Switch Codes: Codes for Fully Parallel Reconstruction

Zhiying Wang, *Member, IEEE*, Han Mao Kiah, Yuval Cassuto, *Senior Member, IEEE*, and Jehoshua Bruck, *Fellow, IEEE*

***Abstract*— Network switches and routers scale in rate by distributing the packet read/write operations across multiple memory banks. Rate scaling is achieved so long as sufficiently many packets can be written and read in parallel. However, due to the non-determinism of the read process, parallel pending read requests may contend on memory banks, and thus significantly lower the switching rate. In this paper we provide a constructive study of codes that guarantee fully parallel data reconstruction without contention. We call these codes "switch codes", and construct three optimal switch-code families with different parameters. All the constructions use only simple XOR-based encoding and decoding operations, an important advantage when operated in ultra-high speeds. Switch codes achieve their good performance by spanning simultaneous disjoint local-decoding sets for all their information symbols. Switch codes may be regarded as an extreme version of the previously studied batch codes, where the switch version requires parallel reconstruction of all the information symbols.**

***Index Terms*— Distributed-storage codes, network switches, batch codes, combinatorial designs.**

I. Introduction

Consider a shared memory system required to serve write and read requests at a certain rate. In the write path, k fixed-size packets arrive each time unit, and need to be stored in the memory system. In the read path, each time unit the memory system needs to output a requested set of k previously written packets. To meet these requirements, the system uses n banks of physical memory, where each memory bank works at a rate of one packet write and one packet read each time unit. The design objective of the system is to minimize the number of banks n that are needed to fulfill the above mentioned read/write specifications. Figure 1 gives a pictorial description of such a memory system.

Zhiying Wang is with the Center for Pervasive Communications and Computing, University of California Irvine, Irvine CA, USA (email: zhiying@uci.edu).

Han Mao Kiah is with the School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore (email: hmkiah@ntu.edu.sg).

Yuval Cassuto is with the Department of Electrical Engineering, Technion – Israel Institute of Technology, Haifa Israel (email: ycassuto@ee.technion.ac.il).

Jehoshua Bruck is with the Department of Electrical Engineering, California Institute of Technology, Pasadena CA USA (email: bruck@caltech.edu).

The work of Z. Wang was partially supported by NSF grant CCF-1566587. Part of this work was done when H. M. Kiah was a postdoctoral research associate at the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign. The work for H. M. Kiah has been supported in part by Singapore Ministry of Education (MOE) Tier 1 grant 2016-T1-001-156. The work of Y. Cassuto was supported in part by the Israel Science Foundation joint ISF-UGC program. The authors wish to thank Omer Shaked for his contributions to the development of the switch-code model. Part of the results in the paper were presented at the 2013 and 2015 IEEE International Symposia on Information Theory held in Istanbul and Hong Kong, respectively.

The main application for such a memory system is within *network switches* (and similarly routers), wherein the memory system is used as a *switching fabric* writing packets upon their inbound arrival, and later reading them for their outbound transmission. Two features of the abstract system model are especially fitting for switching applications: 1) the symmetry between read and write rates – each at k packets per time unit, and 2) flexibility to choose the k read packets from the currently stored packets. The first feature is required for flow conservation in the switch, and the second provides flexibility to accommodate priorities, congestion, blocking, and other factors affecting the packet read schedule.

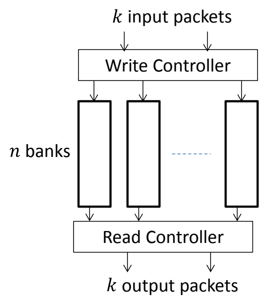

Fig. 1. A memory system supporting write and read of k packets per time unit, and employing n physical memory banks.

The main challenge faced by the switch memory system is *contention* between the requested packets on the bandwidth of the memory banks. Simply put: if a bank is used to output one of its packets in a time unit, then it cannot output another packet in the same time unit. For example, consider the simple case of $k = 2$ packets used with $n = 2$ banks. This scenario is depicted in Figure 2. Packets are marked by letters progressing lexicographically with arrival time. Packets arrived in the same time unit are called a *generation*. Each generation contains $k = 2$ packets and are stored in the $n = 2$ banks. So for the write path $n = 2$ banks are sufficient. However, it is clear that in the read path the system of Figure 2 does not work, because requests like (A, E) or (D, F) cannot be served at a single time unit. From the

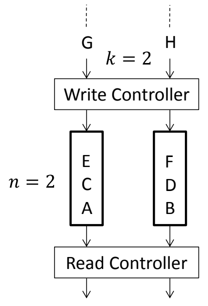

Fig. 2. Packet placement in a memory system for $k = 2$ input/output packets, with $n = 2$ memory banks. 2 banks are *not* sufficient to allow reading any 2 packets without contention.

example of Figure 2 it is clear that supporting arbitrary packet requests in the read path necessitates using $n > k$ banks, while introducing redundancy to the write path in the form of writing more than k packets each time unit.

Minimizing the redundancy to meet the requirements for the system in Figure 1 is best done with a precise coding model. Under such a model, the packets written to memory are computed by some encoder function, in a way that the packets requested for read can be computed by some decoder accessing at most one packet per bank. Such codes are the topic of this paper, and we refer to them as *switch codes*. In the coding formulation of the switch memory problem we use the terms *packet* and *symbol* interchangeably. In that respect, writing a packet to the i-th memory bank is synonymous to placing a symbol at the i-th coordinate of the codeword. To specify switch codes, it is useful to first discuss the desired features we seek in our codes.

1) **Redundancy measure**. We choose the most natural measure of redundancy, which is $n - k$.
2) **Domain of encoding function**. Each packet written to the memory is obtained in general as the output of some encoding function. In the paper we restrict ourselves to encoding functions that are *intra-generation* and *fixed*. Intra-generation means that only the k incoming packets are used as input arguments to the encoding function. Fixed means that we use the same encoding function for every generation of k packets. We note that more general encoding functions may in addition use the packets already stored in the system, and may vary the encoding according to the instantaneous state of the system.
3) **Request flexibility**. Specification of which sets of k packets must be decoded in a single time unit. In the paper we consider two such specifications: one is the strongest model guaranteeing *any* k of the stored packets; another is a model we call the *one-burst* request model, which finds natural motivation when the memory system is used in a network switch. More details on the one-burst model are given in Section IV.
4) **Complexity metrics**. The high packet read/write rates considerably limit the complexity afforded by the encoder and decoder. In the paper we seek codes that minimize three types of complexity: 1) *encoding degree*, the number of input packets used to compute a written packet, 2) *decoding degree*, the number of stored packets used to reconstruct a requested packet, and 3) *arithmetic complexity*, where we restrict the codes to perform only simple binary exclusive OR (XOR) operations.

The switch codes we construct in the rest of the paper are ones embodying the above features. In particular, consider the case where k requested symbols are from k different generations. Since the encoder is intra-generation, decoding one requested symbol involves accessing a set of helper symbols from the corresponding generation. By the constraint of accessing at most one symbol per memory bank, the helper symbols for the k requested symbols should all be disjoint. It is clear that for fixed encoding functions, if requests from k generations can be decoded by disjoint helper symbols, so can those from less than k generations. Therefore, we define a switch code as a code where the requested k symbols are decoded from disjoint sets of codeword symbols.

A. Known work and the contributions of the paper

A previously studied coding model called *batch codes* is especially useful for this paper's interest in switch codes. First proposed by Ishai et. al [1], batch codes seek low-redundancy storage that allows flexible simultaneous data reconstruction. The batch-code model is very broad, so we focus here on the particular cases that can give switch codes. Intra-generation switch codes for the any-k read flexibility model can be readily obtained from a subclass of batch codes called *primitive multiset batch codes*, PMBC for short. The *primitive* feature means that the write path is also limited to one packet per bank (like the read), and the *multiset* feature allows requesting the same symbol index from different write generations. It is clear that both of these features are necessary to obtain any-k switch codes. In addition, batch codes specify distinct numbers of packets for the write and read paths, that is k_{in} packets to write and k_{out} packets to read each time unit[1]. So altogether a PMBC with $k_{\text{in}} = k_{\text{out}} = k$ gives a switch code for the any-k read flexibility model. As a result, our construction in the first part of the paper in Section III is indeed a PMBC with $k_{\text{in}} = k_{\text{out}} = k$.

The principal contribution of Section III is in fact *not* the construction itself, which is a simple concatenation of the well-known *simplex* code. Rather, our contribution is an explicit *deterministic* decoding algorithm that achieves guaranteed success with strictly optimal (not just in the limit) redundancy given the average encoding degree.

[1]In [1] k_{in} is denoted n and k_{out} is denoted k.

Moreover, the algorithm has decoding degree only 2. Previously a randomized decoder was only known[2] [1] to achieve the same decoding capability in the limit and its success is promised only with high probability. Two more PMBCs are proposed by Ishai et. al in [1]: a *non-binary Reed Muller code* and one called *subcube code*. These constructions, however, suffer from much higher complexity: high encoding and decoding degrees, and high arithmetic complexity in the case of the non-binary Reed Muller code. A complete comparison is provided in the closing of Section III. More recent work on PMBCs includes [2], [3], which offer new constructions but not for the case $k_{\text{in}} = k_{\text{out}}$.

In Section IV we move away from known PMBCs to construct switch codes with *constant encoding and decoding degrees*. That is, for any k we restrict each coded packet to be computed from at most 3 incoming packets, and each output packet to be computed from at most 3 stored packets. These restrictions come from the practical difficulty to XOR together many packets at the extremely high read/write rates found in network switches. We choose the constant to be 3 because the case of degree 2 has a trivial optimal solution of storing the XORs between every pair of the k incoming packets. Low encoding degrees imply a lower bound on redundancy, so even though our codes in Section IV have optimal redundancy for their encoding degrees, the redundancy is high compared to our codes in Section III. We note here that the codes in Section IV are constructed for a weaker read-flexibility model we call *one-burst requests*. One-burst requests are requests for k arbitrary packets with the only restriction that at most one packet index is requested from multiple generations. As constant-degree all-k switch codes seem hard to come by, one-burst codes provide similar utility in natural realizations of the memory system in switches. In normal use, the read requests to the switch memory are queued, and then one-burst requests allow shortening the longest queue at each time unit, which minimizes the worst-case read delay. The model of one-burst requests is further justified in Section IV using a simple quantitative model of a queued switch memory.

In a broader sense switch codes are related to *locally decodable codes* (e.g., [4]), because the need to simultaneously reconstruct k symbols from disjoint code indices generally implies that each information symbol can be recovered locally from few code symbols. Locally decodable codes in general do not qualify as switch codes, but they go in this direction if they satisfy the *smoothness property*: for any information symbol, all the local queries used to decode that symbol cover the n codeword symbols uniformly. With that property local recoverability can be extended from individual symbols to a sequence of multiple requested symbols. A probabilistic decoder only needs to leave sufficiently many codeword symbols to recover the latter information symbols, and thanks to uniformity, success with high probability can be proved without care to which symbols are chosen for each recovered symbol [1]. However, the smoothness property is not tight enough to compete with the parameters we achieve here. Moreover, probabilistic decoding is considered for such codes, while for switch codes deterministic decoding is required.

Finally switch codes are related to local codes with *multiple repair alternatives* (e.g. [5]–[8]), which were proposed for distributed storage, but known codes for that model have a big gap between the number of input symbols k_{in} and the number of requested symbols k_{out}.

[2]The algorithm in [1] is given for a code more general than simplex code, called *subset code*.

II. Definitions and Notations

In the rest of the paper, we use $[i]$ to denote the set $\{1, 2, \dots, i\}$ for $i \in \mathbb{N}^+$, and $[i, j]$ to denote the set $\{i, i+1, \dots, j\}$ for $i \leqslant j \in \mathbb{Z}$. We use boldface to represent a vector. For a vector $\mathbf{x} = (x_0, \dots, x_{n-1})$, we represent by $w(\mathbf{x}) = \sum_{i=0}^{n-1} x_i$. For a set S, its cardinality is denoted by $|S|$. For a vector $\mathbf{x} = (x_0, \dots, x_{n-1})$ and a subset $S = \{s_1, \dots, s_{|S|}\} \subseteq [0, n-1]$, where $0 \leqslant s_1 < \cdots < s_{|S|} \leqslant n-1$, we denote by $\mathbf{x}_S = (x_{s_1}, \dots, x_{s_{|S|}})$ the vector of elements with coordinates in S. We use log to denote logarithm of base 2.

This paper's definition of a switch code strongly builds on a previously defined object called primitive multiset batch code (PMBC) [1], which we now define using terminologies related to the switch code problem. Informally, an (n, k, R) PMBC over the alphabet $\mathcal{X}$ encodes an information vector $\mathbf{u} = (u_0, \dots, u_{k-1})$ of length k into a codeword vector $\mathbf{x} = (x_0, \dots, x_{n-1})$ of length n. Let $\mathbf{L} = (l_0, \dots, l_{k-1})$ be the *request vector*, where the i-th information symbol is requested l_i times, $i \in [0, k-1]$. Denote by $w(\mathbf{L}) = \sum_{i=0}^{k-1} l_i$ the *request weight*. Note that the ordering of the k requested symbols does not matter, hence we assume wlog that the first l_0 requested symbols are u_0, the next l_1 requested symbols are u_1, and so on. For any request of weight $w(\mathbf{L}) = R$, there exist disjoint sets $S_1, \dots, S_R \subseteq [0, n-1]$, such that u_i can be recovered from the codeword symbols indexed by S_j, namely, $\mathbf{x}_{S_j}$, for any $i \in [0, k-1]$, $j \in [\sum_{t=0}^{i-1} l_t + 1, \sum_{t=0}^{i} l_t]$. More formally, a PMBC can be defined as follows.

Definition 1 (PMBC) *An (n, k, R) PMBC on the alphabet $\mathcal{X}$ consists of*

1) *an encoding function*
$$\varphi : \mathcal{X}^k \to \mathcal{X}^n,$$
2) *a decoding set function*
$$\xi : \mathcal{L} \to \mathcal{S},$$
where $\mathcal{L} = \{(l_0, \dots, l_{k-1}) : \sum_{i=0}^{k-1} l_i = R\}$ is the set of requests of weight R, and $\mathcal{S} = \{(S_1, S_2, \dots, S_R) : S_j \subseteq [0, n-1], S_j \cap S_{j'} =$

$\emptyset$, for all $1 \leqslant j \neq j' \leqslant R\}$ is the collection of R disjoint sets, and

3) *decoding recovery functions*

$$\psi_{S,i} : \mathcal{X}^{|S|} \to \mathcal{X}.$$

The functions satisfy the following: for all inputs $\mathbf{u} \in \mathcal{X}^k$ and request vectors $\mathbf{L} = (l_0, \dots, l_{k-1}) \in \mathcal{L}$, if $\varphi(\mathbf{u}) = \mathbf{x}$ and $\xi(\mathbf{L}) = (S_1, \dots, S_R)$, then for all $i \in [0, k-1], l_i \neq 0$, and all $j \in [\sum_{t=0}^{i-1} l_t + 1, \sum_{t=0}^{i} l_t]$,

$$\psi_{S_j,i}(\mathbf{x}_{S_j}) = u_i.$$

We call k the *input size* or the *code dimension*, and R the *request weight*. Now we define switch codes as the following special case of PMBC.

Definition 2 (switch code) *An (n, k) switch code is an $(n, k, R = k)$ PMBC.*

Note that a PMBC with $R = k$ is sufficient to guarantee read success for the abstract model of Figure 1. This is because any k packets previously stored in the memory system can be specified as k (*information symbol*, *generation*) pairs, and the disjointedness of the decoding sets allows recovering symbols from different generations independently without contention. More specifically, when multiple requested packets have the same *information symbol* index i but different *generation* index $g_1, g_2, \dots, g_{l_i}$, they correspond to l_i times request for u_i in PMBC. For any $1 \leqslant x \leqslant l_i$, one can recover the i-th symbol in generation g_x using symbols in S_j from the same generation, where j can be set as $j = \sum_{t=0}^{i-1} l_t + x$. As the sets $S_1, \dots, S_k$ are disjoint, no read contention occurs.

Even though there may be other codes that also achieve read success for the model in Figure 1 (e.g., one may code across generations), for the scope of this paper, we only consider PMBC as our switch codes. Other coding schemes are left as an interesting further research direction.

In addition to the restriction $R = k$, the switch codes we present here are defined over binary alphabets, and all encoding and decoding operations are simple bit-wise XOR operations. For simplicity we denote the binary XOR operation by "$+$". Accordingly, we later refer to the information and codeword symbols as *bits*, while having in mind that the constructions can be trivially extended to packets with an arbitrary number of bits.

For a linear code, if a codeword symbol is a linear combination of d information symbols, then its *encoding degree* is d. A codeword symbol is called *systematic* or a *singleton* if it equals to an information symbol, hence having $d = 1$. For some code and a request, if there exist disjoint sets to recover the requested symbols, then we say there is a *solution* to the request, or the request is *solvable*. The set of codeword symbols indexed by S_i is called *helpers* or a *helper set* for the requested information symbol. The largest helper set among the requested symbols is called the *decoding degree*.

Example 3 *Consider the simple $(n = 3, k = 2)$ switch code defined by the encoding function $(x_0, x_1, x_2) = (u_0, u_1, u_0 + u_1)$. For the request vector $(l_0, l_1) = (1, 1)$ the decoding sets are $S_1 = \{0\}$ and $S_2 = \{1\}$. The decoding functions are $u_0 = x_0$, $u_1 = x_1$. The last symbol x_2 is not used for this decoding instance. Consider a second request vector $(l_0, l_1) = (2, 0)$, and then the decoding sets are $S_1 = \{0\}$ and $S_2 = \{1, 2\}$. The decoding functions are $u_0 = x_0$, $u_0 = x_1 + x_2$.*

A. Redundancy lower bound given encoding degree

Consider a linear switch code. Suppose the encoding degree of codeword symbol indexed i is denoted d_i. The average encoding degree is $\bar{d} = \frac{1}{n}\sum_{i=0}^{n-1} d_i$. To reduce the implementation complexity of the codes, it is desirable that the code symbols have low encoding degrees. However, this reduction of complexity comes with an inherent cost of higher code redundancy. The following proposition gives a precise formulation of this fact.

Proposition 4 *An (n, k) switch code with average encoding degree $\bar{d}$ satisfies*

$$n \geqslant \frac{k^2}{\bar{d}}.$$

Proof: Consider a request where an information symbol is requested k times. Since we have k disjoint solutions for this information symbol, it has to appear in k codeword symbols. Summing over all k different information symbols results in a total of k^2 appearances. Therefore, the sum of degrees satisfies $\sum_{i=0}^{n-1} d_i \geqslant k^2$. And the statement holds for the average degree. ■

Our focus in the paper is linear switch that are systematic, that is, the k information symbols appear as codeword symbols. Suppose that the non-systematic symbols have a constant encoding degree d. Therefore, we have k systematic symbols with degree 1, and $n - k$ non-systematic symbols with degree d. Then we have the following redundancy bound.

Corollary 5 *A systematic linear switch code with constant encoding degree d for non-systematic symbols satisfies*

$$n - k \geqslant \frac{k(k-1)}{d}.$$

The codes we present in the sequel are shown to be optimal with respect to the bound of Proposition 4 or Corollary 5.

III. Optimal All-k Read Switch Codes

In this section we construct the first family of switch codes with optimal redundancy given the average encoding degree, which is $O(\log k)$. The code is binary, and the decoding degree is $r = 2$. The key component in the construction is an optimal guaranteed-decoding PMBC constructed from the well-known simplex code [9], and concatenated to obtain a switch code satisfying $R = k$. The (optimal) codeword length is $n = O(k^2/\log k)$.

We first construct binary PMBC from simplex codes, and get codes with length $2^k - 1$, dimension k, and decoding degree 2. We then show that this construction solves arbitrary requests of weight 2^{k-1}. From this construction we obtain an $R = k$ switch code through simple concatenation. Lastly, we prove the optimality of our construction and compare to previously known ones.

An (N, K) *simplex code* is constructed as follows. For every non-empty subset of $[0, K-1]$, form a bit in the codeword that is the XOR of the elements in the subset. Hence, a simplex code of dimension K, $K \geqslant 1$, has codeword length $N = 2^K - 1$.

Construction 0 (PMBC from simplex code) *Fix R such that $\log R$ is an integer. A $(2R-1, 1+\log R, R)$ PMBC is obtained from the $(N = 2R-1, K = 1+\log R)$ simplex code.*

Before proving that Construction 0 indeed gives a PMBC with request weight R, we note that in itself this construction is not very useful because the code length is exponential in the input size. What does turn out to be useful is the following concatenation of Construction 0.

Construction 1 (switch code from simplex concatenation) *Let $K \geqslant 2$. Define $m = \lfloor 2^{K-1}/K \rfloor$ and $k = mK$. Consider m groups of K information symbols. A $(m(2^K - 1), k)$ switch code can obtained by concatenating m codewords of the $(2^K - 1, K)$ simplex code.*

Suppose the generator matrix of the $(N = 2^K - 1, K)$ simplex code is given by (I_K, G), where I_K is the $K \times K$ identity matrix, and G is a $K \times (N-K)$ matrix. Then the code given by Construction 1 has generator matrix

$$\begin{pmatrix} I_K & 0 & \cdots & 0 & G & 0 & \cdots & 0 \\ 0 & I_K & \cdots & 0 & 0 & G & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & I_K & 0 & 0 & \cdots & G \end{pmatrix}.$$

We next show that the (N, K) simplex code in Construction 0 solves an arbitrary request of weight $R = 2^{K-1} = \frac{N+1}{2}$ and has decoding degree 2. For example, the code (u_0, u_1, u_0+u_1) in Example 3 is a simplex code with $K = 2$, and any request of weight $2^{K-1} = 2$ can be solved.

In the following, we use the non-empty subset $T \subseteq [0, K-1]$ to represent the corresponding bit in the

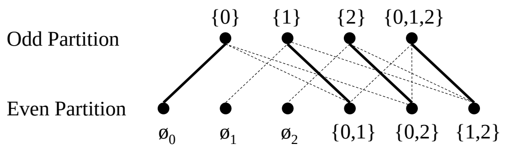

Fig. 3. A bipartite graph on $K = 3$ input bits. Every edge corresponds to a possible helper pair. The set of solid-line edges is a solution to the request $\mathbf{L} = (4, 0, 0)$, or four times the input bit $\{0\}$.

codeword of the simplex code. (The sets here pertaining to the simplex code should not be confused with the decoding sets in Definition 1.) The systematic bits are the sets of size one, namely, $\{j\}$ for any $j \in [0, K-1]$. By abuse of notation, we write "+" to denote the XOR of two bits, or equivalently, the symmetric difference of two sets. It is clear that any codeword bit T can be recovered from the XOR of the two bits $T + T'$ and T', for arbitrary T'. Therefore, the simplex code can recover any code bit from up to 2 other code bits. In particular, any information bit can be computed from 2 codeword bits.

We observe from the following graphical view that 2^{K-1} is an upper bound on the request weight for the simplex code. Consider a graph where every non-empty subset T is a vertex, and every edge (T, T') corresponds to a solution to an information bit, namely, $|T + T'| = 1$. Also add to this graph K dummy vertices corresponding to the empty set, denoted by ϕ_i, $i \in [0, K-1]$, along with K edges $(\{i\}, \phi_i)$. See Figure 3 for an example. First, notice that this graph represents all possible solutions of information bits with decoding degree no more than 2. Next, notice that this is a bipartite graph where the partition of the vertices is determined by the parity of $|T|$. The even partition is of size $2^{K-1} + K - 1$ (including K copies of the empty set), while the odd partition is of size 2^{K-1}. A disjoint solution for some request vector can be viewed as a matching in the graph, and apparently the size of the matching, or the request weight, cannot exceed 2^{K-1}.

The following definitions are useful to describe the decoder of the simplex code.

Definition 6 *A request vector $\mathbf{L}$ on K input bits is said to be* short *if its weight satisfies*

$$w(\mathbf{L}) \leqslant f(K) \triangleq \frac{K}{K+1} 2^{K-1}.$$

Definition 7 *A solution to a request vector is said to be* type I *if singletons are not used in the solution, and the decoding degree is 2.*

For example, let $K = 4$, and consider a short request $\mathbf{L} = (1, 1, 1, 1)$ of weight no more than $f(K) = 32/5$. Namely, every information bit is requested once. It can be solved by the helper pairs $(\{0, 1, 2\}, \{1, 2\})$, $(\{1, 2, 3\}, \{2, 3\})$, $(\{2, 3, 0\}, \{3, 0\})$, $(\{3, 0, 1\}, \{0, 1\})$, which is a type I solution, since no singletons are used.

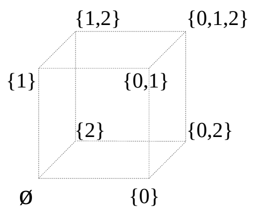

Fig. 4. Partitions on $K = 3$ input bits labeled $I = \{0, 1, 2\}$. Every two parallel faces form a partition. For example, the face on the right is $A_0 = \{\{0\}, \{0, 1\}, \{0, 2\}, \{0, 1, 2\}\}$ containing element "0", and the face on the left is $\overline{A_0}$. One can see that any edge connecting the left and the right faces corresponds to a solution to the bit $\{0\}$. Moreover, since the pair $\{\emptyset, \{1\}\}$ solves the bit $\{1\}$, we have that the pair $\{\delta_0^{-1}(\emptyset), \delta_0^{-1}(\{1\})\} = \{\{0\}, \{0, 1\}\}$ also solves the same bit.

We later show in Lemma 12 that if $K \geqslant 7$, then a short request has a type I solution. One important idea in our decoder is that, we decompose a request vector as the sum of short requests and the remaining request. We solve the short requests by non-singletons, and solve the remaining possibly with singletons. For $K \leqslant 7$, we then employed a computer search to determine the solutions for this finite set of requests.

Definition 8 *Let I be a set of integers of size K. For each nonempty subset X of I, there is a corresponding codeword bit in the simplex code on K inputs. For any $i \in I$, we partition the collection of all subsets of I into two classes:*

$$A_i = \{T \subseteq I : i \in T\}, \text{ and } \overline{A_i} = \{T \subseteq I : i \notin T\}. \tag{1}$$

Also define a mapping between them:

$$\delta_i : A_i \to \overline{A_i}, T \mapsto T \setminus \{i\}.$$

This mapping can be shown to be 1-1.

Figure 4 shows an example of the partitions on $K = 3$ inputs. The above partition forms a recursive structure of the codeword bits. Furthermore, any solution to the information bit $\{i\}$ with decoding degree 2 must be a pair

$$\{T, \delta_i(T)\}, \text{ for some } T \in A_i. \tag{2}$$

Besides, if the information bit $\{j\}$, $j \neq i$, can be solved by a pair $\{U, T\}$ where $U, T \in \overline{A_i}$, then it can also be solved by the pair

$$\{\delta_i^{-1}(U), \delta_i^{-1}(T)\} = \{U \cup \{i\}, T \cup \{i\}\}.$$

We outline how to solve a request of weight 2^{K-1} in a simplex code with an example.

Example 9 *Consider a request vector*

$$\mathbf{L} = (42, 39, 36, 33, 31, 28, 27, 10, 10)$$

for $K = 9, w(\mathbf{L}) = 2^{K-1} = 256$. Crucial to the proof is that the entries in the request vector are in non-increasing order. We write $\mathbf{L}$ as $\mathbf{L} = \mathbf{L}_1 + 2\mathbf{L}_2 + \mathbf{L}_3$ with

$$\begin{aligned} \mathbf{L}_1 &= (42, 0, 0, 0, 0, 0, 0, 0, 0), \\ \mathbf{L}_2 &= (0, 19, 18, 16, 15, 14, 13, 5, 5), \\ \mathbf{L}_3 &= (0, 1, 0, 1, 1, 0, 1, 0, 0). \end{aligned}$$

Lemma 12 below shows that $\mathbf{L}_2$ can be solved by type I solution on $K - 1 = 8$ inputs, which we explain in the next paragraph. This solution can then be duplicated in the two partitions of A_0 and $\overline{A_0}$, respectively. Thus we can solve $2\mathbf{L}_2$, while we use the singletons to solve $\mathbf{L}_3$. Finally, we demonstrate in Theorem 13 that there are sufficiently many pairs as in (2) still available for the information bit $\{0\}$ in the request vector $\mathbf{L}_1$.

To show that $\mathbf{L}_2$ has a type I solution, we view $\mathbf{L}_2$ as a short request on $K - 1$ inputs and write $\mathbf{L}_2 = (19, 18, 16, 15, 14, 13, 5, 5)$. Again this is in non-increasing order. Now consider

$$\mathbf{L}_2' = (19, 18, 16, 16, 14, 14, 6, 6),$$

and write $\mathbf{L}_2' = \mathbf{L}_4 + 2\mathbf{L}_5$, where

$$\begin{aligned} \mathbf{L}_4 &= (19, 0, 0, 0, 0, 0, 0, 0), \\ \mathbf{L}_5 &= (0, 9, 8, 8, 7, 7, 3, 3). \end{aligned}$$

Notice that if $\mathbf{L}_2'$ is type-I solvable, so is $\mathbf{L}_2$. We show $\mathbf{L}_5$ is type-I solvable using the induction base case in Lemma 12 on $K - 2 = 7$ inputs. In general if $K - 2 > 7$, $\mathbf{L}_5$ can be shown to be a short request on $K - 2$ inputs, and thus has a type I solution using the induction hypothesis in Lemma 12. Similar to the argument in the previous paragraph, consider all codeword bits on the $K - 1$ inputs labeled $I = \{1, 2, \ldots, K - 1\}$. The solution to $\mathbf{L}_5$ can be duplicated in the two partitions A_1 and $\overline{A_1}$, respectively. Finally, we solve $\mathbf{L}_4$ using the remaining pairs as in (2), which is proved in Lemma 10.

Our first lemma is a recursive construction of type I solutions.

Lemma 10 *Let $\mathbf{L} = (l_0, l_1, \ldots, l_{K-1})$ be a request. Set $\mathbf{L}' = (0, \lceil l_1/2 \rceil, \ldots, \lceil l_{K-1}/2 \rceil)$. If there is a type I solution to $\mathbf{L}'$ in $\overline{A_0}$, and $2w(\mathbf{L}') \leqslant 2^{K-1} - l_0 - K$, then there is a type I solution to $\mathbf{L}$.*

Proof: $\mathbf{L}'$ has a type I solution. For every helper pair $\{U, T\}$ in this solution on $K - 1$ inputs, we generate two helper pairs on K inputs labeled $[0, K - 1]$, that solve the same information bit: the first pair is

$$\{U \cup \{0\}, T \cup \{0\}\}, \tag{3}$$

and both helpers belong to A_0; the second is $\{U, T\}$ and both helpers belong to $\overline{A_0}$. Since U, T belong to a type I solution, they are not singletons. Then $\{U \cup \{0\}, T \cup \{0\}\}$ are not singletons either. Moreover, all the helper pairs are disjoint. So we have a type I solution to the request $2\mathbf{L}'$.

Let B, C be the set of helpers for $2\mathbf{L}'$ generated in $A_0, \overline{A_0}$, respectively, which are both of size $2w(\mathbf{L}')$. Notice that the function δ_0 defines a 1-1 mapping from B to

C, and accordingly from $A_0\backslash B$ to $\overline{A_0}\backslash C$. In other words, if we pick any unused element $T \in A_0\backslash B$, then we have $T\backslash\{0\} \in \overline{A_0}\backslash C$, and they can form the helper pair for $\{0\}$. As a result, there are $|A_0\backslash B| = 2^{K-1} - 2w(\mathbf{L}')$ remaining ways to solve $\{0\}$, and K of them involve singletons in either $A_0\backslash B$ or $\overline{A_0}\backslash C$. Hence the number of helper pairs for $\{0\}$ that do not use singletons is

$$|A_0\backslash B| - K = 2^{K-1} - 2w(\mathbf{L}') - K \geqslant l_0, \quad (4)$$

where the last inequality follows from the lemma assumption. So we have a type I solution to $\mathbf{L}$. ∎

The following is a lemma on a small input size, and forms the base case of our proof.

Lemma 11 *Consider a simplex code with K input bits.*

(i) *There is a type I solution to any request of weight $2^{K-1} - K$, for all $3 \leqslant K \leqslant 7$.*
(ii) *There is a solution of decoding degree 2 to any request of weight 2^{K-1}, for all $2 \leqslant K \leqslant 7$.*

Proof: For any request vector $\mathbf{L} = (l_0, \ldots, l_{K-1})$, assume $l_0 \geqslant l_1 \geqslant \ldots \geqslant l_{K-1}$ without loss of generality. It suffices to demonstrate the claims in (i) and (ii) for requests of weight exactly $2^{K-1} - K$ and 2^{K-1}, respectively.

(i) is proved by using Lemma 10 and computer search (see details in the Appendix). When $K = 3$, it is straightforward to verify that there is a type I solution to any request of weight one.

When $4 \leqslant K \leqslant 7$, we assume that there is a type I solution to any request of weight $2^{K-2} - K + 1$ for a simplex code with $K - 1$ input bits. For a request $\mathbf{L}$ of weight $2^{K-1} - K$, whenever $\mathbf{L}$ and its corresponding $\mathbf{L}'$ satisfy (4), we apply Lemma 10 to obtain a type I solution for $\mathbf{L}$. However, there are cases where (4) is not satisfied. Specifically, via a computer, we found there are 3, 30, 638, and 32715 such requests for $K = 4, 5, 6,$ and 7, respectively. For these requests, we conducted a computer search and the results are available at the second author's website[3].

We prove (ii) by induction. Again, it is straightforward to verify (ii) for $K = 2$.

When $3 \leqslant K \leqslant 7$, we assume (ii) for a simplex code with $K - 1$ input bits. We consider two cases.

If $l_{K-1} \geqslant 1$, let $\mathbf{L}' = (l_0 - 1, \ldots, l_{K-1} - 1)$, and $\mathbf{L}'' = (1, \ldots, 1)$. Notice that $\mathbf{L}'$ can be solved by (i) and $\mathbf{L}''$ can be solved by singletons, and the two sets of helpers for $\mathbf{L}'$ and $\mathbf{L}''$ are disjoint. Since $\mathbf{L} = \mathbf{L}' + \mathbf{L}''$, the request $\mathbf{L}$ is solvable.

When $l_{K-1} = 0$, let $\mathbf{L} = \mathbf{L}' + \mathbf{L}''$ such that $w(\mathbf{L}') = w(\mathbf{L}'') = 2^{K-2}$. By induction hypothesis, we have solutions in $\overline{A_{K-1}}$ for $\mathbf{L}'$ and $\mathbf{L}''$ for a simplex code with $K - 1$ input bits. For any pair $\{U, T\}$ in the solution for $\mathbf{L}''$, we create the helper pair $\{U \cup \{K-1\}, T \cup \{K-1\}\}$ and we see that the latter collection of pairs is a solution for $\mathbf{L}''$. Furthermore, the new helper pairs belong to A_{K-1}, while the helper pairs for $\mathbf{L}'$ belong to $\overline{A_{K-1}}$. Therefore, all helper pairs are pairwise disjoint and we have a solution for $\mathbf{L}$. ∎

Lemma 12 *For a simplex code on K input bits, there is a type I solution to any short request for $K \geqslant 7$.*

Proof: We prove by induction on K. When $K = 7$, this is proven in Lemma 11 (i), by observing that $f(K) < 2^{K-1} - K$ when $K = 7$.

When $K \geqslant 8$, assume that there is a type I solution to any short request for the simplex code on $K - 1$ input symbols. We consider a short request $\mathbf{L} = (l_0, \ldots, l_{K-1})$ with $l_0 \geqslant \ldots \geqslant l_{K-1}$. Partition all codeword bits on K inputs labeled $[0, K-1]$ into two parts: $A_0, \overline{A_0}$ as defined by (1).

Let $\mathbf{L}' = (0, \lceil l_1/2 \rceil, \ldots, \lceil l_{K-1}/2 \rceil)$, and we invoke the recursive construction given by Lemma 10. To this end, we first show that $\mathbf{L}'$ is short and hence, type I-solvable by the induction hypothesis. Indeed, since $l_0 \geqslant 2^{K-1}/K$, we have that

$$\begin{aligned} w(\mathbf{L}') &\leqslant \frac{1}{2}\left(\sum_{i=1}^{K-1} l_i + K - 1\right) \\ &\leqslant \frac{1}{2}\left(\frac{K-1}{K} f(K) + K - 1\right) \\ &\leqslant f(K-1) \text{ for } K \geqslant 7. \end{aligned}$$

Next, we check that (4) holds for $\mathbf{L}$ and $\mathbf{L}'$. We separate into two cases:

When $K \geqslant 9$, observe that

$$\begin{aligned} |A_0\backslash B| - K &= 2^{K-1} - 2w(\mathbf{L}') - K \\ &\geqslant 2^{K-1} - (w(\mathbf{L}) - l_0 + K - 1) - K \\ &\geqslant 2^{K-1} - (f(K) - l_0 + K - 1) - K \\ &\geqslant l_0. \end{aligned}$$

When $K = 8$, the first two inequalities imply $|A_0\backslash B| - K \geqslant l_0 - 0.78$, and due to integrailty of both $|A_0\backslash B| - K$ and l_0, we get the last inequality $|A_0\backslash B| - K \geqslant l_0$.

Therefore, the conditions for Lemma 10 are met and there exists a type I solution to the short request $\mathbf{L}$, completing the induction. ∎

Theorem 13 *Let $\mathbf{L}$ be a request of weight 2^{K-1} for the simplex code on K input bits. Then $\mathbf{L}$ is solvable with decoding degree 2.*

Proof: When $K \leqslant 7$, this is true by Lemma 11(ii).

When $K \geqslant 8$, let $\mathbf{L} = (l_0, \ldots, l_{K-1})$ with $l_0 \geqslant \ldots \geqslant l_{K-1}$. Then rewrite $\mathbf{L}$ as $\mathbf{L} = (l_0, 0, \ldots, 0) + 2\mathbf{L}_2 + \mathbf{L}_3$, where $\mathbf{L}_2 = (0, \lfloor l_1/2 \rfloor, \ldots, \lfloor l_{K-1}/2 \rfloor)$, and $\mathbf{L}_3 = (0, l_1 \bmod 2, \ldots, l_{K-1} \bmod 2)$. We check that $\mathbf{L}_2$ is a

[3]`www.ntu.edu.sg/home/hmkiah/simplexSwitch.html`

short request on $K-1$ inputs, namely $w(\mathbf{L}_2) \leqslant f(K-1)$, and has a type I solution by Lemma 12 for $K \geqslant 8$. Then, with singletons, we can solve $\mathbf{L}_3$. Finally, using a similar argument as in Lemma 10, we have l_0 pairs to solve $\{0\}$ (using possibly some singletons), since

$$|A_0 \backslash B| - w(\mathbf{L}_3) = 2^{K-1} - 2w(\mathbf{L}_2) - w(\mathbf{L}_3) = l_0,$$

where A_0 is the collection of subsets of $[0, K-1]$ containing "0", and B is the set of helpers for $\mathbf{L}_2$ belonging to A_0, defined similar to (3). ■

Having proved Theorem 13, we can conclude that Construction 0 indeed provides a PMBC with the specified parameters.

Remark: The proofs of Theorem 13 and the preceding lemmas provide us with a recursive algorithm to find a solution for an arbitrary request of weight at most 2^{K-1}. The recursion ends at the base case of 7 input bits, and the complexity of the algorithm is linear in K. We also comment that the solutions for the case where $K \leqslant 7$ are stored online, and the compressed file has a size of less than 2MB.

Corollary 14 *Let $K \geqslant 2$. Define $m = \lfloor 2^{K-1}/K \rfloor$ and $k = mK$. Construction 1 gives a $(m(2^K-1) \approx \frac{(2k-1)k}{1+\log k}, mK = k)$ switch code with decoding degree 2.*

Proof: Consider m groups of K information symbols. Consider any request of weight $k = mK \leqslant 2^{K-1}$. If it only contains information bits belonging to one group, then the statement holds by Theorem 13. If it contains information bits from different groups, then for every group we get a request of weight less than $R = 2^{K-1}$, and can solve it by Theorem 13 considering codeword bits from the respective group. ■

Next, we show the optimality of our construction using Proposition 4.

Proposition 15 *When $2^{K-1}/K$ is an integer, Construction 1 is optimal in terms of codeword length with respect to its average encoding degree.*

Proof: For Construction 1, the average degree is also the average degree of the (N, K) simplex code:

$$\bar{d} = \frac{\sum_{i=1}^{K} i\binom{K}{i}}{2^K - 1} = \frac{K2^{K-1}}{2^K-1} = \frac{(1+\log k)k}{2k-1}.$$

Given this value of $\bar{d}$, the upper bound from Proposition 4 is given by

$$n \geqslant \frac{k^2}{(1+\log k)k/(2k-1)} = \frac{(2k-1)k}{1+\log k},$$

establishing the optimality of Construction 1. ■

A. Comparison with known PMBCs

We now provide a comparison between our new construction and known PMBC constructions from [1]. Specifically, we focus on the two classes of binary PMBC: subcube codes and subset codes.

1) Subcube codes [1]: Fix parameters l and t to be positive integers. Let G_l be the $l \times (l+1)$ matrix given by $(I_l, \mathbf{1})$, where I_l is the $l \times l$ identity matrix, and $\mathbf{1}$ is the all-one column vector. In other words, G_l is the generator matrix of a code with a single parity bit.

A *subcube code* with parameters l and t is then the linear code generated by the matrix $G(l,t) \triangleq G_l^{\otimes t}$, where $A^{\otimes t}$ denotes the Kronecker product of t A's. Hence, we check easily that $n = (l+1)^t$, $k = l^t$. Ishai *et. al* [1] demonstrated that the subcube code can solve requests of weight 2^t, and hence is a $((l+1)^t, l^t, 2^t)$ PMBC. To get a switch code with $R = k$ we set $l = 2$, and get a $(3^t, 2^t)$ switch code. The latter has a lower redundancy compared to Construction 1, but suffers from a much higher average encoding degree of $d = (4/3)^t = \Theta(k^{0.415})$, compared to $O(\log k)$ in Construction 1. In addition, unlike Construction 1 which has constant decoding degree of 2, the decoding degree of the subcube code is k. In particular, consider 2^t requests for any given information bit. Then there exists a helper set of size $l^t = k$.

2) Subset codes [1]: The subset code is closely related to the simplex code we use for Construction 1 – in fact the simplex code can be obtained as a special case. Fix parameters w and K to be positive integers with $w < K$. Consider subsets of $[0, K-1]$ and let the information bit x_T correspond to a subset T of size w. Then a subset code with parameters K and w is a code whose codeword bits are indexed by subsets of $[0, K-1]$ with size at most w. Every codeword bit x_S is given by $\sum_{S \subseteq T,\ |T|=w} x_T$. Hence, we have the codeword length $n = \sum_{j=0}^{w} \binom{K}{j}$ and the dimension $k = \binom{K}{w}$. It can be seen that if we set $w = K-1$ we get a code that is isomorphic to the simplex code. Setting $w < K-1$ increases the dimension of the code and reduces the code length, thus resulting in a higher rate while trading off the request weight. Ishai et. al [1] derived a clever randomized algorithm to decode subset codes with high probability when w is a constant fraction of K. But the resulting code parameters do not give code families with $R \approx k$, and the probabilistic asymptotic analysis of the decoder makes it hard to evaluate the solvability for a fixed block-length code.

IV. Constant Encoding Degree Constructions

In the previous section we constructed optimal-redundancy switch codes that solve any-k requests with decoding degree of 2. However, despite the significant improvement over prior work, the average encoding degree of Construction 1 is still logarithmic in k. For the sake of low-complexity implementation in ultra-fast switching environments, we now turn to seek switch-code constructions with constant encoding degree that does not grow with k. Moving to constant degrees comes with the caveat that the redundancy must grow as mandated by Corollary 5. Here we consider the case of degree $d = 3$, which is the first non-trivial case because $d = 1$ is pure replication, and $d = 2$ has a trivial optimal solution of taking the XORs of all pairs of information bits in addition to the systematic bits. Optimal codes with $d = 3$ may give a more reasonable tradeoff between encoding complexity and redundancy. Unfortunately, finding optimal $d = 3$ switch codes for any-k requests turned out rather difficult. Hence the codes we construct in this section address a weaker – but well motivated – request model.

Definition 16 (one-burst request) *A request is called a* one-burst request *if its request vector* $\mathbf{L} = (l_0, \ldots, l_{k-1})$ *has at most one element strictly greater than* 1. *The value of the multiplicity satisfying* $l > 1$ *is called the* burst length, *and all the information symbols* j *such that* $l_j = 1$ *are called* uniques.

One-burst requests are especially important in switching applications. When the information-symbol indices are associated with input ports, one-burst requests can be used to multiply-serve the port that instantaneously has the longest queue of pending requests, thus shortening the worst-case delay of packets. A quantitative justification of the one-burst request model now follows with an illustrative example. Suppose the information indices that are needed for read come from some probability distribution. An information index $i \in \{0, \ldots, k-1\}$ is needed for read ℓ_i times (from different generations), and for simplicity we assume that the ℓ_is are Poisson distributed i.i.d. with a load parameter λ. From these random indices the read requests to the switch code are generated. Note that the total number of needed symbols (sum of all ℓ_i) may be larger than k, in which case the read controller chooses a size-k subset of symbols to request from the switch code, and *queues* the rest for a subsequent request. Our main interest in the following example is to see how large λ must be as a function of the switch-code request model such that k symbols can be read in one time unit.

Define ℓ to be the random variable that represents the number of elements read in one time unit. We want to find λ such that the expectation $E(\ell) = k$. Note that requiring a high λ implies a high queuing load, and long delays of symbol (packet) reads. First, without any switch code, symbol i is read only if $\ell_i \geqslant 1$, for all i. Let $\mathbb{1}_i$ be the indicator random variable for the event $\{\ell_i \geqslant 1\}$. Thus,

$$E(\ell) = E(\sum_{i=0}^{k-1} \mathbb{1}_i) = kE(\mathbb{1}_i) = k(1 - e^{-\lambda}),$$

which is equal to k only when $\lambda \to \infty$. Second, with an any-k switch code, the number of read elements is $\sum_{i=0}^{k-1} \ell_i$. Thus, $E(\ell) = k$ if $\lambda = 1$. In between, with a one-burst switch code, the number of read elements is the maximal ℓ_i plus all other $\mathbb{1}_j$. We take $k = 10, 20, 30, 100, 1000$ as examples. From order statistics [10], $E(\ell) = k$ corresponds to $\lambda \approx 1.41, 1.73, 1.95, 2.71, 4.45$, respectively. Compared to the no coding case where $\lambda \to \infty$, one-burst codes provide a much more graceful intermediate solution. In addition to this expected analysis, the ability to serve the longest queue with a burst request effectively trims all significant deviations from short balanced queue lengths.

The switch codes of this section are constructed to guarantee decoding of one-burst requests. We note that the redundancy lower bound of Corollary 5 applies even for the weaker one-burst model. This can be seen from the fact that the particular request used in the proof of Proposition 4 and hence Corollary 5 is a one-bust request with burst length k and no uniques. For $d = 3$, the redundancy should thus be $n - k \geqslant k(k-1)/3$. In fact the constructions in this section matches this lower bound, and hence are optimal given the encoding degree.

A. Framework from block designs

A natural tool to construct switch codes with constant encoding degree is *combinatorial block designs*. A block design is a set of *elements* together with a family of subsets or *blocks* whose members satisfy certain properties. We pursue this direction here with two constructions in the next two sub-sections. It is important to note that one can not reduce the switch-code construction problem to finding block designs in existing families. The problem is that the properties of the block designs available in the literature are not sufficient to get the required solvability for the switch code. Instead, the first of our constructions will derive a sufficiently strong block design from scratch, and the second will work a modification of a specific block design family to get another one that works. We next define the terminology and notation that will be useful for the subsequent constructions.

When block designs are used to construct switch codes, an element corresponds to an information bit, and a block (or subset) represents a parity bit, summing (XORing) the information elements contained in it. The constructed switch code is composed of the information bits (elements) and the parity bits (blocks). For simplicity, we also view every information bit as a subset of size 1. For example, the bits $u_i + u_j + u_h$, $u_j + u_h + u_l$ are associated with the blocks $\{i, j, h\}$, $\{j, h, l\}$, respectively. In this example we have an intersection of size 2 between the two blocks. Since $\{j, h\} \subset \{i, j, h\}$ and $\{j, h\} \subset \{j, h, l\}$, we say that the pair $\{j, h\}$ *appears* in both of these blocks. Two blocks of size 3 with intersection size 2 are central in our constructions, because they represent code bits with encoding degree 3 that can be used to solve an information bit with decoding degree 3. In particular, information bit $\{i\}$ can be recovered by taking the sum of the following subsets:

$$\{l\}, \{i, j, h\}, \{j, h, l\}. \qquad (5)$$

We call these 3 code bits *helpers* for the element i, where the set of size 1 is called a *systematic helper* and the sets of size 3 are called *parity helpers*.

Next we define a type of block design whose properties are desirable but not sufficient for a switch code.

Definition 17 *A balanced incomplete block design (BIBD) with parameters $(k, b, \alpha, r, t, \lambda)$ is a system with*

1) *a total of k elements, $\mathcal{P} = \{0, 1, \dots, k-1\}$,*
2) *b blocks, $\mathcal{B} = \{B_1, B_2, \dots, B_b\}, where B \subseteq \mathcal{P}$ for all $B \in \mathcal{B}$,*
3) *each block with size α, namely, $|B| = \alpha$ for all $B \in \mathcal{B}$,*
4) *every element repeats r times, i.e., $|\{B \in \mathcal{B} : i \in B\}| = r$ for all $i \in \mathcal{P}$, and*
5) *every subset of size t appears exactly λ times, i.e., $|\{B \in \mathcal{B} : B' \subseteq B\}| = \lambda$ for all $B' \subseteq \mathcal{P}, |B'| = t$.*

A triple system *with $\lambda = 2$ is a BIBD such that each block contains $\alpha = 3$ elements out of a total of k elements, and every $t = 2$ elements appears exactly twice in the blocks.*

Since the code is systematic and a request might require k bits from the same element, we require that every element repeats $r = k - 1$ times in the blocks. From the definition one can see that a carefully designed triple system may give a degree-3 switch code. By simple counting argument, we see that the number of parity bits is $b = k(k-1)/3$, and is optimal by Corollary 5.

Example 18 *Consider the following triple system with $k = 6$ elements and $k(k-1)/3 = 10$ blcoks: $\{0, 1, 2\}$, $\{0, 2, 3\}$, $\{0, 1, 4\}$, $\{1, 2, 5\}$, $\{0, 3, 5\}$, $\{2, 3, 4\}$, $\{0, 4, 5\}$, $\{1, 4, 3\}$, $\{1, 5, 3\}$, $\{2, 5, 4\}$. Every element repeats $k - 1 = 5$ times. Consider the corresponding switch code with 6 systematic bits and 10 parity bits. Suppose the request vector is $\mathbf{L} = (6, 0, 0, 0, 0, 0)$), then we can solve it in the following way:*

$$\{0\} \tag{6}$$
$$\{1\}, \{0, 3, 5\}, \{1, 5, 3\}$$
$$\{2\}, \{0, 4, 5\}, \{2, 5, 4\} \tag{7}$$
$$\{3\}, \{0, 1, 4\}, \{1, 4, 3\} \tag{8}$$
$$\{4\}, \{0, 2, 3\}, \{2, 3, 4\}$$
$$\{5\}, \{0, 1, 2\}, \{1, 2, 5\}$$

We can see that every code bit was used exactly once, hence we are able to solve bit $\{0\}$ six times from disjoint helper sets. Similarly, one can check that it is possible to solve any bit six times from disjoint sets. And also any one-burst together with arbitrary uniques can be solved with disjoint helper sets, as long as the request weight is $k = 6$. For example, for $\mathbf{L} = (3, 1, 0, 0, 1, 1)$ we can use equations (6)(7)(8) and singletons $\{1\}$, $\{4\}$, $\{5\}$.

Motivated by this example, in the following subsections we construct families of switch codes from triple systems.

B. Linear construction

In the following construction, which we call the *linear construction*, the solvability of one-burst requests is proven with an explicit decoding algorithm. The key idea is to have a simple way to pick the pair (j, h) in (5) given the requested information bit $\{i\}$ and the systematic helper $\{l\}$. One candidate of a simple mapping from (i, l) to (j, h) is a linear function. The following construction uses a block design specified through such a linear mapping.

In the construction we assume that all elements of the block design and operations are for the finite field $\mathbb{F}_k$, for a prime k. But the resulting switch code is still binary.

Construction 2 (linear construction) *Pick a prime $k > 3$, such that -3 is a quadratic residue modulo k. For every distinct pair of $i, l \in \mathbb{F}_k$, define (j, h) as functions of (i, l) by the following linear mapping, $i, j, h, l \in \mathbb{F}_k$:*

$$\begin{bmatrix} j \\ h \end{bmatrix} = A \begin{bmatrix} i \\ l \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} i \\ l \end{bmatrix}.$$

We denote by this linear mapping $(i, l) \mapsto (j, h)$. Here the coefficients are

$$a = d = \frac{1}{2} + \frac{\sqrt{-3}}{6}, b = c = \frac{1}{2} - \frac{\sqrt{-3}}{6}. \tag{9}$$

Then take blocks $\{i, j, h\}, \{j, h, l\}$ for all quadruples (j, h, i, l) that satisfy the linear system, and remove multiplicities. Use each block as a parity bit in the codeword, and include systematic bits in the codeword.

An example of Construction 2 for $k = 7$ is given in Figure 5. We list some of the calculations leading to the blocks of Figure 5. For $k = 7$ we have

$$A = \begin{bmatrix} 2 & 6 \\ 6 & 2 \end{bmatrix}.$$

To see, for example, how block 1 in Figure 5 is obtained, we show in the following table the mapping of three pairs of (i, l) indices to (j, h) indices

(i, l)	(j, h)	blocks $\{i, j, h\}, \{j, h, l\}$
$(0, 1)$	$(6, 2)$	$\{0, 6, 2\}, \{6, 2, 1\}$
$(4, 2)$	$(6, 0)$	$\{4, 6, 0\}, \{6, 0, 2\}$
$(3, 6)$	$(0, 2)$	$\{3, 0, 2\}, \{0, 2, 6\}$

In fact, if we swap the roles of i, l, we get the same table again with j, h swapped. We treat the instances as the same one after swapping i, l and j, h. Therefore the block $\{0, 2, 6\}$ is generated three times, once in each of the listed $(i, l) \mapsto (j, h)$ mappings. This means that, in effect, the block can be used for three different (unordered) request and helper pairs (i, l).

A similar enumeration for $k = 19$ gives 114 blocks, where the mapping is

$$A = \begin{bmatrix} 17 & 3 \\ 3 & 17 \end{bmatrix}.$$

Before formally showing that Construction 2 solves one-burst requests, we first prove some facts about the

block index	1	2	3	4	5	6	7	8	9	10	11	12	13	14
block elements	0	1	0	2	0	3	0	1	0	2	0	1	3	1
	2	2	4	4	4	4	1	3	2	3	1	5	5	2
	6	6	5	5	6	6	3	4	3	5	5	6	6	4

Fig. 5. A linear construction with $k = 7$ and 14 blocks. Each column is a block. Notice that every block can be used as a parity helper for six possible elements. Take block 1 as an example: blocks 1,2 can help solve element 0 or 1; blocks 1,5 can help solve element 2 or 4; blocks 1,9 can help solve element 3 or 6.

linear mapping A when $k > 3$ is a prime and -3 is a quadratic residue modulo k.

Lemma 19 (Uniqueness) *For any pair (j, h) there is a unique pair (i, l) such that $(i, l) \mapsto (j, h)$.*

Proof: This follows directly from the invertibility of A, implied by $det(A) = \sqrt{-3}/3 \neq 0$. ■

Lemma 20 (Completeness) *For any pair of distinct indices (i, l), the mapping $(i, l) \mapsto (j, h)$ results in i, l, j, h that are all distinct.*

Proof: The possibility of $j = h$ is excluded by the symmetries $a = d$, $b = c$ in A. Assume by contradiction that the mapping gives $j = i$. (The other violating cases $j = l$, $h = i$, $h = l$ can be similarly contradicted.) In that case we have $ai + bl = i$, which after substitution of a,b from (9) gives $b(l - i) = 0$, a contradiction given that $i \neq l$. ■

The meaning of Lemma 20 is that every pair of requested symbol i and systematic helper l uniquely defines two corresponding parity helpers $\{i, j, h\}$, $\{l, j, h\}$.

Lemma 21 (Symmetry) *If $(i, l) \mapsto (j, h)$, then $(l, i) \mapsto (h, j)$.*

Lemma 21 follows directly by the symmetries $a = d$, $b = c$ in A. This lemma implies that we can exchange the roles of requested symbol and systematic helper and get the same parity helper sets.

Lemma 22 (Multiplicity) *Every parity helper set $\{i, j, h\}$ is generated by three distinct instances of the mapping A, where swapping the requested symbol and the systematic helper does not count as a new instance.*

Proof: If the helper set $\{i, j, h\}$ exists in the construction it means that two of its elements, (j, h) without loss of generality, are the image of the mapping A when acting on a pair (i, l), for some element l. That is, $(i, l) \mapsto (j, h)$, and from Lemma 19 the pair (i, l) is unique to generate $\{i, j, h\}$ with (j, h) as the image. Exchanging the roles of i, j, h, there are at most 3 instances that generates the helper set. Next we examine the element

$$l' \triangleq \frac{j - bh}{a}.$$

By the relations $ai + bl = j$, $bi + al = h$ we write $j - bh = (a - b^2)i + (b - ab)l$. Now for the specific parameters $a = \frac{1}{2} + \frac{\sqrt{-3}}{6}, b = \frac{1}{2} - \frac{\sqrt{-3}}{6}$, we have

$$\frac{a - b^2}{a} = \frac{1 - ab}{b}, \quad \frac{b - ab}{a} = -\frac{a^2}{b},$$

which gives the first equality in

$$\frac{i - ah}{b} = \frac{j - bh}{a} = l'.$$

Now in addition we have the relations

$$ah + bl' = i, \quad bh + al' = j,$$

which imply that $(h, l') \mapsto (i, j)$, and thus show another instance of the helper set $\{i, j, h\}$. In a similar way we can show that $\{i, j, h\}$ is also generated by the mapping $(j, l'') \mapsto (h, i)$, for

$$l'' = \frac{h - aj}{b} = \frac{i - bj}{a}.$$

By Lemma 19 the latter two instances of $\{i, j, h\}$ are also uniquely generated by (h, l') and (j, l''), respectively. By Lemma 20 the 3 instances are all distinct. ■

We are now ready to prove the properties of the linear construction.

Theorem 23 *Construction 2 solves any one-burst request of weight k with decoding degree 3, and has $k(k - 1)/3$ parity bits, where $k > 3$ is a prime and -3 is a quadratic residue modulo k.*

Proof: First we show the number of parity bits. By Lemmas 20, 21, every pair of (unordered) distinct indices from $\{1, \ldots, k\}$ defines two parity helper sets. By Lemma 22 every parity helper set is generated by three pairs of indices. Thus all together there are $\binom{k}{2} \times 2/3 = k(k - 1)/3$ parity bits.

Next we show that any one-burst request is solvable. Let $\mathbf{L}$ be the request, and denote by m the bit index with $l_m > 1$ in $\mathbf{L}$. If no such m exists, the request can be trivially solved from the systematic bits. Because the request weight is $\sum_{i=0}^{k-1} l_i = k$, we know that there are $l_m - 1$ bit indices that are not requested. We denote the set of these indices by U, with $|U| = l_m - 1$. Now to solve $\mathbf{L}$ we take all index pairs $(m, u) : u \in U$, and for each such pair apply the mapping $(m, u) \mapsto (j_u, h_u)$. Then we solve each of $l_m - 1$ requests of m using the parity helper sets $\{m, j_u, h_u\}$, $\{u, j_u, h_u\}$ and the systematic bit $\{u\}$. The remaining request of m is solved from the systematic bit, and the requests of the uniques $\{i \neq m : i \notin U\}$ are also solved from their systematic bits.

To prove that decoding always succeeds, we need to show that there is no set $\{x, y, z\}$ that is generated twice. We first observe that if there exists such a set it does not contain m: two sets $\{m, j_u, h_u\}$ and $\{m, j_v, h_v\}$ must be distinct by Lemma 19, and by Lemma 20 m cannot be in a set $\{u, j_u, h_u\}$ for any $u \in U$. So we now only need to prove that for distinct $u, v \in U$, we get $\{u, j_u, h_u\} \neq \{v, j_v, h_v\}$ as sets. Assume by way of contradiction that $\{u, j_u, h_u\} = \{v, j_v, h_v\}$. It is sufficient to consider the two possible cases below, and the other two possible cases are symmetrically identical.

- For some y, $(m, u) \mapsto (v, y)$ and $(m, v) \mapsto (u, y)$. From linearity we get that $(0, u - v) \mapsto (v - u, 0)$, but this violates Lemma 20 requiring all-distinct elements in the mapping.
- For some y, $(m, u) \mapsto (v, y)$ and $(m, v) \mapsto (y, u)$. From linearity we get that $(0, u-v) \mapsto (v-y, y-u)$. But this requires $v - y = b(u - v)$ and $y - u = a(u-v)$, which sums to $a+b = -1$, a contradiction to (9).

Hence the helper sets are disjoint, and the proof is completed. ■

We note here that the construction does not always work for a general type of request. For example, if the request is $\mathbf{L} = (3, 4, 0, 0, 0, 0, 0)$ (with two bursts) for the code in Figure 5, then it is not possible to find distinct sets $\{x, y, z\}$ for recovering both bits $\{0\}$ and $\{1\}$ from parity helper sets.

Finally, we show that there exit infinitely many input/request sizes that satisfy the constraint in Construction 2. More specifically, by the results from Number Theory [11], we can obtain the input/request sizes k supported by the linear construction, as characterized in the following lemma. Then by Dirichlet's Theorem [11], we can conclude that Construction 2 gives an infinite family of codes.

Lemma 24 *The following three conditions are equivalent for a prime $k > 3$:*
(i) -3 is a perfect square mod k.
(ii) $(k-1)/3$ is an integer.
(iii) $k \equiv 1 \mod 6$.
Moreover, there are infinitely many k satisfying the above conditions.

Proof: By the result of [11, Theorem 96, page 75], condition (i) is equivalent to condition (iii). One can easily check the equivalence of (ii) and (iii). By Dirichlet's Theorem [11, Theorem 15, page 13], for given positive coprime numbers a, b, there are infinitely many primes of the form $a \times n + b$ for $n \in \mathbb{Z}^+$. Thus there are infinitely many k satisfying condition (iii). ■

C. Top-down construction

In this sub-section our objective is to construct a switch-code family for one-burst requests, which gives codes for more values of k than offered by Construction 2 in Section IV-B. In particular, the *top-down* construction will be shown to give switch codes for any $k \equiv 1, 4 \mod 12$, not necessarily a prime number. Moreover, the top-down construction can be easily generalized to parity degrees $d > 3$. The disadvantage of the top-down construction over the previous linear construction is that it can only guarantee decoding a burst of length $(k-1)/3+1$ or less.

The main idea of the top-down construction is to obtain parity helper sets of size 3 by breaking blocks of size 4 into triples, in the way described below. Let us start with an example. Consider the block $\{i, l, j, h\}$ of size 4 and we use its subsets of size 3 as parity sets:

$$\{i, l, j\}, \{i, l, h\}, \{i, j, h\}, \{l, j, h\}. \qquad (10)$$

For any two elements $m, u \in \{i, l, j, h\}$, we can solve bit m using the systematic bit u and the two triples

$$\{i, l, j, h\} \backslash \{m\}, \qquad (11)$$

$$\{i, l, j, h\} \backslash \{u\}, \qquad (12)$$

as parity helper sets.

Construction 3 (top-down construction) *Let D_4 be a BIBD with k elements, and $k(k-1)/12$ blocks of size 4. Moreover, every element repeats $(k-1)/3$ times in D_4, and every pair appears once. We call such BIBD a quadruple system. For each block $\{i, l, j, h\}$ of D_4 take the four sets $\{i, l, j\}$, $\{i, l, h\}$, $\{i, j, h\}$, $\{l, j, h\}$ as parity sets, and include systematic bits in the codeword.*

Regard the size-3 sets taken by Construction 3 as the blocks of a block design D. It is easy to check that D is a triple system with $k(k-1)/3$ blocks, $k-1$ repeats, and every pair appears exactly twice. Before showing the properties of Construction 3, let us examine an example of a quadruple system D_4.

Example 25 *Take D_4 as the BIBD with $k = 25$ elements and 50 blocks of size 4 (see [12]). Every integer in $[0, 24]$ is represented as $5i + j$, for some $i, j \in [0, 4]$. The design is formed by dicyclic solution in two families:*

$$5(a, a, a+1, a+4) + (b, b+1, b, b+4),$$

$$5(c, c, 2+c, 3+c) + (d, d+2, d, d+3),$$

where $a, b, c, d \in [0, 4]$ and additions inside brackets are computed modulo 5. For example $\{0, 1, 5, 24\}$ is the quadruple corresponding to $a = 0, b = 0$ in the first family, and $\{8, 5, 18, 21\}$ corresponds to $c = 1, d = 3$ in the second family. When using this D_4 in Construction 3 we transforms the 50 blocks into 200 triples.

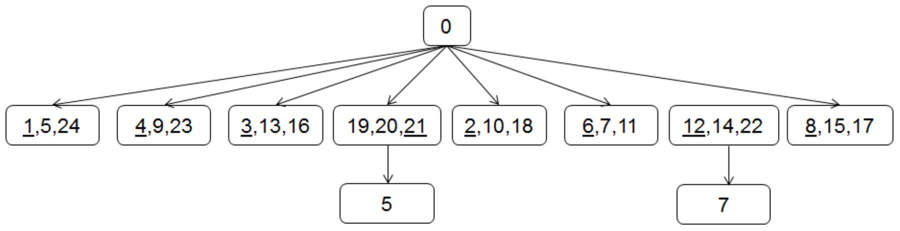

Fig. 6. Example of decoding a request with burst length 9, where $T = \{1, 2, \ldots, 8\}$ are unrequested elements. The element requested with multiplicity is the root of the tree 0. The underlined element in each child of 0 is a systematic helper used to solve 0. For the children sets $\{19, 20, 21\}$ and $\{12, 14, 22\}$, all elements are requested, and therefore the chosen systematic helpers 21 and 12 need 5 and 7, respectively, as their helpers.

To understand how Construction 3 solves one-burst requests, we give a decoding example for the D_4 of Example 25. Let the request be $\mathbf{L} = (9, 0, \ldots, 0, 1, \ldots, 1)$, that is, bit 0 is requested with multiplicity 9, bits $1, \ldots, 8$ are not requested, and bits $9, \ldots, 24$ are uniques. To solve the request, we draw a tree rooted at the element with the burst request, 0 in this example. The children of the root 0 are all triples whose union with 0 are blocks in D_4. For $k = 25$ there are 8 such triples, see Figure 6.

In each child of 0, if the triple contains an unrequested element u, we solve 0 with systematic bit u and two parity helper triples as specified in (11), (12), where $m = 0$. For example, we solve 0 by $\{1\}$ and $\{0, 5, 24\}$, $\{1, 5, 24\}$, by $\{4\}$ and $\{0, 9, 23\}$, $\{4, 9, 23\}$, and so on. For the child triples that do not have an unrequested element, pick an arbitrary element m in the triple and use it as the systematic helper. However, since m is also requested, we solve it using any unused and unrequested element $\{v\}$ and parity helpers $\{m, j(m, v), h(m, v)\}$, $\{v, j(m, v), h(m, v)\}$ coming from D_4's block $\{m, v, j(m, v), h(m, v)\}$. For example, in the triple $\{19, 20, 21\}$ of Figure 6 all symbols are requested, so we choose $m = 21$ as the systemic helper and solve it with the help of the unused and unrequested $\{v = 5\}$, together with triples from the block $\{8, 5, 18, 21\}$ in D_4. Then we are free to use the systematic bit $\{m\}$ to solve 0 with the child triple – in the example we use $\{21\}$ and $\{0, 19, 20\}$,$\{19, 20, 21\}$ to solve 0.

We formalize the above procedure with a greedy decoder, given in Algorithm 1, for the maximal burst length $(k-1)/3 + 1$. In the notation of Algorithm 1, bit a is requested with multiplicity b, bits $a_1, \ldots, a_{b-1}$ are not requested, and bits $[0, k-1] \setminus \{a, a_1, \ldots, a_{b-1}\}$ are requested once. The algorithm uses the tree of depth 1 rooted at bit a similar to Figure 6. From each child-triple of a Algorithm 1 assigns an element h_i as the systematic helper to solve a, together with two corresponding parity helpers. In Line 4, h_i is assigned as an unrequested element. When h_i is assigned in Line 11, h_i is both a requested element and a helper for a. Then the algorithm assigns in Line 12 an unused and unrequested element a_i as the systematic helper to solve h_i, together with two corresponding parity helpers. Finally, the uniques excluding h_i's in Line 11 are solved by singletons. One can easily see that the decoding degree is 3.

Algorithm 1 Decoding algorithm for top-down Construction 3 with one burst of length $b = (k-1)/3 + 1$ for bit a.

```
1: Initialize a set $T$ as the unrequested elements $T = \{a_1, \ldots, a_{b-1}\}$.
2: for $i = 1$ to $b - 1$ do
3:     if there exists a member of $T$ in child $i$ of $a$ then
4:         assign this member as $h_i$
5:         remove this member from $T$
6:         solve $a$ with systematic helper $h_i$
7:     end if
8: end for
9: for $i = 1$ to $b - 1$ do
10:    if $h_i$ is not assigned then
11:        assign any element in child $i$ as $h_i$
12:        assign any member in $T$ as helper for $h_i$ (w.l.o.g. call this member $a_i$)
13:        remove this member from $T$
14:        solve $a$ with systematic helper $h_i$
15:        solve the unique $h_i$ with systematic helper $a_i$
16:    end if
17: end for
18: solve the remaining uniques by singletons
```

We are now ready to prove that Construction 3 with Algorithm 1 guarantees decoding success.

Theorem 26 *Construction* 3 *solves any one-burst request with burst length at most* $(k-1)/3 + 1$ *with decoding degree 3.*

Proof: Let us first show that Algorithm 1 solves requests with one burst of length $b = (k-1)/3 + 1$. We first check that the systematic helpers are disjoint. Since each index pair appears once in D_4, and the children of a correspond to the blocks containing a in D_4, we know the elements of the children are all distinct. Hence $h_i \neq h_j$ for $i \neq j$. When h_i is assigned in Line 11, we know that $a_i \neq h_i$ since h_i is a requested element but a_i is not.

We next need to show that the parity helpers used in this assignment are also distinct. We use the ordered-pair notation (m, u) to denote that element m is solved with element u as systematic helper. We consider two pairs (m_1, u_1) and (m_2, u_2) in the following 4 cases.

- (a, h_i) and (a, h_j) for some $i \neq j \in [1, b-1]$, each assigned in Line 4 or 11 of Algorithm 1. Since any pair appears once in D_4 and $h_i \neq h_j$, we know the quadruples containing $\{a, h_i\}$ and $\{a, h_j\}$ are distinct, moreover, the parity helpers are distinct.
- (h_i, a_i) and (h_j, a_j) for some $i \neq j \in [1, b-1]$, each assigned in Line 12. Note that by the above decoding algorithm h_i, h_j, a_i, a_j are all distinct. The corresponding parities are
$$\{h_i, x, y\}, \{x, y, a_i\}$$
$$\{h_j, z, w\}, \{z, w, a_j\}$$
for some x, y, z, w. The former are sub-blocks of $A = \{h_i, a_i, x, y\}$ and the latter of $B = \{h_j, a_j, z, w\}$, both in D_4. Since D_4 contains each pair only once, these assignments share a parity only if $A = B$, or equivalently $\{x, y\} = \{h_j, a_j\}, \{z, w\} = \{h_i, a_i\}$. But even in this case, all the 4 parity helpers are distinct.
- (a, h_i) and (h_j, a_j) for some $i \neq j \in [1, b-1]$, the first assigned in Line 4 or 11 and the second in Line 12. Like in the previous case a, h_i, h_j, a_j are all distinct, and therefore the parities are distinct.
- (a, h_i) and (h_i, a_i) for some $i \in [1, b-1]$, the first assigned in Line 11 and the second in Line 12 of the same iteration. But notice that a_i does not belong to the i-th child of a, otherwise it would have been assigned in Line 4. So $\{a, h_i\}$ and $\{h_i, a_i\}$ appear as pairs in different blocks of D_4, and therefore the parity helper subsets of these blocks must be distinct.

Finally, suppose the burst length is smaller than $(k-1)/3+1$. A simple modification of Algorithm 1 can address this case. Suppose some a_i is also requested, $i \in [1, b-1]$. If a_i was assigned as a helper of a, do not use it to solve a but read a_i instead. If a_i was assigned as a helper of h_i, do not use h_i to solve a, but read a_i, h_i instead. ■

The following shows the existence of the top-down construction by the existence of D_4 (see e.g. [13]).

Corollary 27 *There exists an (n, k) switch code with $n = k + k(k-1)/3$ that solves any one-burst request of weight no more than $(k-1)/3+1$, for any $k \equiv 1, 4$ mod 12.*

This construction can be generalized to parities that are XOR of more than three elements. For example, if one has a block design of block size $d+1$, then by similarly breaking it down to blocks of size d, one may use one systematic helper and two parity helpers to solve a bit. Meanwhile since the parity has higher degree, we may expect to get smaller redundancy.

V. Concluding Remarks

The constructions given in this paper provide guaranteed, maximally parallel, efficient reconstruction of data from a distributed memory. While all three switch-code constructions are optimal, the amount of redundancy they use may be too high for a cost-effective deployment in switches. Reducing the code redundancy may be achieved if (and only if) some relaxations of the problem model are applied. For example, further restricting the type of packet requests seems most promising for that objective.

In addition, the focus of this paper is on codes that guarantee worst-case decoding performance, while in real-world switching it may suffice to provide probabilistic guarantees assuming some distribution on the requests. Examples of probabilistic decoding can be found for locally decodable codes [4] and batch codes [1]. We conjecture that the redundancy of switch codes can be reduced under probabilistic decoding, with arbitrarily small error probability.

We have restricted our encoders to be intra-generation and fixed, which is motivated by lowering the complexity of the writing process to the memory banks. However, it remains open to demonstrate whether there is advantage of inter-generation and non-fixed encoders. Notice that under inter-generation encoders, requests from multiple generations can be decoded jointly, and the disjointedness of the helper sets is no longer a necessary constraint. As a result, more flexibility is allowed for designing the codes, and may lead to better redundancy.

Appendix
Heuristic Search Algorithm

In this section, we describe the heuristic search algorithm used in the proof of Lemma 11, shown in Algorithm 2. In this algorithm, the input $L = (l_0, l_1, \dots, l_{K-1})$ is a request vector with weight $w(L) = 2^{K-1} - K$. Even though we did not prove that this algorithm guarantees a type I solution, we ran the algorithm and successfully found type I solutions for all requests L with $l_0 \geqslant l_1 \geqslant \dots \geqslant l_{K-1}$ and weight $w(L) = 2^{K-1} - K$, $3 \leqslant K \leqslant 7$. In fact, we further reduced the amount of search using recursion as described in the proof of Lemma 11 (i).

First, we define two functions used in the algorithm. Let I be a set of integers. For $i \in I$, define a function Δ_i as follows. For $T \subseteq I$,
$$\Delta_i(T) = \begin{cases} T \cup \{i\}, & \text{if } i \notin T, \\ T \backslash \{i\}, & \text{else.} \end{cases}$$
For a set $X \subseteq \mathcal{P}(I)$, where $\mathcal{P}(I)$ is the power set of I, we write
$$\Delta_i(X) = \{\Delta_i(T) : T \in X\}.$$
For a set X, let rand(X) be a function that returns a row vector of all elements in X randomly ordered. This random function is used in Line 10, where row vector

Algorithm 2 Find type I solution for request vector L. The input $L = (l_0, l_1, \dots, l_{K-1})$ is a request vector with weight $w(L) = 2^{K-1} - K$, $K \geqslant 3$. The output is the type I solution for L.

```
1:  $L_0 \leftarrow L$
2:  for $i = 0$ to $K-2$ do
        Suppose $L_i = (0, \dots, 0, a_i, a_{i+1}, \dots, a_{K-1})$
3:      Compute $F_i = (0, \dots, 0, a_i, 0, \dots, 0)$
        Compute $L_{i+1} \leftarrow (0, \dots, 0, 0, \lfloor \frac{a_{i+1}}{2} \rfloor, \dots, \lfloor \frac{a_{K-1}}{2} \rfloor)$
        Compute $P_{i+1} \leftarrow (0, \dots, 0, 0, a_{i+1} \bmod 2, \dots, a_{K-1} \bmod 2)$
4:  end for
5:  loop
6:    Initialize available set $AV_0$ to be all non-singleton elements, $AV_0 \leftarrow \{T \subseteq \{0, 1, \dots, K-1\} : |T| \geqslant 2\}$
7:    for $i = 0$ to $K-2$ do
8:      for every requested symbol $r$ in $P_{i+1}$ do
9:        Set the preference set, $PR \leftarrow \{T \in AV_i : \Delta_i(T) \notin AV_i\}$
10:       Set the candidate vector, $CN \leftarrow (\text{rand}(PR), \text{rand}(AV_i \backslash PR))$
11:       Search for every element $T$ of $CN$. Use helper pair $\{T, \Delta_r(T)\}$ for request $r$ if $\Delta_r(T) \in AV_i$
12:       Remove $T, \Delta_r(T)$ from $AV_i$
13:       If no solution was found, go to Line 6
14:     end for
15:     $AV_{i+1} \leftarrow \{T \subseteq \{i+1, \dots, K-1\} : T \in AV_i, \Delta_i(T) \in AV_i\}$
16:   end for
17:   Noticing $L_{K-1} = (0, \dots, 0)$, assign the solution for $P_{K-1}$ as empty
18:   for $i = K-2$ to $0$ do
19:     For every helper pair $\{T, \Delta_r(T)\}$ for $L_{i+1}$, construct another helper pair $\{\Delta_i(T), \Delta_i(\Delta_r(T))\}$ for $L_{i+1}$,
        where $T \subseteq \{i+1, \dots, K-1\}$
20:     From $AV_i$ remove all solutions to $2L_{i+1}$
21:     Find helpers $T, \Delta_i(T) \in AV_i$ for requests of $F_i$
22:     If no solution was found for $F_i$, go to Line 6
23:     Construct solution to $L_i$ as the solutions to $F_i, P_{i+1}, 2L_{i+1}$
24:   end for
25: end loop
```

CN is set to be a concatenation of the two random row vectors.

The algorithm works as follows: we first **decompose** the request vector L as in Lines 1 to 4. After that, we try to find a type I solution to L using **forward** iterations (Lines 6 to 16) and **backward** iterations (Lines 17 to 24). These iterations are repeated multiple loops (Lines 5 and 25). For every loop, we randomly search for solutions. If the search fails, we start a new loop (Lines 13 and 22).

We first decompose request L in $K-1$ iterations as in Lines 1 to 4. We denote by L_i the remaining vector in iteration i, for $0 \leqslant i \leqslant K-2$, and set $L_0 \leftarrow L$. The first $i-1$ coordinates of L_i are zeros, for $1 \leqslant i \leqslant K-2$. In each iteration, suppose the remaining vector is $L_i = (0, \dots, 0, a_i, \dots, a_{K-1})$. Let us denote

$$\begin{aligned} F_i = & \quad (0, \dots, 0, a_i, 0, \dots, 0), \\ L_{i+1} = & \quad (0, \dots, 0, 0, \lfloor \tfrac{a_{i+1}}{2} \rfloor, \dots, \lfloor \tfrac{a_{K-1}}{2} \rfloor), \\ P_{i+1} = & \quad (0, \dots, 0, 0, a_{i+1} \bmod 2, \dots, a_{K-1} \bmod 2). \end{aligned}$$

Then L_i is decomposed into 3 parts:

$$L_i = F_i + P_{i+1} + 2L_{i+1}.$$

Note that when $i = K-2$, suppose $L_{K-2} = (0, \dots, 0, a_{K-2}, a_{K-1})$, then

$$a_{K-1} \leqslant \frac{l_{K-1}}{2^{K-2}} \leqslant \frac{2^{K-1} - K}{2^{K-2}} < 2.$$

Therefore, $a_{K-1} = 0$ or 1, and

$$L_{K-1} = (0, 0, \dots, 0).$$

In forward iterations (Lines 6 to 16), we find a type I solution for every P_{i+1}, $i = 0, \dots, K-2$. In order to solve P_{i+1}, we use $AV_i \subseteq \mathcal{P}(\{i, \dots, K-1\})$ to store all available elements. For iteration $i = 0$, AV_i is all non-singleton elements (Line 6). The forward iteration i is illustrated in Figure 7 (a). For every request r of P_{i+1}, $i+1 \leqslant r \leqslant K-1$, we randomly pick a helper pair $\{T, \Delta_r(T)\}$, both of which are available (Line 11). Then the helpers are marked unavailable (Line 12). For iteration $i+1$, an element $T \subseteq \{i+1, \dots, K-1\}$ is available if both $T, \Delta_i(T) \in AV_i$ (Line 15). This means that if only one of $T, \Delta_i(T)$ is unavailable for iteration i, one element is "wasted". The wasted elements will not be used as helpers in future forward iterations. Therefore, we would like to avoid waste as much as possible. We note that when $i \geqslant 1$, the wasted elements may still be used as helpers in backward iterations $0, 1, \dots, i-1$. The reason of choosing AV_{i+1} as Line 15 will be clear later.

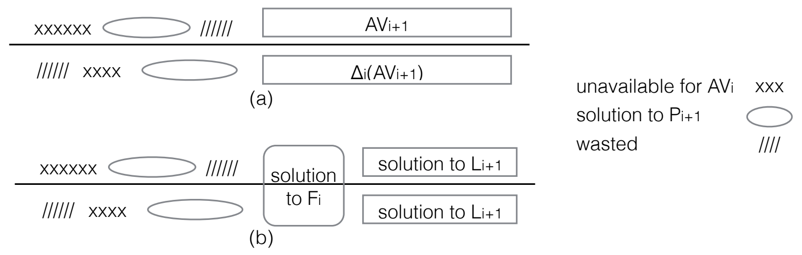

Fig. 7. (a) Forward iteration i, (b) backward iteration i. In both figures, above the horizontal line are all elements of $\mathcal{P}(\{i+1,\ldots,K-1\})$, and below the horizontal line are all elements of $\Delta_i(\mathcal{P}(\{i+1,\ldots,K-1\}))$. In (a), at the beginning of the forward iteration i, elements "xxxx" are unavailable. Then we search for the solution to P_{i+1}, represented by the ovals. Finally AV_{i+1} are assigned as in Line 15, and elements "///" are wasted. In (b), at the beginning of the backward iteration i, solution to L_{i+1} is already found in AV_{i+1}, shown above the horizontal line. We duplicate it in $\Delta_i(AV_{i+1})$, shown below the horizontal line. Moreover, solution to P_{i+1} is also known. We search for the solution to F_i, and thus obtain the solution to L_i.

In order to expedite the search, we do not search exhaustively. Instead we search for a helper where T is from a preference set PR (Line 10). The set $PR \subseteq AV_i$ is defined in Line 9. One can see that if we use $T \in PR$ as a helper, then $\Delta_i(T)$ is already unavailable, which reduces waste in Line 15.

Suppose that after the forward iterations we have found a solution to P_{i+1} from $\mathcal{P}(\{i,\ldots,K-1\})$, for $i=0,\ldots,K-2$. Next we find a solution to L_i from $\mathcal{P}(\{i,\ldots,K-1\})$, for all $i=K-1,K-2,\ldots,0$ (Lines 17 to 24) in the backward iterations. This is illustrated in Figure 7 (b). For the initial iteration, $i=K-1$, the solution to $L_{K-1}=(0,\ldots,0)$ is empty (Line 17). For iteration $i<K-1$, given a helper pair $\{T,\Delta_r(T)\}$ for L_{i+1}, we construct another pair $\{\Delta_i(T),\Delta_i(\Delta_r(T))\}$ (Line 19). Then we look for solutions to F_i from AV_i (Line 21). Finally the solution to L_i is composed of the solutions to $F_i, P_{i+1}, 2L_{i+1}$ (Line 23).

Lemma 28 *If a solution to L is found in Algorithm 2, then it is a valid type I solution.*

Proof: It is easy to see that the helpers found in Lines 11, 19, and 21 are correct. Next we show that the helpers are disjoint and type I.

We claim that the solution to L_i (Line 23) is a subset of AV_i', where AV_i' is the availability set at Line 8 for forward iteration i. Namely, AV_0' is defined in Line 6, and AV_{i+1}' is defined in Line 15, $i=0,\ldots,K-2$. Clearly, at any step after Line 8, $AV_i \subseteq AV_i'$.

This claim is proved by induction. For $i=K-1$, the solution is empty, so the claim holds. Suppose the claim holds for iteration $i+1$. So at Line 19, $T,\Delta_r(T) \in AV_{i+1} \subset AV_i$, and by Line 15 we conclude that $\Delta_i(T),\Delta_i(\Delta_r(T))$ are also in AV_i. So the solutions to $2L_{i+1}$ are in AV_i'. By Line 11 and Line 21, the solutions to P_{i+1} and F_i are also in AV_i'. Namely, the solution to L_i is a subset of AV_i', thus the claim holds for iteration i.

Note that AV_0' are all the non-singleton elements as in Line 6. Moreover, every used solution is marked unavailable (Line 20). We conclude that if solution to $L=L_0$ is found, then it is valid and type I. ∎

References

[1] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Batch codes and their applications," in *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pp. 262–271, ACM, June 2004.

[2] A. Rawat, Z. Song, A. Dimakis, and A. Gal, "Batch codes through dense graphs with high girth," in *Information Theory Proceedings (ISIT), 2015 IEEE International Symposium on*, IEEE, June 2015.

[3] H. Lipmaa and V. Skachek, "Linear batch codes," in *Proc. 4th International Castle Meeting on Coding Theory and Applications, Palmela, Portugal*, September 2014.

[4] S. Yekhanin, "Locally decodable codes," *Foundations and Trends in Theoretical Computer Science*, vol. 6, no. 3, pp. 139–255, 2012.

[5] F. Oggier and A. Datta, "Self-repairing homomorphic codes for distributed storage systems," in *Computer Communications, the 29th IEEE Intl. Conference on (INFOCOM)*, IEEE, April 2011.

[6] L. Pamies-Juarez, H. D. L. Hollmann, and F. E. Oggier, "Locally repairable codes with multiple repair alternatives," *CoRR*, vol. abs/1302.5518, 2013.

[7] A. Rawat, D. Papailiopoulos, A. Dimakis, and S. Vishwanath, "Locality and availability in distributed storage," in *Information Theory (ISIT), 2014 IEEE International Symposium on*, June 2014.

[8] I. Tamo and A. Barg, "Bounds on locally recoverable codes with multiple recovering sets," in *Information Theory (ISIT), 2014 IEEE International Symposium on*, June 2014.

[9] W. Huffman and V. Pless, *Fundamentals of Error-Correcting Codes*. Cambridge, UK: Cambridge university press, 2003.

[10] B. Arnold, N. Balakrishnan, and H. Nagaraja, *A First Course in Order Statistics*. Society for Industrial and Applied Mathematics, 2008.

[11] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers (Fifth Edition)*. Oxford University Press, 1980.

[12] R. Fisher and F. Yates, *Statistical tables for biological, agricultural, and medical research*. Longman Group United Kingdom, June 1995.

[13] C. J. Colbourne and J. H. Dinitz, *Handbook of Combinatorial Designs (Second Edition)*. CRC Press, 2007.

Zhiying Wang received the B.Sc. degree in Information Electronics and Engineering from Tsinghua University in 2007, M. Sc. and Ph.D degrees in Electrical Engineering from California Institute of Technology in 2009 and 2013, respectively. She was a postdoctoral fellow in Department of Electrical Engineering, Stanford University. She is currently Assistant Professor at Center for Pervasive Communications and Computing, Univeristy of California, Irvine. Dr. Wang received NSF Center for Science of Information (CSoI) Postdoctoral Research Fellow, 2013. She was the recipient of IEEE Communication Society Data

Storage Best Paper Award for 2013. Her research focuses on information theory, coding theory, with an emphasis on coding for storage devices and systems and compression for genomic information.

Han Mao Kiah received his Ph.D. degree in mathematics from the Nanyang Technological University, Singapore in 2014. From 2014 to 2015, he was a Postdoctoral Research Associate at the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign. Currently, he is a lecturer at the School of Physical and Mathemati cal Sciences, Nanyang Technological University, Singapore. His research interests include combinatorial design theory, coding theory, and enumerative combinatorics.

Yuval Cassuto (S'02-M'08-SM'14) is a faculty member at the Andrew and Erna Viterbi Department of Electrical Engineering, Technion – Israel Institute of Technology. His research interests lie at the intersection of the theoretical information sciences and the engineering of practical computing and storage systems.

During 2010-2011 he has been a Scientist at EPFL, the Swiss Federal Institute of Technology in Lausanne. From 2008 to 2010 he was a Research Staff Member at Hitachi Global Storage Technologies, San Jose Research Center. From 2000 to 2002, he was with Qualcomm, Israel R&D Center, where he worked on modeling, design and analysis in wireless communications.

He received the B.Sc degree in Electrical Engineering, summa cum laude, from the Technion, Israel Institute of Technology, in 2001, and the MS and Ph.D degrees in Electrical Engineering from the California Institute of Technology, in 2004 and 2008, respectively.

Dr. Cassuto has won the 2010 Best Student Paper Award in data storage from the IEEE Communications Society, as well as the 2001 Texas Instruments DSP and Analog Challenge $100,000 prize.

Jehoshua Bruck is the Gordon and Betty Moore Professor of computation and neural systems and electrical engineering at the California Institute of Technology (Caltech). His current research interests include information theory and its applications to memory systems, including molecular memories (DNA), associative memories (the brain), and nonvolatile memories (flash).

Dr. Bruck received the B.Sc. and M.Sc. degrees in electrical engineering from the Technion-Israel Institute of Technology, in 1982 and 1985, respectively, and the Ph.D. degree in electrical engineering from Stanford University, in 1989.

His industrial and entrepreneurial experiences include working with IBM Research where he participated in the design and implementation of the first IBM parallel computer; cofounding and serving as Chairman of Rainfinity (acquired in 2005 by EMC), a spin-off company from Caltech that created the first virtualization solution for Network Attached Storage; as well as cofounding and serving as Chairman of XtremIO (acquired in 2012 by EMC), a start-up company that created the first scalable all-flash enterprise storage system.

Dr. Bruck is a recipient of the Feynman Prize for Excellence in Teaching, the Sloan Research Fellowship, the National Science Foundation Young Investigator Award, the IBM Outstanding Innovation Award and the IBM Outstanding Technical Achievement Award.