The Fundamental Theorem of Distributed Storage Systems Revisited

Ping Hu
Department of Electronic Engineering
City University of Hong Kong
Email: ping.hu@my.cityu.edu.hk

Kenneth W. Shum
Institute of Network Coding
the Chinese University of Hong Kong,
Shenzhen Key Lab. of Network Coding
Key Tech. and Application, China.
Email: wkshum@inc.cuhk.edu.hk

Chi Wan Sung
Department of Electronic Engineering
City University of Hong Kong
Email: albert.sung@cityu.edu.hk

***Abstract*—The fundamental theorem of distributed storage systems characterizes the maximum file size that can be stored with certain assumptions on file retrieval and node repair. The result is composed of two parts, namely, the min-cut bound and that the bound can be achieved by linear network code with bounded field size. The derivation of the min-cut bound is re-examined and illuminated by making an implicit step explicit. Furthermore, a simple alternative proof for the achievability of the min-cut bound is presented, which is based on the construction of the generic storage code, a restricted form of generic network code. The proof techniques in this paper are expected to be extensible to other more complex models of distributed storage systems.**

I. Introduction

Distributed storage systems (DSS) have attracted a lot of research attention in recent years, partly because of the successful application of network coding theory to the design of DSS with repair considerations. In the seminal work of [1], the dynamics of node failures and repairs in DSS is modeled as a single-source multicast acyclic network, and the fundamental tradeoff between storage efficiency and repair bandwidth in DSS is revealed by characterizing the minimum-cut of the corresponding information flow graph.

Like the study of any communication channel or network, the fundamental question to ask in relation to DSS is this: What is its capacity in the information-theoretic sense? The answer to this type of questions naturally consists of two parts, an upper bound of the capacity and the achievability of the bound. The work of [1] gives the first part of the answer by deriving the min-cut bound for the information flow graph of DSS and provide a partial answer to the second by using results from network coding theory, which can be applied to DSS with a *bounded* number of node failures and repairs. A more complete answer to the second part is given in [2], which proves that the min-cut bound can be achieved by a linear network code with a field size which depends only on the number of storage nodes, but *not* on the number of failures and repairs that occur in the system.

The contribution of this paper is two-fold. First, we prove a combinatorial result (in Lemma 1) about the size of the boundary of a cut in the information flow graph. This result is crucial in computing the min-cut bound, but is assumed implicitly in the proof of the min-cut bound in [1]. We re-derive the min-cut bound on the capacity of a DSS in the first part of this paper. Second, we provide an alternative proof for the achievability of the fundamental tradeoff between storage and repair bandwidth by linear network codes in [2]. The proof of [2] relies on a path-weaving procedure, which is highly dependent on the structure of the information flow graph. In this paper, we give a deterministic algorithm for constructing a linear regenerating code, based on a restricted form of generic network code [3]. We call it a *generic storage code*. The procedure of constructing a generic storage code is conceptually simpler than the one in [2], but requires a larger field size. Our construction is easily transplantable to other models in distributed storage systems with different assumptions.

We remark that the results in this paper cannot be directly obtained from the original linear network coding theory [3], [4], which considers multicast networks that are finite and fixed in advance. For DSS, the flow graph is unbounded because there is no limit on the number of node failures, but the results in [3], [4] assume finite networks and finitely many receiving nodes. Our result, as well as that in [2], shows the existence of a capacity-achieving linear code, whose field size grows exponentially on the number of storage nodes. It is an open question whether the field size can be reduced to polynomial growth.

II. Distributed Storage Systems

In a distributed storage system (DSS), there are n storage nodes. A file of B symbols from an alphabet set Σ has to be stored in the DSS. This file can later be retrieved by a data collector, which connects to any k storage nodes. Each storage node is able to store α symbols. These storage nodes are not reliable and may fail after prolonged use. If a node fails, a replacement node, called *newcomer*, is created by

This work was partially supported by a grant from the University Grants Committee of the Hong Kong Special Administrative Region, China (Project No. AoE/E-02/08), and Shenzhen Key Laboratory of Network Coding Key Technology and Application (ZSDY20120619151314964), and Peacock Scheme (KQCX20130628164008004), Shenzhen, China.

The authors are listed alphabetically.

978-1-4799-5999-0/14/$31.00 ©2014 IEEE

downloading data from any d of the surviving nodes. These d nodes are called *helper nodes*, and the number of symbols a newcomer downloads from each of them is denoted by β. The total number of symbols downloaded, $d\beta$, is called the *repair bandwidth*. The newcomer, just like the original failed node, is allowed to store at most α symbols. The newcomer is not required to store exactly the same symbols as the original failed node. This repair mechanism is commonly called *functional repair*.

A DSS that satisfies the above requirement is denoted by DSS(n, k, d, α, β). As time evolves, nodes fail and are replaced by newcomers. The particular failure order of the storage nodes and the choice of helper nodes for each newcomer defines an *instance* of a DSS. Each DSS instance can be represented by a directed acyclic graph (DAG), called *information flow graph*, which consists of three types of vertices: a source node S, storage nodes, In_i's and Out_i's, and data collectors DC_i's. For $i = 1, 2, \ldots, n$, there is a directed edge from S to In_i with infinite capacity, and a directed edge from In_i to Out_i with capacity α. When the k-th node failure in the DSS occurs, where $k = 1, 2, \ldots,$ a corresponding newcomer joins the system. We denote it by node i, where $i = n + k$. As the original storage nodes, it is represented by In_i and Out_i, which are connected by a directed edge from the former to the latter with a capacity of α. Furthermore, In_i is connected to d helper nodes by adding a directed edge from each of $\mathsf{Out}_{i_1}, \mathsf{Out}_{i_2}, \ldots, \mathsf{Out}_{i_d}$ to In_i, where $i_j < i$ for $j = 1, 2, \ldots, d$. For all possible choices of k surviving storage nodes, a corresponding DC_i is added. Let the k nodes to which DC_i connects be nodes $i_1, i_2, \ldots, i_k$. Then there is an edge of infinite capacity from each of $\mathsf{Out}_{i_1}, \mathsf{Out}_{i_2}, \ldots, \mathsf{Out}_{i,k}$ to DC_i. For each DSS instance, there is a corresponding information flow graph. The collection of graphs for all possible instances of DSS(n, k, d, α, β) is denoted by $\mathcal{G}(n, k, d, \alpha, \beta)$, or simply $\mathcal{G}$ if no ambiguity could occur.

Let $\mathrm{I}(v)$ and $\mathrm{O}(v)$ denote the set of incoming edges and the set of outgoing edges of $v \in V$. We define the important concepts of *cut* and *flow* below:

Definition 1. *An s-t* cut *$K = (S, T)$ of a DAG, $G = (V, E)$, is a partition of V into two disjoint subsets, S and T, such that $s \in S$, $t \in T$ and there is at least one edge joining S and T. The* cut-set *of K is $\{(u, v) \in \mathcal{E} : u \in S, v \in T\}$. The* cut-value *of K is the sum of the capacity of all edges in the cut-set of K.*

Definition 2. *A* flow *$\boldsymbol{f} = \{f_e : e \in E\}$ in a DAG, $G = (V, E)$, from S to DC is a valid assignment of a nonnegative integer f_e to every edge $e \in E$ such that f_e is less than or equal to the capacity of e and $\sum_{e \in I(v)} f_e = \sum_{e \in O(v)} f_e$ for any $e \in V \setminus \{\mathsf{S}, \mathsf{DC}\}$. The* flow-value *of $\boldsymbol{f}$ is $\sum_{e \in O(\mathsf{S})} f_e$. The maximum flow-value from S to DC is denoted by maxflow(DC).*

Figure 1 shows an example of a graph in $\mathcal{G}(4, 2, d, \alpha, \beta)$ with only one DC shown. (The other DC's are omitted for clarity.) We are interested to find the minimum value of all the cuts, with respect to all DC's, in any $G \in \mathcal{G}$. A possible cut with value $\alpha + 2\beta$ is also shown in the figure. A rigorous min-cut analysis will be done in the next section.

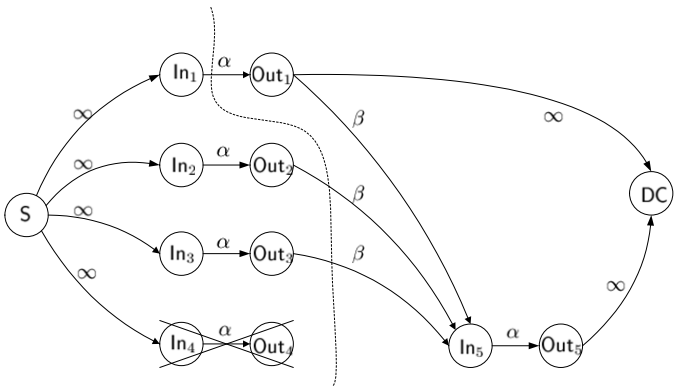

Fig. 1. An example of a graph in $\mathcal{G}(4, 2, d, \alpha, \beta)$.

III. Min-Cut Bound

Given an information flow graph $G_0 \in \mathcal{G}$, we construct an auxiliary graph $G = (V, E)$ as follows: First, we combine In_i and Out_i into a single vertex v_i, which represents storage node i, for all i. Denote the source node and an arbitrary data collector by s and t, respectively, where $s, t \in V$. Furthermore, denote the set of all child vertices of s by V_s and the set of all parent vertices of t by V_t. Since the data collector connects to k storage nodes for file retrieval, we have $|V_t| = k$. Note that the in-degree of each vertex $v \in V \setminus (V_s \cup \{s, t\})$ is equal to d, since v is a newcomer and has contacted d storage nodes for repair. To simplify the proof below, for every $v \in V_s$, we split its incoming edge into d parallel edges, so that all vertices, except s and t, in the auxiliary graph, G, have in-degree d. It is clear that G is also a DAG.

Definition 3. *Given an s-t cut $K = (S, T)$ of a DAG, $G = (V, E)$, a vertex $v \in T$ is said to be on the* boundary *of the cut, denoted by ∂K, if v is the head of any edge in the cut-set of K.*

Note that a cut $K = (S, T)$ can also be specified by the cut-set of K, or by ∂K, as there is a one-to-one correspondence between (S, T) and the cut-set of K, and also between (S, T) and ∂K.

Lemma 1. *For any s-t cut $K = (S, T)$ in an auxiliary graph G, if $t \notin \partial K$, then $|\partial K| \geq \min\{d, k\}$.*

Proof: Consider an arbitrary cut $K = (S, T)$ which satisfies $t \notin \partial K$. It is clear that $V_t \subset T$, and therefore $|T| \geq k$. First, consider the case $d \leq k$. Since G is a DAG, there is a topological ordering of the vertices, which means that the vertices are ordered one after another in a way such that $(v_i, v_j) \in E$ implies $i < j$. The first d vertices in T must belong to ∂K, since their in-degrees are all equal to d. Hence, $|\partial K| \geq d = \min\{d, k\}$. Next, consider the case $d > k$. If $|T| \geq d$, then by the same reasoning as before, $|\partial K| \geq d > k = \min\{d, k\}$. If $k \leq |T| < d$, then $\partial K = T$

and hence $|\partial K| \geq k = \min\{d, k\}$. The statement is proved by combining the two cases. ∎

The above result can be used to analyze the min-cut in the information flow graph. The proof is essentially the same as in [1, Lemma 2]. We include the proof for the sake of completeness.

Lemma 2 (Min-Cut Bound [1])**.** *Given any $G_0 \in \mathcal{G}(n, k, d, \alpha, \beta)$, the value of any s-t cut of G_0 is bounded below by*

$$\sum_{i=0}^{\min\{d,k\}-1} \min\{(d-i)\beta, \alpha\}. \tag{1}$$

Furthermore, the lower bound is tight, meaning that there exists a DC *in G_0 such that the corresponding s-t cut has value equal to* (1)*.*

Proof: Given an information flow graph G_0, consider an arbitrary cut $K_0 = (S_0, T_0)$. We can exclude the case where t belongs to ∂K_0, since all incoming edges of t have infinite capacity. We can also exclude the case where $\mathsf{In}_i \in T$ and $\mathsf{Out}_i \in S$ for some i, since the cut-value can be reduced by moving Out_i from S to T.

Now given K_0, we can construct a corresponding cut K in the auxiliary graph as follows: If either In_i or Out_i is in ∂K_0, then let v_i be in ∂K. By Lemma 1, there are at least $\min\{d, k\}$ vertices in ∂K, which implies that there are also at least $\min\{d, k\}$ vertices in ∂K_0.[1]

Since G_0 is a DAG, there is a topological ordering of its vertices. Let Out_1 be the topologically first output node in T. Then either $\mathsf{In}_1 \in \partial K_0$ or $\mathsf{Out}_1 \in \partial K_0$. If $\mathsf{Out}_1 \in \partial K_0$, then its single incoming edge contributes α to the cut-value. If $\mathsf{In}_1 \in \partial K_0$, we can exclude the case where $\mathsf{In}_1 \in V_s$, since the incoming edge of In_1 has infinite capacity. Therefore, we only need to consider the case where $\mathsf{In}_1 \notin V_s$. Its d incoming edges contribute a total of $d\beta$ to the cut-value. Combining the two cases, a value of $\min\{d\beta, \alpha\}$ is contributed to the cut-value.

Next, consider Out_j, the topologically j-th output node in T. Again either $\mathsf{In}_j \in \partial K_0$ or $\mathsf{Out}_j \in \partial K_0$. If $\mathsf{Out}_j \in \partial K_0$, then its single incoming edge contributes α to the cut-value. If $\mathsf{In}_j \in \partial K_0$, again we can exclude the case where $\mathsf{In}_j \in V_s$. Then at least $d - (j-1)$ of the incoming edges of In_j must be in the cut-set of K_0, since at most $j-1$ of its incoming edges can be connected to other out-vertices in T, namely $\mathsf{Out}_1, \mathsf{Out}_2, \ldots, \mathsf{Out}_{j-1}$. Therefore, its incoming edges contribute at least $(d-j+1)\beta$ to the cut-value. Combining the two cases, a value of $\min\{(d-j+1)\beta, \alpha\}$ is contributed to the cut-value.

Since there are at least $\min\{d, k\}$ vertices in ∂K_0 and adding any one or more vertices to T will not decrease the cut-value, the minimum cut-value is therefore given by the expression in (1).

[1]The fact that there cannot be less than $\min\{d, k\}$ vertices in ∂K_0 was taken for granted in the last paragraph of the proof of [1, Lemma 2].

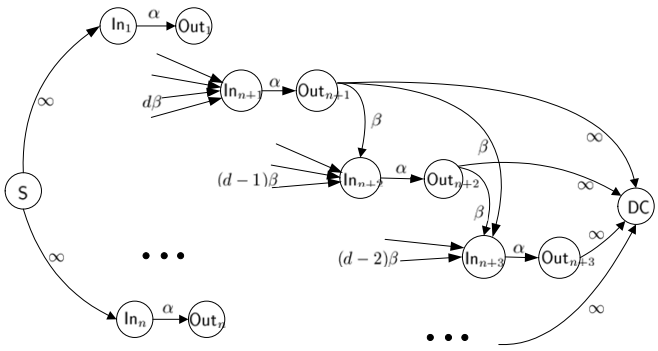

Fig. 2. Graphical illustration of the bound.

Now we show that the lower bound in (1) is tight by constructing an information flow graph and a cut achieving equality in (1). Initially, there are n storage nodes connected to S. Consider k newcomers indexed by $n+1, n+2, \ldots, n+k$. For $i = 1, 2, \ldots, k$, In_{n+i} connects to $\mathsf{Out}_{n+i-d}, \ldots, \mathsf{Out}_{n+i-1}$. Consider a data collector that connects to the last k Out_i's. A cut K that achieves the lower bound can be constructed as follows: For $i = 0, 1, \ldots, k-1$, if $(d-i)\beta \leq \alpha$, then $\mathsf{In}_i \in \partial K$; otherwise, $\mathsf{Out}_i \in \partial K$. A graphical illustration is shown in Figure 2.

∎

Following [1], we assume without loss of generality that $d \geq k$. By Lemma 2, we have the following theorem:

Theorem 3. *Let B be the size of the file stored in DSS(n, k, d, α, β). If $d \geq k$, then*

$$B \leq \sum_{i=0}^{k-1} \min\{(d-i)\beta, \alpha\}. \tag{2}$$

Proof: By the max-flow bound [5, Theorem 18.3], we have

$$B \leq \min_i \text{maxflow}(\mathsf{DC}_i). \tag{3}$$

Since $d \geq k$, by Lemma 2, the minimum cut-value is bounded below by the right-hand side of (2). By the well-known max-flow min-cut theorem [6], we have

$$\min_i \text{maxflow}(\mathsf{DC}_i) \leq \sum_{i=0}^{k-1} \min\{(d-i)\beta, \alpha\}. \tag{4}$$

The statement then follows from (3) and (4). ∎

IV. Generic Storage Code

In this section, we first present some basic concepts in network coding theory from [7]. Then, we introduce the refined information flow graph for DSS. Next, we define *generic storage code*, a concept in relation to the refined information flow graph.

A. Basic Concepts of Linear Network Code

Consider a single-source acyclic communication network and its corresponding graph, $G = (V, E)$. Let the alphabet Σ be the finite field $GF(q)$. A *linear network code* can be

specified by the set of all local encoding kernels, $\{l_{d,e} \in \Sigma : d \in \mathrm{I}(v), e \in \mathrm{O}(v), v \in V\}$.

Suppose the message to be transmitted form the source node is an ω-dimensional column vector $\boldsymbol{x}$ over $GF(q)$. We add ω imaginary edges terminating at S and assign each of them with a distinct column vector in the ω-dimensional standard basis. These vectors are referred to as the global encoding kernels of the imaginary edges. For each edge $e = (u, v) \in E$, we iteratively define its global encoding kernel by

$$\boldsymbol{g}_e \triangleq \sum_{d \in \mathrm{I}(u)} l_{d,e} \boldsymbol{g}_d.$$

The transmitted symbol on edge e is $\boldsymbol{x}^T \boldsymbol{g}_e$. For an edge set P, denote the set of the corresponding global encoding kernels by $\ker(P) \triangleq \{\boldsymbol{g}_e : e \in P\}$, and the linear span of $\ker(P)$ by

$$\mathrm{vspace}(P) \triangleq \mathrm{span}(\ker(P)).$$

For a vertex u, define

$$\mathrm{vspace}(u) \triangleq \mathrm{span}(\ker(\mathrm{I}(u))).$$

A sequence of edges $e_1, e_2, \ldots, e_n$ forms a *path* if Head(e_i) = Tail(e_{i+1}) for $1 \leq i \leq n-1$. Two paths are *edge-disjoint* if they do not have any edge in common. A set of edges is said to be *path-independent* if each edge in this set is on a path originating from an imaginary edge and these paths are edge-disjoint. An edge set P is said to be *regular* with respect to a linear network code if the global encoding kernels in $\ker(P)$ are linearly independent.

B. Refined Information Flow Graph

Given any information flow graph for a DSS, we construct a *refined information flow graph* by introducing the concept of *repair stage*. In regard to the refined information flow graph, the repair process of a node is called a repair stage. From S to In_i's is called stage -1. The original n storage nodes is said to be in stage 0. In stage $s > 0$, the out-vertex of each storage node, except the one failed in stage $s-1$, is connected to an auxiliary out-vertex by a directed edge of capacity α. The out-vertex of node i in stage s is denoted by $\mathsf{Out}_i^{(s)}$ and that in stage $s+1$ is denoted by $\mathsf{Out}_i^{(s+1)}$. Let the index of the newcomer in stage $s+1$ be k. We re-label the Out_k vertex of the newcomer as $\mathsf{Out}_k^{(s+1)}$. Since each storage node has capacity α, every edge from S to In_i with infinite capacity is replaced by an edge of capacity α. Furthermore, each edge of capacity c is replaced by c parallel edges of unit capacity. Denote the set of all the edges in stage s, except the incoming edges of data collectors, by E_s. Note that the refined information flow graph represents a single-source multi-cast acyclic network. An example with $n = 4, d = 3$ is shown in Figure 3. In the example, node 1 fails in stage 0 and node 2 fails in stage 1.

C. Generic Storage Code

Since a DC can connect to nodes in the same stage, we only need to ensure the path-independent sets of *edges in the same stage* are regular. A code that satisfies this requirement may be regarded as a restricted form of a generic network code [3], [7]. We call it *generic storage code*, which is formally defined below:

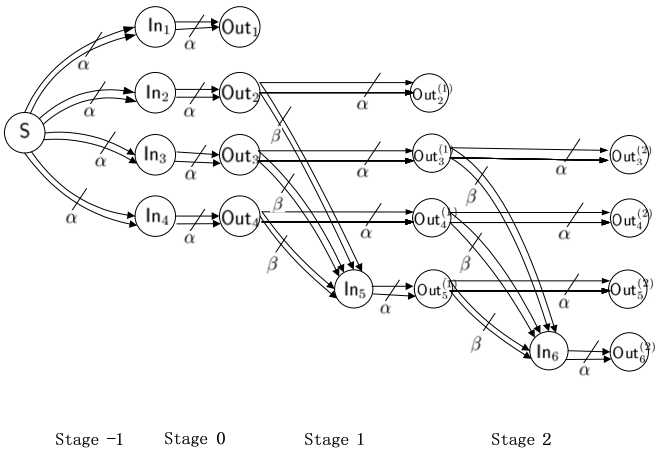

Fig. 3. An example of a refined information flow graph.

Definition 4. *An ω-dimensional linear network code on a refined information flow graph is said to be an* ω-dimensional generic storage code *if every path-independent ω-subsets of E_s is regular, for any stage $s = 0, 1, 2, \ldots$.*

V. Achievability

In this section, we prove the achievability of the capacity bound in Theorem 3 by generic storage code.

Lemma 4. *Let G be a refined information flow graph G such that there is at least one path-independent set of edges of size ω in each stage. An ω-dimensional generic storage code on G over GF(q) can be constructed, provided that $q > \binom{n\alpha+d\beta}{\omega}$.*

Proof: Let q be a prime power greater than $\binom{n\alpha+d\beta}{\omega}$. We prove the statement by mathematical induction on the number of repair stages in the refined information flow graph. We want to maintain the inductive invariant that, in any stage, any path-independent set of edges is regular.

Consider a refined information graph with no repair stage, (i.e. with only stages -1 and 0). First, note that any ω-subset of the edges in stage -1 is path-independent. We claim that there exists a linear code such that all these ω-subsets are regular. For the first ω edges in stage -1, it is clear that they can be assigned linearly independent global encoding kernels. For each of the subsequent edges in stage -1, we can pick a vector $\boldsymbol{x} \notin \mathrm{vspace}(\zeta)$, where ζ is any $(\omega-1)$-subset of edges that have already been assigned global encoding kernels. This can be done by picking a generator matrix of an ω-dimensional Reed-Solomon code of length $n\alpha$. We can also assign the global encoding kernels sequentially, since

$$\Big|\bigcup_{\zeta} \mathrm{vspace}(\zeta)\Big| \leq \binom{n\alpha}{\omega-1} q^{\omega-1} < q^{\omega}.$$

Let the global encoding kernels of α incoming edges of every In_i, $i = 1, 2, \ldots, n$ be the same as those of its α outgoing

edges. Since in stage -1, any ω-subset of the $n\alpha$ edges is regular, so is any ω-subset of the $n\alpha$ edges in stage 0.

Assume that for any refined information graph with $s \geq 0$ repair stages, (i.e., $s+1$ stages including stage 0), a generic storage code has been constructed. By definition, any path-independent ω-subset of E_s is regular with respect to the constructed network code. In stage $s+1$, there are $n-1$ auxiliary out-vertices of the surviving nodes in stage s, and one newcomer. Let the set of indices of the $n-1$ surviving nodes be S, and the index of the newcomer be k. For $i \in S$, $\mathsf{Out}_i^{(s)}$ has α incoming edges and α outgoing edges connecting to $\mathsf{Out}_i^{(s+1)}$. Let the global encoding kernels of these α outgoing edges be the same as those of the α incoming edges. Let the d helper nodes of newcomer k be indexed by $h_1, h_2, \ldots, h_d \in S$, where $h_1 < h_2 < \cdots < h_d$. Let U be the edge set which consists of all incoming and outgoing edges of $\mathsf{Out}_i^{(s)}$ for $i \in S$ and all incoming edges of $\mathsf{Out}_k^{(s+1)}$. It remains to determine the global encoding kernels for all the incoming edges and outgoing edges of In_k in such a way that any path-independent ω-subset of U is regular. This can be done by Algorithm 1, which is adapted from [5, Algorithm 19.34].

Algorithm 1 Assign Global Encoding Kernels for the Newcomer k

Input: $\{\boldsymbol{g}_e : e \in \mathrm{I}(\mathsf{Out}_i^{(s)})$ for all $i \in S\}$ and $h_1, h_2, \ldots, h_d$
Output: $\{\boldsymbol{g}_e : e \in \mathrm{I}(\mathsf{In}_k)$ or $e \in \mathrm{I}(\mathsf{Out}_k^{(s+1)})\}$

1: $U_0 := \{e \in \mathrm{I}(\mathsf{Out}_i^{(s)})$ for all $i \in S\}$;
2: **for** $i := 1, 2, \ldots d$ **do**
3: **for** $j := 1, 2, \ldots \beta$ **do**
4: $e :=$ the j-th incoming edges of In_k from $\mathsf{Out}_{h_i}^{(s)}$;
5: Choose a vector $\boldsymbol{x} \in \mathrm{vspace}(\mathsf{Out}_{h_i}^{(s)})$ such that $\boldsymbol{x} \notin \mathrm{vspace}(\zeta)$, where ζ is any ω-subset of U_0 such that ζ is regular and $\mathrm{vspace}(\mathsf{Out}_{h_i}^{(s)}) \not\subset \mathrm{vspace}(\zeta)$;
6: $\boldsymbol{g}_e := \boldsymbol{x}$ and $U_0 := U_0 \cup \{e\}$;
7: **end for**
8: **end for**
9: **for** $j := 1, 2, \ldots \alpha$ **do**
10: $e :=$ the j-th incoming edges of $\mathsf{Out}_k^{(s+1)}$;
11: Choose a vector $\boldsymbol{x} \in \mathrm{vspace}(\mathsf{In}_k)$ such that $\boldsymbol{x} \notin \mathrm{vspace}(\zeta)$, where ζ is any ω-subset of U_0 such that ζ is regular and $\mathrm{vspace}(\mathsf{In}_k) \not\subset \mathrm{vspace}(\zeta)$;
12: $\boldsymbol{g}_e := \boldsymbol{x}$ and $U_0 := U_0 \cup \{e\}$;
13: **end for**

By construction, it can be seen that any path-independent ω-subset of U is regular. In the algorithm, the vector $\boldsymbol{x}$ in line 5 can always be found. To see this, notice that there are at most $(n\alpha + d\beta)$ edges in U_0 and the number of possible choices of ζ is at most $\binom{n\alpha+d\beta}{\omega}$. Denote the dimension of $\mathrm{vspace}(\mathsf{Out}_{h_i}^{(s)})$ by ν. Since $\mathrm{vspace}(\mathsf{Out}_{h_i}^{(s)}) \not\subset \mathrm{vspace}(\zeta)$, the dimension of $\mathrm{vspace}(\mathsf{Out}_{h_i}^{(s)}) \cap \mathrm{vspace}(\zeta)$ is less than or equal to $\nu - 1$. Thus,

$$\left|\mathrm{vspace}(\mathsf{Out}_{h_i}^{(s)}) \cap \Big(\bigcup_{\zeta} \mathrm{vspace}(\zeta)\Big)\right| \leq \binom{n\alpha + d\beta}{\omega} q^{\nu-1} < q^{\nu}.$$

Likewise, the vector $\boldsymbol{x}$ in line 11 can also be found. ∎

Theorem 5. *A file with size $B = \omega \triangleq \sum_{i=0}^{k-1} \min\{(d-i)\beta, \alpha\}$ can be stored in a DSS(n, k, d, α, β) by an ω-dimensional generic storage code over GF(q), where $q > \binom{n\alpha+d\beta}{\omega}$.*

Proof: We have shown the existence of generic storage code in Lemma 4. Now we show any collector can retrieve the file based on the generic storage code. By Lemma 2, there are at least ω disjoint paths terminating at any k out-vertices, in every stage $s \geq 0$. Thus there are at least one path-independent set, say P with size ω within the incoming edges of these k out-vertices. By the definition of generic storage code, the dimension of $\ker(P)$ is ω, and the file with size $B = \omega$ can be decoded. ∎

VI. Concluding Remarks

We present a more rigorous derivation of the min-cut bound, and make explicit that finding the minimum number of nodes on the cut-boundary is a necessary step of the proof. We hope that this provides more insight to the problem, which may be useful for analyzing min-cut bounds for future DSS models with different assumptions. To prove the achievability of the min-cut bound, a new concept called generic storage code is introduced. We remark that the corresponding induction invariant in Lemma 4 is larger than necessary in proving the achievability result. Instead, the induction invariant found in [2] requires a smaller inductive invariant. Hence, the field size requirement in [2] is smaller than that of generic storage code. Another approach, which uses exact-repair regenerating codes, can also obtain field sizes that remain constant for an unbounded number of failures. Exact-repair regenerating codes, however, do not exist for the entire tradeoff curve [8]. Among all these approaches, the use of generic storage code is more flexible in the sense that it does not rely on the structure of the information flow graph, rendering it more applicable to other DSS models.

References

[1] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 56, no. 9, pp. 4539–4551, Sep. 2010.

[2] Y. Wu, "Existence and construction of capacity-achieving network codes for distributed storage," *IEEE J. on Selected Areas in Commun.*, vol. 28, no. 2, pp. 277–288, Feb. 2010.

[3] S.-Y. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Trans. Inf. Theory*, vol. 49, no. 2, pp. 371–381, Feb. 2003.

[4] R. Koetter and M. Médard, "An algebraic approach to network coding," *IEEE/ACM Trans. Networking*, vol. 11, no. 5, pp. 782–795, Oct. 2003.

[5] R. W. Yeung, *Information Theory and Network Coding*. Springer, 2008.

[6] L. K. Ford Jr. and D. R. Fulkerson, *Flows in Network*. Princeton University Press, 1962.

[7] M. Tan, R. W. Yeung, S.-T. Ho, and N. Cai, "A unified framework for linear network coding," *IEEE Trans. Inf. Theory*, vol. 57, no. 1, pp. 416–423, Jan. 2011.

[8] C. Tian, "Characterizing the rate region of the (4,3,3) exact-repair regenerating codes," *IEEE J. on Selected Areas in Commun.*, vol. 32, no. 5, pp. 967–975, May 2014.