Cache-Aided Polar Coding: From Theory to Implementation

Yasser Fadlallah, Othmane Oubejjaa, Sarah Kamel, Philippe Ciblat, Michèle Wigger, Jean-Marie Gorce

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. Finally, real testbed measurements are presented, which validate the practical implementation.

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 further allow to decrease the network load by a multiplicative factor. These works modelled the communication link from the server to the users as a noise-free shared link. Subsequently, [16]–[18] showed that further reductions in network load are possible on this classic single-server multi-user network in asymmetric or fading scenarios by exploiting the noisy nature of the channel through a joint design of the multicast transmissions and channel coding, see also [17], [19]–[28]. These works proved for various scenarios that jointly designed schemes achieve 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]–[18] for the single-server multi-user broadcast network; 2) Provide a prototype implementation of the proposed scheme 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*

Y. Fadlallah is with theCS department, University of Sciences and Arts in Lebanon (USAL), y.fadlallah@usal.edu.lb.

O. Oubejjaa, S. Kamel, P. Ciblat, and M. Wigger are with LTCI, Telecom Paris, IP Paris, F-91120 Palaiseau, France, {oubejjaa,sarah.e.kamel}@gmail.com and {philippe.ciblat,michele.wigger}@telecom-paris.fr.

J.-M. Gorce is with CITI Laboratory, INRIA, jean-marie.gorce@inria.fr.

Funding support is acknowledged from the European Research Council (ERC) under the European Union's Horizon 2020, grant agreement No 715111.

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-padds 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., [29], [30], 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.

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 [31] and solved by applying a minimum graph colouring algorithm to a *conflict graph* that represents the contents that have to be delivered [32]. (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 GRASP [33] graph colouring algorithm for caching 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* [29] 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

test-bed 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:

$$W_1, \cdots, W_N, \tag{1}$$

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 with parameter s as follows

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

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 N_{chunks} *chunks* of equal size F_{chunk}, and each cache-aided user $k \in \mathcal{K}_{\text{cache}}$ independently downloads each chunk of each file W_n to its cache memory Z_k with a given probability αp_n, for a fixed $\alpha \in (0, 1)$. The required cache size thus is

$$\sum_{n=1}^{N} \sum_{m=1}^{N_{\text{chunks}}} x_{m,n} F_{\text{chunk}}, \tag{3}$$

where $\{x_{m,n}\}_m$ is a Bernouilli-distributed sequence with $\Pr(x_{m,n} = 1) = \alpha p_n$. The weak law of large numbers implies that for sufficiently large file sizes $F \gg F_{\text{chunk}}$, i.e., for N_{chunks} large enough, cache size (3) is not much larger than αF with high probability. Equivalently, each user in $\mathcal{K}_{\text{cache}}$ stores approximately a proportion αp_n of each file W_n in its cache memory.

B. Delivery Phase over a Wireless Channel

In the subsequent *the 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.

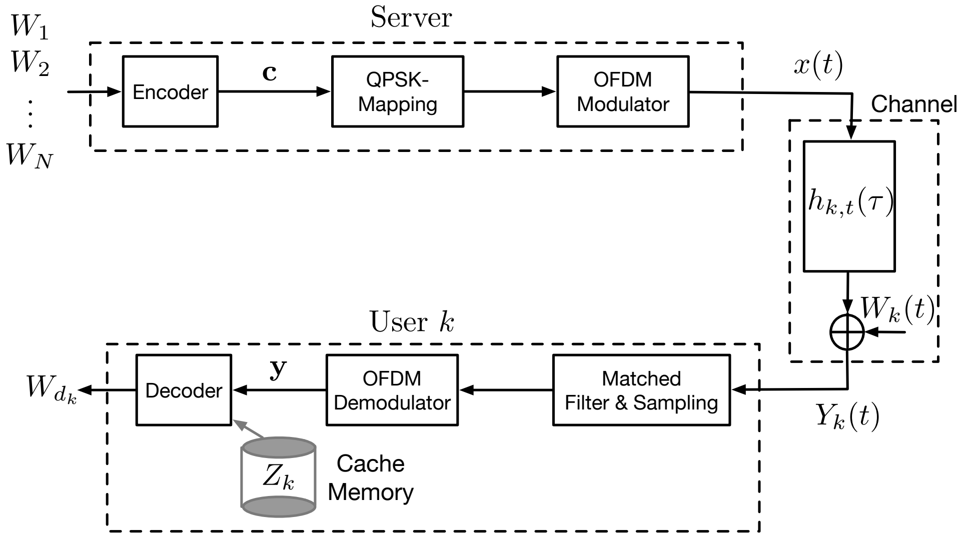

Fig. 1: System for Delivery Phase Transmission.

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:

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

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.

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 Figure 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 low-pass filter, sampler with sampling period T_s, and OFDM demodulator. Parameters N_c and N_{cp} are chosen[1] such that for each subcarrier $f \in \{1, \ldots, N_c\}$ and OFDM symbol $j = 1, 2, \ldots$, the relation between the corresponding QPSK symbol $x(f, j)$ and output $\tilde{y}_k(f, j)$ can be written as

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

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

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

[1] we consider that $N_{cp}T_s$ smaller than the dispersion time of the channel given in Eq. (4), and T much smaller than the coherence time of the channel. Consequently, the amplitude $a_{k,m}(t)$ remains unchanged during the duration of a single OFDM symbol.

2) Encoding and Decoding: The server constructs the input signal bit-stream $\mathbf{c}$ in function of all messages $W_1, \ldots, W_N$, the users' demands $d_1, \ldots, d_K$ and their cache contents $Z_1, \ldots, Z_{K_0}$. 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 and IV) 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 IV, 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.

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 $b = 1, 2, \ldots$. 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 subsection III-B. Then next 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 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:

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

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

For the purpose of the example, chunks c and e are divived 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 Subsection II-A, Users 1–3 have the following cache contents:

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

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

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

(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

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

$$W'_{d_2} = b \tag{11b}$$

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

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

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 $r_4^{(b)} := r_{\max} - r_{\text{cache,max}}^{(b)}$, where

$$r_{\max} := 2/3 \tag{12}$$

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

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

Channel encoding in block 1 (Figure 2):
The server first appends $n\big(r_{\max} - r_1\big)$, $n\big(r_{\max} - r_2\big)$, and $n\big(r_{\max} - r_3\big)$ 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 beginnig of subchunk e_1. Then it takes the componentwise XOR of the four resulting $nr_{\max} = 2n/3$-length packets $\bar{m}_a, \bar{m}_b, \bar{m}_{c_1}, \bar{m}_{e_1}$ to form

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

The encoder applies a standard blocklength-n Polar encoder [29] 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.

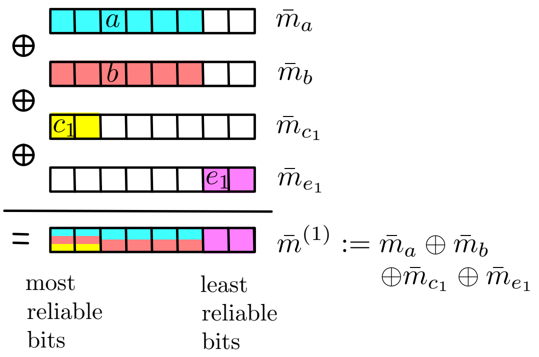

Fig. 2: Creation of "information bits" for the Polar encoder used in block 1.

Channel decoding in block 1 (Figures 3–5):

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

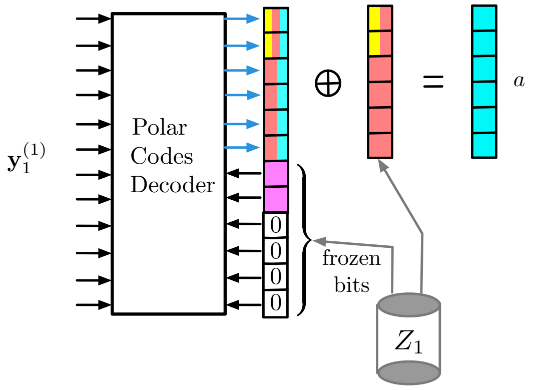

Fig. 3: Decoding procedure at the cache-aided User 1.

$$f_1 = e_1 \parallel [0 \cdots 0], \tag{15}$$

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 f_1 to the outputs $\mathbf{y}_1^{(1)}$, where it ensures again that these frozen bits are used in decreasing reliability order. If the Polar decoder succeeds, it produces the string

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

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

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

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

$$f_3 = a_2 \oplus b_2 \parallel a_3 \oplus b_3 \parallel e_1 \parallel [0 \cdots 0], \tag{18}$$

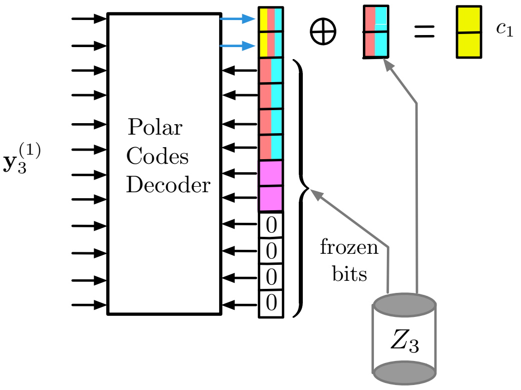

Fig. 4: Decoding procedure at the cache-aided User 3.

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

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

and user 3 can decode its desired subchunk c_1 by forming

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

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

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

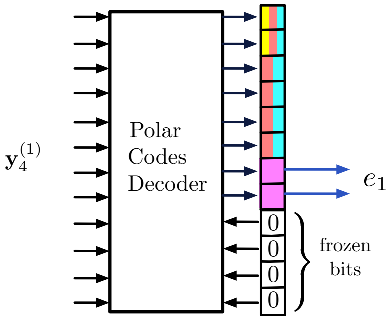

Fig. 5: Example of the decoding procedure at the cache-free User 4.

Channel encoding and decodings in block 2 (Figure 6):

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 $nr^{(2)}_{\text{cache,max}} = n/2$ zeros at the beginning of subchunk e_2. Then it takes the componentwise XOR of the three resulting $nr_{\max}$-length strings $\bar{m}_{\tilde{a}}, \bar{m}_{c_2}, \bar{m}_{e_2}$:

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

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).

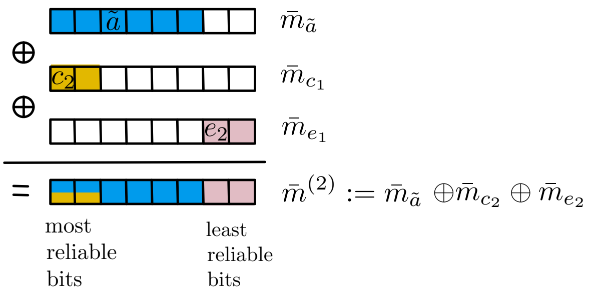

Fig. 6: Creation of "information bits" for the Polar encoder used in block 2.

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 (Figure 7):

The encoder directly forms the $nr_{\max}$-length bit string $\bar{m}^{(3)}$ by appending $\tilde{e}$ *after* c_3:

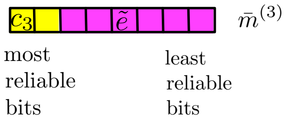

Fig. 7: Creation of "information bits" for the Polar encoder used in block 3.

$$\bar{m}^{(3)} := [c_3 \,||\, \tilde{e}]. \tag{23}$$

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

$$f_3 = \tilde{e} \,||\, [0 \cdots 0]. \tag{24}$$

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

$$e_1 = [\bar{m}^{(3)}]_{(nr^{(3)}_{\text{cache,max}}+1):nr_4}. \tag{25}$$

Channel encoding and decodings in block 4 (Figure 8):

The encoder directly forms the $nr_{\max}$-length bit string $\bar{m}^{(4)}$ by appending $\hat{e}$ *after* e_3:

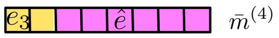

Fig. 8: Creation of "information bits" for the Polar encoder used in block 4.

$$\bar{m}^{(4)} := [e_3 \,||\, \hat{e}]. \tag{26}$$

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

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

IV. Channel Coding in a Single Block—The General Scheme

Assume that the data assignment step associates with a given 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

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

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

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

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 $\mathbf{y}_k$ 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 $\bar{m}_k$, and combines all these strings to

$$\bar{m}_{\text{cache}} = \bigoplus_{k \in \mathcal{G}_{\text{cache}}} \bar{m}_k. \tag{29}$$

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

$$\bar{m}_{\text{nocache}} = \underbrace{[0 \cdots 0]}_{nr_{\text{cache,max}} \text{ symbols}} \parallel m_{k_{\text{nocache},1}} \parallel \cdots \parallel m_{k_{\text{nocache},|\mathcal{G}_{\text{nocache}}|}}, \tag{30}$$

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

$$\bar{m} = \bar{m}_{\text{cache}} \oplus \bar{m}_{\text{nocache}}, \tag{31}$$

to a standard blocklength-n Polar encoder, for which it ensures that the Polar code assigns the n bits of $\bar{m}$ 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.

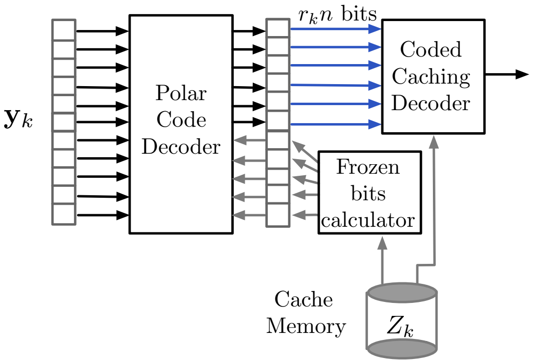

Fig. 9: Example of the delivery decoding for a single block at a cache-aided user i_ℓ.

Decoding at cache-aided user $k \in \mathcal{G}_{\text{cache}}$ (Figure 9):

User $k \in \mathcal{G}_{\text{cache}}$ retrieves the bit-strings $\{m_j \colon j \in \mathcal{G}_{\text{cache}} \backslash \{k\}\}$ and $\{m_j \colon j \in \mathcal{G}_{\text{nocache}}\}$ from its cache memory, forms the zero-padded or concatenated strings $\{\bar{m}_j \colon j \in \mathcal{G}_{\text{cache}} \backslash \{k\}\}$ and $\bar{m}_{\text{nocache}}$ as described above to compute

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

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

$$c_k := [\bar{m}_{\text{dec},k}]_{nr_k} \tag{33}$$

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

and applies the standard Polar decoding algorithm with the $n(1-r_k)$-length frozen-bit vector $f_k\|[0\cdots 0]$ 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:

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

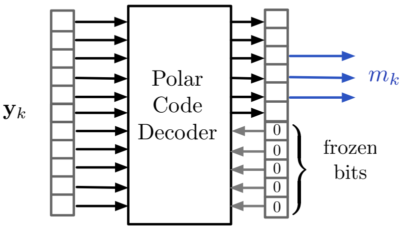

Fig. 10: Example of the delivery decoding for a single block at a user k_{nocache} without cache.

Decoding at cache-free user $k \in \mathcal{G}_{\text{cache}}$ (Figure 10): 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 $n(1-r_{\max})$ 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

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

V. Data Assignment Procedures

The data assignment procedure consists 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 a code of blocklength n is used for transmission over the channel.

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

Similar to [32]–[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

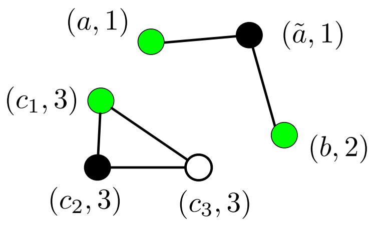

Fig. 11: Illustration of the subchunk conflict-graph with a minimum graph colouring for the example in Section III-B. This conflict graph/graph colouring is used in the more complex procedure.

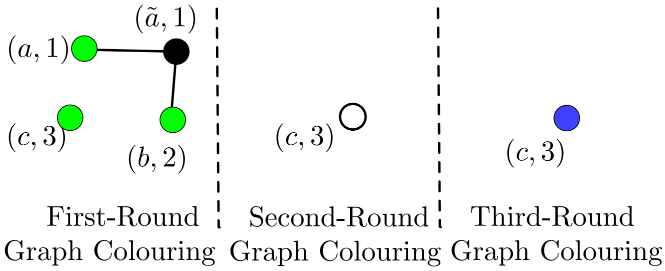
$(\tilde{a},1)$
$(a,1)$
$(c,3)$
$(b,2)$
$(c,3)$
$(c,3)$
First-Round Graph Colouring
Second-Round Graph Colouring
Third-Round Graph Colouring

Fig. 12: 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.

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 $\left\lceil \frac{F_{\text{chunk}}}{nr_k} \right\rceil$ 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 k indicates the intended user and ℓ the number of the subchunk within W'_{d_k}. Two vertices (ℓ, k) and (ℓ', k') are connected in the conflict graph *unless* the ℓ-th subchunk of W'_{d_k} is cached at user k' and the ℓ'-th subchunk of $W_{d'_k}$ is cached at 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.

The subchunk conflict graph of the example in Section III-B and a possible minimum colouring are shown in Figure 11, where for simplicity, subchunks are labelled with letters $a, \tilde{a}, b, c_1, c_2, c_3$ as introduced in Section III-B. 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 files 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 the following, we explain a simpler (generally less efficient) way to assign the subchunks to the various transmission blocks.

2) Simpler procedure: The idea is to create the conflict graph as in [32]. I.e., a node of the conflict graph is associated to an entire chunk and is labelled 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, and 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 (sub)chunk of size nr_k bits, and assigns all subchunks of the same colour to the same transmission block. It further eliminates all

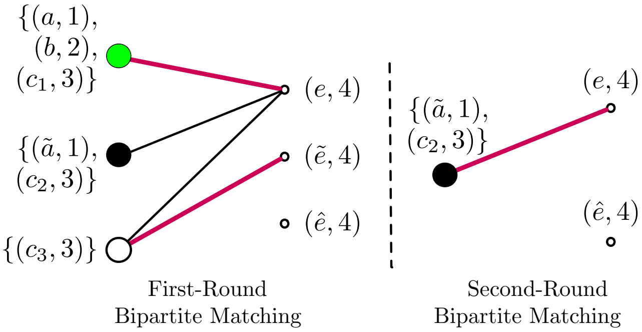

Fig. 13: Illustration of the iterated bipartite graph and a maximum bipartite matching for the example in Section III-B.

chunks from the graph, for which all subchunks have already been assigned and starts the next round.

Figure 12 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 $r_3 = 1/6$, 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 of 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. At first, the server constructs a bi-partite graph that on the left-hand side (LHS) contains a node for each group of subchunks created in the Section V-A, and on the right-hand side (RHS) 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 and a node on the RHS if, and only if, *all* users on the LHS have stored the chunk on the RHS in their cache memories. 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 $\{(l_{k_{\text{cache}}}, k_{\text{cache}}) \colon \mathcal{G}_{\text{cache}} \in \mathcal{K}_{\text{cache}}\}$ and on the RHS a single chunk $(l_{k_{\text{nocache}}}, k_{\text{nocache}})$, the server retrieves a subchunk of chunk (l, k_{nocache}) of size $nr_{k_{\text{nocache}}}$ bits (if possible) and it assigns this subchunk to the same transmission block as the group $\{(l_{k_{\text{cache}}}, k_{\text{cache}}) \colon \mathcal{G}_{\text{cache}} \in \mathcal{K}_{\text{cache}}\}$ was assigned in the first step. It then retrieves the freshly assigned bits from the chunk on the LHS, and retrieves the chunk all-together from the graph if it is empty. Moreover, if the full $nr_{k_{\text{nocache}}}$ bits could be assigned (i.e., the chunk on the RHS of the edge indeed contained so many bits), the server also removes the group of chunks on the LHS of the edge. Then, it starts the next-following round based on 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, each such node is assigned to a new transmission block.

Figure 13 shows the procedure for our example of Subsection 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 egde 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.

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: Subsection VI-A, explains the common setup for both the computer simulations and the testbed implementation; Subsection VI-B presents the simulation results; and Subsection VI-C the testbed measurements.

A. Practical Setup

1) Physical layer model: For data transmission, the employed modulation is QPSK-OFDM 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 $T = 80$ μs. All physical layer parameters are presented in Table I.

Bandwidth (MHz)	$B = 1$
Sampling Period (μs)	$T_s = 1$
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 (μs)	$T = 80$

TABLE I: Parameters of the physical layer.

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 $\lceil 1024/48 \rceil = 22$ 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 finally conclude that since each OFDM symbol spans 80 channel uses, a frame spans in total $80(22 + N_h)$ channel uses.

The data assignment and channel coding are done according to the procedures described in Sections III–V, 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 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:
$$\text{BER}_k := \frac{\#\text{wrongly-decoded bits of } W'_{d_k}}{\#\text{bits of } W'_{d_k}}. \tag{37}$$
- **Sum-Throughput**: The sum-throughput η 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:
$$\eta := \frac{\sum_k \#\text{successfully-decoded bits of } W'_{d_k}}{80 \cdot (22 + N_h) \cdot \#\text{transmitted codewords}}. \tag{38}$$

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:

- *No caching*: Users are cache-free. They thus are successively served through individual transmissions.
- *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 [17]*: XORed strings can be of different lengths. Chunks for cache-free users are sent in separate codewords.

B. Simulation Results

In our simulations we emulate a frequency-selective channel as described in Eq. (4). where the delays and path attenuations are given by [38]
$$a_{k,m}(t) = A_{k,m} \cdot \mathsf{SoS}_{k,m}(j), \quad t \in \big((j-1)\cdot T,\ j \cdot T\big], \tag{39}$$
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 $\mathsf{SoS}_{k,m}(j)$ which is obtained by
$$\mathsf{SoS}_{k,m}(j) = \mathsf{SoS}^{(\text{Re})}_{k,m}(j) + i. \cdot \mathsf{SoS}^{(\text{Im})}_{k,m}(j) \tag{40}$$
with
$$\mathsf{SoS}^{(\text{Re})}_{k,m}(j) = \frac{1}{\sqrt{L_s}} \sum_{\ell=1}^{L_s} \cos\left(2\pi f_d jT \cos\left(\frac{2\pi\ell - \pi + \theta_{k,m}(j)}{4L_s} + \phi_{k,m,\ell}\right)\right) \tag{41a}$$
and
$$\mathsf{SoS}^{(\text{Im})}_{k,m}(j) = \frac{1}{\sqrt{L_s}} \sum_{\ell=1}^{L_s} \cos\left(2\pi f_d jT \sin\left(\frac{2\pi\ell - \pi + \theta_{k,m}(j)}{4L_s} + \psi_{k,m,\ell}\right)\right). \tag{41b}$$

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

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

where $u_{k,m}(j)$ is uniformly distributed over $[0, \pi)$ and independent over OFDM symbols $j = 1, 2, \ldots$, paths $m = 1, \ldots, L$, 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 δ is modified when $\theta_{k,m}$ reaches the value π or $-\pi$. The terms $\phi_{k,m,\ell}$ and $\psi_{k,m,\ell}$ are iid 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.

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

TABLE II: Parameters of the channel model.

According to [38], the correlation in time within one path is given by $\mathrm{E}[\mathsf{SoS}_{k,m}(j)\mathsf{SoS}_{k,m}(j+j')^*] = J_0(2\pi f_d j' T)$ 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 get $T_c = 9,549\mu$s which is equivalent 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 $d_1, \ldots, d_K$ 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. The caching parameter is set to $\alpha = 0.1$, and as a result each user with a cache memory stores approximately 10% of all the chunks of each file W_k. 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.

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 III: Configuration 1 with a single cache-free user.

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

TABLE IV: Configuration 2 with two cache-free users.

Let us first consider a memoryless Gaussian channel. In Figure 14, 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 behaviours can be observed for Configuration 2. Figures 15 and 16 show the sum-throughput η in function of the SNR at the weakest user 1 for the various schemes, under Configurations 1 and 2, respectively. We remark a 10% increase in throughput of the proposed scheme compared to a scheme without coded 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 Figure 17, 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 various schemes behave analogously, and the same conclusion applies as for the Gaussian channel. Figures 18 and 19 show the sum-throughput η 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 Figure 20, we plot the sum-throughput η over the specified frequency-selective channel in function of the Zipf parameter s for an 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.

C. Testbed Measurements

The proposed scheme of Sections III–V (with the simplified conflict graph construction in Section V) was also tested on the FIT/CorteXlab testbed (http://www.cortexlab.fr/), which is composed of SISO or MIMO software defined radio (SDR) nodes [36]. The testbed is hosted in a 180 m^2 shielded room, which is partly covered with electromagnetic wave absorbing material, and its layout is shown in Fig. 21. 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.

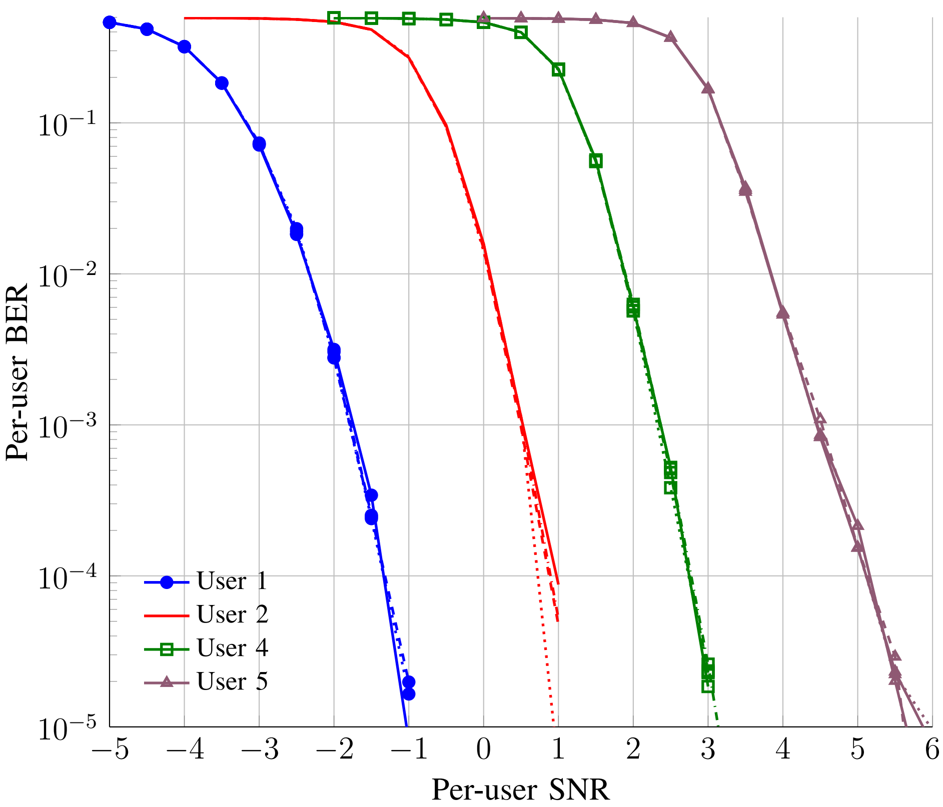

Fig. 14: Per-user BERs versus the per-user SNRs over a Gaussian channel under Configuration 1. Solid line: *proposed scheme*; dashed line: *generalized coded caching*; dashdotted line: *coded caching*; dotted line: *no coded caching*.

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 22. 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. 23, where compared to the header in [37] we added a field indicating whether a cache-free user is served (in this practical implementation 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 a corresponding subchunk ID number. The header is variable length because the number of served users n varies over the codewords and because we chose a variable-length description for the subchunk IDs. (Please see [39] for more details on the practical implementation of the header.)

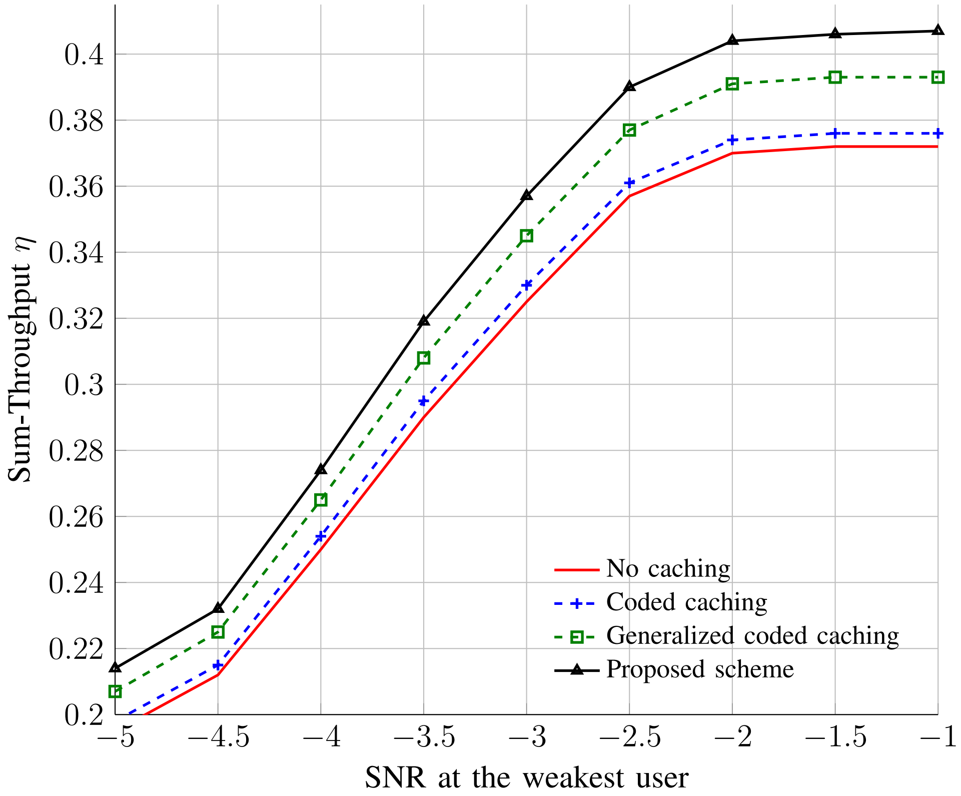

Fig. 15: Sum-throughputs η versus SNR at the weakest user 1 over a Gaussian channel under Configuration 1.

In our testbed implementation with $K = 5$ users, 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 [40], and channel estimation using the Least Square method [41]. 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, 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 21: 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 $d_1, \ldots, d_k$, where the parameter of the Zipf distribution is set to $s = 1$.

In Figure 24, we display the sum-throughput η 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 15 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

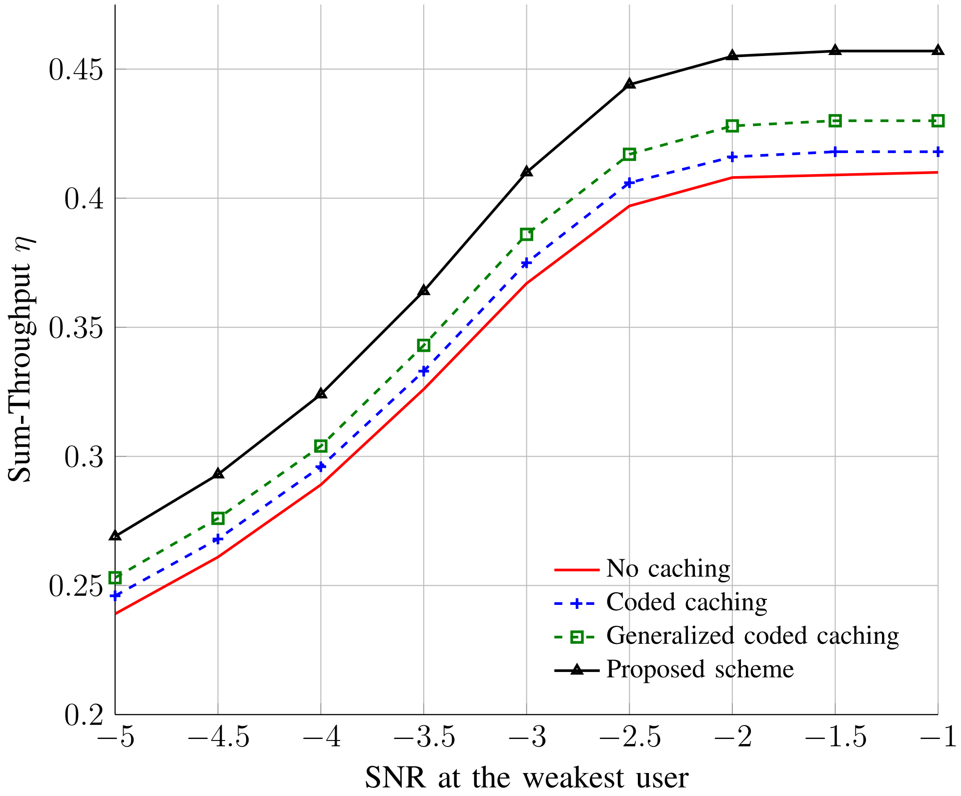

Fig. 16: Sum-throughputs η versus SNR at the weakest user 1 over a Gaussian channel under Configuration 2.

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

TABLE V: Users configuration in the testbed setup.

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

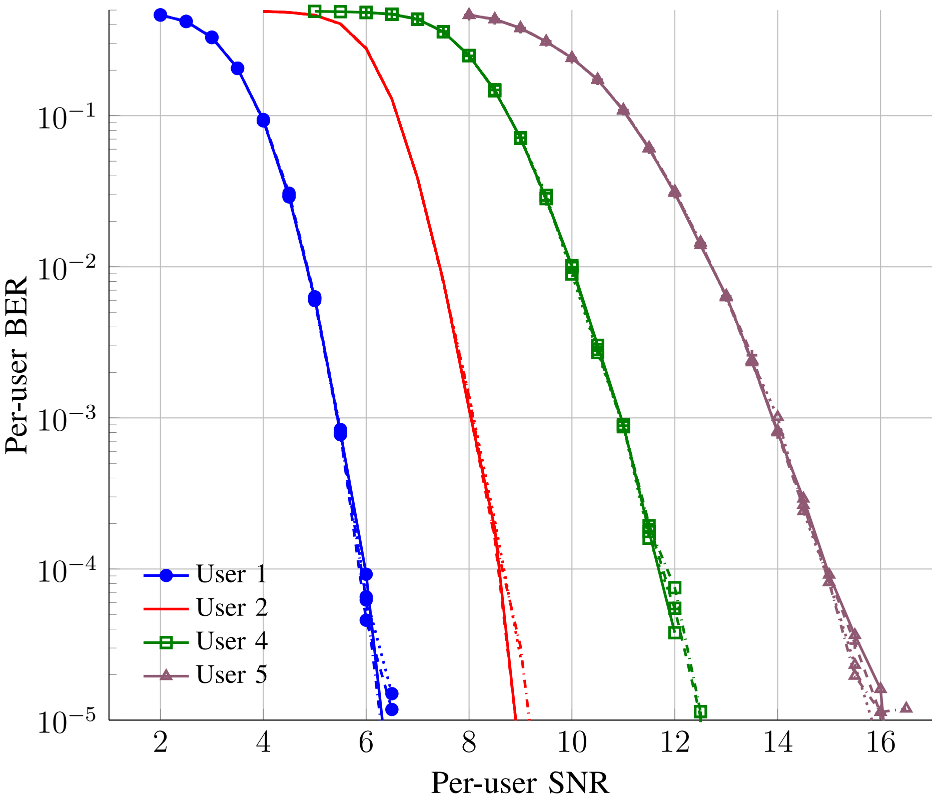

Fig. 17: Per-user BERs versus per-user SNRs over the specified frequency-selective channel under Configuration 1. Solid lines: *proposed scheme*; dashed lines: *generalized coded caching*; dashdotted lines: *coded caching*; dotted lines: *no coded caching*.

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 Transactions on Information Theory*, vol. 60, no. 5, pp. 2856–2867, 2014.

[2] ——, "Decentralized coded caching attains order-optimal memory-rate tradeoff," *IEEE/ACM Transactions on Networking*, vol. 23, no. 4, pp. 1029–1040, 2015.

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

[4] C. Tian, "A note on the fundamental limits of coded caching," *CoRR*, vol. abs/1503.00010, 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 *2016 IEEE International Symposium on Information Theory*, July 2016, pp. 135–139.

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

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

[8] M. M. Amiri, Q. Yang, and D. Gündüz, "Coded caching for a large number of users," in *2016 IEEE Information Theory Workshop (ITW)*, Sept 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 Transactions on Information Theory*, vol. 64, no. 2, pp. 1281–1296, Feb 2018.

[10] J. Gómez-Vilardebó, "Fundamental limits of caching: Improved bounds with coded prefetching," *IEEE Transactions on Communications*, vol. 66, 2018.

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

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

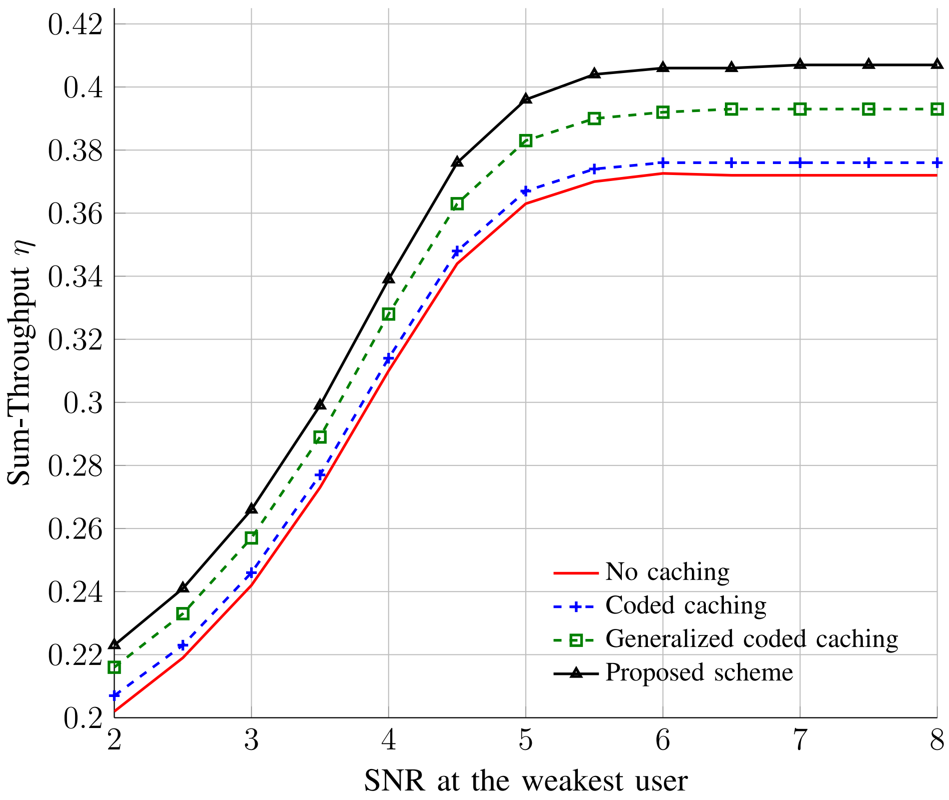

Fig. 18: Sum-throughputs η versus SNR at the weakest user 1 over the specified frequency-selective channel under Configuration 1.

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

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

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

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

[17] S. Saeedi Bidokhti, M. Wigger, and A. Yener, “Benefits of cache assignment on degraded broadcast channels,” *IEEE Transactions on Information Theory*, vol. 65, no. 11, pp. 6999–7019, 2019.

[18] S. S. Bidokhti, M. Wigger, A. Yener, and A. El Gamal, “State-adaptive coded caching for symmetric broadcast channels,” in *2017 Asilomar Conference on Signals, Systems, and Computers*, 2017, pp. 646–650.

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

[20] A. S. Cacciapuoti, M. Caleffi, M. Ji, J. Llorca, and A. Tulino, “Speeding up future video distribution via channel-aware caching-aided coded multicast,” *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 8, pp. 2207–2218, May 2016.

[21] M. Mohammadi Amiri and D. Gündüz, “Cache-aided content delivery over erasure broadcast channels,” *IEEE Transactions on Communications*, vol. 66, no. 1, pp. 370–381, 2018.

[22] J. Hachem, N. Karamchandani, and S. Diggavi, “Content caching and delivery over heterogeneous wireless networks,” in *2015 IEEE Conference on Computer Communications (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 Transactions on Information 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 *2016 International Symposium on Turbo-codes and Iterative Information Processing (ISTC)*, 2016, pp. 370–374.

[25] I. Bergel and S. Mohajer, “Cache-aided communications with multiple antennas at finite snr,” *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 8, pp. 1682–1691, 2018.

[26] X. You, Y. Wu, and J. Chen, “Cache-aided broadcast channels with user cooperation,” in *2019 Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2019, pp. 544–551.

[27] J. Chen, H. Yin, X. You, Y. Geng, and Y. Wu, “Centralized coded caching with user cooperation,” in *2019 IEEE Information Theory Workshop (ITW)*, 2019, pp. 1–5.

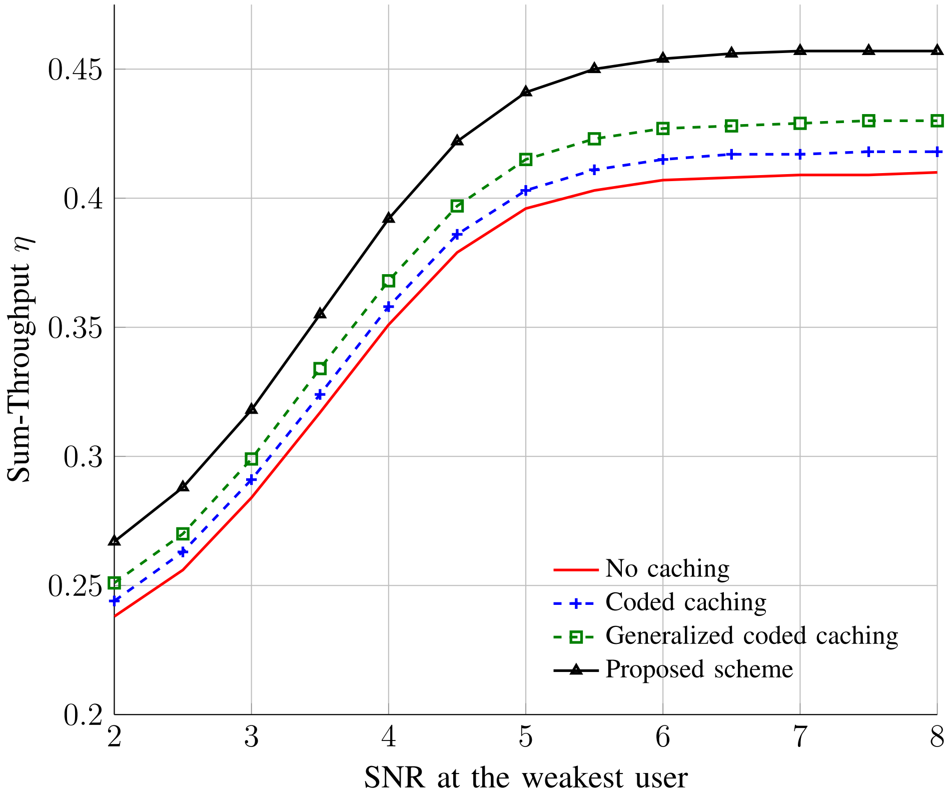

Fig. 19: Sum-throughputs η versus SNR at the weakest user 1 over the specified frequency-selective channel under Configuration 2.

[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 *2020 IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 1687–1692.

[29] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[30] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2213–2226, 2015.

[31] Y. Birk and T. Kol, "Informed-source coding-on-demand (iscod) over broadcast channels," in *1998 IEEE Conference on Computer Communications (INFOCOM)*, 1998, pp. 1257–1264.

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

[33] G. Vettigli, M. Ji, A. M. Tulino, J. Llorca, and P. Festa, "An efficient coded multicasting scheme preserving the multiplicative caching gain," in *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 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 *2014 International Symposium on Wireless Communications Systems (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 *2015 IEEE International Conference on Communications (ICC)*, 2015, pp. 3801–3806.

[36] L. S. Cardoso, A. Massouri, B. Guillon, P. Ferrand, F. Hutu, G. Villemaud, T. Risset, and J.-M. Gorce, "CorteXlab: A Facility for Testing Cognitive Radio Networks in a Reproducible Environment," in *in Proc. 9th International Conference on Cognitive Radio Oriented Wireless Networks and Communications (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 Communications Magazine*, vol. 55, no. 2, pp. 106–113, 2017.

[38] A. Alimohammad, S. F. Fard, B. F. Cockburn, and C. Schlegel, "Compact Rayleigh and Rician fading simulator based on random walk processes," *IET Communications*, vol. 3, no. 8, pp. 1333–1342, Jul. 2009.

[39] Y. Fadlallah, O. Oubejja, and S. Kamel, "Cache-aided polar piggyback coding implementation for testbed measurements," Available at `https://perso.telecom-paris.fr/wigger/Cortex_code/`.

[40] T. M. Schmidl and D. C. Cox, "Robust Frequency and Timing Synchronization for OFDM," *IEEE Transactions on Communications*, vol. 45, no. 12, pp. 1613–1621, Dec 1997.

[41] J. van de Beek, O. Edfors, M. Sandell, S. Wilson, and P. Borjesson, "On channel estimation in OFDM systems," in *1995 IEEE Vehicular Technology Conference (VTC)*, 1995, pp. 815–819.

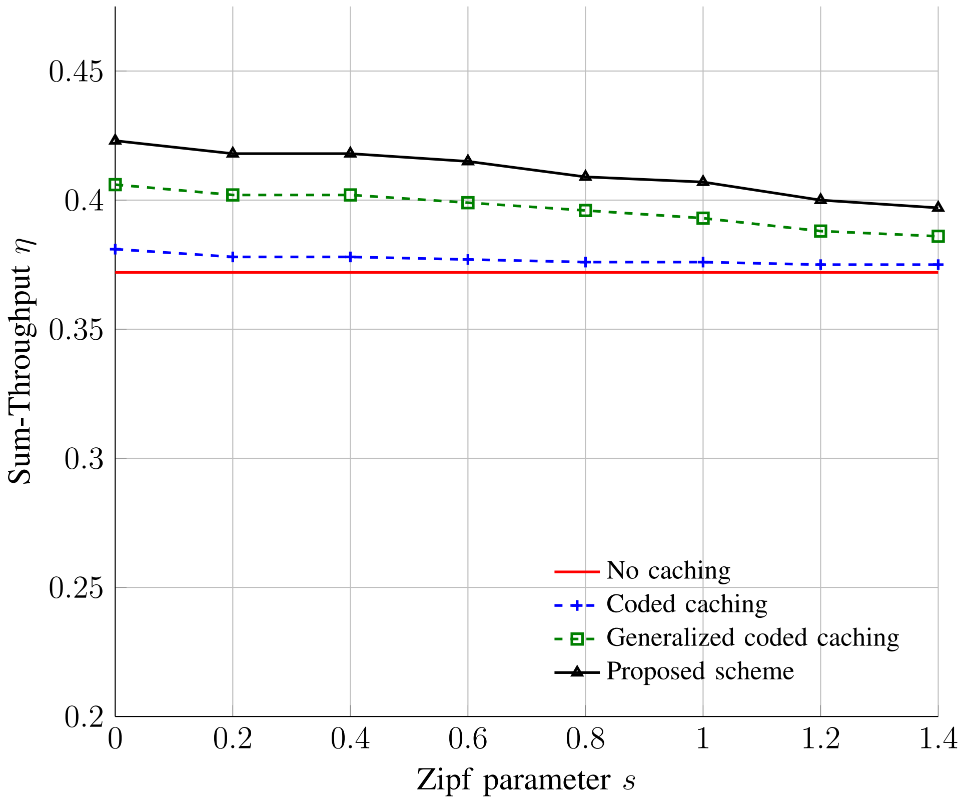

Fig. 20: Sum-throughputs η versus s for SNR at the weakest user of 8dB over the specified frequency-selective channel under Configuration 1.

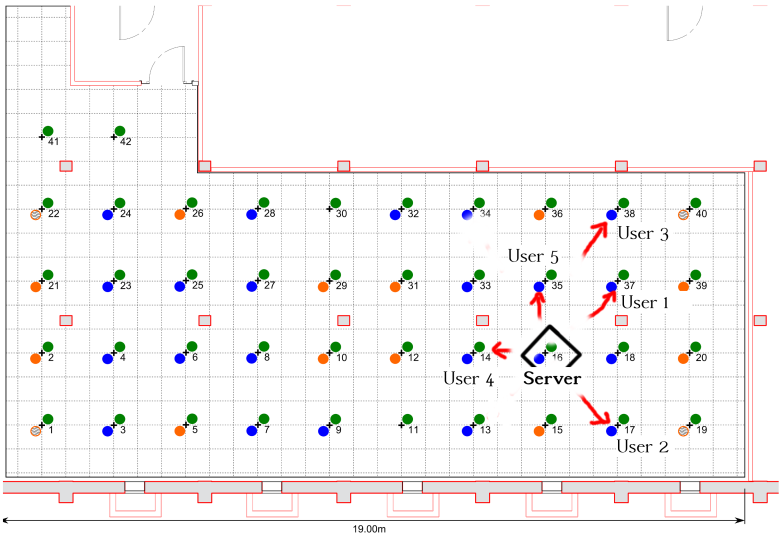

Fig. 21: Testbed room.

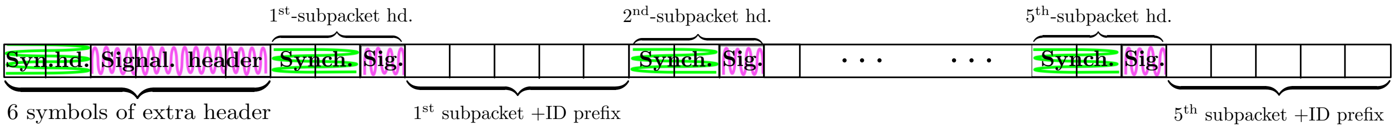

Fig. 22: Illustration of the frame structure related to a single codeword.

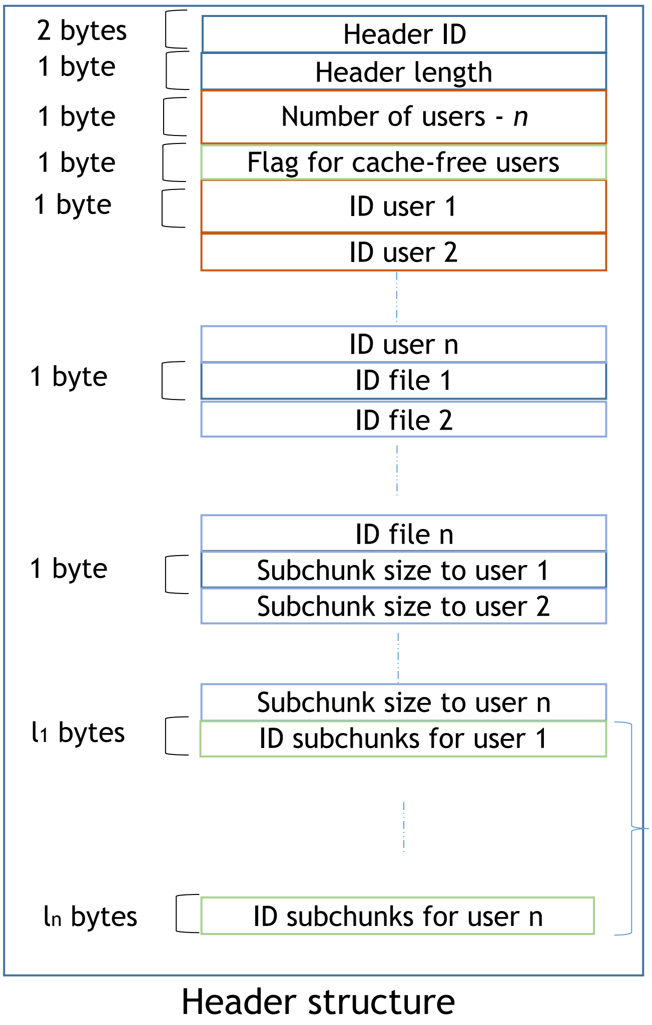

Fig. 23: Illustration of the entries of the extra signaling header.

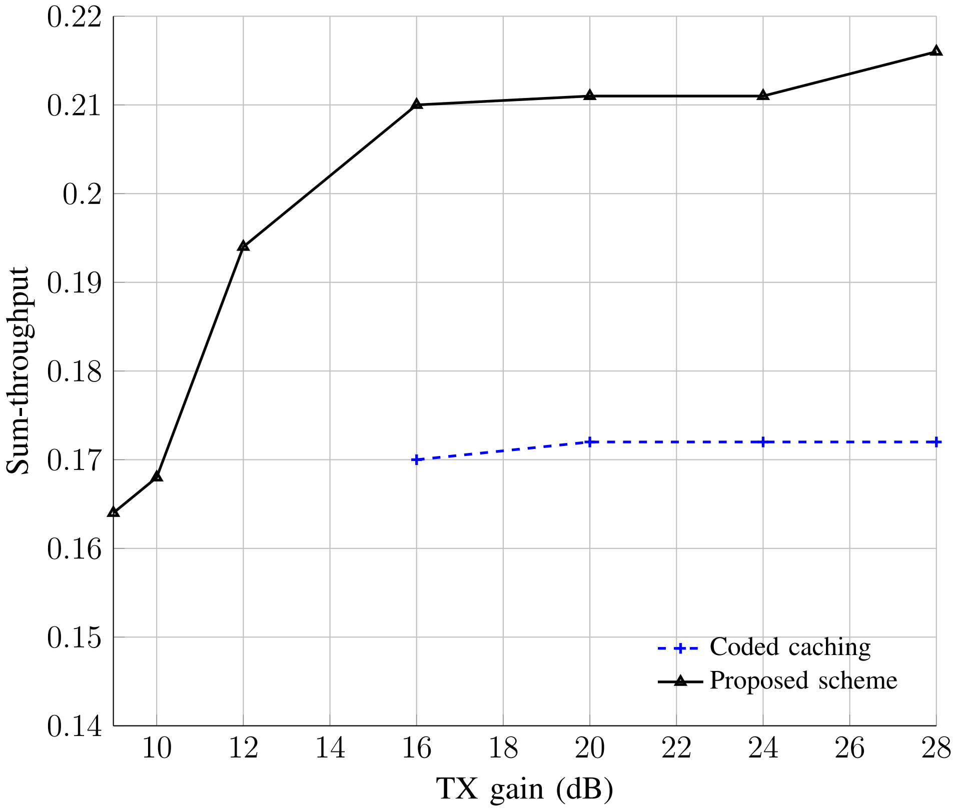

Fig. 24: Sum-throughputs η versus TX gain for our proposed scheme and for the coded caching scheme.