ISIT'2013 1569719135

Codes for Network Switches

Zhiying Wang*, Omer Shaked†, Yuval Cassuto†, and Jehoshua Bruck*
*Electrical Engineering Department, California Institute of Technology, Pasadena, CA 91125, USA
†Electrical Engineering Department, Technion – Israel Institute of Technology, Technion City, Haifa 32000, Israel
zhiying@caltech.edu, shakedomer@gmail.com, ycassuto@ee.technion.ac.il, bruck@caltech.edu

***Abstract*—A network switch routes data packets between its multiple input and output ports. Packets from input ports are stored upon arrival in a switch fabric comprising multiple memory banks. This can result in memory contention when distinct output ports request packets from the same memory bank, resulting in a degraded switching bandwidth. To solve this problem, we propose to add redundant memory banks for storing the incoming packets. The problem we address is how to minimize the number of redundant memory banks given some guaranteed contention resolution capability. We present constructions of new switch memory architectures based on different coding techniques. The codes allow decreasing the redundancy by $1/2$ or $2/3$, depending on the request specifications, compared to non-coding solutions.**

I. Introduction

Multi-port switches are commonly used as data processing and routing devices in computer networks. With the growth of the amount of computing devices, network links, and data transmission, switches are facing the challenge of serving growing rates of packet transmissions on an increasing number of ports. In this paper, we focus on the switch memory sub-system used to store packets between arrival from input ports to departure to output ports. Assume for now that the switch memory sub-system has the same number of input and output ports, say R ports. At a time slot, each input port writes one data packet into the memory, and each output port reads one data packet from the memory. The R packets written in the same time slot are called a *generation*, and they share the same row in subsequent diagrams. The total bandwidth of the memory sub-system should be R packets per time slot for reading, and same for writing. As the number of ports R grows, as well as the rate of each port, an increasing number of memory banks is also used to parallelize data writing and reading on slower, and more affordable, memory devices. In the paper we assume to have multiple memory banks, each operating at a speed of one packet write and one packet read per time slot. Figure 1 (a) is an example with $R = 2$ ports and 2 banks. (A, B), (C, D), (E, F) are three generations. So far it appears that R memory banks are sufficient. For example, at the current slot we write in 2 units (E, F) (at top) and read out 2 units (A, B) (from bottom), as specified by the switch rate requirements.

However, the destinations of the packets may not be the same as their memory bank index. For example, if the output request is (A, C), then due to the memory rate limitation, this request cannot be served in one time slot. Hence we need to add redundant memory banks such that an arbitrary output request can be served. A naive solution is to replicate the original information R times and use R^2 banks. Since no more than R packets are required from the same input port, we can always serve any request of R packets in one time slot. See Figure 1 (b) for an example. However, can we attain the same flexibility with lower redundancy using coding techniques? Figure 1 (c) answers this question to the affirmative for $R = 2$. By adding the parity memory, any request of size 2 can be served such that every memory reads at most one packet. For example, if the output request is (A, C), then we can read $A, D, C + D$ from the three memories. We define a *switch code* for R ports and n memory banks as a way to represent the input packets, such that any request of R output packets can be obtained by reading at most one packet from each memory bank. A switch code in principle is not confined to encode over a single generation, but coding across multiple generations adds load on the memory through recoding, or requires to manage a separate packet cache. Therefore, in this paper we focus on switch codes with intra-generation encoding functions.

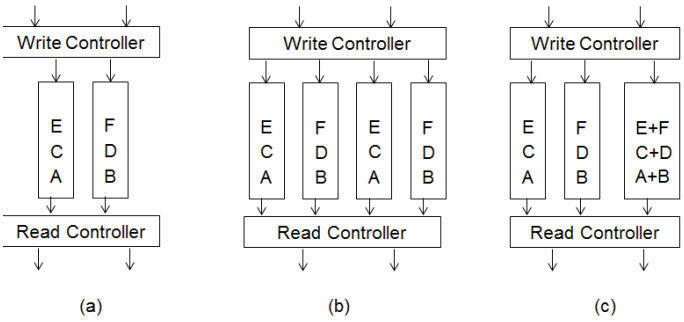

Figure 1. Codes for a switch with $R = 2$ ports. (a) Direct input and output. (b) Replication. (c) The code with the parity. The last memory computes XOR of the two packets from each generation.

Intuitively, a switch code requires that each input symbol (or packet) can be recovered from multiple small subsets of code symbols. This property will allow the simultaneous reconstruction of arbitrary R symbols requested from the switch memory, each accessing a disjoint subset of code symbols. For an extreme case of requesting R generations of the same symbol, a symbol should be recovered by R disjoint subsets. Recovering symbols from small codeword subsets is addressed by *locally decodable codes* [8], where every symbol can be recovered with high probability from a small number of symbols in the presence of errors. Theoretically speaking, it is possible to use Reed Muller codes to construct switch codes with block length (number of banks) scaling as $n = R^{1+\epsilon}$, for small ϵ, compared to $n = O(R^2)$ in our constructions. However, such a construction is impractical for a multitude of

reasons.

1) R must be extremely large (thousands or more, depending on ϵ) for decoding success with high probability.
2) Encoding and decoding operate on a number of packets that grow as a function of R. If we restrict the size of the recovery set to be a constant r then the code length would be $\exp(R^{1/(r-1)})$, in contrast, our construction operates on a recovery set of size at most 3 and requires a code length of about R^2.
3) Encoding and decoding require arithmetic over large finite fields.

In contrast, the switch codes proposed here are given as explicit families of codes starting from small R values, they guarantee decoding success, encode and decode over a small fixed number of packets, and require only simple bit-wise XOR operations between packets. These properties are critical for implementation in high-speed switches.

Another similar notion with local-reconstruction property is *locally repairable codes* [3] [6], where a symbol can be recovered from some (fixed set of) other symbols. In addition, self-repairing codes [5] are erasure codes such that a symbol can be repaired from some (fixed number of) other fragments given the number of erasures. We would like to point out that some of these above codes do allow multiple disjoint recovery, but the number of disjoint recoveries dose not grow fast enough with R.

The first result of this paper is a code construction using parities of two symbols. This construction uses $R(R-1)/2$ redundant memories, and can serve any request of R output symbols, which amounts to half the redundancy of a non-coding scheme. The second result focuses on one-burst requests, i.e., multiple requested packets from one input port, and no more than one request for the others. It is practical to consider this special type of request because a single burst can be used to empty the longest request queue, and consequently shorten the worst-case delay. We generalize the code construction such that the parities are XOR of more than two symbols, and obtain a better code rate. In particular, we construct codes based on block designs. When there are three symbols for each parity, the code has $R(R-1)/3$ redundant memories.

We will set up the switch code problem formally in Section II. Then we construct codes of pairs of parities in Section III, and codes based on block designs in Section IV. Finally, Section V concludes the paper.

II. Problem Settings

Notations: For simplicity, we denote by "+" binary XOR. The set of integers $\{i, i+1, \ldots, j\}$ is denoted by $[i, j]$ for $i \leq j$ and $\{1, 2, \ldots, i\}$ is written as $[i]$ for integer $i \geq 1$.

We will use a bit (also considered as a symbol or a packet) to represent an entry in a memory bank. Suppose there are R_1 input bits and R_2 requested bits. That is, the number of input and output ports is R_1, R_2, respectively. In this paper we assume they are the same $R_1 = R_2 = R$ unless stated otherwise. And let n be the total number of memory banks, $n \geq R$. We say $n - R$ is the amount of *redundancies*. For the i-th generation, let $U_i = (u_{i,0}, u_{i,1}, \ldots, u_{i,R-1})$ be the R-bit input, and $X_i = (x_{i,0}, x_{i,1}, \ldots, x_{i,n-1})$ the n-bit written vector. The encoding of a switch code is a function f from the input bits to the written bits: $f : \{U_i\}_{i \geq 0} \mapsto \{X_i\}_{i \geq 0}$. Notice that the function can be computed across generations. But as mentioned, we concentrate on functions within a generation. A function can be different from one generation to another. However, there are only a finite number of functions from R input bits to n written bits. If we consider large enough memory banks, we can always find R generations with the same encoding function. Therefore, w.l.o.g., we restrict the encoding function to be identical in every generation: $f : U \mapsto X$, where $U = (u_0, u_1, \ldots, u_{R-1})$ and $X = (x_0, x_1, \ldots, x_{n-1})$ are input and written bits of one generation, $u_i, x_j \in \mathbb{F}_2$ for all $i \in [0, R-1]$, $j \in [0, n-1]$. We call the subscript i of u_i an information or systematic *element*.

The output *request* is $R = R_2$ arbitrary information bits from the previous generations. A *solution* to the request is a way to compute these R bits from no more than one bit from each memory. If we have a solution for bits from R different generations, then since the encoding is identical for all generations, obviously we have a solution for bits from less than R generations. Therefore we can limit our attention to the worst case of bits from R generations, and the order of the bits and generation indices are not of importance. A *request vector* is an ordered R-tuple of indices, $L = (l_0, l_1, \ldots, l_{R-1})$, where the i-th output bit is an element $l_i \in [0, R-1]$, and $l_i \leq l_j$ for all $0 \leq i \leq j \leq R-1$. For example, if $R = 5$ then $L = (0, 2, 2, 2, 3)$ requires one bit from elements $0, 3$ and three bits from element 2. A solution is a partition of the memory banks $[0, n-1]$ into R parts, together with a set of R decoding functions. Denote the decoding partition by $M = ((m_{0,0}, \ldots, m_{0,t_0}), (m_{1,0}, \ldots, m_{1,t_1}), \ldots, (m_{R-1,0}, \ldots, m_{R-1,t_{R-1}}))$, and the decoding functions by g_i, $i \in [0, R-1]$, $u_{l_i} = g_i(x_{m_{i,0}}, x_{m_{i,1}}, \ldots, x_{m_{i,t_i}})$. Since M is a partition, each memory bank is read from no more than once, therefore satisfies the constraint of bandwidth. If a memory bank is not used, we put it in an arbitrary part i, and the decoding function does not use it. An (n, R) *switch code* is the encoding and decoding functions that satisfy the above definitions.

Example 1 *Consider the code in Figure 1 (c). The encoding function is $(x_0, x_1, x_2) = (u_0, u_1, u_0 + u_1)$. If the output is (A, D) then the request vector is $(0, 1)$ and the decoding partition is $((0, 2), (1))$. The decoding functions are $u_0 = x_0 + 0 \cdot x_2$, $u_1 = x_1$. It can be seen that the partition is not unique since we can put memory 2 in either of the two parts. If the output is (A, C) on the other hand, the request vector is $(0, 0)$, the decoding partition is $((0), (1, 2))$ and the decoding functions are $u_0 = x_0$, $u_0 = x_1 + x_2$.*

Theorem 1 *Without coding, or only through replications, the number of memories is at least $n = R_1 R_2$ given R_1 input ports and R_2 output ports.*

We have the above theorem because we can assume that

the encoding function is identical in every generation. In this paper, we will show that through coding, we can decrease the redundancy by a half or two thirds, depending on the parameters. We will focus on *systematic codes* where the information bits are stored in the first R memories, and the rest are parity bits.

One can think of different variations of switch codes, and they can be applied in different queueing scenarios. A special kind of request is *one-burst request*, where only one integer repeats in the vector L. That is, $l_0 < l_1 < \cdots < l_i = l_{i+1} = \cdots = l_j < l_{j+1} < \cdots < l_{R-1}$ for some $0 \le i \le j \le R-1$. The burst length, or the number of repetitions is $j-i+1$, and l_k, $k \notin [i,j]$, are called *singletons*. Another type is *limited-repetition request*, namely each integer repeats at most a certain number of times. A third generalization is *consecutive-generation request*, where the requested bits come from up to t consecutive generations, $t \le R$. In this case we will use t vectors to represent requests from each generation, and the sum of the vector lengths is R.

III. Codes with Pair (degree 2) Parities

In this section we construct switch codes with $n = (R_1+1)\lceil R_2/2 \rceil$. First we solve for the case of $R = R_1 = R_2$ and then for the general case. In addition, we construct codes for consecutive-generation requests.

Construction 1 *Let $R = R_1 = R_2$. The code is constructed as the information bits and XOR of all pairs of information bits. Figure 1 is an example of this code for $R = 2$. When $R = 3$, the code is $X = (u_0, u_1, u_2, u_0+u_1, u_0+u_2, u_1+u_2)$.*

Theorem 2 *The $(n = R(R+1)/2, R)$ switch code in Construction 1 can solve any request.*

Proof: We solve every information bit u_i by either read it directly or by XOR of u_j and u_i+u_j, for some $j \ne i, j \in [0, R-1]$. Each of the R systematic elements will be used to obtain one information bit. For two different systematic elements i, j, the corresponding parities (if used) are also different. Otherwise this parity is simply XOR of x_i, x_j and we can read them directly. Thus all memories used are disjoint. ∎

Systematic codes whose parities are computed from pairs can be represented by an undirected graph $\mathcal{G}$. Here information elements are vertices and parity of pairs are edges. The number of edges indicates the redundancy. We call such codes *systematic parity-pair* codes. The above construction corresponds to the complete graph K_R.

The following is a construction for different R_1 and R_2 values. When $R = R_1 = R_2$ is even, the redundancy is the same as Construction 1, but reconstruction requires XOR of a larger number of bits.

Construction 2 *First consider the (R_1+1, R_1) parity code $X = (u_0, \ldots, u_{R_1-1}, x_{R_1})$ with $x_{R_1} = \sum_{i=0}^{R_1-1} u_i$. Any request of size 2 can be solved in X: one bit can be read directly, the other bit can be solved through all the rest elements, or read directly. Second duplicate $\lceil R_2/2 \rceil$ times the code X. Thus any request of size R_2 can be solved and the number of memories is $n = (R_1+1)\lceil R_2/2 \rceil$.*

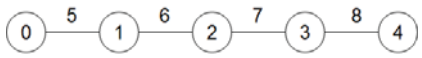

Figure 2. Graphic representation of a $(9,5)$ code. It solves all 2-consecutive-generation requests.

Next we consider requests from t consecutive generations, $t \le R$. Clearly such requests are also t-repetition-limited.

Lemma 3 *A lower bound for the number of redundancies of a systematic pair-parity code is $n - R \ge \lceil R(t-1)/2 \rceil$, when a request is restricted to t consecutive generations.*

Proof: A necessary condition for the existence of a solution for t requests from the same element is this element being contained in at least t memory banks. Or the degree of each vertex in $\mathcal{G}$ should be at least $t-1$. ∎

Theorem 4 *Construct the $(2R-1, R)$ switch code as*

$$x_i = \begin{cases} u_i, & i \in [0, R-1] \\ u_{i-R} + u_{i-R+1}, & i \in [R, 2R-2] \end{cases}$$

Then this code solves any 2-consecutive-generation request, and has optimal redundancy for a systematic pair-parity code.

See proof in the detailed version of this paper [7]. This construction can be associated with a graph $\mathcal{G}$ that is a line connecting all elements. Figure 2 shows an example of this code. The labels of the vertices and edges are indices of the memory bank. Suppose the request is $(0,2)$ from the first generation and $(2,3,4)$ from the second. Then it can be solved by reading $0, 2$ of the first generation, and $1, 6, 7, 8$ of the second.

IV. Codes Based on Block Designs

In this section we generalize the above section and construct switch codes by XORing triples of information bits. We first derive a lower bound on the redundant bits with respect to the degree of parity bits (number of bits XORed). Then construct codes with $n = R(R+2)/3$ based on block designs that solve one-burst requests. Notice that Theorem 1 still applies for one-burst requests, so our codes reduce the redundancy to $1/3$ compared to non-coding schemes.

Consider a bipartite graph with variable vertices (information bits) and check vertices (parity bits). If an information bit is added to a parity bit, there is an edge between them. If the graph has constant degree d, or every parity XORs d information bits, similar to Lemma 3 we have the following lower bound of redundancy.

Lemma 5 *A systematic switch code with constant degree parities satisfies $n - R \ge R(R-1)/d$.*

This lemma shows that a systematic code may have less redundancies with larger parity degree. Next we will construct codes with degree $d = 3$ and redundancy $R(R-1)/3$.

In Construction 1, an output bit u_i is solved by either itself or XOR of a parity bit $u_i + u_j$ and an information bit u_j.

Similarly, when the parity has degree 3, we will solve a bit u_i by either itself or XOR of two parity bits $u_i + u_j + u_k$, $u_j + u_k + u_l$, and an information bit u_l. Associate each parity bit with the block (or subset) of information elements it involves. In this section, the notations of the binary sum and the block will both represent a parity bit. For example, the bits $u_i + u_j + u_k$, $u_j + u_k + u_l$ are associate with $\{i, j, k\}$, $\{j, k, l\}$, respectively, then we see an intersection of size two between these two blocks. In other words, the pair $\{j, k\}$ appears in both of these blocks. We say the bit i can be solved by

$$\{l\}, \{i, j, k\}, \{j, k, l\}. \tag{1}$$

We call these three bits systematic and parity *helpers* of the output element i.

A *balanced incomplete block design* is a system with b blocks of size k and a total of R elements. Moreover, it requires that every element repeats r times and every subset of size t appears exactly λ times. A *triple system* with $\lambda = 2$ is a balanced incomplete block design such that each block contains 3 elements out of a total of R elements, and every pair of elements appear exactly twice in the blocks. Since the code is systematic and a request might require R bits from the same element, every element repeats $R-1$ times in the blocks. By counting we know there should be a total of $R(R-1)/3$ parity or redundant bits. A switch code does not necessarily corresponds to a block design but we will show that this approach gives us some nice constructions.

Example 2 *Consider the following triple system with* $R = 6$ *elements and* $R(R-1)/3 = 10$ *parity blocks:* $\{0,1,2\}, \{0,2,3\}, \{0,1,4\}, \{1,2,5\}, \{0,3,5\}, \{2,3,4\}, \{0,4,5\}, \{1,4,3\}, \{1,5,3\}, \{2,5,4\}$. *Every element repeats* $R-1 = 5$ *times. Let a switch code contain the* 6 *systematic bits and* 10 *parity bits. Suppose the request is* $(0,0,0,0,0,0)$, *then we can solve it in the following way:* $\{0\}; \{1\}, \{0,3,5\}, \{1,5,3\}; \{2\}, \{0,4,5\}, \{2,5,4\}; \{3\}, \{0,1,4\}, \{1,4,3\}; \{4\}, \{0,2,3\}, \{2,3,4\}; \{5\}, \{0,1,2\}, \{1,2,5\}$. *We can see that by using one bit from all of the memories, we are able to output element 0 six times. Similarly, one can check that it is possible to output any element six times. And any one-burst request can be solved by using solutions excluding the singletons. For example, for* $(0,0,0,1,4,5)$ *we can read singletons* $1,4,5$, *and use* $2,3$ *as systematic helpers for the burst. Therefore, this code serves any one-burst request.*

The above example shows that by using triples instead of pairs as parities, we decrease the number of redundancies from 15 to 10 when $R = 6$. Next we will induce three different kinds of switch codes from triple systems.

Top-down Construction.

The main idea of the top-down construction is to break blocks of size 4 into triples. Let us start with an example. Consider the block $\{1,2,3,4\}$ of size 4 and its subsets of size 3:

$$\{1,2,3\}, \{1,2,4\}, \{1,3,4\}, \{2,3,4\}. \tag{2}$$

Take any two triples out of these four, we get two parities that look exactly the same as those in (1). In other words, the subsets of block of size 4 are good candidates to construct our triple system. To output element i, we can use j as the systematic helper, for any $j \neq i$. And the corresponding parities are

$$\{1,2,3,4\} \backslash \{j\} \tag{3}$$

$$\{1,2,3,4\} \backslash \{i\} \tag{4}$$

It should be noted that no matter which j we use, (4) is always a parity helper. In other words, among the three elements j, $j \neq i$, we can only use one of them as a systematic helper.

Suppose we have a balanced incomplete block design D_1 with R elements, $R(R-1)/12$ blocks of size 4. Moreover every element repeats $(R-1)/3$ times and every pair appears once. We call such systems quadruple systems. For simplicity, assume $(R-1)/3$ is an integer. Now construct another block design D by changing every block in D_1 into four blocks of size 3. For example, if $\{1,2,3,4\}$ is a block in D_1, then triples in (2) all belong to D. It is easy to check that D is a triple system with $R(R-1)/3$ blocks, $R-1$ repeats, and every pair appears exactly twice.

Example 3 *Take* D_1 *as the block design with* $R = 25$ *elements and* 50 *blocks of size 4 (see [2]). The design is formed by dicyclic solution in two families:* $\{a_1, a_2, b_1, e_5\}$, $\{a_1, a_3, c_1, d_4\}$. *All the blocks are generated by cyclic shift on the letters and then on the subscripts. Then we label the elements from 0 to 24 and break blocks down to* 200 *triples.*

We have an output-helper graph like Figure 3. In this graph, every vertex is a systematic element. A directed edge (i, j) means that j is a possible systematic helper of i. Since every pair (i, j) appears in D_1, every output i can use any systematic helper $j \neq i$. So the graph is actually complete. Figure 3 only illustrates the edges used in a decoding procedure. We group three helpers (like $1, 5, 24$) to emphasize that only one can be used. We call every group a *helper group* or a *group* for a given output element. Once we choose a systematic helper from a group, the edge corresponds to two parity helpers. For example, if output is 0 and helper is 1, then the edge between them means parities $\{1, 5, 24\}$, $\{0, 5, 24\}$.

The following theorem states that if we have a quadruple system D_1 then we have a switch code serving one-burst requests, where the size of burst is a third of ports. In fact, D_1 exists for all R such that $R \equiv 1, 4 \bmod 12$ (see e.g. [1]).

Theorem 6 *The top-down construction serves any one-burst request with burst length* $(R-1)/3 + 1$.

The proof can be found in [7]. We will use Example 3 to illustrate the decoding algorithm. Given a request vector $(0, \ldots, 0, a_9, \ldots, a_{24})$ where $a_0 = 0$ is repeated $(R-1)/3 + 1 = 9$ times, and the rest are singletons. Besides $\{0\}$ itself, choose one systematic helper $h_1, \ldots, h_8$ from each group to solve 0 as follows. Initialize a set T as the elements not requested $T = \{a_1, \ldots, a_8\}$. For all $i \in [8]$, if there exists a member of T in group i, assign it as h_i and remove it from T. For the rest groups, choose arbitrarily an element from the group as h_i. And use one element (w.l.o.g. assume element a_i)

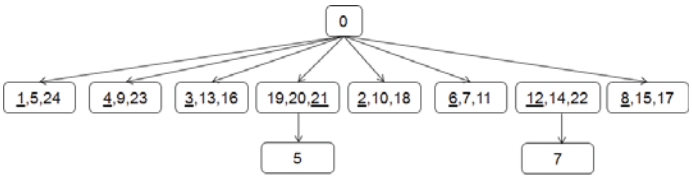

Figure 3. Output a burst of length 9. Assume $(0,\ldots,0,9,10,\ldots,24)$ is the request and $T=\{1,2,\ldots,8\}$ are unrequested elements. Use the underlined $(h_1,h_2,\ldots,h_8)=(1,4,\ldots,8)$ as systematic helpers for the burst of 0. If a requested singleton is not underlined, simply read it. Otherwise use a_i as the helper of h_i, e.g., the singleton 21 has the systematic helper 5.

from T as a systematic helper for h_i, because h_i is a requested singleton and $a_i \neq h_i$. See Figure 3 for an example.

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 k, then by breaking it down to blocks of size $k-1$, one may use one systematic helper and two parity helpers to output a bit. Meanwhile since the parity has higher degree, we may expect to get smaller redundancy.

Linear Construction.

The idea is to have a simple way to decide the pair j,k in (1) given the output i and the systematic helper l. Then try to see weather we can use each triple as many times as possible, i.e., three times for different outputs. One of the simplest mapping from (i,l) to (j,k) would be a linear function. The resulting design has the same parameters as the previous constructions, namely $(R-1)/3$ triples and every pair appears twice.

To emphasize that j,k are functions of i,l, we rename $f=j, g=k$. We are going to define the pair f,g by the following linear mapping:

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

In other words, $f=f(i,l), g=g(i,l)$ are functions of i,l. When we omit the arguments, we write f,g to denote $f(i,l), g(i,l)$, respectively. Since every pair should appear in the design, we would like the mapping to be bijection. So matrix A should be invertible. In fact, due to the symmetry of the roles of i and l, if we switch i and l, the resulting pair should be g,f. In other words, $a=d, b=c$.

Thus we have triples $\{i,f,g\},\{l,f,g\}$ as blocks. We would like to use each triple 3 times for different values of i,l.

$$(i,f,g)=(g(i',g),f(i',g),g)=(f(i'',f),f,g(i'',f)),$$
$$(l,f,g)=(f(f,l'),f,g(f,l'))=(g(g,l''),f(g,l''),g).$$

Here all the triples are ordered. The values of a,b,c,d should guarantee that there exist solutions of i',i'',l',l'' given arbitrary i,l. The solutions to the above equations are: $a=d=\frac{1}{2}+\frac{\sqrt{-3}}{6}, b=c=\frac{1}{2}-\frac{\sqrt{-3}}{6}$, or exchange a,d with b,c, which is equivalent. It can be checked that this ensures the matrix A to be invertible. We will choose the number of elements $R>3$ to be a prime number such that -3 mod R is a perfect square. We note here that by the formula for the Legendre symbol [4] which indicates whether -3 is a perfect square, this condition is *equivalent to* R being a prime and $(R-1)/3$ being an integer.

For example, let $R=7$, then $a=d=2, b=c=6$ and one can check that it actually gives a triple system of 14 blocks with every pair appearing twice.

Theorem 7 *The linear construction serves any one-burst request and has $R(R-1)/3$ redundancies, when $R>3$ is a prime and -3 mod R is a perfect square.*

The main idea of the proof is to show that disjoint systematic helpers correspond to disjoint parity helpers (see [7]). This construction has the advantage of no constraint on the burst length. With any of the above constructions one can build switch codes that serve burst requests. As mentioned, if we have the ability to solve burst requests, we can lower the worst-case delay in the memories.

V. Conclusions

In this paper we proposed the switch code problem, that is, how to use coding techniques to solve unbalanced requests in memories of network switches. We constructed pair-parity codes with $R(R-1)/2$ redundancies and those with $R(R-1)/3$ redundancies using different ideas of block designs. Better understanding on block designs can result in good switch codes. At the same time, good switch codes will be good combinatorial designs with certain parameters.

The subject of switch code still requires more studies. For example a lower bound on redundancy without degree constraint is unknown, and a network-coding type of argument may lead to interesting results. It is also our future work to study constructions with more than three elements in a parity, and also codes with different degrees of the parities. Finally, coding across generations may lead to even smaller redundancy.

Acknowledgement

The authors thank the anonymous referee for pointing to the Reed Muller code parameters that give asymptotically efficient switch codes. This work was supported in part by the Marie Curie CIG grant and by the Intel ICRI-CI center. It is also partially supported by NSF grant CIF-1218005, BSF grant 2010075, and a gift from Northrop Grumman.

References

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

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

[3] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of codeword symbols," *Information Theory, IEEE Transactions on*, vol. 58, no. 11, pp. 6925–6934, nov. 2012.

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

[5] F. Oggier and A. Datta, "Self-repairing homomorphic codes for distributed storage systems," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 1215–1223.

[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] Z. Wang, O. Shaked, Y. Cassuto, and J. Bruck, "Codes for network switches." [Online]. Available: http://paradise.caltech.edu/etr.html

[8] S. Yekhanin, "Locally decodable codes," *Computer Science–Theory and Applications*, pp. 289–290, 2011.