



CHALMERS

Chalmers Publication Library

Using Short Synchronous WOM Codes to Make WOM Codes Decodable

This document has been downloaded from Chalmers Publication Library (CPL). It is the author's version of a work that was accepted for publication in:

IEEE Transactions on Communications (ISSN: 0090-6778)

Citation for the published paper:

Bitouze, N. ; Graell i Amat, A. ; Rosnes, E. (2014) "Using Short Synchronous WOM Codes to Make WOM Codes Decodable". IEEE Transactions on Communications, vol. 62(7), pp. 2156-2169.

<http://dx.doi.org/10.1109/TCOMM.2014.2323308>

Downloaded from: <http://publications.lib.chalmers.se/publication/200620>

Notice: Changes introduced as a result of publishing processes such as copy-editing and formatting may not be reflected in this document. For a definitive version of this work, please refer to the published source. Please note that access to the published version might require a subscription.

Chalmers Publication Library (CPL) offers the possibility of retrieving research publications produced at Chalmers University of Technology. It covers all types of publications: articles, dissertations, licentiate theses, masters theses, conference papers, reports etc. Since 2006 it is the official tool for Chalmers official publication statistics. To ensure that Chalmers research results are disseminated as widely as possible, an Open Access Policy has been adopted. The CPL service is administrated and maintained by Chalmers Library.

(article starts on next page)

Using Short Synchronous WOM Codes to Make WOM Codes Decodable

Nicolas Bitouzé, Alexandre Graell i Amat, *Senior Member, IEEE*, and Eirik Rosnes, *Senior Member, IEEE*

Abstract—In the framework of write-once memory (WOM) codes, it is important to distinguish between codes that can be decoded directly and those that require the decoder to know the current generation so as to successfully decode the state of the memory. A widely used approach to constructing WOM codes is to design first nondecodable codes that approach the boundaries of the capacity region, and then make them decodable by appending additional cells that store the current generation, at an expense of rate loss. In this paper, we propose an alternative method to make nondecodable WOM codes decodable by appending cells that also store some additional data. The key idea is to append to the original (nondecodable) code a short synchronous WOM code and write generations of the original code and of the synchronous code simultaneously. We consider both the binary and the nonbinary case. Furthermore, we propose a construction of synchronous WOM codes, which are then used to make nondecodable codes decodable. For short-to-moderate block lengths, the proposed method significantly reduces the rate loss as compared to the standard method.

Index Terms—Coding theory, decodable codes, flash memories, synchronous write-once memory (WOM) codes.

I. INTRODUCTION AND DEFINITIONS

The write-once memory (WOM) model was introduced in [1] to study storage devices consisting of q -ary ($q \geq 2$) memory cells whose values cannot be decreased. It was originally introduced to model the behavior of optical disks and study coding schemes that would allow one to write data several times on a disk even though each bit can only be written once. By allowing data from a previous write to be “forgotten” when a new write occurs, one can show that the total amount of information that can be stored on such a disk is greater if several small pieces of information are stored and forgotten one after the other than if the whole disk is written at once. The model is now mainly studied because of its similarity with flash memories, on which the value of a cell can be decreased, but at an extremely high cost. Since the original paper by Rivest and Shamir [1], several other works on this topic have appeared, both in terms of code constructions,

capacity, and error-correction. See, for instance, [2–13] and references therein. Recently, lattice-based constructions have been proposed. For instance, in [14, 15] lattice-based t -write codes for multilevel cells were presented. For applications to flash memories, see [16–18].

The fundamental problem in the WOM model is, considering an array of n empty q -ary cells, to know how much information one can store using exactly t writes (also called *generations*). The coding schemes that are used to fulfill this goal are called *t -write WOM codes*. The following definition is taken from [11].

Definition 1: An $[n, t : M_1, \dots, M_t]_q$ t -write q -ary WOM code C is a coding scheme for n q -ary WOM cells, which consists of t pairs of encoding and decoding maps \mathcal{E}_i and \mathcal{D}_i ($1 \leq i \leq t$) such that

- 1) $\mathcal{E}_1 : \{1, \dots, M_1\} \rightarrow \{0, \dots, q-1\}^n$.
- 2) For $2 \leq i \leq t$:
 - $\mathcal{E}_i : \{1, \dots, M_i\} \times \text{Im}(\mathcal{E}_{i-1}) \rightarrow \{0, \dots, q-1\}^n$,
 - $\forall (m, \mathbf{b}) \in \{1, \dots, M_i\} \times \text{Im}(\mathcal{E}_{i-1})$,
 $\forall j \in \{1, \dots, n\}, (\mathcal{E}_i(m, \mathbf{b}))_j \geq (\mathbf{b})_j$.
- 3) For $1 \leq i \leq t$, $\mathcal{D}_i : \{0, \dots, q-1\}^n \rightarrow \{1, \dots, M_i\}$, and
 - $\forall m \in \{1, \dots, M_1\}, \mathcal{D}_1(\mathcal{E}_1(m)) = m$,
 - for $2 \leq i \leq t, \forall (m, \mathbf{b}) \in \{1, \dots, M_i\} \times \text{Im}(\mathcal{E}_{i-1})$,
 $\mathcal{D}_i(\mathcal{E}_i(m, \mathbf{b})) = m$.

For simplicity, in the remainder of the paper, we will refer to WOM codes simply as codes. The rate of the above code, referred to as the WOM-rate, or sometimes just as the rate of the code, is defined as follows [11].

Definition 2: The rate of generation $i \in \{1, \dots, t\}$ of an $[n, t : M_1, \dots, M_t]_q$ q -ary code C is

$$R_i(C) \triangleq \frac{\log_2 M_i}{n}$$

and the WOM-rate of C is defined as

$$R(C) \triangleq \sum_{i=1}^t R_i(C) = \frac{\sum_{i=1}^t \log_2 M_i}{n}.$$

The fundamental problem of the WOM model is therefore to find a code of maximum WOM-rate given t and q , and sometimes n .

For some codes, the state of the cells is enough to determine the current generation (i.e., how many times the memory has been written). However, some codes have a structure such that the same state of the memory can appear at different generations. This is not a problem if the same state of the memory at different generations corresponds to the same message, but when it is not the case, the decoder has to

A. Graell i Amat was supported by the Swedish Research Council under Grant #2011-5961. E. Rosnes was supported by Simula@UiB. The material in this paper was presented in part at the 2012 IEEE International Symposium on Information Theory, Cambridge, MA, July 2012.

N. Bitouzé was with the Department of Electronics, Institut Télécom-Télécom Bretagne, CS 83818 - 29238 Brest Cedex 3, France. He is now with the Department of Electrical Engineering, University of California, Los Angeles (UCLA), Los Angeles, CA 90095-1594. E-mail: bitouze@ucla.edu.

A. Graell i Amat is with the Department of Signals and Systems, Chalmers University of Technology, Gothenburg, Sweden. E-mail: alexandre.graell@chalmers.se.

E. Rosnes was with Ceragon Networks AS, Kokstadveien 23, N-5257 Kokstad, Norway. He is now with the Selmer Center, Department of Informatics, University of Bergen, N-5020 Bergen, Norway, and the Simula Research Lab. E-mail: eirik@ii.uib.no.

be given the knowledge of the current generation in order to successfully decode the memory. We say that a code is *decodable* if for any state of the cells \mathbf{b} and any i_1 and i_2 with $\mathbf{b} \in \text{Im}(\mathcal{E}_{i_1}) \cap \text{Im}(\mathcal{E}_{i_2})$, $\mathcal{D}_{i_1}(\mathbf{b}) = \mathcal{D}_{i_2}(\mathbf{b})$. A code that does not satisfy this property is called *nondecodable*. A stronger property is given in [1]: a code is called *synchronous*¹ if the current state of the memory provides enough information to know the current generation, i.e., the sets $\text{Im}(\mathcal{E}_i)$ are disjoint for $1 \leq i \leq t$. Synchronous codes are decodable. However, the reverse does not always hold. The work in [1] also considers a way to guarantee synchronisness: *laminar* codes are codes such that the *weight* of the cells, defined as the L_1 -norm of the q -ary cell vector, is an injective function of the generation, i.e., for $\mathbf{b}_1 \in \text{Im}(\mathcal{E}_{i_1})$ and $\mathbf{b}_2 \in \text{Im}(\mathcal{E}_{i_2})$, $w(\mathbf{b}_1) = w(\mathbf{b}_2) \Rightarrow i_1 = i_2$. In the binary case, the weight reduces to the standard Hamming weight. The authors of [1] give a construction of laminar codes for $n = t$ being a power of two, with WOM-rate $\log_2(t)/2$. However, synchronous codes have not been extensively studied in the literature. Note that nonsynchronous codes can still be directly decoded if, when the decoder cannot determine the current generation, the choice of \mathcal{D}_i has no impact on the decoded symbol. In Section II, we give examples of laminar, synchronous (but nonlaminar), and decodable (but nonsynchronous) codes.

A nondecodable $[n_{\text{nd}}, t_{\text{nd}} : M_1, \dots, M_{t_{\text{nd}}}]_2$ binary code C can be made decodable (and even synchronous) by simply concatenating k instances of C with a block of $t_{\text{nd}} - 1$ cells that store the current generation (by being filled one by one at each write, starting at the second generation). The resulting code is a synchronous code with parameters $[kn_{\text{nd}} + t_{\text{nd}} - 1, t_{\text{nd}} : M_1^k, \dots, M_{t_{\text{nd}}}^k]_2$. As k goes to infinity, the WOM-rate of this code approaches the WOM-rate of the original code, $R(C)$.

Most of the state-of-the-art high-rate codes are not *directly* decodable. Indeed, a common approach in the literature is to design (nondecodable) codes that approach the boundaries of the capacity region (see, e.g., [11, 12]), and then make them decodable using the method above. However, for short-to-moderate block lengths, making a nondecodable code decodable by appending $t_{\text{nd}} - 1$ cells containing no data can significantly degrade its WOM-rate. For instance, consider $n = 6$ and $t = 4$, and assume that we do not know a decodable code of length 6. In this case, we could select a nondecodable 4-write code of length 3, and append 3 cells to store the current generation. The resulting WOM-rate is half the original one, as the additional cells only carry information about the current generation.

In this paper, we propose a different approach to make a nondecodable t_{nd} -write code C decodable. Our main focus is on binary codes, but we also extend our results to $q > 2$. The key idea is to append (for a t_{nd} -write nondecodable binary code of length n_{nd}) $t_{\text{nd}} - 1$ additional cells which store not only the current generation but also new data, by using a t_{nd} -write synchronous code with length $t_{\text{nd}} - 1$, and writing generations of C and of the synchronous code simultaneously. Since synchronous codes are at the basis of the proposed

method, we consider first the construction of synchronous codes. Our main focus is on laminar codes. The construction of synchronous (laminar) codes was already addressed in [1]. However, [1] only considered the case where $n = t$ and t is a power of 2. Here, we construct small laminar codes for both $n = t$ and $n > t$, and propose a construction for synchronous codes of higher values of t . Lifting the constraint $n = t$ allows to achieve higher WOM-rates. The obtained codes are then used to make nondecodable codes decodable. Whereas the main focus of this paper is on *unrestricted-rate* codes [12], i.e., we allow the individual writes to use a different number of inputs, we also extend our construction to *fixed-rate* codes, i.e., codes for which all writes store the same number of messages.

The remainder of this paper is organized as follows. In Section II, we introduce the main idea to turn nondecodable codes into decodable ones, and provide some examples. In Section III, we consider a simple family of laminar codes with $n = t$, as well as very short codes from this family. We also give bounds on the sizes of their generations, and construct better laminar codes with $n > t$ by local manipulations of the codes with $n = t$. In Section IV, we propose a construction of synchronous codes with good properties to reach higher values of t by concatenating instances of a synchronous code using a second synchronous code to decide, at each generation, which of the instances of the first code are going to be modified. In Section V, we study the case of fixed-rate codes, and we extend our results on the binary case to nonbinary scenarios in Section VI. Finally, in Section VII, we compare our method of making nondecodable codes decodable with the method that only adds cells containing no data. Some conclusions are drawn in Section VIII.

II. MAIN IDEA AND EXAMPLES

Let C be a nondecodable code with parameters $[n_{\text{nd}}, t_{\text{nd}} : M_1, \dots, M_{t_{\text{nd}}}]_2$, and WOM-rate R_{nd} . The standard approach to turn C into a decodable code is to append $t_{\text{nd}} - 1$ cells that store the current generation, thus obtaining a code of length $n = n_{\text{nd}} + t_{\text{nd}} - 1$. This incurs a rate loss

$$\gamma_{\text{basic}} = \frac{R_{\text{nd}} - R_{\text{nd}} \frac{t_{\text{nd}}}{n}}{R_{\text{nd}}} = \frac{t_{\text{nd}} - 1}{n}. \quad (1)$$

The main idea in this paper is very simple: instead of adding cells that do not contain information, we append to the original code cells that also store actual data. This is achieved by appending to C a t_{nd} -write synchronous code of length $n_{\text{sync}} = t_{\text{nd}} - 1$, and writing generations of C and of the synchronous code simultaneously. Appending a synchronous code to C results in an overall decodable (and also synchronous) code (the synchronisness of the appended code guarantees that by observing the $t_{\text{nd}} - 1$ new cells, the decoder can always determine the current generation, and use this knowledge to decode the overall code), while allowing to store extra data.

Let $R_{\text{sync}} > 0$ be the WOM-rate of the synchronous code that we append to the nondecodable code. The rate loss

¹Our concept of a synchronous code is equivalent to the concept of an *almost-synchronous* code from [1].

introduced by this method, denoted by γ_{sync} , is

$$\begin{aligned} \gamma_{\text{sync}} &= \frac{R_{\text{nd}} - (R_{\text{nd}}(n - n_{\text{sync}}) + R_{\text{sync}}n_{\text{sync}})/n}{R_{\text{nd}}} \\ &= \frac{n_{\text{sync}}}{n} \left(1 - \frac{R_{\text{sync}}}{R_{\text{nd}}}\right) \end{aligned} \quad (2)$$

which is smaller than γ_{basic} , since we can choose $n_{\text{sync}} = t_{\text{nd}} - 1$ (or slightly above). Note that γ_{sync} is decreasing with R_{sync} when n , n_{sync} , and $R_{\text{nd}} > 0$ are fixed. The main ingredient of the proposed technique is therefore a t_{nd} -write synchronous code of length $t_{\text{nd}} - 1$. To increase R_{sync} one may also consider synchronous codes with n_{sync} slightly larger than $t_{\text{nd}} - 1$ (the length of the resulting overall code would be slightly larger than that of the code obtained applying the standard method. However, the increase in length is compensated by a larger WOM-rate R_{sync}).

The following sections are devoted to the construction of t -write synchronous codes of length $n = t - 1$ (or slightly larger) to be used to make a nondecodable code decodable as explained above. Ideally, we would like to design synchronous codes that maximize the WOM-rate. However, this is overly complex. Instead we first construct small laminar codes, and then propose a construction method to construct synchronous codes for larger values of t by concatenating smaller codes. The use of laminar codes makes the computer search more tractable.

The construction method in Section IV requires component codes which do not contain the all-zero codeword. Therefore, in Section III we construct small laminar codes which do not contain the all-zero codeword. Note that for codes that do not contain the all-zero codeword, the number of writes is limited by the code length, $t \leq n$. Thus, our approach is to construct $(t - 1)$ -write synchronous codes with length $n = t - 1$ from component codes which do not contain the all-zero codeword, and then obtain a t -write synchronous code with length $n = t - 1$ by simply adding a generation that only contains the all-zero codeword.

To ease the understanding of the paper, in the following we clarify this and the concepts of synchronous, laminar, and decodable (but not synchronous) codes with some examples. For later use, if an $[n, t : M_1, \dots, M_t]_q$ code is synchronous, we will frequently use the superscript “sync”, $[n, t : M_1, \dots, M_t]_q^{\text{sync}}$. Also, in the binary case, the cells that can be written from 0 to 1 but not from 1 to 0 are called *wits* [1].

Example 1: An example of a binary $[4, 4 : 4, 2, 2, 1]_2$ laminar code is depicted in Fig. 1 by a state diagram describing all four writes. The four-bit vector in each state is the memory-state. The different types of edges (solid, dashed, dotted, and dash-dotted) correspond to different input data bits. As can be seen from the figure, the weight of the cells uniquely identifies the generation.

Example 2: An example of a quaternary $[2, 4 : 2, 2, 3, 3]_4$ synchronous (but nonlaminar) code is depicted in Fig. 2 by a state diagram describing all four writes. The two-symbol vector in each state is the memory-state. The different types of edges (solid, dashed, and dotted) correspond to different input data symbols. As can be seen from the figure, the cells of the memory can not be in the same state at different generations,

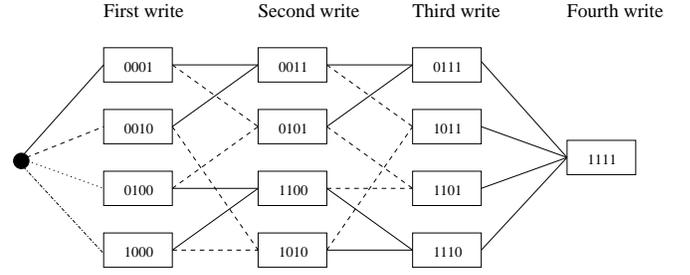


Fig. 1. A binary $[4, 4 : 4, 2, 2, 1]_2$ laminar four-write code. The different types of edges (solid, dashed, dotted, and dash-dotted) correspond to different input data bits.

which implies that the code is synchronous, but the weight (or L_1 -norm) of the cell state (22) of the third generation and the weight of the cell state (31) (or (13)) of the fourth generation are the same. Thus, the weight is *not* an injective function of the generation, and the code is not laminar.

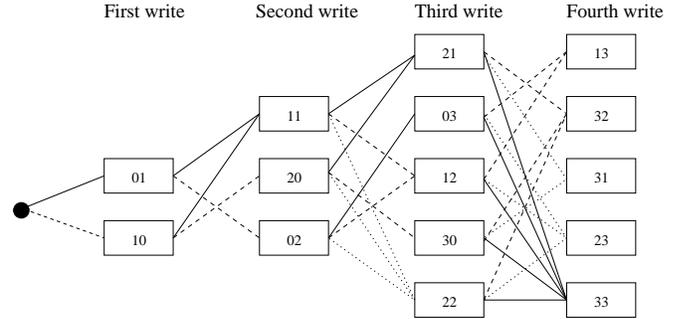


Fig. 2. A quaternary $[2, 4 : 2, 2, 3, 3]_4$ synchronous (but nonlaminar) four-write code. The different types of edges (solid, dashed, and dotted) correspond to different input data symbols.

Example 3: A simple example of a decodable (but nonsynchronous) binary code, taken from [1], that enables two bits to be written into three memory cells twice, is given Table I, which describes the encoding and decoding rules for the code. The code is nonsynchronous, since for the second write, if the information to be encoded does not change, then the state of the memory does not change either. Thus, the current state of the memory does not provide enough information to tell the current generation.

TABLE I
A BINARY $[3, 2 : 4, 4]_2$ DECODABLE (BUT NONSYNCHRONOUS) CODE.

Data bits	First write	Second write (if data changes)
00	000	111
10	100	011
01	010	101
11	001	110

Example 4: By adding a generation containing the all-zero codeword prior to all other generations of the $[4, 4 : 4, 2, 2, 1]_2$ code from Example 1 (and depicted in Fig. 1), the code is turned into a $[4, 5 : 1, 4, 2, 2, 1]_2$ code. The WOM-rate is the

same, but the number of writes is now the length plus one. The code is depicted in Fig. 3.

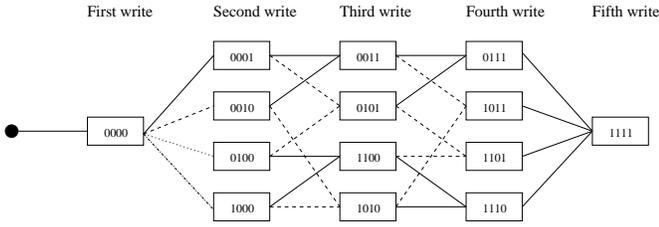


Fig. 3. A binary $[4, 5 : 1, 4, 2, 2, 1]_2$ code obtained from the code of Example 1 by adding a generation prior to all other generations containing the all-zero codeword only. The different types of edges (solid, dashed, dotted, and dash-dotted) correspond to different input data bits.

III. SMALL LAMINAR WOM CODES

In this section, we construct small laminar codes. We first consider codes with $n = t$ that write exactly 1 wit at each generation, and then construct codes with $n > t$.

An exhaustive search for laminar codes that maximize the WOM-rate is unfeasible even for very short codes. Thus, to simplify the search, we use a greedy algorithm that maximizes the values of M_i generation by generation. Consider a code C with $n = t$ that writes exactly 1 wit per generation, and a generation $i > 1$. Assuming that the previous generations are already fixed, the condition we have on M_i is that for every $\mathbf{x} \in \text{Im}(\mathcal{E}_{i-1})$, and for every $m \in \{1, \dots, M_i\}$, there exists $\mathbf{y} \in \text{Im}(\mathcal{E}_i)$ such that $\mathbf{x} \leq \mathbf{y}$ and $\mathcal{D}_i(\mathbf{y}) = m$ (where $\mathbf{x} \leq \mathbf{y}$ if $x_k \leq y_k$ for all k , $1 \leq k \leq n$). Denote by $E(n, i)$ the set of binary vectors of length n and Hamming weight i . It follows that at each generation i , $\text{Im}(\mathcal{E}_i) \subseteq E(n, i)$. We use this set inclusion to make our maximization at each generation completely independent from the other generations, at the cost of optimality.

Let us define the equivalence relation \equiv_i^n on $\text{Im}(\mathcal{E}_i)$ by $\mathbf{y} \equiv_i^n \mathbf{y}'$ if and only if $\mathcal{D}_i(\mathbf{y}) = \mathcal{D}_i(\mathbf{y}')$. Let us refer to the equivalence classes of this relation as the *codeword classes* of C at generation i . Codeword classes are subsets $Y \subseteq E(n, i)$ for which, if we do not take the previous generations into account, the following must hold

$$\forall \mathbf{x} \in E(n, i-1), \exists \mathbf{y} \in Y : \mathbf{x} \leq \mathbf{y}. \quad (3)$$

We are also interested in the partitions of $E(n, i)$ as a set of valid codeword classes. If \mathcal{Y} denotes such a partition, we want that

$$\forall Y \in \mathcal{Y}, \forall \mathbf{x} \in E(n, i-1), \exists \mathbf{y} \in Y : \mathbf{x} \leq \mathbf{y}. \quad (4)$$

Each valid partition \mathcal{Y} corresponds to a valid decoding map (modulo reordering), and thus each cardinality $|\mathcal{Y}|$ to a valid M_i . We are therefore interested in finding the maximum cardinality of such a partition. We make the following important definition.

Definition 3: Let $A(n, i)$ be the maximum cardinality of a partition \mathcal{Y} of $E(n, i)$ satisfying (4).

We now give an upper bound on $A(n, i)$.

Proposition 1: Let $B(n, i)$ be defined by

$$B(n, i) \triangleq \left\lfloor \frac{\binom{n}{i}}{\min_{Y \text{ s.t. (3) holds}} |Y|} \right\rfloor.$$

Then, the maximum cardinality $A(n, i)$ of a partition \mathcal{Y} that satisfies (4) is upper-bounded by $A(n, i) \leq B(n, i)$.

Proof: Let \mathcal{Y} be any partition of $E(n, i)$. Then,

$$|\mathcal{Y}| \cdot \left(\min_{Y \text{ s.t. (3) holds}} |Y| \right) \leq \sum_{Y \in \mathcal{Y}} |Y| = |E(n, i)| = \binom{n}{i}.$$

This holds in particular when \mathcal{Y} is of maximum cardinality. \square

This bound can be computed using a computer search for the smallest Y that satisfies (3). The search is relatively slow, but notice that by lower-bounding $|Y|$ by $\left\lceil \frac{|E(n, i-1)|}{i} \right\rceil$ (each element $\mathbf{y} \in E(n, i)$ covers exactly i elements $\mathbf{x} \in E(n, i-1)$), we obtain a closed-form bound,

$$A(n, i) \leq B(n, i) \leq \left\lfloor \frac{\binom{n}{i}}{\left\lceil \frac{|E(n, i-1)|}{i} \right\rceil} \right\rfloor = \left\lfloor \frac{\binom{n}{i}}{\left\lceil \frac{\binom{n}{i-1}}{i} \right\rceil} \right\rfloor.$$

While the closed-form bound can be computed efficiently and is reached for some values of (n, i) (for instance, for $n \leq 3$, or for $i \leq 2$, or $i = n$), even for relatively low values of n and i , it can be strictly higher than $A(n, i)$. For instance, $A(4, 3) = 1$, while the closed-form bound is 2. Indeed, $E(4, 3) = \{1110, 1101, 1011, 0111\}$ and $E(4, 2) = \{1100, 1010, 1001, 0110, 0101, 0011\}$, and while each element of $E(4, 3)$ covers 3 elements of $E(4, 2)$, it is not possible to pick two elements of $E(4, 3)$ such that the subsets of $E(4, 2)$ that they cover are disjoint. Therefore, the codeword classes in $E(4, 3)$ have cardinality at least 3, and not $\frac{|E(n, i-1)|}{i} = 2$.

For very small values of n , the exact value of $A(n, i)$ can be computed by conducting a simple exhaustive search on the set of codeword classes. Values of $B(n, i)$ are also obtained with an exhaustive search, but on the minimum size of codeword classes, which is significantly faster. The results of the two searches are reported for $n \leq 16$ in Table II. The values in bold font are $A(n, i)$, the others are $B(n, i)$. The few values of $A(n, i)$ that were computed exactly match $B(n, i)$, so it is unknown whether there are pairs (n, i) such that $A(n, i) < B(n, i)$. Note that these values are constructive. For instance, a $[4, 4 : 4, 3, 1, 1]_2^{\text{sync}}$ and a $[5, 5 : 5, 3, 2, 1, 1]_2^{\text{sync}}$ code can be obtained from the search. The upper bounds from Table II in italics match the exact values of $A(n, i)$ by Propositions 4, 5, and 7, or by the lower bounds of Propositions 2 and 3, and are also constructive (see Section III-A below).

A. Bounds on the Sizes of Generations

We give bounds on the sizes of the generations of the codes defined above. In particular, we give lower bounds that are constructive and allow us to effectively build codeword classes for the corresponding generations.

TABLE II

UPPER BOUND $B(n, i)$ ON $A(n, i)$. VALUES IN BOLD ARE EXACT VALUES FOR $A(n, i)$ FOUND BY COMPUTER SEARCH ($A(n, i) = B(n, i)$ IN ALL CASES). THE VALUES FOR $B(n, i)$ IN ITALICS MATCH THE EXACT VALUES OF $A(n, i)$ BY PROPOSITIONS 4, 5, AND 7, OR BY THE LOWER BOUNDS OF PROPOSITIONS 2 AND 3.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
n																
1	<i>1</i>															
2	2	<i>1</i>														
3	3	<i>1</i>	1													
4	4	3	<i>1</i>	1												
5	5	3	2	<i>1</i>	1											
6	6	5	3	2	<i>1</i>	1										
7	7	5	5	2	2	<i>1</i>	1									
8	8	7	5	5	2	2	<i>1</i>	1								
9	9	7	6	5	3	2	2	<i>1</i>	1							
10	10	9	6	5	4	3	2	2	<i>1</i>	1						
11	11	9	7	6	5	4	3	2	2	<i>1</i>	1					
12	12	11	8	6	6	5	3	3	2	<i>1</i>	<i>1</i>	1				
13	13	11	10	7	6	5	4	3	3	2	2	<i>1</i>	1			
14	14	13	10	9	7	6	5	5	3	3	2	2	<i>1</i>	1		
15	15	13	13	9	9	6	5	5	4	3	3	2	2	<i>1</i>	1	
16	16	15	13	13	9	9	7	6	5	4	3	3	2	2	<i>1</i>	1

For $\mathbf{x} = (x_1, \dots, x_n) \in \{0, 1\}^n$ and $\mathbf{x}' = (x'_1, \dots, x'_n) \in \{0, 1\}^{n'}$, we denote by $\mathbf{x} \cdot \mathbf{x}'$ the vector of $\{0, 1\}^{n+n'}$ that is the concatenation of \mathbf{x} and \mathbf{x}' :

$$\mathbf{x} \cdot \mathbf{x}' = (x_1, \dots, x_n, x'_1, \dots, x'_n).$$

We also call \mathcal{Y} a *suitable partition* of $E(n, i)$ if (4) holds, and we do not mind if the union of the elements of \mathcal{Y} is only a strict subset of $E(n, i)$.

Proposition 2: For any $n \geq 2$ and $2 \leq i \leq n$, $A(n, i) \geq \min(A(n-1, i-1), A(n-1, i))$.

Proof: Let \mathcal{Y} be a suitable partition of $E(n-1, i)$ and \mathcal{Z} a suitable partition of $E(n-1, i-1)$ such that $|\mathcal{Y}| = A(n-1, i)$ and $|\mathcal{Z}| = A(n-1, i-1)$. Consider two bijections $f_{\mathcal{Y}} : \{1, \dots, A(n-1, i)\} \rightarrow \mathcal{Y}$ and $f_{\mathcal{Z}} : \{1, \dots, A(n-1, i-1)\} \rightarrow \mathcal{Z}$. Now, define a suitable partition \mathcal{Y}' of $E(n, i)$ as the union for all $1 \leq k \leq \min(A(n-1, i-1), A(n-1, i))$ of the codeword classes

$$(f_{\mathcal{Y}}(k).0) \cup (f_{\mathcal{Z}}(k).1).$$

There is no collision between these codeword classes, since we can sort their elements according to their last symbol, and for a given last symbol, the first $n-1$ symbols of the codewords in a codeword class match a (suitable) partition of $E(n-1, i)$ or one of $E(n-1, i-1)$. The cardinality of \mathcal{Y}' is $\min(A(n-1, i-1), A(n-1, i))$. \square

Proposition 3: For $n \geq 1$, $A(2n, 2) \geq 2A(n, 2) + 1$.

Proof: Let \mathcal{Y} be a partition of $E(n, 2)$ with cardinality $A(n, 2)$ such that for all $Y \in \mathcal{Y}$ and for all $\mathbf{x} \in E(n, 1)$, there is $\mathbf{y} \in Y$ such that $\mathbf{x} \leq \mathbf{y}$. Notice that any $\mathbf{y} \in Y \subseteq E(n, 2)$ can be written as the sum of two weight-one words of length n . Let us denote by \mathbf{e}_j^n the word of length n whose only nonzero coordinate is a 1 at index j . Then every $\mathbf{y} \in Y$ can be written $\mathbf{y} = \mathbf{e}_j^n + \mathbf{e}_k^n$. Let $\mathcal{Y}' \subseteq E(2n, 2)$ be defined by the union of 3 sets of codeword classes as follows.

1) For each $Y \in \mathcal{Y}$, the codeword class

$$\{\mathbf{e}_j^{2n} + \mathbf{e}_k^{2n} | \mathbf{e}_j^n + \mathbf{e}_k^n \in Y\} \cup \{\mathbf{e}_{j+n}^{2n} + \mathbf{e}_{k+n}^{2n} | \mathbf{e}_j^n + \mathbf{e}_k^n \in Y\}.$$

2) For each $Y \in \mathcal{Y}$, the codeword class

$$\{\mathbf{e}_j^{2n} + \mathbf{e}_{k+n}^{2n} | \mathbf{e}_j^n + \mathbf{e}_k^n \in Y\} \cup \{\mathbf{e}_{j+n}^{2n} + \mathbf{e}_k^{2n} | \mathbf{e}_j^n + \mathbf{e}_k^n \in Y\}.$$

3) The codeword class

$$\{\mathbf{e}_j^{2n} + \mathbf{e}_{j+n}^{2n} | 1 \leq j \leq n\}.$$

These codeword classes are trivially disjoint, and each of them covers $E(2n, 1)$. Thus, $A(2n, 2) \geq A(n, 2) + A(n, 2) + 1 = 2A(n, 2) + 1$. \square

Proposition 4: For any $n \geq 1$, $A(n, 1) = n$.

Proof: Partition $E(n, 1)$ into n singletons $\{\mathbf{e}_j^n\}$ for $1 \leq j \leq n$. \square

Proposition 5: For any $n \geq 0$, $A(2^n, 2) = 2^n - 1$.

Proof: We use $A(2n, 2) \geq 2A(n, 2) + 1$ from Proposition 3 and the simple bound $A(n, i) \leq n - i + 1$, which for $i = 2$ becomes $A(n, 2) \leq n - 1$, and proceed by induction. $A(1, 2) = 0$. Assuming $A(2^n, 2) = 2^n - 1$, we have $A(2^{n+1}, 2) \geq 2 \cdot (2^n - 1) + 1 = 2^{n+1} - 1$, and we have $A(2^{n+1}, 2) \leq 2^{n+1} - 1$. \square

Proposition 6: For any $n \geq 1$, $A(2n+1, 2) \leq 2n - 1$.

Proof: This is the bound $A(n, i) \leq \left\lfloor \frac{\binom{n}{i}}{\binom{n-i}{i}} \right\rfloor$ applied

to $i = 2$. \square

Proposition 7: For any $n \geq 1$, $A(2^n + 1, 2) = 2^n - 1$.

Proof: $A(2^n + 1, 2) \geq 2^n - 1$ comes from a direct use of Proposition 2 on the results of Propositions 4 and 5. $A(2^n + 1, 2) \leq 2^n - 1$ comes from Proposition 6. \square

B. Lamellar WOM Codes with $n > t$

The constraint $n = t$ results in relatively low WOM-rates. Lifting this constraint allows to achieve higher WOM-rates. Lamellar codes with n slightly larger than t can easily be derived from the codes with $n = t$ above by merging several generations together: taking, as the new set of codeword classes, the union of the sets of codeword classes of two or more consecutive generations.² For instance, the $[4, 4 : 4, 3, 1, 1]_2^{\text{sync}}$

² n should remain small, because we do not expect to find synchronous codes of WOM-rate higher than nondecodable ones, thus a larger number of cells should be reserved to the nondecodable code.

code can be turned into a $[4, 3 : 4, 3, 2]_2^{\text{sync}}$ code by merging its third and fourth generations together. Instead of having one codeword class at generation 3 ($\{1110, 1101, 1011, 0111\}$) and one at generation 4 ($\{1111\}$), now the third generation has two codeword classes: $\{1110, 1101, 1011, 0111\}$ and $\{1111\}$, and there is no fourth generation anymore. Likewise, a $[5, 3 : 5, 3, 4]_2^{\text{sync}}$ code (of WOM-rate 1.181) can be derived from the $[5, 5 : 5, 3, 2, 1, 1]_2^{\text{sync}}$ code by merging the last three generations together. However, consider the codeword classes of vectors of weight 4. These were constructed in order to cover every word of weight 3, while they now only have to cover every word of weight 2. The optimization also did not allow codeword classes of mixed weights. We can reorganize the set of vectors of weight 3 or more into a better balanced set of codeword classes. In (5), we give the codeword classes of the third generation of a $[5, 3 : 5, 3, 6]_2^{\text{sync}}$ code (of WOM-rate 1.298) obtained by reorganizing the third generation of the $[5, 3 : 5, 3, 4]_2^{\text{sync}}$ code,

$$\begin{aligned} &\{01111, 11001, 10110\}, \{10111, 11100, 01011\}, \\ &\{11011, 01110, 10101\}, \{11101, 00111, 11010\}, \\ &\{11110, 10011, 01101\}, \{11111\}. \end{aligned} \quad (5)$$

For comparison, the 4 codeword classes of the third generation of the $[5, 3 : 5, 3, 4]_2^{\text{sync}}$ code are

$$\begin{aligned} &\{11100, 11010, 10101, 01011, 00111\} \text{ (weight 3 only)}, \\ &\{11001, 10110, 10011, 01110, 01101\} \text{ (weight 3 only)}, \\ &\{11110, 11101, 11011, 10111, 01111\} \text{ (weight 4 only)}, \\ &\{11111\} \text{ (weight 5)}. \end{aligned} \quad (6)$$

Other choices can be made regarding which generations to merge to obtain a 3-write code from the $[5, 5 : 5, 3, 2, 1, 1]_2^{\text{sync}}$ code, but lower WOM-rates are obtained.

IV. A CONSTRUCTION FOR SYNCHRONOUS WOM CODES OF HIGHER t

In this section, we propose a construction to obtain synchronous codes for higher values of t by concatenating n' instances of a synchronous code of length n , and using a second synchronous code of length n' to decide, at each generation, which of the n' instances of the first code are going to be modified.

Theorem 1: Let C be a binary $[n, t : M_1, \dots, M_t]_2$ synchronous code of WOM-rate R , and C' a binary $[n', t' : M'_1, \dots, M'_{t'}]_2$ synchronous code of WOM-rate R' , both not containing the all-zero codeword. Then there exists a binary $[nn', tt' : M_1M'_1, \dots, M_1M'_{t'}, \dots, M_tM'_1, \dots, M_tM'_{t'}]_2$ synchronous code C_1 of WOM-rate $R_1 = \frac{t'}{n}R + \frac{t}{n'}R'$.

This construction is based on three algorithms.

- 1) An algorithm to determine the current generation i of C_1 from the state of the nn' memory cells.
- 2) An encoding algorithm, whose input range depends on i .
- 3) A decoding algorithm.

For $p \in \{1, \dots, t\}$ and $l \in \{1, \dots, t'\}$, we denote by \mathcal{E}_p and \mathcal{D}_p the encoding and decoding maps, respectively, of C at generation p , and by \mathcal{E}'_l and \mathcal{D}'_l the encoding and

decoding maps, respectively, of C' at generation l . We also write $\text{Im}(\mathcal{E}_0) = \{\mathbf{0}_n\}$ (resp. $\text{Im}(\mathcal{E}'_0) = \{\mathbf{0}_{n'}\}$) to denote the fact that the state of a block prior to any write by C (resp. C') is the all-zero codeword of length n (resp. n'). We then denote by g (resp. g') the function that takes a codeword from C (resp. C') and returns the unique generation of C (resp. C') of which it is a codeword. Formally,

$$\begin{aligned} g : \bigcup_{p \in \{0, \dots, t\}} \text{Im}(\mathcal{E}_p) &\rightarrow \{0, \dots, t\} \\ &\mathbf{b} \mapsto p \text{ s.t. } \mathbf{b} \in \text{Im}(\mathcal{E}_p), \\ g' : \bigcup_{l \in \{0, \dots, t'\}} \text{Im}(\mathcal{E}'_l) &\rightarrow \{0, \dots, t'\} \\ &\mathbf{b}' \mapsto l \text{ s.t. } \mathbf{b}' \in \text{Im}(\mathcal{E}'_l). \end{aligned}$$

The fact that C and C' are synchronous guarantees that p and l are unique.

Algorithm 1 Algorithm to Recover the Current Generation

- 1: Input: $\mathbf{b}_1, \dots, \mathbf{b}_{n'}$
 - 2: Output: p, l, i , and \mathbf{b}'
 - 3: $p \leftarrow 0$
 - 4: **for** $k \leftarrow 1$ **to** n' **do**
 - 5: $p_k \leftarrow g(\mathbf{b}_k)$
 - 6: **if** $p_k > p$ **then**
 - 7: $p = p_k$
 - 8: **for** $k \leftarrow 1$ **to** n' **do**
 - 9: $b'_k \leftarrow p_k + 1 - p$ \triangleright Should always be 0 or 1
 - 10: **if** $\mathbf{b}' = \mathbf{1}_{n'}$ or $g'(\mathbf{b}') = t'$ **then**
 - 11: $l \leftarrow 0$
 - 12: $p \leftarrow p + 1$
 - 13: $\mathbf{b}' \leftarrow \mathbf{0}_{n'}$
 - 14: **else**
 - 15: $l \leftarrow g'(\mathbf{b}')$
 - 16: $i \leftarrow (p - 1)t' + l$
-

The key idea is that the nn' wits of C_1 are divided into n' blocks of n wits denoted by \mathbf{b}_k for $k \in \{1, \dots, n'\}$, and the tt' generations are divided into t stages of t' generations. For $p \in \{1, \dots, t\}$ and $l \in \{1, \dots, t'\}$, generation $i = (p - 1)t' + l$ of C_1 is the l -th generation of the p -th stage. At this point, we guarantee that each of the n' blocks of n wits contains a codeword $\mathbf{b}_k \in \text{Im}(\mathcal{E}_{p-1}) \cup \text{Im}(\mathcal{E}_p)$. We call $\mathbf{b}' = (b'_1, \dots, b'_{n'}) \in \text{Im}(\mathcal{E}'_l)$ the binary vector of length n' with entries $b'_k = g(\mathbf{b}_k) - p + 1$, $k \in \{1, \dots, n'\}$. Then, Algorithm 1 can take a codeword of C_1 , and use functions g and g' to determine the current generation i .

Both the encoder and the decoder first use this algorithm to determine the current generation i (actually, they use p and l). They also use the value of \mathbf{b}' . Algorithm 2, described below, is the encoding algorithm, which takes a message $m_1 \in \{1, \dots, M_p M'_{l+1}\}$ and encodes it. This message is decomposed into a message $m \in \{1, \dots, M_p\}$ and a message $m' \in \{1, \dots, M'_{l+1}\}$. We then compute the new \mathbf{b}' as $\mathcal{E}'_{l+1}(m', \mathbf{b}')$ and compare the positions at which it differs from the old one. These positions are the indices $k \in \{1, \dots, n'\}$ of the blocks that will be written (hence switching from generation $p - 1$ to generation p). The only requirement on how these blocks will be written is that after this encoding stage, the modulo M_p sum (in $\{1, \dots, M_p\}$) of the $\mathcal{D}_p(\mathbf{b}_k)$

for $\mathbf{b}_k \in \text{Im}(\mathcal{E}_p)$ is m . Algorithm 2 shows a simple way to achieve this.

Algorithm 2 Encoding Algorithm

```

1: Input:  $\mathbf{b}_1, \dots, \mathbf{b}_{n'}$ , and  $\mathbf{b}'$ ,  $m_1$ ,  $p$ , and  $l$ 
2: Output:  $\mathbf{b}_1, \dots, \mathbf{b}_{n'}$ 
3:  $m \leftarrow 1 + \lfloor (m_1 - 1) / M'_{l+1} \rfloor$ 
4:  $m' \leftarrow 1 + ((m_1 - 1) \bmod M'_{l+1})$ 
5:  $\hat{\mathbf{b}}' \leftarrow \mathcal{E}'_{l+1}(m', \mathbf{b}')$ 
6: for  $k \leftarrow 1$  to  $n'$  do
7:   if  $b'_k = 1$  then
8:      $m \leftarrow m - \mathcal{D}_p(\mathbf{b}_k)$ 
9:   else if  $b'_k = 0 \wedge \hat{b}'_k = 1$  then
10:     $k_0 \leftarrow k$ 
11:  $m \leftarrow 1 + ((m - 1) \bmod M_p)$ 
12: for  $k \leftarrow 1$  to  $n'$  do
13:   if  $\hat{b}'_k = 1 \wedge b'_k = 0 \wedge k \neq k_0$  then
14:     $\mathbf{b}_k \leftarrow \mathcal{E}_p(M_p, \mathbf{b}_k)$ 
15:  $\mathbf{b}_{k_0} \leftarrow \mathcal{E}_p(m, \mathbf{b}_{k_0})$ 

```

The messages m and m' can be decoded by decoding \mathbf{b}' with the decoder of C' , and then decoding every block \mathbf{b}_k for $\mathbf{b}_k \in \text{Im}(\mathcal{E}_p)$ with the decoder of C , and finally taking the modulo M_p sum (in $\{1, \dots, M_p\}$) of the decoded messages. The original message m_1 is then $m_1 = (m - 1)M'_l + m'$. See Algorithm 3 for details.

Algorithm 3 Decoding Algorithm

```

1: Input:  $\mathbf{b}_1, \dots, \mathbf{b}_{n'}$ , and  $\mathbf{b}'$ ,  $p$ , and  $l$ 
2: Output:  $m$ ,  $m'$ , and  $m_1$ 
3:  $m' \leftarrow \mathcal{D}'_l(\mathbf{b}')$ 
4:  $m \leftarrow 0$ 
5: for  $k \leftarrow 1$  to  $n'$  do
6:   if  $b'_k = 1$  then
7:      $m \leftarrow m + \mathcal{D}_p(\mathbf{b}_k)$ 
8:  $m \leftarrow 1 + ((m - 1) \bmod M_p)$ 
9:  $m_1 \leftarrow (m - 1)M'_l + m'$ 

```

Let us now establish the WOM-rate R_1 of C_1 .

$$\begin{aligned}
R_1 &= \frac{\sum_{p=1}^t \sum_{l=1}^{t'} \log_2(M_p M'_l)}{nn'} \\
&= \frac{1}{nn'} \left(\log_2 \left(\prod_{p=1}^t M_p^{t'} \right) + \log_2 \left(\prod_{l=1}^{t'} (M'_l)^t \right) \right) \\
&= \frac{1}{nn'} (t' \cdot nR + t \cdot n'R') = \frac{t'}{n'} R + \frac{t}{n} R'.
\end{aligned}$$

Example 5: Let C be the $[4, 3 : 4, 3, 2]_2^{\text{sync}}$ code defined by

	1	2	3	4
\mathcal{D}_1^{-1}	{0001}	{0010}	{0100}	{1000}
\mathcal{D}_2^{-1}	{1100, 0011}	{1010, 0101}	{1001, 0110}	–
\mathcal{D}_3^{-1}	{0111, 1011, 1101, 1110}	{1111}	–	–

and C' the $[2, 2 : 2, 1]_2^{\text{sync}}$ code defined by

	1	2
$(\mathcal{D}'_1)^{-1}$	{01}	{10}
$(\mathcal{D}'_2)^{-1}$	{11}	–

The code C_1 obtained with the construction is a $[8, 6 : 8, 4, 6, 3, 4, 2]_2^{\text{sync}}$ code. Consider that the eight cells are in state $(\mathbf{b}_1, \mathbf{b}_2) = (1100, 0010)$. Let us first consider the decoding of the message following Algorithms 1 and 3. The generation in C of the first block \mathbf{b}_1 is 2, and that of the second block \mathbf{b}_2 is 1, thus $p = 2$ (the highest of the two) and $\mathbf{b}' = (10)$. The fact that C' is synchronous guarantees that only one encoding function of C' has \mathbf{b}' in its range: here, it is the encoding function for $l = 1$. Thus, we are at the first generation ($l = 1$) of the second stage ($p = 2$), so the overall generation is $i = (p - 1)t' + l = (2 - 1) \times 2 + 1 = 3$. The flow of Algorithm 1 is illustrated in Fig. 4.

For the decoding part, we have $m' = \mathcal{D}'_1(10) = 2$ and m as the modulo M_p sum (in $\{1, \dots, M_p\}$) of $\mathcal{D}_p(\mathbf{b}_k)$ for all indices k of a block at generation p of C . Here, there is only one block at generation $p = 2$ for C : block $\mathbf{b}_1 = (1100)$, therefore $m = (\mathcal{D}_2(\mathbf{b}_1) - 1) \bmod 3 + 1 = 1$. The original message pair was therefore $(1, 2)$. This can be mapped to $m_1 \in \{1, \dots, M_p M'_l\}$ by $m_1 = (m - 1)M'_l + m'$, which gives $m_1 = 0 \times 2 + 2 = 2$. The flow of Algorithm 3 is illustrated in Fig. 5.

For the encoding part, let us now encode a new message $m_1 = 2 \in \{1, 2, 3\}$ for generation 4 following Algorithm 2. Our new m and m' are 2 and 1, respectively, so that $(m - 1)M'_{l+1} + m' = (2 - 1) \times 1 + 1 = 2$. $\mathbf{b}' = (10)$ will become $\mathbf{b}' = (11)$ because $\mathcal{E}'_2(1, 10) = (11)$. Therefore, the second block is going to be written (because the second wit of \mathbf{b}' changes). We first decode all the blocks already at generation $p = 2$: here, we only have one block at generation $p = 2$, and $\mathcal{D}_2(\mathbf{b}_1) = \mathcal{D}_2(1100) = 1$. We therefore encode in the second block \mathbf{b}_2 a message $m_0 = (m - \mathcal{D}_2(\mathbf{b}_1) - 1) \bmod M_p + 1$, where $M_p = M_2 = 3$ and $m = 2$. Thus, $m_0 = 1$ and \mathbf{b}_2 is replaced by $\mathcal{E}_p(1, 0010) = (0011)$. The state of the cells is $(1100, 0011)$ after this encoding phase. The flow of Algorithm 2 is illustrated in Fig. 6.

We remark that the construction above requires that code C does not contain the all-zero codeword. In that case, if the all-zero codeword of C is written in a block, the generation of C_1 would be improperly identified and the component m' of the message could not be written/decoded. The construction also requires C' to not contain the all-zero codeword, in which case the component m of the message could not be written/decoded when the all-zero codeword is chosen for C' .

As a final remark, note that the construction above resembles a tensor-product code construction, but with some important differences. For instance, it is required that the different blocks contain codewords from C of neighboring generations.

A. Results

Let us denote by $F(C, C')$ the code obtained by applying the construction of Theorem 1 to C and C' . We can iterate the

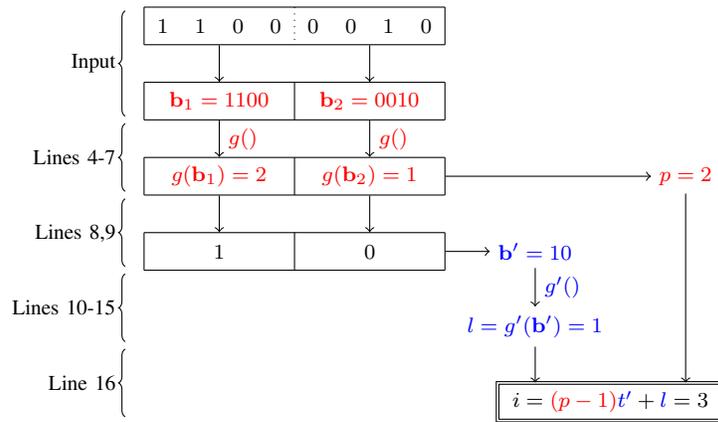


Fig. 4. Example of a run of Algorithm 1.

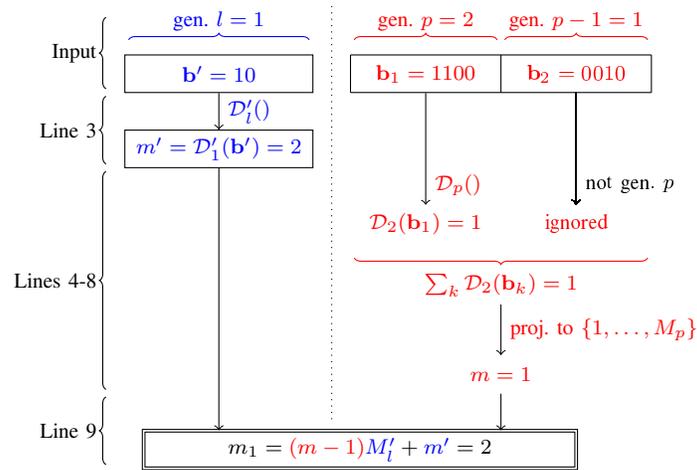


Fig. 5. Example of a run of Algorithm 3.

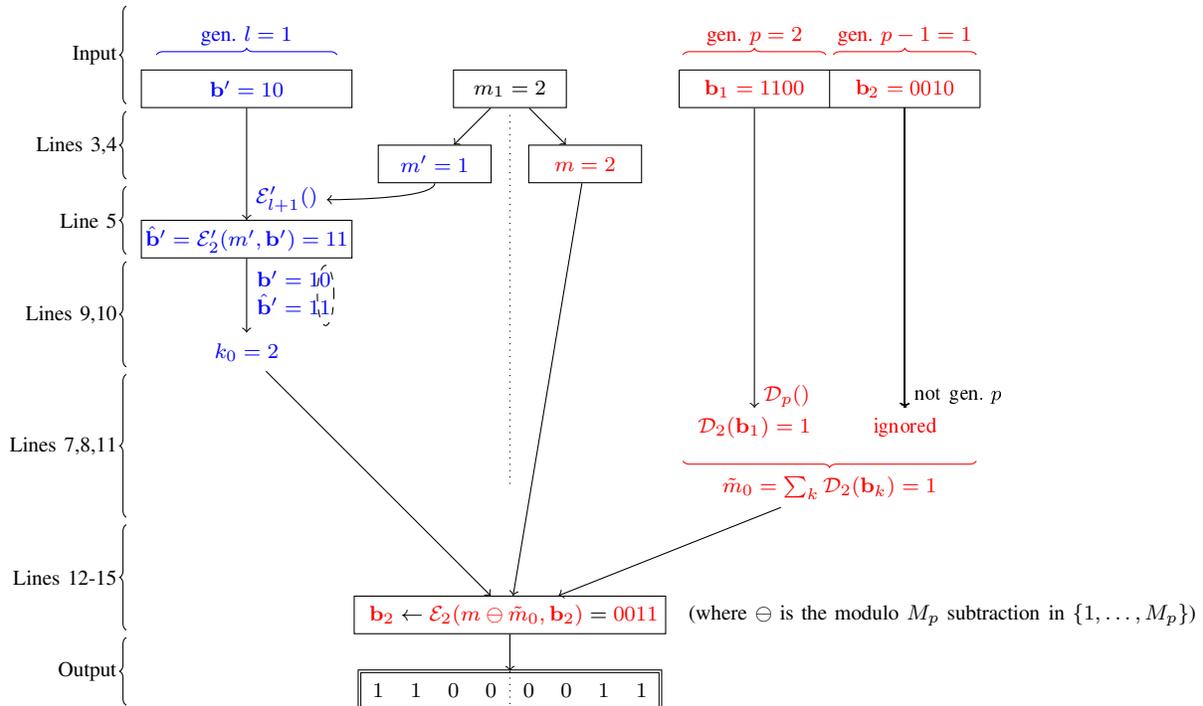


Fig. 6. Example of a run of Algorithm 2.

above construction by choosing C and C' , and then defining $C_0 = C$ and $C_m = F(C_{m-1}, C')$ for all $m > 0$. This generates codes with even higher values of t , which have to be compared with a construction of synchronous codes from [1] (where $n = t$ is any power of two and the WOM-rate is $\log_2(t)/2$). Notice that the two constructions happen to match when we take as $C = C'$ the trivial $[2, 2 : 2, 1]_2^{\text{sync}}$ code.

First, we restrict ourselves to codes with $n = t$ (which are easier to compare) and we fix $C' = C$. The WOM-rate of the t_m -write code C_m after m iterations of the construction is

$$R(C_m) = mR(C) = \log_t(t_m)R(C) = \frac{R(C)}{\log_2(t)} \log_2(t_m).$$

Therefore, for codes with $n = t$, the higher $\frac{R(C)}{\log_2(t)}$ is, the better this iterated construction works. The code that maximizes this ratio among those found by our computer search is the one with $n = t = 2$ (with $\frac{R(C)}{\log_2(t)} = \frac{1}{2}$), making the codes from [1] the best in terms of asymptotic WOM-rate until codes for higher values of $n = t$ are found. For instance, Table II suggests that a $[8, 8 : 8, 7, 5, 5, 2, 2, 1, 1]_2^{\text{sync}}$ code could exist, with a ratio of 0.519 (and even better synchronous codes could exist even for $n = t = 8$, if we remove the added constraints from Section III). However, when t is not a power of two, our construction can yield codes where t has either 2, 3, or 5 as a divisor, but no other prime divisors, i.e., the number of writes is of the form $2^a 3^b 5^c$. This is achieved by mixing different elementary codes C' with 2, 3, and 5 generations, instead of always using the $t = 2$ code. This is a much denser coverage of the potential values of t . Furthermore, if we consider codes with n slightly greater than t , we can reach higher WOM-rates at equal values of t . Consider, for instance, the code $F(C, C')$ with C the $[4, 3 : 4, 3, 2]_2^{\text{sync}}$ code and C' the $[2, 2 : 2, 1]_2^{\text{sync}}$ code. The construction then yields a $[8, 6 : 8, 4, 6, 3, 4, 2]_2^{\text{sync}}$ code of WOM-rate 1.521 (larger than $\log_2(t)/2$ both for $t = 6$ and $t = 8$). This is the example code of Example 5.

V. FIXED-RATE WOM CODES

In Sections III and IV we did not impose any constraint on the values $\{M_i\}$. Therefore, the obtained codes are in general unrestricted-rate codes, i.e., the codes store in general a different number of messages at different generations. Appending these codes to a nondecodable code to make it decodable will clearly result into an unrestricted-rate code.

In this section, given a fixed-rate nondecodable code, we consider the problem of efficiently generating a fixed-rate decodable code. Note that the standard method of appending $t_{\text{nd}} - 1$ cells to a t_{nd} -write nondecodable code that only store the current generation results in a fixed-rate code, since it does not change the values of $\{M_i\}$. However, we can also improve the WOM-rate of the overall code, by appending a short synchronous code as in the previous sections, with the additional constraint that the synchronous code must also be fixed-rate. We are therefore interested in finding short synchronous fixed-rate codes.

The main result of this section is that the construction of Section IV yields a fixed-rate code when applied to two fixed-rate codes. To find fixed-rate synchronous codes for many

values of t , one therefore only has to find a few such codes for small values of t . In the following, we propose two such codes.

- A $[3, 2 : 2, 2]_2^{\text{sync}}$ code of WOM-rate 2/3 given by

$$\begin{array}{c|cc} & 1 & 2 \\ \hline \mathcal{D}_1^{-1} & \{001\} & \{010\} \\ \mathcal{D}_2^{-1} & \{110, 101\} & \{011\} \end{array}.$$

- A $[5, 3 : 4, 4, 4]_2^{\text{sync}}$ code of WOM-rate 1.2 where the classes are:

- At generation 1: $\{00001\}$, $\{00010\}$, $\{00100\}$, and $\{01000\}$.
- At generation 2: $\{11000, 10100, 10010, 10001\}$, $\{01100, 00011\}$, $\{01010, 00101\}$, and $\{01001, 00110\}$.
- At generation 3: the same codeword classes as in (6).

We remark that fixed-rate codes have not only lower WOM-rate than unrestricted-rate codes, but when we add the constraint that the codes must be synchronous and with $n = t - 1$, this gets even worse as the last generation of a synchronous code with $n = t - 1$ will always have size 1, forcing the size of every generation to be 1 for a fixed-rate code, and making its WOM-rate 0. This explains why the two codes that we give have n larger than $t - 1$.

VI. EXTENSION TO q -ARY WOM CODES

The proposed method of Section II for making a nondecodable code decodable in the binary case can be extended to the problem of making nondecodable q -ary codes decodable for $q > 2$. The number of additional cells required to make a q -ary t_{nd} -write nondecodable code decodable is $\left\lceil \frac{t_{\text{nd}} - 1}{q - 1} \right\rceil$. Indeed, during each of the last $t_{\text{nd}} - 1$ generations, the sum of the values in the additional cells is increased by at least 1, and this sum is at most $q - 1$ times the number of additional cells. We consider the problem of building synchronous q -ary $(t_{\text{nd}} - 1)$ -write codes with length $\left\lceil \frac{t_{\text{nd}} - 1}{q - 1} \right\rceil$ (or slightly above) which do not contain the all-zero codeword, since, as in the binary case, we can later add an extra generation containing only the all-zero codeword, turning the code into a t_{nd} -write code of length $\left\lceil \frac{t_{\text{nd}} - 1}{q - 1} \right\rceil$. If $t_{\text{nd}} \leq q$, then only one additional cell is required. This case applies to the codes in [11], for instance, with $q = 8$ and $t_{\text{nd}} = 2, 3, 4, 5, 6, 7$, or $q = 4$ and $t_{\text{nd}} = 2, 3, 4$. Then, the WOM-rate of a code is determined entirely by the assignment of the q possible values of the cell to its generations. For instance, if $q = 5$ and $t_{\text{nd}} = 3$, we can choose $\text{Im}(\mathcal{E}_1) = \{0, 1\}$, $\text{Im}(\mathcal{E}_2) = \{2, 3\}$, and $\text{Im}(\mathcal{E}_3) = \{4\}$. The WOM-rate of the resulting code would therefore be $\log_2(2 \times 2 \times 1)$. Maximizing the WOM-rate of the code is equivalent to maximizing the product $\prod_{i=1}^{t_{\text{nd}}} M_i$ where the only constraints on the M_i 's are that they are integers from $\{1, \dots, q\}$ and that $\sum_{i=1}^{t_{\text{nd}}} M_i \leq q$. Maximizing a product of integers given their sum is achieved by choosing them as close to each other as possible, here by picking $M_i \in \{\lfloor q/t_{\text{nd}} \rfloor, \lceil q/t_{\text{nd}} \rceil\}$ for all i . Let us consider the two extreme regimes. If $t_{\text{nd}} = q/2$ (resp. $t_{\text{nd}} > q/2$), we pick $M_i \in \{2, 2\}$ (resp. $M_i \in \{1, 2\}$) and the resulting WOM-rate

TABLE III

$A_q(n, i)$ IN THE TERNARY CASE ($q = 3$). THE VALUES ARE CONSTRUCTIVE (I.E., THEY CORRESPOND TO ACTUAL CODES FOUND BY AN EXHAUSTIVE SEARCH). VALUES IN ITALICS CAN ALSO BE TAKEN FROM PROPOSITION 9.

i	1	2	3	4	5	6	7	8
n								
1	<i>1</i>	1						
2	2	2	1	1				
3	3	3	2	1	1	1		
4	4	4	3	3	1	1	1	1
5	5	5	-	-	-	-	-	-

TABLE IV

$A_q(n, i)$ IN THE QUATERNARY CASE ($q = 4$). THE VALUES ARE CONSTRUCTIVE (I.E., THEY CORRESPOND TO ACTUAL CODES FOUND BY AN EXHAUSTIVE SEARCH). VALUES IN ITALICS CAN ALSO BE TAKEN FROM PROPOSITION 9.

i	1	2	3	4	5	6	7	8	9
n									
1	<i>1</i>	1	1						
2	2	2	2	1	1	1			
3	3	3	3	2	1	1	1	1	1
4	4	4	4	3	-	-	-	-	-

is $\log_2(2^{q-t_{\text{nd}}}) = q - t_{\text{nd}}$ (resp. $\log_2(2^{q-t_{\text{nd}}}) = q - t_{\text{nd}}$), while if t_{nd} is small compared to q , the optimal WOM-rate can be closely approximated by $\log_2\left(\prod_{i=1}^{t_{\text{nd}}} q/t_{\text{nd}}\right) = t_{\text{nd}} \log_2(q/t_{\text{nd}})$.

If $t_{\text{nd}} > q$, then several additional cells are required. Using a computer search, we can find a few very short synchronous codes for $q > 2$ under the same constraints as the codes from Section III (laminar, with $n = \lfloor \frac{t}{q-1} \rfloor$, and where generation i is built assuming that all codewords of weight (or L_1 -norm) $i-1$ are used by generation $i-1$). Furthermore, in analogy with the binary case, we make the following important definition.

Definition 4: Let $E_q(n, i)$ be the set of q -ary vectors of length n and weight i , and $A_q(n, i)$ the maximum size of a partition \mathcal{Y} of $E_q(n, i)$ so that

$$\forall Y \in \mathcal{Y}, \forall \mathbf{x} \in E_q(n, i-1), \exists \mathbf{y} \in Y : \mathbf{x} \leq \mathbf{y}.$$

As in the binary case, we would like to compute $A_q(n, i)$ for different values of n and i . Tables III and IV show the results of such a search for $q = 3, 4$ and small values of n . As an example, a $[2, 6 : 2, 2, 2, 1, 1, 1]_4^{\text{sync}}$ code of WOM-rate $3/2$ (which corresponds to the second row of Table IV) given by

	1	2
\mathcal{D}_1^{-1}	{01}	{10}
\mathcal{D}_2^{-1}	{11}	{20, 02}
\mathcal{D}_3^{-1}	{21, 03}	{12, 30}
\mathcal{D}_4^{-1}	{13, 31, 22}	-
\mathcal{D}_5^{-1}	{23, 32}	-
\mathcal{D}_6^{-1}	{33}	-

was found.

A. Bounds on the Sizes of Generations

The bounds from Section III-A can also be extended to the q -ary case for laminar codes with $n = \lfloor \frac{t}{q-1} \rfloor$ and the size

of each generation maximized assuming no knowledge of the previous generation.

Proposition 8: For any $n \geq 2$, $q \geq 2$, and $2 \leq i \leq n$, $A_q(n, i) \geq \min(A_q(n-1, i-1), A_q(n-1, i))$.

Proof: The proof follows the same lines as the proof of Proposition 2, with the suitable partition \mathcal{Y}' of $E_q(n, i)$ defined as the union for all $1 \leq k \leq \min(A_q(n-1, i-1), A_q(n-1, i))$ of the codeword classes

$$(f_{\mathcal{Y}}(k).0) \cup \bigcup_{s=1}^{q-1} (f_{\mathcal{Z}}(k).s).$$

□

Proposition 9: For any $n \geq 1$ and $q \geq 2$, $A_q(n, 1) = n$.

Proof: Same proof as for Proposition 4. □

Proposition 10: For $n \geq 1$ and $q \geq 3$, $A_q(n, 2) \geq A_2(n, 2) + 1$.

Proof: Consider a suitable partition \mathcal{Y} of $E_2(n, 2)$ of cardinality $A_2(n, 2)$. Now consider $\mathcal{Y}' = \mathcal{Y} \cup \{2\mathbf{e}_k^n \mid 1 \leq k \leq n\}$. The cardinality of \mathcal{Y}' is $A_2(n, 2) + 1$, the words in its codeword classes have weight 2, and they belong to $E_q(n, 2)$. There is no collision since \mathcal{Y} has no collision, and the words we add are not in $E_2(n, 2)$. □

Proposition 11: For $n \geq 0$ and $q \geq 3$, $A_q(2^n, 2) \geq 2^n$.

Proof: It follows from direct application of Propositions 5 and 10. □

Proposition 12: For $n \geq 0$ and $q \geq 3$, $A_q(2n+1, 2) \geq 2n+1$.

Proof: The idea is to consider a codeword class whose circular permutations do not overlap. For $n = 3$, such a codeword class is $\{0002000, 0010100, 0100010, 1000001\}$.

Formally, let us consider the following codeword class Y_0 of $E_q(2n+1, 2)$:

$$Y_0 = \{\mathbf{e}_{n+1-k} + \mathbf{e}_{n+1+k} \mid 0 \leq k \leq n\}.$$

Y_0 covers $E_q(2n+1, 1)$. If $\mathcal{Y} = \{Y_0, Y_1, \dots, Y_{2n}\}$ is the family of the circular permutations of Y_0 , then \mathcal{Y} is a suitable partition of $E_q(2n+1, 2)$. Indeed, for a given right circular permutation of $(\mathbf{e}_{n+1-k} + \mathbf{e}_{n+1+k})$, k can be identified as follows.

- The vector has a 2 if and only if $k = 0$.
- Otherwise, it has two 1's at indices i_1 and i_2 with $i_1 < i_2$. If $i_2 - i_1$ is even, $k = \frac{i_2 - i_1}{2}$ and we have permuted $(\mathbf{e}_{n+1-k} + \mathbf{e}_{n+1+k})$ to the right $i_1 - n - 1 + k$ times. If $i_2 - i_1$ is odd, $k = \frac{2n+1+i_1-i_2}{2}$ and we have permuted $(\mathbf{e}_{n+1-k} + \mathbf{e}_{n+1+k})$ to the right $i_2 - n - 1 + k$ times.

The cardinality of \mathcal{Y} is $2n+1$, which is a lower bound on the maximum cardinality of a suitable partition of $E_q(2n+1, 2)$. □

Finally, we remark that the lower bounds of Propositions 8, 10, 11, and 12 match the exact values of $A_q(n, i)$ from Tables III and IV for several values of (n, i) .

B. The Construction from Section IV

The construction of Section IV can be extended to q -ary codes as follows.

Theorem 2: Let C be an $[n, t : M_1, \dots, M_t]_q$ synchronous q -ary code of WOM-rate R , and C' an $[n', t'] :$

$M'_1, \dots, M'_{t'}$ synchronous binary code of WOM-rate R' , both not containing the all-zero codeword. Then there exists an $[nn', tt' : M_1 M'_1, \dots, M_1 M'_{t'}, \dots, M_t M'_1, \dots, M_t M'_{t'}]_q$ synchronous q -ary code C_1 of WOM-rate $R_1 = \frac{t'}{n'}R + \frac{t}{n}R'$.

Proof: The proof that C_1 is a valid synchronous q -ary code is the same as in the binary case. \square

Notice that the code C' in the construction is still binary: the requirement is that C and C_1 must have the same alphabet size. Using a q' -ary code (with $q' > 2$) instead of a binary code is also possible regardless of C and C_1 . When C' is binary, the two values 0 and 1 will be matched, at each stage, to 2 successive generations $p-1$ and p of C . In the first stage they are matched to generations 0 (i.e., empty memory) and 1, then to generations 1 and 2, and so on. However, when C' is q' -ary with $q' > 2$, each stage has q' possible values to match to q' generations. For instance, if $q' = 4$, the values (0, 1, 2, 3) will be matched to generations (0, 1, 2, 3) of C at stage 1, then to generations (3, 4, 5, 6) at stage 2, generations (6, 7, 8, 9) at stage 3, and so on.

If a nonbinary code C' is to be used, then either C or C' must have a suitable structure. The following conditions, for example, would ensure this.

- A first sufficient condition is that each write of C' increases the sum of the values of its cells by exactly one. This prevents the following situation from happening. Consider the case where at the first generation of a nonbinary C' , a cell can go both from 0 to 1 and from 0 to 2 depending on which message we encode. Then, in the corresponding block, we will write a codeword of C of either generation 1 or generation 2. When encoding a pair (m, m') of messages, the number of messages among which we can choose m therefore depends on m' , which means that the encoder cannot predict how much data it will be able to store at a given generation.
- Another possible condition to avoid the above issue is that we choose a fixed-rate code C . In the previous example, if $M_1 = M_2$, it does not matter if we do not know whether we will be using generation 1 or generation 2 of C ; we have the same number of messages to choose from anyway.

As an example, a $[4, 10 : 4, 2, 4, 2, 6, 3, 4, 2, 2, 1]_4^{\text{sync}}$ code can be constructed in the following way. First, a $[2, 4 : 2, 2, 3, 3]_4^{\text{sync}}$ code can be made by merging together the last three generations of the $[2, 6 : 2, 2, 2, 1, 1, 1]_4^{\text{sync}}$ code displayed above in Section VI by taking as the new set of codeword classes the union of the sets of codeword classes of the three last generations, and reorganizing them, as explained for the binary case in Section III-B. Also, if the codeword classes are reorganized properly, then an additional codeword class $\{22\}$ can be added to the third generation, resulting in the following $[2, 4 : 2, 2, 3, 3]_4^{\text{sync}}$ code

	1	2	3
\mathcal{D}_1^{-1}	{01}	{10}	–
\mathcal{D}_2^{-1}	{11}	{20, 02}	–
\mathcal{D}_3^{-1}	{21, 03}	{12, 30}	{22}
\mathcal{D}_4^{-1}	{13, 32}	{31, 23}	{33}

of WOM-rate 2.5850.³ This is the example code of Example 2. Obviously, a $[2, 5 : 2, 2, 3, 2, 1]_4^{\text{sync}}$ code can be made by splitting the fourth generation into the two generations $\{\{13, 32\}, \{31, 23\}\}$ and $\{33\}$. Finally, by using the construction of Theorem 2, using the $[2, 5 : 2, 2, 3, 2, 1]_4^{\text{sync}}$ code as C and the $[2, 2 : 2, 1]_2^{\text{sync}}$ code from Example 5 as C' , a $[4, 10 : 4, 2, 4, 2, 6, 3, 4, 2, 2, 1]_4^{\text{sync}}$ code of WOM-rate 3.5425 can be constructed.

VII. RESULTS AND COMPARISON WITH THE STANDARD METHOD

In this section, we use the synchronous codes derived in the previous sections to construct decodable codes from nondecodable ones as explained in Section II (binary case) and Section VI (nonbinary case). We compare the proposed method with the basic method that adds $\left\lceil \frac{t_{\text{nd}}-1}{q-1} \right\rceil$ cells containing no data. For this comparison, we consider two different target code lengths, $n = 64$ and $n = 256$. We then assume for each value of n and for some specific values of t_{nd} , that there exists a t_{nd} -write code with WOM-rate equal to the best (i.e., of highest WOM-rate) codes from [11, 12], and with length $n_{\text{nd}} = n - n_{\text{sync}}$, where n_{sync} is the length of the synchronous code. Note that we do not use the actual code lengths at which these state-of-the-art WOM-rates are reached because they are very large [19] and not explicitly stated in [11, 12]. However, this gives a meaningful comparison, since the rate loss with our approach (see (2)) is an increasing function of R_{nd} when n , n_{sync} , and $R_{\text{sync}} > 0$ are fixed. Since no code (for any block length) of strictly higher WOM-rate than the ones reported in [11, 12] is (as far as we can tell) currently known, and considering a specific block length n_{nd} will likely reduce the WOM-rate of the best nondecodable code, the comparison is a sort of worst-case scenario for our approach.

The results for the binary case are reported in Tables V and VI. We consider values for t_{nd} between 4 and 7. The second column of each table reports the state-of-the-art WOM-rate of nondecodable codes, for each value of t_{nd} . The third column shows the WOM-rate that is obtained by appending $t_{\text{nd}} - 1$ cells with no data to a length $n_{\text{nd}} = n - (t_{\text{nd}} - 1)$ code with WOM-rate equal to the one reported in the second column. The next two columns show, for various synchronous codes, the WOM-rate that we obtain for the same target length. The $[3, 4 : 1, 3, 1, 1]_2^{\text{sync}}$, $[4, 5 : 1, 4, 3, 1, 1]_2^{\text{sync}}$, $[5, 6 : 1, 5, 3, 2, 1, 1]_2^{\text{sync}}$, and $[6, 7 : 1, 6, 5, 3, 1, 1, 1]_2^{\text{sync}}$ codes are obtained by adding to the codes $[3, 3 : 3, 1, 1]_2^{\text{sync}}$, $[4, 4 : 4, 3, 1, 1]_2^{\text{sync}}$, $[5, 5 : 5, 3, 2, 1, 1]_2^{\text{sync}}$, and $[6, 6 : 6, 5, 3, 1, 1, 1]_2^{\text{sync}}$ from Section III a generation containing the all-zero codeword.⁴ The $[5, 4 : 1, 5, 3, 6]_2^{\text{sync}}$ code is obtained in a similar manner from the $[5, 3 : 5, 3, 6]_2^{\text{sync}}$ code in Section III-B, and the $[8, 7 : 1, 8, 4, 6, 3, 4, 2]_2^{\text{sync}}$ code

³By adding a generation containing the all-zero codeword, we get a $[2, 5 : 1, 2, 2, 3, 3]_4^{\text{sync}}$ code of the same WOM-rate, which is significantly higher than the corresponding *worst-case* WOM-rate of the synchronous lattice-based code from [14, Table I].

⁴Note that from Table II, $B(6, 4) = 2$, which implies that a $[6, 6 : 6, 5, 3, 2, 1, 1]_2^{\text{sync}}$ code may exist. However, we have not been able to identify such a code in a (nonexhaustive) computer search. The best code found was a $[6, 6 : 6, 5, 3, 1, 1, 1]_2^{\text{sync}}$ code.

TABLE V

WOM-RATES OF BINARY DECODABLE CODES OBTAINED BY CONCATENATING SYNCHRONOUS CODES, WITH TARGET CODE LENGTH $n = 64$. THE NUMBERS IN THE PARENTHESES (IN COLUMNS THREE AND FIVE) ARE THE RATE LOSSES (COMPUTED FROM (1) AND (2), RESPECTIVELY) IN PERCENT, WHILE THE RATE LOSS REDUCTION FACTOR IS THEIR FRACTION.

t_{nd}	Rate of nondec. code from [12, Table VI]	Rate of dec. with no data	With data		Rate loss reduction factor
			Sync. code	Rate	
4	1.8566	1.7696 (4.69%)	$[3, 4 : 1, 3, 1, 1]_2$	1.7943 (3.35%)	1.40
			$[5, 4 : 1, 5, 3, 6]_2$	1.8130 (2.35%)	2.00
5	1.9689	1.8458 (6.25%)	$[4, 5 : 1, 4, 3, 1, 1]_2$	1.9019 (3.41%)	1.84
6	2.1331	1.9665 (7.81%)	$[5, 6 : 1, 5, 3, 2, 1, 1]_2$	2.0431 (4.22%)	1.85
7	2.1723	1.9686 (9.38%)	$[6, 7 : 1, 6, 5, 3, 1, 1, 1]_2$	2.0701 (4.71%)	1.99
			$[8, 7 : 1, 8, 4, 6, 3, 4, 2]_2$	2.0909 (3.75%)	2.50

TABLE VI

WOM-RATES OF BINARY DECODABLE CODES OBTAINED BY CONCATENATING SYNCHRONOUS CODES, WITH TARGET CODE LENGTH $n = 256$. THE NUMBERS IN THE PARENTHESES (IN COLUMNS THREE AND FIVE) ARE THE RATE LOSSES (COMPUTED FROM (1) AND (2), RESPECTIVELY) IN PERCENT, WHILE THE RATE LOSS REDUCTION FACTOR IS THEIR FRACTION.

t_{nd}	Rate of nondec. code from [12, Table VI]	Rate of dec. with no data	With data		Rate loss reduction factor
			Sync. code	Rate	
4	1.8566	1.8348 (1.17%)	$[3, 4 : 1, 3, 1, 1]_2$	1.8410 (0.84%)	1.40
			$[5, 4 : 1, 5, 3, 6]_2$	1.8457 (0.59%)	2.00
5	1.9689	1.9381 (1.56%)	$[4, 5 : 1, 4, 3, 1, 1]_2$	1.9521 (0.85%)	1.84
6	2.1331	2.0914 (1.95%)	$[5, 6 : 1, 5, 3, 2, 1, 1]_2$	2.1106 (1.05%)	1.85
7	2.1723	2.1214 (2.34%)	$[6, 7 : 1, 6, 5, 3, 1, 1, 1]_2$	2.1467 (1.18%)	1.99
			$[8, 7 : 1, 8, 4, 6, 3, 4, 2]_2$	2.1520 (0.94%)	2.50

is obtained by adding a generation with the all-zero codeword to the $[8, 6 : 8, 4, 6, 3, 4, 2]_2^{\text{sync}}$ code from the construction of Section IV.

To better quantify the gains of the proposed approach, we have included in the tables the rate losses compared to the nondecodable code, and also their fraction (the rate loss reduction factor), which quantifies the reduction in rate loss of the proposed approach compared to the basic approach of appending $t_{\text{nd}} - 1$ cells containing no data. For both lengths, our technique yields higher WOM-rates compared to just appending a block of $t_{\text{nd}} - 1$ cells with no information. For instance, for $t_{\text{nd}} = 7$ and $n = 64$, the rate loss with the basic approach is as high as 9.38%. With the improved approach the rate loss is reduced to 3.75%, which is a reduction by a factor of 2.5 (see the sixth column of Table V). As can be seen from the tables, the rate loss of the basic approach grows with t_{nd} . In all cases we are able to demonstrate a rate loss reduction factor of 1.8 to 2.5 using our approach, which is significant. Furthermore, the tabulated WOM-rates are (to the best of our knowledge) also higher than the best WOM-rates for binary multiple-write codes (and hence better than the WOM-rates of any directly decodable code) known prior to [12], which justifies our approach.

The results for the nonbinary case with $q = 4$ are reported in Tables VII and VIII for $n = 64$ and $n = 256$, respectively. Here, we consider values for t_{nd} between 5 and 11. As in the binary case, the second column of each table reports the state-of-the-art WOM-rate of nondecodable quaternary codes, for each value of t_{nd} that we consider. The third column shows the WOM-rate that would be obtained by appending $\lceil \frac{t_{\text{nd}}-1}{3} \rceil$ cells containing no data to a code of length $n - \lceil \frac{t_{\text{nd}}-1}{3} \rceil$ and WOM-rate equal to the one reported in the second column. Note

that similar to the binary case, the codes that we have constructed in Section VI can be extended by a single generation containing the all-zero codeword only. Thus, when we speak below about codes that are constructed in previous sections, we implicitly assume that they have been extended in this way. Now, the codes $[2, 7 : 1, 2, 2, 2, 1, 1, 1]_4^{\text{sync}}$ and $[3, 10 : 1, 3, 3, 3, 2, 1, 1, 1, 1, 1]_4^{\text{sync}}$ are taken from Section VI (the second and third rows of Table IV, respectively), the codes $[3, 8 : 1, 3, 3, 3, 2, 1, 1, 3]_4^{\text{sync}}$ and $[3, 9 : 1, 3, 3, 3, 2, 1, 1, 1, 2]_4^{\text{sync}}$ are obtained by merging the last three (resp. two) generations of the $[3, 10 : 1, 3, 3, 3, 2, 1, 1, 1, 1, 1]_4^{\text{sync}}$ code, and the codes $[2, 5 : 1, 2, 2, 3, 3]_4^{\text{sync}}$, $[2, 6 : 1, 2, 2, 3, 2, 1]_4^{\text{sync}}$, and $[4, 11 : 1, 4, 2, 4, 2, 6, 3, 4, 2, 2, 1]_4^{\text{sync}}$ are taken from Section VI-B. Note that as in the binary case our technique yields higher WOM-rates compared to just appending a block of $\lceil \frac{t_{\text{nd}}-1}{3} \rceil$ cells with no information, for both target lengths. Also, as in the binary case, the rate loss of the basic approach grows with t_{nd} , and we demonstrate a rate loss reduction by a factor between 1.5 and 4.0 in all cases considered, which is significant.

For the ternary case, to the best of our knowledge, no tables of the best possible WOM-rates have been presented in the literature. There are however constructions that can be used. See, for instance, [12, Theorem 7] for constructing q -ary 2-write codes. Here, we will use a construction from [11] (which was inspired by a similar idea proposed in [20]) giving a q -ary $2(q-1)$ -write code of WOM-rate $(q-1)R_2$, where R_2 is the best possible WOM-rate of a 2-write binary code. Thus, there exists a ternary 4-write code of WOM-rate $(3-1) \cdot 1.4928 = 2.9856$ where the WOM-rate of the 2-write code is taken from [12, Table VI]. Now, from the second row of Table III, we can see that there exists a $[2, 3 : 2, 2, 2]_3^{\text{sync}}$ code (by merging

TABLE VII

WOM-RATES OF QUATERNARY ($q = 4$) DECODABLE CODES OBTAINED BY CONCATENATING SYNCHRONOUS CODES, WITH TARGET CODE LENGTH $n = 64$. THE NUMBERS IN THE PARENTHESES (IN COLUMNS THREE AND FIVE) ARE THE RATE LOSSES (COMPUTED FROM (1) AND (2), RESPECTIVELY) IN PERCENT, WHILE THE RATE LOSS REDUCTION FACTOR IS THEIR FRACTION.

t_{nd}	Rate of nondec. code	Rate of dec. with no data	With data Sync. code	Rate	Rate loss reduction factor
5	3.9328 [11]	3.8099 (3.13%)	$[2, 5 : 1, 2, 2, 3, 3]_4$	3.8907 (1.07%)	2.92
6	4.2594 [11]	4.1263 (3.13%)	$[2, 6 : 1, 2, 2, 3, 2, 1]_4$	4.1979 (1.44%)	2.17
7	4.3394 [11]	4.2038 (3.13%)	$[2, 7 : 1, 2, 2, 2, 1, 1, 1]_4$	4.2507 (2.04%)	1.53
8	4.5088 ^a	4.2975 (4.69%)	$[3, 8 : 1, 3, 3, 3, 2, 1, 1, 3]_4$	4.4121 (2.14%)	2.19
9	4.5836 ^a	4.3687 (4.69%)	$[3, 9 : 1, 3, 3, 3, 2, 1, 1, 1, 2]_4$	4.4743 (2.38%)	1.97
10	4.6932 ^a	4.4732 (4.69%)	$[3, 10 : 1, 3, 3, 3, 2, 1, 1, 1, 1, 1]_4$	4.5631 (2.77%)	1.69
11	4.7193 ^b	4.4243 (6.25%)	$[4, 11 : 1, 4, 2, 4, 2, 6, 3, 4, 2, 2, 1]_4$	4.6457 (1.56%)	4.01

^aObtained by applying Construction A from [11] to the WOM-rates from [12, Table VI].

^bObtained by applying Construction A from [11] to the WOM-rates from the recursion for \mathcal{R}'_t of Section VI in [12].

TABLE VIII

WOM-RATES OF QUATERNARY ($q = 4$) DECODABLE CODES OBTAINED BY CONCATENATING SYNCHRONOUS CODES, WITH TARGET CODE LENGTH $n = 256$. THE NUMBERS IN THE PARENTHESES (IN COLUMNS THREE AND FIVE) ARE THE RATE LOSSES (COMPUTED FROM (1) AND (2), RESPECTIVELY) IN PERCENT, WHILE THE RATE LOSS REDUCTION FACTOR IS THEIR FRACTION.

t_{nd}	Rate of nondec. code	Rate of dec. with no data	With data Sync. code	Rate	Rate loss reduction factor
5	3.9328 [11]	3.9021 (0.78%)	$[2, 5 : 1, 2, 2, 3, 3]_4$	3.9223 (0.27%)	2.92
6	4.2594 [11]	4.2261 (0.78%)	$[2, 6 : 1, 2, 2, 3, 2, 1]_4$	4.2440 (0.36%)	2.17
7	4.3394 [11]	4.3055 (0.78%)	$[2, 7 : 1, 2, 2, 2, 1, 1, 1]_4$	4.3172 (0.51%)	1.53
8	4.5088 ^a	4.4560 (1.17%)	$[3, 8 : 1, 3, 3, 3, 2, 1, 1, 3]_4$	4.4846 (0.54%)	2.19
9	4.5836 ^a	4.5299 (1.17%)	$[3, 9 : 1, 3, 3, 3, 2, 1, 1, 1, 2]_4$	4.5563 (0.60%)	1.97
10	4.6932 ^a	4.6382 (1.17%)	$[3, 10 : 1, 3, 3, 3, 2, 1, 1, 1, 1, 1]_4$	4.6607 (0.69%)	1.69
11	4.7193 ^b	4.6456 (1.56%)	$[4, 11 : 1, 4, 2, 4, 2, 6, 3, 4, 2, 2, 1]_4$	4.7009 (0.39%)	4.01

^aObtained by applying Construction A from [11] to the WOM-rates from [12, Table VI].

^bObtained by applying Construction A from [11] to the WOM-rates from the recursion for \mathcal{R}'_t of Section VI in [12].

the last two generations) that does not contain the all-zero codeword. Assuming a block length of $n = 64$, our method gives a WOM-rate of 2.9392, while the method of appending $\left\lceil \frac{4-1}{3-1} \right\rceil = 2$ cells with no data gives a WOM-rate of only 2.8923. This amounts to a rate loss reduction by a factor of 2.01.

VIII. CONCLUSION

In this paper, we proposed short synchronous WOM codes as a basic tool to make nondecodable codes decodable while preserving the WOM-rate as much as possible. We considered both binary and nonbinary codes, as well as the fixed-rate and the unrestricted-rate setups. We constructed short synchronous (laminar) codes for small values of t . We also proposed a construction method to build synchronous codes for higher values of t by concatenating shorter synchronous codes. Compared to the construction by Rivest and Shamir, which considers $n = t$ with t being a power of 2, our construction is more general, since it lifts both constraints. Finally, we used the obtained synchronous codes to make some nondecodable codes decodable. Compared to the standard approach of appending cells containing no data, the proposed approach achieves a significant reduction of the rate loss for short-to-moderate block lengths.

ACKNOWLEDGMENTS

The authors wish to thank S. Kayser for valuable discussions and the anonymous reviewers for their valuable comments and suggestions that helped improve the presentation of the paper.

REFERENCES

- [1] R. L. Rivest and A. Shamir, "How to reuse a "write-once" memory," *Information and Control*, vol. 55, no. 1-3, pp. 1-19, Oct./Nov./Dec. 1982.
- [2] F. Merkkx, "Womcodes constructed with projective geometries," *Traite-ment du Signal*, vol. 1, no. 2-2, pp. 227-231, 1984.
- [3] A. Fiat and A. Shamir, "Generalized "write-once" memories," *IEEE Trans. Inf. Theory*, vol. 30, no. 3, pp. 470-480, May 1984.
- [4] C. Heegard, "On the capacity of permanent memory," *IEEE Trans. Inf. Theory*, vol. 31, no. 1, pp. 34-42, Jan. 1985.
- [5] G. D. Cohen, P. Godlewski, and F. Merkkx, "Linear binary code for write-once memories," *IEEE Trans. Inf. Theory*, vol. 32, no. 5, pp. 697-700, Sep. 1986.
- [6] G. Zémor and G. D. Cohen, "Error-correcting WOM-codes," *IEEE Trans. Inf. Theory*, vol. 37, no. 3, pp. 730-734, May 1991.
- [7] F.-W. Fu and A. J. H. Vinck, "On the capacity of generalized write-once memory with state transitions described by an arbitrary directed acyclic graph," *IEEE Trans. Inf. Theory*, vol. 45, no. 1, pp. 308-313, Jan. 1999.
- [8] E. Yaakobi, P. H. Siegel, A. Vardy, and J. K. Wolf, "Multiple error-correcting WOM-codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Austin, TX, Jun. 2010, pp. 1933-1937.
- [9] E. Yaakobi, S. Kayser, P. H. Siegel, A. Vardy, and J. K. Wolf, "Efficient two-write WOM-codes," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Dublin, Ireland, Aug./Sep. 2010.

- [10] S. Kayser, E. Yaakobi, P. H. Siegel, A. Vardy, and J. K. Wolf, "Multiple-write WOM-codes," in *Proc. 48th Annual Allerton Conf. Commun., Control, and Computing*, Monticello, IL, Sep./Oct. 2010, pp. 1062–1068.
- [11] R. Gabrys, E. Yaakobi, L. Dolecek, P. H. Siegel, A. Vardy, and J. K. Wolf, "Non-binary WOM-codes for multilevel flash memories," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Paraty, Brazil, Oct. 2011, pp. 40–44.
- [12] E. Yaakobi, S. Kayser, P. H. Siegel, A. Vardy, and J. K. Wolf, "Codes for write-once memories," *IEEE Trans. Inf. Theory*, vol. 58, no. 9, pp. 5985–5999, Sep. 2012.
- [13] A. Shpilka, "New constructions of WOM codes using the Wozencraft ensemble," *IEEE Trans. Inf. Theory*, vol. 59, no. 7, pp. 4520–4529, Jul. 2013.
- [14] A. Bhatia, M. Qin, A. R. Iyengar, B. M. Kurkoski, and P. H. Siegel, "Lattice-based WOM codes for multilevel flash memories," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 933–945, May 2014.
- [15] A. Bhatia, A. R. Iyengar, and P. H. Siegel, "Multilevel 2-cell t -write codes," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Lausanne, Switzerland, Sep. 2012, pp. 247–251.
- [16] A. Jiang, "On the generalization of error-correcting WOM codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Nice, France, Jun. 2007, pp. 1391–1395.
- [17] A. Jiang and J. Bruck, "Joint coding for flash memory storage," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Toronto, ON, Canada, Jul. 2008, pp. 1741–1745.
- [18] H. MahdaviFar, P. H. Siegel, A. Vardy, J. K. Wolf, and E. Yaakobi, "A nearly optimal construction of flash codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Seoul, Korea, Jun./Jul. 2009, pp. 1239–1243.
- [19] S. Kayser, private communication.
- [20] Q. Huang, S. Lin, and K. A. S. Abdel-Ghaffar, "Error-correcting codes for flash coding," *IEEE Trans. Inf. Theory*, vol. 57, no. 9, pp. 6097–6108, Sep. 2011.