $r_3 = 1/6$ , and the cache-free user 4 at rate  $r_{\text{max}} = 2/3$ .

*Channel Encoding in Block 1 (Fig. 3):* The server first appends  $n(r_{\text{max}} - r_1)$ ,  $n(r_{\text{max}} - r_2)$ , and  $n(r_{\text{max}} - r_3)$  zeros at the end of the three chunks  $a, b, c_1$ , and it appends  $nr_{\text{cache,max}}^{(1)} = n/2$  zeros at the beginning of subchunk  $e_1$ , so as to make them all of same length. Then it takes the component-wise XOR of the four resulting  $nr_{\text{max}} = 2n/3$ -length packets  $\bar{m}_a, \bar{m}_b, \bar{m}_{c_1}, \bar{m}_{e_1}$  to form

$$\bar{m}^{(1)} := \bar{m}_a \oplus \bar{m}_b \oplus \bar{m}_{c_1} \oplus \bar{m}_{e_1}. \quad (14)$$

The encoder applies a standard blocklength- $n$  Polar encoder [30] to the information bits  $\bar{m}^{(1)}$ , where it ensures that these bits are mapped to the information bits of the Polar code in decreasing order of reliability. The codeword produced by the Polar code is transmitted over the channel in block 1.

*Channel Decoding in Block 1 (Figs. 4–6):* Let  $\mathbf{y}_1^{(1)}, \mathbf{y}_2^{(1)}, \mathbf{y}_3^{(1)}, \mathbf{y}_4^{(1)}$  denote the output vectors of the OFDM modulators at the four receivers corresponding to the first block.

We explain how the users in  $\mathcal{G}^{(1)}$  recover their desired chunks from these vectors.

User 1 retrieves (sub)chunks  $b, c_1$ , and  $e_1$  from its cache memory to form the  $n(1 - r_1)$ -length frozen-bit vector

$$\mathbf{f}_1 = e_1 \parallel [0 \cdots 0], \quad (15)$$

where  $\parallel$  stands for string concatenation and  $[0 \cdots 0]$  is an all-zero vector of length  $n(r_{\text{max}}^{(1)} - r_4)$ . It applies the standard Polar decoding algorithm with the frozen bit vector  $\mathbf{f}_1$  to the outputs  $\mathbf{y}_1^{(1)}$ , where it ensures again that these frozen bits are used in

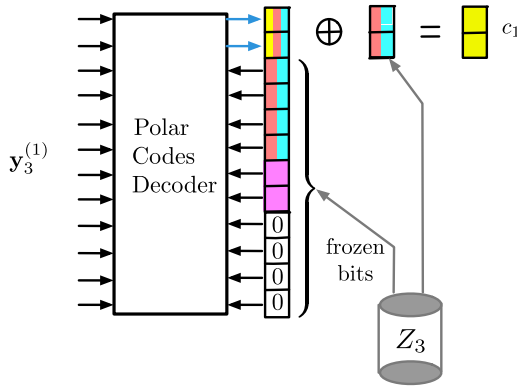


Fig. 5. Decoding procedure at the cache-aided User 3.

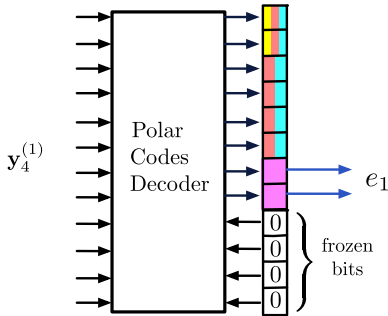


Fig. 6. Decoding procedure at the cache-free User 4.

decreasing reliability order. If the Polar decoder succeeds, it produces the string

$$\bar{m}_{\text{cache},1}^{(1)} := [\bar{m}_a \oplus \bar{m}_b \oplus \bar{m}_c]_{nr_1}, \quad (16)$$

where the operator  $[x]_\ell$  returns the first  $\ell$  bits of the bit-string  $x$ . In this case, user 1 can obtain the desired chunk  $a = [\bar{m}_a]_{nr_1}$  by forming

$$a = \bar{m}_{\text{cache},1}^{(1)} \oplus \underbrace{[\bar{m}_b]_{nr_1}}_{=b} \oplus [\bar{m}_c]_{nr_1}. \quad (17)$$

User 2 proceeds in an entirely analog way, where it simply exchanges the roles of the chunks  $a$  and  $b$  and uses its outputs  $\mathbf{y}_2$  instead of  $\mathbf{y}_1$ .

User 3 retrieves (sub)chunks  $a$ ,  $b$ , and  $e_1$  from its cache memory, creates the  $n(1 - r_3)$ -length frozen-bit vector

$$f_3 = a_2 \oplus b_2 \parallel a_3 \oplus b_3 \parallel e_1 \parallel [0 \cdots 0], \quad (18)$$

and applies the Polar decoding algorithm with this frozen bit vector  $f_3$  to the received vector  $\mathbf{y}_3^{(1)}$ . If the Polar decoder succeeds, it produces

$$\bar{m}_{\text{cache},3}^{(1)} := a_1 \oplus b_1 \oplus c_1, \quad (19)$$

and user 3 can decode its desired subchunk  $c_1$  by forming

$$c_1 = \bar{m}_{\text{cache},3}^{(1)} \oplus a_1 \oplus b_1. \quad (20)$$

The cache-free user 4 applies the Polar decoder to  $\mathbf{y}_4^{(1)}$  with the standard all-zero frozen bit vector of length  $n(1 - r_{\max}) = 1/3n$ . If successful, the Polar decoder returns the XORed

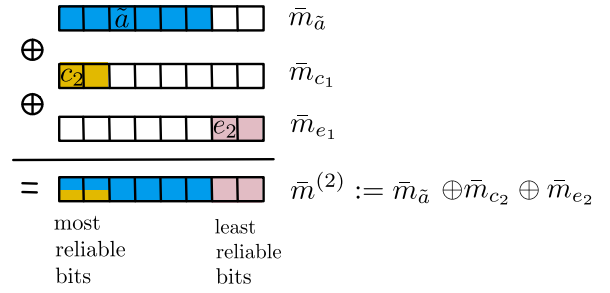


Fig. 7. Creation of “information bits” for the Polar encoder used in block 2.

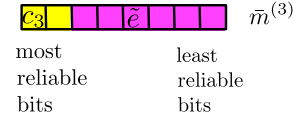


Fig. 8. Creation of “information bits” for the Polar encoder used in block 3.

sequence  $\bar{m}^{(1)}$ , and user 4 can obtain its desired chunk  $e_1$  by retrieving the appropriate bits:

$$e_1 = [\bar{m}^{(1)}]_{(nr_{\text{cache,max}}+1):nr_4}. \quad (21)$$

*Channel Encoding and Decodings in Block 2 (Fig. 7):* The encoder appends  $n(r_{\max} - r_1)$  and  $n(r_{\max} - r_3)$  zeros at the end of the two (sub)chunks  $\tilde{a}$  and  $c_2$ , and it appends  $nr_{\text{cache,max}}^{(2)} = n/2$  zeros at the beginning of subchunk  $e_2$ . Then it takes the componentwise XOR of the three resulting  $nr_{\max}$ -length strings  $\bar{m}_{\tilde{a}}$ ,  $\bar{m}_{c_2}$ ,  $\bar{m}_{e_2}$ :

$$\bar{m}^{(2)} := \bar{m}_{\tilde{a}} \oplus \bar{m}_{c_2} \oplus \bar{m}_{e_2}. \quad (22)$$

Finally, the server applies a standard blocklength- $n$  Polar encoder to the information bits  $\bar{m}^{(2)}$ , where it ensures that these bits are mapped to the information bits of the Polar code in decreasing order of reliability (Z-parameter).

Decoding is similar to block 1, but where the users consider the block-2 outputs  $\mathbf{y}_1^{(2)}$ ,  $\mathbf{y}_3^{(2)}$ ,  $\mathbf{y}_4^{(2)}$ , chunk  $a$  is replaced by  $\tilde{a}$ , subchunks  $c_1$  and  $e_1$  by  $c_2$  and  $e_2$ , and chunk  $b$  and the corresponding zero-padded string  $\bar{m}_b$  are ignored.

*Channel encoding and decodings in block 3 (Fig. 8):* The encoder directly forms the  $nr_{\max}$ -length bit string  $\bar{m}^{(3)}$  by appending  $\tilde{e}$  after  $c_3$ :

$$\bar{m}^{(3)} := [c_3 \parallel \tilde{e}]. \quad (23)$$

It applies a standard blocklength- $n$  Polar encoder to the information bits  $\bar{m}^{(3)}$ , where it respects the decreasing order of reliability.

To decode, User 3 applies a standard Polar decoder to its block-3 outputs  $\mathbf{y}_3^{(3)}$  using the  $n(1 - r_3) = 5n/6$ -length frozen bit vector

$$f_3 = \tilde{e} \parallel [0 \cdots 0]. \quad (24)$$

If successful, the Polar decoder directly returns  $c_3$ .

User 4 applies the Polar decoder to  $\mathbf{y}_4^{(3)}$  with the standard all-zero frozen bit vector of length  $n(1 - r_{\max}) = n/3$ . If successful, the Polar decoder returns  $\bar{m}^{(3)}$ , and user 4 obtains



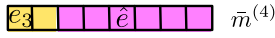


Fig. 9. Creation of “information bits” for the Polar encoder used in block 4.

its desired chunk  $\tilde{e}$  by retrieving the appropriate bits:

$$e_1 = \left[ \tilde{m}^{(3)} \right]_{(nr_{\text{cache,max}}+1):nr_4} \quad (25)$$

*Channel Encoding and Decodings in Block 4 (Fig. 9):* The encoder directly forms the  $nr_{\text{max}}$ -length bit string  $\tilde{m}^{(4)}$  by appending  $\hat{e}$  to  $e_3$ :

$$\tilde{m}^{(4)} := [e_3 \parallel \hat{e}]. \quad (26)$$

It applies a standard blocklength- $n$  Polar encoder to the information bits  $\tilde{m}^{(4)}$ .

User 4 applies the Polar decoder to  $\mathbf{y}_4^{(4)}$  with the standard all-zero frozen bit vector of length  $n(1 - r_{\text{max}}) = n/3$ . If successful, the Polar decoder returns  $e_3$  and  $\hat{e}$ .

#### IV. DATA ASSIGNMENT PROCEDURES

Recall that the data assignment procedure assigns all the bits of the non-cached message bits  $\{W'_{d_k}\}_{k \in \mathcal{K}}$  into the various transmission blocks  $b = 1, 2, \dots$

In this paper we propose a data assignment procedure consisting of two subprocedures. A first procedure partitions all subchunks intended to *cache-aided users* into groups, where all subchunks within a group will be transmitted during the same block. The second procedure assigns each subchunk intended to *cache-free users* to one of the groups established in the first procedure. We describe the details of both procedures in the following two subsections.

It is assumed that the server has an estimate of the data rates at which the various users can decode and that a code of blocklength  $n$  is used for transmission over the channel.

##### A. Step 1: Assignment of (Sub)Chunks for Cache-Aided Users—Minimum Graph Colouring on Conflict Graphs

Similar to [14], [33]–[35], the relevant information about the various cache contents is captured in a *conflict graph*. The conflict graph however has to be adapted to account for the fact that users will be served at different rates. We describe two procedures, a stronger procedure with higher implementation complexity and a weaker procedure that is simpler. The two procedures are also explained in form of pseudo codes in Algorithms 3 and 4. Both procedures rely on the *Greedy Randomized Adaptive Search Procedure* (GRASP) for graph colouring introduced in [33] as well as on a *Conflict graph* construction. For completeness, these building stones are recalled in the pseudo codes of Algorithms 1 and 2, respectively.

1) *More Complex Procedure:* For each cache-aided user  $k \in \mathcal{K}_{\text{cache}}$ , the server splits each chunk of  $W'_{d_k}$  into  $\lceil \frac{F_{\text{chunk}}}{nr_k} \rceil$  subchunks, each one consisting of no more than  $nr_k$  bits, and constructs a conflict graph based on these *subchunks*. Specifically, it assigns a vertex to each subchunk of  $\{W'_{d_k}\}_{k \in \mathcal{K}_{\text{cache}}}$  and labels this vertex by a pair  $(l, k)$ , where

---

##### Algorithm 1: GRASP Graph Colouring Procedure from [33]

---

**Result:** GRASP Graph Colouring of vertices  $\mathcal{V}$  with respect to edge set  $\mathcal{E}$  using randomization parameter  $\beta \in [0, 1]$  and *MaxI* Iterations

```

for  $i = 1, \dots, \text{MaxI}$  do
  Initialize colouring  $\mathcal{C} = \emptyset$ ;
   $\tilde{\mathcal{V}} = \mathcal{V}$ ;
  for  $\ell = 1, \dots, |\mathcal{V}|$  do
    Set  $d_{\min}$  and  $d_{\max}$  to smallest and largest degrees in  $\tilde{\mathcal{V}}$ ;
    Randomly pick a vertex  $\tilde{v} \in \tilde{\mathcal{V}}$  with degree at least  $d_{\min} + \beta(d_{\max} - d_{\min})$ ;
    if  $\exists$  colour  $c$  of  $\mathcal{C}$  that is not assigned to any neighbor of  $\tilde{v}$  then
      | Add  $\tilde{v}$  to  $\mathcal{C}$  with colour  $c$ ;
    else
      | Add  $\tilde{v}$  to  $\mathcal{C}$  with a new colour  $c'$ ;
    end
    Remove  $\tilde{v}$  from  $\tilde{\mathcal{V}}$ ;
  end
end
 $\mathcal{C}^* = \mathcal{C}$ ;
for all colours  $c$  of  $\mathcal{C}$  do
  for all  $\tilde{v} \in \mathcal{C}$  of colour  $c$  do
    | If possible, assign a different colour  $\tilde{c} \neq c$  of  $\mathcal{C}$  to  $\tilde{v}$ ;
  end
  if colour  $c$  has been completely eliminated from  $\mathcal{C}$  then
    |  $\mathcal{C}^* = \mathcal{C}$ ;
  else
    |  $\mathcal{C} = \mathcal{C}^*$ ;
  end
end
Return  $\mathcal{C}^*$ ;

```

---



---

##### Algorithm 2: Conflict\_Graph( $\mathcal{V}$ )

---

**Result:** Construction of Conflict Graph  $\mathcal{E}$  on a set of chunks or subchunks  $\mathcal{V}$

```

Initialize edge set of  $\mathcal{V}$  to  $\mathcal{E} = \emptyset$ ;
for all distinct pairs  $v = (l, k) \in \mathcal{V}$  and  $\tilde{v} = (\tilde{l}, \tilde{k}) \in \mathcal{V}$  do
  if  $l$ -th (sub)chunk of  $W'_{d_k}$  is not in  $Z_{\tilde{k}}$  OR  $\tilde{l}$ -th (sub)chunk of  $W'_{d_{\tilde{k}}}$  is not in  $Z_k$  then
    | Edge  $\mathcal{E}(v, \tilde{v}) = \text{true}$ ;
  end
end
Return  $\mathcal{E}$ ;

```

---

$k$  indicates the intended user and  $l$  the number of the subchunk within  $W'_{d_k}$ . Two vertices  $(l, k)$  and  $(\tilde{l}, \tilde{k})$  are connected in the conflict graph *unless* the  $l$ -th subchunk of  $W'_{d_k}$  is cached at user  $\tilde{k}$  and the  $\tilde{l}$ -th subchunk of  $W'_{d_{\tilde{k}}}$  is cached at



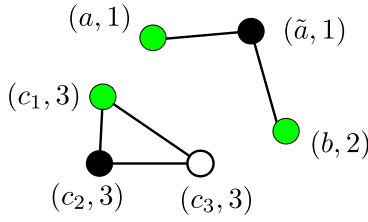


Fig. 10. Illustration of the subchunk conflict-graph with a minimum graph colouring for the example in Section III-B. This conflict graph/graph colouring is used in the more complex procedure.

---

**Algorithm 3: More Complex Data-Assignment for Cache-Aided Users**


---

**Result:** Assignment of all bits of  $\{W'_{d_k}\}_{k \in \mathcal{K}_{\text{cache}}}$  to blocks  $b = 1, 2, \dots$ ,

Initialize vertex set  $\mathcal{V}_{\text{sub}} = \emptyset$ ;

**for**  $k \in \mathcal{K}_{\text{cache}}$  **do**

    Split each chunk of  $W'_{d_k}$  into  $\lceil \frac{F_{\text{chunk}}}{nr_k} \rceil$  disjoint subchunks with at most  $F_{\text{chunk}}$  bits;

    For each subchunk add a vertex  $v = (l, k)$  to  $\mathcal{V}_{\text{sub}}$ , where  $l$  is a unique subchunk identifier;

**end**

$\mathcal{E}_{\text{sub}} = \text{Conflict\_Graph}(\mathcal{V}_{\text{sub}})$ ;

$\mathcal{C}_{\text{sub}} = \text{GRASP\_Graph\_Colouring}(\mathcal{V}_{\text{sub}}, \mathcal{E}_{\text{sub}})$ ;

To each transmission block  $b = 1, 2, \dots$  assign all subchunks (vertices) of  $\mathcal{C}_{\text{sub}}$  of a given colour  $c$ ;

---

user  $k$ . After constructing the conflict graph, the server runs a minimum graph colouring algorithm such as the GRASP-algorithm proposed in [33] on the subchunk conflict-graph. All subchunks with the same colour are assigned to the same transmission block  $b = 1, 2, \dots$

The subchunk conflict graph of the example in Section III-B and a possible minimum colouring are shown in Fig. 10, where for simplicity, subchunks are labeled with letters  $a, \tilde{a}, b, c_1, c_2, c_3$  as introduced in Section III-B. (Correspondence with the notation in the previous paragraph is obtained by noting that  $a$  and  $\tilde{a}$  are chunks of file  $W'_{d_1}$  and are thus denoted  $(1, 1), (2, 1)$  in the text;  $b$  is a chunk of file  $W'_{d_2}$  and thus is denoted  $(1, 2)$  in the text; and  $c_1, c_2, c_3$  are subchunks of file  $W'_{d_3}$  and thus denoted  $(1, 3), (2, 3), (3, 3)$  in the text.) As we see, (sub)chunks  $a, b, c_1$  are assigned to the same colour and will thus be sent in the same subblock, (sub)chunks  $\tilde{a}$  and  $c_2$  have the same colour and will be sent in the same block, and subchunk  $c_3$  has a colour on its own and will be sent in a separate third block.

For large file sizes  $F$  and small rates  $r_k$  the set  $\{W'_{d_k}\}$  consists of many subchunks. In this case, the complexity of the proposed graph colouring procedure can be prohibitively large. In fact, the GRASP graph colouring algorithm of [33] has complexity  $O(|\mathcal{V}_{\text{sub}}|^2)$ , for  $\mathcal{V}_{\text{sub}}$  denoting the set of subchunks. The size of this set grows proportionally with the number of chunks (and thus with the file size  $F$ ) and inversely proportional with the rates  $\{r_k\}_{k \in \mathcal{K}_{\text{cache}}}$ .

2) *Simpler Procedure:* In the following, we explain a simpler (generally less efficient) way to assign the subchunks

---

**Algorithm 4: Simpler Data-Assignment for Cache-Aided Users**


---

**Result:** Assignment of all bits of  $\{W'_{d_k}\}_{k \in \mathcal{K}_{\text{cache}}}$  to blocks  $b = 1, 2, \dots$ ,

Initialize vertex sets  $\mathcal{V}_{\text{chunk}} = \tilde{\mathcal{V}}_{\text{chunk}} = \emptyset$ ;

**for**  $k \in \mathcal{K}_{\text{cache}}$  **do**

    For each chunk of  $W'_{d_k}$  add a vertex  $v = (j, k)$  to  $\mathcal{V}_{\text{chunk}}$ , where  $j$  is a unique chunk-identifier;

**end**

$\mathcal{E}_{\text{chunk}} = \text{Conflict\_Graph}(\mathcal{V}_{\text{chunk}})$ ;

$b=1$ ;

**while**  $\mathcal{V}_{\text{chunk}}$  *nonempty* **do**

**if**  $\mathcal{V}_{\text{chunk}} \neq \tilde{\mathcal{V}}_{\text{chunk}}$  **then**

$\mathcal{C}_{\text{chunk}} = \text{GRASP\_Graph\_Colouring}(\mathcal{V}_{\text{chunk}}, \mathcal{E}_{\text{chunk}})$ ;

**end**

$\tilde{\mathcal{V}}_{\text{chunk}} = \mathcal{V}_{\text{chunk}}$ ;

**for** colours  $c = 1, 2, \dots$  of  $\mathcal{C}_{\text{chunk}}$  **do**

**for**  $v = (j, k) \in \mathcal{C}_{\text{chunk}}$  of colour  $c$  **do**

            Assign the first  $nr_k$  bits of the  $j$ -th chunk of  $W'_{d_k}$  to transmission block  $b$ ;

            Remove the assigned bits from the  $j$ -th chunk of  $W'_{d_k}$ ;

**if**  $j$ -th chunk of  $W'_{d_k}$  is empty **then**

                Remove  $v = (j, k)$  from  $\mathcal{V}_{\text{chunk}}$

**end**

**end**

$b \rightarrow b + 1$ ;

**end**

**end**

---

to the various transmission blocks. The idea is to create the conflict graph as in [14], i.e., a node of the conflict graph is associated to an *entire chunk* and is labeled by  $(l, k)$ , where  $k$  denotes the intended cache-aided user and  $l$  the number of the chunk within  $W'_{d_k}$ . As before, two vertices  $(l, k)$  and  $(l', k')$  are connected in this graph *unless* the  $l$ -th chunk of  $W'_{d_k}$  is cached at user  $k'$  and the  $l'$ -th chunk of  $W'_{d_{k'}}$  is cached at user  $k$ .

In this simplified procedure, the server assigns the subchunks to the various transmission blocks in an iterative way. In each round, it constructs a conflict graph containing the remaining chunks. If the conflict graph has changed from the previous iteration, the server applies a minimum graph-colouring algorithm on the *chunk* conflict-graph, and otherwise it uses the same graph colouring as in the previous iteration. Then, it retrieves for each node  $(l, k)$  in the conflict graph the next subchunk of size  $nr_k$  bits, and assigns all subchunks of the same colour to the same transmission block. It further eliminates all chunks from the graph, for which all subchunks have already been assigned and starts the next round.

Fig. 11 shows the simplified procedure (the conflict graphs and a minimum graph colouring for each round) for the example in Section III-B. In round 1 the server retrieves the entire chunks  $a$  and  $b$ , because users 1 and 2 are served at rates  $r_1 = r_2 = 1/2$  and in the example  $F_{\text{chunk}} = n/2$ , and it retrieves subchunk  $c_1$  because user 3 is served at rate  $r_3 = 1/6$ ,

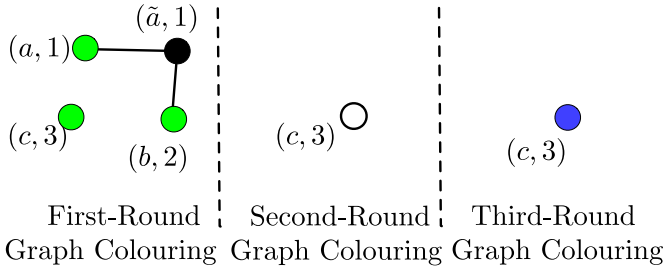


Fig. 11. Illustration of the chunk conflict-graph with an iterative minimum graph colouring for the example in Section III-B. This conflict graph/graph colouring is used in the simpler procedure.

which corresponds to a single sub-chunk. The server then assigns  $a, b, c_1$  to transmission block 1 because they have the same colour, and it assigns  $\tilde{a}$  to the second transmission block, because it has a different colour. In round 2, the server retrieves subchunk  $c_2$  (and in round 3 it retrieves subchunk  $c_3$ ), because  $r_3 = 1/6$ , and assigns it to the next transmission block. In this simpler procedure we use 4 colours and thus 4 transmission blocks, compared to only 3 for the more complex procedure. So the reduced complexity of the simpler procedure can come at the expense of reduced performance.

Irrespective of which of the two procedures is used, the server assigns the subchunks to the transmission blocks in a way that each subchunk transmitted in a given block is stored at all users served in this block except, of course, the user to which it is intended.

### B. Step 2: Assignment of (Sub)Chunks for Cache-Free Users—Maximum Matching on Bipartite Graph

We describe how to assign the subchunks intended for the cache-free users to the various transmission blocks, or rather to the groups (colours) formed in the previous subsection. We summarize this assignment in Algorithm 5. At first, the server constructs a bi-partite graph that on the left-hand side (LHS) contains for each block  $b = 1, 2, \dots, B_{\max, \text{cache}}$  a node with all subchunks  $\{(l, k_{\text{cache}}) : k_{\text{cache}} \in \mathcal{G}_{\text{cache}}^{(b)}\}$  that we assigned to block  $b$  in the previous Section IV-A, and on the right-hand side (RHS) it contains a node for each chunk of  $\{W'_{d_k}\}_{k \in \mathcal{K}_{\text{nocache}}}$  (i.e., each chunk intended for a cache-free user). There is an edge between a node on the LHS  $\{(l, k_{\text{cache}}) : k_{\text{cache}} \in \mathcal{G}_{\text{cache}}^{(b)}\}$  and a node  $(l, k)$  on the RHS if, and only if, all users  $k_{\text{cache}} \in \mathcal{G}_{\text{cache}}^{(b)}$  have stored the  $l$ -th chunk of  $W'_{d_k}$  in their cache memories. (So here the edges are constructed in a different way than in the conflict graph.)

Consider again an iterative procedure, where in each round the server runs a maximum bipartite matching algorithm on this graph. For each edge in the resulting matching, which connects on the LHS a group of chunks  $\{(l, k_{\text{cache}}) : k_{\text{cache}} \in \mathcal{G}_{\text{cache}}^{(b)}\}$  and on the RHS a single chunk  $(l, k)$ ,  $k \in \mathcal{K}_{\text{nocache}}$ , the server retrieves a subchunk of size  $nr_k$  bits (if possible) from chunk  $(l, k)$  and it assigns this subchunk to the same transmission block  $b$ . It then retrieves the freshly assigned bits from the chunk on the RHS, and retrieves the chunk all-together from the graph if it is empty. Moreover, the group of

---

### Algorithm 5: Data-Assignment for Cache-Free Users

---

**Result:** Assignment of all bits of  $\{W'_{d_k}\}_{k \in \mathcal{K}_{\text{nocache}}}$  to blocks  $b = 1, 2, \dots$  assuming  $B_{\max, \text{cache}}$  is the number of blocks assigned in Algorithms 3 or 4 and  $\mathcal{G}_{\text{cache}}^{(b)}$  contains all users with bits assigned to block  $b$ .

Set  $\mathcal{V}_{\text{left}} = \{1, \dots, B_{\max, \text{cache}}\}$ ;  
Initialize vertex set  $\mathcal{V}_{\text{right}} = \emptyset$ ;  
**for**  $k \in \mathcal{K}_{\text{nocache}}$  **do**  
    For each chunk of  $W'_{d_k}$  add a vertex  $v = (j, k)$  to  $\mathcal{V}_{\text{right}}$ , where  $j$  is a unique chunk identifier;  
**end**  
Initialize edge set  $\mathcal{E} = \emptyset$ ;  
**for**  $b \in \mathcal{V}_{\text{left}}, v = (l, k) \in \mathcal{V}_{\text{right}}$  **do**  
    **if**  $l$ -th chunk of  $W'_{d_k}$  is in all cache memories of users  $\mathcal{G}_{\text{cache}}^{(b)}$  **then**  
         $\mathcal{E}(b, v) = \text{true}$   
    **end**  
**end**  
**while**  $\mathcal{V}_{\text{left}} \neq \emptyset$  and  $\mathcal{V}_{\text{right}} \neq \emptyset$  **do**  
    Find maximum matching  $\mathcal{E}_{\text{MaxMatching}}$  of bipartite graph  $(\mathcal{V}_{\text{left}}, \mathcal{V}_{\text{right}}, \mathcal{E})$ ;  
    **for all edges in**  $e \in \mathcal{E}_{\text{MaxMatching}}$  **do**  
        Let  $b_e$  the left-vertex of  $e$  and  $v_e = (l, k)$  the right-vertex;  
        **if**  $l$ -th chunk of  $W'_{d_k}$  contains more than  $nr_k$  bits **then**  
            Assign next  $nr_k$  bits of  $l$ -th chunk of  $W'_{d_k}$  to block  $b_e$ ;  
            Remove the assigned bits from  $l$ -th chunk of  $W'_{d_k}$ ;  
        **else**  
            Assign all bits of  $l$ -th chunk of  $W'_{d_k}$  to block  $b_e$ ;  
            Remove node  $v_e = (l, k)$  from  $\mathcal{V}_{\text{right}}$ ;  
        **end**  
        Remove node  $b_e$  from  $\mathcal{V}_{\text{left}}$ ;  
    **end**  
**end**  
**if**  $\mathcal{V}_{\text{right}} \neq \emptyset$  **then**  
    Assign the remaining bits to new blocks  
     $b = B_{\max, \text{cache}} + 1, \dots$ ;  
**end**

---

chunks on the LHS of the edge is also removed.<sup>3</sup> After having treated all the edges of the bipartite matching, the procedure is restarted in a next iteration, now using the modified bipartite graph. The iteration stops when the bipartite graph is empty, or when it only contains nodes on one of the two sides. In the latter case, remaining nodes on the RHS (if any) are grouped to transmission blocks in a way that the total number of bits

<sup>3</sup>An improved algorithm could be envisioned where bits to multiple cache-free users can be assigned to the same block. For simplicity we refrain from using this generalization.

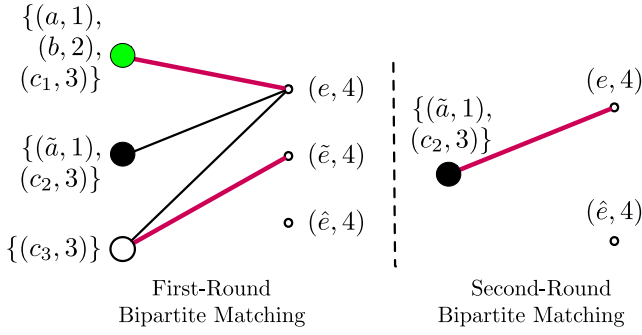


Fig. 12. Illustration of the iterated bipartite graph and a maximum bipartite matching for the example in Section III-B.

sent in each new block  $b = B_{\max, \text{cache}} + 1, \dots$  does not exceed  $nr_k$ , for any user  $k \in \mathcal{K}_{\text{nocache}}$  served in this block.

Figure 12 shows the procedure for our example of Section III-B, assuming that the complex procedure is used in Step 1. In the first round, the bipartite matching connects chunk  $e$  to the group  $\{a, b, c_1\}$ . Since user 4 is meant to decode at rate  $r_{\max} = 2/3$  and users 1 and 2 at rates  $r_1 = r_2 = 1/2$  (user 3 at a smaller rate  $1/6$ ), a subchunk  $e_1$  of size  $n/6$  is assigned to the same transmission block as the group  $(a, b, c_1)$ . Moreover, the group  $(a, b, c_1)$  is retrieved from the graph, but not chunk  $e$ . The matching in this first round also connects chunk  $\tilde{e}$  to subchunk  $\{c_3\}$ . Since user 4 is of rate  $2/3$  and subchunk  $c_3$  of rate  $1/6$ , the entire chunk  $\tilde{e}$ , which is of rate  $1/2$ , is assigned to the same transmission block as  $c_3$ . Both (sub)chunks  $c_1$  and  $\tilde{e}$  are removed from the graph for the next round. In the second round, the bipartite matching connects the group  $\{\tilde{a}, c_2\}$  to  $e$ , and the server thus assigns subchunk  $e_2$  which is of rate  $1/6$  (because  $\tilde{a}$  is of rate  $1/2$  and user 4 is meant to decode  $r_{\max} = 2/3$ ) to the same transmission block as the group  $\{\tilde{a}, c_2\}$ . Both end points of the edge can be removed from the graph for the next round. In the third round, the graph thus consists only of two chunks  $e$  (actually only the subchunk  $e_3$  associated with  $e$ ) and  $\hat{e}$  from the RHS. As their sizes are compatible, these (sub)chunks are concatenated and assigned to one new transmission block.

## V. CHANNEL CODING IN A SINGLE BLOCK—THE GENERAL SCHEME

In the preceding Section IV, we explained how the various subchunks of files  $\{W'_{d_k}\}$  are assigned to the different transmission blocks. In this section, we explain the channel encoding and decoding in a given block  $b = 1, 2, \dots$ . For ease of notation we shall drop the subscript  $b$  indicating the block.

We simply assume that the data assignment procedure in the previous section associates with the current block the sets of cache-aided users  $\mathcal{G}_{\text{cache}} \in \mathcal{K}_{\text{cache}}$  and cache-free users  $\mathcal{G}_{\text{nocache}} \in \mathcal{K}_{\text{nocache}}$  and for each  $k \in \mathcal{G} := \mathcal{G}_{\text{cache}} \cup \mathcal{G}_{\text{nocache}}$  a bit-string  $m_k$  of length  $r_k n$  bits. Define

$$r_{\text{cache}, \max} := \max_{k \in \mathcal{G}_{\text{cache}}} r_k \quad (27)$$

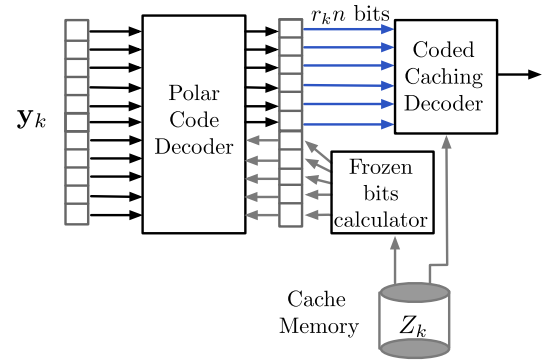


Fig. 13. Example of the delivery decoding for a single block at a cache-aided user  $k \in \mathcal{K}_{\text{cache}}$ .

and in case  $\mathcal{G}_{\text{nocache}}$  is nonempty

$$r_{\max} := r_{\text{cache}, \max} + \sum_{k \in \mathcal{G}_{\text{nocache}}} r_k \quad (28)$$

As we will see, transmission in this block is reliable if each cache-aided user  $k \in \mathcal{G}_{\text{cache}}$  can reliably decode at rate  $r_k$  and each cache-free user  $k \in \mathcal{G}_{\text{nocache}}$  can decode at rate  $r_{\max}$ . Denote by  $y_k$  user  $k$ 's output observed in this block.

*Encoding:* For any *cache-aided user*  $k \in \mathcal{G}_{\text{cache}}$ , the encoder appends  $n(r_{\max} - r_k)$  zeros *at the end* of bit-string  $m_k$  to form the  $n$ -length bitstring  $\tilde{m}_k$ , and combines all these strings to

$$\tilde{m}_{\text{cache}} = \bigoplus_{k \in \mathcal{G}_{\text{cache}}} \tilde{m}_k. \quad (29)$$

It further constructs the bit-string  $\tilde{m}_{\text{nocache}}$  by concatenating all bits of strings  $\{m_k : k \in \mathcal{G}_{\text{nocache}}\}$  and appends  $nr_{\text{cache}, \max}$  zeros *at the beginning* of the string:

$$\tilde{m}_{\text{nocache}} = \underbrace{[0 \dots 0]}_{nr_{\text{cache}, \max} \text{ symbols}} \parallel m_{k_{\text{nocache}, 1}} \parallel \dots \parallel m_{k_{\text{nocache}, |\mathcal{G}_{\text{nocache}}|}}, \quad (30)$$

where for ease of notation we denote the users of  $\mathcal{G}_{\text{nocache}}$  by  $k_{\text{nocache}, 1}, \dots, k_{\text{nocache}, |\mathcal{G}_{\text{nocache}}|}$ . The encoder then feeds the  $nr_{\max}$ -length bit-string

$$\tilde{m} = \tilde{m}_{\text{cache}} \oplus \tilde{m}_{\text{nocache}}, \quad (31)$$

to a standard blocklength- $n$  Polar encoder, for which it ensures that the Polar code assigns the  $n$  bits of  $\tilde{m}$  to the information bits and the frozen bits of the Polar code in decreasing order of reliability ( $Z$ -parameter). It finally transmits this codeword over the channel.

*Decoding at Cache-Aided User  $k \in \mathcal{G}_{\text{cache}}$*  (Fig. 13): User  $k \in \mathcal{G}_{\text{cache}}$  retrieves the bit-strings  $\{m_j : j \in \mathcal{G}_{\text{cache}} \setminus \{k\}\}$  and  $\{m_j : j \in \mathcal{G}_{\text{nocache}}\}$  from its cache memory, forms the zero-padded or concatenated strings  $\{\tilde{m}_j : j \in \mathcal{G}_{\text{cache}} \setminus \{k\}\}$  and  $\tilde{m}_{\text{nocache}}$  as described above to compute

$$\tilde{m}_{\text{dec}, k} = \left( \bigoplus_{j \in \mathcal{G}_{\text{cache}} \setminus \{k\}} \tilde{m}_j \right) \oplus \tilde{m}_{\text{nocache}}. \quad (32)$$

It divides this  $nr_{\max}$ -length bit-string as

$$c_k := [\tilde{m}_{\text{dec}, k}]_{nr_k} \quad (33)$$

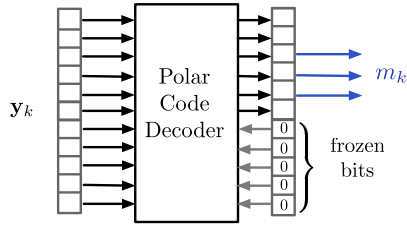


Fig. 14. Example of the delivery decoding for a single block at a user  $k_{\text{nocache}}$  without cache.

$$f_k := [\bar{m}_{\text{dec},k}]_{(nr_k+1):nr_{\text{max}}}, \quad (34)$$

and applies the standard Polar decoding algorithm with the  $n(1-r_k)$ -length frozen-bit vector  $f_k \parallel [0 \cdots 0]$  to its output vector  $\mathbf{y}_k$ , where it ensures that the bits are used in decreasing reliability order. If the Polar decoder succeeds, it produces the  $nr_k$ -length bit-string  $[\bar{m}_{\text{cache}}]_{nr_k}$  and user  $k$  can retrieve its desired bits  $m_k = [\bar{m}_k]_{nr_k}$  by forming:

$$m_k = [\bar{m}_{\text{cache}}]_{nr_k} \oplus c_k. \quad (35)$$

*Decoding at Cache-Free User  $k \in \mathcal{G}_{\text{cache}}$  (Fig. 14):* Each cache-free user  $k \in \mathcal{G}_{\text{nocache}}$  applies a standard Polar decoder with an all-zero frozen bit vector  $f_k$  of length  $n(1-r_{\text{max}})$  to its outputs  $\mathbf{y}_k$ . If successful, the Polar decoder returns the XORed sequence  $\bar{m}$  in (31). User  $k$  can then obtain the desired message  $m_k$  by retrieving the appropriate bits from  $\bar{m}$ . Specifically, when  $k$  is the  $i$ -th user in  $\mathcal{G}_{\text{nocache}}$ , i.e.,  $k = k_{\text{nocache},i}$ , user  $k$  obtains  $m_k$  as

$$m_k = [\bar{m}]_{n(r_{\text{cache,max}} + \sum_{j=1}^{i-1} r_{k_{\text{nocache},j}} + 1):n(r_{\text{cache,max}} + \sum_{j=1}^i r_{k_{\text{nocache},j}})}. \quad (36)$$

## VI. NUMERICAL RESULTS

This section presents the bit-error rates (BER) and the sum-throughputs of our proposed scheme obtained through computer simulation and a testbed implementation. In our computer simulations we compare the performance of our proposed scheme also to previous schemes.

This section is organized into three subsections: Section VI-A explains the common setup for both the computer simulations and the testbed implementation; Section VI-B presents the simulation results; and Section VI-C the testbed measurements.

### A. Practical Setup

1) *Physical Layer Model:* A QPSK-OFDM modulation is used with 64 carriers (48 data carriers, 4 pilot carriers, and 12 null carriers) and a cyclic prefix of length 16. A bandwidth of 1 MHz is considered, and thus the OFDM symbol duration is  $T = 80 \mu\text{s}$ . All physical layer parameters are presented in Table I.

2) *Codeword and Frame Model:* We use Polar codes of blocklength  $n = 2048$ . Since we employ a QPSK modulation, any codeword requires 1024 channel uses. Moreover, since only 48 subcarriers of each OFDM symbol are used to transmit codeword symbols, any codeword is spread over

TABLE I  
PARAMETERS OF THE PHYSICAL LAYER

Bandwidth (MHz)	$B = 1$
Sampling Period ( $\mu\text{s}$ )	$T_s = 1$
Number of carriers per OFDM symbol	$N_c = 64$
Number of data carriers per OFDM symbol	48
Length of cyclic prefix per OFDM symbol	$N_{cp} = 16$
OFDM symbol duration ( $\mu\text{s}$ )	$T = 80$

$\lceil 1024/48 \rceil = 22$  OFDM symbols. Any frame contains a single codeword and in the testbed implementation also various headers, where these latter contain information for channel decoding (e.g., number of XORs, chunks involved in the XOR, etc), synchronization, and channel estimation. We denote by  $N_h$  the overhead in OFDM symbols for each frame due to headers. In our numerical simulations we simply assume  $N_h = 0$ . For the testbed measurements we will have a variable-length  $N_h$ , which depends on the number of users served by the codeword; in our testbed implementation it is mostly  $N_h = 24$ . We conclude that since each OFDM symbol spans 80 channel uses, a frame spans in total  $80(22 + N_h)$  channel uses. The headers that we use in our testbed measurements are shown in Figs. 23 and 24 at the end of the paper and discussed in more details in Section VI-C. They are based on the implementation of the simpler coded caching scheme in [37] and could be reduced in size. For the parameters used in this paper ( $K = 5$  users, 22 OFDM symbols per codeword, and  $N_h \approx 24$ ), the header reduces the throughput approximately by a factor 2. The throughput loss would be even larger for increasing number of users or increasing number of (sub)chunks. (For  $K = 32$  users, which is the typical number of active users scheduled in a 4G cell, and all rates either  $r_{\text{min}} = 1/6$  or integer multiples thereof,  $N_h$  would be around 30 and the header would reduce the rate approximately by a factor  $2/5$ .) Notice that any sort of coded caching scheme encounters the problem of a increased header size. Coded caching however also allows to reduce the number of transmitted packets, and thus the number of headers, which can partly mitigate this degradation, see [37] for more information. Further improvements can be obtained with the coded caching schemes proposed in [39]–[41], which operate using fewer chunks (i.e., having smaller subpacketization levels). The data assignment and channel coding are done according to the procedures described in Sections III–IV, where for the data assignment we use the simplified procedure.

3) *Figures of Merit:* Two figures of merit have been considered for evaluating the cache-aided coding scheme.

- *Per-User BER:* The  $\text{BER}_k$  of user  $k$  is the number of unsuccessfully transmitted information bits for user  $k$  during the delivery phase (i.e., the number of wrongly-decoded bits of  $W'_{d_k}$  at user  $k$ ) divided by the total number of information bits transmitted to user  $k$  during the delivery phase:

$$\text{BER}_k := \frac{\text{\#wrongly-decoded bits of } W'_{d_k}}{\text{\#bits of } W'_{d_k}}. \quad (37)$$

- *Sum-Throughput*: The sum-throughput  $\eta$  is the number of successfully transmitted information bits to any of the  $K$  users, divided by the total number of channel uses (i.e., the number of samples sent by the transmitter during the entire transmission window). Since we always transmit entire codewords (plus headers), we have:

$$\eta := \frac{\sum_k \#\text{successfully-decoded bits of } W'_{dk}}{80 \cdot (22 + N_h) \cdot \#\text{transmitted codewords}}. \quad (38)$$

4) *Coding Schemes*: We compare our proposed coding scheme (which allows XORing strings of different sizes and appends (sub)chunks to cache-free users) to three other schemes.

- *Uncoded Caching*: Users are successively served through individual transmissions. No multicast is performed. (Notice that because of the chosen normalizations the same BER and throughput curves also apply to a standard transmission scheme without caching.)
- *Coded Caching [2]*: All XORed strings are of same length as in [2], and thus the rate of the XOR needs to be adjusted to the weakest cache-aided user served by the XOR. Chunks for cache-free users are sent in separate codewords.
- *Generalized Coded Caching [19]*: XORed strings can be of different lengths. Chunks for cache-free users are sent in separate codewords.

In our simulations of the coded caching scheme, we use the implementation of [37] combined with simple Polar code transmissions of the multicast messages to the cache-aided users. Cache-free users are scheduled in separate transmission blocks where they are served in a traditional manner using Polar codes. The generalized coded caching scheme is implemented using the data assignment and channel coding procedures described in Sections III–IV but without cache-free users. Cache-free users are again served using separate standard Polar codes.

The same placement procedure is used for all the schemes.

## B. Simulation Results

In our simulations we emulate a frequency-selective channel as described in Eq. (4), where the path attenuations are given by [42]

$$a_{k,m}(t) = A_{k,m} \cdot \text{SoS}_{k,m}(j), \quad t \in ((j-1) \cdot T, j \cdot T], \quad (39)$$

with  $\{|A_{k,m}|^2\}_m$  the power delay profile of user  $k$ . As  $T$  is the OFDM symbol duration, the path attenuation may change every OFDM symbol through the term  $\text{SoS}_{k,m}(j)$  which is obtained by

$$\text{SoS}_{k,m}(j) = \text{SoS}_{k,m}^{(\text{Re})}(j) + i \cdot \text{SoS}_{k,m}^{(\text{Im})}(j) \quad (40)$$

with

$$\text{SoS}_{k,m}^{(\text{Re})}(j) = \frac{1}{\sqrt{L_s}} \sum_{\ell=1}^{L_s} \cos \left( 2\pi f_d j T \cos \left( \frac{2\pi \ell - \pi + \theta_{k,m}(j)}{4L_s} + \phi_{k,m,\ell} \right) \right) \quad (41a)$$

TABLE II  
PARAMETERS OF THE CHANNEL MODEL

Number of paths	$L = 3$
Delay Profile ( $\mu\text{s}$ )	$\forall k, \tau_{k,1} = 0, \tau_{k,2} = 2, \tau_{k,3} = 4$
Power Delay Profile	$\forall k,  A_{k,1} ^2=0.5,  A_{k,2} ^2=0.3,  A_{k,3} ^2=0.2$
Number of sin.	$L_s = 8$
Central Frequency (GHz)	$f_0 = 2.4$
Doppler frequency (Hz)	$f_d = 40$
$\delta$	$10^{-6}$

and

$$\text{SoS}_{k,m}^{(\text{Im})}(j) = \frac{1}{\sqrt{L_s}} \sum_{\ell=1}^{L_s} \cos \left( 2\pi f_d j T \sin \left( \frac{2\pi \ell - \pi + \theta_{k,m}(j)}{4L_s} + \psi_{k,m,\ell} \right) \right). \quad (41b)$$

Here,  $\theta_{k,m}(j)$  is the random walk process defined as:

$$\theta_{k,m}(j) = [\theta_{k,m}(j-1) + \delta \cdot u_{k,m}(j)]_{-\pi}^{\pi}, \quad j = 1, 2, \dots, \quad (42)$$

where  $u_{k,m}(j)$  is uniformly distributed over  $[0, \pi)$  and independent over OFDM symbols  $j = 1, 2, \dots$ , paths  $m = 1, \dots, L$ , and users  $k$ ;  $\theta_{k,m}(0)$  is uniformly distributed over  $[-\pi, \pi)$ ; and the operator  $[\cdot]_{-\pi}^{\pi}$  stands for  $\max(-\pi, \min(\cdot, \pi))$ . Notice that the sign of  $\delta$  is modified when  $\theta_{k,m}$  reaches the value  $\pi$  or  $-\pi$ . The terms  $\phi_{k,m,\ell}$  and  $\psi_{k,m,\ell}$  are i.i.d. processes uniformly distributed over  $[-\pi, \pi)$ . The parameters in Eqs. (41)–(42) are chosen according to Table II. Notice that the Doppler frequency corresponds to walking speed.

According to [42], the correlation in time within one path is given by  $E[\text{SoS}_{k,m}(j)\text{SoS}_{k,m}(j+j')^*] = J_0(2\pi f_d j' T)$  where  $J_0$  is the zero-order Bessel function of the first kind. If we define the coherence time of the channel, denoted by  $T_c$ , as the closest zero of the autocorrelation function, we get

$$T_c \approx \frac{2.4}{2\pi f_d}.$$

According to Tables I–II, we have  $T_c = 9,549\mu\text{s}$ , which corresponds to 119 OFDM symbols. The channel is thus constant over one OFDM symbol as required. Diversity is only ensured through frequency diversity because any given codeword experiences only one channel realization.

In our numerical simulations, all users are assumed to have perfect channel state information. As mentioned, we neglect the headers, and thus set  $N_h = 0$ . We consider a system with  $N = 20$  files and  $K = 5$  users.

We average our simulation results over 500 realizations of the random user demands  $d_1, \dots, d_K$  drawn according to a Zipf-distributed file popularity (see Section II) and 200 realizations of the random channel outcome. The parameter of the Zipf distribution is set to  $s = 1$  unless otherwise stated, which corresponds to a non-uniform distribution. The caching parameter is set to  $\alpha = 0.1$ . In our setup, we have  $\alpha p_n N \leq 1$  for all files  $n = 1, \dots, N$ , and as a result each cache-aided user stores a fraction  $\alpha p_n N$  of all chunks of file  $W_n$  in its



TABLE III  
CONFIGURATION 1 WITH A SINGLE CACHE-FREE USER

Users	1	2	3	4	5
SNR offset	0 dB	2 dB	2 dB	5 dB	7 dB
Cache-aided	yes				no
Data rate	$r_1 = \frac{1}{6}$	$r_2 = \frac{1}{3}$	$r_3 = \frac{1}{3}$	$r_4 = \frac{1}{2}$	$r_{\max} = \frac{2}{3}$

TABLE IV  
CONFIGURATION 2 WITH TWO CACHE-FREE USERS

Users	1	2	3	4	5
SNR offset	0 dB	2 dB	5 dB	6 dB	8 dB
Cache-aided	yes			no	
Data rate	$r_1 = \frac{1}{6}$	$r_2 = \frac{1}{3}$	$r_3 = \frac{1}{2}$	$r_{\max} = \frac{2}{3}$	

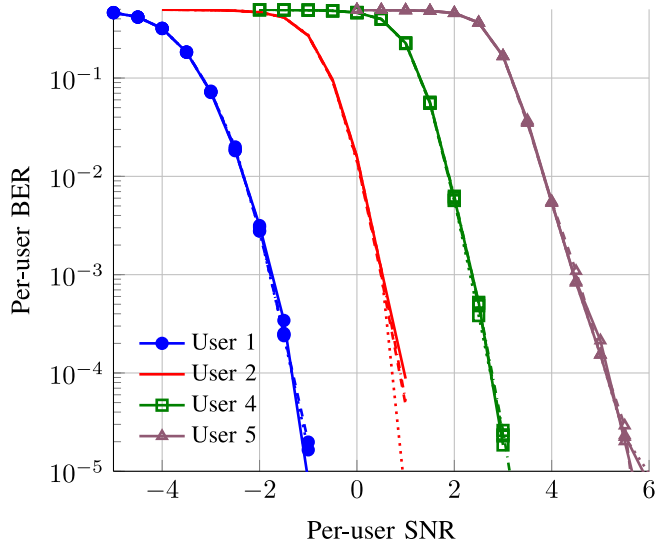


Fig. 15. Per-user BERs versus the per-user SNRs over a Gaussian channel under Configuration 1. Solid line: *proposed scheme*; dashed line: *generalized coded caching*; dashdotted line: *coded caching*; dotted line: *uncoded caching*.

cache memory, for each  $n = 1, \dots, N$ . The size of each file is  $F = 66\,528$  bits. Each file consists of  $N_{\text{chunk}} = 33$  chunks and each chunk corresponds to  $F_{\text{chunk}} = 2016$  bits, which is almost the size of a single codeword.

We study the two user configurations (cache availability, SNR offset, and data rate) presented in Tables III and IV.

Let us first consider a memoryless Gaussian channel. In Fig. 15, we plot the per-user BER over a Gaussian channel versus the per-user SNR for different caching schemes under Configuration 1, where only User 5 has no cache memory. User 3's performances are not plotted since they are very close to those of User 2, as the two have same SNR offsets and same coding rates  $r_2 = r_3$ . We remark that the per-user BER is almost regardless of the applied scheme, and the SNR shifts roughly correspond to the pre-defined SNR offsets. Since in our proposed scheme the subchunks for user 5 are mostly sent on less reliable bits of the Polar codes (they are mostly appended after the XORs) than for the other schemes, this implies that the Polar code is well "polarized" in the sense that all reliable bits have approximately same probabilities of error. Similar behaviors can be observed for Configuration 2. Figs. 16 and 17 show the sum-throughput  $\eta$  in function of the SNR at the weakest user 1 for the various schemes,

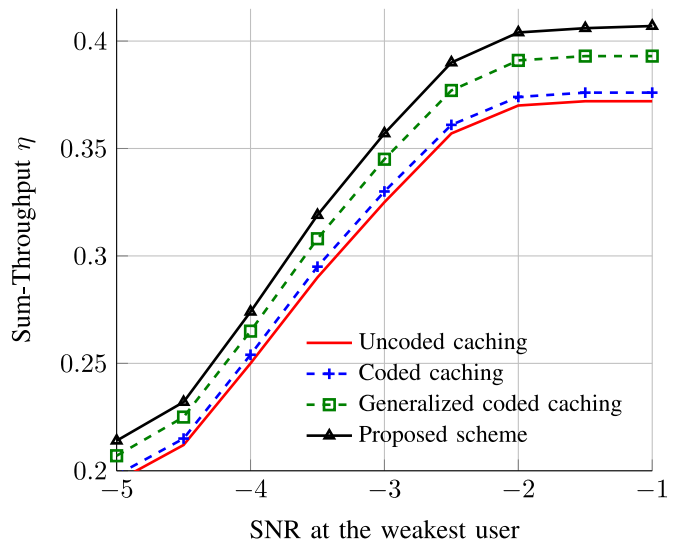


Fig. 16. Sum-throughputs  $\eta$  versus SNR at the weakest user 1 over a Gaussian channel under Configuration 1.

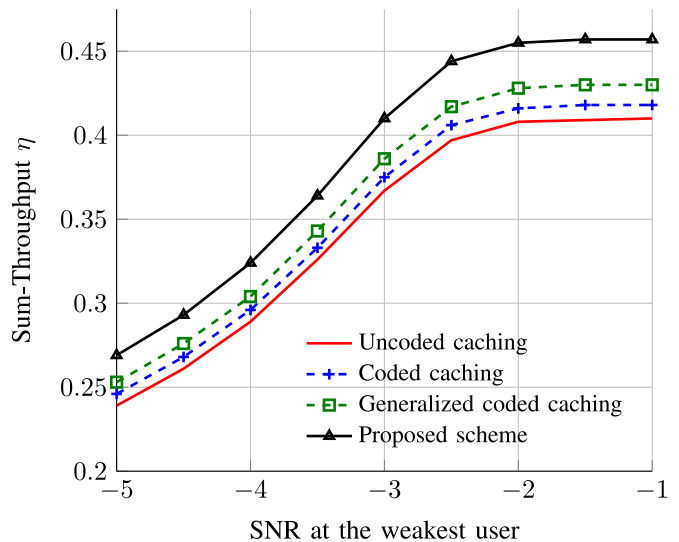


Fig. 17. Sum-throughputs  $\eta$  versus SNR at the weakest user 1 over a Gaussian channel under Configuration 2.

under Configurations 1 and 2, respectively. We remark a 10% increase in throughput of the proposed scheme compared to a scheme with only uncoded caching. We further observe that the gain is progressively achieved by introducing coded caching, XORing strings of different rates, and appending communications to cache-free users on the multi-cast XORs for cache-aided users. The last feature of our proposed scheme seems to provide the largest gains in these examples. This is more visible in Configuration 2 where the number of cache-free users is higher and thus appending their request chunks to the XORed strings within the same polar codeword is of greater interest.

We now consider the frequency-selective channel described previously. In Fig. 18, we plot the per-user BER versus the per-user SNRs for the various schemes and under Configuration 1. (User 3's BER curve is close to User 2's curve and omitted.) Like for the Gaussian channel, the BER curves of the



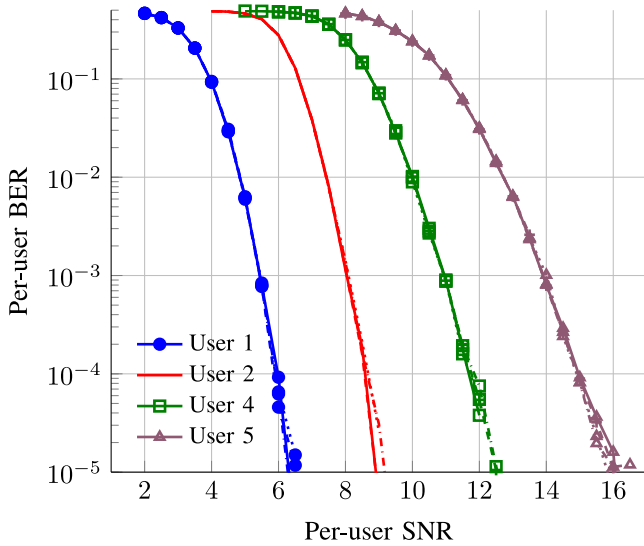


Fig. 18. Per-user BERs versus per-user SNRs over the specified frequency-selective channel under Configuration 1. Solid lines: *proposed scheme*; dashed lines: *generalized coded caching*; dashdotted lines: *uncoded caching*; dotted lines: *uncoded caching*.

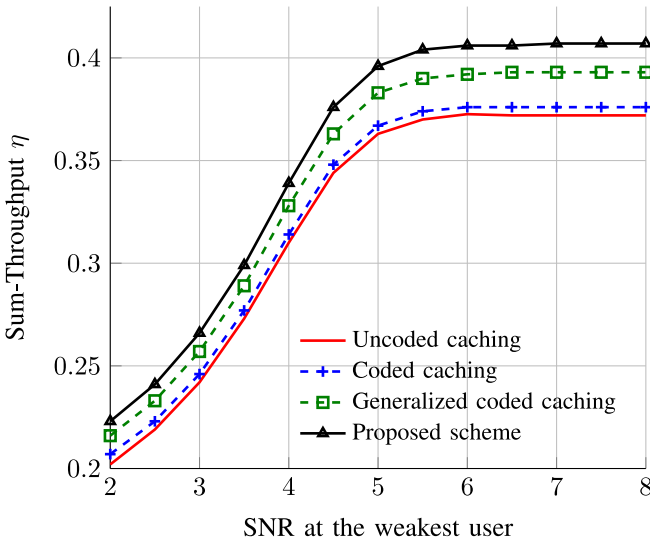


Fig. 19. Sum-throughputs  $\eta$  versus SNR at the weakest user 1 over the specified frequency-selective channel under Configuration 1.

various schemes behave analogously, and the same conclusion applies as for the Gaussian channel. Figs. 19 and 20 show the sum-throughput  $\eta$  in function of the SNR at the weakest user 1 for the various schemes and under Configurations 1 and 2, respectively. Our proposed scheme achieves again a significant throughput gain over the coded caching scheme. We further remark that for large SNR values the same throughput is achieved as on the Gaussian channel because in this case most codewords are successfully decoded over both channels.

In Fig. 21, we plot the sum-throughput  $\eta$  over the specified frequency-selective channel in function of the Zipf parameter  $s$  for a SNR at the weakest user equal to 8dB (high SNR regime) under Configuration 1. We remark that the gap between caching approaches increases when  $s$  decreases, i.e., when the popularity of the files becomes more uniform.

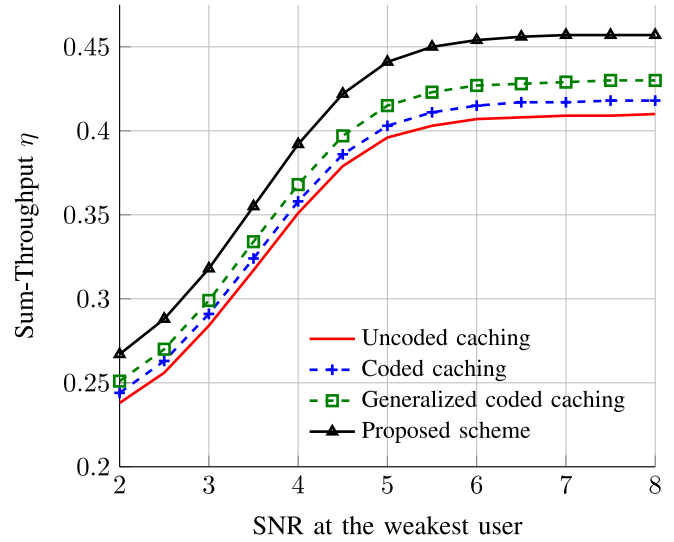


Fig. 20. Sum-throughputs  $\eta$  versus SNR at the weakest user 1 over the specified frequency-selective channel under Configuration 2.

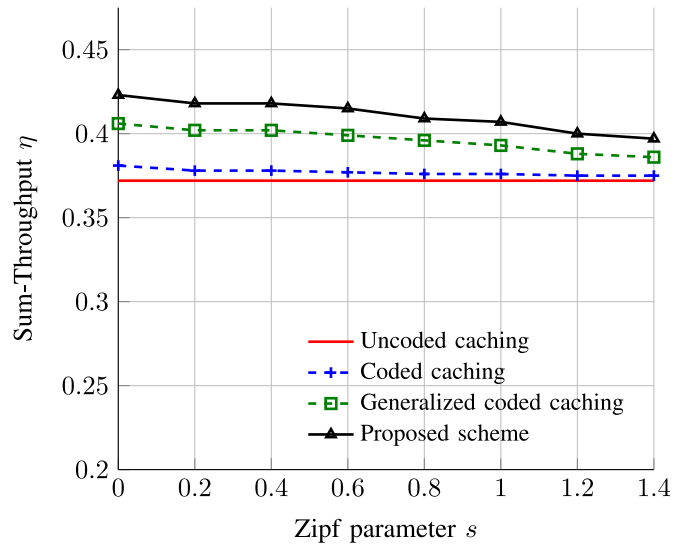


Fig. 21. Sum-throughputs  $\eta$  versus  $s$  for SNR at the weakest user of 8dB over the specified frequency-selective channel under Configuration 1.

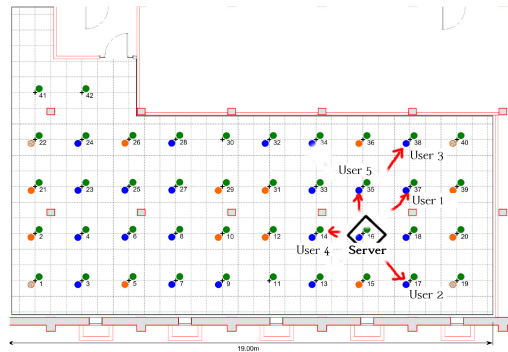


Fig. 22. Testbed room.

### C. Testbed Measurements

The proposed scheme of Sections III–IV (with the simplified conflict graph construction in Section IV) was also tested on the FIT/Cortexlab testbed (<http://www.cortexlab.fr/>), which

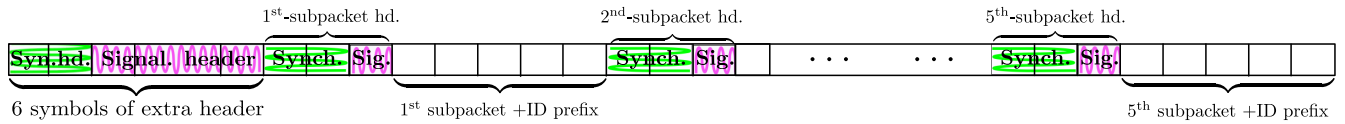


Fig. 23. Illustration of the frame structure related to a single codeword.

is composed of SISO or MIMO software defined radio (SDR) nodes [36]. The testbed is hosted in a  $180\text{ m}^2$  shielded room, which is partly covered with electromagnetic wave absorbing material, and its layout is shown in Fig. 22. User nodes are placed over a regular grid with an inter-node distance of 1.8 meters, and allow any physical layer implementation on both hardware and software. A unified server is available for starting, coordinating, and collecting the results of the experiments. As a development tool, the GNU Radio software is employed for real-time experimentation.

The physical layer of CorteXlab is described in Table I and is closely related to IEEE802.11p, except for the occupied bandwidth.

The coefficients  $\{h_k(f, j)\}$  in Eq. (5) are not time-varying since the nodes and the electromagnetic environment are both static, and thus the channel coefficients  $\{h_k(f, j)\}$  do not depend on  $j$ . Echoes are mainly attenuated by the wave-absorbing walls and can be neglected. The channel is thus almost flat and the coefficients  $\{h_k(f, j)\}$  do not depend on  $f$  neither. Consequently, the channel between the server and the user  $k$  is a static Gaussian channel (AWGN) with an attenuation factor depending on the server-user distance and the quality of the hardware at the user node.

The frame related to a single Polar codeword has the following structure, see also Figure 23. Notice that the framing structure has not been optimized for our context but follows the general framework of FIT/CorteXlab and the structure proposed in [37]. The codeword is split into 5 subpackets, each of length at most  $\lceil 2048/5 \rceil = 410$  bits, and an ID prefix of 16 bits is added at the beginning of each subpacket to number the subpacket. Since the symbols are QPSK-modulated and each OFDM symbol contains 48 data carriers, each subpacket including its ID prefix can be sent with  $\lceil (410+16)/2/48 \rceil = 5$  OFDM symbols. For synchronization and signaling purposes, a header consisting of 3 OFDM symbols (2 for synchronization and 1 for signaling) is added to each subpacket+ID prefix. At the beginning of the five subpackets+ID prefix+headers pertaining to the same codeword, an extra header of variable-length is added, where the first 2 symbols are meant for synchronization and the rest for signaling. The signaling header is described in Fig. 24, where compared to the header in [37] we added a field indicating whether a cache-free user is served (according to the data assignment procedure in Section IV-B, we serve at most one cache-free user with a single codeword) and for each served user we indicate the size of its subchunk as well as the corresponding subchunk ID numbers. (In our implementation each chunk is cut into portions of  $r_{\min}n$  bits and a different ID is associated to each portion. These IDs correspond to the subchunk ID numbers.) The header is variable length because the number of

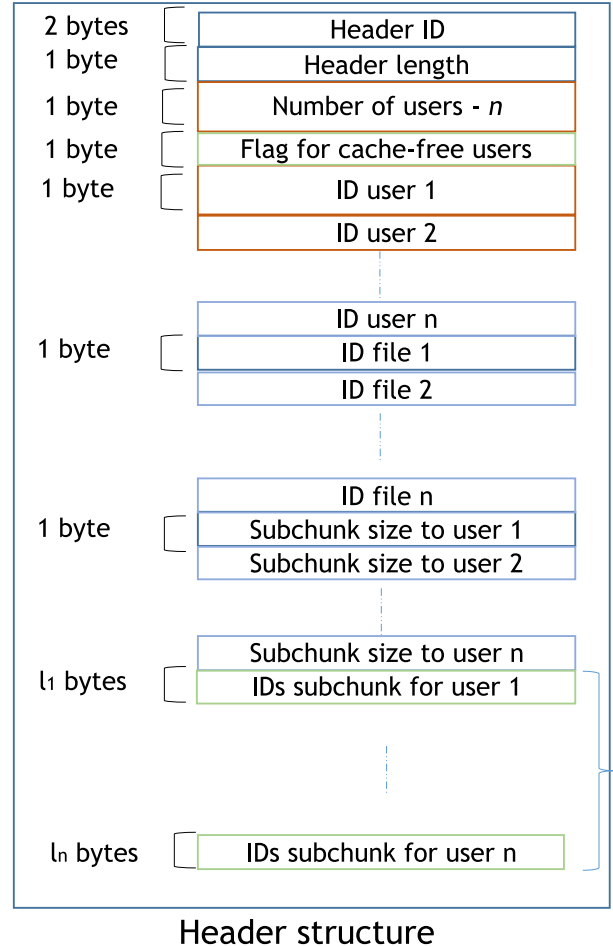


Fig. 24. Illustration of the entries of the extra signaling header.

served users varies over the codewords and because we chose a variable-length description for the subchunk IDs. (Please see [43] for more details on the practical implementation of the header.) In our set-up with  $K = 5$  users and  $r_{\min} = 1/6$ , the signaling header of most frames consists of 4 OFDM symbols. Under this assumption, the frame corresponding to each codeword consists of  $6 + 5(3 + 5) = 46$  OFDM symbols. Since the information in a codeword would fit into  $\lceil 2048/2/48 \rceil$  OFDM symbols, the overhead related to the headers in a typical frame is  $N_h = 46 - \lceil 2048/2/48 \rceil = 46 - 22 = 24$  symbols.

Synchronization is performed using the well-known Schmidl & Cox timing and frequency offset estimators [44], and channel estimation using the Least Square method [45]. Synchronization information in the headers is related to these algorithms. The signaling information in the extra headers describes the frame ID, the users served by the packet, the number and the IDs of the subchunks combined in the XOR,

TABLE V  
USERS CONFIGURATION IN THE TESTBED SETUP

Users	1	2	3	4	5
SNR offset	0 dB	-1 dB	-13 dB	-11 dB	3 dB
Cache-aided	yes				no
Data rate	$r_1 = \frac{1}{6}$	$r_2 = \frac{1}{3}$	$r_3 = \frac{1}{3}$	$r_4 = \frac{1}{2}$	$r_{\max} = \frac{2}{3}$

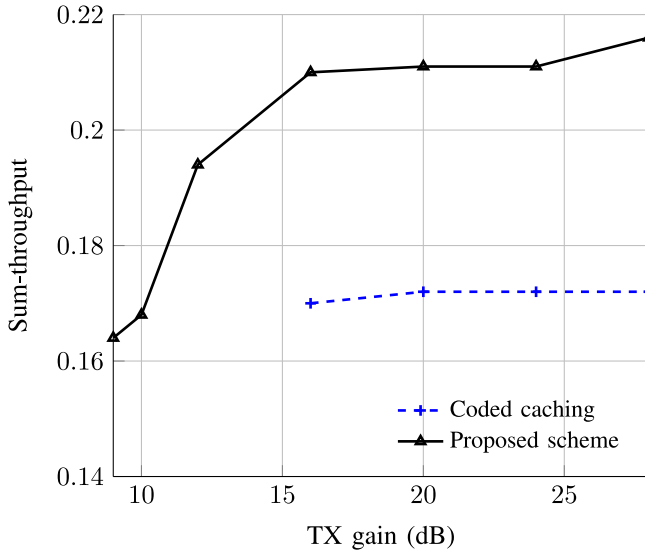


Fig. 25. Sum-throughputs  $\eta$  versus TX gain for our proposed scheme and for the coded caching scheme.

and the chunks intended for users without cache memories. The subpacket's header provides only information about physical layer parameters used in the corresponding subpacket.

To decrease the probability of packet loss, the transmitter power-boosts the 6 OFDM symbols of the extra header at the beginning of each frame by +12dB.

Our testbed implementation uses 6 nodes of the FIT/CorteXlab testbed as illustrated in Figure 22: 1 represents the server and the remaining 5 represent users. Among these 5 users, 4 use the hard disk at their nodes as cache memories (Users 1–4) but the last one doesn't. Table V indicates the relative SNR offsets between the five users, whether they have cache memories, and our choice of the data rates used for the proposed scheme. Measurements are performed over 10 realizations of the random user demands  $d_1, \dots, d_k$ , where the parameter of the Zipf distribution is set to  $s = 1$ .

In Fig. 25, we display the sum-throughput  $\eta$  of our proposed scheme and of the standard coded caching scheme in function of a transmitter (TX) gain that is artificially introduced at the transmitter side. We remark that at high TX gains the sum-throughput is roughly half the sum-throughput of Figure 16 which has been simulated under the same cache configuration and the same choice of rates but under different SNR offsets (see Tables III and V). At high SNR, the effect of the SNR offsets between the users and of the channel statistics vanishes because all users can decode with almost zero probability of error. The rate is simply determined by the size of the headers (which corresponds roughly to half the frame in this measurement setup) and the ability of combining different chunks in

the same codewords, which only depends on the choice of the rates but not on the channels.

## VII. CONCLUDING REMARKS

We have introduced a cache-aided coding scheme for single-server multi-user networks in a decentralized caching scenario. The scheme is based on Polar-codes and a novel data-assignment technique. Its main novelties are to allow to XOR strings of different sizes according to the capacity of the cache-aided users and to append data intended to the cache-free users. These appended data has to be cached at all other users in a transmission and so can be used as “frozen bits” in their Polar decodings. The paper further proposes a new data assignment algorithm for the assignment of data and users to be served by each Polar codeword. The new algorithm is based on iterative applications of the GRASP minimum graph-colouring algorithm, which allows to account for different coding rates at the various users, and a bipartite matching algorithm, which allows to combine also transmissions to cache-free users. Numerical simulations and real testbed measurements of this practical coded caching scheme were performed, and showed that our new practical scheme achieves higher throughputs than the existing schemes even in the finite block-length regime.

## REFERENCES

- [1] M. A. Maddah-Ali and U. Niesen, “Fundamental limits of caching,” *IEEE Trans. Inf. Theory*, vol. 60, no. 5, pp. 2856–2867, May 2014.
- [2] M. A. Maddah-Ali and U. Niesen, “Decentralized coded caching attains order-optimal memory-rate tradeoff,” *IEEE/ACM Trans. Netw.*, vol. 23, no. 4, pp. 1029–1040, Aug. 2015.
- [3] Z. Chen, P. Fan, and K. B. Letaief, “Fundamental limits of caching: Improved bounds for users with small buffers,” *IET Commun.*, vol. 10, no. 17, pp. 2315–2318, 2016.
- [4] C. Tian, “A note on the fundamental limits of coded caching,” 2015, [Online]. Available: <http://arxiv.org/abs/1503.00010>.
- [5] K. Wan, D. Tuninetti, and P. Piantanida, “On caching with more users than files,” in *Proc. IEEE Int. Symp. Inf. Theory*, Jul. 2016, pp. 135–139.
- [6] S. Sahraei and M. Gastpar, “K users caching two files: An improved achievable rate,” in *Proc. Annu. Conf. Inf. Sci. Syst.*, Mar. 2016, pp. 620–624.
- [7] C. Tian and J. Chen, “Caching and delivery via interference elimination,” *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1548–1560, Mar. 2018.
- [8] M. M. Amiri, Q. Yang, and D. Gündüz, “Coded caching for a large number of users,” in *Proc. IEEE Inf. Theory Workshop (ITW)*, Sep. 2016, pp. 171–175.
- [9] 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.
- [10] J. Gómez-Vilardebó, “Fundamental limits of caching: Improved bounds with coded prefetching,” *IEEE Trans. Commun.*, vol. 66, no. 10, pp. 4488–4497, Oct. 2018.
- [11] H. Ghasemi and A. Ramamoorthy, “Improved lower bounds for coded caching,” *IEEE Trans. Inf. Theory*, vol. 63, no. 7, pp. 4388–4413, Jul. 2017.
- [12] A. Sengupta, R. Tandon, and T. C. Clancy, “Improved approximation of storage-rate tradeoff for caching via new outer bounds,” in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2015, pp. 1691–1695.
- [13] R. Pedarsani, M. A. Maddah-Ali, and U. Niesen, “Online coded caching,” *IEEE/ACM Trans. Netw.*, vol. 24, no. 2, pp. 836–845, Apr. 2016.
- [14] 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.
- [15] 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.

- [16] R. Timo and M. Wigger, "Joint cache-channel coding over erasure broadcast channels," in *Proc. Int. Symp. Wireless Commun. Syst. (ISWCS)*, 2015, pp. 201–205.
- [17] S. S. Bidokhti, M. Wigger, A. Yener, and A. El Gamal, "State-adaptive coded caching for symmetric broadcast channels," in *Proc. Asilomar Conf. Signals Syst. Comput.*, 2017, pp. 646–650.
- [18] S. S. Bidokhti, M. Wigger, and R. Timo, "Noisy broadcast networks with receiver caching," *IEEE Trans. Inf. Theory*, vol. 64, no. 11, pp. 6996–7016, Nov. 2018.
- [19] S. S. Bidokhti, M. Wigger, and A. Yener, "Benefits of cache assignment on degraded broadcast channels," *IEEE Trans. Inf. Theory*, vol. 65, no. 11, pp. 6999–7019, Nov. 2019.
- [20] A. S. Cacciapuoti, M. Caleffi, M. Ji, J. Llorca, and A. Tulino, "Speeding up future video distribution via channel-aware caching-aided coded multicasting," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 8, pp. 2207–2218, Aug. 2016.
- [21] 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.
- [22] J. Hachem, N. Karamchandani, and S. Diggavi, "Content caching and delivery over heterogeneous wireless networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2015, pp. 756–764.
- [23] 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.
- [24] S. Yang, K. H. Ngo, and M. Kobayashi, "Content delivery with coded caching and massive MIMO in 5G," in *Proc. Int. Symp. Turbo-Codes Iterative Inf. Process. (ISTC)*, 2016, pp. 370–374.
- [25] 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.
- [26] X. You, Y. Wu, and J. Chen, "Cache-aided broadcast channels with user cooperation," in *Proc. Annu. Allerton Conf. Commun. Control Comput. (Allerton)*, 2019, pp. 544–551.
- [27] J. Chen, H. Yin, X. You, Y. Geng, and Y. Wu, "Centralized coded caching with user cooperation," in *Proc. IEEE Inf. Theory Workshop (ITW)*, 2019, pp. 1–5.
- [28] M. Salman and M. K. Varanasi, "An upper bound on the capacity-memory tradeoff of interleavable discrete memoryless broadcast channels with uncoded prefetching," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2020, pp. 1687–1692.
- [29] M. M. Amiri and D. Gündüz, "Caching and coded delivery over Gaussian broadcast channels for energy efficiency," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 8, pp. 1706–1720, Aug. 2018.
- [30] E. Arıkan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, Jul. 2009.
- [31] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Trans. Inf. Theory*, vol. 61, no. 5, pp. 2213–2226, May 2015.
- [32] Y. Birk and T. Kol, "Informed-source coding-on-demand (ISCOD) over broadcast channels," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 1998, pp. 1257–1264.
- [33] G. Vettigli, M. Ji, A. M. Tulino, J. Llorca, and P. Festa, "An efficient coded multicasting scheme preserving the multiplicative caching gain," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2015, pp. 251–256.
- [34] M. Ji, A. M. Tulino, J. Llorca, and G. Caire, "On the average performance of caching and coded multicasting with random demands," in *Proc. Int. Symp. Wireless Commun. Syst. (ISWCS)*, 2014, pp. 922–926.
- [35] M. Ji, K. Shanmugam, G. Vettigli, J. Llorca, A. M. Tulino, and G. Caire, "An efficient multiple-groupcast coded multicasting scheme for finite fractional caching," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2015, pp. 3801–3806.
- [36] L. S. Cardoso *et al.*, "CorteXlab: A facility for testing cognitive radio networks in a reproducible environment," in *Proc. 9th Int. Conf. Cogn. Radio Oriented Wireless Netw. Commun. (CROWNCOM)*, Oulu, Finland, Jun. 2014, pp. 503–507. [Online]. Available: <https://hal.inria.fr/hal-01101087>
- [37] Y. Fadlallah, A. M. Tulino, D. Barone, G. Vettigli, J. Llorca, and J.-M. Gorce, "Coding for caching in 5G networks," *IEEE Commun. Mag.*, vol. 55, no. 2, pp. 106–113, Feb. 2017.
- [38] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *Proc. IEEE INFOCOM*, Mar. 1999, pp. 126–134.
- [39] 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.
- [40] 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.
- [41] Q. Yan, M. Cheng, X. Tang, and Q. Chen, "Placement delivery array design through strong edge coloring of bipartite graphs," *IEEE Commun. Lett.*, vol. 22, no. 2, pp. 236–239, Feb. 2018.
- [42] A. Alimohammad, S. F. Fard, B. F. Cockburn, and C. Schlegel, "Compact Rayleigh and Rician fading simulator based on random walk processes," *IET Commun.*, vol. 3, no. 8, pp. 1333–1342, Jul. 2009.
- [43] Y. Fadlallah, O. Oubejja, and S. Kamel, "Cache-Aided Polar Piggyback Coding Implementation for Testbed Measurements." [Online]. Available: [https://perso.telecom-paris.fr/wigger/Cortex\\_code/](https://perso.telecom-paris.fr/wigger/Cortex_code/) (Accessed: Nov. 22, 2021).
- [44] T. M. Schmidl and D. C. Cox, "Robust frequency and timing synchronization for OFDM," *IEEE Trans. Commun.*, vol. 45, no. 12, pp. 1613–1621, Dec. 1997.
- [45] J. van de Beek, O. Edfors, M. Sandell, S. Wilson, and P. Borjesson, "On channel estimation in OFDM systems," in *Proc. IEEE Veh. Technol. Conf. (VTC)*, 1995, pp. 815–819.



**Yasser Fadlallah** (Member, IEEE) received the Telecommunication Engineering Diploma degree from the Faculty of Engineering, Lebanese University, Lebanon, in 2009, the M.S. degree from the Université de Bretagne Occidentale, France, in 2010, and the Ph.D. degree from Télécom Bretagne, France, in 2013. In 2012, he was a visiting Ph.D. student with the Coding and Signal Transmission Laboratory, University of Waterloo, Canada. From 2013 to 2014, he held a Research and Development Engineer position with Orange Labs, Paris. In 2015, he held a postdoctoral research position with INRIA until September 2016, and then joined the University of Sciences and Arts in Lebanon, where he currently holds an associate professor position. His research interests lie in the wireless communications area, focusing on interference management, advanced and low-complexity receivers, wireless caching, and multiple antennas systems.



**Othmane Oubejja** received the M.Sc. degree from Ecole Nationale Supérieure d'Ingénieurs de Limoges, Limoges, France, in 2016. He currently works with STMicroelectronics, Grenoble, France, as a Digital Design Engineer. He majored in electronics and telecommunications and his research interests include digital signal processing, filtering and control, wireless communications, RTL design, and channel coding.



**Sarah Kamel** received the Engineering Diploma degree in communications and computer science from the Faculty of Engineering, Lebanese University, in 2013, and the M.Sc. degree in digital communication systems and the Ph.D. degree in communications and electronics from Telecom ParisTech in 2013 and 2017, respectively. From 2013 to 2017, she was a Ph.D. candidate with CEA LIST Communicating Systems Lab, working on cryptographic and information-theoretic aspects of security in wireless communications. From 2017 to 2018, she was a Postdoctoral Researcher with the Digital Communications Group, Telecom ParisTech. Since 2018, she has been a Research Engineer with the Optical Communication Technology Lab, Huawei Technologies Paris Research Center. Her research interests include machine learning and coding for optical communications, coded caching, physical layer security, polar and lattice codes, and lattice-based cryptography.





**Philippe Ciblat** was born in Paris, France, in 1973. He received the Engineering degree from Telecom Paris and the M.Sc. degree in automatic control and signal processing from University Paris-Saclay, France, both in 1996, the Ph.D. degree from University Gustave Eiffel, France, in 2000, and the HDR degree in 2007. In 2001, he was a Postdoctoral Researcher with the University of Louvain, Belgium. At the end of 2001, he joined the Communications and Electronics Department, Telecom Paris, as an Associate Professor, where he has been a (Full)

Professor since 2011. His research areas include statistical signal processing, signal processing for digital communications, signal processing over graphs, resource allocation, and distributed optimization. He served as an Associate Editor for the *IEEE COMMUNICATIONS LETTERS* from 2004 to 2007. From 2008 to 2012, he served as an Associate Editor and then a Senior Area Editor for the *IEEE TRANSACTIONS ON SIGNAL PROCESSING*. Since 2018, he has been serving as an Associate Editor for the *IEEE TRANSACTIONS ON SIGNAL AND INFORMATION PROCESSING OVER NETWORKS*. From 2014 to 2019, he was a member of IEEE Technical Committee “Signal Processing for Communications and Networking.”



**Jean-Marie Gorce** (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in electrical engineering from the Institut National des Sciences Appliquées (INSA), Lyon, France, in 1993 and 1998, respectively. He was a Co-Founder of the Centre for Innovation, Telecommunications and Integration of Services (CITI Lab), in 2001. He was a Visiting Scholar with Princeton University, Princeton, NJ, USA, from 2013 to 2014. He has been the Principal Investigator of several French and European sponsored projects related to wireless

networks. He is currently the Scientific Coordinator for the Experimental Facility FIT-CorteXlab. He is also a Professor with INSA, Université de Lyon and the Head for Science with INRIA Lyon. He has co-published more than 150 conference and journal articles. His research interests include wireless networking and communication theory, focusing on realistic modeling, wireless system optimization, and performance assessment considering both infrastructure-based and ad-hoc networks. He holds the Internet of Things Industrial and Research Chair of INSA Lyon sponsored by SPIE ICS. He is an Associate Editor of the *EURASIP Journal on Wireless Communications and Networking* (Springer).



**Michèle Wigger** (Senior Member, IEEE) received the M.Sc. degree (with Distinction) in electrical engineering and the Ph.D. degree in electrical engineering from ETH Zurich in 2003 and 2008, respectively. In 2009, she was first a Postdoctoral Fellow with the University of California, San Diego, USA, and then joined Telecom Paris, France, where she is currently a Full Professor. She has held visiting professor appointments with the Technion-Israel Institute of Technology and ETH Zurich. Her research interests are in multiterminal information

theory, in particular in distributed source coding and in capacities of networks with states, feedback, user cooperation, or caching. She has previously served as an Associate Editor for the *IEEE COMMUNICATION LETTERS* and for Shannon Theory for the *IEEE TRANSACTIONS ON INFORMATION THEORY*. From 2016 to 2019, she also served on the Board of Governors of the IEEE Information Theory Society.