When Can Intelligent Helper Node Selection Improve the Performance of Distributed Storage Networks?

Imad Ahmad, *Member, IEEE,* and Chih-Chun Wang, *Senior Member, IEEE*

Abstract*—The concept of distributed storage networks (DSNs) mostly follows two main modeling assumptions: Any k out of n surviving nodes should be able to reconstruct the protected file; and if one node fails, the replacement node can access d *helper nodes* to repair its content either *functionally* or *exactly.

Two major existing approaches for DSNs are the so-called *regenerating codes* (RCs) and *locally repairable codes* (LRCs), which have different design philosophies and focus on distinct applications. Instead of being limited by the framework of either RCs or LRCs, this work answers a fundamental question for general DSNs: For an arbitrarily given (n, k, d) value, whether there exists an intelligent helper node selection design that can strictly improve the storage-bandwidth tradeoff when compared to naive *blind* helper selection. Surprisingly, the answer is negative for a large set of (n, k, d) values. Namely, for those (n, k, d) values even the best helper selection design offers no gain over a blind solution. We call those (n, k, d) values *indifferent-to-helper-selection* (ITHS). The main contribution of this work is a necessary and sufficient condition that characterizes whether an (n, k, d) value is ITHS. As a fundamental study, this work assumes functional repair with unlimited computing power for encoding/decoding and focuses on the fundamental performance limits of intelligent helper selection. A new helper selection scheme, termed *family helper selection*, is proposed and used in the achievability analysis. For some scenarios, the proposed scheme is indeed optimal (as good as any helper selection one can design).

***Index Terms*—Distributed storage networks, regenerating codes, locally repairable codes, family helper selection, helper nodes, network coding**

I. Introduction

THE need for storing very large amounts of data reliably is one of the major reasons that has pushed for distributed storage systems. Examples of distributed storage systems include data centers [9] and peer-to-peer systems [4], [23]. One way to protect against data loss is by replication coding, i.e, if a disk in the network fails, it can be replaced and its data can be recovered from a replica disk. Another way is to use maximum distance separable (MDS) codes. Recently, regenerating codes (RCs) [7] have been proposed to further reduce the repair-bandwidth of MDS codes. Locally repairable codes (LRC) [10], [17], [18] on the other hand were later proposed to use a small number of helper nodes d during repair, which is in contrast with RCs that were originally designed for large d (i.e., $d \geq k$).

Regardless of the coding technique being used, e.g., replication coding, MDS codes, RCs, LRCs, or any other existing constructions [12], [30], in general a distributed storage network (DSN) operates under the following two major assumptions: (1) any k out of n surviving nodes should be able to reconstruct the protected file; (2) if only one node fails, the replacement node can access d *helper nodes* to repair its content[1] either in a *functional* sense or in an *exact* sense. The (n, k, d) values of interest depend on the underlying application. For example, peer-to-peer storage applications may require codes with large n values due to its peer-to-peer nature; applications with a very hostile/noisy environment may require codes of small k (or equivalently large $n-k$) for the added level of protection; and codes of small d could lower I/O overhead during repair and thus mitigate some performance bottlenecks in cloud storage systems. Codes of different (n, k, d) values are generally not directly comparable since they likely have different application scenarios. For the same (n, k, d) value, one way to compare the performance of two DSN codes is to analyze their corresponding *storage and repair-bandwidth tradeoff* curves [7], the main performance metric adopted in this paper.

For any arbitrarily given (n, k, d) value, one critical component of a DSN code is how to choose the d helpers, out of the $(n-1)$ remaining nodes, during the repair operations. For example, in the original storage versus repair-bandwidth analysis of RCs [7], it is assumed that the newcomer does not distinguish/choose its helpers. That is, the newcomer *blindly* chooses its d helpers. We term such a solution the *blind helper selection scheme*, which will be formally defined in Section II-B. In contrast, the original design of LRCs [10], [17], [18] uses a fixed helper selection table for each newcomer. Namely, each newcomer has a predetermined fixed set of d helpers. We term such type of helper selection the *stationary helper selection*, also see our definition in Section II-B. Intuitively, intelligently designing the helpers for each newcomer should enhance the performance of a DSN code since the newcomer can now choose to access

Manuscript received April 14, 2017; accepted November 4, 2017. This work was supported in parts by NSF under Grant CCF-0845968, Grant CNS-0905331, Grant CCF-1422997, Grant ECCS-1407604, and Grant CCF-1618475. Part of the results was presented in the 2014 Allerton Conference on Communication, Control, and Computing.

Communicated by A. Dimakis, Associate Editor for Coding Techniques.

I. Ahmad is currently with AT&T Labs {imadfahmad@gmail.com} and C.-C. Wang is with the School of Electrical and Computer Engineering, Purdue University, West Lafayette {chihw@purdue.edu}. This work was completed while I. Ahmad was a Ph.D. student at Purdue University, West Lafayette.

[1]The replacement node is often called the *newcomer* and the helper nodes are called *helpers*.

only those "good" helpers among the remaining nodes[2]. On the other hand, blind helper selection offers the maximum level of flexibility/reliability in the sense that the newcomer can access "any" d helpers, which is very beneficial when some nodes may be temporarily unsuitable to serve as helpers (e.g., multiple node failures, congestion at a part of the DSN, etc.). Therefore, depending on how the system designer values performance versus flexibility/reliability, one can design DSN codes with different helper selection strategies.

Such an observation on the benefits and costs of different helper selection schemes prompts the following fundamental question:

> Question 1: For an arbitrarily given (n, k, d) value, what is the performance/reliability tradeoff among different helper selection methods?

As a first step toward answering this open fundamental question, this work focuses on the following closely related, more rigorously formulated question that can be viewed as a zeroth-order approximation of Question 1:

> Question 2: For an arbitrarily given (n, k, d) value, does there exist an intelligent helper selection design that can strictly improve the storage-bandwidth tradeoff when compared to *blind helper selection*?

To investigate Question 2, we assume functional repair and unlimited computing power for encoding/decoding so that we can isolate the fundamental impact of intelligent helper selection. Specifically, we compare the optimal DSN performance under a given intelligent helper selection scheme and the best corresponding code design, versus the optimal DSN performance under blind helper selection and the best corresponding codes. Namely, only the helper selection scheme varies and the codes are assumed to be optimally constructed.[3]

The performance improvement is defined by whether the storage-bandwidth tradeoff curve is strictly improved by intelligent helper selection. The tradeoff curves are characterized by a new min-cut-based analysis similar to the one used in [7].

Surprisingly, our results show that for a large set of (n, k, d) values, the answer to Question 2 is negative. Namely, for those (n, k, d) values even the best helper selection design offers no gain over a blind solution. We call those (n, k, d) values *indifferent-to-helper-selection* (ITHS), which will be formally defined in Section III-C. We then derive a necessary and sufficient condition that characterizes whether an (n, k, d) value is ITHS. Like most information-theoretic studies, our results consist of two parts. [The converse direction:] We prove that for a set of (n, k, d) values even the best intelligent helper selection scheme has the same performance as the blind helper selection. [The achievability part:] We prove that for any (n, k, d) value that is *not* in that set, there exists a helper selection scheme such that its storage-bandwidth tradeoff curve is strictly better than that of the blind helper selection.

Our converse and achievability results together provide a tight characterization that exhaustively determines whether an (n, k, d) value is ITHS or not. For comparison, there is no existing study that is dedicated to the converse direction, i.e., proving no intelligent helper selection can do better than blind selection. In the achievability direction, several existing results, e.g. [8], [17], can be viewed as achievability results that show that at least for *some special* (n, k, d) *values*[4] there exists an intelligent helper selection that strictly outperforms blind helper selection. The main contribution of this work is to examine *all* (n, k, d) *values* and answer Question 2 for each (n, k, d) combination by either proving the converse or the achievability. By answering Question 2 for arbitrary (n, k, d), our results provide a rigorous benchmark/guideline when designing the next-generation smart helper selection solutions.

As a byproduct of the achievability direction, a new helper selection scheme, termed *family helper selection*, is proposed and analyzed. We prove that for some scenarios family helper selection is indeed optimal, i.e., as good as any helper selection one can design. The optimality of a helper selection scheme will be formally defined in Section III-C.

The rest of this paper is organized as follows. Section II provides the basic model of DSNs and a rigorous definition of helper selection schemes. Section III outlines the graph-based approach that will be used to study the helper selection problem. Section IV gives a preview of our main results in this paper. Section V states the main results of this paper. Section VI proposes two new helper selection schemes, termed the family helper selection and family-plus helper selection schemes. Section VII concludes this paper. In order not to disrupt the flow of the paper, we state our results/theorems in the main body of the paper and relegate most of the proofs to the appendices. The performance analysis of the proposed schemes are also relegated to the appendix.

II. A General Model of Distributed Storage Networks

A. The Parameters of a Distributed Storage Network

1) Parameters n and k: We denote the total number of nodes in a storage network by n. For any $1 \leq k \leq n-1$, we say that a code can satisfy the reconstruction requirement if any k nodes can be used to reconstruct the original data/file. For example, consider a network of 7 nodes. A $(7, 4)$ binary Hamming code can be used to protect the data. We say that the Hamming code can satisfy the reconstruction requirement for $k = 6$ since any 6 nodes can construct the original file. By the same definition, the Hamming code can also satisfy the reconstruction requirement for $k = 5$, but cannot satisfy the reconstruction requirement for $k = 4$ (since a Hamming code is not an MDS code). The smallest k value that the $(7, 4)$ Hamming code can satisfy is thus 5. In this work, we denote this smallest k value as k^* and we thus say that the $(7, 4)$

[2]See Appendix A for an example illustrating choosing the "good" helpers.

[3]How to implement the "best code" under a given helper selection method is not our main focus, although our achievability analysis does involve some explicit code construction and the code construction results are reported separately in a companying paper [3].

[4]For example, the construction in [8], [17] requires $n \cdot d$ be even, and the construction in [17], used to achieve LRCs bound, requires that n is divisible by $(d + 1)$. In our work, we consider arbitrary (n, k, d) combinations.

Hamming code[5] has $(n, k^*) = (7, 5)$. It is important to see the distinction that the value of k is related to the desired protection level of the system while the value of k^* is related to the actual protection level offered by the specific distributed storage code implementation.

2) Parameter d: We denote the number of nodes that a newcomer can access during repair by d. For example, [7] proposes the concept of RCs that achieves the design goal $(n, k, d) = (10, 7, 9)$. Specifically, each newcomer can access $d = 9$ helpers and any $k = 7$ nodes can be used to reconstruct the original file. At the same time, [7] also provides RCs to achieve the design goal when $(n, k, d) = (10, 7, 5)$. However, those RCs can be an overkill in this scenario of $(n, k, d) = (10, 7, 5)$ since any RC construction in [7] that can achieve $(n, k, d) = (10, 7, 5)$ can always achieve $k^* = d = 5$. As a result, even though the high-level design goal is to only protect against $10 - 7 = 3$ failures, the RC in [7] cannot take advantage of this relatively low protection-level requirement since the construction in [7] always has $k^* \leq d = 5$, which is strictly smaller than the design requirement $k = 7$.

Note that the above observation does not mean that the system designer should never use the RCs in [7] when the design goal is $(n, k, d) = (10, 7, 5)$. The reason is that RCs in [7] have many other advantages that may be appealing in practice, e.g., some very efficient algebraic code construction methods [26], allowing repair with $(n - d)$ simultaneous failures, and admitting efficient collaborative repair when more than one node fails [27]. The fact that $k^* \leq d$ for the RCs in [7] simply means that when the requirement is $(n, k, d) = (10, 7, 5)$, the system designer should be aware that the RCs in [7] do not take full advantage of the relatively loose required protection level since we have in this scenario $k > d \geq k^*$. The development of the LRCs in [10], [17], [18] is in part motivated by the need of DSN codes for the applications with $k \gg d$ when the original RCs always have $k^* \leq d$.

In this work, we are interested in analyzing the optimal DSN performance under the combination of an arbitrarily given helper selection scheme with the best corresponding code implementation. Based exclusively on the min-cut analysis, our results essentially characterize the performance of the best possible codes without delving into their actual construction. For example, the best codes in this work are not limited to any specific form, e.g., RCs, LRCs, etc. Since the actual protection level k^* depends on the underlying code implementation, in all our discussions we focus on the design target k instead of the actual performance parameter k^*. For any given (n, k, d) values, the goal is to compare the best performance of any possible helper selection scheme that can still satisfy the desired (n, k, d) values regardless whether they offer over-protection ($k > k^*$) or not.

3) Repair Mode: We assume functional repair throughout this work unless specified otherwise.

[5]In the error-correcting coding literature, k represents the number of systematic bits. In the distributed storage literature, k^* represents the minimum number of nodes that can still guarantee data reconstruction. Both definitions are different from the k definition in this work, which is the target data protection level.

4) The Parameter Tuple (n, k, d) and Other Notation: From the above definitions, the n, k, and d values must satisfy

$$2 \leq n, \quad 1 \leq k \leq n - 1, \quad \text{and} \quad 1 \leq d \leq n - 1. \qquad (1)$$

In all the results of this work, we assume *implicitly* that the n, k, and d values satisfy (1). The overall file size is denoted by $\mathcal{M}$. The storage size for each node is α, and during the repair process, the newcomer requests β amount of traffic from each of the helpers. The total repair-bandwidth is thus $\gamma \triangleq d\beta$. We use the notation $(\cdot)^+$ to mean $(x)^+ = \max(x, 0)$. We also define the indicator function as follows

$$1_{\{B\}} = \begin{cases} 1 & \text{if condition } B \text{ is true} \\ 0 & \text{otherwise} \end{cases}.$$

In this work, we consider exclusively single node failures at any given time. The setting of multiple simultaneous failed nodes [8], [13], [27] is beyond the scope of this work. We consider the multiple failures scenario in a separate work, see [2].

B. Defining Helper Selection Schemes

The previous subsection defined the basic parameters $(n, k, d, \mathcal{M}, \alpha, \beta)$ of a DSN. We now rigorously define *helper selection* in this subsection.

We consider the most general form of helper selection. That is, the helper selection at current time τ can depend on the history of the failure patterns for all the previous time slots 1 to $(\tau - 1)$. As a result, we define the helper selection scheme of a DSN code as follows.

Definition 1: For each time τ, define F_τ as the index of the failed node at time τ. Define $\mathcal{A}_\tau \triangleq \{1, 2, \cdots, n\} \backslash \{F_\tau\}$ as the indices of all *available* nodes at time τ that can serve as helpers for newcomer F_τ. Define $2^{\mathcal{A}_\tau}$ as the collection of all subsets of $\mathcal{A}_\tau$, also known as the powerset of $\mathcal{A}_\tau$. Define $\left[2^{\mathcal{A}_\tau}\right]_d$ as the collection of all subsets of $\mathcal{A}_\tau$ such that each subset contains exactly d nodes.

Definition 2: The helper selection scheme of a DSN code is defined by an infinite series of functions $D_\tau(\cdot)$ for $\tau \in \{1, 2, \cdots\}$ such that for each τ, the function $D_\tau(\{F_l\}_{l=1}^{\tau})$ takes F_l, the index of the failed node at time l, for all $l = 1$ to τ, as input and returns a non-empty subset of $\left[2^{\mathcal{A}_\tau}\right]_d$.

Namely, the DSN helper selection rule $D_\tau(\cdot)$ at time τ will take all the history of the failed nodes as input and output a non-empty collection of subsets of available nodes (i.e., $\mathcal{A}_\tau$) such that each subset has exactly d nodes.

After $D_\tau(\cdot)$ outputs a collection of candidate helper sets, the newcomer F_τ at time τ will choose *arbitrarily* one of them, access the d nodes in the chosen helper set, and request data for repair. The above definition of helper selection is of the most general form and includes all existing designs as special cases.

Example 1: The blind helper selection scheme in the original RCs paper [7] can be viewed as setting

$$D_\tau(\{F_l\}_{l=1}^{\tau}) = \left[2^{\mathcal{A}_\tau}\right]_d. \qquad (2)$$

Namely, $D_\tau(\{F_l\}_{l=1}^{\tau})$ returns *all* the size-d subsets of $\mathcal{A}_\tau$ and the newcomer can choose any one of them. Equivalently, it allows the newcomer to blindly choose any d helpers out of the $(n-1)$ available nodes in $\mathcal{A}_\tau$.

Example 2: The table-based helper selection scheme in the LRCs works [10], [16]–[18] can be viewed as setting

$$D_\tau(\{F_l\}_{l=1}^{\tau}) = f(F_\tau) \tag{3}$$

for some function $f : \{1, \cdots, n\} \mapsto \left(2^{[2^{\mathcal{A}_\tau}]_d}\right) \backslash \{\emptyset\}$. Specifically, the function $f(\cdot)$ takes the index of the newcomer F_τ as input, and returns a non-empty collection of size-d subsets of $\mathcal{A}_\tau$. For example, in [17] the function $f(\cdot)$ outputs exactly one size-d subset of $\mathcal{A}_\tau$ while in [16] the function $f(\cdot)$ outputs multiple size-d subsets. The main difference between (3) and Definition 2 is that the function $f(\cdot)$ does not depend on the time index τ and thus can be viewed as a stationary table that describes how to choose a fixed set of d helpers for each newcomer, regardless of the time instant of interest.

For future reference, we use the following different terms to refer to different types of helper selections.

Definition 3: We call any helper selection scheme that can be characterized by Definition 2 a *dynamic helper selection* scheme. The collection of all such schemes is denoted as $\mathcal{DHS}$.

Definition 4: We call any helper selection scheme that is of the form in Eq. (3) a *stationary helper selection* scheme. The collection of all such schemes is denoted as $\mathcal{SHS}$.

Definition 5: The $D_\tau(\cdot)$ of a blind helper selection scheme is given by Eq. (2). We use BHS to denote a blind helper selection scheme. Note that unlike $\mathcal{DHS}$ and $\mathcal{SHS}$ that describe different *classes* of schemes, BHS is one specific *instance* of helper selection design.

The class $\mathcal{DHS}$ is the most general class of helper selection and it allows newcomer F_τ to adaptively and dynamically select its helpers for every time instant τ, thus the name *dynamic* helper selection. The helper selection class $\mathcal{SHS}$ uses the same helper choices over time, thus the name *stationary* helper selection. By definition, one always has

$$\mathcal{DHS} \supseteq \mathcal{SHS} \ni \text{BHS}. \tag{4}$$

C. Existing Results With Different Helper Selection Methods

We now make some detailed comparison to existing results. Regenerating codes (RCs) are distributed storage codes that minimize the repair-bandwidth (given a storage constraint) while assuming blind helper selection (2) is used. In comparison, codes with local repair or (when with all-symbol locality) *locally repairable codes (LRC)*, recently introduced in [10], are codes that minimize the number of helpers participating in the repair of a failed node while assuming stationary helper selection (3) is used. Subsequent development has been done on LRCs in [13], [14], [17], [22], [28], [29].

Table I compares the setting of the original RCs and LRCs. First introduced in [7], original RCs were proposed under the functional repair scenario, i.e., nodes of the storage network are allowed to store any combination of the original packets as long as the reliability requirement is satisfied. In subsequent works [5], [6], [15], [19]–[21], [24]–[26], [33], RCs were considered under the exact repair scenario in which nodes have to store the same original packets at any given time. In contrast, LRCs are almost always considered under the exact repair scenario. In terms of the helper selection mechanisms, the original RCs are based on blind helper selection, while LRCs are based on stationary helper selection. The (n, k, d) range of RCs versus LRCs is also listed. The original RCs were designed for storage networks with large d values, whereas LRCs are designed for small d values.

Our approach is not directly comparable to either RCs or LRCs as we focus on exploring the fundamental effects of helper selection under arbitrary (n, k, d) values, not the actual code construction.[6] For reference, Table I also lists the setting used in this work. In this work, we assume unlimited computing power in encoding/decoding and thus functional repair in our derivation. We are not limited to any specific helper selection methods but are interested in quantifying the performance of the best possible dynamic helper selection scheme specified in Definition 2. We are also not restricted to any special subset of (n, k, d) values but are interested in exhaustively examining all (n, k, d) values, including both the converse and the achievability analysis. Our main contribution is to decide for each (n, k, d) value whether it is *indifferent-to-helper-selection* (ITHS).

Compared to this work, the closest setting in the existing literature is in [11]. That work finds upper bounds on the file size $\mathcal{M}$ when $\alpha = d\beta$ and $\alpha = \beta$ for functional repair with the best possible dynamic helper selection scheme. However, [11] only considers the special case of $k = n-1$. Also, it is not clear whether the provided upper bounds for $k = n-1$ are tight or not. A byproduct of the results of this work shows that the upper bounds in [11] are tight in some cases and loose in others, see Corollaries 4 and 5 in Appendix K.

III. A Graph-Based Analysis of Helper Selection

The previous section defined the DSN model under consideration and the corresponding helper selection schemes. The rest of this work, including the objective functions of ours, the converse, and the achievability results, is based exclusively on the *information-flow-graph-based analysis* of the DSN model. Strictly speaking, a graph-based min-cut analysis alone is not sufficient to characterize the storage-bandwidth tradeoff curve of DSNs. Only when the min-cut analysis is complemented by explicit code design can the tradeoff curve be completely characterized. See the discussion in [32]. Since the scope of the graph-based results is already substantial,[7] the code construction results of ours will be reported in a separate paper [3].

In this section, we rigorously formulate the graph-based performance analysis and introduce some necessary notation.

[6]Another existing work [13], [14] uses RCs in the construction of the LRCs (as local codes) in an attempt to examine the repair-bandwidth performance of LRCs. This line of work again focuses on explicit code construction and analysis, which can serve as part of the achievability results for special (n, k, d) values without any exploration of the corresponding converse part.

[7]In a similar fashion as in the original RCs work, the graph-based analysis and the corresponding code construction are reported in two separate papers, [7] and [32], respectively.

TABLE I
THE COMPARISON TABLE AMONG BLIND-REPAIR REGENERATING CODES, LOCALLY REPAIRABLE CODES, AND THE APPROACH ADOPTED IN THIS PAPER

	Original RCs [7], [15], [19]–[21], [24], [26], [33]	Locally Repairable Codes [6], [10], [13], [14], [17], [22], [28], [29]	This work studies the effects of different helper selection methods.
Repair Mode	Functional/Exact Repair	Exact Repair	Functional Repair
Helper Selection	Blind	Stationary (Fixed over time)	Dynamic (helper choices may depend on failure history)
(n, k, d) range	Designed for $k \leq d$	Designed for $k > d$	Examines the effects for all (n, k, d) values
Contribution	Storage/repair-bandwidth characterization for the worst possible helper selection	Storage/repair-bandwidth characterization for the specific stationary helper selection of the proposed exact-repair local code	Examines the storage/repair-bandwidth tradeoff curves and decides whether an (n, k, d) value is *indifferent-to-helper-selection*

A. The Information-Flow Graph and the Corresponding Definitions

As discussed in [7], the performance of a distributed storage network can be characterized by the concept of information flow graphs (IFGs). IFGs depict the storage in the network and the communication that takes place during repair. For readers who are not familiar with IFGs, we provide its detailed description in Appendix B for completion.

Intuitively, each IFG reflects one unique history of the failure patterns and the helper selection choices from time 1 to $(\tau - 1)$ [7]. Consider any given helper selection scheme $H \in \mathcal{DHS}$. Since there are infinitely many different failure patterns (since we consider $\tau = 1$ to ∞), there are infinitely many IFGs corresponding to the same given helper selection scheme H.

Specifically, at any time τ the helper selection rule $\mathcal{D}_\tau(\cdot)$ will output a collection of size-d subsets of $\mathcal{A}_\tau$, say I such subsets have been returned. Then the newcomer F_τ can arbitrarily select one of the I subsets and access the corresponding d helpers. Each of these I choices will represent one way of "growing the IFG." There are thus I different ways how an IFG can grow from time $(\tau - 1)$ to time τ for the given newcomer F_τ. For example, if the LRCs (considering the case with single failure) in [10], [17], [18] are used, then $\mathcal{D}_\tau$ only returns 1 size-d subset, i.e., $I = 1$. Therefore, there is only one way[8] of growing the IFG given F_τ. However, if BHS in [7] is used, then $\binom{n-1}{d}$ different size-d subsets are returned. There are thus $\binom{n-1}{d}$ different ways of growing the IFG given F_τ.

Definition 6: For any DSN with (n, k, d, α, β) and any helper selection scheme $H \in \mathcal{DHS}$, we denote the collection of all IFGs that can possibly be constructed under H by $\mathcal{G}_H(n, k, d, \alpha, \beta)$. We define $\mathcal{G}(n, k, d, \alpha, \beta) \triangleq \bigcup_{\forall H \in \mathcal{DHS}} \mathcal{G}_H(n, k, d, \alpha, \beta)$ as the collection of the IFGs generated by all helper selection schemes one can possibly design.

Lemma 1: By the above definition, we have

$$\mathcal{G}_{\mathrm{BHS}}(n, k, d, \alpha, \beta) = \bigcup_{\forall H \in \mathcal{DHS}} \mathcal{G}_H(n, k, d, \alpha, \beta).$$

Proof: [The $\subseteq$ direction:] This direction is straightforward due to (4). [The $\supseteq$ direction:] Since the $D_\tau(\cdot)$ of BHS returns all size-d subsets of $\mathcal{A}_\tau$, the IFG of BHS can grow in any possible way. Therefore, it contains all IFGs generated by any $H \in \mathcal{DHS}$. ■

For easier reference, we sometimes drop the input argument and use $\mathcal{G}$, $\mathcal{G}_H$, and $\mathcal{G}_{\mathrm{BHS}}$ as shorthands for $\mathcal{G}(n, k, d, \alpha, \beta)$, $\mathcal{G}_H(n, k, d, \alpha, \beta)$, and $\mathcal{G}_{\mathrm{BHS}}(n, k, d, \alpha, \beta)$, respectively. Note that by Lemma 1 we always have $\mathcal{G} = \mathcal{G}_{\mathrm{BHS}}$.

B. The Graph-Based Min-Cut Analysis for DSNs

Given an IFG $G \in \mathcal{G}$ and a data collector $t \in \mathrm{DC}(G)$, we use $\mathrm{mincut}_G(s, t)$ to denote the *minimum cut value* [31] separating s, the root node (source node) of G, and t.

The two papers [7] and [32] jointly prove that if BHS is used, then a DSN code with parameters (n, k, d, α, β) can protect a file of size $\mathcal{M}$ if and only if

$$\min_{G \in \mathcal{G}_{\mathrm{BHS}}} \min_{t \in \mathrm{DC}(G)} \mathrm{mincut}_G(s, t) \geq \mathcal{M}. \tag{5}$$

Namely, characterizing the performance of the BHS scheme is equivalent to characterizing the min-cut value in the left-hand side of (5). Another important contribution of [7] is a closed-form expression of the left-hand side of (5):

$$\min_{G \in \mathcal{G}_{\mathrm{BHS}}} \min_{t \in \mathrm{DC}(G)} \mathrm{mincut}_G(s, t) = \sum_{i=0}^{k-1} \min((d - i)^+ \beta, \alpha), \tag{6}$$

which allows us to numerically check whether (5) is true and thus explicitly quantifies the performance of the BHS scheme.

By a similar analysis as in [7] and [1], one can also prove that if a helper selection scheme $H \in \mathcal{DHS}$ is used, then a DSN code with parameters (n, k, d, α, β) can protect a file of size $\mathcal{M}$ only if

$$\min_{G \in \mathcal{G}_H} \min_{t \in \mathrm{DC}(G)} \mathrm{mincut}_G(s, t) \geq \mathcal{M}. \tag{7}$$

Note that unlike the BHS case, for general $H \in \mathcal{DHS}$ only the "only if" direction has been established. Whether the "if" direction is true remains an open question, the proof of which, if true, likely requires explicit code construction that is in parallel with the graph-based analysis derived in this work.[9]

The rest of this work focuses on quantifying the left-hand side of (7) and uses the corresponding graph-based analysis

[8]Obviously, different F_τ nodes will lead to different ways of growing the IFG.

[9]The general belief in the community is that the "if" direction is also true since there exists no contradicting evidence in all the known functional repair results of DSNs.

(7) as an equivalent characterization of the performance of a DSN code, which may not be true if the "if" direction (7) turns out to be false. In a companying paper [3], the use of (7) is justified for the case of $\alpha = d\beta$ with a new explicit code construction that complements the graph-based analysis in this work.

C. Optimality and Other Notation

In Section II, we defined the DSN model and the helper selection schemes. In Sections III-A and III-B, we formulated the IFGs, the corresponding graph-based analysis, and discussed why the graph-based condition (7) is used to characterize the performance of a DSN code under helper selection scheme H. We are now ready to define the main objectives of this work.

Definition 7: A parameter tuple (n, k, d) of DSNs is *indifferent-to-helper-selection* (ITHS) if for all $\alpha, \beta > 0$ and all $H \in \mathcal{DHS}$, we have

$$\begin{aligned}&\min_{G\in\mathcal{G}_H}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}_G(s,t)\\&=\min_{G\in\mathcal{G}_{\mathrm{BHS}}}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}_G(s,t).\end{aligned}\tag{8}$$

Note that by Lemma 1 we have $\mathcal{G}_H \subseteq \mathcal{G}_{\mathrm{BHS}}$ for all $H \in \mathcal{DHS}$. Therefore, we always have

$$\begin{aligned}&\min_{G\in\mathcal{G}_H}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}_G(s,t)\\&\geq\min_{G\in\mathcal{G}_{\mathrm{BHS}}}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}_G(s,t).\end{aligned}\tag{9}$$

Recall that (7) characterizes the largest file size the helper selection scheme H can protect. (9) thus implies that BHS, the simplest helper selection scheme, also has the worst performance since the file size it can protect is the smallest. On the other hand, Definition 7 implies that for the set of (n, k, d) values that are ITHS, even the best helper selection scheme $H \in \mathcal{DHS}$ cannot protect a file size that is strictly greater than the file size protected by BHS, regardless of the (α, β) value. Therefore, such an (n, k, d) tuple is indifferent to the underlying helper selection scheme. For the ITHS (n, k, d), there is thus no need to design intelligent helper selection rules $D_\tau(\cdot)$. The main contribution of this work is to derive a necessary and sufficient condition for the set of ITHS (n, k, d) values.

Definition 8: For any given (n, k, d) value, a helper selection scheme $H \in \mathcal{DHS}$ is *optimal* if for any helper selection scheme $\tilde{H} \in \mathcal{DHS}$ and any (α, β), the following is true

$$\begin{aligned}&\min_{G\in\mathcal{G}_H}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}_G(s,t)\\&\geq\min_{G\in\mathcal{G}_{\tilde{H}}}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}_G(s,t).\end{aligned}$$

Definition 8 implies that an optimal helper selection scheme H can protect the largest file size $\mathcal{M}$ among all possible helper selection schemes $\tilde{H}$. Using Definition 8, one can have an equivalent definition of ITHS:

Corollary 1: An (n, k, d) value is *indifferent-to-helper-selection* if and only if BHS is optimal for this (n, k, d) value.

Definition 9: For any given (n, k, d) value, a helper selection scheme $H \in \mathcal{DHS}$ is *bandwidth-optimal*, if for any helper selection scheme $\tilde{H} \in \mathcal{DHS}$ and any β, the following is true

$$\begin{aligned}&\max_{\alpha}\min_{G\in\mathcal{G}_H}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}_G(s,t)\\&\geq\max_{\alpha}\min_{G\in\mathcal{G}_{\tilde{H}}}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}_G(s,t).\end{aligned}$$

Definition 9 implies that if storage is not a concern (thus the operation of $\max_\alpha$), then for the same repair bandwidth β a bandwidth-optimal helper selection scheme H can protect the largest file size among all possible helper selection schemes $\tilde{H}$.

We are also interested in the bandwidth-storage tradeoff specified in (7) of a given helper selection scheme H. For example, the minimum bandwidth regenerating (MBR) and minimum storage regenerating (MSR) points of a given helper selection scheme H can be defined by

Definition 10: For any given $(n, k, d, \mathcal{M})$ values, the MBR point $(\alpha_{\text{MBR}}, \beta_{\text{MBR}})$ of a helper scheme H is defined by

$$\beta_{\text{MBR}} \triangleq \min\{\beta : (\alpha,\beta) \text{ satisfies (7) and } \alpha = \infty\} \tag{10}$$

$$\alpha_{\text{MBR}} \triangleq \min\{\alpha : (\alpha,\beta) \text{ satisfies (7) and } \beta = \beta_{\text{MBR}}\}. \tag{11}$$

Definition 11: For any given $(n, k, d, \mathcal{M})$ values, the MSR point $(\alpha_{\text{MSR}}, \beta_{\text{MSR}})$ of a helper scheme H is defined by

$$\alpha_{\text{MSR}} \triangleq \min\{\alpha : (\alpha,\beta) \text{ satisfies (7) and } \beta = \infty\} \tag{12}$$

$$\beta_{\text{MSR}} \triangleq \min\{\beta : (\alpha,\beta) \text{ satisfies (7) and } \alpha = \alpha_{\text{MSR}}\}.$$

Specifically, the MBR and MSR are the two extreme ends[10] of the bandwidth-storage tradeoff curve in (7). Both the MBR and MSR points of a given $H \in \mathcal{DHS}$ will be discussed in our main results.

IV. Preview of the Results

In the following, we give a brief preview of our results through concrete examples to illustrate the main contributions of this work. Although we only present here results of special (n, k, d) values as a preview, the main results in Section V are for general (n, k, d) values.

Result 1: The parameter $(n, k, d) = (6, 3, 4)$ is ITHS. That is, BHS is optimal and there is no need to design any intelligent helper selection method.

Result 2: The parameter $(n, k, d) = (6, 4, 4)$ is not ITHS. That is, BHS is strictly sub-optimal. Specifically, we have designed a helper selection scheme in Section VI, termed family helper selection, which is provably optimal.[11] Also see

[10]An alternative definition of the MSR point is as follows. A scheme achieves the MSR point if each node stores only $\alpha = \frac{\mathcal{M}}{k}$ packets, which is different from the definition we used in (12). For example, when $(n, k, d) = (5, 3, 2)$, one can prove that $\min_{H\in\mathcal{DHS}} \alpha_{\text{MSR}} = \frac{\mathcal{M}}{2}$ based on the definition in (12). We thus say that the MSR point of the best possible scheme is $\alpha^*_{\text{MSR}} = \frac{\mathcal{M}}{2}$ for $(n, k, d) = (5, 3, 2)$. In contrast, the alternative MSR definition will say that the MSR point does not exist for the parameter $(n, k, d) = (5, 3, 2)$ since no scheme can achieve

$$\alpha = \frac{\mathcal{M}}{k} = \frac{\mathcal{M}}{3} < \alpha^*_{\text{MSR}} = \frac{\mathcal{M}}{2}.$$

[11]For example, the LRCs in [13], [14], [17] for $(n, k, d) = (6, 4, 4)$ are no better than the curve of the optimal scheme in Fig. 1.

Definition 8. In Fig. 1, the storage-bandwidth tradeoff curve of the optimal scheme is plotted against the BHS scheme with file size $\mathcal{M} = 1$. For this $(n, k, d) = (6, 4, 4)$, the curve of the optimal scheme is not only derived by the min-cut analysis but also complemented by explicit code construction[12] reported separately in [3].

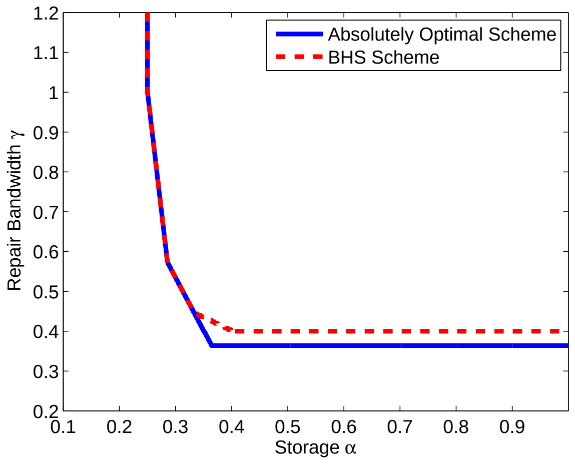


Fig. 1. The storage-bandwidth tradeoff curves of BHS versus the optimal helper selection scheme for $(n, k, d) = (6, 4, 4)$ and file size $\mathcal{M} = 1$.

Result 3: The parameter $(n, k, d) = (5, 3, 2)$ is not ITHS and BHS is strictly sub-optimal. For this (n, k, d) value, our family helper selection scheme is again provably optimal. In Fig. 2 the optimal curve is derived by both a min-cut analysis and explicit code construction [3]. An interesting phenomenon is that the optimal tradeoff curve has only one corner point $(\alpha, \gamma) = (0.5, 0.5)$ and we can achieve this point by the exact-repair scheme in [3]. Note that since the optimal tradeoff curve has only a single corner point, it means that the family helper selection scheme achieves *simultaneously* the smallest storage (the MSR point) and the smallest bandwidth (the MBR point) that any scheme can possibly achieve for this (n, k, d).

Result 4: The parameter $(n, k, d) = (20, 10, 10)$ is not ITHS and BHS is strictly sub-optimal. Our achievability scheme, family helper selection, shows a clear improvement over BHS in Fig. 3. Unfortunately, we do not know what is the optimal helper selection scheme for this (n, k, d) value.

Result 5: A byproduct of our achievability results is the new family helper selection scheme that can be applied to arbitrary (n, k, d) values. The min-cut analysis of family helper selection is provided in this work and the companying exact-repair code construction for the MBR point is reported in [3]. Numerically, the family helper selection demonstrates good performance in all (n, k, d) cases. Analytically, we prove that it is *bandwidth-optimal* (see Definition 9) for all (n, k, d) values satisfying (i) $n \neq 5$, $k = n - 1$, and $d = 2$; (ii) n is even, $k = n - 1$, and $d = 3$; (iii) $n \notin \{7, 9\}$, $k = n - 1$, and $d = 4$; (iv) n is even, $n \notin \{8, 14\}$, $k = n - 1$, and $d = 5$; and (v) $n \notin \{10, 11, 13\}$, $k = n - 1$, and $d = 6$; and many other cases.

[12]There are 3 corner points on the family helper selection scheme curve and they are $(\alpha, \gamma) = (0.25, 1)$, $(\frac{2}{7}, \frac{4}{7})$, and $(\frac{4}{11}, \frac{4}{11})$. The two corners $(\alpha, \gamma) = (0.25, 1)$ and $(\frac{2}{7}, \frac{4}{7})$ can be achieved by the scheme in [32] and the new corner point $(\alpha, \gamma) = (\frac{4}{11}, \frac{4}{11})$ is achieved by the exact-repair code in [3].

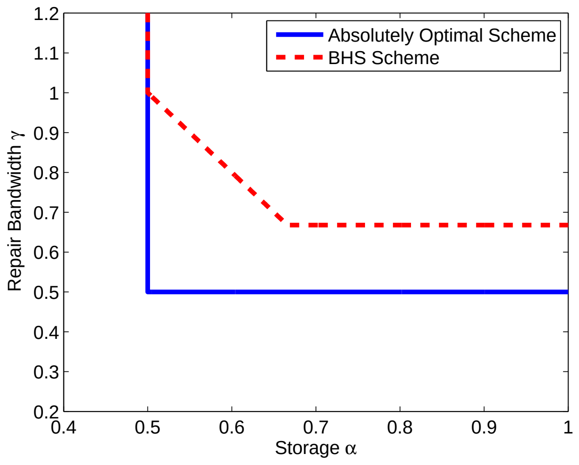


Fig. 2. The storage-bandwidth tradeoff curves of BHS versus the optimal scheme for $(n, k, d) = (5, 3, 2)$ and file size $\mathcal{M} = 1$.

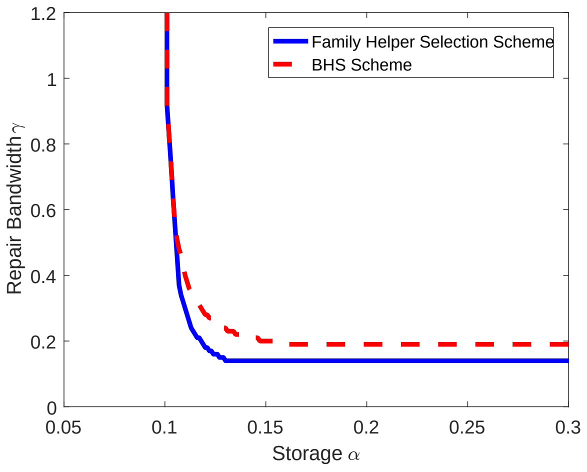


Fig. 3. The storage-bandwidth tradeoff curves of BHS versus the family helper selection scheme for $(n, k, d) = (20, 10, 10)$ and file size $\mathcal{M} = 1$.

V. Main Results

Proposition 1 (The converse): An (n, k, d) is *indifferent-to-helper selection* if at least one of the following two conditions holds:

$$\text{Condition 1: } d = 1, k = 3, \text{ and } n \text{ is odd;} \tag{13}$$

$$\text{or Condition 2: } k \leq \left\lceil \frac{n}{n-d} \right\rceil. \tag{14}$$

Proposition 2 (The achievability): An (n, k, d) is not *indifferent-to-helper selection* if neither (13) nor (14) holds.

Proposition 1 is proved by a new min-cut based analysis, which is relegated to Appendix C. Proposition 2 is proved by analyzing a new scheme, termed the family helper selection, and its extension. The description of the family helper selection scheme is provided in Section VI and the corresponding graph-based analysis is relegated to Appendix D. The proofs of the converse and the achievability parts can be read separately since their analysis are based on different techniques.

Corollary 2: Let H denote any given helper selection scheme for any arbitrarily given $(n, k, d, \alpha, \beta, \mathcal{M})$. Let k^*_H denote the smallest value satisfying that any k^*_H nodes can be used to reconstruct the file. If the (n, k, d) value is ITHS, then we must have $k^*_H \leq d$.

Proof: Proposition 2 shows that any (n, k, d) that is ITHS must satisfy either Conditions 1 or 2. Suppose $d = 1$. Then, by the proof of Proposition 1 in Appendix C, we can show that $k^* = 1$. Therefore, whenever Condition 1 holds, we always have $k^*_H = 1 \leq d = 1$. If Condition 2 holds and $d = 1$, then by the same argument we again have $k^*_H \leq d$.

Suppose Condition 2 holds and $d > 1$. Since we always have $k^* \leq k$ and $k \leq n-1$, we then have $k^* \leq k \leq \min(n-1, \left\lceil \frac{n}{n-d} \right\rceil)$. By some simple algebraic arguments that are provided in Appendix E-A2, we always have $d \geq \min(n-1, \left\lceil \frac{n}{n-d} \right\rceil)$ if $d > 1$. Jointly, we thus have $k^* \leq d$. The proof is complete. ■

Corollary 3: An (n, k, d) is not *indifferent-to-helper selection* if $1 < d < k$.

Proof: Assume $1 < d < k$. Since $1 < d$, Condition 1 does not hold. Since $k \leq n-1$ by (1), we then have $1 < d < n-1$. By some simple algebraic arguments that are provided in Appendix E-A2, $1 < d < n-1$ implies $d \geq \left\lceil \frac{n}{n-d} \right\rceil$. Thus, $1 < d < k$ implies that Condition 2 does not hold. By Proposition 2, the proof is complete. ■

Corollary 3 shows that all the (n, k, d) values in the design scope of LRCs (assuming $d > 1$) are not ITHS.

Proposition 3: For any (n, k, d) values satisfying simultaneously the following three conditions:

$$d \text{ is even,} \tag{15}$$

$$n = d + 2, \tag{16}$$

$$\text{and } k = \frac{n}{2} + 1, \tag{17}$$

the family helper selection scheme is optimal.

Proposition 4: For any (n, k, d) values satisfying simultaneously the following two conditions

$$k = n - 1, \tag{18}$$

$$\text{and } \exists \text{ integers } n_1, n_2, \cdots, n_B, \text{ such that } \forall b = 1, \cdots, B,$$
$$n_b > d,\ n_b \bmod (n_b - d) = 0, \text{ and } n = \sum_{i=1}^{B} n_i, \tag{19}$$

the extension of the family helper selection scheme, described in Section VI-B, is bandwidth optimal.

The implication of Propositions 3 and 4 is significant, as it proves that two very specific helper selection schemes, the family scheme and its extension the family-plus scheme (see Section VI-B), are optimal and bandwidth-optimal under certain (n, k, d) values. The optimality is established among all dynamic helper selection schemes (see Definition 2), which is the most general form of helper selection schemes one can possibly design. For example, for the case of $(n, k, d) = (19, 18, 4)$ and $\alpha = d\beta$, the family-plus scheme is bandwidth optimal since we can divide the 19 nodes into $B = 3$ groups of $(n_1, n_2, n_3) = (8, 6, 5)$.

The proofs of Propositions 3 and 4 are relegated to Appendices E and F, respectively. Some corollaries of Propositions 3 and 4 are provided in Appendix K, which show that the bounds in [11] are tight in some cases but loose in other cases.

Propositions 1 to 4 can be used to derive the results in Section IV. Specifically, Result 1 is derived by Proposition 1. Result 2 is a combination of Propositions 2 and 3. Result 3 follows from Proposition 2 and the simple observation that for $(n, k, d) = (5, 3, 2)$ the tradeoff curve of any helper selection scheme is no better than

$$2 \min(2\alpha, \beta) \geq \mathcal{M},$$

which can be achieved by the proposed family helper selection scheme.

Result 4 follows from Proposition 2 and the explicit expression of the storage-bandwidth tradeoff of the family helper selection scheme that is provided in Proposition 5 in Appendix D-B. Result 5 follows from Proposition 4 directly.

The description of the family helper selection scheme, the DSN code used in our achievability proof, is provided in the next section, while the corresponding analysis and all the other proofs are relegated to the appendices.

VI. New Achievability Schemes

We propose a pair of new helper selection schemes, termed the *family helper selection scheme* and the *family-plus helper selection scheme*, respectively. The designs of these two schemes are to realize the potential gain of intelligent helper selection in an idealistic setting and thus may not consider fully the practical issues like complexity, reliability, and flexibility. On the other hand, the proposed schemes have very simple structure and, as will be proven in Appendix D, are capable of realizing a substantial fraction of the gain of intelligent helper selection. For example, for any (n, k, d) that is not indifferent-to-helper-selection, the proposed schemes always strictly outperform the BHS scheme.

A. Family Helper Selection Schemes

To describe the family helper selection scheme, we first arbitrarily sort all storage nodes and denote them by 1 to n. We then define a *complete family* as a group of $(n-d)$ physical nodes. The first $(n-d)$ nodes are grouped as the first complete family and the second $(n-d)$ nodes are grouped as the second complete family and so on. In total, there are $\left\lfloor \frac{n}{n-d} \right\rfloor$ complete families. The remaining $n \bmod (n-d)$ nodes are grouped as an *incomplete family*.

The family helper selection scheme belongs to the general class of stationary helper selection in Definition 4. That is, the corresponding helper selection rule $D_\tau(\cdot)$ is of the form of (3) and can be described by a function $f(F_\tau)$ that takes the index of the failed node F_τ as input and returns a subset of d nodes in the available nodes $\mathcal{A}_\tau$. If the failed node $F_\tau = i$ is in a complete family, then we let $f(i)$ return "all the nodes *not* in the same family of node i." That is, a newcomer only seeks

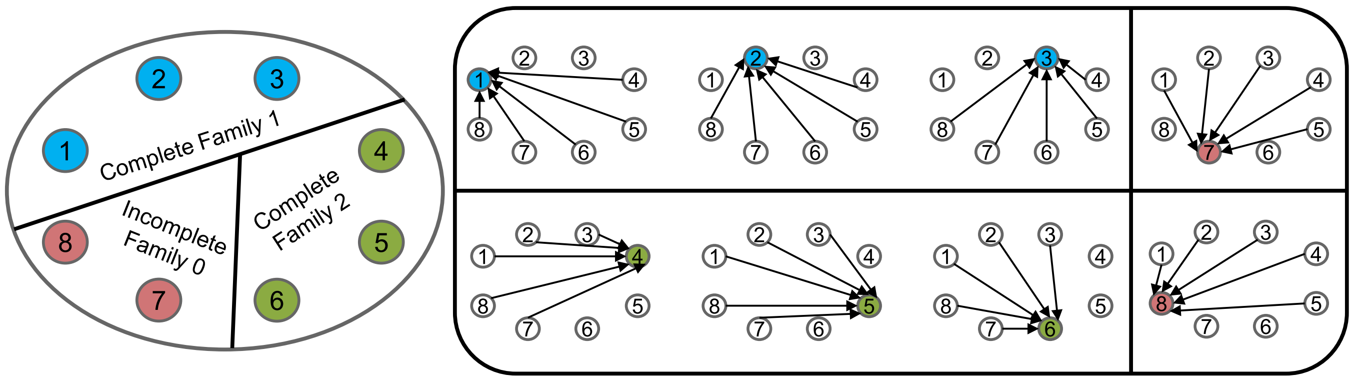


Fig. 4. The family helper selection scheme for $(n,d)=(8,5)$ and the illustration of the repair process of each of the 8 nodes.

help from *outside* its family. The intuition is that we would like each family to preserve as much information (or equivalently as diverse information) as possible. To that end, we design the helper selection sets such that each newcomer refrains from requesting help from its own family. If the failed node $F_\tau = i$ is in the incomplete family,[13] then we let $f(i)$ return the first d nodes, i.e., $f(i)=\{1,\cdots,d\}$. The description of the family helper selection scheme is complete.

After the family helper selection chooses the d helpers, each helper will send β packets to the newcomer and repair its storage content, which is of α packets. The actual code construction (how to choose the β packets and how to repair the α packets) is not part of the helper selection scheme, and we simply assume the best possible code construction is used.[14] For interested readers, see [3] for the explicit code design after selecting the helpers.

For example, suppose that $(n,d)=(8,5)$. There are 2 complete families, $\{1,2,3\}$ and $\{4,5,6\}$, and 1 incomplete family, $\{7,8\}$. See Fig. 4 for illustration. The family helper selection scheme for this example is illustrated in Fig. 4. Let us say node 4 fails. The corresponding newcomer will access nodes $\{1,2,3,7,8\}$ for repair since nodes 1, 2, 3, 7, and 8 are outside the family of node 4. If node 7 (a member of the incomplete family) fails, then the newcomer will access nodes 1 to 5 for repair.

B. The Family-plus Helper Selection Scheme

In the family helper selection scheme, there are $\left\lfloor \frac{n}{n-d} \right\rfloor$ complete families and 1 incomplete family (if $n \bmod (n-d) \neq 0$). For the scenario in which the n and d values are comparable, we have many complete families. By partitioning the nodes into many families and by imposing helper selection rules based on which family a node belongs to, the family helper selection scheme is capable of harvesting a significant fraction of the benefits of choosing good helpers. However, when n is large but d is small, we only have one complete family and one incomplete family. With only two families, the benefit of family helper selection is not as great. In this subsection, we propose the *family-plus helper selection* scheme that targets this scenario, i.e., when n is large but d is small.

The main idea is as follows. We first partition the n nodes into B disjoint groups. Each group has n_b nodes, where $b=1,\cdots,B$. We require that $n_b \geq (d+1)$ for each $b=1,\cdots,B$. After partitioning, we apply the family helper selection scheme to the individual groups. The description of family-plus helper selection is complete. Note that unlike the family helper selection scheme, which is completely determined for any given (n,k,d) value, the family-plus helper selection scheme is parameterized by the parameters $n_1,n_2,\cdots,n_B$ that are being used. Different choices of $n_1,n_2,\cdots,n_B$ will lead to different performance of the family-plus helper selection scheme. On the other hand, having the new parameters $n_1,n_2,\cdots,n_B$ also means that family-plus helper selection is strictly more general than the family helper selection scheme and includes the latter as a special case, since we can always choose $B=1$ and $n_1=n$.

For example, suppose $d=2$ and $n=9$. We can choose $B=3$ and $n_1=n_2=n_3=3$, or we can choose $B=2$, $n_1=4$, and $n_2=5$. Suppose we choose the latter. Then the first group contains nodes $\{1,2,3,4\}$ and the second group contains nodes $\{5,6,7,8,9\}$. We then apply the family helper selection scheme to the individual groups. Since group 1 has four nodes and since $d=2$, nodes $\{1,2\}$ will be labeled as the first complete family and nodes $\{3,4\}$ will be the second complete family. If node 2 fails, then it will choose nodes $\{3,4\}$ as its helpers. Similarly, if node 3 fails, it will choose nodes $\{1,2\}$ as its helpers.

In the second group, nodes $\{5,6,7\}$ will form a complete family and nodes $\{8,9\}$ will form an incomplete family. If node 6 fails, it will request help from both nodes 8 and 9. If node 9 fails, it will request help from nodes $\{5,6\}$, the first $d=2$ nodes of this group. All the repair operations for nodes 5 to 9 are completely separated from the operations of nodes 1 to 4.

One can easily see that when n is large and d is small, we can choose a large B and let each n_b be comparable to d. The

[13]All the concepts and intuition are based on complete families. The incomplete family is used to make the scheme consistent and applicable to the case when $n \bmod (n-d) \neq 0$.

[14]One way to envision the code construction is to assume that the coding vectors are chosen continuously randomly from the Euclidean space rather than a finite field. This way, one can circumvent the technicality of the size of the finite field versus the infinite number of instances in the information flow graphs $\mathcal{G}$.

family-plus helper selection scheme will then divide the nodes into B groups and there will be many families within each group since n_b is comparable to d. This is in contrast with the family helper selection scheme, which has only one complete and one incomplete family when $n \gg d$. By dividing the nodes into smaller groups and by creating many families within each group, family-plus helper selection scheme will have better performance than the family helper selection scheme when $n \gg d$.

C. Performance of the Proposed Schemes

Both proposed schemes are easy to describe and implement. However, their performance characterization is non-trivial and requires a substantial amount of new notation to describe the results. Specifically, one has to carefully find the largest file size $\mathcal{M}$ that can still satisfy the min-cut condition (7) under the family and family-plus helper selection schemes, assuming a certain pair of (α, β) is used. In order not to disrupt the flow of this work, the performance analysis of both schemes are relegated to the appendices. The storage-bandwidth tradeoff curve and the MBR point of the family helper selection scheme are provided in Appendix D-B. The storage-bandwidth tradeoff curve of the family-plus helper selection scheme is provided in Appendix D-C.

VII. Conclusion

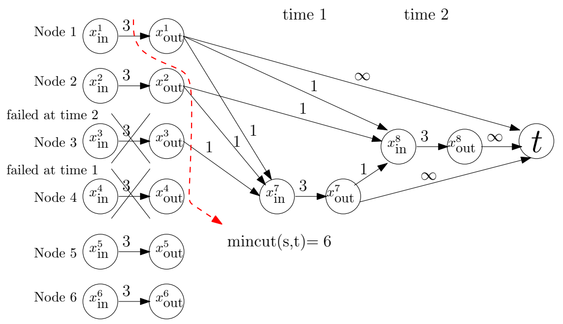
In practice, it is intuitive that the newcomer should access only those "good" helpers. However, this paper has shown that for some (n, k, d) values, even the most intelligent helper selection scheme is no better than the simplest blind helper selection scheme. We term those (n, k, d) values as being *indifferent-to-helper-selection* (ITHS). This paper has provided a necessary and sufficient condition that characterizes the set of ITHS (n, k, d) values. A byproduct of our analysis is a new class of low-complexity solutions termed the *family helper selection scheme*, which is provably optimal or bandwidth-optimal for a variety of (n, k, d) values.

Appendices

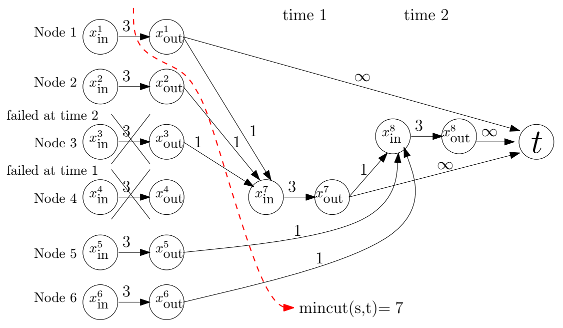
In our appendices, those sections that are provided only for the purpose of completeness will start with "Supplemental:". For readers who are familiar with the material and/or who prefer not to disrupt the logic flow can simply skip those appendices.

Appendix A
Supplemental: An Example Illustrating Choosing the "Good" Helpers

Fig. 5 shows an example illustrating choosing the "good" helpers and how it protects a file of larger size. The parameters of the storage network in this figure are $(n, k, d, \alpha, \beta) = (6, 3, 3, 3, 1)$. The goal of this example is to store a data object of size $\mathcal{M} = 7$ such that the network can tolerate $n - k = 3$ failures. Without loss of generality, we assume that node 4 fails in time 1 and the helpers of the newcomer (replacing node 4) are nodes 1, 2, and 3. Now assume that node 3 fails in time 2. We will demonstrate in the following how the helper choice at time 2 (for replacing node 3) will substantially affect the reliability of the DSN.



(a) Choosing the helper nodes arbitrarily.



(b) Choosing the "good" helper nodes.

Fig. 5. An example illustrating choosing the "good" helper nodes for $(n, k, d, \alpha, \beta) = (6, 3, 3, 3, 1)$ and file size $\mathcal{M} = 7$.

Choice 1: Suppose the helpers of node 3 in time 2 are nodes 1, 2, and 4. See Fig. 5(a). Now, we consider the data collector t which would like to reconstruct the original file of size 7 from nodes 1, 3, and 4. By noticing that one of the edge cuts from the virtual source to the data collector has value 6 (see the red dashed curve in Fig. 5(a)), it is thus impossible for the data collector to reconstruct the original file when the helper selection is made blindly. One can actually prove that the largest file size that can be protected by BHS is indeed $\mathcal{M}^*_{\text{BHS}} = 6$.

Choice 2: Suppose the helpers of node 3 in time 2 are nodes 4, 5, and 6. See Fig. 5(b). Now we consider the same data collector t that accesses nodes 1, 3, and 4. One can verify that the min-cut value from source s to the data collector t is 7, which is equal to the target file size 7. Furthermore, one can check the rest $\binom{6}{3} - 1 = 19$ different ways of setting up the data collectors and they all have $\text{mincut}(s, t) \geq 7$. The above observation illustrates that helper selection choice (Choice 2) can strictly improve the min-cut value of the network.

The choice of the helpers in this example follows the family helper selection scheme described in Section VI. In Appendix D-B, it is proven that not only we can improve the min-cut value in the end of the first 2 time slots, but the min-cut-value is always ≥ 7 even after arbitrarily many failure/repair stages with intelligent helper selection for each time slot. The protected file size is $\mathcal{M} = 7 > \mathcal{M}^*_{\text{BHS}} = 6$.

Appendix B
Supplemental: The Information Flow Graph

We provide in this appendix the description of the information flow graph (IFG) that was first introduced in [7].

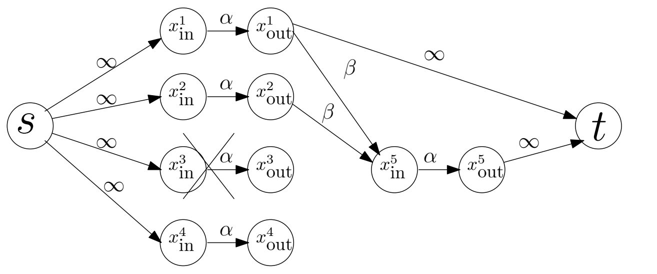


Fig. 6. An example of the information flow graph with $(n, k, d) = (4, 2, 2)$.

As shown in Fig 6, an IFG has three different kinds of nodes. It has a single *source* node s that represents the source of the data object. It also has nodes x^i_{in} and x^i_{out} that represent storage node i of the IFG. A storage node is split into two nodes so that the IFG can represent the storage capacity of the nodes. We often refer to the pair of nodes x^i_{in} and x^i_{out} simply by storage node i. In addition to those nodes, the IFG has *data collector* (DC) nodes. Each data collector node is connected to a set of k active storage nodes, which represents the party that is interested in extracting the original data object initially produced by the source s. Fig. 6 illustrates one such data collector, denoted by t, which connects to $k = 2$ storage nodes. A more detailed description of the IFG is provided as follows.

The IFG evolves with time. In the first stage of an information flow graph, the source node s communicates the data object to all the initial nodes of the storage network. We represent this communication by edges of infinite capacity as this stage of the IFG is virtual. See Fig. 6 for illustration. This stage models the encoding of the data object over the storage network. To represent storage capacity, an edge of capacity α connects the input node of a storage node to the corresponding output node. When a node fails in the storage network, we represent that by a new stage in the IFG where, as shown in Fig. 6, the newcomer connects to its helpers by edges of capacity β resembling the amount of data communicated from each helper. The set of d helpers is chosen arbitrarily from the candidate helper sets specified by the helper selection rule $D_\tau(\cdot)$ described in Definition 2. Namely, if $D_\tau(\cdot)$ returns I different sets, each containing d nodes, then there are I different ways to choose the helper set. The IFG can arbitrarily choose one helper set and the newcomer will access those helpers with edges of capacity β.

We note that although the failed node still exists in the IFG, it cannot participate in helping future newcomers. Accordingly, we refer to failed nodes by *inactive* nodes and existing nodes by *active* nodes. Namely, there are always n active nodes in an IFG but there are τ inactive nodes, where τ specifies the time instant, or, equivalently, how many nodes have failed. By the nature of the repair problem, the IFG is always acyclic.

Given an IFG G, we use $\mathrm{DC}(G)$ to denote the collection of all $\binom{n}{k}$ *data collector nodes* in G [7]. Each data collector $t \in \mathrm{DC}(G)$ represents one unique way of choosing k out of n active nodes when reconstructing the file.

Appendix C
Proof of Proposition 1 — The Converse Direction

The proof of Proposition 1, the converse direction, consists of two different parts. Part I: Suppose (14) holds. In this case we prove that regardless how we design the helper selection rules, there exists a "harmful" sub-structure of k active nodes in the IFG, provided every node has been repaired at least once. The harmful sub-structure has the property that if we let the data collector $t \in \mathrm{DC}(G)$ connect to those k nodes, then the "min-cut value" of the given helper selection is identical to the min-cut value of BHS. Therefore, the (n, k, d) value is ITHS.

Part II: Suppose (13) holds. In this case, we do not always have a harmful sub-structure even if every node has been repaired at least once. Instead, we will prove that for any given helper selection scheme, there exists a very specific sequence of node failures such that after the failure sequence, there exists a harmful sub-structure with $k = 3$ active nodes. Then, we let the data collector $t \in \mathrm{DC}(G)$ connect to those k nodes and the min-cut value of the given helper selection is again identical to the min-cut value of BHS. Therefore, the (n, k, d) value is ITHS.

The difference between Parts I and II is that in Part I every failure sequence will lead to the existence of a k-node harmful sub-structure as long as each node has been repaired at least once. In contrast, in the proof of Part II, only some specific failure sequence will lead to a harmful sub-structure.

A. Part I of the Proof

Before proving Part I, we introduce the following definition and lemma.

Definition 12: A set of m active storage nodes (input-output pairs) of an IFG is called an m-set if the following conditions are satisfied simultaneously. (i) Each of the m active nodes has been repaired at least once; and (ii) Jointly the m nodes satisfy the following property: Consider any two distinct active nodes x and y in the m-set and, without loss of generality, assume that x was repaired before y. Then there exists an edge in the IFG that connects x_{out} and y_{in}.

The best way to interpret the m-set is to view it as a generalized version of *the m-clique* in the context of IFGs. As will be proven later, the m-set turns out to be a "harmful sub-structure" in an IFG. We then have the following lemma.

Lemma 2: Fix any arbitrary helper selection scheme H. Consider an arbitrary $G \in \mathcal{G}_H(n, k, d, \alpha, \beta)$ such that each active node in G has been repaired at least once. Then, there exists a $\left\lceil \frac{n}{n-d} \right\rceil$-set in G.

Proof: We prove this lemma by proving the following stronger claim: Consider any integer value $m \geq 1$. There exists an m-set in every group of $(m-1)(n-d)+1$ active nodes that have been repaired at least once in the past. Since the G we consider has n active nodes and each of them has been repaired at least once, the above claim implies that G must contain a $\left\lceil \frac{n}{n-d} \right\rceil$-set.

We prove this claim by induction on the value of m. When $m = 1$, by the definition of the m-set, any group of 1 active node in G forms a 1-set. The claim thus holds naturally.

Suppose the claim is true for all $m < m_0$, we now claim that in every group of $(m_0 - 1)(n - d) + 1$ active nodes of G there exists an m_0-set. The reason is as follows. Consider an arbitrary, but fixed group of $(m_0 - 1)(n - d) + 1$ active nodes, denoted by S_0. We use y to denote the youngest active node in S_0 (the one which was repaired last). Obviously, there are $(m_0 - 1)(n - d)$ active nodes in S_0 other than y. We then notice that any newcomer can "choose to access" d helpers out of $n - 1$ surviving nodes during its repair. Equivalently, it also implies that any newcomer can "choose to avoid" $(n - 1) - d$ nodes when selecting its helpers since there are exactly $(n-1)$ available nodes during each repair.

Since node y avoided $(n-1)-d$ nodes during its repair and since node y is the youngest of S_0, node y could have avoided *at most*[15] $(n - 1) - d$ nodes out of the other $(m_0 - 1)(n - d)$ active nodes in S_0. As a result, node y must have requested help from at least $(m_0 - 1)(n - d) - (n - 1 - d) = (m_0 - 2)(n - d) + 1$ helpers from the node set S_0. Or equivalently, node y is connected to at least $(m_0-1)(n-d)-(n-1-d) = (m_0-2)(n-d)+1$ nodes in S_0. Among those nodes, choose exactly $(m_0 - 2)(n - d) + 1$ nodes in any arbitrary way and denote the chosen set by S_1. By our construction, we clearly have $(S_1 \cup \{y\}) \subseteq S_0$.

By the induction assumption, among those $(m_0 - 2)(n - d) + 1$ nodes in S_1, there exists an $(m_0 - 1)$-set. Note that by our construction, node y is connected to *all* nodes in S_1. Therefore, node y is connected to all nodes in this $(m_0 - 1)$-set. Node y and this (m_0-1)-set, all part of the original node set S_0, thus jointly form an m_0-set. The proof of this claim is thus complete and hence the proof of Lemma 2. ■

We now establish the relationship between the existence of a k-set and the (n, k, d) being ITHS.

Lemma 3: Consider any fixed (n, k, d). If for all helper selection schemes $H \in \mathcal{DHS}$, there exists a k-set in at least one $G \in \mathcal{G}_H$, then the given (n, k, d) value is indifferent-to-helper-selection.

Proof: For any helper selection scheme $H \in \mathcal{DHS}$, consider the corresponding $G \in \mathcal{G}_H$ that contains a k-set. We then consider a data collector of G that connects to all k nodes of the k-set. Call this data collector t. If we focus on the edge-cut that separates source s and the k node pairs connected to t, one can use the same analysis as in [7, Lemma 2] and derive "$\text{mincut}_G(s,t) \leq \sum_{i=0}^{k-1} \min((d-i)^+\beta, \alpha)$" for the specific choice of $G \in \mathcal{G}_H$ and the specific choice of t. By further taking the minimum over all $t \in \text{DC}(G)$ and all $G \in \mathcal{G}_H$, we have

$$\min_{G\in\mathcal{G}_H} \min_{t\in\text{DC}(G)} \text{mincut}_G(s,t) \leq \sum_{i=0}^{k-1} \min((d-i)^+\beta, \alpha). \tag{20}$$

[15]It is "at most" because node y can choose to avoid some nodes outside S_0 as well.

On the other hand, by Lemma 1 we have $\mathcal{G}_H \subseteq \mathcal{G}_{\text{BHS}}$. Therefore, we also have

$$\begin{aligned}&\min_{G\in\mathcal{G}_H} \min_{t\in\text{DC}(G)} \text{mincut}_G(s,t)\\ &\geq \min_{G\in\mathcal{G}_{\text{BHS}}} \min_{t\in\text{DC}(G)} \text{mincut}_G(s,t).\end{aligned} \tag{21}$$

Then, (20), (21), and (6) jointly imply that equality (8) holds for the given H. Since we considered in the beginning an arbitrary $H \in \mathcal{DHS}$, we have proven that the given (n, k, d) must be indifferent-to-helper-selection. ■

The rest of the proof of Part I is as follows. Consider any given helper selection scheme $H \in \mathcal{DHS}$ and any $G \in \mathcal{G}_H$ such that all n active nodes of G have been repaired at least once. By Lemma 2, there exists a $\left\lceil \frac{n}{n-d} \right\rceil$-set in G. Since in Part I of the proof we assume (14), i.e., $k \leq \left\lceil \frac{n}{n-d} \right\rceil$, there also exists a k-set in $G \in \mathcal{G}_H$. Since we started with arbitrary $H \in \mathcal{DHS}$, Lemma 3 then implies that the given (n, k, d) is indifferent-to-helper-selection. The proof of Part I is complete.

B. Part II of the Proof

In Part II of the proof, we assume (13) is true, i.e., $d = 1$, $k = 3$, and n is odd. In contrast to Part I that focuses on the harmful sub-structure called m-set, Part II focuses on a different harmful sub-structure called the *3-chain*. We then prove that for any given $H \in \mathcal{DHS}$, we can find a sequence of node failures such that the resulting $G \in \mathcal{G}_H$ has a 3-chain, cf. Lemma 2 in Part I. Finally, we prove the relationship between the existence of a 3-chain and the (n, k, d) being ITHS, cf. Lemma 3 in Part I. The proof of Part II is then complete.

Definition 13: A set of 3 active storage nodes (input-output pairs) of an IFG G is called a 3-chain if the following conditions are satisfied simultaneously. Denote the 3 active nodes by x, y, and z. (i) x is repaired before y and y is repaired before z; (ii) $(x_{\text{out}}, y_{\text{in}})$ is an edge in G; and (iii) either $(x_{\text{out}}, z_{\text{in}})$ is an edge in G or $(y_{\text{out}}, z_{\text{in}})$ is an edge in G.

The intuition of this definition is self-explanatory: Either $x \leftarrow y \leftarrow z$ forms a chain in G or $y \rightarrow x \leftarrow z$ forms a chain in G, where $a \leftarrow b$ represents an edge $(a_{\text{out}}, b_{\text{in}})$ and $a \rightarrow b$ represents an edge $(b_{\text{out}}, a_{\text{in}})$.

Lemma 4: Assume (13) and fix any arbitrary helper selection scheme H. There exists a $G^* \in \mathcal{G}_H$ that contains a 3-chain.

Proof: We prove Lemma 4 by a sequential construction. Start from any $G \in \mathcal{G}_H$ with all n nodes having been repaired at least once. We arbitrarily choose one active node in G and denote it by $w^{(1)}$. We let $w^{(1)}$ fail and denote the newcomer that replaces $w^{(1)}$ by $y^{(1)}$. The helper selection scheme H will choose one helper node (since $d = 1$) and we denote that helper node as $x^{(1)}$. The new IFG after this failure and repair process is denoted by $G^{(1)}$. Note that $G^{(1)}$ is based on the initial $G \in \mathcal{G}_H$ with one additional node failure and the corresponding repair. By construction, $x^{(1)}$, an active node, is repaired before the newcomer $y^{(1)}$ and there is an edge $(x^{(1)}_{\text{out}}, y^{(1)}_{\text{in}})$ in $G^{(1)}$.

Starting now from $G^{(1)}$, we choose another $w^{(2)}$ which is not one of $x^{(1)}$ and $y^{(1)}$ and let this node fail. Such $w^{(2)}$ always exists since n is odd, see (13). We use $y^{(2)}$ to denote the newcomer that replaces $w^{(2)}$. The helper selection scheme H will again choose a helper node based on the history of the failure pattern, see Definition 2. We denote the new IFG (after the helper selection made by scheme H) as $G^{(2)}$. If the helper node of $y^{(2)}$ is $x^{(1)}$, then the three nodes $(x^{(1)}, y^{(1)}, y^{(2)})$ are the (x, y, z) nodes satisfying properties (i), (ii) and the first half of (iii) in Definition 13. If the helper node of $y^{(2)}$ is $y^{(1)}$, then the three nodes $(x^{(1)}, y^{(1)}, y^{(2)})$ are the (x, y, z) nodes satisfying properties (i), (ii) and the second half of (iii) in Definition 13. In both cases, we can stop our construction and let $G^* = G^{(2)}$. We say that the sequential construction is complete in the second round.

Suppose neither of the above two is true, i.e., the helper of $y^{(2)}$ is neither $x^{(1)}$ nor $y^{(1)}$. Then, we denote the helper of $y^{(2)}$ by $x^{(2)}$. Note that after this step, $G^{(2)}$ contains two disjoint pairs of active nodes such that there is an edge $(x^{(m)}_{\text{out}}, y^{(m)}_{\text{in}})$ in $G^{(2)}$ for $m = 1, 2$.

We can repeat this process for the third time by failing a node $w^{(3)}$ that is none of $\{x^{(m)}, y^{(m)} : \forall m = 1, 2\}$. We can always find such a node $w^{(3)}$ since n is odd, see (13). Again, let $y^{(3)}$ denote the newcomer that replaces $w^{(3)}$ and the scheme H will choose a helper for $y^{(3)}$. The new IFG after this failure and repair process is denoted by $G^{(3)}$. If the helper of $y^{(3)}$ is $x^{(m)}$ for some $m = 1, 2$, then the three nodes $(x^{(m)}, y^{(m)}, y^{(3)})$ are the (x, y, z) nodes satisfying properties (i), (ii) and the first half of (iii) in Definition 13. If the helper node of $y^{(3)}$ is $y^{(m)}$ for some $m = 1, 2$, then the three nodes $(x^{(m)}, y^{(m)}, y^{(3)})$ are the (x, y, z) nodes satisfying properties (i), (ii) and the second half of (iii) in Definition 13. In both cases, we can stop our construction and let $G^* = G^{(3)}$ and we say that the construction is complete in the third round. If neither of the above two is true, then we denote the helper of $y^{(3)}$ by $x^{(3)}$, and repeat this process for the fourth time and so on.

We now observe that if the construction is not complete in the m_0-th round, then we will have m_0 disjoint active node pairs $\{(x^{(m)}, y^{(m)}) : \forall m = 1, 2, \cdots, m_0\}$ in $G^{(m_0)}$. Since n is odd, we can always start the next round, the $(m_0 + 1)$-th round, by finding a node $w^{(m_0+1)}$ that is none of $\{x^{(m)}, y^{(m)} : \forall m = 1, 2, \cdots, m_0\}$. On the other hand, we cannot repeat this process indefinitely since we only have a finite number of n active nodes in the network. Therefore, the construction must be complete in the $\tilde{m}$-th round for some finite $\tilde{m}$. If the helper of $y^{(\tilde{m})}$ is $x^{(m)}$ for some $m = 1, 2, \cdots, \tilde{m} - 1$, then the three nodes $(x^{(m)}, y^{(m)}, y^{(\tilde{m})})$ are the (x, y, z) nodes satisfying properties (i), (ii) and the first half of (iii) in Definition 13. If the helper node of $y^{(\tilde{m})}$ is $y^{(m)}$ for some $m = 1, 2, \cdots, \tilde{m}-1$, then the three nodes $(x^{(m)}, y^{(m)}, y^{(\tilde{m})})$ are the (x, y, z) nodes satisfying properties (i), (ii) and the second half of (iii) in Definition 13. Let $G^* = G^{(\tilde{m})}$ denote the final IFG. The three nodes $(x^{(m)}, y^{(m)}, y^{(\tilde{m})})$ form a 3-chain in the final graph G^*. The proof is thus complete. ■

The rest of the proof of Part II is as follows. By Lemma 4, for any (n, k, d) satisfying (13) and any $H \in \mathcal{DHS}$, we can find a $G^* \in \mathcal{G}_H$ that contains a 3-chain, denoted by $\{x, y, z\}$. Fix such G^*, and we then let $t^* \in \mathrm{DC}(G^*)$ denote the data collector that is connected to $\{x, y, z\}$. By properties (i) to (iii) in Definition 13 we can see that node x is a vertex-cut separating source s and the data collector t^*. The min-cut value separating s and t^* thus satisfies $\mathrm{mincut}_{G^*}(s, t^*) \leq \min(d\beta, \alpha) = \sum_{i=0}^{k-1} \min((d-i)^+\beta, \alpha)$, where the inequality follows from x being a vertex-cut separating s and t^* and the equality follows from (13) (which implies $d = 1$ and $k = 3$). By taking the minimum over all $t \in \mathrm{DC}(G^*)$ and all $G \in \mathcal{G}_H$

$$\min_{G\in\mathcal{G}_H} \min_{t\in\mathrm{DC}(G)} \mathrm{mincut}_G(s, t) \leq \sum_{i=0}^{k-1} \min((d - i)^+\beta, \alpha). \tag{22}$$

Finally, by the same arguments as used in the proof of Lemma 3, we thus have that inequality (22) must be an equality. We notice that the right-hand side of (22) is identical to (6). Since we considered in the beginning an arbitrary $H \in \mathcal{DHS}$, we have proven that the given (n, k, d) must be indifferent-to-helper-selection. The proof of Part II is complete.

Appendix D
Proof of Proposition 2

The proof of the achievability direction, Proposition 2, is based exclusively on analyzing the family helper selection and family-plus helper selection schemes described in Section VI. Specifically, we first find the closed-form expression of the performance of the described two schemes and then use it to show that when the (n, k, d) value satisfies neither (13) nor (14), the performance of the above two schemes is strictly better than that of BHS.

In this appendix, we first introduce in Appendix D-A new notation that will be needed when stating the closed-form expression of the performance of the two schemes. We then summarize the performance of the two schemes in Appendices D-B and D-C by stating several new propositions. Appendix D-D discusses how to use the proposed schemes to prove the achievability direction.

The detailed proofs of the new propositions are provided in separate appendices, Appendix G and Appendix J.

A. New Notations

We first focus on the family helper selection scheme. The description of the helper selection scheme is quite simple, i.e., dividing the nodes into families of $(n - d)$ nodes and each newcomer requests help from *outside* its own family. On the other hand, the corresponding analysis is highly-nontrivial since we are interested in arbitrary (n, k, d) values, not just a subset of special (n, k, d). Therefore, all the combinations of (n, k, d) need to be carefully analyzed, including many corner cases. To describe its performance with full generality, we need the following new notation.

1) The Family Index Vector: By the description of the family helper selection scheme, we have in total $\left\lceil \frac{n}{n-d} \right\rceil$ number of families, which are indexed from 1 to $\left\lceil \frac{n}{n-d} \right\rceil$. However, since the incomplete family has different properties from the complete families, we replace the index of the incomplete family with 0. Therefore, the family indices become from 1 to $c_{\text{last}} \triangleq \left\lfloor \frac{n}{n-d} \right\rfloor$ and then 0, where c_{last} is the index of the last

Complete family. If there is no incomplete family, we simply omit the index 0.

For any newcomer i, we let D_i denote the helper set for newcomer i. Namely, if node i belongs to a complete family, then D_i contains all nodes that are *not* in the same family as i. If node i belongs to the incomplete family, then the family helper selection scheme will set $D_i = \{1, \cdots, d\}$, the first d nodes. However, if we take a close look, then node i of the incomplete family will request help from *all* the members of the first $(c_{\text{last}} - 1)$ complete families, *but only from* the first $d - (n-d)(c_{\text{last}} - 1) = n \bmod (n-d)$ members of the last complete family. Among the $(n-d)$ members in the last complete family, we thus need to distinguish those members who will be helpers for incomplete family members, and those who will not. Therefore, *we add a negative sign to the family indices of those who will "not" be helpers for the incomplete family.*

From the above discussion, we can now list the family indices of the n nodes as an n-dimensional *family index vector*. To illustrate our definition, consider the case of $(n,d) = (8,5)$ as an example. Each complete family contains $n-d = 3$ nodes. There are two complete families, nodes 1 to 3 and nodes 4 to 6. Nodes 1 to 3 have family indices being 1 and nodes 4 to 6 have family indices being 2. Nodes 7 and 8 belong to the incomplete family and thus have family index 0. However, the third member of the second complete family, node 6, is not a helper for the incomplete family members, nodes 7 and 8, since $D_7 = D_8 = \{1, \cdots, d\} = \{1, 2, \cdots, 5\}$. Therefore, we replace the family index of node 6 by -2. In sum, the *family index vector* of this $(n,d) = (8,5)$ example becomes $(1,1,1,2,2,-2,0,0)$. Mathematically, we can write the family index vector as

$$\Bigg(\overbrace{1, \cdots, 1}^{n-d}, \overbrace{2, \cdots, 2}^{n-d}, \cdots, \overbrace{c_{\text{last}}, \cdots, c_{\text{last}}}^{d-(c_{\text{last}}-1)(n-d)}, \overbrace{-c_{\text{last}}, \cdots, -c_{\text{last}}}^{c_{\text{last}}\cdot(n-d)-d}, \overbrace{0, \cdots, 0}^{n \bmod (n-d)}\Bigg). \quad (23)$$

The operation and implementation of the family helper selection scheme is not based on the above definition of the family index vector. However, the family index vector is very useful when describing the performance of the family helper selection scheme. For example, if a node i has family index being 0, then our definition automatically implies that it belongs to the incomplete family and its helpers are those nodes of family index being strictly positive. Similarly, if a node i has a non-zero family index, say x, then it must belong to a complete family and its helpers are those nodes of family index not equal to x nor $-x$.

2) The Family Index Permutation π_f and the function $y_i(\pi_f)$: A *family index permutation* is a permutation of the family index vector defined in (23), which we denote by π_f. Continuing from the previous example, one instance of family index permutations is $\pi_f = (1,1,0,2,0,-2,1,2)$.

For any given family index permutation π_f, we define $[\pi_f]_i$ as the i-th coordinate value of the vector π_f. For example, if $\pi_f = (1,1,0,2,0,-2,1,2)$, then $[\pi_f]_4 = 2$ since the fourth coordinate of π_f is 2. For any given i and π_f, we define the following value $y_i(\pi_f)$ as follows. If $[\pi_f]_i = 0$, then $y_i(\pi_f)$ returns the number of j satisfying both (i) $j < i$ and (ii) $[\pi_f]_j > 0$; If $[\pi_f]_i \neq 0$, then $y_i(\pi_f)$ returns the number of j satisfying both (i) $j < i$ and (ii) $|[\pi_f]_j| \neq |[\pi_f]_i|$. I.e., those js for which the absolute value of the j-th coordinate and the i-th coordinate of π_f are different.

Although the description of $y_i(\pi_f)$ is complicated, the intuition is simple. Namely, $y_i(\pi_f)$ returns the number of "nodes" in the first $(i-1)$ coordinates of π_f that can be a helper of node $[\pi_f]_i$. Also, see the discussion in the end of Appendix D-A1. For example, if $\pi_f = (1,2,-2,1,0,0,1,2)$, then $(y_1(\pi_f), y_2(\pi_f), \cdots, y_8(\pi_f)) = (0,1,1,2,3,3,4,5)$.

3) Rotating Family Index Permutation: A *rotating family index permutation (RFIP)* π_f^* is a special family index permutation that puts the family indices of (23) in an $(n-d) \times \left\lceil \frac{n}{n-d} \right\rceil$ table column-by-column and then reads it row-by-row. Fig. 7 illustrates the construction of the RFIP for $(n,d) = (8,5)$. The input is the family index vector $(1,1,1,2,2,-2,0,0)$ and the output is the RFIP $\pi_f^* = (1,2,0,1,2,0,1,-2)$.

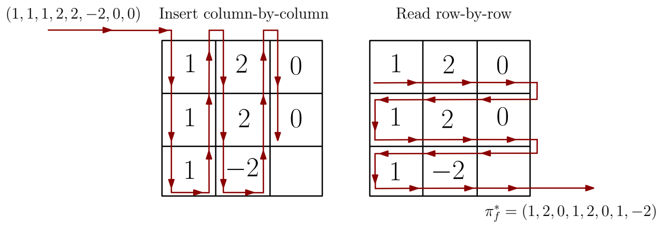


Fig. 7. The construction of the RFIP for $(n,d) = (8,5)$.

It is very straightforward to write down the expression of RFIP π_f^* in a way similar to (23). However, such an expression will be quite cumbersome and we thus omit the exact analytical definition and rely on the operational definition provided in Fig. 7 and in the previous paragraph.

B. Performance of the Family Helper Selection Scheme

Proposition 5: We use FHS to denote the family helper selection scheme. We then have that

$$\min_{G \in \mathcal{G}_{\text{FHS}}} \min_{t \in \text{DC}(G)} \text{mincut}_G(s,t) = \min_{\forall \pi_f} \sum_{i=1}^{k} \min\left((d - y_i(\pi_f))\beta, \alpha\right), \quad (24)$$

where π_f can be any family index permutation and the corresponding $y_i(\pi_f)$ is defined in Appendix D-A2.

The proof of Proposition 5 is relegated to Appendix G.

Combining Proposition 5 and (7), we can derive the new storage-bandwidth tradeoff (α vs. β) for the family helper selection scheme. For example, Fig. 3 plots α versus $\gamma \triangleq d\beta$ for the (n,k,d) values $(20,10,10)$ with file size $\mathcal{M} = 1$. As can be seen in Fig. 3, the MBR point (the smallest γ value) of FHS uses only 73.33% of the repair-bandwidth of the MBR point of BHS ($\gamma_{\text{MBR}} = \frac{2}{15}$ vs. $\frac{2}{11}$). It turns out that for any (n,k,d) values, the biggest improvement of FHS over BHS always happens at the MBR point.[16] The intuition is that choosing the good helpers is most beneficial when the per-node storage α is no longer a bottleneck (thus the MBR point).

The right-hand side of (24) involves taking the minimum over all possible permutation π_f, which, when focusing only on the first k coordinates, has $\mathcal{O}\left(\left(\frac{n}{n-d}\right)^k\right)$ distinct possibilities. As a result, computing the entire storage-bandwidth tradeoff curve of the family helper selection scheme is of complexity $\mathcal{O}\left(\left(\frac{n}{n-d}\right)^k\right)$. The following proposition shows that if we are interested in the most beneficial point, the MBR point, then we can compute the corresponding α and β values in polynomial time.

Proposition 6: For the MBR point of (24), i.e., when α is sufficiently large, the minimizing family index permutation is the RFIP π_f^* defined in Appendix D-A3. That is, for the family helper selection scheme the α, β, and γ values of the MBR point can be computed by

$$\alpha_{\text{MBR}} = \gamma_{\text{MBR}} = d\beta_{\text{MBR}} = \frac{d\mathcal{M}}{\sum_{i=1}^{k}(d - y_i(\pi_f^*))}. \qquad (25)$$

The proof of Proposition 6 is relegated to Appendix J.

Using Proposition 6 above, we can quickly find the MBR point of the family helper selection tradeoff curve in Fig. 3 without taking the minimum over all π_f. Specifically, this is done by first finding the RFIP π_f^* for the parameter $(n, k, d) = (20, 10, 10)$, which is $\pi_f^* = (1, 2, 1, 2, \ldots, 1, 2)$. Then we compute the corresponding value $\sum_{i=1}^{k}(d - y_i(\pi_f^*)) = 75$. Recall that $\mathcal{M}$ is assumed to be 1 in Fig. 3. Using (25), we thus get that $\gamma_{\text{MBR}} = \frac{2}{15}$ when family helper selection is applied to $(n, k, d) = (20, 10, 10)$.

C. Performance of the Family-plus Helper Selection Schemes

Recall that unlike the family helper selection scheme, the family-plus helper selection scheme is not unique under a given (n, k, d) value. Instead, it depends on a set of auxiliary parameters $n_1, \cdots, n_B$ satisfying $\sum_{b=1}^{B} n_b = n$ and $\min_b n_b \geq d + 1$. See Section VI-B. The performance of the family-plus helper selection scheme of parameters $\{n_b\}$ is characterized as follows.

Proposition 7: We use $\text{FP}\{n_b\}$ to denote the family-plus helper selection scheme with auxiliary parameters $\{n_b\}$. We then have

$$\min_{G \in \mathcal{G}_{\text{FP}\{n_b\}}} \min_{t \in \text{DC}(G)} \text{mincut}(s, t) =$$
$$\min_{\mathbf{k} \in \mathcal{K}} \sum_{b=1}^{B} \min_{G_b \in \mathcal{G}_{\text{FHS}}(n_b, k_b, d, \alpha, \beta)} \min_{t_b \in \text{DC}(G_b)} \text{mincut}_{G_b}(s, t_b), \qquad (26)$$

where $\mathbf{k}$ is a B-dimensional integer-valued vector, and $\mathcal{K}$ contains all the $(k_1, k_2, \cdots, k_B)$ vectors satisfying $0 \leq k_b \leq n_b$ and $\sum_{b=1}^{B} k_b = k$. Note that for any given $\mathbf{k}$, the right-hand side of (26) can be evaluated by Proposition 5.

Proof: We observe that any IFG $G \in \mathcal{G}_{\text{FP}\{n_b\}}$ is a union of B "parallel" IFGs. Specifically, each parallel IFG is an entry of the IFG collection $\mathcal{G}_{\text{FHS}}(n_b, -, d, r, \alpha, \beta)$ where instead of having a parameter k we use "$-$" to mean that we temporarily ignore the placement of the data collectors. We let the B parallel IFGs share the same common root, denoted by s. The reason behind the above observation is that all the repair operations are performed within each group (out of totally B groups). Therefore, the overall IFG will simply be the union of B parallel IFGs that share a common root s.

For any data collector t in $G_{\text{FP}\{n_b\}}$, we use k_b to denote the number of active nodes that t accesses in group b. Therefore, the $\text{mincut}_G(s, t)$ of the overall IFG is simply the summation of the $\text{mincut}_{G_b}(s, t_b)$ for each constituent IFG for all $b \in \{1, \cdots, B\}$ where t_b corresponds to the "sub-data-collector" of group b and G_b is the b-th parallel IFG. Since we run the family helper selection scheme in each of the b-th group, G_b is a member of $\mathcal{G}_{\text{FHS}}(n_b, k_b, d, \alpha, \beta)$. By further minimizing over all possible data collectors t (thus minimizing over $\{k_b\}$), we get (26). ■

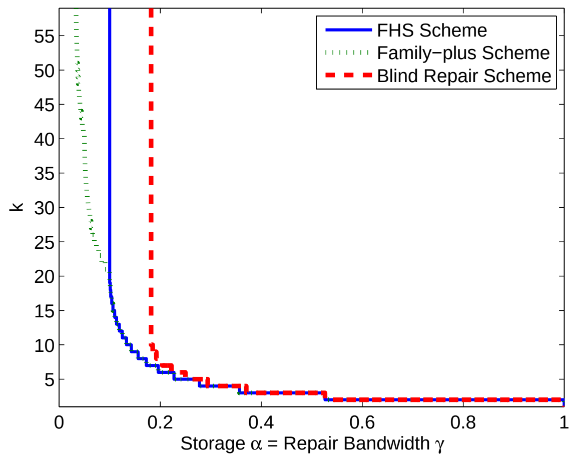


Fig. 8. The k value versus repair-bandwidth γ curve comparison between family helper selection, family-plus, and blind helper selection at the MBR point for $(n, d) = (60, 10)$ and file size $\mathcal{M} = 1$.

In Fig. 8, we plot the k versus γ curves for the blind helper selection, family helper selection, and the family-plus helper selection schemes for the case of $(n, d) = (60, 10)$ using Proposition 6 and by analyzing the minimum-bandwidth regenerating point of family-plus helper selection, respectively. As can be seen in Fig. 8, when $k > d$, BHS stops improving any further since RCs with BHS always have $k^* \leq d$ and thus overprotect the data when the protection-level requirement $k > d$. Therefore, BHS is not able to take advantage of the looser protection-level requirement when $k > d$. In contrast, the bandwidth consumption of FHS continues to decrease until the improvement stops when $k > 2d$. The reason is that, for $(n, d) = (60, 10)$, FHS only has two families. The family-plus scheme, on the other hand, divides $n = 60$ nodes into $B = 3$ groups, $n_1 = n_2 = n_3 = 20$, and each group has

[16]If we compare the min-cut value of FHS in (24) with the min-cut value of BHS in (6), we can see that the greatest improvement happens when the new term $(d - y_i(\pi_f))\beta \leq \alpha$ for all i. These are the mathematical reasons why the MBR point sees the largest improvement.

2 complete families (6 families in total). As a result, the family-plus scheme can continue harvesting the looser and looser protection-level requirement even when $k > 2d$ and the bandwidth consumption keeps decreasing continuously. For example, when $k = 40$, the repair-bandwidth of the family-plus helper selection scheme is only 28% of the repair-bandwidth of BHS.

D. Proving the Achievability Result by Family and Family-plus Helper Selection Schemes

To prove the achievability direction, Proposition 2, we assume that the (n, k, d) value satisfies neither (13) nor (14). One can verify by exhaustively considering all scenarios that for those (n, k, d) values of interest, one of the following three conditions must hold:

$$\text{Condition 3: } d \geq 2 \text{ and } k > \left\lceil \frac{n}{n-d} \right\rceil; \quad (27)$$

$$\text{Condition 4: } d = 1, k > 2, \text{ and } n \text{ is even}; \quad (28)$$

$$\text{or Condition 5: } d = 1, k > 3, \text{ and } n \text{ is odd}. \quad (29)$$

Case 1: If (27) holds, we first note that since $k > \left\lceil \frac{n}{n-d} \right\rceil$, we must also have $d \leq n-2$. Otherwise, we will have $k > n$, which contradicts (1). We then observe[17] that whenever $2 \leq d \leq n-2$, we must also have $d > \left\lceil \frac{n}{n-d} \right\rceil - 1$. As a result, (27) implies that $\min(d+1, k) > \left\lceil \frac{n}{n-d} \right\rceil$.

We now analyze the family helper selection scheme when (27) holds and we focus on the corresponding MBR point. The MBR performance of FHS is summarized in Proposition 6. For reference, the MBR point of the blind helper selection scheme is derived in [7]:

$$\alpha_{\text{MBR}} = \gamma_{\text{MBR}} = d\beta_{\text{MBR}} = \frac{d\mathcal{M}}{\sum_{i=1}^{k}(d-(i-1))^+}. \quad (30)$$

We then notice that by the definition of $y_i(\pi_f)$ in Appendix D-A2, we always have $y_i(\pi_f) \leq \min(d, i-1)$ regardless which π_f we consider. The reason is that $y_i(\pi_f)$ basically returns the number of nodes in the first $(i-1)$ coordinates of π_f that can be a helper of node $[\pi_f]_i$. Therefore, we have

$$\begin{aligned}\sum_{i=1}^{k}(d - y_i(\pi_f^*)) &\geq \sum_{i=1}^{\min(d+1,k)}(d - y_i(\pi_f^*)) \\ &\geq \sum_{i=1}^{\min(d+1,k)}(d-(i-1)) \quad (31)\\ &= \sum_{i=1}^{k}(d-(i-1))^+.\end{aligned}$$

On the other hand, since there are exactly $\left\lceil \frac{n}{n-d} \right\rceil$ families in FHS, among the first $\min(d+1,k)$ coordinates of the RFIP π_f^* there is at least one family index that is repeated. Then again by the definition of $y_i(\pi_f^*)$, we have a strict inequality $y_{i_0}(\pi_f^*) < (i-1)$ for some $i_0 \leq \min(d+1,k)$. Therefore the inequality (31) must also be a strict inequality. As a result, the MBR point of the family helper selection scheme (25) is strictly smaller than the MBR of blind helper selection (30). The family helper selection scheme thus strictly improves the performance and the given (n, k, d) value is not indifferent-to-helper-selection.

Case 2: If (28) holds, since $n > k$ (by (1)) we have $n \geq 4$. We now analyze the family-plus helper selection scheme. Recall that the family-plus helper selection scheme can be tuned by different choices of $\{n_b : b = 1, \cdots, B\}$. We consider a special choice of $\{n_b\}$ parameters as follows. Since $d = 1$ and n is even, we choose $B = \frac{n}{2}$ and choose each $n_b = 2d = 2$ for $b = 1$ to B. We denote this special choice of parameters by $\{n_b^*\}$.

We now apply Proposition 7 to characterize the performance of FP$\{n_b^*\}$. Consider any vector $\mathbf{k} \in \mathcal{K}$ defined in Proposition 7. Since $k > 2$ and $k_b \leq n_b^* = 2$ for all b, there are at least two distinct b values with $k_b \geq 1$. Using this observation and (26), we have that

$$\begin{aligned}\min_{G \in \mathcal{G}_{\text{FP}\{n_b^*\}}} \min_{t \in \text{DC}(G)} \text{mincut}_G(s,t) &\geq 2\min(d\beta, \alpha) \\ &> \min(\beta, \alpha), \quad (32)\end{aligned}$$

where the first inequality follows from: (i) considering only those b values with $k_b \geq 1$; (ii) plugging in the min-cut formula in Proposition 5; and (iii) only counting the first term "$i = 1$" when summing up for all $i = 1$ to k_b. The second inequality follows from $d = 1$ and the fact that both β and α must be strictly positive. Also, notice that the performance characterization of BHS in (6) becomes $\min(\beta, \alpha)$ when (28) holds. Therefore, (32) implies that the family-plus helper selection scheme with $\{n_b^*\}$ strictly outperforms BHS. The proof of Case 2 is complete.

Case 3: If (29) holds, since $n > k$ (by (1)) we have $n \geq 5$. We again analyze the family-plus helper selection scheme with a special choice of $\{n_b\}$ parameters as follows. Since $d = 1$ and n is odd, we choose $B = \frac{n-1}{2}$ and choose each $n_b = 2$ for $b = 1$ to $B - 1$ and set $n_B = 3$. We denote this special choice of parameters by $\{n_b^*\}$. Note that since $n \geq 5$, we must have at least two groups since $B \geq 2$.

We now apply Proposition 7 to characterize the performance of FP$\{n_b^*\}$. Consider any vector $\mathbf{k} \in \mathcal{K}$ defined in Proposition 7. Since $k > 3$ and $k_b \leq n_b^* \leq 3$ for all b, there are at least two distinct b values with $k_b \geq 1$. Then by a verbatim analysis as in Case 2, the family-plus helper selection scheme with $\{n_b^*\}$ strictly outperforms BHS when (29) holds. The proof of Case 3 is complete.

By the discussion in Cases 1 to 3, the proof of Proposition 2 is complete.

Appendix E
Proof of Proposition 3

The proof of Proposition 3 consists of two parts. Firstly, we characterize the storage-tradeoff curve of the family helper selection scheme by the following lemmas:

[17]A detailed proof of this simple algebraic observation can be found in the proof of Lemma 5 around (37) in Appendix E.

Lemma 5: For any (n,k,d) values satisfying $d \geq 2$ and $k = \left\lceil \frac{n}{n-d} \right\rceil + 1$, we have

$$\min_{G\in\mathcal{G}_{\rm FHS}} \min_{t\in {\rm DC}(G)} {\rm mincut}(s,t) = \sum_{i=3}^{k} \min((d-(i-1))\beta,\alpha) + 2\min(d\beta,\alpha). \quad (33)$$

After proving Lemma 5 we will prove the following lemma:

Lemma 6: Consider any fixed (n,k,d) values that satisfy (15) to (17) and any $G \in \mathcal{G}(n,k,d,\alpha,\beta)$ where all the active nodes of G have been repaired at least once. There always exists a data collector, denoted by $t_2 \in {\rm DC}(G)$, such that

$${\rm mincut}_G(s,t_2) \leq \sum_{i=3}^{k} \min((d-(i-1))\beta,\alpha) + 2\min(d\beta,\alpha). \quad (34)$$

We then notice that any (n,k,d) values that satisfy (15) to (17) must also satisfy the conditions in Lemma 5. Together Lemmas 5 and 6 imply Proposition 3 since it says that no matter how we design the helper selection scheme H, the resulting $G \in \mathcal{G}_H \subseteq \mathcal{G}$ will have $\min_{t\in{\rm DC}(G)} {\rm mincut}_G(s,t) \leq \min_{\tilde{G}\in\mathcal{G}_{\rm FHS}} \min_{t\in{\rm DC}(\tilde{G})} {\rm mincut}_{\tilde{G}}(s,t)$. The proof of Proposition 3 is thus complete.

The proofs of Lemmas 5 and 6 will be provided in the following two subsections.

A. Proof of Lemma 5

The proof of Lemma 5 relies on the performance characterization of the family helper selection scheme in Proposition 5 of Appendix D-B. We consider two sub-cases separately.

1) Case 1: $d \geq k-1 = \left\lceil \frac{n}{n-d} \right\rceil$: Since there are $\left\lceil \frac{n}{n-d} \right\rceil$ number of families (complete plus incomplete families) and $k = \left\lceil \frac{n}{n-d} \right\rceil + 1$, any family index permutation π_f has at least one pair of indices of the same family in its first k coordinates. By the same analysis as in the paragraph right after (30), we have $y_i(\pi_f) \leq i-1$; and for at least one $i_0 \leq k$, we have $y_{i_0}(\pi_f) \leq i_0 - 2$. Since $y_i(\pi_f)$ is always non-negative, we implicitly have $i_0 \geq 2$.

Using (24), the above observation implies that

$$\min_{G\in\mathcal{G}_{\rm FHS}} \min_{t\in {\rm DC}(G)} {\rm mincut}(s,t) = \min_{\forall \pi_f} \sum_{i=1}^{k} \min\left((d-y_i(\pi_f))\beta,\alpha\right) \geq \min_{2\leq i_0\leq k} C_{i_0} \quad (35)$$

where

$$C_{i_0} = \sum_{i=1}^{i_0-1} \min((d-(i-1))\beta,\alpha) + \min((d-(i_0-2))\beta,\alpha) + \sum_{i=i_0+1}^{k} \min((d-(i-1))\beta,\alpha).$$

Inequality (35) follows from replacing $y_i(\pi_f)$ by its upper bound $i-1$ for general i and the special $y_{i_0}(\pi_f)$ by its upper bound i_0-2.

We now prove that inequality (35) is actually an equality. Recall that the family index of the last complete family is $c_{\rm last} = \left\lfloor \frac{n}{n-d} \right\rfloor$ and recall that we assume $k = \left\lceil \frac{n}{n-d} \right\rceil + 1$. For any $2 \leq i_0 \leq k$ we then define $\pi_f^{[i_0]}$ as a family index permutation satisfying the following requirement.

Case 1.1: If $n \bmod (n-d) \neq 0$, then we have $k = c_{\rm last} + 2$. We then require that the first k coordinates of $\pi_f^{[i_0]}$, in this order, are

$$\overbrace{1,2,\cdots,i_0-1}^{i_0-1 \text{ terms}},1,\overbrace{i_0,\cdots,c_{\rm last}}^{c_{\rm last}-(i_0-1) \text{ terms}},0.$$

Case 1.2: If $n \bmod (n-d) = 0$, then we have $k = c_{\rm last} + 1$. We then require that the first k coordinates of $\pi_f^{[i_0]}$ are

$$\overbrace{1,2,\cdots,i_0-1}^{i_0-1 \text{ terms}},1,\overbrace{i_0,\cdots,c_{\rm last}}^{c_{\rm last}-(i_0-1) \text{ terms}}.$$

Since all k coordinates of $\pi_f^{[i_0]}$ have different values except that the values of both the first and the i_0-th coordinate are 1, and since none of the first k coordinates of $\pi_f^{[i_0]}$ has its value being $-c_{\rm last}$, we have

$$\sum_{i=1}^{k} \min\left(\left(d-y_i\left(\pi_f^{[i_0]}\right)\right)\beta,\alpha\right) = C_{i_0}.$$

Thus, (35) can be strengthened as the following equality

$$\min_{G\in\mathcal{G}_{\rm FHS}} \min_{t\in {\rm DC}(G)} {\rm mincut}(s,t) = \min_{2\leq i_0\leq k} C_{i_0}.$$

By observing that the right-hand side of (33) is identical to C_2 (i.e., C_{i_0} with $i_0 = 2$), what remains to be proved is that $\min_{2\leq i_0\leq k} C_{i_0} = C_2$.

To that end, we notice that

$$C_{i_0} - C_2 = \min((d-1)\beta,\alpha) - \min(d\beta,\alpha) + \min((d-(i_0-2))\beta,\alpha) - \min((d-(i_0-1))\beta,\alpha). \quad (36)$$

Since we always have $C_{i_0} - C_2 = 0$ when $i_0 = 2$, we only consider the i_0 values satisfying $3 \leq i_0 \leq k$. We then observe that the α value in (36) is compared to four different values: $(d-(i_0-1))\beta$, $(d-(i_0-2))\beta$, $(d-1)\beta$, and $d\beta$, listed from the smallest to the largest. Depending on the relative order between α and these 4 values, we have 5 cases:

- If $\alpha \leq (d-(i_0-1))\beta$, then
$$C_{i_0} - C_2 = \alpha - \alpha + \alpha - \alpha = 0.$$
- If $(d-(i_0-1))\beta \leq \alpha \leq (d-(i_0-2))\beta$, then
$$C_{i_0} - C_2 = \alpha - \alpha + \alpha - (d-(i_0-1))\beta \geq \alpha - \alpha + \alpha - \alpha = 0.$$
- If $(d-(i_0-2))\beta < \alpha \leq (d-1)\beta$ (this case does not exist for $i_0 = 3$), then
$$C_{i_0} - C_2 = \alpha - \alpha + (d-(i_0-2))\beta - (d-(i_0-1))\beta = \beta \geq 0.$$

- If $(d-1)\beta < \alpha \leq d\beta$, then

$$C_{i_0} - C_2 = \alpha - (d-1)\beta + \beta \geq \alpha - \alpha + \beta \geq 0.$$

- If $\alpha \geq d\beta$, then $C_{i_0} - C_2 = (d-1)\beta - d\beta + \beta = 0$.

We have shown by the above that $C_{i_0} \geq C_2$ for all $3 \leq i_0 \leq k$. Therefore, we have proved $\min_{2 \leq i_0 \leq k} C_{i_0} = C_2$ and the equality in (33).

2) Case 2: $d < k-1 = \left\lceil \frac{n}{n-d} \right\rceil$: Before proceeding, we first argue that among all (n,k,d) values satisfying $d \leq \left\lceil \frac{n}{n-d} \right\rceil - 1$ are either $d=1$ or $d=n-1$. We prove this argument by contradiction. Suppose $d \leq \left\lceil \frac{n}{n-d} \right\rceil - 1$ and $2 \leq d \leq n-2$. For any $2 \leq d \leq n-2$, we have

$$\begin{aligned} 0 \leq \left\lceil \frac{n}{n-d} \right\rceil - 1 - d &= \left\lceil 1 + \frac{d}{n-d} \right\rceil - 1 - d \\ &= \left\lceil \frac{d}{n-d} \right\rceil - d \\ &\leq \left\lceil \frac{d}{2} \right\rceil - d \quad (37) \\ &= \begin{cases} -\frac{d}{2}, & \text{if } d \text{ is even} \\ \frac{1-d}{2}, & \text{if } d \text{ is odd} \end{cases} \\ &< 0, \quad (38) \end{aligned}$$

where we get (37) by our assumption that $d \leq n-2$ and (38) follows from the assumption that $d \geq 2$. The above contradiction implies that when $d \leq \left\lceil \frac{n}{n-d} \right\rceil - 1$ we have either $d=1$ or $d=n-1$.

Since Lemma 5 requires $d \geq 2$, the only remaining possibility in this case of $d \leq \left\lceil \frac{n}{n-d} \right\rceil - 1$ is when $d = n-1$. However, this is also impossible since Lemma 5 also assumes $k = \left\lceil \frac{n}{n-d} \right\rceil + 1$, which, together with $d = n-1 \leq \left\lceil \frac{n}{n-d} \right\rceil - 1$, implies $k \geq n+1$, an impossible parameter value violating (1). Hence, the proof of Lemma 5 is complete.

B. Proof of Lemma 6

Lemma 6 can be viewed as the converse of Lemma 5, i.e., it shows that the performance of any helper selection scheme $H \in \mathcal{DHS}$ is no better than that of the family helper selection scheme.

To prove Lemma 6 we follow the same approach as used in proving the converse direction, Proposition 1, in Appendix C. That is, suppose the (n,k,d) value satisfies (15) to (17). Then regardless how we design the helper selection rules, there exists a "harmful" sub-structure of k active nodes in the IFG G after every node has been repaired at least once. And if we let the data collector $t_2 \in \mathrm{DC}(G)$ connect to those k nodes, then (34) will hold.

We first define the harmful sub-structure as follows.

Definition 14: A set of m active storage nodes (input-output pairs) of an IFG is called an $(m,2)$-set if the following conditions are satisfied simultaneously. (i) Each of the m active nodes has been repaired at least once; (ii) for easier reference, we use x_1 and x_2 to denote the oldest and the second-oldest nodes, respectively, among the m nodes of interest. If we temporarily add an edge connecting $x_{1,\text{out}}$ and $x_{2,\text{in}}$, then we require that the m nodes of interest form an m-set as defined in Definition 12.

Intuitively, an $(m,2)$-set can be viewed as a generalized version of the m-set in Definition 12. That is, in an m-set all m nodes are directly connected by single edges. In an $(m,2)$-set, the only possible "disconnect" among the m nodes is between $x_{1,\text{out}}$ and $x_{2,\text{in}}$ and every other node pair must be connected. Note that whether $x_{2,\text{in}}$ and $x_{1,\text{out}}$ are actually connected or not is of no significance in this definition. Therefore, an m-set is always an $(m,2)$-set but not vice versa.

The following lemma proves the existence of an $(m,2)$-set in an IFG, which has a similar role as Lemma 2 in Appendix C.

Lemma 7: Consider any $G \in \mathcal{G}(n,k,d,\alpha,\beta)$ where (n,k,d) satisfy (15) to (17). Consider any $m \geq 2$, there exists an $(m,2)$-set in every group of $2(m-1)$ active nodes.

Proof: The proof is similar to the proof of Lemma 2. We prove this lemma by induction. Lemma 7 holds naturally for the case of $m=2$ since every group of 2 active nodes, by Definition 14, is a $(2,2)$-set.

Now, assume that Lemma 7 holds for $m \leq m_0 - 1$. Consider any set of $2(m_0-1)$ active nodes of G. We call this set S_0 and we use y to denote the youngest node of S_0. Since there are $2(m_0-1)-1$ nodes in $S_0 \backslash \{y\}$ and since $d = n-2$ means that each node can avoid connecting to at most 1 active node, we thus have that y is connected to at least $2(m_0-1)-2$ older nodes in S_0. Call this set of $2(m_0-2)$ nodes S_1. By the induction assumption, there exists an $(m_0-1,2)$-set in S_1. We now argue that the union of this $(m_0-1,2)$-set and node y, called S_2, form an $(m_0,2)$-set. The reason is that the first and the second oldest nodes in this $(m_0-1,2)$-set are also the first and the second oldest nodes in S_2 since node y is the youngest in S_0. Also, since node y is connected to all nodes in S_1, it must also connect to all nodes in this $(m_0-1,2)$-set. As a result, S_2 satisfies properties (i) and (ii) in Definition 14 and thus form an $(m_0,2)$-set. Hence, the proof is complete. ■

Lemma 7 shows that for any (n,k,d) value satisfying (15) to (17) and for any $G \in \mathcal{G}(n,k,d,\alpha,\beta)$ where all the active nodes of G have been repaired at least once, there exist a $(\frac{n}{2}+1,2)$-set. We then consider a data collector that connects to this $(\frac{n}{2}+1,2)$-set and we denote it by t_2.

We now apply a similar analysis as in the proof of [7, Lemma 2] to prove that (34) is true for the t_2 we are considering. Denote the storage nodes (input-output pair) of this $(\frac{n}{2}+1,2)$-set by $x^1, x^2, \ldots, x^{\frac{n}{2}+1}$, where node x^1 is the oldest and node $x^{\frac{n}{2}+1}$ is the youngest. Define cut $(U,\overline{U})$ between t_2 and s as the following: We start with $\overline{U} = \{t_2\}$. Then for each $i \in \{1,3,4,\ldots,\frac{n}{2}+1\}$, if $\alpha \leq (d-(i-1))\beta$ then we further include x^i_{out} in $\overline{U}$; otherwise, we include both x^i_{out} and x^i_{in} in $\overline{U}$. For $i=2$, if $\alpha \leq d\beta$, then we include x^2_{out} in $\overline{U}$; otherwise, we include both x^2_{out} and x^2_{in} in $\overline{U}$. Finally, we set U to include any node in the IFG that has not been included in $\overline{U}$. It is not hard to see that the above construction of $(U,\overline{U})$ leads to a cut and the corresponding cut-value is no larger than $\sum_{i=3}^{k} \min((d-(i-1))\beta,\alpha) + 2\min(d\beta,\alpha)$. Since the min-cut value between (s,t_2) is no larger than the cut-value of one specific cut, we have proven (34). The proof of Lemma 6 is thus complete.

Appendix F
Proof of Proposition 4

The following existing result in [11, Theorem 5.2] will be used extensively in this proof. Specifically, [11, Theorem 5.2] proved that for $k = n-1$ and $\alpha = d\beta$,

$$\min_{G\in\mathcal{G}_H}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}_G(s,t) \leq \frac{nd\beta}{2} \tag{39}$$

for any arbitrary helper selection scheme $H \in \mathcal{DHS}$. We first prove Proposition 4 for the simplest case of $B = 1$. Namely, there is only 1 group of nodes and it satisfies $n \bmod (n-d) = 0$. We will prove that in this case, the repair bandwidth of the MBR point of the family helper selection scheme is indeed $\frac{nd\beta}{2}$. Therefore, the family helper selection is *bandwidth-optimal*.

Later in a separate sub-section we prove Proposition 4 for arbitrary $B \geq 2$.

A. Proof of Proposition 4 for the Case of $B = 1$

When $B = 1$, the family-plus helper selection scheme collapses to the original family helper selection scheme. Since $\alpha = d\beta$, we know by Proposition 6 that

$$\min_{G\in\mathcal{G}_{\mathrm{FHS}}}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}_G(s,t) = \sum_{i=1}^{n-1}(d - y_i(\pi_f^*))\beta. \tag{40}$$

Now, when $n \bmod (n-d) = 0$, we have no incomplete family in the family helper selection scheme and the RFIP π_f^* has the following form

$$\pi_f^* = (1,2,\cdots,c_{\text{last}},1,2,\cdots,c_{\text{last}},\cdots,1,2,\cdots,c_{\text{last}}), \tag{41}$$

where recall that $c_{\text{last}} = \left\lfloor\frac{n}{n-d}\right\rfloor = \frac{n}{n-d}$ is the index of the last complete family. Using (41), we get that

$$y_i(\pi_f^*) = i - 1 - \left\lfloor\frac{i-1}{c_{\text{last}}}\right\rfloor. \tag{42}$$

The reason behind (42) is the following. Examining the definition of $y_i(\cdot)$, we can see that $y_i(\cdot)$ counts all the coordinates $j < i$ of π_f^* that have a family index different than the family index at the i-th coordinate. For each coordinate i, with the aid of (41), there are $\left\lfloor\frac{i-1}{c_{\text{last}}}\right\rfloor$ coordinates in π_f^* preceding it with the same family index. Therefore, in total there are $i-1-\left\lfloor\frac{i-1}{c_{\text{last}}}\right\rfloor$ coordinates in π_f^* preceding the i-th coordinate with a different family index, thus, we get (42).

By (40) and (42), we get

$$\begin{aligned}\min_{G\in\mathcal{G}_{\mathrm{FHS}}}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}_G(s,t) &= \sum_{i=0}^{n-2}\left(d-i+\left\lfloor\frac{i}{\frac{n}{n-d}}\right\rfloor\right)\beta\\
&= \sum_{i=0}^{n-1}\left(d-i+\left\lfloor\frac{i}{\frac{n}{n-d}}\right\rfloor\right)\beta \qquad (43)\\
&= \left(nd - \frac{(n-1)n}{2} + \sum_{i=0}^{n-1}\left\lfloor\frac{i}{\frac{n}{n-d}}\right\rfloor\right)\beta\\
&= \left(nd - \frac{(n-1)n}{2} + \frac{n}{n-d}\sum_{\tilde{i}=0}^{n-d-1}\tilde{i}\right)\beta\\
&= \left(nd - \frac{(n-1)n}{2} + \frac{n(n-d-1)}{2}\right)\beta\\
&= \frac{nd\beta}{2}, \qquad (44)\end{aligned}$$

where we get (43) by the fact that $d-(n-1)+\left\lfloor\frac{n-1}{c_{\text{last}}}\right\rfloor = d-(n-1)+(n-d-1) = 0$. Since the repair bandwidth of family helper selection in (44) matches the upper bound in (39), the proof is thus complete.

B. Proof of Proposition 4 for the Case of $B \geq 2$

By Proposition 7 and the fact that $k = n-1$, we must have all but one $k_b = n_b$ and the remaining one $k_b = n_b - 1$. Without loss of generality, we assume $k_1 = n_1 - 1$ and all other $k_b = n_b$ for $b = 2$ to B for the minimizing $\mathbf{k}$ vector in (26). Since $n_1 \bmod (n_1 - d) = 0$, by the proof for the case of $B = 1$, the first summand of (26) must be equal to $\frac{n_1 d\beta}{2}$.

For the case of $b = 2$ to B, we have $k_b = n_b$ instead of $k_1 = n_1 - 1$. However, if we examine the proof for the case of $B = 1$, especially (43), we can see that (44) holds even for the case of $k = n$. By applying (44) again, the b-th summand of (26), $b = 2$ to B, must be $\frac{n_b d\beta}{2}$ as well.

Finally, by Proposition 7 we then have

$$\min_{G\in\mathcal{G}_{\mathrm{FP}\{n_b\}}}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}_G(s,t) = \sum_{b=1}^{B}\frac{n_b d\beta}{2} = \frac{nd\beta}{2}. \tag{45}$$

Since the repair bandwidth of family plus helper selection in (45) matches the upper bound in (39), the proof is thus complete.

Appendix G
Proof of Proposition 5

The proof of Proposition 5 consists of three parts:

Part I: We denote the collection of helper sets of a family helper selection scheme by $\{D_1, D_2, \ldots, D_n\}$, where each D_i is the helpers when node i fails. We will prove the following lemma.

Lemma 8: For the family helper selection scheme, we have

$$\begin{aligned}&\min_{G\in\mathcal{G}_{\mathrm{FHS}}}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}(s,t)\\
&\qquad\geq \min_{\mathbf{r}\in R}\sum_{i=1}^{k}\min((d-z_i(\mathbf{r}))\beta,\alpha), \qquad (46)\end{aligned}$$

where $\mathbf{r}$ is a k-dimensional integer-valued vector, $R = \{(r_1, r_2, \cdots, r_k) : \forall i \in \{1, \cdots, k\}, 1 \leq r_i \leq n\}$, and $z_i(\cdot)$ is a function $z_i : \{1, \cdots, n\}^k \mapsto \mathbb{N}$ defined as $z_i(\mathbf{r}) = |\{a \in D_{r_i} : \exists j < i, a = r_j\}|$, where $\mathbb{N}$ is the set of all positive integers and D_{r_i} is the helper set of node r_i.

Intuitively, the function $z_i(\mathbf{r})$ returns "how many of the first $(i-1)$ coordinates of $\mathbf{r}$ can potentially be a helper of the i-th node r_i." For example, suppose $n = 6$, $k = 4$, $D_3 = \{1, 4\}$, and $\mathbf{r} = (1, 2, 1, 3)$. Then for the fourth node in $\mathbf{r}$, which is node $r_4 = 3$, we have $z_4(\mathbf{r}) = |\{a \in D_3 : \exists j < 4, a = r_j\}| = 1$ since out of the two elements in $D_3 = \{1, 4\}$, only node 1 has appeared in the first three coordinates of $\mathbf{r}$. Note that even though node 1 appears twice in the first three coordinates of $\mathbf{r}$, the definition of $z_4(\mathbf{r})$ only counts it once and the output is thus 1.

Part II: We will prove the following lemma.

Lemma 9:

$$\min_{G \in \mathcal{G}_{\text{FHS}}} \min_{t \in \text{DC}(G)} \text{mincut}_G(s, t) \leq \min_{\forall \pi_f} \sum_{i=1}^{k} \min\left(\left(d - y_i(\pi_f)\right)\beta, \alpha\right). \quad (47)$$

Part III: Lemmas 8 and 9 provide a pair of lower and upper bounds on the min-cut value of the family helper selection scheme. The final piece of the proof of Proposition 5 is the following lemmas.

Recall that in (46) of Lemma 8, the minimizing vector $\mathbf{r}$ is chosen from $R = \{(r_1, r_2, \cdots, r_k) : \forall i \in \{1, \cdots, k\}, 1 \leq r_i \leq n\}$. If we define

$$R_2 = \{\mathbf{r} \in R : \text{all } k \text{ coordinates of } \mathbf{r} \text{ have distinct values}\},$$

then it is clear that $R_2 \subsetneq R$. As a result we immediately have

$$\min_{\mathbf{r} \in R_2} \sum_{i=1}^{k} \min((d - z_i(\mathbf{r}))\beta, \alpha) \geq \min_{\mathbf{r} \in R} \sum_{i=1}^{k} \min((d - z_i(\mathbf{r}))\beta, \alpha). \quad (48)$$

We now present two additional lemmas:

Lemma 10:

$$\min_{\forall \pi_f} \sum_{i=1}^{k} \min\left(\left(d - y_i(\pi_f)\right)\beta, \alpha\right) \leq \min_{\mathbf{r} \in R_2} \sum_{i=1}^{k} \min((d - z_i(\mathbf{r}))\beta, \alpha).$$

Lemma 11:

$$\min_{\mathbf{r} \in R_2} \sum_{i=1}^{k} \min((d - z_i(\mathbf{r}))\beta, \alpha) = \min_{\mathbf{r} \in R} \sum_{i=1}^{k} \min((d - z_i(\mathbf{r}))\beta, \alpha). \quad (49)$$

Note that Lemmas 8 and 9 imply

$$\begin{aligned} &\min_{\forall \pi_f} \sum_{i=1}^{k} \min\left(\left(d - y_i(\pi_f)\right)\beta, \alpha\right) \\ &\geq \min_{G \in \mathcal{G}_{\text{FHS}}} \min_{t \in \text{DC}(G)} \text{mincut}(s, t) \\ &\geq \min_{\mathbf{r} \in R} \sum_{i=1}^{k} \min((d - z_i(\mathbf{r}))\beta, \alpha). \end{aligned}$$

On the other hand, Lemmas 10 and 11 imply

$$\begin{aligned} &\min_{\forall \pi_f} \sum_{i=1}^{k} \min\left(\left(d - y_i(\pi_f)\right)\beta, \alpha\right) \\ &\leq \min_{\mathbf{r} \in R} \sum_{i=1}^{k} \min((d - z_i(\mathbf{r}))\beta, \alpha). \end{aligned}$$

Together, we thus have proven (24). The proof of Proposition 5 is thus complete.

The proofs of Lemmas 8 to 11 are provided in the following subsections separately.

A. Proof of Lemma 8

The proof of Lemma 8 closely follows the proof of [7, Lemma 2].

Consider any IFG $G \in \mathcal{G}_{\text{FHS}}$. Consider any data collector t of G and call the set of k active output nodes it connects to V. Since all the incoming edges of t have infinite capacity, we can assume without loss of generality that the minimum cut $(U, \overline{U})$ satisfies $s \in U$ and $V \subseteq \overline{U}$.

Let $\mathcal{C}$ denote the set of edges in the minimum cut. Let x^i_{out} be the chronologically i-th output node in $\overline{U}$, i.e., from the oldest to the youngest. Since $V \subseteq \overline{U}$, there are at least k output nodes in $\overline{U}$. We now consider the oldest k output nodes of $\overline{U}$, i.e., x^1_{out} to x^k_{out}. For $i = 1$ to k, let r_i denote the node index of x^i_{out}. Obviously, the vector $\mathbf{r} \triangleq (r_1, \cdots, r_k)$ belongs to R.

Consider x^1_{out}, we have two cases:

- If $x^1_{\text{in}} \in U$, then the edge $(x^1_{\text{in}}, x^1_{\text{out}})$ is in $\mathcal{C}$.
- If $x^1_{\text{in}} \in \overline{U}$, since x^1_{in} has an in-degree of d and x^1_{out} is the oldest node in $\overline{U}$, all the incoming edges of x^1_{in} must be in $\mathcal{C}$.

From the above discussion, these edges related to x^1_{out} contribute at least a value of $\min((d - z_1(\mathbf{r}))\beta, \alpha)$ to the min-cut value since by definition $z_1(\mathbf{r}) = 0$. Now, consider x^2_{out}, we have three cases:

- If $x^2_{\text{in}} \in U$, then the edge $(x^2_{\text{in}}, x^2_{\text{out}})$ is in $\mathcal{C}$.
- If $x^2_{\text{in}} \in \overline{U}$ and $r_1 \in D_{r_2}$, since one of the incoming edges of x^2_{in} can be from x^1_{out}, then at least $(d-1)$ incoming edges of x^2_{in} are in $\mathcal{C}$.
- If $x^2_{\text{in}} \in \overline{U}$ and $r_1 \notin D_{r_2}$, since no incoming edges of x^2_{in} are from x^1_{out}, then all d incoming edges of x^2_{in} are in $\mathcal{C}$.

Therefore, these edges related to x^2_{out} contribute a value of at least $\min((d - z_2(\mathbf{r}))\beta, \alpha)$ to the min-cut value, where the definition of $z_2(\mathbf{r})$ takes care of the second and the third cases. Consider x^3_{out}, we have five cases:

- If $x_{\text{in}}^3 \in U$, then the edge $(x_{\text{in}}^3, x_{\text{out}}^3)$ is in $\mathcal{C}$.
- If $x_{\text{in}}^3 \in \overline{U}$ and $r_1 = r_2 \in D_{r_3}$, since one of the incoming edges of x_{in}^3 can be from x_{out}^2, then at least $(d-1)$ incoming edges of x_{in}^3 are in $\mathcal{C}$. Note that there cannot be an incoming edge of x_{in}^3 from x_{out}^1 since x_{in}^3 only connects to active output nodes at the time of repair and x_{out}^1 is no longer active since x_{out}^2 (of the same node index $r_2 = r_1$) has been repaired after x_{out}^1.
- If $x_{\text{in}}^3 \in \overline{U}$; $r_1, r_2 \in D_{r_3}$; and $r_1 \neq r_2$; since one of the incoming edges of x_{in}^3 can be from x_{out}^1 and another edge can be from x_{out}^2 , then at least $(d-2)$ incoming edges of x_{in}^3 are in $\mathcal{C}$.
- If $x_{\text{in}}^3 \in \overline{U}$ and only one of r_1 or r_2 is in D_{r_3}, since one of the incoming edges of x_{in}^3 is from either x_{out}^1 or x_{out}^2, then at least $(d-1)$ incoming edges of x_{in}^3 are in $\mathcal{C}$.
- If $x_{\text{in}}^3 \in \overline{U}$ and $r_1, r_2 \notin D_{r_3}$, then at least d incoming edges of x_{in}^3 are in $\mathcal{C}$.

Therefore, these edges related to x_{out}^3 contribute a value of at least $\min((d - z_3(\mathbf{r}))\beta, \alpha)$ to the min-cut value, where the definition of $z_3(\mathbf{r})$ takes care of the second to the fifth cases.

In the same manner, we can prove that the chronologically i-th output node in $\overline{U}$ contributes at least a value of $\min((d - z_i(\mathbf{r}))\beta, \alpha)$ to the min-cut value. If we sum all the contributions of the oldest k output nodes of $\overline{U}$ we get (46), a lower bound on the min-cut value.

B. *Proof of Lemma 9*

We prove Lemma 9 by explicitly constructing a $G \in \mathcal{G}_{\text{FHS}}$ and a cut $(U, \overline{U})$ for which the cut value equals to the right-hand side of (47). Then by taking the minimum over G and taking the minimum over all cuts $(U, \overline{U})$, we have (47).

Our construction of G and $(U, \overline{U})$ is as follows. Denote the smallest information flow graph in $\mathcal{G}_{\text{FHS}}(n, k, d, \alpha, \beta)$ by G_0. Specifically, all its nodes are intact, i.e., none of its nodes has failed before. Consider the family index permutation π_f that attains the minimization of the right-hand side of (47) and call it $\tilde{\pi}_f$. Fail each active node in $\{1, 2, \cdots, n\}$ of G_0 exactly once in a way that the sequence of the family indices of the failed nodes is $\tilde{\pi}_f$. Along this failing process, we repair the failed nodes according to the family helper selection scheme.

For example, let $(n, d) = (8, 5)$ and suppose the minimizing family index permutation is $\tilde{\pi}_f = (1, 2, 1, -2, 0, 0, 1, 2)$. Then, if we fail nodes 1, 4, 2, 6, 7, 8, 3, and 5 in this sequence, the corresponding family index sequence will be $(1, 2, 1, -2, 0, 0, 1, 2)$, which matches the given $\tilde{\pi}_f$. Note that the node failing sequence is not unique in our construction. For example, if we fail nodes 3, 5, 2, 6, 8, 7, 1, and 4 in this sequence, the corresponding family index vector is still $(1, 2, 1, -2, 0, 0, 1, 2)$. Any node failing sequence that matches the given $\tilde{\pi}_f$ will suffice in our construction. We denote the resulting new information flow graph by G'.

Consider a data collector t in G' that connects to the oldest k newcomers. (Recall that in our construction, G' has exactly n newcomers.) Now, by the same arguments as in [7, Lemma 2], we will prove that $\text{mincut}_{G'}(s, t) = \sum_{i=1}^{k} \min\left((d - y_i(\tilde{\pi}_f))\beta, \alpha\right)$ for the specifically constructed G' and t.

Number the storage nodes (input-output pair) of the k nodes t is connected to by $[1], [2], \ldots, [k]$. Here we use $[i]$ to denote the i-th oldest newcomer, which is generally not node i but a node with its family index equal to the i-th coordinate of $\tilde{\pi}_f$.

Define cut $(U, \overline{U})$ between t and s as the following: for each $i \in \{1, \ldots, k\}$, if $\alpha \leq (d - y_i(\tilde{\pi}_f))\beta$ then we include the output node of node $[i]$, i.e., $x_{\text{out}}^{[i]}$, in $\overline{U}$; Otherwise, we include both $x_{\text{out}}^{[i]}$ and $x_{\text{in}}^{[i]}$ in $\overline{U}$. It is not hard to see that the cut-value of the cut $(U, \overline{U})$ in our construction is equal to $\sum_{i=1}^{k} \min\left((d - y_i(\tilde{\pi}_f))\beta, \alpha\right)$ by counting the contribution of the cut-value for each node $x_{\text{out}}^{[i]}$ (and $x_{\text{in}}^{[i]}$). Our construction is complete and the proof of Lemma 9 is complete.

C. *Proof of Lemma 10*

We notice that any $\mathbf{r} \in R_2$ corresponds to the first k coordinates of a permutation of the node indices $(1, 2, 3, \cdots, n)$. For easier reference, we use $\bar{\mathbf{r}}$ to represent an n-dimensional permutation vector such that the first k coordinates of $\bar{\mathbf{r}}$ match $\mathbf{r}$. One can view $\bar{\mathbf{r}}$ as the extended version of $\mathbf{r}$ from a partial k-dimensional permutation to a complete n-dimensional permutation vector. Obviously, the choice of $\bar{\mathbf{r}}$ is not unique. The following discussion holds for any $\bar{\mathbf{r}}$.

For any $\mathbf{r} \in R_2$, we first find its extended version $\bar{\mathbf{r}}$. We then construct π_f from $\bar{\mathbf{r}}$ by transcribing the permutation of the node indices $\bar{\mathbf{r}}$ to the corresponding family indices. For example, consider the parameter values $(n, k, d) = (8, 4, 5)$. Then, one possible choice of $\mathbf{r} \in R_2$ is $\mathbf{r} = (3, 5, 2, 4)$ and the corresponding extended version $\bar{\mathbf{r}}$ being $(3, 5, 2, 4, 1, 6, 7, 8)$. The transcribed family index vector is $\pi_f = (1, 2, 1, 2, 1, -2, 0, 0)$. We now argue that $z_i(\mathbf{r}) = y_i(\pi_f)$ for all $i = 1$ to k. The reason is that the definition of $y_i(\pi_f)$ is simply a transcribed version of the original definition of $z_i(\mathbf{r})$ under the node-index to family-index translation. In sum, the above argument proves that for any $\mathbf{r} \in R_2$, there exists a π_f satisfying

$$\sum_{i=1}^{k} \min((d - z_i(\mathbf{r}))\beta, \alpha) = \sum_{i=1}^{k} \min\left((d - y_i(\pi_f))\beta, \alpha\right).$$

The proof of Lemma 10 is thus complete.

D. *Proof of Lemma 11*

The proof of Lemma 11 is the longest and the most delicate proof in this work. The reason is due to the combinatorial nature of the statement in (49) of Lemma 11. Specifically, since $|R| = n^k$ and $|R_2| = \frac{n!}{(n-k)!}$, there are $n^k - \frac{n!}{(n-k)!}$ number of $\mathbf{r} \in R \backslash R_2$ and one has to prove that none of those $\mathbf{r} \in R \backslash R_2$ can further lower the value of $\sum_{i=1}^{k} \min((d - z_i(\mathbf{r}))\beta, \alpha)$ in (49). Furthermore, different $\mathbf{r}$ vectors could lead to different orders between the two values $(d - z_i(\mathbf{r}))\beta$ and α, which, in turn, would change the minimum value of $\min((d - z_i(\mathbf{r}))\beta, \alpha)$. The order between $(d - z_i(\mathbf{r}))\beta$ and α needs to be carefully considered in the proof. Finally, Lemma 11 is applicable to arbitrary (n, k, d) values. Therefore, one has to devise a proof that is applicable to both the cases of "$n \mod (n-d) = 0$" and "$n \mod (n-d) \neq 0$," the latter condition means that the proof has to take care of the non-integrality of $\frac{n}{n-d}$.

We prove Lemma 11 by explicit construction. For any vector $\mathbf{r} \in R$, we will use the following subroutine, called MODIFY, to iteratively change $\mathbf{r}$ in four major steps. The end result of the subroutine MODIFY is a vector $\mathbf{r}' \in R_2$ that satisfies

$$\sum_{i=1}^{k} \min((d - z_i(\mathbf{r}))\beta, \alpha) \geq \sum_{i=1}^{k} \min((d - z_i(\mathbf{r}'))\beta, \alpha). \tag{50}$$

Since any $\mathbf{r} \in R$ can be used to find a $\mathbf{r}' \in R_2$ that satisfies (50), we thus have

$$\begin{aligned}&\min_{\mathbf{r}' \in R_2} \sum_{i=1}^{k} \min((d - z_i(\mathbf{r}'))\beta, \alpha) \\ &\leq \min_{\mathbf{r} \in R} \sum_{i=1}^{k} \min((d - z_i(\mathbf{r}))\beta, \alpha),\end{aligned}$$

which, together with (48), implies (49). The proof of Lemma 11 is thus complete.

In the following, we first describe the subroutine MODIFY and then prove that for any $\mathbf{r} \in R$ as input, the output of MODIFY is indeed an $\mathbf{r}' \in R_2$ satisfying (50). Due to the intricate definitions/operations of the subroutine MODIFY, a detailed example illustrating MODIFY is provided in Appendix H to complement the following algorithmic description of MODIFY. It is recommended to refer to the illustrative example when reading the description of MODIFY.

§ *The Description of* MODIFY*:*

Input: Any k-dimensional vector $\mathbf{r} \in R$.

Output: A k-dimensional vector $\mathbf{r}' \in R_2$ satisfying (50).

Step 1: If there are $i, j \in \{1, \cdots, k\}$ such that $i < j$ and the i-th and the j-th coordinates of $\mathbf{r}$ are equal, i.e., $r_i = r_j$, then we can do the following modification. For convenience, we denote the value of $r_i = r_j$ by h. Suppose that node h belongs to the Q-th family. We now check whether there is any value γ satisfying simultaneously (i) $\gamma \in \{1, 2, \cdots, n\} \backslash h$; (ii) node γ is also in the Q-th family; and (iii) γ is not equal to any of the coordinates of $\mathbf{r}$. If such γ exists, we replace the j-th coordinate of $\mathbf{r}$ by γ. Specifically, after this modification, we will have $r_i = h$ and $r_j = \gamma$.

Repeat this step until either there is no repeated $r_i = r_j$, or until no such γ can be found.

Step 2: After finishing Step 1, we perform the following modification. Recall that c_{last} is the index of the last complete family. If there exists a node $h \in \{1, 2, \cdots, n\}$ satisfying (i) node h belongs to a complete family, say the Q-th family for some $Q \in \{1, 2, \cdots, c_{\text{last}}\}$; (ii) there are distinct $i, j \in \{1, \cdots, k\}$ satisfying $r_i = r_j = h$ and $i < j$, then do the following.

Find the largest $j_1 \in \{1, \cdots, n\}$ such that node $r_{j_1} = h$ and find the largest $j_2 \in \{1, \cdots, n\}$ such that r_{j_2} belongs to the Q-th family (the same family of node h). If $j_1 = j_2$, then we set $\mathbf{r}' = \mathbf{r}$. If $j_1 \neq j_2$, then we swap the values of r_{j_1} and r_{j_2} to construct $\mathbf{r}'$. That is, we first set $\mathbf{r}' = \mathbf{r}$ for all coordinates except for the j_1-th and the j_2-th coordinates, and then set $r'_{j_1} = r_{j_2}$ and $r'_{j_2} = r_{j_1}$. After we have constructed new $\mathbf{r}'$, we now check whether there is any value $\gamma \in \{1, \cdots, n\}$ satisfying simultaneously (i) node γ belongs to a complete family (not necessarily the Q-th family); and (ii) γ is not equal to any of the coordinates of $\mathbf{r}'$. If such γ exists, we replace the j_2-th coordinate of $\mathbf{r}'$ by γ, i.e., set $r'_{j_2} = \gamma$.

We notice that (if such node h exists,) Step 2 generates a new vector $\mathbf{r}'$ from $\mathbf{r}$. If $\mathbf{r}' \neq \mathbf{r}$, then replace the values of $\mathbf{r}$ by $\mathbf{r}'$ and then repeat Step 2. If $\mathbf{r}' = \mathbf{r}$, then we move on to the next step.

After finishing the above two steps, the current vector $\mathbf{r}$ must be in one of the following cases. Case 1: No two coordinates are equal, i.e., $r_i \neq r_j$ for all pairs $i < j$; Case 2: there exist a pair $i < j$ such that $r_i = r_j$. We have three sub-cases for Case 2. Case 2.1: All such (i, j) pairs must satisfy that node r_i belongs to the incomplete family. Case 2.2: All such (i, j) pairs must satisfy that node r_i belongs to a complete family. Case 2.3: some (i, j) pair has $r_i = r_j$ belonging to a complete family and some other (i', j') pair has $r_{i'} = r_{j'}$ belonging to the incomplete family. The subroutine MODIFY then proceeds differently according to different cases.

If the $\mathbf{r}$ vector is in Case 1, then such $\mathbf{r}$ belongs to R_2 already and our construction is complete. The subroutine MODIFY simply returns $\mathbf{r}$. If $\mathbf{r}$ belongs to Case 2.1, then go to Step 3. If $\mathbf{r}$ belongs to Case 2.2, go to Step 4. It turns out that Case 2.3 is an impossible case. Namely, after executing Steps 1 and 2, the resulting $\mathbf{r}$ will never be in Case 2.3. We will provide a proof that Case 2.3 is impossible in our analysis of MODIFY.

Step 3: We use (i, j) to denote the pair of values such that $r_i = r_j$ and $i < j$. Denote the value of $r_i = r_j$ by h. Since we are in Case 2.1, node h belongs to the incomplete family. Find the largest $j_1 \in \{1, \cdots, n\}$ such that node $r_{j_1} = h$ and find the largest $j_2 \in \{1, \cdots, n\}$ such that r_{j_2} belongs to the incomplete family. If $j_1 = j_2$, then we set $\mathbf{r}' = \mathbf{r}$. If $j_1 \neq j_2$, then we swap the values of r_{j_1} and r_{j_2} to construct $\mathbf{r}'$. That is, we first set $\mathbf{r}' = \mathbf{r}$ for all coordinates except for the j_1-th and the j_2-th coordinates, and then set $r'_{j_1} = r_{j_2}$ and $r'_{j_2} = r_{j_1}$.

Recall that we use $c_{\text{last}} \triangleq \left\lfloor \frac{n}{n-d} \right\rfloor$ to denote the family index of the last complete family. We now choose arbitrarily a γ value from $\{(n-d)(c_{\text{last}} - 1) + 1, \ldots, (n-d)c_{\text{last}}\}$. Namely, γ is the index of a node of the last complete family. Fix the γ value. We then replace the original $\mathbf{r}$ vector by $\mathbf{r}'$ except for the j_2-th coordinate. I.e., we set $r_i = r'_i$ for all $i \neq j_2$. After that, we set $r_{j_2} = \gamma$.

If there exists $i \neq j_2$ such that $r'_i = \gamma$, then after setting $r_{j_2} = \gamma$ that particular i will lead to $r_i = r_{j_2} = \gamma$. In this case, we start the entire subroutine MODIFY from Step 1 by using the latest $\mathbf{r}$ as the new input. Namely, we recursively call the subroutine MODIFY and output MODIFY($\mathbf{r}$). If none of the coordinates[18] of $\mathbf{r}'$ has value γ, then one can easily see that after setting $r_{j_2} = \gamma$ there exists no $i < j$ satisfying "$r_i = r_j$ belong to a complete family" since we are in Case 2.1 to begin with. In this case, we are thus either in Case 1 or Case 2.1. If the new $\mathbf{r}$ is now in Case 1, then we stop the modification process and simply return the new $\mathbf{r}$ as the final output. If the new $\mathbf{r}$ is still in Case 2.1, we will then repeat Step 3.

[18]By our definition of j_2, node r'_{j_2} always belongs to the incomplete family and thus $r'_{j_2} \neq \gamma$ since by our construction node γ belongs to the last complete family.

Step 4: We use (i,j) to denote the pair of values such that $r_i = r_j$ and $i < j$. Denote the value of $r_i = r_j$ by h. Since we are in Case 2.2, node h belongs to a complete family. Suppose h is in the Q-th complete family. Find the largest $j_1 \in \{1,\cdots,n\}$ such that node $r_{j_1} = h$ and find the largest $j_2 \in \{1,\cdots,n\}$ such that r_{j_2} belongs to the Q-th complete family. If $j_1 = j_2$, then we set $\mathbf{r}' = \mathbf{r}$. If $j_1 \neq j_2$, then we swap the values of r_{j_1} and r_{j_2} to construct $\mathbf{r}'$. That is, we first set $\mathbf{r}' = \mathbf{r}$ for all coordinates except for the j_1-th and the j_2-th coordinates, and then set $r'_{j_1} = r_{j_2}$ and $r'_{j_2} = r_{j_1}$.

We now find a γ value such that (i) node γ belongs to the incomplete family; and (ii) γ is not equal to any of the coordinates of $\mathbf{r}'$. In our analysis of MODIFY, we will prove that such a γ always exists. Once the γ value is found, we replace the original $\mathbf{r}$ vector by $\mathbf{r}'$ except for the j_2-th coordinate. I.e., we set $r_i = r'_i$ for all $i \neq j_2$. After that, we set $r_{j_2} = \gamma$. If the new $\mathbf{r}$ is now in Case 1, then we stop the modification process and return $\mathbf{r}$. Otherwise, $\mathbf{r}$ must still be in Case 2.2 since we replace r_{j_2} by a γ that does not appear in $\mathbf{r}$ before. In this scenario, we will then repeat Step 4.

An example demonstrating the above iterative process is provided in Appendix H.

The analysis of MODIFY consists of four major parts. Firstly, we need to prove that the γ value specified in Step 4 always exists. (The γ value in Step 3 always exists but the γ values in Steps 1 and 2 may or may not exist.) Secondly, we need to prove that Case 2.3 is an impossible case. Thirdly, we need to prove that MODIFY terminates after a finite number of time, i.e., no infinite loop. Fourthly, if we use $\mathbf{r}$ to denote the input of MODIFY and use $\mathbf{r}'$ to denote the final output of MODIFY once it terminates, obviously we have $\mathbf{r}' \in R_2$ since $\mathbf{r}' \in R_2$ is the terminating condition of MODIFY. We still need to prove that $\mathbf{r}$ and $\mathbf{r}'$ satisfy (50). We provide the analysis of MODIFY in Appendix I.

APPENDIX H
SUPPLEMENTAL: AN ILLUSTRATIVE EXAMPLE FOR THE MODIFY PROCEDURE

For illustration, we apply the procedure MODIFY to the following example with $(n,d) = (8,5)$ and $k = 8$. Recall that family 1 contains nodes $\{1,2,3\}$, family 2 (last complete family) contains nodes $\{4,5,6\}$, and the incomplete family, family 0, contains nodes $\{7,8\}$. Suppose the initial $\mathbf{r}$ vector is $\mathbf{r} = (1,2,2,2,4,7,7,7)$. We will use MODIFY to convert $\mathbf{r}$ to a vector $\mathbf{r}' \in R_2$.

We first enter Step 1 of the procedure. We observe[19] that $r_3 = r_4 = 2$ ($i = 3$ and $j = 4$) and node 2 belongs to the first family. Since node 3 is also in family 1 and it is not present in $\mathbf{r}$, we can choose $\gamma = 3$. After replacing r_4 by 3, the resulting vector is $\mathbf{r} = (1,2,2,3,4,7,7,7)$. Next, we enter Step 1 for the second time. We observe that $r_7 = r_8 = 7$. Since node 8 is in family 0 and it is not present in $\mathbf{r}$, we can choose $\gamma = 8$. The resulting vector is $\mathbf{r} = (1,2,2,3,4,7,7,8)$. Next, we enter Step 1 for the third time. For the new $\mathbf{r}$, we have $r_2 = r_3 = 2$ and $r_6 = r_7 = 7$, but for both cases we cannot find the desired γ value. As a result, we cannot proceed any further by Step 1. For that reason, we enter Step 2.

We observe that for $r_2 = r_3 = 2$, we find $j_1 = 3$, the last coordinate of $\mathbf{r}$ equal to 2, and $j_2 = 4$, the last coordinate of $\mathbf{r}$ that belongs to family 1. By Step 2, we swap r_3 and r_4, and the resultant vector is $\mathbf{r} = (1,2,3,2,4,7,7,8)$. Now, since node 5 belongs to family 2, a complete family, and it is not present in $\mathbf{r}$, we can choose $\gamma = 5$. After replacing r_{j_2} by γ, the resultant vector is $\mathbf{r} = (1,2,3,5,4,7,7,8)$. Next, we enter Step 2 for the second time. Although $r_6 = r_7 = 7$, we notice that node 7 is in family 0. Therefore, we do nothing in Step 2.

After Step 2, the latest $\mathbf{r}$ vector is $\mathbf{r} = (1,2,3,5,4,7,7,8)$, which belongs to Case 2.1. Consequently, we enter Step 3. In Step 3, we observe that $j_1 = 7$, the last coordinate of $\mathbf{r}$ being 7, and $j_2 = 8$, the last coordinate of $\mathbf{r}$ that belongs to the incomplete family, family 0. Thus, we swap r_7 and r_8, and the resultant vector is $\mathbf{r} = (1,2,3,5,4,7,8,7)$. Now, we choose arbitrarily a γ value from $\{4,5,6\}$, the last complete family. Suppose we choose[20] $\gamma = 6$. The resultant vector is $\mathbf{r} = (1,2,3,5,4,7,8,6)$. Since we have no other repeated nodes of family 0, the procedure finishes at this point. Indeed, we can see that the final vector $\mathbf{r}' = (1,2,3,5,4,7,8,6) \in R_2$, which has no repeated nodes and is the result expected. One can also verify that $\mathbf{r}$ and $\mathbf{r}'$ satisfy (50).

We would also like to point out that $\mathbf{r} = (1,2,2,2,4,7,7,7)$ and $\mathbf{r}' = (1,2,3,5,4,7,8,6)$ are quite different. Namely, when searching for an $\mathbf{r}' \in R_2$ that satisfies (50), we often have to search over a wide scope. That is the reason behind such a long and delicate subroutine MODIFY.

APPENDIX I
THE ANALYSIS OF THE SUBROUTINE "MODIFY"

A. Part I

As discussed in Appendix G-D, we execute Step 4 of MODIFY if we are in "Case 2.2: All such (i,j) pairs must satisfy that node r_i belongs to a complete family." after executing Steps 1 and 2. Step 4 then finds a γ value such that (i) node γ belongs to the incomplete family; and (ii) γ is not equal to any of the coordinates of $\mathbf{r}$. We now prove that such a γ always exists.

To that end, we first argue that any node $\tilde{\gamma}$ that belongs to a complete family must appear in one of the coordinates of $\mathbf{r}$. The reason is that otherwise we would have continued executing Step 2 since we can replace one of the repeated node index by $\tilde{\gamma}$. Since every node in any complete family must appear in $\mathbf{r}$ at least once, there are at least $(n-d)\left\lfloor \frac{n}{n-d} \right\rfloor + 1$ number of coordinates of $\mathbf{r}$ referring to a node in one of the complete families. This in turn implies that there are at most $n - \left((n-d)\left\lfloor \frac{n}{n-d} \right\rfloor + 1\right) = (n \bmod (n-d)) - 1$ number of coordinates of $\mathbf{r}$ referring to a node in the incomplete family. Since there are $n \bmod (n-d)$ distinct nodes in the incomplete family, there must exist a γ value such that node γ belongs

[19]We also observe that $r_2 = r_3 = 2$ and we can choose $i = 2$ and $j = 3$ instead. Namely, the choice of (i,j) is not unique. In MODIFY, any choice satisfying our algorithmic description will work.

[20]We can also choose $\gamma = 4$ or 5. For those choices, the iterative process will continue a bit longer but will terminate eventually.

to the incomplete family and γ does not appear in any one of the coordinates of $\mathbf{r}$. The proof of Part I is complete.

B. Part II

As discussed in Appendix G-D, there are four cases after finishing Step 2: Cases 1, 2.1, 2.2, and 2.3, where Case 2.3 is "some (i, j) pair has $r_i = r_j$ belonging to a complete family and some other (i', j') pair has $r_{i'} = r_{j'}$ belonging to the incomplete family." We now prove that Case 2.3 is an impossible case.

The reason is as follows. Suppose some (i, j) pair has $r_i = r_j$ belonging to a complete family. Since we no longer can execute Step 2 anymore, by the same argument as in Part I in Appendix I-A, there must exist a γ value such that node γ belongs to the incomplete family and γ does not appear in any one of the coordinates of $\mathbf{r}$. However, the existence of such γ and the second half of Case 2.3, which says that there exists a pair $i' < j'$ satisfying $r_{i'} = r_{j'}$ and belonging to the incomplete family, imply that we can perform Step 1 and replace the value of $r_{j'}$ by γ. This contradicts the fact that we have exhausted Step 1 before moving on to Step 2. The proof of Part II is complete.

C. Part III

In this part of the analysis, we prove MODIFY terminates in a finite number of rounds. To that end, for any vector $\mathbf{r}$ we define a non-negative function $T(\mathbf{r})$ by

$$T(\mathbf{r}) = |\{(i,j) : i < j, r_i = r_j \text{ is a complete family node}\}| + 2|\{(i,j) : i < j, r_i = r_j \text{ is an incomplete family node}\}|. \quad (51)$$

Namely, $T(\mathbf{r})$ counts how many pairs of (i, j) that have the same $r_i = r_j$ being in a complete family, plus two times how many pairs of (i, j) that have the same $r_i = r_j$ being in the incomplete family.

We then argue that $T(\mathbf{r})$ strictly decreases whenever we successfully finish executing one of the four steps. As a result, the subroutine MODIFY terminates in a finite number of rounds since $T(\mathbf{r})$ is lower bounded by zero. The proof is thus complete. In the following, we examine the effect of Steps 1 to 4 on the value of $T(\mathbf{r})$.

If any of Steps 1, 2, and 4 is successfully executed, then r_j is replaced by a γ not equal to any coordinate of $\mathbf{r}$. As a result $T(\mathbf{r})$ decreases strictly.

If Step 3 is successfully executed, then the value of the j_2-th coordinate r_{j_2}, a node in the incomplete family, is replaced by a γ that is in the last complete family. Therefore, the second term of (51) decreases by at least two. On the other hand, since we are now in Case 2.1, there is at most one coordinate of $\mathbf{r}$ having value γ. The increase of the first term is thus at most one. The net effect of the first and the second terms of (51) thus strictly decreases the value of $T(\mathbf{r})$.

D. Part IV

For each step of MODIFY, we use $\mathbf{r}$ to denote the input (original) vector and $\mathbf{w}$ to denote the output (modified) vector. In what follows, we will prove that the $\mathbf{r}$ and $\mathbf{w}$ vectors always satisfy

$$\sum_{m=1}^{k} \min((d - z_m(\mathbf{w}))\beta, \alpha) \leq \sum_{m=1}^{k} \min((d - z_m(\mathbf{r}))\beta, \alpha). \quad (52)$$

Namely, each step always decreases the value of $\sum_{m=1}^{k} \min((d - z_m(\mathbf{r}))\beta, \alpha)$. Therefore, the initial input $\mathbf{r}$ and the final output $\mathbf{r}'$ of MODIFY must satisfy (50).

1) Analysis of Step 1: Suppose that we found such γ. Denote the vector after we replaced the j-th coordinate with γ by $\mathbf{w}$. We observe that for $1 \leq m \leq j-1$, we will have $z_m(\mathbf{r}) = z_m(\mathbf{w})$ since $r_m = w_m$ over $1 \leq m \leq j-1$. Also, we will have $z_j(\mathbf{r}) = z_j(\mathbf{w})$ since the new $w_j = \gamma$ belongs to the Q-th family, the same family as node r_j. For $j+1 \leq m \leq k$, we will have $z_m(\mathbf{w}) \geq z_m(\mathbf{r})$. The reason is that by our construction, we have $w_j = \gamma \neq r_j = r_i = w_i$. For any $m > j$, $z_m(\mathbf{r})$ only counts the repeated $r_i = r_j$ once. Therefore, $z_m(\mathbf{w})$ will count the same w_i as well. On the other hand, $z_m(\mathbf{w})$ may sometimes be larger than $z_m(\mathbf{r})$, depending on whether the new $w_j = \gamma$ belongs to D_{w_m} or not. The fact that $z_m(\mathbf{w}) \geq z_m(\mathbf{r})$ for all $m = 1$ to k implies (52).

2) Analysis of Step 2 — The Swapping Operation: Step 2 consists of two parts. Firstly, if $j_1 \neq j_2$ then we swap the values of r_{j_1} and r_{j_2}. Secondly, after the swap (if $j_1 \neq j_2$) then we replace the r_{j_2} value by γ. We now argue that both the swap operation and the replacement-by-γ will satisfy (52).

To prove that swapping the values of r_{j_1} and r_{j_2} still satisfies (52), we slightly abuse the notation and use $\mathbf{w}$ to denote the resulting vector after swapping the j_1-th and the j_2-th coordinates of $\mathbf{r}$ (but before replacing r_{j_2} by γ). For the case of $1 \leq m \leq j_1 - 1$, we have $z_m(\mathbf{w}) = z_m(\mathbf{r})$ since for $1 \leq m \leq j_1 - 1$, $r_m = w_m$. We also have $z_{j_1}(\mathbf{w}) = z_{j_1}(\mathbf{r})$ since both r_{j_1} and $w_{j_1} = r_{j_2}$ are from the same family Q.

For the case of $j_1 + 1 \leq m \leq j_2 - 1$, we have $z_m(\mathbf{w}) \geq z_m(\mathbf{r})$. The reason is as follows. We first observe that $w_{j_1} = r_{j_2} \neq r_{j_1} = r_i = w_i$ for some $i < j_1$. The reason is that by our construction $r_i = r_j = h$ and we set j_1 to be the largest coordinate that still satisfies $r_{j_1} = h$. This construction implies $j_1 \geq j > i$. For any $j_1 + 1 \leq m \leq j_2 - 1$, $z_m(\mathbf{r})$ only counts the repeated appearances $r_i = r_{j_1}$ once. Therefore, $z_m(\mathbf{w})$ will count the same w_i as well. On the other hand, $z_m(\mathbf{w})$ may sometimes be larger than $z_m(\mathbf{r})$, depending on whether the new $w_{j_1} = r_{j_2}$ belongs to D_{w_m} or not. We thus have $z_m(\mathbf{w}) \geq z_m(\mathbf{r})$ for $j_1 + 1 \leq m \leq j_2 - 1$.

For the case of $m = j_2$, we notice that $z_{j_2}(\mathbf{w})$ will not count w_{j_1} since $w_{j_1} = r_{j_2}$ and $w_{j_2} = r_{j_1}$ are both in the Q-th family. Similarly, $z_{j_2}(\mathbf{r})$ will not count r_{j_1} since r_{j_1} and r_{j_2} are both in the Q-th family. On the other hand, since both w_{j_2} and r_{j_2} are in the Q-th family and $w_l = r_l$ for all $l \in \{1, 2, \cdots, j_2 - 1\} \backslash \{j_1\}$, both $z_{j_2}(\mathbf{w})$ and $z_{j_2}(\mathbf{r})$ will count in the same way for the l-th coordinate for all $l \in \{1, 2, \cdots, j_2 - 1\} \backslash \{j_1\}$. As a result, we have $z_{j_2}(\mathbf{w}) = z_{j_2}(\mathbf{r})$.

For the case of $j_2 + 1 \leq m \leq k$, we argue that $z_m(\mathbf{w}) = z_m(\mathbf{r})$. This is true because of the definition of $z_m(\cdot)$ and the fact that both $j_1 < m$ and $j_2 < m$. In summary, we have proved $z_m(\mathbf{w}) \geq z_m(\mathbf{r})$ for $m = 1$ to k, which implies (52).

3) Analysis of Step 2 — The Replacement-By-γ Operation: We now consider the replacement-by-γ operation. We slightly abuse the notation and let $\mathbf{r}$ denote the vector after swapping (if necessary) and let $\mathbf{w}$ denote the final resulting vector after replacing r_{j_2} by γ.

We first notice that after swapping, the vector $\mathbf{r}$ always has $r_{j_2} = h$ and $\mathbf{w}$ is identical to $\mathbf{r}$ except that $w_{j_2} = \gamma$. For $1 \leq m \leq j_2 - 1$, $z_m(\mathbf{w}) = z_m(\mathbf{r})$ since $r_m = w_m$ over this range of m. We now consider the case of $m = j_2$. Suppose node γ belongs to the Q_γ-th family. Note that by our construction both $Q, Q_\gamma \in \{1, 2, \cdots, c_{\text{last}}\}$ are indices of a complete family.

We then notice that by the definition of $z_m(\cdot)$ and the definition of the family helper selection scheme, $z_{j_2}(\mathbf{w})$ corresponds to "the number of nodes *not* in the Q_γ-th family, but appear in one of the first $(j_2 - 1)$ coordinates of $\mathbf{w}$." Similarly, $z_{j_2}(\mathbf{r})$ corresponds to "the number of nodes *not* in the Q-th family, but appear in one of the first $(j_2 - 1)$ coordinates of $\mathbf{r}$'.' Since the $(j_2 - 1)$-th coordinates of $\mathbf{w}$ and $\mathbf{r}$ are identical, we thus have

$$z_{j_2}(\mathbf{w}) - z_{j_2}(\mathbf{r}) = \mathsf{term1} - \mathsf{term2}$$

where $\mathsf{term1}$ is "the number of distinct nodes in the Q-th family that appear in the first $(j_2 - 1)$ coordinates of $\mathbf{r}$" and $\mathsf{term2}$ is "the number of distinct nodes in the Q_γ-th family that appear in the first $(j_2 - 1)$ coordinates of $\mathbf{r}$."

Since we start Step 2 only after Step 1 cannot proceed any further, it implies that all distinct $(n - d)$ nodes of family Q must appear in $\mathbf{r}$ otherwise we should continue Step 1 rather than go to Step 2. Then by our specific construction of j_2, after swapping, all distinct $(n-d)$ nodes of family Q must appear in the first $(j_2 - 1)$-th coordinates of $\mathbf{r}$. Therefore $\mathsf{term1} = (n - d)$. Since there are exactly $(n - d)$ distinct nodes in the Q_γ-th family, by the definition of $\mathsf{term2}$, we must have $\mathsf{term2} \leq (n - d)$. The above arguments show that $\mathsf{term2} \leq \mathsf{term1} = (n-d)$, which implies the desired inequality $z_{j_2}(\mathbf{w}) - z_{j_2}(\mathbf{r}) \geq 0$.

We now consider the case when $m \geq j_2 + 1$. In this case, we still have $z_m(\mathbf{w}) \geq z_m(\mathbf{r})$. The reason is that by our construction, we have $w_{j_2} = \gamma \neq r_{j_2} = r_i = w_i$ for some $i < j_2$. For any $m > j_2$, $z_m(\mathbf{r})$ only counts the repeated $r_i = r_{j_2}$ once. Therefore, $z_m(\mathbf{w})$ will count the same w_i as well. On the other hand, $z_m(\mathbf{w})$ may sometimes be larger than $z_m(\mathbf{r})$, depending on whether the new $w_{j_2} = \gamma$ belongs to D_{w_m} or not. The fact that $z_m(\mathbf{w}) \geq z_m(\mathbf{r})$ for all $1 \leq m \leq k$ implies (52).

4) Analysis of Step 3: Step 3 consists of two parts. Firstly, if $j_1 \neq j_2$ then we swap the values of r_{j_1} and r_{j_2}. Secondly, after the swap (if $j_1 \neq j_2$) then we replace the r_{j_2} value by γ. By the verbatim argument as used in proving Step 2, one can prove that (52) still holds after the swapping operation and thus omit the corresponding proof. In the following, we focus on the replacement-by-γ operation and let $\mathbf{r}$ denote the vector after swapping (if necessary) and let $\mathbf{w}$ denote the final resulting vector after replacing r_{j_2} by γ.

Recall that $r_{j_2} = h$ belongs to the incomplete family and recall that $w_{j_2} = \gamma$ is a node from the last complete family. For $1 \leq m \leq j_2 - 1$, since we have $r_m = w_m$ for all $1 \leq m \leq j_2 - 1$, we must have $z_m(\mathbf{r}) = z_m(\mathbf{w})$. We now consider the case of $m = j_2$. By the definition of $z_m(\cdot)$ and the definition of the family helper selection scheme, we have

$$z_{j_2}(\mathbf{w}) - z_{j_2}(\mathbf{r}) = \mathsf{term1} - \mathsf{term2}$$

where $\mathsf{term1}$ is "the number of distinct nodes in the incomplete family that appear in the first $(j_2 - 1)$ coordinates of $\mathbf{w}$" and $\mathsf{term2}$ is "the number of distinct nodes in the last complete family that simultaneously (i) belong to the helper set of the incomplete family and (ii) appear in the first $(j_2 - 1)$ coordinates of $\mathbf{r}$." Also note that the first (j_2-1)-th coordinates of $\mathbf{w}$ and $\mathbf{r}$ are identical.

Since we have finished executing Step 1 and we are currently in Case 2.1, it means that all $n \bmod (n - d)$ nodes in the incomplete family appear in the vector $\mathbf{r}$. By our construction of j_1 and j_2, after swapping, all $n \bmod (n - d)$ nodes in the incomplete family must appear in the first $(j_2 - 1)$ coordinates of $\mathbf{r}$. Therefore, $\mathsf{term1} = n \bmod (n - d)$. Since there are exactly $n \bmod (n - d)$ distinct nodes in the last complete family that belong to the helper set of the incomplete family, by the definition of $\mathsf{term2}$, we must have $\mathsf{term2} \leq n \bmod (n - d)$. The above arguments show that $\mathsf{term2} \leq \mathsf{term1} = n \bmod (n - d)$, which implies the desired inequality $z_{j_2}(\mathbf{w}) - z_{j_2}(\mathbf{r}) \geq 0$.

For the case of $j_2 + 1 \leq m$, we also have $z_m(\mathbf{w}) \geq z_m(\mathbf{r})$. The reason is that by our construction, we have $w_{j_2} = \gamma \neq r_{j_2} = r_i = w_i$ for some $i < j_2$. For any $m \geq j_2 + 1$, $z_m(\mathbf{r})$ only counts the repeated $r_i = r_{j_2}$ once. Therefore, $z_m(\mathbf{w})$ will count the same w_i as well. On the other hand, $z_m(\mathbf{w})$ may sometimes be larger than $z_m(\mathbf{r})$, depending on whether the new $w_{j_2} = \gamma$ belongs to D_{w_m} or not. We have thus proved that $z_m(\mathbf{w}) \geq z_m(\mathbf{r})$ for all $m = 1$ to k, which implies (52).

5) Analysis of Step 4: Step 4 again consists of the swapping operation and the replacement-by-γ operation. The proof for the swapping operation is identical as that of Steps 2 and 3 and is thus omitted. We now consider the replacement-by-γ operation.

Specifically, $r_{j_2} = h$, which belongs to the Q-th complete family, is replaced with γ, a node of the incomplete family. For $1 \leq m \leq j_2 - 1$, $z_m(\mathbf{w}) = z_m(\mathbf{r})$ since $w_m = r_m$ over this range of m. For $m = j_2$, we have to consider two subcases. Subcase 1: If the Q-th family is the last complete family, then

$$z_{j_2}(\mathbf{w}) - z_{j_2}(\mathbf{r}) = \mathsf{term1} - \mathsf{term2}$$

where $\mathsf{term1}$ is the number of distinct nodes in the Q-th family that simultaneously (i) belong to the helper set of the incomplete family and (ii) appear in the first $(j_2 - 1)$ coordinates of $\mathbf{w}$, and $\mathbf{term2}$ is the number of distinct nodes in the incomplete family that appear in the first $(j_2 - 1)$ coordinates of $\mathbf{r}$. Also note that the first (j_2-1)-th coordinates of $\mathbf{w}$ and $\mathbf{r}$ are identical.

Since we have finished executing Step 1 and are currently in Case 2.2, it means that all $(n-d)$ nodes in the Q-th family must appear in the first $(j_2 - 1)$ coordinates of $\mathbf{r}$. Therefore, the value of $\mathsf{term1}$ is $n \bmod (n-d)$, since only $n \bmod (n-d)$ nodes (out of $n - d$ nodes) of the last complete family are in the helper set of the incomplete family. Since there are exactly $n \bmod (n-d)$ distinct nodes in the incomplete family, by the definition of $\mathsf{term2}$, we must have $\mathsf{term2} \leq n \bmod (n - d)$.

The above arguments show that $\mathsf{term2} \leq \mathsf{term1} = n \bmod (n-d)$, which implies the desired inequality $z_{j_2}(\mathbf{w}) - z_{j_2}(\mathbf{r}) \geq 0$.

Subcase 2: If the Q-th family is not the last complete family, then

$$z_{j_2}(\mathbf{w}) - z_{j_2}(\mathbf{r}) = \mathsf{term1} - (\mathsf{term2.1} + \mathsf{term2.2})$$

where $\mathsf{term1}$ is the number of distinct nodes in the Q-th family that appear in the first $(j_2 - 1)$ coordinates of $\mathbf{w}$; $\mathsf{term2.1}$ is the number of distinct nodes in the last complete family that simultaneously (i) do not belong to the helper set of the incomplete family and (ii) appear in the first $(j_2 - 1)$ coordinates of $\mathbf{r}$; $\mathsf{term2.2}$ is the number of distinct nodes in the incomplete family that appear in the first $(j_2 - 1)$ coordinates of $\mathbf{r}$. Also note that the first $(j_2 - 1)$-th coordinates of $\mathbf{w}$ and $\mathbf{r}$ are identical.

Since we have finished executing Step 1 and are currently in Case 2.2, it means that all $(n-d)$ nodes in the Q-th family must appear in the first (j_2-1) coordinates of $\mathbf{r}$. Therefore, the value of $\mathsf{term1}$ is $(n-d)$. Since there are exactly $n - (n \bmod (n-d))$ distinct nodes in the last complete family that do not belong to the helper set of the incomplete family, we must have $\mathsf{term2.1} \leq n - (n \bmod (n-d))$. Since there are exactly $n \bmod (n-d)$ distinct nodes in the incomplete family, we must have $\mathsf{term2.2} \leq n \bmod (n-d)$. The above arguments show that $\mathsf{term2.1}+\mathsf{term2.2} \leq \mathsf{term1} = (n-d)$, which implies the desired inequality $z_{j_2}(\mathbf{w}) - z_{j_2}(\mathbf{r}) \geq 0$.

For $j_2 + 1 \leq m \leq k$, we note that $r_{j_2} = h = r_i = w_i$ for some $i < j_2$. For any $m \geq j_2 + 1$, $z_m(\mathbf{r})$ only counts the repeated $r_i = r_{j_2}$ once while $z_m(\mathbf{w})$ will count the same w_i as well. On the other hand, $z_m(\mathbf{w})$ may sometimes be larger than $z_m(\mathbf{r})$, depending on whether the new $w_{j_2} = \gamma$ belongs to D_{w_m} or not. We have thus proved that $z_m(\mathbf{w}) \geq z_m(\mathbf{r})$ for all $m = 1$ to k, which implies (52).

Appendix J
Proof of Proposition 6

To characterize the MBR point of the family helper selection scheme, we simply set $\alpha = d\beta$. Then by the min-cut characterization in Proposition 5, we have

$$\beta_{\text{MBR}} = \frac{\mathcal{M}}{\min_{\pi_f} \sum_{i=1}^{k} (d - y_i(\pi_f))}.$$

In the following, we prove that the RFIP π_f^* minimizes

$$\sum_{i=1}^{k} (d - y_i(\pi_f)), \tag{53}$$

which thus completes the proof of Proposition 6.

To that end, we define

$$y_{\text{offset}}(\pi_f) = \sum_{i=1}^{k} (i - 1 - y_i(\pi_f)). \tag{54}$$

Notice that a family index permutation that minimizes $y_{\text{offset}}(\cdot)$ also minimizes (53). Therefore, any minimizing family index permutation for (53), call it $\pi_f^{\min}$, must satisfy

$$y_{\text{offset}}(\pi_f^{\min}) = \min_{\forall \pi_f} y_{\text{offset}}(\pi_f). \tag{55}$$

Consider the following two cases:

Case 1: $n \bmod (n-d) = 0$, i.e., we do not have an incomplete family.

Consider any family index permutation π_f and let l_m be the number of the first k coordinates of π_f that have value m. Suppose the i-th coordinate of π_f is m. Then, we notice that the expression "$(i-1) - y_i(\pi_f)$" counts the number of appearances of the value m in the first $i-1$ coordinates of π_f (recall that there is no incomplete family in this case). If we use $[\pi_f]_i$ to denote the i-th coordinate of π_f, then for any $m \in \left\{1, \cdots, \frac{n}{n-d}\right\}$ we have

$$\sum_{i:1\leq i\leq k, [\pi_f]_i = m} ((i-1) - y_i(\pi_f)) = \sum_{j=1}^{l_m} (j-1) \tag{56}$$

where the index j is used to count through the j-th time when $[\pi_f]_i = m$ is satisfied. By summing over all possible m values, (56) implies

$$y_{\text{offset}}(\pi_f) = \sum_{i=1}^{l_1} (i-1) + \sum_{i=1}^{l_2} (i-1) + \cdots + \sum_{i=1}^{l_{\frac{n}{n-d}}} (i-1). \tag{57}$$

We now prove the following claim.

Claim 1: The above equation implies that a family index permutation is a minimizing permutation $\pi_f^{\min}$ if and only if

$$|l_i - l_j| \leq 1 \text{ for all } i, j \text{ satisfying } 1 \leq i, j \leq \frac{n}{n-d}. \tag{58}$$

Proof: We first prove the only if direction by contradiction. The reason is as follows. If $l_i > l_j + 1$ for some $1 \leq i, j \leq \frac{n}{n-d}$, then we consider another family permutation π'_f and denote its corresponding l values by l', such that $l'_i = l_i - 1$, $l'_j = l_j + 1$, and all other ls remain the same. Clearly from (57), such π'_f will result in strictly smaller $y_{\text{offset}}(\pi'_f) < y_{\text{offset}}(\pi_f)$. Note that such π'_f with the new $l'_i = l_i - 1$, $l'_j = l_j + 1$ always exists. The reason is the following. By the definition of l_j and the fact that π_f is a family index permutation, we have $0 \leq l_j \leq (n-d)$ for all $j = 1, \cdots, \frac{n}{n-d}$. The inequality $l_i > l_j + 1$ then implies $l_i > 1$ and $l_j < (n-d) - 1$. Therefore, out of the first k coordinates of π_f, at least one of them will have value i; and out of the last $(n-k)$ coordinates of π_f, at least one of them will have value j. We can thus swap arbitrarily one of the family indices i from the first k coordinates with another family index j from the last $(n-k)$ coordinates and the resulting π'_f will have the desired l'_i and l'_j.

We now prove the if direction, i.e., any π_f satisfying (58) is a minimizing permutation. Consider any π_f satisfying (58) and the corresponding $\left\{l_m : m = 1, \cdots, \frac{n}{n-d}\right\}$. We then use π'_f to denote one permutation that minimizes (55), which may be different from π_f and denote the corresponding l variables by $\left\{l'_m : m = 1, \cdots, \frac{n}{n-d}\right\}$. By the only-if direction we have just proven, $\left\{l'_m : m = 1, \cdots, \frac{n}{n-d}\right\}$ must satisfy (58) as well.

We then observe the following facts. Fact 1: $\sum_{m=1}^{\frac{n}{n-d}} l_m = k$ by our construction of $\{l_m\}$. Fact 2: If $\frac{n}{n-d} = 3$, $k = 5$, l_1 to l_3 satisfy (58), and the summation $l_1 + l_2 + l_3 = k =$

5, then among l_1, l_2, and l_3, two of them must be 2 and the other one must be 1. Namely, the *distribution* of $\{l_m\}$ is uniquely determined by the two conditions $\sum_{m=1}^{\frac{n}{n-d}} l_m = k$ and (58). Since both $\{l_m\}$ and $\{l'_m\}$ satisfy $\sum_{m=1}^{\frac{n}{n-d}} l_m = k$ and (58), both $\{l_m\}$ and $\{l'_m\}$ must have the same *distribution* (or equivalently the histograms $\{l_m\}$ and $\{l'_m\}$ must be identical).

On the other hand, we observe that the value of $y_{\text{offset}}(\cdot)$ depends only on the distribution of $\{l_i\}$, see (57). As a result, $y_{\text{offset}}(\pi_f) = y_{\text{offset}}(\pi'_f)$. Therefore, any π_f satisfying (58) is a minimizing $\pi_f^{\min}$.

■

Finally, by the construction of the RFIP π_f^*, it is easy to verify that the RFIP π_f^* satisfies (58). Therefore, the RFIP π_f^* is a minimizing permutation for this case.

Case 2: $n \bmod (n-d) \neq 0$, i.e., when we do have an incomplete family. In this case, we are again interested in minimizing (53), and equivalently minimizing (54). Unfortunately, the proof is much longer as we need to handle the non-integrality of $\frac{n}{n-d}$ even though the spirit of the proof is the same.

To that end, we first prove the following claim.

Claim 2: Recall that $c_{\text{last}} = \left\lfloor \frac{n}{n-d} \right\rfloor$ is the index of the last complete family. Find the largest $1 \leq j_1 \leq k$ such that the j_1-th coordinate of π_f is 0. If no such j_1 can be found, we set $j_1 = 0$. Find the smallest $1 \leq j_2 \leq k$ such that the j_2-th coordinate of π_f is $-c_{\text{last}}$. If no such j_2 can be found, we set $j_2 = k+1$. We claim that if we construct j_1 and j_2 based on a π_f that minimizes $\sum_{i=1}^{k}(d - y_i(\pi_f))$, we must have $j_1 < j_2$.

Proof: We prove this claim by contradiction. Consider a minimizing family index permutation π_f and assume $j_2 < j_1$. Per our construction, we quickly have $1 \leq j_2 < j_1 \leq k$. We then swap the j_2-th coordinate and the j_1-th coordinate of π_f, and call the new family index permutation π'_f. Specifically, π'_f has the same values as π_f on all its coordinates except at the j_2-th coordinate it has the value 0 and at the j_1-th coordinate it has the value $-c_{\text{last}}$. We will later prove that

$$\sum_{i=1}^{k}(d - y_i(\pi_f)) - \sum_{i=1}^{k}(d - y_i(\pi'_f)) > 0,$$

which contradicts the assumption that π_f is a minimizing family index permutation.

To that end, we first compare the values of $y_m(\pi'_f)$ and $y_m(\pi_f)$ for all $m \in [1,k]$. We observe that for all $1 \leq m \leq j_2 - 1$ we have that $y_m(\pi'_f) = y_m(\pi_f)$ since the first $j_2 - 1$ coordinates of the two family index permutations are equal.

Now consider the case of $m = j_2$. Since the j_2-th coordinate of π_f, i.e., $[\pi_f]_{j_2}$, is $-c_{\text{last}}$, the function $y_{j_2}(\pi_f)$ counts how many of the first $(j_2 - 1)$ coordinates of π_f have values in $\{1, 2, \cdots, c_{\text{last}} - 1, 0\}$. Equivalently, $y_{j_2}(\pi_f)$ counts how many of the first $(j_2 - 1)$-th coordinates of π_f have values being neither c_{last} nor $-c_{\text{last}}$. However, by our definition of j_2, none of the first $(j_2 - 1)$ coordinates has value being $-c_{\text{last}}$. We thus have

$$y_{j_2}(\pi_f) = j_2 - 1 - \lambda_{\{c_{\text{last}}\}}^{[1,j_2)}, \tag{59}$$

where $\lambda_{\{c_{\text{last}}\}}^{[1,j_2)}$ counts how many of the coordinates of π_f in the range of $[1, j_2)$ (i.e., the first $(j_2 - 1)$ coordinates) have values being c_{last}.

Similarly, since $[\pi'_f]_{j_2} = 0$, the function $y_{j_2}(\pi'_f)$ counts how many of the first $(j_2 - 1)$ coordinates of π'_f have values in $\{1, 2, \cdots, c_{\text{last}}\}$. Equivalently, $y_{j_2}(\pi'_f)$ counts how many of the first $(j_2 - 1)$-th coordinates of π'_f have values being neither $-c_{\text{last}}$ nor 0. However, by our definition of j_2, none of the first $(j_2 - 1)$ coordinates of π'_f has value being $-c_{\text{last}}$. We thus have

$$y_{j_2}(\pi'_f) = j_2 - 1 - \phi_{\{0\}}^{[1,j_2)}, \tag{60}$$

where $\phi_{\{0\}}^{[1,j_2)}$ counts how many of the coordinates of π'_f in the range of $[1, j_2)$ have values being 0. Note that since we are comparing π_f and π'_f, we use different counting notation λ and ϕ, respectively, for different permutations of interest.

For the case of $j_2 + 1 \leq m \leq j_1 - 1$, if $[\pi_f]_m \in \{-c_{\text{last}}, c_{\text{last}}\}$, then $y_m(\pi'_f) = y_m(\pi_f) + 1$; otherwise, $y_m(\pi'_f) = y_m(\pi_f)$. The reason behind this is the following. We have $[\pi_f]_{j_2} = -c_{\text{last}}$ and $[\pi'_f]_{j_2} = 0$. Therefore, the new function $y_m(\pi'_f)$ behaves in an identical way as the original function $y_m(\pi_f)$ if $[\pi_f]_m \notin \{-c_{\text{last}}, c_{\text{last}}\}$. To see this, consider the case of $[\pi_f]_m \in \{1, 2, \cdots, c_{\text{last}} - 1\}$, then both $y_m(\pi'_f)$ and $y_m(\pi_f)$ will count their corresponding j_2-th coordinate. Similarly, if $[\pi_f]_m = 0$, then neither $y_m(\pi'_f)$ nor $y_m(\pi_f)$ will count their corresponding j_2-th coordinate. Only when $[\pi_f]_m \in \{-c_{\text{last}}, c_{\text{last}}\}$ will $y_m(\pi'_f)$ and $y_m(\pi_f)$ behave differently. Specifically, $y_m(\pi'_f)$ will count its j_2-th coordinate since $[\pi'_f]_{j_2} = 0$ but $y_m(\pi_f)$ will not count its j_2-th coordinate since $[\pi'_f]_{j_2} = -c_{\text{last}}$.

For the case of $m = j_1$, since $[\pi_f]_{j_1} = 0$, we have that $y_{j_1}(\pi_f)$ counts how many of the first $(j_1 - 1)$ coordinates of π_f have values in $\{1, 2, \cdots, c_{\text{last}}\}$. Or equivalently, $y_{j_1}(\pi_f)$ counts how many of the first $(j_1 - 1)$ coordinates of π_f have values being neither $-c_{\text{last}}$ nor 0. Thus, we have that

$$y_{j_1}(\pi_f) = j_1 - 1 - \lambda_{\{-c_{\text{last}},0\}}^{[1,j_1)} \tag{61}$$

where $\lambda_{\{-c_{\text{last}},0\}}^{[1,j_1)}$ counts how many of the coordinates of π_f in the range of $[1, j_1)$ have values being one of the two values $\{-c_{\text{last}}, 0\}$.

Since $[\pi'_f]_{j_1} = -c_{\text{last}}$, we have that $y_{j_1}(\pi'_f)$ counts how many of the first $(j_1 - 1)$ coordinates of π'_f have values being neither $-c_{\text{last}}$ nor c_{last}. We thus have

$$y_{j_1}(\pi'_f) = j_1 - 1 - \phi_{\{c_{\text{last}}\}}^{[1,j_2)} - \phi_{\{-c_{\text{last}},c_{\text{last}}\}}^{(j_2,j_1)}, \tag{62}$$

where $\phi_{\{c_{\text{last}}\}}^{[1,j_2)} = \phi_{\{-c_{\text{last}},c_{\text{last}}\}}^{[1,j_2)}$ counts how many of the coordinates of π'_f in the range of $[1, j_2)$ have values being c_{last} since per our construction, none of them can be of value $-c_{\text{last}}$; $[\pi'_f]_{j_2} = 0$ will not be counted; and $\phi_{\{-c_{\text{last}},c_{\text{last}}\}}^{(j_2,j_1)}$ counts how many of the coordinates of π'_f in the range of (j_2, j_1) have values being one of $\{-c_{\text{last}}, c_{\text{last}}\}$.

Finally, for the case of $j_1 + 1 \leq m \leq k$, we have that $y_m(\pi'_f) = y_m(\pi_f)$ since the order of the values preceding the m-th coordinate in a permutation does not matter for $y_m(\cdot)$.

From the above discussion of the $y_m(\pi'_f)$ and $y_m(\pi_f)$, we can now compute the following difference

$$\begin{aligned}
&\sum_{m=1}^{k}(d - y_m(\pi_f)) - \sum_{m=1}^{k}(d - y_m(\pi'_f)) \\
&\quad = \sum_{m=1}^{k}(y_m(\pi'_f) - y_m(\pi_f)) \\
&\quad = \sum_{m=j_2}^{j_1}(y_m(\pi'_f) - y_m(\pi_f)) && (63) \\
&\quad = (y_{j_2}(\pi'_f) - y_{j_2}(\pi_f)) + \phi^{(j_2,j_1)}_{\{c_{\text{last}},-c_{\text{last}}\}} \\
&\qquad + (y_{j_1}(\pi'_f) - y_{j_1}(\pi_f)) && (64) \\
&\quad = \left(\lambda^{[1,j_2)}_{\{c_{\text{last}}\}} - \phi^{[1,j_2)}_{\{0\}}\right) + \phi^{(j_2,j_1)}_{\{c_{\text{last}},-c_{\text{last}}\}} \\
&\qquad + \left(\lambda^{[1,j_1)}_{\{0,-c_{\text{last}}\}} - \phi^{[1,j_2)}_{\{c_{\text{last}}\}} - \phi^{(j_2,j_1)}_{\{c_{\text{last}},-c_{\text{last}}\}}\right) && (65) \\
&\quad = \lambda^{[1,j_1)}_{\{0,-c_{\text{last}}\}} - \phi^{[1,j_2)}_{\{0\}} && (66) \\
&\quad > 0, && (67)
\end{aligned}$$

where (63) follows from $y_i(\pi'_f) = y_i(\pi_f)$ for all $i < j_2$ and for all $i > j_1$; (64) follows from our analysis about $y_i(\pi'_f) = y_i(\pi_f) + 1$ when the i-th coordinate of π_f belongs to $\{-c_{\text{last}}, c_{\text{last}}\}$ and $y_i(\pi'_f) = y_i(\pi_f)$ otherwise, and there are thus $\phi^{(j_2,j_1)}_{\{c_{\text{last}},-c_{\text{last}}\}}$ coordinates between the $(j_2 + 1)$-th coordinate and the $(j_1 - 1)$-th coordinate of π'_f that satisfy $y_i(\pi'_f) = y_i(\pi_f) + 1$; (65) follows from (59) to (62); (66) follows from that $\phi^{[1,j_2)}_{\{c_{\text{last}}\}} = \lambda^{[1,j_2)}_{\{c_{\text{last}}\}}$ since the first $(j_2 - 1)$ coordinates of π'_f and π_f are identical; and (67) follows from the facts that $\lambda^{[1,j_1)}_{\{0\}} \geq \lambda^{[1,j_2)}_{\{0\}} = \phi^{[1,j_2)}_{\{0\}}$ and that $\lambda^{[1,j_1)}_{\{-c_{\text{last}}\}} \geq 1$ since $[\pi_f]_{j_2} = -c_{\text{last}}$. By (67), we have that π'_f has a strictly smaller "$\sum_{i=1}^{k}(d - y_i(\cdot))$". As a result, the case of $j_1 > j_2$ is impossible.

By the construction of j_1 and j_2, it is obvious that $j_1 \neq j_2$. Hence, we must have $j_1 < j_2$. The proof of this claim is complete. ■

Claim 2 provides a necessary condition on a minimizing permutation vector. We thus only need to consider permutations for which $j_1 < j_2$. That is, instead of taking the minimum over all π_f, we now take the minimum over only those π_f satisfying $j_1 < j_2$.

This observation is critical to our following derivation. The reason is that if we consider a permutation π_f that has $1 \leq j_2 < j_1 \leq k$, then the expression "$(j_1 - 1) - y_{j_1}(\pi_f)$" is not equal to the number of appearances of the value 0 in the first $j_1 - 1$ coordinates of π_f (recall that $[\pi_f]_{j_1} = 0$). Instead, by the definition of $y_i(\cdot)$, $(j_1 - 1) - y_{j_1}(\pi_f)$ is the number of appearances of the values 0 *and* $-c_{\text{last}}$ in the first $(j_1 - 1)$ coordinates of π_f. Therefore, we cannot rewrite (54) as (57) if $1 \leq j_2 < j_1 \leq k$.

On the other hand, Claim 2 implies that we only need to consider those π_f satisfying $j_1 < j_2$. We now argue that given any π_f satisfying $j_1 < j_2$, for all $i = 1$ to k, the expression $(i-1)-y_i(\pi_f)$ is now representing the number of appearances of m and $-m$ in the first $(i-1)$ coordinates of π_f, where m is the *absolute value* of the i-th coordinate of π_f. The reason is as follows. Let m denote the absolute value of the i-th coordinate of π_f. If $m \neq 0$, then by the definition of $y_i(\pi_f)$, we have that $(i - 1) - y_i(\pi_f)$ represents the number of appearances of m in the first $(i - 1)$ coordinates of π_f. If $m = 0$, then by the definition of $y_i(\pi_f)$, we have that $(i - 1) - y_i(\pi_f)$ represents the number of appearances of 0 and $-c_{\text{last}}$ in the first $(i - 1)$ coordinates of π_f.

However, since $[\pi_f]_i = 0$, we have $i \leq j_1$ by the construction of j_1. Since $j_1 < j_2$, we have $i < j_2$. This implies that in the first $(i - 1)$ coordinates of π_f, none of them is of value $-c_{\text{last}}$. As a result, we have that $(i - 1) - y_i(\pi_f)$ again represents the number of appearances of 0 in the first $(i - 1)$ coordinates of π_f.

We now proceed with our analysis while only considering those π_f satisfying $j_1 < j_2$ as constructed in Claim 2. Let l_j be the number of the first k coordinates of π_f that have values j or $-j$. We can then rewrite (54) by

$$\begin{aligned}
y_{\text{offset}}(\pi_f) = \sum_{i=1}^{l_0}(i-1) + \sum_{i=1}^{l_1}(i-1) + \\
\sum_{i=1}^{l_2}(i-1) + \cdots + \sum_{i=1}^{l_{\lfloor \frac{n}{n-d} \rfloor}}(i-1). \quad (68)
\end{aligned}$$

The above equation implies that for any family index permutation π_f satisfying $j_1 < j_2$, it is a minimizing permutation $\pi_f^{\min}$ if and only if either

$$\begin{cases}
l_0 = n \bmod (n-d), \\
|l_i - l_j| \leq 1 \text{ for all } i, j \text{ satisfying } 1 \leq i, j \leq c_{\text{last}}, \\
l_i \geq l_0 \text{ for all } i \text{ satisfying } 1 \leq i \leq c_{\text{last}}.
\end{cases} \quad (69)$$

or

$$|l_i - l_j| \leq 1, \text{for all } i, j \text{ satisfying } 0 \leq i, j \leq c_{\text{last}}. \quad (70)$$

If we compare (69) and (70) with (58) in Claim 1, we can see that (70) is similar to (58). The reason we need to consider the situation described in (69) is that the range of l_0 is from 0 to $n \bmod (n - d)$ while the range of all other l_is is from 0 to $(n - d)$. Therefore, we may not be able to make l_0 as close to other l_is (within a distance of 1) as we would have hoped for due to this range discrepancy. For some cases, the largest l_0 we can choose is $n \bmod (n-d)$, which gives us the first scenario when all the remaining l_is are no less than this largest possible l_0 value. If l_0 can also be made as close to the rest of l_is, then we have the second scenario.

The proof that (69) and (70) are the if-and-only-if condition on $\pi_f^{\min}$ can be completed using the same arguments as in the proof of Claim 1. Finally, notice that the RFIP π_f^* satisfies (69) or (70) and has $j_1 < j_2$. As a result, π_f^* must be one of the minimizing permutations $\pi_f^{\min}$. The proof of Proposition 6 is hence complete.

Appendix K
Supplemental: Byproducts of Propositions 3 and 4

Some byproducts of our results are that they can be used to prove whether some existing bounds are tight or not.

Existing Bound #1: [11, Theorem 5.4] proved that when $k = n - 1$ and $\alpha = \beta$, there exists an upper bound $\mathcal{M}_{\text{UB},1} \triangleq \frac{nd\alpha}{d+1}$ such that for any dynamic helper selection scheme $H \in \mathcal{DHS}$, the largest file size it can protect must satisfy $\mathcal{M} \leq \mathcal{M}_{\text{UB},1}$. It was not clear whether such a bound is tight or not. More precisely, for any given (n, k, d, α, β) value let $\mathcal{M}^* = \max_{H \in \mathcal{DHS}} \mathcal{M}(H)$ where $\mathcal{M}(H)$ is the largest file size that can be protected under a helper selection scheme H. The question to be answered is whether there exists an (n, k, d, α, β) value satisfying $k = n - 1$ and $\alpha = \beta$ such that $\mathcal{M}^* < \mathcal{M}_{\text{UB},1}$ with strict inequality. The following corollary answers this question affirmatively.

Corollary 4: When $(n, k, d) = (4, 3, 2)$ and $\alpha = \beta$, we have $\mathcal{M}^* = 2\alpha < \mathcal{M}_{\text{UB},1} = \frac{nd\alpha}{d+1} = \frac{8\alpha}{3}$. Therefore, the upper bound $\mathcal{M}_{\text{UB},1}$ in [11, Theorem 5.4] is loose.

Proof: By Proposition 5, when $(n, k, d) = (4, 3, 2)$ and $\alpha = \beta$, the family helper selection scheme can protect a file of size $\mathcal{M} = 2\alpha$. We then notice that $(n, k, d) = (4, 3, 2)$ satisfies Proposition 3 and, therefore, the family helper selection scheme is absolutely optimal and $\mathcal{M}^* = 2\alpha$. The proof is complete. ■

Existing Bound #2: [11, Theorem 5.2] proved that for $k = n - 1$ and $\alpha = d\beta$, there exists an upper bound $\mathcal{M}_{\text{UB},2} \triangleq \frac{nd\beta}{2}$ such that for any dynamic helper selection scheme $H \in \mathcal{DHS}$, the largest file size it can protect must satisfy $\mathcal{M} \leq \mathcal{M}_{\text{UB},2}$. It was not clear whether such an upper bound is tight or not. The question to be answered is whether there exists an (n, k, d, α, β) value satisfying $k = n - 1$ and $\alpha = d\beta$ such that $\mathcal{M}^* = \mathcal{M}_{\text{UB},2}$, where $\mathcal{M}^* = \max_{H \in \mathcal{DHS}} \mathcal{M}(H)$. The following corollary answers this question affirmatively.

Corollary 5: For any (n, k, d) values satisfying $k = n - 1$ and the conditions in Proposition 4, we have $\mathcal{M}^* = \frac{nd\beta}{2} = \mathcal{M}_{\text{UB},2}$. Therefore, the upper bound $\mathcal{M}_{\text{UB},2}$ in [11, Theorem 5.2] is tight for some (n, k, d) values.

This corollary is a direct byproduct of the proof of Proposition 4.

References

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[2] I. Ahmad and C.-C. Wang, "Locally repairable regenerating codes: Node unavailability and the insufficiency of stationary local repair," *[Online]. Available: arXiv:1505.05124 [cs.IT].*

[3] ——, "When can helper node selection improve regenerating codes? Part II: An explicit exact-repair code construction," *[Online]. Available: arXiv:1604.08230 [cs.IT].*

[4] R. Bhagwan, K. Tati, Y. C. Cheng, S. Savage, and G. M. Voelker, "Total recall: System support for automated availability management," in *Proc. 1st Conf. on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, Mar. 2004, pp. 25–25.

[5] V. R. Cadambe, S. A. Jafar, H. Maleki, K. Ramchandran, and C. Suh, "Asymptotic interference alignment for optimal repair of mds codes in distributed storage," *IEEE Trans. Inf. Theory*, vol. 59, no. 5, pp. 2974–2987, 2013.

[6] V. R. Cadambe and A. Mazumdar, "Bounds on the size of locally recoverable codes," *IEEE Trans. Inf. Theory*, vol. 61, no. 11, pp. 5787–5794, 2015.

[7] 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, 2010.

[8] S. El Rouayheb and K. Ramchandran, "Fractional repetition codes for repair in distributed storage systems," in *Proc. 48th Annual Allerton Conf. on Comm., Contr., and Computing.*, Monticello, IL, Sep. 2010, pp. 1510–1517.

[9] S. Ghemawat, H. Gobioff, and S. T. Leung, "The google file system," in *Proc. 19th ACM Symp. on Operating Systems Principles (SOSP)*, Bolton Landing, NY, Oct. 2003, pp. 29–43.

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

[11] H. D. L. Hollmann, "On the minimum storage overhead of distributed storage codes with a given repair locality," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Honolulu, HI, Jun. 2014, pp. 1041–1045.

[12] C. Huang, M. Chen, and J. Li, "Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems," *ACM Transactions on Storage (TOS)*, vol. 9, no. 1, p. 3, 2013.

[13] G. M. Kamath, N. Prakash, V. Lalitha, and P. V. Kumar, "Codes with local regeneration and erasure correction," *IEEE Trans. Inf. Theory*, vol. 60, no. 8, pp. 4637–4660, 2014.

[14] G. M. Kamath, N. Silberstein, N. Prakash, A. S. Rawat, V. Lalitha, O. O. Koyluoglu, P. Kumar, and S. Vishwanath, "Explicit mbr all-symbol locality codes," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Istanbul, Turkey, Jul. 2013, pp. 504–508.

[15] J. Li, X. Tang, and C. Tian, "Enabling all-node-repair in minimum storage regenerating codes," *[Online]. Available: arXiv:1604.07671 [cs.IT]*, 2016.

[16] L. Pamies-Juarez, H. D. L. Hollmann, and F. Oggier, "Locally repairable codes with multiple repair alternatives," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Istanbul, Turkey, Jul. 2013, pp. 892–896.

[17] D. S. Papailiopoulos and A. G. Dimakis, "Locally repairable codes," *IEEE Trans. Inf. Theory*, vol. 60, no. 10, pp. 5843–5855, 2014.

[18] N. Prakash, G. M. Kamath, V. Lalitha, and P. V. Kumar, "Optimal linear codes with a local-error-correction property," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Cambridge, MA, Jul. 2012, pp. 2776–2780.

[19] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal exact-regenerating codes for distributed storage at the msr and mbr points via a product-matrix construction," *IEEE Trans. Inf. Theory*, vol. 57, no. 8, pp. 5227–5239, 2011.

[20] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, "Explicit construction of optimal exact regenerating codes for distributed storage," in *Proc. 47th Annu. Allerton Conf. Communication, Control, and Computing*, Monticello, IL, Sep. 2009, pp. 1366–1373.

[21] A. S. Rawat, O. O. Koyluoglu, and S. Vishwanath, "Progress on high-rate msr codes: Enabling arbitrary number of helper nodes," in *Proc. IEEE Information Theory and Applications Workshop (ITA)*, La Jolla, CA, Feb. 2016, pp. 1–6.

[22] A. S. Rawat, O. O. Koyluoglu, N. Silberstein, and S. Vishwanath, "Optimal locally repairable and secure codes for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 60, no. 1, pp. 212–236, 2014.

[23] S. Rhea, C. Wellis, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz, "Maintenance-free global data storage," *Internet Computing, IEEE*, vol. 5, no. 5, pp. 40–49, 2001.

[24] B. Sasidharan, N. Prakash, M. N. Krishnan, M. Vajha, K. Senthoor, and P. V. Kumar, "Outer bounds on the storage-repair bandwidth trade-off of exact-repair regenerating codes," *International Journal of Information and Coding Theory*, vol. 3, no. 4, pp. 255–298, 2016.

[25] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, "Distributed storage codes with repair-by-transfer and nonachievability of interior points on the storage-bandwidth tradeoff," *IEEE Trans. Inf. Theory*, vol. 58, no. 3, pp. 1837–1852, 2012.

[26] ——, "Interference alignment in regenerating codes for distributed storage: Necessity and code constructions," *IEEE Trans. Inf. Theory*, vol. 58, no. 4, pp. 2134–2158, 2012.

[27] K. W. Shum and Y. Hu, "Cooperative regenerating codes," *IEEE Trans. Inf. Theory*, vol. 59, no. 11, pp. 7229–7258, 2013.

[28] N. Silberstein, A. S. Rawat, O. O. Koyluoglu, and S. Vishwanath, "Optimal locally repairable codes via rank-metric codes," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Istanbul, Turkey, Jul. 2013, pp. 1819–1823.

[29] I. Tamo, D. S. Papailiopoulos, and A. G. Dimakis, "Optimal locally repairable codes and connections to matroid theory," *IEEE Trans. Inf. Theory*, vol. 62, no. 12, pp. 6661–6671, 2016.

[30] I. Tamo, Z. Wang, and J. Bruck, "Zigzag codes: Mds array codes with optimal rebuilding," *IEEE Trans. Inf. Theory*, vol. 59, no. 3, pp. 1597–1616, 2013.

[31] D. B. West, *Introduction to graph theory*. Prentice Hall Upper Saddle River, NJ., 2001, vol. 2.

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

[33] Y. Wu and A. G. Dimakis, "Reducing repair traffic for erasure coding-based storage via interference alignment," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Seoul, South Korea, Jul. 2009, pp. 2276–2280.

Imad Ahmad (S'15–M'17) received the B.E. degree in electrical and computer engineering from the American University of Beirut, Beirut, Lebanon, in 2009, and the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, in 2016.

He is currently a Senior Member of Technical Staff at AT&T Labs, San Ramon, CA. His research interests are in information theory, machine learning, networking, and signal processing.

Chih-Chun Wang is a Professor of the School of Electrical and Computer Engineering of Purdue University. He received the B.E. degree in E.E. from National Taiwan University, Taipei, Taiwan in 1999, the M.S. degree in E.E., the Ph.D. degree in E.E. from Princeton University in 2002 and 2005, respectively. He worked in Comtrend Corporation, Taipei, Taiwan, as a design engineer in 2000 and spent the summer of 2004 with Flarion Technologies, New Jersey. In 2005, he held a post-doctoral researcher position in the Department of Electrical Engineering of Princeton University. He joined Purdue University in 2006, and became a Professor in 2017. He is currently a senior member of IEEE and served an associate editor of IEEE Transactions on Information Theory during 2014 to 2017. He served as the technical co-chair of the 2017 IEEE Information Theory Workshop. His current research interests are in the delay-constrained information theory and network coding. Other research interests of his fall in the general areas of networking, optimal control, information theory, detection theory, and coding theory.

Dr. Wang received the National Science Foundation Faculty Early Career Development (CAREER) Award in 2009.