Coding for Improved Throughput Performance in Network Switches

Rami Cohen, *Graduate Student Member, IEEE,* and Yuval Cassuto, *Senior Member, IEEE*

***Abstract*—Network switches and routers need to serve packet writes and reads at rates that challenge the most advanced memory technologies. As a result, scaling the switching rates is commonly done by parallelizing the packet I/Os using multiple memory units. For improved read rates, packets can be coded upon write, thus giving more flexibility at read time to achieve higher utilization of the memory units. This paper presents a detailed study of coded network switches, and in particular how to design them to maximize the throughput advantages over standard uncoded switches. Toward that objective the paper contributes a variety of algorithmic and analytical tools to improve and evaluate the throughput performance. The most interesting finding of this study is that the placement of packets in the switch memory is the key to both high performance and algorithmic efficiency. One particular placement policy we call "design placement" is shown to enjoy the best combination of throughput performance and implementation feasibility.**

I. Introduction

With the increasing demand for network bandwidth, network switches (and routers) face the challenge of serving growing data rates. Currently the most viable way to scale switching rates is by parallelizing the writing and reading of packets between multiple memory units (MUs) in the switch fabric. However, this introduces the problem of *memory contention*, whereby multiple requested packets need to access the same bandwidth-limited MUs. Our ability to avoid such contention in the write stage is limited, as the reading schedule of packets is not known upon arrival of the packets to the switch. Thus, efficient packet placement and read policies are required, such that memory contention is mitigated.

For greater flexibility in the read process, coded switches introduce *redundancy* to the packet-write path. This is done by calculating additional coded chunks from an incoming packet, and writing them along with the original packet chunks to MUs in the switch memory. A coding scheme takes an input of k packet chunks and encodes them into a codeword of n chunks ($k \leq n$), where the redundant $n-k$ chunks are aimed at providing improved read flexibility. Thanks to the redundancy, only a subset of the coded chunks is required for reconstructing the original (uncoded) packet. Thus, packets may be read even when only a part of their chunks is available to read without contention. One natural coding approach is to use $[n,k]$ *maximum distance separable* (MDS) codes, which have the attractive property that *any* k chunks taken from the n code chunks can be used for the recovery of the original k packet chunks. Although MDS codes provide the maximum flexibility, we show in our results that good switching performance can be obtained even with much weaker (and lower cost) codes, such as binary cyclic codes.

In the coded switching paradigm we propose in this paper, our objective is to maximize the number of full packets read from the switch memory simultaneously in a read cycle. The packets to read at each read cycle are specified in a request issued by the control plane of the switch. As we shall see, coding the packets upon their write can significantly increase the number of read packets, in return to a small increase in the write load to store the redundancy. Thus coding can significantly increase the overall switching throughput. In this paper we identify and study two key components for high-throughput coded switches: 1) *Read algorithms* that can recover the maximal number of packets given an arbitrary request for previously written packets, and 2) *Placement policies* determining how coded chunks are placed in the switch MUs. Our results contribute art and insight for each of these two components, and more importantly, they reveal the tight relations between them. At a high level, the choice of placement policy can improve both the *performance* and the *computational efficiency* of the read algorithm. To show the former, we derive a collection of analysis tools to calculate and/or bound the performance of a read algorithm given the placement policy in use. For the latter, we show a huge gap between an NP-hard optimal read problem for one policy (*unrestricted* placement), and extremely efficient optimal read algorithms for two others (*cyclic* and *design* placements).

The use of coding for improved memory read rates joins a large body of recent work aimed at objectives of a similar flavor, see e.g. the survey in [1]. In [2], [3], the effect of MDS coding on content download time was analyzed for two content access models, where an improvement in performance was achieved. In [4], latency delay was reduced by choosing MDS codes of appropriate rates. Latency comparison between a simple replication scheme and MDS codes was pioneered by Huang et al. [5] using queuing theory. It was shown that for $k = 2$, the average latency for serving a packet decreases significantly when a certain scheduling model is used. This analysis was later extended by Shah et al. in [6], [7], where bounds on latency performance under multiple scheduling policies were investigated. In [8] and then in [9], switch coding is done under a strong model guaranteeing simultaneous reconstruction of worst-case packet requests.

This paper is structured as follows. In Section II, we provide the *switch setting* and define formally the problem of

The authors are with the Andrew and Erna Viterbi Faculty of Electrical Engineering, Technion – Israel Institute of Technology, Haifa, Israel (email: rc@campus.technion.ac.il, ycassuto@ee.technion.ac.il)

Parts of this work were presented at the 2015 IEEE International Symposium on Information Theory (ISIT), Hong Kong, China and at the 2016 IEEE International Symposium on Information Theory (ISIT), Barcelona, Spain.

This work was supported in part by the Israel Science Foundation, by the Intel ICRI-CI center, and by the Israel Ministry of Science and Technology.

maximizing the read throughput. We choose a simple model of a shared-memory switch, which allows defining a clean *throughput-optimization* problem. It is important to note that all the paper's results demonstrated on this simple model can be extended to more realistic setups, with the same underlying ideas at play. In Section III, we define a *full-throughput* instance as one in which the switch is able to read all the requested packets in the same read cycle. Full-throughput instances are the most desired operation mode for a switch, because there is no need for queueing unfulfilled packet requests. We derive necessary and sufficient conditions for an instance to be full-throughput, and specify placement policies motivated by these conditions. Read algorithms are provided in Section IV for maximizing the instantaneous throughput at a read cycle. For the cyclic and design placements we show efficient polynomial time optimal read algorithms, which are also practical enough to implement in a switch environment. Probabilistic analysis of the average read throughput is provided in Section V. We derive an upper bound for the unrestricted placement, a lower and an upper bound for the cyclic placement, and exact full-throughput analysis for the design placement. Simulations results are given in Section VI. Finally, the paper is concluded in Section VII.

II. Problem Setting and Formulation

Given our objective to improve the switch throughput, we now define the system setting for our proposed solution, and pose the throughput-maximization problem within this setting.

A. The switch setting

Consider a switch composed of N parallel memory units (MUs) serving writes and reads of incoming and outgoing packets, respectively. Each MU is capable of storing a certain number of bits on a write cycle, and retrieving this number of bits on a read cycle. A data packet might be too large to fit in a single MU. Thus, an incoming packet is partitioned into k chunks of the same size, each stored in a distinct MU on the same write cycle. Upon read request of a packet, all k chunks of the packet need to be retrieved by the k MUs storing it, after which it can be delivered to the output port. Because there are multiple packet requests pending on the same MUs simultaneously, contention may occur between chunks of different packets stored in the same MUs.

To reduce the amount of contention in packet reading, we propose in this paper to encode the incoming packets with an $[n, k]$ code, which means that the k chunks of the data packet are encoded to $n \geq k$ chunks. The n fixed-size encoded chunks are stored in n distinct MUs out of the N MUs in the system ($1 \leq k \leq n \leq N$). Between packets overlap is allowed, i.e., chunks of two or more packets may share one or more MUs. For the code we mostly[1] assume the maximum distance separable (MDS) property, which means that any subset of k chunks of the n encoded chunks is sufficient for recovering the original packet. We mention here the Reed-Solomon (RS) codes [10], [11], which are an important family of MDS codes widely used in storage systems for improved reliability. An RS code exists for every choice of $k \leq n \leq q$, where q is the code alphabet size, which is a prime power. RS encoding/decoding can be performed efficiently [10], [12].

In a typical switch, a large number of packets is stored in memory at any given time. Out of these many packets, L particular packets are requested at each read cycle. To maximize the read throughput, at each read cycle the switch needs to recover a maximal number of the L requested packets. We next define this throughput maximization problem formally.

B. The maximal-throughput read problem

A request arrives for L packets, with the objective to read as many out of these packets in a single read cycle. The locations of each packet's chunks are known, and we wish to find methods for reading as many packets as possible simultaneously, with the constraint that each MU can be accessed *only once* in a read cycle, delivering at most one chunk. An instance of the problem is illustrated in Fig. 1, where encoded data chunks of multiple packets appear in the same column representing an MU. Let us denote by L^* the maximal number of packets that can be read, out of the L packets requested from the switch memory. We consider the following notion of throughput as a performance measure.

Definition 1: (Instantaneous Throughput) The instantaneous throughput ρ of the system is defined as

$$\rho = \frac{L^* k}{N}. \tag{1}$$

That is, ρ is the fraction of active MUs serving packets out of the N MUs in the system, and it is a monotonically increasing function of L^*. Clearly $0 \leq \rho \leq 1$, because the total number of read chunks cannot be more than N. Note that given L, maximizing the instantaneous throughput is equivalent to maximizing L^*, because k and N are constants. In the sequel we refer to the instantaneous throughput as simply *throughput*. Later in the paper we also discuss the *average throughput* $\bar{\rho}$, defined as the value of ρ averaged over read cycles.

We name the problem of maximizing the throughput ρ as the $[n, k]$-*maximal throughput problem*, or nkMTP. Recall that for reading a packet, k MUs are required, which are not used to read chunks of any other packet. Thus, an nkMTP solution amounts to finding the maximal number of *disjoint* k-sets, leading to the following set-theory formulation of nkMTP. Consider the N MUs as the elements of the set $\mathcal{S} = \{0, 1, 2, ..., N-1\}$. Each packet $i = 1, 2, ..., L$ is stored in MUs indexed by a subset $\mathcal{S}_i$ of $\mathcal{S}$, where $|\mathcal{S}_i| = n$ and the subsets may overlap. Then nkMTP can be formulated as follows.

Problem 1: (nkMTP)

Input: The set $\mathcal{S} = \{0, 1, 2, ..., N-1\}$ and L subsets of $\mathcal{S}$, $\mathcal{S}_i \subseteq \mathcal{S}$, such that $|\mathcal{S}_i| = n$.

Output: Subsets $\mathcal{S}'_i \subseteq \mathcal{S}_i$ such that $|\mathcal{S}'_i| = k$, $\mathcal{S}'_i \cap \mathcal{S}'_j = \emptyset$ $(i \neq j)$ and the number of subsets is maximal.

Example 1: Consider an nkMTP instance with $N = 5, L = 3$ and $n = 3$, where the packets are stored in the MUs indexed

[1]Part of our results in the sequel do not require the code to have such a strong property.

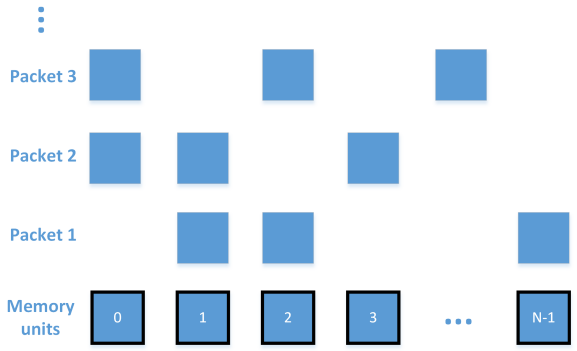


Fig. 1: Illustration of nkMTP ($n = 3$).

by the sets $\mathcal{S}_1 = \{0, 1, 2\}, \mathcal{S}_2 = \{1, 3, 4\}, \mathcal{S}_3 = \{2, 3, 4\}$. If $k = n = 3$, at most one packet can be read, since $\mathcal{S}_i \cap \mathcal{S}_j \neq \emptyset$ for $i, j \in \{1, 2, 3\}$. If $k = 2$, a possible solution is $\mathcal{S}'_1 = \{0, 1\}$ and $\mathcal{S}'_2 = \{3, 4\}$ with $L^* = 2$. Finally, if $k = 1$ all the packets can be read, and one possible solution is $\mathcal{S}'_1 = \{0\}, \mathcal{S}'_2 = \{1\}$ and $\mathcal{S}'_3 = \{2\}$.

An nkMTP instance can be represented as a graph as well. Consider a *bipartite* graph $G = (X_G, Y_G, E_G)$, where X_G and Y_G are the two disjoint sets of vertices of G, and E_G is the set of edges of G. Thinking of X_G as packets and of Y_G as MUs, a vertex $x \in X_G$ is connected to a vertex $y \in Y_G$ if one of the encoded chunks of the packet x is stored in the MU y. In Fig. 2, Example 1 is represented on a graph. This graph interpretation will be used later to obtain further insights on the problem. An algorithm that guarantees maximal throughput for any instance is called an *optimal read algorithm*. A straightforward approach for solving an nkMTP instance is to consider all possible assignment configurations of MUs to packets. However, this approach is clearly inefficient as its complexity scales exponentially with L. We observe that polynomial-time optimal read algorithms exist in the general case for specific values of k and n. These read algorithms are obtained by interpreting nkMTP for these parameters as known *graph matching* problems whose efficient solutions are known.

Theorem 1: For $k = 1, n \geq 1$ or $k = n = 2$, nkMTP is solvable in polynomial time.

Proof Consider a graph representation G of an nkMTP instance. When $k = 1, n \geq 1$, maximizing the throughput is equivalent to finding a *maximum bipartite matching* [13] in G. That is, a subgraph of G with the largest number of *matched* pairs (x, y), $x \in X_G, y \in Y_G$, such that each pair is connected by an edge and the edges are pairwise non-adjacent. When $k = n = 2$, consider the N MUs as the vertices of a (unipartite) graph, where an edge in this graph connects two MUs shared by the same packet. A maximum matching in this graph will provide the largest number of disjoint pairs of MUs, each pair serving a packet, corresponding to a maximum-throughput solution. Efficient maximum-matching algorithms are known in both cases [13]. ∎

In practice, larger k and n values might be of interest. However, nkMTP turns out to be NP-hard in this case, as shown in the following theorem.

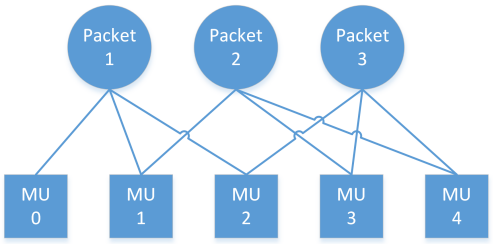


Fig. 2: nkMTP from Example 1 formulated on a graph. There are three packets, each stored as $n = 3$ encoded chunks in n MUs.

Theorem 2: nkMTP is NP-hard for $3 \leq k \leq n$.

To prove Theorem 2, we *reduce* the l-set packing (l-SP) problem [14], known to be NP-hard, to nkMTP. In l-SP, there are L sets, each of size l, and the problem is to find the maximal number of pairwise disjoint sets. By the reduction, we basically show that an l-SP instance can be transformed to an nkMTP instance with any k and n in the range, which implies that nkMTP is at least as hard as l-SP. The details of the reduction are provided in Appendix A. The consequence of the hardness result of Theorem 2 is that no efficient optimal algorithms are expected to be found for solving (an arbitrary instance of) nkMTP when $3 \leq k \leq n$.

Surprisingly, this hardness result does not imply the intractability of optimal coded switching. The main observation we make in this work is that *clever chunk placement* at the write path can yield more structured nkMTP instances, which do admit efficient optimal read algorithms. In the rest of this paper we develop algorithmic and analytic tools that reveal the interesting interplay in coded switches between packet placement, computation efficiency, and throughput performance.

III. Full-Throughput Conditions and Placement Policies

In this section, we start with providing necessary and sufficient conditions for a *full-throughput* solution, i.e., $L^* = L$ read packets. This is desired in practice to avoid delaying or reordering the read packets before fulfilling the read request. These conditions will be used later toward specifying packet placement policies, and analyzing the performance of read algorithms. Subsequently, we define the three policies this paper considers for placing packets in the switch memory: *unrestricted*, *cyclic*, and *design*.

A. Full-throughput conditions

To find a *necessary* condition for the existence of a full-throughput solution, note that each read packet requires at least k MUs not used by any other packet. Thus, at least kL MUs must be *covered* by the requested packets, such that the following inequality

$$\left| \bigcup_{i=1}^{L} \mathcal{S}_i \right| \geq kL \tag{2}$$

must hold in any nkMTP instance with a full-throughput solution. We refer to (2) as the *coverage condition*. Note that when $k = n$ this condition (with equality) becomes sufficient as well, as the condition implies in this case that there is no contention between packets. We now move to find a *sufficient* condition for the existence of a full-throughput solution. Let us extend the set notation to represent intersections of MU sets, that is, $\mathcal{S}_{\mathcal{I}} \triangleq \bigcap_{j\in\mathcal{I}} \mathcal{S}_j$, for $\mathcal{I} \subseteq \{1, 2, ..., L\}$.

Theorem 3: Let $\mathcal{S}_1, ..., \mathcal{S}_L$ ($L \geq 2$) be the MU sets of an nkMTP instance. Then an $L^* = L$ solution exists if

$$\forall i, j : i \neq j,\ |\mathcal{S}_i \cap \mathcal{S}_j| \leq \frac{2(n-k)}{L-1} \triangleq t_{\max}. \tag{3}$$

Proof Denote by $\Phi_{s,\mathcal{L}}$ the sum of cardinalities of intersections of s distinct sets taken from the MU sets indexed by a certain set $\mathcal{L}$

$$\Phi_{s,\mathcal{L}} = \sum_{\mathcal{I}\subseteq\mathcal{L},|\mathcal{I}|=s} |\mathcal{S}_{\mathcal{I}}|\,. \tag{4}$$

As an example, if $\mathcal{L}$ is the set $\{1, 2, 3\}$, then $\Phi_{2,\mathcal{L}}$ is $|\mathcal{S}_1 \cap \mathcal{S}_2| + |\mathcal{S}_1 \cap \mathcal{S}_3| + |\mathcal{S}_2 \cap \mathcal{S}_3|$. As we saw in Section II-B, an nkMTP instance can be represented as a bipratite graph with the packets and MUs being the disjoint vertex sets. In this representation, packet vertices need to be matched to disjoint sets of k MU vertices. According to the extended Hall's theorem [15], all the L packet vertices can be matched (i.e., an $L^* = L$ solution exists) if and only if

$$\left|\bigcup_{j\in\mathcal{L}} \mathcal{S}_j\right| \geq k\,|\mathcal{L}| \tag{5}$$

for every subset $\mathcal{L} \subseteq \{1, 2, ..., L\}$. In words, at least $k|\mathcal{L}|$ distinct MUs should be present in each $\mathcal{L}$ sub-family of the L MU sets. Using the inclusion-exclusion principle, (5) is equivalent to the requirement

$$n|\mathcal{L}| - \Phi_{2,\mathcal{L}} + \sum_{s=3}^{|\mathcal{L}|} (-1)^{s-1}\Phi_{s,\mathcal{L}} \geq k|\mathcal{L}| \tag{6}$$

for every $\mathcal{L} \subseteq \{1, 2, ..., L\}$, $|\mathcal{L}| \geq 2$ (for $|\mathcal{L}| = 1$, (6) reduces to the requirement $n \geq k$ that always holds). The sum $\sum_{s=3}^{|\mathcal{L}|} (-1)^{s-1}\Phi_{s,\mathcal{L}}$ is non-negative, as it compensates for over-subtraction of pairwise intersection cardinalities in the inclusion-exclusion process. Therefore, (6) holds if the inequality

$$\Phi_{2,\mathcal{L}} \leq |\mathcal{L}|\,(n-k) \tag{7}$$

holds for every $\mathcal{L}$. We can bound $\Phi_{2,\mathcal{L}}$ by bounding the pairwise intersection cardinalities

$$\Phi_{2,\mathcal{L}} = \sum_{i\neq j\subseteq\mathcal{L}} |\mathcal{S}_i \cap \mathcal{S}_j| \leq \binom{|\mathcal{L}|}{2} \max_{i\neq j\subseteq\mathcal{L}} |\mathcal{S}_i \cap \mathcal{S}_j|\,. \tag{8}$$

Finally, combining (8) and (7), the inequality (7) holds when

$$\max_{i\neq j\subseteq\mathcal{L}} |\mathcal{S}_i \cap \mathcal{S}_j| \leq \frac{2(n-k)}{|\mathcal{L}|-1}. \tag{9}$$

We now observe that the condition of the theorem (3) implies (9) because $|\mathcal{L}| \leq L$ for every $\mathcal{L}$. ∎

We refer to condition (3) as the *pairwise condition*. The full-throughput coverage and pairwise conditions above will serve us later for specifying placement policies and analyzing their throughput performance. The sufficient pairwise condition will give lower bounds on average throughput, and the necessary coverage condition will give upper bounds. We next turn to specify three placement policies for the switch write path: the *unrestricted*, *cyclic* and *design* placements. In subsequent sections these placement policies are given efficient read algorithms and performance analysis.

B. Unrestricted placement

In the first placement policy we consider, the n chunks of a packet may be placed in any set of n MUs taken from the N MUs in the system. That is, the set of a packet MU indices can be one of the $\binom{N}{n}$ n-subsets of $\mathcal{S} = \{0, 1, ..., N-1\}$. We term this policy as *unrestricted* placement, but note that no probability distribution is assumed. This placement policy is the most general as no structure is imposed on the placement of packet chunks to memory.

Example 2: Assume that $N = 5$ and $n = 3$. There are $\binom{N}{n} = 10$ possible MU sets when the unrestricted placement policy is used: $\{0, 1, 2\}$, $\{0, 1, 3\}$, $\{0, 1, 4\}$, $\{0, 2, 3\}$, $\{0, 2, 4\}$, $\{0, 3, 4\}$, $\{1, 2, 3\}$, $\{1, 2, 4\}$, $\{1, 3, 4\}$ and $\{2, 3, 4\}$.
This placement policy is convenient to implement, because it has maximal flexibility to choose MUs to write based on load and available space. However, this comes with a price, as solving efficiently an arbitrary unrestricted placement instance amounts to solving nkMTP, shown to be NP-hard in Section II. Therefore, in the rest of this section we propose two additional placement policies, which will be shown later to admit efficient optimal read algorithms.

C. Cyclic placement

In the second placement policy we propose, termed as *cyclic* placement, we add a structure constraint on the MUs chosen to store packet chunks. The constraint is that the possible MU sets are composed of n *cyclic consecutive* MU indices. The number of possible MU sets is N (assuming that $n < N$), which is smaller in all non-trivial cases than the $\binom{N}{n}$ sets in the unrestricted placement. An MU set in a cyclic instance can be conveniently thought of as an *arc* covering n cyclic consecutive points out of N points on a circle, where the points are considered as MUs. An example for a circle-arc representation of a cyclic instance is shown in Fig. 3.

Example 3: Assume that $N = 5$ and $n = 3$. There are $N = 5$ possible MU sets when the cyclic placement policy is used: $\{0, 1, 2\}, \{1, 2, 3\}, \{2, 3, 4\}, \{3, 4, 0\}$ and $\{4, 0, 1\}$. Note that these sets are contained in the sets of Example 2.

A further restriction of the cyclic placement policy gives a simple placement policy where the N MUs are statically partitioned to N/n disjoint sets of n consecutive MUs (assuming that n divides N), and each packet is restricted to one of these sets. In this case, the packets are restricted to MU sets that are mutually disjoint. However, using the full cyclic (non partitioned) placement is beneficial for increased flexibility at the read path.

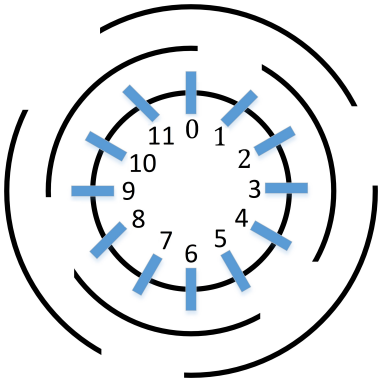


Fig. 3: A cyclic instance in a circle-arc representation. The marks on the inner circle represent $N = 12$ MUs, where the $L = 6$ outer arcs represent packets stored each in $n = 4$ cyclic consecutive MUs.

D. Design placement

In the third policy we consider, our aim is to guarantee a full-throughput solution, i.e., $L^* = L$ read packets. Motivated by the sufficient condition of Theorem 3, we propose to construct a collection of MU sets with overlap at most $t_{\max} = 2(n-k)/(L-1)$, using *combinatorial block designs*. To find such MU sets, we use the so called t-designs [16] with carefully chosen parameters. A t-(N, n, λ) design consists of n-element subsets (*blocks*) taken from a set of N elements, such that every t elements taken from the set appear in exactly λ subsets. 2-designs are of particular interest in the literature, and they are known as *balanced incomplete block design* (BIBD).

While it is not a trivial problem to construct combinatorial designs with arbitrary parameters, many design families are known within the vast literature on this topic [17], [18]. When $\lambda = 1$, t-designs are known as *Steiner systems*, and they contain (when exist) $b = \binom{N}{t}/\binom{n}{t}$ blocks [16]. Note the relation between N, n, t and b, demonstrating that these values cannot be chosen arbitrarily. In general, a large value of b is desired (i.e., large number of blocks) to have fewer occurrences where two requested packets use the same block as their MU set. We use the notation t-(N, n) for Steiner systems (where $\lambda = 1$ is implied). Our interest lies in block designs with $t = t_{\max} + 1$ and $\lambda = 1$, such that the pairwise intersection cardinality is at most $t_{\max}$. We term a placement method where packets are constrained to such MU sets as *design* placement. In such a placement, we are guaranteed the existence of an $L^* = L$ solution if the packets are placed in L distinct MU sets.

Example 4: Consider the set $\mathcal{M} = \{0, 1, 2, 3, 4, 5, 6\}$. Its subsets (blocks) $\mathcal{M}_1 = \{0, 1, 2\}, \mathcal{M}_2 = \{0, 3, 4\}, \mathcal{M}_3 = \{0, 5, 6\}, \mathcal{M}_4 = \{1, 3, 5\}, \mathcal{M}_5 = \{1, 4, 6\}, \mathcal{M}_6 = \{2, 3, 6\}$ and $\mathcal{M}_7 = \{2, 4, 5\}$ form a 2-$(7, 3, 1)$ BIBD (which is a Steiner system). There are $\binom{7}{2}/\binom{3}{2} = 7$ blocks in this design, known as the *Fano plane* [18]. It can be seen that no two blocks intersect on more than one element (and each pair of elements is contained in exactly one block), such that this design can be used when $t_{\max} = 1$ is desired. Since $n = 3$ in this design, this value of $t_{\max}$ guarantees a solution for either $k = 2$ and $L = 3$ or $k = 1$ and $L = 5$.

An alternative for constructing MU sets with overlap at most $t_{\max}$ is the use of *constant-weight codes*. A binary (N, d, n) constant-weight code contains binary codewords of length N, each with n non-zero coordinates, such that the Hamming distance between every two vectors (i.e., the number of coordinates in which they differ) is at least d. The supports (i.e., the non-zero coordinates) of the codewords form an $(n - d/2 + 1)$-$(N, n, 1)$ *packing* [19], in which each $(n - d/2 + 1)$-subset appears *at most* once. A packing can be thought of as a relaxed version of a block design, which similarly satisfies pairwise intersection of at most $t_{\max}$ when setting $d = 2(n - t_{\max})$. As a consequence, we can use $(N, 2(n - t_{\max}), n)$ constant-weight codes to construct MU sets with the desired $t_{\max}$ intersection property. As a large number of valid MU sets is desired, we are interested in constant-weight codes with the maximum possible number of codewords for the given parameters. Constructions of constant-weight codes and lower/upper bounds on the maximum number of codewords for certain parameters N, d and n, denoted $A(N, d, n)$, are provided e.g. in [20], [21], [19]. For simplicity, we include MU sets constructed using constant-weight codes under the design placement, even though such constructions are packings rather than block designs.

Example 5: Consider the binary vectors of length $N = 5$ with exactly $n = 3$ non-zero coordinates. These vectors form an $(N, 1, n)$ constant-weight code. The corresponding MU sets are the codeword supports, which appear in Example 2.

IV. Read Algorithms

As we saw in Section II, nkMTP is intractable for general instances obtained when the unrestricted placement is used. In this section, we provide explicit and efficient optimal read algorithms for the cyclic and design policies.

A. Cyclic placement

In this subsection, we provide an efficient optimal algorithm for finding a maximum-throughput solution in the cyclic case. We start with the following important observation.

Lemma 4: Assume an nkMTP instance with cyclic placement of the packet chunks. Then there exists an optimal (attaining the maximal L^*) solution where the k MUs assigned to each read packet are cyclic consecutive. In addition, there is at least one packet in such a solution that is assigned its first k MUs.

Proof We show that any optimal solution for the cyclic placement can be transformed into an optimal solution with a cyclic consecutive assignment of MUs to each packet. Assume an optimal solution with a gap in packet j's assignment (i.e., the k assigned MUs to packet j are not cyclic consecutive). If the MUs in the gap are not assigned to any other packet, then clearly we can exchange MUs between the gap and the assigned MUs to obtain an assignment with no gap. Let us now consider a case where the MUs in the gap were assigned to other packets. Because of the cyclic placement and the fixed

n, the packets assigned the gap MUs overlap with packet j on either the MUs before the gap or the MUs after the gap. In either case we can exchange between MUs in the gap and MUs assigned to packet j to obtain a cyclic consecutive assignment, as the MUs in the overlap can serve any of the overlapping packets. Finally, there is at least one packet in a cyclic consecutive solution that is assigned its first k MUs. If there is no such packet, we can shift the solution counter-clockwise until this condition is met. ∎

Based on Lemma 4, we propose an efficient algorithm for solving a cyclic instance. For convenience, we assume a circle-arc representation (see Section III-C). Define an order of the packets with respect to packet j, such that the packets are sorted according to their arcs' starting points relatively to packet j's starting point in clockwise order. We denote such an order of the packets by $\left\{\tilde{\mathcal{S}}_i^{(j)}\right\}_{i=1}^{L}$. That is, $\tilde{\mathcal{S}}_1^{(j)}$ is packet j, $\tilde{\mathcal{S}}_2^{(j)}$ is a packet whose starting point is next after packet j's starting point in clockwise order, and so on.

Example 6: Consider the cyclic instance in Fig. 3, where the order is with respect to the topmost packet arc ($\{11,0,1,2\}$). The ordered packets are $\{11,0,1,2\}$, $\{1,2,3,4\}$, $\{3,4,5,6\}$, $\{5,6,7,8\}$, $\{7,8,9,10\}$ and $\{9,10,11,0\}$.

In the algorithm we begin with two empty sets Λ and Ω, which will eventually contain the read packets and their assigned MUs, respectively. We also initialize the sets Λ_j and Ω_j (for $j=1,2,...,L$) as empty sets. The following algorithm solves optimally a cyclic nkMTP instance.

Algorithm 1: (Cyclic placement, optimal read algorithm)
For $j=1,2,...,L$, do:

1) Set $i:=1$.
2) If $\left|\tilde{\mathcal{S}}_i^{(j)}\right| \geq k$, add i to Λ_j, and add the first k MUs in $\tilde{\mathcal{S}}_i^{(j)}$ to Ω_j. Remove the added MUs from all other packets.
3) Set $i:=i+1$. If $i \leq L$, go to Step 2. Otherwise, go to Step 4.
4) If $|\Lambda_j| > |\Lambda|$, set $\Lambda := \Lambda_j$, $\Omega := \Omega_j$.

Theorem 5: The set of packets Λ and their corresponding MUs in Ω found by Algorithm 1 are an optimal solution to a cyclic nkMTP instance.

Proof According to Lemma 4, there exists an optimal solution where the k MUs assigned to a read packet are cyclic consecutive and there is a packet j_0 in this solution that is assigned its first k MUs. We prove that Algorithm 1 finds the optimal solution in iteration $j=j_0$. Consider the order $\left\{\tilde{\mathcal{S}}_i^{(j_0)}\right\}_{i=1}^{L}$. We show that if packet i' was added to Λ_{j_0} in Step 2, then this packet appears in the optimal solution. This is proved by induction on i'. Assume all packets $1,\ldots,i'-1$ (ordered with respect to packet j_0) can be chosen as in Step 2. Then we show that the i'-th packet can be chosen in the same way. We assume by contradiction that $|\tilde{\mathcal{S}}_{i'}^{(j_0)}| \geq k$ and there is no optimal solution that contains packet i'. Then we look at the packet i'' whose starting point follows the starting point of packet i' (with respect to packet j_0), such that packet i'' appears in the optimal solution. From the fact that its k assigned MUs are cyclic consecutive it is possible to shift the assignment to the first MU index in $\tilde{\mathcal{S}}_{i'}^{(j_0)}$, and replace i'' by i' in the optimal solution without affecting the selection of packets following packet i''. This is a contradiction.

The proof is completed by observing that maximizing the size of the packet set Λ_j over all indices j is guaranteed to give the optimal solution, because at least one packet j is qualified as a j_0 that is in the optimal solution with its first k MUs. ∎

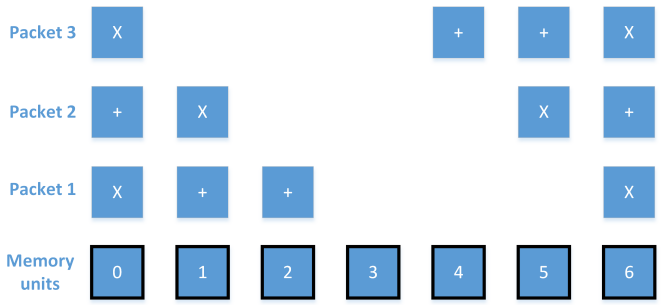


Fig. 4: A solution of a cyclic instance ($k=2, n=4$). '+' denotes an MU assigned to the packet where 'X' denotes an MU not assigned to the packet (erasure).

Algorithm 1 requires simple sorting and comparison operations, resulting in $\mathcal{O}(L^2)$ complexity. Since the solution method assures that MUs assigned to each read packet are cyclic consecutive, the non-assigned MUs can be regarded as $n-k$ erased symbols, or as a cyclic *burst* of $n-k$ erasures. An example is shown in Fig. 4. This erasure structure suggests the use of $[n,k]$ *binary cyclic* codes (not necessarily MDS), which are especially efficient at recovering burst erasures. Cyclic codes are linear codes, with the property that a cyclic shift of a codeword produces a codeword as well. These codes are capable of recovering from any cyclic burst erasure of length up to $n-k$ [22]. The use of binary cyclic codes simplifies the coding process considerably. The reason is that non-trivial MDS codes require non-binary field arithmetic and impose certain restrictions on the code parameters, which can mostly be lifted once cyclic binary codes are used.

Example 7: Consider the (systematic) cyclic code $\mathcal{C} = \{0000, 0101, 1010, 1111\}$, where $k=2$ bits are encoded to $n=4$ bits. There are n possible cyclic bursts of length $n-k=2$. Assume a burst in the first 2 codeword positions. Then the remaining bits at the last 2 positions are all distinct: $00, 01, 10$ and 11, determining uniquely the codeword. The same holds for any cyclic burst erasure of length $n-k=2$.

B. Design placement

We now turn to provide an efficient optimal read algorithm for an nkMTP instance of the design placement. The algorithm we propose owes its efficiency to the sufficient pairwise condition satisfied by design-placement instances. As we show below, if the pairwise condition is satisfied, an optimal solution does not need to assign MUs contained in sets of *more than two* packets. This fact turns out to imply an extremely simple assignment algorithm. In the typical case $k > n/2$, a certain set (design block) of n MUs can not serve more than one

packet. Thus, we consider design instances with L packets stored in L *distinct* MU sets (otherwise a subset of the packets stored in distinct blocks is considered). Denote by $\dot{\mathcal{S}}_i$ the MUs indexed in $\mathcal{S}_i$ that are not shared by any other MU set, and by $\dot{\mathcal{S}}_{ij}$ the MUs indexed in both $\mathcal{S}_i$ and $\mathcal{S}_j$ but not in any other MU set. In the rest of this sub-section, we show that for each i, at least k MUs from the sets $\dot{\mathcal{S}}_i$ and $\dot{\mathcal{S}}_{ij}$ $(j \neq i)$ can be assigned to packet i (such that these MUs do not serve any other packet). At a high level, the assignment that guarantees k MUs to packet i is all of $\dot{\mathcal{S}}_i$, and half of each $\dot{\mathcal{S}}_{ij}$. We present this more formally in the following Algorithm 2. The algorithm is initialized with empty sets $\mathcal{S}'_i$ $(i = 1, 2, ..., L)$ that will eventually contain an optimal assignment of MUs to the packets. We use the notation $\lfloor x \rfloor$ (resp. $\lceil x \rceil$) for the floor (resp. ceiling) value of x, i.e., the largest integer not greater than x (resp. the smallest integer not smaller than x).

Algorithm 2: (Design placement, optimal read algorithm)
For each packet $i = 1, ..., L$, do:

1) Add the MUs in $\dot{\mathcal{S}}_i$ to $\mathcal{S}'_i$.
2) For each j such that $\left|\dot{\mathcal{S}}_{ij}\right|$ is even, add $\left|\dot{\mathcal{S}}_{ij}\right|/2$ MUs from $\dot{\mathcal{S}}_{ij}$ to $\mathcal{S}'_i$ (disjoint from the MUs added to $\mathcal{S}'_j$ in a different iteration).
3) For each j such that $\left|\dot{\mathcal{S}}_{ij}\right|$ is odd, add either $\left\lceil \left|\dot{\mathcal{S}}_{ij}\right|/2 \right\rceil$ or $\left\lfloor \left|\dot{\mathcal{S}}_{ij}\right|/2 \right\rfloor$ MUs from $\dot{\mathcal{S}}_{ij}$ to $\mathcal{S}'_i$, according to the policy specified below under *floor/ceil balancing*.

Floor/ceil balancing. For Algorithm 2 we need to specify whether to assign $\left\lceil \left|\dot{\mathcal{S}}_{ij}\right|/2 \right\rceil$ or $\left\lfloor \left|\dot{\mathcal{S}}_{ij}\right|/2 \right\rfloor$ MUs in the odd case (step 3). We show such an assignment that for each i balances the number of floors and ceils sufficiently to guarantee at least k assigned MUs. Construct an undirected graph U whose vertices are the packet indices, and connect two vertices i and j by an edge if $\left|\dot{\mathcal{S}}_{ij}\right|$ is odd. Remove from the graph vertices not connected by an edge to any other vertex. An *orientation* of U is an assignment of a direction to each edge in U (leading to a directed graph). There always exists an orientation of an undirected graph such that the number of edges entering and exiting every vertex differ by at most one [23]. This orientation can be found in time linear in the number of edges [13]. We denote such an orientation by $\vec{U}$, and an example is shown in Fig. 5. Given $\vec{U}$, we can rewrite step 3 in Algorithm 2 in a precise way

3) For each j such that $\left|\dot{\mathcal{S}}_{ij}\right|$ is odd
   a) If the edge between i and j is oriented towards i in $\vec{U}$, add $\left\lceil \left|\dot{\mathcal{S}}_{ij}\right|/2 \right\rceil$ MUs (not added earlier) from $\dot{\mathcal{S}}_{ij}$ to packet i.
   b) Otherwise, add $\left\lfloor \left|\dot{\mathcal{S}}_{ij}\right|/2 \right\rfloor$ MUs (not added earlier) from $\dot{\mathcal{S}}_{ij}$ to packet i.

Theorem 6: The sets $\mathcal{S}'_i$ in Algorithm 2 give a full-throughput solution to any design instance.

Proof First, an MU added to $\mathcal{S}'_i$ is not added to any $\mathcal{S}'_{j\neq i}$. The reason is that MUs in $\dot{\mathcal{S}}_i$ are added to $\mathcal{S}'_i$ only, and two *disjoint* subsets of MUs (using complementary ceiling/floor operations) are taken from $\dot{\mathcal{S}}_{ij}$ to $\mathcal{S}'_i$ and $\mathcal{S}'_j$ only. Define the function $f_{ij}(x)$ as $\lceil x \rceil$ if the edge between i and j is oriented towards i in $\vec{U}$, and $\lfloor x \rfloor$ otherwise. If i and j are not connected in $\vec{U}$ (i.e., $\left|\dot{\mathcal{S}}_{ij}\right|$ is even), $f_{ij}(x)$ is simply x. The cardinality of $\mathcal{S}'_i$ is then

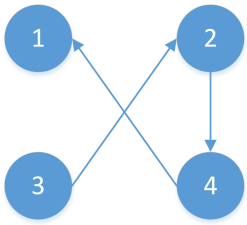


Fig. 5: An orientation where the number of edges entering and exiting a vertex differ by at most one.

$$\left|\dot{\mathcal{S}}_i\right| + \sum_{j\neq i} f_{ij}\left(\left|\dot{\mathcal{S}}_{ij}\right|/2\right). \tag{10}$$

In the rest of this proof, we show that (10) is lower-bounded by k. For odd-cardinality sets $\dot{\mathcal{S}}_{ij}$, $\left\lfloor \left|\dot{\mathcal{S}}_{ij}\right|/2 \right\rfloor$ equals $\left|\dot{\mathcal{S}}_{ij}\right|/2 - 1/2$, and $\left\lceil \left|\dot{\mathcal{S}}_{ij}\right|/2 \right\rceil$ equals $\left|\dot{\mathcal{S}}_{ij}\right|/2 + 1/2$. Since the number of edges entering and exiting a vertex differ by at most one, the number of floor operations in (10) might exceed the number of ceiling operations by at most one. Therefore, (10) is lower-bounded by

$$\left|\dot{\mathcal{S}}_i\right| + \frac{1}{2}\sum_{j\neq i}\left|\dot{\mathcal{S}}_{ij}\right| - \frac{1}{2}. \tag{11}$$

According to the inclusion-exclusion principle,

$$\left|\dot{\mathcal{S}}_i\right| = \sum_{\mathcal{J}\supseteq\{i\}} (-1)^{|\mathcal{J}|-1} \left|\mathcal{S}_{\mathcal{J}}\right|, \tag{12}$$

$$\left|\dot{\mathcal{S}}_{ij}\right| = \sum_{\mathcal{J}\supseteq\{i,j\}} (-1)^{|\mathcal{J}|} \left|\mathcal{S}_{\mathcal{J}}\right|. \tag{13}$$

Substitute $\left|\dot{\mathcal{S}}_i\right|$ and $\left|\dot{\mathcal{S}}_{ij}\right|$ in (11) by the sums expanding them in (12)-(13). Each set $\mathcal{J} \supseteq \{i\}$ appears in the combined sums once due to $\left|\dot{\mathcal{S}}_i\right|$, and additional $|\mathcal{J}| - 1$ times (weighted by $1/2$ and with an opposite sign) due to the summation of $\left|\dot{\mathcal{S}}_{ij}\right|$ over $j \neq i$. Therefore, (11) equals

$$\begin{aligned}
&\frac{1}{2}\sum_{\mathcal{J}\supseteq\{i\}} (-1)^{|\mathcal{J}|-1}\left(3 - |\mathcal{J}|\right)\left|\mathcal{S}_{\mathcal{J}}\right| - \frac{1}{2} \\
&= n - \frac{1}{2} - \frac{1}{2}\sum_{j\neq i}\left|\mathcal{S}_{ij}\right| + \frac{1}{2}\sum_{\substack{\mathcal{J}\supseteq\{i\},\\ |\mathcal{J}|\geq 4}} (-1)^{|\mathcal{J}|}\left(|\mathcal{J}| - 3\right)\left|\mathcal{S}_{\mathcal{J}}\right|.
\end{aligned} \tag{14}$$

We claim that the last sum in (14) is non-negative. This sum counts the number of occurrences of MUs in intersection sets of 4 packets or more that include packet i, multiplied by the factor $(|\mathcal{J}| - 3)/2$, and with alternating signs. Consider a certain MU shared by exactly $T \geq 4$ packets including packet i. This MU appears in $\binom{T-1}{|\mathcal{J}|-1}$ intersection sets of cardinality $4 \leq |\mathcal{J}| \leq T$ (we subtract 1 as the packet index i is always

contained in $\mathcal{J}$). Therefore, the contribution of this MU to the count is

$$\begin{aligned}&\frac{1}{2}\sum_{|\mathcal{J}|=4}^{T}(-1)^{|\mathcal{J}|}(|\mathcal{J}|-3)\binom{T-1}{|\mathcal{J}|-1} \qquad (15)\\ &=\frac{1}{2}\sum_{|\mathcal{J}|=3}^{T-1}(-1)^{|\mathcal{J}|+1}(|\mathcal{J}|-2)\binom{T-1}{|\mathcal{J}|}\\ &=\frac{1}{2}\sum_{|\mathcal{J}|=0}^{2}(-1)^{|\mathcal{J}|}(|\mathcal{J}|-2)\binom{T-1}{|\mathcal{J}|}=(T-3)/2\geq 0,\end{aligned}$$

where we used the binomial identities

$$\sum_{j=0}^{T}(-1)^j\binom{T}{j}=\sum_{j=0}^{T}j(-1)^j\binom{T}{j}=0. \qquad (16)$$

This establishes the non-negativity of the last sum in (14). $|\mathcal{S}_{ij}|\leq t_{\max}$, so we conclude that (14) and thus (10) are lower-bounded by

$$n-\frac{1}{2}-\frac{1}{2}\sum_{j\neq i}t_{\max}=n-\frac{1}{2}-\frac{1}{2}\left(L-1\right)t_{\max}=k-\frac{1}{2}. \qquad (17)$$

The number of MUs added to packet i in (10) is necessarily integer. Thus, we have the integer value (10) lower-bounded by the non-integer value (17). This means that (10) is in fact lower-bounded by the ceiling value of (17), i.e., by k. ∎

Algorithm 2 requires the construction of the sets $\dot{\mathcal{S}}_i$ and $\dot{\mathcal{S}}_{ij}$, which can be performed in $\mathcal{O}(Ln)$ operations by running over the elements in the sets $\mathcal{S}_i$. We then have to find a balanced path in a graph that is complete in the worst case (i.e., when $|\dot{\mathcal{S}}_{ij}|$ are all odd), with $\mathcal{O}(L^2)$ edges. The path is found in linear-time in the number of edges, such that the total complexity of Algorithm 2 is $\mathcal{O}(L(n+L))$.

Example 8: Consider the block design of Example 4 (where $n=3$). This design can be used to read $L=3$ packets if $t_{\max}=n-k=1$, i.e., when $k=2$. Assume that the three packets are stored in the MU sets $\mathcal{S}_1=\{1,2,3\}$, $\mathcal{S}_2=\{1,4,5\}$ and $\mathcal{S}_3=\{3,5,6\}$. Then $\dot{\mathcal{S}}_1=\{2\},\dot{\mathcal{S}}_2=\{4\},\dot{\mathcal{S}}_3=\{6\},\dot{\mathcal{S}}_{12}=\{1\},\dot{\mathcal{S}}_{13}=\{3\}$ and $\dot{\mathcal{S}}_{23}=\{5\}$. Since all the pairwise sets are of odd cardinality, U in this case is a complete graph with three vertices. Labeling these vertices $1,2$ and 3, a valid orientation $\vec{U}$ is $1\to 2\to 3\to 1$. Using Algorithm 2, we obtain $\mathcal{S}'_1=\{2,3\},\mathcal{S}'_2=\{1,4\}$ and $\mathcal{S}'_3=\{5,6\}$.

V. Probabilistic analysis

In this section, we consider *ensembles* of random instances characterized by k,n,N,L and the placement policy in use, where an instance is obtained by sampling the L packet sets uniformly and independently from the universe of sets allowed in the corresponding placement policy. As an example, under the cyclic placement, the L packets are assigned L sets independently and uniformly from the universe of cyclic-consecutive MU n-sets. Our primary objective is to calculate or bound the full-throughput probability $\Pr\left(L^*=L\right)$ for the three placement policies discussed above. For the unrestricted and cyclic placements we use the coverage and pairwise conditions (see Section III-A) to obtain upper and lower bounds, respectively, on the full-throughput probability. We later present a convenient tighter probabilistic framework for analyzing the throughput performance for the design placement.

A. Unrestricted placement

Denote the probability of the coverage condition (2) in the unrestricted placement by $p^{\mathrm{un}}_{\mathrm{cover}}$. The full-throughput probability $\Pr\left(L^*=L\right)$ is clearly upper bounded by $p^{\mathrm{un}}_{\mathrm{cover}}$. The coverage condition in the unrestricted case is equivalent to the requirement that the union of L random n-subsets of an N-element set results in a set of cardinality at least kL. A closed-form expression for this probability is provided by the union model [24], which is an extension of the balls-and-bins model [25]. The details are provided in Appendix B. Through this calculation we obtain an upper bound on the full-throughput probability for any combination of k,n,N and L. Exact calculation of the full-throughput probability for the unrestricted placement seems hard, and even a lower bound through the pairwise condition (3) is not available. This lack of positive results for the unrestricted placement is not very surprising given the computational hardness of solving it optimally.

B. Cyclic placement

Considering the circle-arc representation of cyclic instances (see Section III-C), the probability of the coverage condition is the probability that at least kL points of the circle are covered by L random arcs of n cyclic consecutive points each. In [26], [27], the probability distribution of the number of vacant points on a circle once L random arcs are placed without replacement was derived. In our case, replacement is allowed (i.e., the same MU set may serve two packets or more), and for the upper bound on full-throughput probability we are actually interested in the complement distribution of the occupied points. The details are provided in Appendix B. We denote the probability of the coverage condition in the cyclic case by $p^{\mathrm{cyc}}_{\mathrm{cover}}$. For the lower bound, unlike the unrestricted policy, the structure in the cyclic case allows to find the probability of the pairwise condition, which we denote $p^{\mathrm{cyc}}_{\mathrm{pair}}$.

Theorem 7: Consider an instance drawn at random from a cyclic ensemble with parameters N,n,L. The probability that the maximum pairwise intersection cardinality is at most $t_{\max}$ is

$$p^{\mathrm{cyc}}_{\mathrm{pair}}=N^{1-L}\prod_{i=1}^{L-1}\left(N-L\left(n-t_{\max}\right)+i\right). \qquad (18)$$

Proof Consider a circle-arc representation of the cyclic nkMTP instances. Assume clockwise order, and that each packet arc does not precede the first packet arc. Each placed packet prevents the placement of the start of any other packet in its first $n-t_{\max}$ MUs. In a legal placement (i.e., when the pairwise intersection cardinality is at most $t_{\max}$), there are $N-L(n-t_{\max})$ MUs that do not belong to the first $n-t_{\max}$ MUs of any packet. Thus, the number of legal placements

(given the order constraint above) is equivalently the number of ways to partition $N-L(n-t_{\max})$ MUs to L sets of cyclic consecutive MUs. Thinking of the latter MU sets as *gaps*, they can be distributed in $\binom{N-L(n-t_{\max})+L-1}{L-1}$ ways, which is the number of L non-negative integers (gap lengths) whose sum is $N-L(n-t_{\max})$ [16]. Each legal placement is obtained (uniquely) as a combination of 1) the starting MU for the first drawn packet, 2) a gap configuration, and 3) a permutation of the other $L-1$ packets. Hence to get the total number of legal placements we multiply the number of gap configurations by N (the number of possible starting points for the first packet) and by $(L-1)!$ (the number of permutations of $L-1$ packets). After normalizing by the total number of (legal and illegal) placements N^L, we obtain (18). ∎

C. Design placement

The design placement enjoys a sharper characterization of full-throughput instances, which simplifies the probabilistic analysis. It is sufficient that the L MU sets are different design blocks, and the request is full-throughput by the design properties and the sufficient pairwise condition (3). Thus, the probability that a random design instance contains a full-throughput solution is lower-bounded by the probability that the L MU sets are distinct. We denote this probability by $p_{\text{pair}}^{\text{des}}$. To find this probability, we use the balls-and-bins model [25]. In this model, there are L balls and b bins (recall that b is the number of blocks in the design), where the balls are placed independently and uniformly at random in the bins. The probability of L distinct blocks is the probability of L non-empty bins [25], which equals

$$p_{\text{pair}}^{\text{des}} = \binom{b}{L}\frac{1}{b^L}\sum_{j=0}^{L}(-1)^j\binom{L}{j}(L-j)^L. \qquad (19)$$

In the typical case $k > n/2$, each block can serve only one packet, and thus $p_{\text{pair}}^{\text{des}}$ is the *exact* probability of a full-throughput solution in the design case.

To demonstrate the possible improved performance when the design placement is used, assume that $n = k+1$ for a fixed k value. If the desired number of read packets is $L^* = L = 3$, we can take the 2-$(k^2+k+1, k+1)$ Steiner system [16], where the sufficient pairwise condition $t_{\max} = n-k = 1$ is guaranteed by the $t = 2$ parameter of the design. To have a full-throughput solution we need that the $L = 3$ blocks drawn from the $b = k^2+k+1$ blocks of the design will be all distinct. In Fig. 6, we plot $p_{\text{pair}}^{\text{des}}$, which is $\Pr(L^* = L)$ in the design case, in comparison to the unrestricted upper bound and the cyclic lower and upper bounds on $\Pr(L^* = L)$ ($p_{\text{cover}}^{\text{un}}$, $p_{\text{pair}}^{\text{cyc}}$ and $p_{\text{cover}}^{\text{cyc}}$, respectively). We also plot $p_{\text{sim}}^{\text{cyc}}$, the only graph in Fig. 6 obtained using simulations, which is the empirical $\Pr(L^* = L)$ in the cyclic case. The results clearly demonstrate that the design policy exhibits significantly superior performance. It is shown that with a fixed redundancy of 1 chunk per packet, the full-throughput probability of the design placement grows monotonically when k grows and $N = k^2+k+1$ MUs are deployed in the switch.

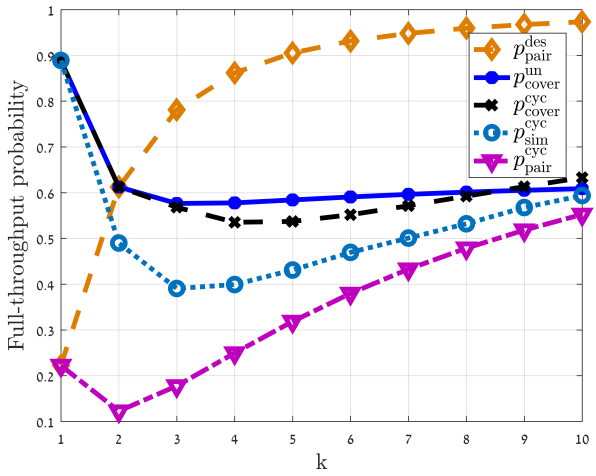
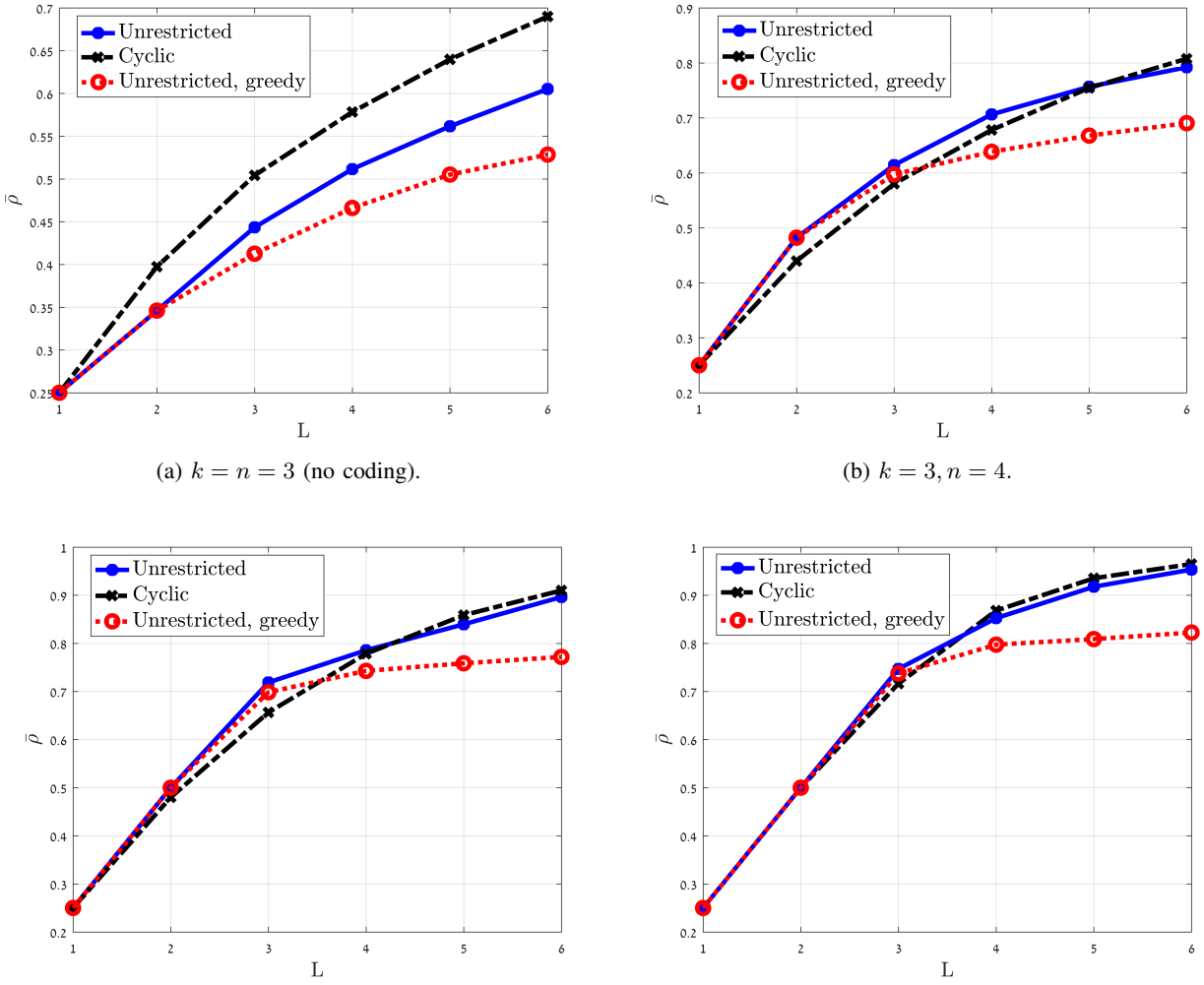


Fig. 6: A comparison of full-throughput performance bounds ($n = k+1, N = k^2+k+1$).

VI. Simulation Results

In this section, we provide simulation results of the average throughput performance of the placement policies proposed in Section III. Recall from Section II-B that the average throughput equals a constant times the average L^* of ensemble instances. Hence evaluating the average throughput can be done by solving random instances optimally, and averaging the resulting L^* values empirically. For the unrestricted placement we solved unrestricted nkMTP instances by an exhaustive-search algorithm (recall that no efficient algorithm is likely to exist in this case, see Section II). We compare it to a greedy (suboptimal) solution, where random packets are assigned k MUs, until no k MUs that can serve a packet remain. To solve cyclic instances optimally, we used Algorithm 1. A comparison of the average throughput $\bar{\rho}$ performance (see Section II-B) of the unrestricted and cyclic placement policies is provided in Fig. 7 for $k = 3$ and $n = 3$ (uncoded case) up to $n = 6$. Several observations follow from these results. First, coding improves throughput performance considerably. Taking the cyclic case as an example when $L = 4$, the throughput performance is improved by 18% ($n = 4$) to 52% ($n = 6$) compared to the uncoded case ($n = 3$). Another important observation is that the unrestricted policy does *not* necessarily lead to better performance compared to the cyclic policy. Actually, the relation between the performance of these schemes depends on the system parameters. We can see that the cyclic case provides higher throughput performance when k is close to n, showing that the structure becomes helpful when the read-flexibility in choosing k MUs decreases. On the other hand, when the redundancy becomes larger (i.e., when n becomes large compared to k), the unrestricted and the cyclic placement policies exhibit similar performance, each with a slight advantage at different L values. When computational complexity is taken into account, the cyclic placement becomes superior over unrestricted, because it outperforms the low-complexity greedy read algorithm.

Another performance measure we investigated is the number of packets that are read (i.e., the value of L^*) with high



(a) $k = n = 3$ (no coding).

(b) $k = 3, n = 4$.

(c) $k = 3, n = 5$.

(d) $k = 3, n = 6$.

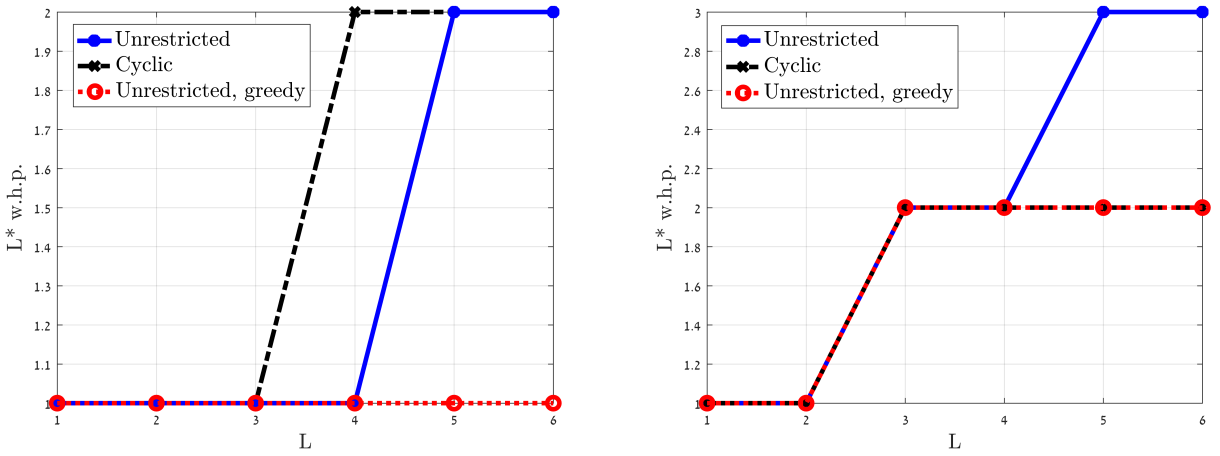
Fig. 7: Average throughput $\bar{\rho}$ performance comparison ($N = 12$).

probability (w.h.p.) in a random instance. In Fig. 8, we show L^* values that were observed with probability at least 0.95. Similarly to the results in Fig. 7a, the cyclic scheme provides better L^* performance in the uncoded case. For moderate n values, the unrestricted policy is better than cyclic, where for larger n values the performance becomes close. Regardless of the placement policy in use, coding improves the throughput performance. For example, when $k = n = 3$ (Fig. 8a) and the load is $L = 3$, we expect to get w.h.p. only one packet at the output. On the other hand, when coding is introduced such that $k = 3$ and $n = 4$, this increases to $L^* = 2$ packets and keeps improving up to $L = L^* = 3$ for $k = 3$ and $n = 6$. That is, when the switch is required to fulfill all L requests w.h.p., coding is an important tool.

In Fig. 9, we compare the probability of a full-throughput solution (i.e., $L^* = L$ solution) for the unrestricted, cyclic and design placement policies for $k = 3, n = 5$ and $L = 3$. We note that in the design case, the probability is obtained analytically using the balls and bins model (See Section V-C), given the number of valid MU sets (blocks). To find the number of blocks, we used constant-weight codes with $n = 5$ and $d = 2(n - t_{\max}) = 6$ (see Section III-D). The (exact) number of blocks (i.e., the maximum number of codewords) in this case is known for N values up to 17 [21]. The graphs in Fig. 9 show that the unrestricted placement is somewhat better than the design placement in terms of full-throughput probability. The reason is the large number of valid MU sets in the unrestricted case, which is $\binom{N}{n}$, compared to the number of blocks in the design case, which is typically much smaller (e.g., 68 blocks when $N = 17$ compared to 6188 subsets). However, as no efficient read algorithm is known for the unrestricted placement, an exhaustive-search algorithm requires $2^{Ln} = 2^{15}$ operations to find an optimal solution. On the other hand, the efficient optimal read algorithm in the design case requires only $L(n + L) = 24$ operations, i.e., a number smaller by four orders of magnitude. This makes the design placement policy appealing in practice due to complexity considerations.

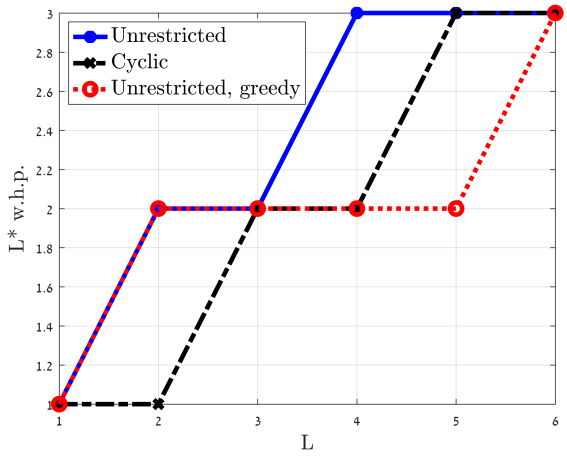
VII. Conclusion

In this paper, we studied placement policies and read algorithms of coded packets in a switch memory. The study

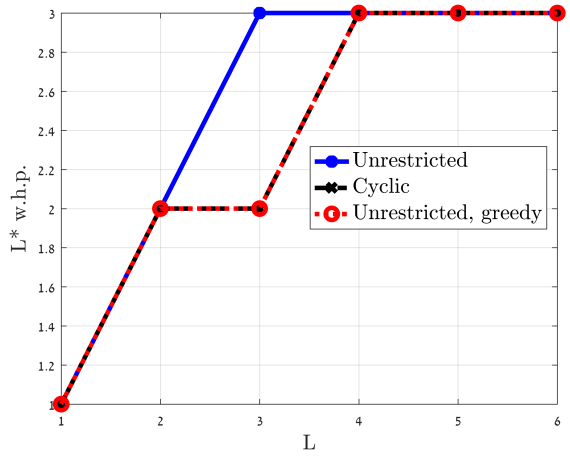


(a) $k = n = 3$ (no coding).

(b) $k = 3, n = 4$.



(c) $k = 3, n = 5$.



(d) $k = 3, n = 6$.

Fig. 8: The expected L^* with probability at least 0.95 as a function of the load L ($N = 12$).

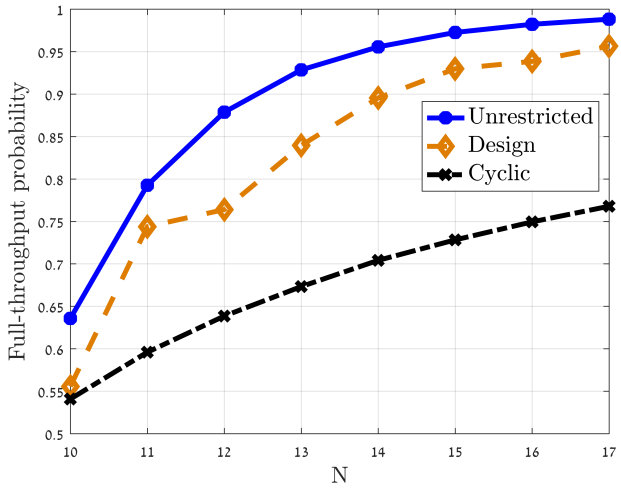


Fig. 9: Full-throughput probability for $k = 3, n = 5, L = 3$ as a function of N.

revealed that coding can significantly improve switching throughput, and that the choice of placement has significant effect on performance and complexity. We proved that in its most general form, the problem of obtaining maximum throughput for a set of requested packets is a hard problem. Therefore, we moved to propose two practical placement policies and efficient optimal read algorithms, with better throughput performance in certain cases compared to the general non-structured placement policy.

We demonstrated tradeoffs between write flexibility, read-algorithm complexity and performance. In particular, we saw that no choice of placement policy is universally optimal, and provided analytic tools for choosing a policy wisely. Our work leaves many interesting problems for future research. For example, one may consider other structured write policies by imposing different constraints rather than restricting pairwise intersections. It is also interesting to consider variable-length codes (i.e., varying values of k and n values for each packet), to match the expected switch load.

APPENDIX A
DETAILED PROOF OF THEOREM 2

To show the hardness of nkMTP when $3 \leq k \leq n$, we define its decision-problem version, which we name M-nkMTP. In the rest of this appendix, we assume that $3 \leq k \leq n$.

Problem 2: (M-nkMTP)

Input: An nkMTP instance and a positive integer M.

Output: "Yes" if there are M subsets $\mathcal{S}'_i \subseteq \mathcal{S}_i$ with the properties $|\mathcal{S}'_i| = k$, $\mathcal{S}'_i \cap \mathcal{S}'_j = \emptyset$ $(i \neq j)$.

For showing that nkMTP is NP-hard we can show that M-nkMTP is NP-complete. Note that M-nkMTP is in NP, since once we are given a collection of M subsets $S'_i \subseteq \mathcal{S}_i$ claimed to be pairwise disjoint, this can be validated in polynomial time. It remains to reduce a known NP-complete problem to M-nkMTP, meaning that we have to show that an efficient solution to M-nkMTP implies an efficient solution to this NP-complete problem. We will reduce the *l-set packing* problem (l-SP), known to be NP-complete for $l \geq 3$ [14], to our problem. l-SP is defined as follows.

Problem 3: (l-SP)

Input: A collection of sets over a certain domain, each of them of size l, and a positive integer M.

Output: "Yes" if there are M pairwise disjoint sets.

M-nkMTP is NP-complete for $3 \leq k = n$, since in this case M-nkMTP and l-SP, for $l = k = n$, are essentially the same. Therefore, it remains to reduce l-SP ($l \geq 3$) to M-nkMTP for $3 \leq k < n$. Let us begin with reducing l-SP to M-nkMTP with $k = l, n = k + 1$.

Consider an instance of l-SP with $l = k$, with M denoting the number of pairwise disjoint subsets required in the solution. Assume that the input to l-SP are L sets $\mathcal{A}_i$ $(i = 1, 2, ..., L)$, where the elements contained in $\mathcal{A}_i$ are $\bigcup_i \mathcal{A}_i = \{a_1, a_2, ..., a_s\}$. For building an instance of M-nkMTP with $k = l, n = k + 1$, do the following:

- Build sets $\mathcal{B}_i$, each of size k, from s new elements $\{b_1, b_2, ..., b_s\}$, such that a one-to-one correspondence between the elements in $\mathcal{A}_i$ and the elements in $\mathcal{B}_i$ exists: $a_j \in \mathcal{A}_i \Leftrightarrow b_j \in \mathcal{B}_i$.
- Add a new element, say θ, which does not belong to either $\mathcal{A}_i$ or $\mathcal{B}_i$, to both sets to obtain the new sets denoted by $\tilde{\mathcal{A}}_i$ and $\tilde{\mathcal{B}}_i$.

The input to M-nkMTP with $n = k + 1$ will be the sets $\tilde{\mathcal{A}}_i$ and $\tilde{\mathcal{B}}_i$, where we ask whether there exist $2M$ subsets of size k each that are pairwise disjoint. If the l-SP instance has a solution of size M for the sets $\mathcal{A}_i$, then clearly the sets $\mathcal{A}_i \subseteq \tilde{\mathcal{A}}_i, \mathcal{B}_i \subseteq \tilde{\mathcal{B}}_i$ serve as solution of size $2M$ to M-nkMTP with $n = k + 1$. On the other hand, if there exists a solution of size $2M$ in the M-nkMTP problem, we have three cases:

1) M subsets $\mathcal{A}'_i \subseteq \tilde{\mathcal{A}}_i$ and M subsets $\mathcal{B}'_i \subseteq \tilde{\mathcal{B}}_i$ appear in the solution. The element θ can appear in only one of the subsets, since they must be pairwise disjoint. If θ belongs to some $\mathcal{A}'_i$, then we have M subsets $\mathcal{B}'_i$ that provide a solution to l-SP (after transforming the elements in $\mathcal{B}'_i$ to the their corresponding elements in $\mathcal{A}'_i$). On the other hand, if θ belongs to some $\mathcal{B}'_i$, then the solution is the sets $\mathcal{A}'_i$.
2) M_1 subsets $\mathcal{A}'_i \subseteq \tilde{\mathcal{A}}_i$ and M_2 subsets $\mathcal{B}'_i \subseteq \tilde{\mathcal{B}}_i$ appear in the solution, where $M_1 < M_2$ and $M_1 + M_2 = 2M$. θ can appear in at most one of the subsets $\mathcal{B}'_i$. In addition, $M < M_2$, and therefore choosing the subsets $\mathcal{B}'_i$ that do not contain θ leads to a solution of l-SP with at least M subsets (again, transformation to the elements of $\mathcal{A}_i$ is required).
3) M_1 subsets $\mathcal{B}'_i \subseteq \tilde{\mathcal{B}}_i$ and M_2 subsets $\mathcal{A}'_i \subseteq \tilde{\mathcal{A}}_i$ appear in the solution, where $M_1 < M_2$ and $M_1 + M_2 = 2M$. A solution of size at least M to l-SP is obtained in a similar way to the previous case.

The transformation $\mathcal{A}_i, \mathcal{B}_i \rightarrow \tilde{\mathcal{A}}_i, \tilde{\mathcal{B}}_i$ is polynomial in L, since it merely requires to build L sets of size k and to add one element to each of the resulting $2L$ sets. Thus, the reduction described above is a polynomial time reduction. Therefore, M-nkMTP is NP-complete for $k \geq 3, n = k + 1$, and it remains to show that M-nkMTP is NP-complete for $k \geq 3, n > k+1$. Consider M-nkMTP with $k \geq 3, n = k + 2$. We can reduce M-nkMTP with $k \geq 3, n = k + 1$ (which we proved to be NP-complete) to M-nkMTP with $k \geq 3, n = k + 2$, similarly to the reduction of l-SP to M-nkMTP with $k = l, n = k + 1$ that was described earlier. Continuing in the same fashion, we are able to reduce M-nkMTP with $n = k + j$ $(k \geq 3, j \geq 1)$ to M-nkMTP with $n = k+j+1$. Finally, we deduce that M-nkMTP is NP-complete for $3 \leq k \leq n$, meaning that nkMTP (the optimization version of M-nkMTP) is NP-hard. ∎

APPENDIX B
$p_{\text{cover}}^{\text{un}}$ AND $p_{\text{cover}}^{\text{cyc}}$

The following derivation of $p_{\text{cover}}^{\text{un}}$ (see Section V-A) is based on the union model [24]. Define the function:

$$I_m(i,n) = \sum_{j=0}^{\min(i,n)-m} (-1)^j \cdot \nu_{m+j}(i,n) \cdot \binom{m+j}{m},$$

where

$$\nu_{m+j}(i,n) = \binom{N}{m+j} \cdot \binom{N-(m+j)}{i-(m+j)} \cdot \binom{N-(m+j)}{n-(m+j)}, \tag{20}$$

for $i = 0, 1, ..., N$. I_m is the number of ways to realize two sets of cardinalities i and n, taken from a set of N elements, such that their intersection is of cardinality m. Define the probability distribution R_m:

$$R_m(i,n) = \frac{I_m(i,n)}{\binom{N}{i} \cdot \binom{N}{n}}, \tag{21}$$

which is the probability that two sets of cardinalities i and n, taken uniformly at random from a set of N elements, have an intersection of cardinality m. Define the following $(N+1) \times (N+1)$ Markov matrix, with indices i, j ranging from 0 to N:

$$(\mathbf{\Gamma})_{i,j} = R_{i+n-j}(i,n). \tag{22}$$

The (i,j) entry of $\mathbf{\Gamma}$ is the probability that the union of a set with i elements and a set with n elements is of cardinality j. Finally, $p_{\text{cover}}^{\text{un}}$ is the sum of the first kL entries in the first row of $\mathbf{\Gamma}^L$ (we assume that $kL \leq N$).

To obtain $p_{\text{cover}}^{\text{cyc}}$, we use the probability distribution on V, the number of vacant points on the circle when L random arcs are placed without replacement. A closed-form expression for this distribution is given in Theorem 1 in [27]. This expression is rather long and depends on the parameter range so we do not provide this here. We are actually interested in the number of *non*-vacant points (i.e., how many MUs are covered by the packets) which is the probability distribution of $N-V$. As in our case the arcs are taken *with* replacement, we condition the probability distribution of $N-V$ by the probability distribution on the number of distinct arcs among L random arcs using the balls-and-bins model (i.e., (19) with $b=L$). We note here that sampling with replacement is discussed as well in Chapter 4.1 of [27].

REFERENCES

[1] A. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 476–489, March 2011.

[2] G. Joshi, Y. Liu, and E. Soljanin, "Coding for fast content download," in *2012 50th Annual Allerton Conference on Communication, Control, and Computing*, Oct 2012, pp. 326–333.

[3] ——, "On the delay-storage trade-off in content download from coded distributed storage systems," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 989–997, May 2014.

[4] G. Liang and U. Kozat, "Fast cloud: Pushing the envelope on delay performance of cloud storage with coding," *IEEE/ACM Transactions on Networking*, vol. 22, no. 6, pp. 2012–2025, Dec 2014.

[5] L. Huang, S. Pawar, H. Zhang, and K. Ramchandran, "Codes can reduce queueing delay in data centers," *2012 IEEE International Symposium on Information Theory Proceedings (ISIT)*, pp. 2766–2770, July 2012.

[6] N. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?" in *2013 51st Annual Allerton Conference on Communication, Control, and Computing*, Oct 2013, pp. 731–738.

[7] ——, "The MDS queue: Analysing the latency performance of erasure codes," in *2014 IEEE International Symposium on Information Theory (ISIT)*, June 2014, pp. 861–865.

[8] Z. Wang, O. Shaked, Y. Cassuto, and J. Bruck, "Codes for network switches," *2013 IEEE International Symposium on Information Theory Proceedings (ISIT)*, pp. 1057–1061, July 2013.

[9] Z. Wang, H. M. Kiah, and Y. Cassuto, "Optimal binary switch codes with small query size," *2015 IEEE International Symposium on Information Theory (ISIT)*, pp. 636–640, June 2015.

[10] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley-Interscience, 2005.

[11] S. Lin and D. J. Costello, *Error Control Coding, Second edition*. Prentice Hall, 2004.

[12] S.-J. Lin, W.-H. Chung, and Y. S. Han, "Novel polynomial basis and its application to reed-solomon erasure codes," in *Proceedings of the 55th Annual Symposium on Foundations of Computer Science (FOCS 14)*, 2014, pp. 316–325.

[13] A. Bondy and U. Murty, *Graph theory*. Springer, 2008.

[14] E. Hazan, S. Safra, and O. Schwartz, "On the complexity of approximating k-set packing," *Comput. Complex.*, vol. 15, no. 1, pp. 20–39, May 2006.

[15] M. Viderman, "LP decoding of codes with expansion parameter above 2/3," *Information Processing Letters*, vol. 113, no. 7, pp. 225 – 228, 2013.

[16] J. H. van Lint and R. M. Wilson, *A course in combinatorics*. Cambridge University Press, 2001.

[17] C. J. Colbourn and J. H. Dinitz, *Handbook of Combinatorial Designs, Second Edition (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC, 2006.

[18] D. R. Stinson, *Combinatorial Designs: Constructions and Analysis*. Springer, 2003.

[19] P. R. J. Ostergard, "Classification of binary constant weight codes," *IEEE Transactions on Information Theory*, vol. 56, no. 8, pp. 3779–3785, Aug 2010.

[20] S. Verdu and V. K. Wei, "Explicit construction of optimal constant-weight codes for identification via channels," *IEEE Transactions on Information Theory*, vol. 39, no. 1, pp. 30–36, Jan 1993.

[21] E. Agrell, A. Vardy, and K. Zeger, "Upper bounds for constant-weight codes," *IEEE Transactions on Information Theory*, vol. 46, no. 7, pp. 2373–2395, Nov 2000.

[22] W. Ryan and S. Lin, *Channel codes: Classical and modern*. Cambridge University Press, 2009.

[23] C. S. J. Nash-Williams, "On orientations, connectivity and odd vertex pairings in finite graphs," *Canad. J. Math*, vol. 12, no. 555-567, p. 8, 1960.

[24] R. Cohen and Y. Cassuto, "Iterative decoding of LDPC codes over the q-ary partial erasure channel," *IEEE Transactions on Information Theory*, vol. 62, no. 5, May 2016.

[25] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[26] L. Holst, "On discrete spacings and the Bose-Einstein distribution," in *Contributions to Probability and Statistics in honour of Gunnar Blom (Ed. by J. Lanke and G. Lindgren)*, 1985, pp. 169–177.

[27] G. Barlevy and H. N. Nagaraja, "Properties of the vacancy statistic in the discrete circle covering problem," FRB of Chicago, Tech. Rep., 2015.

Rami Cohen (S'12) received the B.Sc. in Electrical Engineering and Physics (*cum laude*) and the M.Sc. in Electrical Engineering from the Technion - Israel Institute of Technology in 2010 and 2012, respectively. He is currently pursuing the Ph.D. degree at the Andrew and Erna Viterbi Faculty of Electrical Engineering, Technion – Israel Institute of Technology. His current research interests lie in coding theory and information theory, in particular the analysis and design of codes for practical high-speed memory devices and systems.

Yuval Cassuto (S'02-M'08-SM'14) is a faculty member at the Andrew and Erna Viterbi Faculty 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.