Locally Repairable Regenerating Codes: Node Unavailability and the Insufficiency of Stationary Local Repair

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

***Abstract*—Recent works by Ahmad *et al* and by Hollmann studied the concept of "locally repairable regenerating codes (LRRCs)" that successfully combines functional repair and partial information exchange of regenerating codes (RCs) with the much-desired local repairability feature of locally repairable codes (LRCs).**

One important issue that needs to be addressed by any local repair schemes (including both LRCs and LRRCs) is that sometimes designated helper nodes may be temporarily unavailable, the result of various reasons that include multiple failures, degraded reads, or power-saving strategies to name a few. Under the setting of LRRCs with temporary node unavailability, this work studies the impact of different helper selection methods. It proves that with node unavailability, all existing methods of helper selection, including those used in RCs and LRCs, can be insufficient in terms of achieving the optimal repair-bandwidth. For some scenarios, it is necessary to combine LRRCs with a new class of helper selection methods, termed *dynamic helper selection*, to achieve optimal repair-bandwidth. This work also compares the performance of different classes of helper selection methods and answers the following fundamental question: *is one method of helper selection intrinsically better than the other?* for various scenarios.

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

I. Introduction

Erasure codes (ECs) are efficient in terms of the reliability versus redundancy tradeoff in distributed storage systems. An (n, k) maximum-distance-separable (MDS) code, when applied over a network of n storage nodes, can tolerate $n-k$ simultaneous failures. When a node fails, it is repaired by accessing *any* k surviving nodes, downloading all the coded data stored in these k nodes, and then reconstructing the original data. As a result, we say that the repair of ECs involves "full information exchange" and "exact repair".

Regenerating codes (RCs) [6] were proposed to decrease the communication required during repair, termed the *repair-bandwidth*. The three key ideas that allow RCs to decrease repair-bandwidth are: (i) contact as many nodes as possible during repair or in other words $d = n-1$ nodes, termed *helper nodes*, (ii) download only a partial fraction of the data ($\beta < \alpha$) as opposed to full information-exchange ($\beta = \alpha$), and (iii) allow *functional repair* which is a generalization of exact repair in ECs.

Another type of distributed storage codes is the locally repairable codes (LRCs) [8], [10], [11], [13]–[15] that use a small number of helper nodes d during repair. A closer look at LRCs shows that LRCs resemble ECs in that they operate with $\alpha = \beta$ and under exact repair. The main difference[1] is that ECs access $d = k$ helpers while LRCs use much smaller d values (usually $\ll k$). For that reason, LRCs can be viewed as a generalization of ECs with a much desired feature of local repairability (small d). Inspired by the repair-bandwidth reduction of RCs over ECs, it is thus natural to ask the question: whether it is possible to design *locally repairable regenerating codes* (LRRCs), that simultaneously admit all three features: local repairability ($d < k$), partial information-exchange ($\beta < \alpha$), and functional repair. [3]–[5], [9] are among the first works that studied LRRCs.

Note that whenever one considers a parameter value $d < n-1$, it is critical to design how to select the d helpers during repair. For the RCs in [6], the d helpers are chosen blindly and we term this scheme the *blind helper selection* (BHS) scheme. In LRCs and LRRCs results [3]–[5], [8]–[11], [13]–[15], some form of intelligent helper selection designs is used.

Table I summarizes the differences among ECs, RCs, LRCs, and LRRCs in terms of repair modes, the amount of information exchange, the corresponding helper selection schemes, and the target parameter values. The blue-shaded blocks correspond to the features that are known to be able to reduce the repair-bandwidth. Note that only LRCs and LRRCs employ intelligent helper selection rules.

Despite the preliminary promising results, the LRRCs considered in [3]–[5] and [9] do not consider the following practical issue: due to various reasons that include multiple failures, degraded reads, or power-saving strategies, helper nodes may be temporarily unavailable, say up to r nodes may be temporarily unavailable to help. Therefore, for local repairability to work, the code needs to have an alternative set of helpers. See the column "temporary node unavailability" in

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 2015 IEEE International Symposium on Information Theory.

Communicated by K. Narayanan, Associate Editor for Coding Techniques.

I. Ahmad is currently with AT&T Labs {imad.f.ahmad@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]Another subtle difference is that for ECs, a newcomer can access *any* set of d helpers while for LRCs each newcomer can only access *a predetermined set* of d helpers. As will be rigorously defined in Section II-B, the former is termed the *blind helper selection* and the latter is termed the *stationary helper selection*.

TABLE I
THE COMPARISON TABLE BETWEEN EXISTING CODES AND LRRCS OF THIS WORK.

	Repair Mode	Info-Exchange	Helper Selection	Temporary Node Unavailability	Target Parameters
ECs	Exact	Full	Blind	No Treatment Needed	$d = k$
Original RCs	Functional	Partial	Blind	No Treatment Needed	$d \geq k$
Exact Repair RCs	Exact	Partial	Blind	No Treatment Needed	$d \geq k$
LRCs	Exact	Full	Intelligent	Need Special Treatment	$d < k$
LRRCs of this work	Functional	Partial	Intelligent	Need Special Treatment	All Parameters

Table I. For LRCs, temporary node unavailability (i.e., with parameter $r \geq 1$) has been studied in [12], [14], [15]. In this work, we study the performance of LRRCs [3]–[5] and [9] under different helper selection policies while taking into account temporary node unavailability.

Our studies are centered around two different classes of helper selection schemes. Stationary helper selection (SHS) and a new class proposed in this work called *dynamic helper selection* (DHS). As will be explained in detail in Section II-D, *the helper selection policies of all existing ECs, RCs, LRCs, and LRRCs [3]–[6], [8]–[11], [13]–[15] are special instances of SHS.*[2]

The main contributions of this work are summarized as follows.

Contribution 1: We prove, for the first time in the literature, that both BHS and SHS can be insufficient in achieving the optimal repair-bandwidth[3] when intelligent helper selection is used, i.e., when the constraint that the newcomer should be able to repair from *any* d helper nodes is relaxed. Specifically, we provide an example with $r = 1$ showing that it is necessary to use DHS, which is designed based on a completely different principle, to achieve the optimal repair-bandwidth while the performance of BHS and *any* SHS are strictly suboptimal. Furthermore, the DHS scheme in our example is simultaneously minimum bandwidth regenerating (MBR) and minimum storage regenerating (MSR), attaining a new storage-bandwidth tradeoff point that was previously believed to be not possible except for some trivial degenerate cases. Such an example demonstrates the benefit of DHS and calls for further research on DHS designs.

Contribution 2: For the vast majority of (n, k, d, r) values, we answer whether we can design an SHS or DHS scheme that strictly outperforms BHS. Surprisingly, for many (n, k, d, r) values the answer is no. We say that those (n, k, d, r) values are *indifferent to helper selection* since the performance does not depend on the type of helper selection scheme being used. The concept of indifferent-to-helper-selection will be formally defined in Section II-C. Knowing whether an (n, k, d, r) value is indifferent to helper selection is of significant practical value for the designer to decide whether to simply use BHS or to design more sophisticated helper selection rules.

The rest of this paper is organized as follows. Section II motivates the problem and introduces key definitions and notation. Section III describes information-flow graphs, the main tool we use for the analysis of LRRCs. Section IV presents the main results of this work. Section V presents proofs of the results of Contribution 1. Section VI presents proofs of the results of Contribution 2. Section VII concludes this work.

In Table II we list all the acronyms used in this work.

TABLE II
THE LIST OF ACRONYMS USED IN THIS PAPER.

BHS	Blind Helper Selection
DHS	Dynamic Helper Selection
EC	Erasure Code
IFG	Information Flow Graph
LRC	Locally Repairable Code
LRRC	Locally Repairable Regenerating Code
MBR	Minimum Bandwidth Regenerating
MDS	Maximum Distance Separable
MFHS	Modified Family Helper Selection
MSR	Minimum Storage Regenerating
RC	Regenerating Code
RFIP	Rotating Family Index Permutation
SHS	Stationary Helper Selection
TA	Triangle Avoiding
wSHS	weakly Stationary Helper Selection

II. Problem Statement

A. The Parameters of A Distributed Storage Network

This work follows the same distributed storage network model as introduced in the seminal work [6]. For completeness, we provide in the following detailed definitions of some key parameters. Further explanations of the system model can be found in [6].

1) Parameters n and $\hat{k}$: We denote the total number of nodes in a storage network by n. For any $1 \leq \hat{k} \leq n - 1$, we say that a code can satisfy the reconstruction requirement if any $\hat{k}$ 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 $\hat{k} = 6$ since any 6 nodes can construct the original file. By the same definition, the Hamming code can also satisfy

[2]As explained in Section II-D, the BHS scheme is also a special instance of SHS.

[3]In this work, we focus mainly on the storage and repair-bandwidth tradeoff while completely ignoring the flexibility/reliability offered by the schemes as long as the flexibility meets a certain threshold parametrized by r. See Section II for further details. Some very useful schemes may offer a level of flexibility exceeding the given threshold r. For example, the BHS scheme has the flexibility to choose "any" d out of the $(n - 1)$ remaining nodes, which corresponds to the largest flexibility that far exceeds the given threshold r. Such flexibility is achieved at the cost of using higher repair-bandwidth. We say BHS is bandwidth-suboptimal among those that can meet the given threshold r since the additional flexibility provided by BHS is not part of the design consideration/requirements.

the reconstruction requirement for $\hat{k}=5$, but cannot satisfy the reconstruction requirement for $\hat{k}=4$ (since a Hamming code is not an MDS code). The smallest $\hat{k}$ value that the $(7,4)$ Hamming code can satisfy is thus 5. In the data storage literature, this smallest $\hat{k}$ value is denoted as k and we thus say that the $(7,4)$ Hamming code[4] has $(n,k)=(7,5)$. It is important to see the distinction that the value of $\hat{k}$ 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. Since part of our goal is to compare different schemes/implementations, it is thus crucial to focus on the desired/target data protection level $\hat{k}$ rather than the implementation-dependent actual protection level k.

2) Parameter d: We denote the number of nodes that a newcomer can access during repair by d.

3) Parameter r: We denote the maximum number of nodes that can be temporarily unavailable at any given time by r. Specifically, if we denote the set of unavailable nodes by U, then we must have $|U|\leq r$. If we also denote the failed node by F, the design goal is to repair node F when the nodes in U are unavailable and cannot serve as any of the d helpers. The unavailability of nodes in U may be due to degenerate reads, multiple failures, power-saving strategies or other reasons. In this work we do not consider repair collaboration. That means, even when we have multiple failures, say both nodes i and j fail simultaneously, we repair each node separately. For example, we set $F=i$ and $U=\{j\}$ when repairing node i and we set $F=j$ and $U=\{i\}$ when repairing node j. Some repair cooperation schemes that jointly repair both nodes i and j can be found in [17].

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

5) The Range of the Design Criteria $(n,\hat{k},d,r)$: Due to the nature of the distributed storage problem, we only consider $(n,\hat{k},d,r)$ values that satisfy

$$2\leq n;\ \ 1\leq\hat{k}\leq n-1;\ \ 1\leq d;\ \text{ and } d\leq n-1-r. \tag{1}$$

In all the results in this work, we assume *implicitly* that the n, $\hat{k}$, and d values satisfy (1).

We note that in this paper, for LRRCs, we consider all possible values of $\hat{k}$ and d and we do not impose any relationship between them. In contrast, for example, it is assumed in the literature for RCs that $\hat{k}\leq d$. The reason is that, when RCs are used together with BHS, any RC that can meet the requirement of $(n,\hat{k},d,r)$ can also meet a more stringent requirement of $(n,\min(\hat{k},d),d,r)$ since the actual protection level k of RCs with BHS always satisfies $k\leq d$. Therefore, when characterizing a RC with BHS performance under parameters $(n,\hat{k},d,r)$, one can focus only on the cases of $\hat{k}\leq d$ without loss of generality.

In contrast, the goal of this paper is to compare the best performance of *any possible helper selection scheme* that can still satisfy the desired $(n,\hat{k},d,r)$ values. It turns out that for some intelligent helper selection schemes (e.g., LRCs) that meet the requirement $(n,\hat{k},d,r)$, the actual protection level k satisfies $k>d$. Therefore, a scheme that can meet the requirement of a given $(n,\hat{k},d,r)$ does not necessarily meet the requirement of $(n,\min(\hat{k},d),d,r)$. As a result, unlike the results on RCs, we do not impose the constraint $\hat{k}\leq d$ in this work and allow arbitrary order between $\hat{k}$ and d.

Parameters α, β, and $\mathcal{M}$: 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 $\gamma\triangleq d\beta$.

B. Types of Helper Selection Schemes

Definition 1: Given $(n,\hat{k},d,r)$, a *dynamic helper selection (DHS)* scheme is defined by the functions $\mathcal{D}_\tau(\{F_i\}_{i=1}^\tau,\{U_j\}_{j=1}^\tau)$ for all $\tau\in[1,\infty)$, where τ is the current time of failure/repair, F_i and U_j are the failed node at time i and the set of unavailable nodes at time j, respectively, and $\mathcal{D}_\tau(\cdot,\cdot)$ returns a non-empty collection of node sets that are each of size d and disjoint from $\{F_\tau\}\cup U_\tau$. At time τ, the newcomer arbitrarily chooses any set from the collection of sets returned by this function and accesses the d nodes in this set for repair.

Definition 2: Given $(n,\hat{k},d,r)$, a *stationary helper selection (SHS)* scheme is a DHS scheme where its function $D_\tau(\cdot)$ can be rewritten as

$$\mathcal{D}_\tau(\{F_i\}_{i=1}^\tau,\{U_j\}_{j=1}^\tau)=\mathcal{D}(F_\tau,U_\tau). \tag{2}$$

Definition 3: Given $(n,\hat{k},d,r)$, a *weakly SHS (wSHS)* scheme is a SHS scheme where its function $\mathcal{D}(\cdot)$ can be rewritten as

$$\mathcal{D}(F_\tau,U_\tau)=\mathsf{col}_d(\overline{D}(F_\tau)\backslash U_\tau), \tag{3}$$

where $\overline{D}(\cdot)$ takes F_τ as input and outputs a set of at least $d+r$ nodes that does not contain node F_τ; the $\backslash U_\tau$ operation removes any node in $\overline{D}(F_\tau)$ that is also in U_τ; and the function $\mathsf{col}_d(\cdot)$ then returns the collection of all the subsets of $\overline{D}(F_\tau)\backslash U_\tau$ that contain exactly d nodes.

Definition 4: Given $(n,\hat{k},d,r)$, a *blind helper selection (BHS)* scheme is a wSHS scheme that has the following function

$$\mathcal{D}(F_\tau,U_\tau)=\mathsf{col}_d(([n]\backslash\{F_\tau\})\backslash U_\tau), \tag{4}$$

where again $\mathsf{col}_d(\cdot)$ returns all the subsets of size d of its input and $[n]=\{1,2,\cdots,n\}$.

Namely, BHS chooses $\overline{D}(F_\tau)=[n]\backslash\{F_\tau\}$ in (3) and outputs all the subsets of d nodes that are disjoint from $\{F_\tau\}\cup U_\tau$.

The class of DHS schemes is the most general class of helper selection. The corresponding helper selection at current time τ can depend on the time index τ and the history of node failures and node unavailabilities from all the previous time slots 1 to $(\tau-1)$.

The class of SHS is a subset of the class of DHS schemes. Its function $\mathcal{D}_\tau(\cdot,\cdot)$ does not change with respect to the value of τ and only depends on the input arguments F_τ and U_τ

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

instead of the entire history $\{F_i\}_{i=1}^{\tau}$ and $\{U_j\}_{j=1}^{\tau}$. Being a sub-class of the DHS schemes, the function $\mathcal{D}(F_\tau, U_\tau)$ still outputs a collection of subsets such that each subset has d nodes and is disjoint from $\{F_\tau\} \cup U_\tau$. For shorthand, we may sometimes drop the subscript τ in F_τ and U_τ and write the function of a SHS as $\mathcal{D}(F, U)$. We can see that this construction is stationary because the collection of helper sets does not change with time and only depends on the latest node failure and node unavailability information.

The class of wSHS schemes is a subset of the class of SHS schemes. The idea is that wSHS schemes are SHS schemes that assign a fixed set of at least $d + r$ helper nodes for each node first and then, according to U, a subset of helper nodes is chosen. Note that one can view the set $\overline{D}(F)$ as the *candidate* helper set when node F fails.

The BHS is one specific wSHS scheme. This scheme is the helper selection scheme assumed initially for RCs [6]. We note that DHS, SHS, and wSHS are classes of helper selection schemes whereas BHS is one specific scheme.

C. Capacity of Helper Selection

Given $(n, \hat{k}, d, r, \alpha, \beta)$ values and a helper selection scheme[5] A, we define the rate of scheme A, $R_A(\alpha, \beta)$, as the largest file size $\mathcal{M}$ that can be stored by a distributed storage code/network such that the reconstruction and repair requirements are satisfied. Note that the rate definition depends on the other four parameters $(n, \hat{k}, d, r)$ which should be clear from the context.

We define

$$C_{\text{DHS}}(\alpha, \beta) = \sup_{A \text{ being a DHS scheme}} R_A(\alpha, \beta). \tag{5}$$

In a similar fashion, we define

$$C_{\text{SHS}}(\alpha, \beta) = \max_{A \text{ being an SHS scheme}} R_A(\alpha, \beta). \tag{6}$$

For the BHS scheme, we denote its rate throughout by $R_{\text{BHS}}(\alpha, \beta)$.

Using the above capacity definitions, we can write the following

Proposition 1: For all $(n, \hat{k}, d, r, \alpha, \beta)$ values, we have

$$C_{\text{DHS}}(\alpha, \beta) \geq C_{\text{SHS}}(\alpha, \beta) \geq R_{\text{BHS}}(\alpha, \beta). \tag{7}$$

Proposition 1 is a direct result of the facts that BHS is a special instance of SHS schemes and that SHS is a sub-class of DHS. We can then define the optimality and universal optimality of different helper selection schemes.

Definition 5: For any given $(n, \hat{k}, d, r, \alpha, \beta)$ values, a helper selection scheme A is *optimal* if $R_A(\alpha, \beta) = C_{\text{DHS}}(\alpha, \beta)$. That is, for this pair of (α, β) values, scheme A allows for the protection of the largest possible file size.

Definition 6: For any given $(n, \hat{k}, d, r)$ values, a helper selection scheme A is *universally optimal* if it is optimal for all (α, β) pairs.

[5]A helper selection scheme is described by the function $D_\tau(\{F_i\}_{i=1}^{\tau}, \{U_j\}_{j=1}^{\tau})$ as specified in Definition 1.

Definition 7: We say that a given $(n, \hat{k}, d, r)$ value is *indifferent to helper selection* if the BHS scheme is universally optimal.

Namely, for such an $(n, \hat{k}, d, r)$ value, no intelligent helper selection scheme can outperform the simplest BHS scheme.

For a fixed file size $\mathcal{M}$, the equation $R_A(\alpha, \beta) = \mathcal{M}$ describes the storage-bandwidth tradeoff (α versus β) curve. Oftentimes, we can use this curve for analyzing the performance of distributed storage codes as seen in the literature [6]. For a fixed set of values $(n, \hat{k}, d, r)$ and fixed file size $\mathcal{M}$, we can also define the two extreme points on a storage-bandwidth tradeoff curve of any given helper selection scheme A: the minimum bandwidth regenerating (MBR) and minimum storage regenerating (MSR) points [6]. These points are defined as follows:

Definition 8: For any given $(n, \hat{k}, d, r)$ values and a fixed file size $\mathcal{M}$, the MBR point $(\alpha_{\text{MBR}}, \beta_{\text{MBR}})$ of a helper selection scheme A is defined by

$$\beta_{\text{MBR}}(A) \triangleq \min\{\beta : R_A(\alpha, \beta) \geq \mathcal{M}, \alpha = \infty\} \tag{8}$$

$$\alpha_{\text{MBR}}(A) \triangleq \min\{\alpha : R_A(\alpha, \beta) \geq \mathcal{M}, \beta = \beta_{\text{MBR}}\}. \tag{9}$$

Definition 9: For any given $(n, \hat{k}, d, r)$ values and a fixed file size $\mathcal{M}$, the MSR point $(\alpha_{\text{MSR}}, \beta_{\text{MSR}})$ of a helper selection scheme A is defined by

$$\alpha_{\text{MSR}}(A) \triangleq \min\{\alpha : R_A(\alpha, \beta) \geq \mathcal{M}, \beta = \infty\} \tag{10}$$

$$\beta_{\text{MSR}}(A) \triangleq \min\{\beta : R_A(\alpha, \beta) \geq \mathcal{M}, \alpha = \alpha_{\text{MSR}}\}.$$

Specifically, the MBR and MSR points are the two extreme ends[6] of the bandwidth-storage tradeoff curve $R_A(\alpha, \beta) = \mathcal{M}$. When it is clear from the context which helper selection scheme A we are referring to, we sometimes drop the input argument (A) and use $(\alpha_{\text{MBR}}, \beta_{\text{MBR}})$ and $(\alpha_{\text{MSR}}, \beta_{\text{MSR}})$ for convenience.

Remark: The MSR and MBR points in Definitions 8 and 9 depend on the file size $\mathcal{M}$, which is highly related to the alphabet size and the sub-packetization used in the code construction. However, since in this work we use the information flow graph (see Section III) and the corresponding cut values (as also used by [6]) to define the rate $R_A(\alpha, \beta)$, in the sequel we assume $\mathcal{M}$ is sufficiently large so that the graph-based results in this work are separated from the delicate effects of alphabet size and sub-packetization of the actual code construction.

[6]The MSR and MBR points are defined here in a different manner than in the existing literature. Recall that the definition of the MSR point used in the literature is $\alpha'_{\text{MSR}} = \frac{\mathcal{M}}{k}$ packets and notice that it depends on k and not $\hat{k}$ (see Section II). This definition stems from the facts that ECs and RCs [6] do not consider $\hat{k}$ and assume $k \leq d$ for which the MSR point always satisfies $\alpha'_{\text{MSR}} = \frac{\mathcal{M}}{k}$. However, when we are given $(n, \hat{k}, d, r)$ parameters with a $\hat{k}$ value, we cannot simply replace k with $\hat{k}$ in the MSR point of RCs to get that $\alpha'_{\text{MSR}}(A) = \frac{\mathcal{M}}{\hat{k}}$ for LRRCs since this is not always achievable for $d < \hat{k}$. For example, when $(n, \hat{k}, d, r) = (5, 3, 2, 1)$, one can prove that regardless how one designs the helper selection scheme A, we always have $\alpha_{\text{MSR}}(A) \geq \frac{\mathcal{M}}{2}$. Specifically, the smallest achievable storage $\alpha_{\text{MSR}}(A)$ is lower bounded by $\frac{\mathcal{M}}{2}$ and $\alpha'_{\text{MSR}}(A) = \frac{\mathcal{M}}{\hat{k}} = \frac{\mathcal{M}}{3}$ is not achievable by any helper selection scheme.

D. Helper Selection Schemes in Existing Works

The helper selection schemes of all existing LRC constructions are SHS[7]. Specifically, for $r = 0$ (i.e., nodes are always available), LRCs [8], [13], [14] use SHS where each node is assigned a fixed set of d helper nodes. For $r > 0$, LRCs [10], [15] assign each node a fixed set of $(d+r)$ helper nodes and during repair the newcomer can arbitrarily connect to any d nodes of the $(d+r)$ nodes in its helper set. As defined in Section II-B, such a scheme belongs to the class of wSHS schemes. Almost all LRCs considered in the existing literature use wSHS to handle temporary node unavailability. To our knowledge, the only example in the literature that is not based on wSHS is [12], for which the helper selection is a carefully designed SHS function $\mathcal{D}(F, U)$.

III. Information Flow Graphs and the Corresponding Graph-Based Analysis

Before introducing our main results, we quickly explain the concepts of information flow graphs (IFGs) and the corresponding analysis, which was first introduced in [6]. For readers who are not familiar with IFGs, we provide its detailed description in Appendix A.

Intuitively, each IFG reflects one unique history of the failure patterns and the helper selection choices from time 1 to $(\tau - 1)$ [6]. Consider any given helper selection scheme A which can belong to either DHS or SHS. Since there are infinitely many different failure patterns F_τ and infinitely many different unavailable node sets U_τ (since we consider $\tau = 1$ to ∞), there are infinitely many IFGs corresponding to the same given helper selection scheme A since the IFG grows according to the helper choices $D_\tau(\{F_i\}_{i=1}^{\tau}, \{U_j\}_{j=1}^{\tau})$ for DHS or $D(F_\tau, U_\tau)$ for SHS. We denote the collection of all possible IFGs of a given helper selection scheme A by $\mathcal{G}_A(n, \hat{k}, d, r, \alpha, \beta)$. We define $\mathcal{G}(n, \hat{k}, d, r, \alpha, \beta) = \bigcup_{\forall A} \mathcal{G}_A(n, \hat{k}, d, r, \alpha, \beta)$ as the union over all possible helper selection schemes A. We sometimes drop the input argument and use $\mathcal{G}_A$ and $\mathcal{G}$ as shorthands. The collection $\mathcal{G}$ can also be viewed as the IFGs generated by BHS. The reason is that BHS blindly selects the helpers and thus will take into consideration *all possible ways* of growing the IFG. As a result, $\mathcal{G}_{\text{BHS}} = \mathcal{G} = \bigcup_{\forall A} \mathcal{G}_A(n, \hat{k}, d, r, \alpha, \beta)$.

Given an IFG $G \in \mathcal{G}$, we use $\text{DC}(G)$ to denote the collection of all $\binom{n}{\hat{k}}$ *data collector nodes* in G [6]. Each data collector $t \in \text{DC}(G)$ represents one unique way of choosing $\hat{k}$ out of n active nodes when reconstructing the file. Given an instance of the IFGs $G \in \mathcal{G}$ and a data collector $t \in \text{DC}(G)$, we use $\text{mincut}_G(s, t)$ to denote the *minimum cut value* [19] separating s, the root node (source node) of G, and t.

For a given helper selection scheme A, the concept of IFGs allows us to find the rate $R_A(\alpha, \beta)$ of a distributed storage system with scheme A. Specifically, the results in [1] prove that the following condition is *necessary* for the existence of any distributed storage code with helper selection scheme A that can meet the design requirement $(n, \hat{k}, d, r, \alpha, \beta)$:

$$\min_{G \in \mathcal{G}_A} \min_{t \in \text{DC}(G)} \text{mincut}_G(s, t) \geq \mathcal{M}. \tag{11}$$

If we limit our focus to a distributed storage network with BHS, then the above necessary condition becomes

$$\min_{G \in \mathcal{G}} \min_{t \in \text{DC}(G)} \text{mincut}_G(s, t) \geq \mathcal{M}. \tag{12}$$

Reference [6] later found a closed-form expression of the left-hand side of (12)

$$\min_{G \in \mathcal{G}} \min_{t \in \text{DC}(G)} \text{mincut}_G(s, t) = \sum_{i=0}^{\hat{k}-1} \min((d-i)^+\beta, \alpha), \tag{13}$$

where $(x)^+ = \max(x, 0)$. Specifically, the necessary condition (12) becomes

$$\sum_{i=0}^{\hat{k}-1} \min((d-i)^+\beta, \alpha) \geq \mathcal{M}. \tag{14}$$

Reference [20] further proves that when considering a fixed but sufficiently large finite field $\text{GF}(q)$, (14) is not only necessary but also sufficient for the existence of a BHS-based distributed storage code that meets the design requirement $(n, \hat{k}, d, r, \alpha, \beta)$. Therefore, we can write that

$$R_{\text{BHS}}(\alpha, \beta) = \sum_{i=0}^{\hat{k}-1} \min((d-i)^+\beta, \alpha). \tag{15}$$

IV. The Main Results

The main contributions of this work are the answers to the following two questions. Question 1: When designing an optimal helper selection scheme, is it sufficient[8] to limit the search scope to only considering SHS schemes? Question 2: For arbitrary $(n, \hat{k}, d, r)$, is there a way to quickly determine whether BHS is optimal or not? Question 2 is motivated by our observation that, for some $(n, \hat{k}, d, r)$ values, even the best DHS/SHS schemes do not do better than the simple BHS scheme.

We answer the first question in the following Theorem 1 and answer partially the second question in Theorem 2.

Theorem 1: For $(n, \hat{k}, d, r) = (5, 3, 2, 1)$ and $(5, 4, 2, 1)$, and any arbitrary (α, β) values,

$$C_{\text{SHS}}(\alpha, \beta) = R_{\text{BHS}}(\alpha, \beta). \tag{16}$$

Furthermore, for $(n, \hat{k}, d, r) = (5, 3, 2, 1)$ and $(5, 4, 2, 1)$, for any fixed β there exists a sufficiently large α such that

$$C_{\text{DHS}}(\alpha, \beta) > R_{\text{BHS}}(\alpha, \beta). \tag{17}$$

[7]Although the definition of LRCs does not rule out the possiblity of DHS, all existing LRC constructions belong to the class of SHS.

[8]A simple analogy is as follows. It is known that for binary symmetric channels linear codes are capacity-achieving. Namely, there is little need to search for non-linear codes. When considering network coding, again linear codes are capacity-achieving for the single multicast setting. But the seminal results in [7] prove that linear codes are not sufficient for the multiple unicast setting. For this work, we would like to answer the question whether SHS is sufficient (capacity-achieving) for all $(n, \hat{k}, d, r)$ values.

Theorem 1 proves that when $(n, \hat{k}, d, r) = (5, 3, 2, 1)$ or $(5, 4, 2, 1)$, for any fixed β there exists a sufficiently large α such that

$$C_{\text{DHS}}(\alpha, \beta) > C_{\text{SHS}}(\alpha, \beta) = R_{\text{BHS}}(\alpha, \beta). \tag{18}$$

This answers Question 1 by showing that SHS is not enough to achieve the optimal performance for $(n, \hat{k}, d, r) = (5, 3, 2, 1)$ and $(5, 4, 2, 1)$. An explicit DHS scheme is devised in Section V-B that is universally optimal for these two $(n, \hat{k}, d, r)$ values (see Definitions 5 and 6). A byproduct of our universally optimal DHS scheme is that it achieves the MBR and MSR points simultaneously. Specifically, it simultaneously minimizes the bandwidth and storage for $(n, \hat{k}, d, r) = (5, 3, 2, 1)$ and $(5, 4, 2, 1)$.

The following theorem answers the second question by providing conditions that can be used to determine whether BHS is optimal or not for a given $(n, \hat{k}, d, r)$ value. Since Theorem 2 and Propositions 3 and 4 below involve many different cases, we summarize the results in Table III.

Theorem 2: If the following inequality

$$\hat{k} \leq \left\lceil \frac{n - r}{n - d - r} \right\rceil \tag{19}$$

holds, then for all (α, β) values

$$C_{\text{DHS}}(\alpha, \beta) = C_{\text{SHS}}(\alpha, \beta) = R_{\text{BHS}}(\alpha, \beta). \tag{20}$$

If the following inequality

$$\min(d + 1, \hat{k}) > \left\lceil \frac{n}{n - d - r} \right\rceil \tag{21}$$

holds, then for any fixed β there exists a sufficiently large α such that

$$C_{\text{DHS}}(\alpha, \beta) \geq C_{\text{SHS}}(\alpha, \beta) > R_{\text{BHS}}(\alpha, \beta). \tag{22}$$

There are some $(n, \hat{k}, d, r)$ values that satisfy neither (19) nor (21) for which it remains open whether BHS is optimal or not for those $(n, \hat{k}, d, r)$ values. Therefore, the characterization in Theorem 2 is not tight.

For the cases[9] of $r \leq 1$, we can further sharpen the results as follows.

Proposition 2: [5, Propositions 1 and 2] For any $(n, \hat{k}, d, r)$ values satisfying $r = 0$, if either (19) or

$$d = 1, \ \hat{k} = 3, \ \text{and } n \text{ is odd} \tag{23}$$

holds, then for any (α, β) values

$$C_{\text{DHS}}(\alpha, \beta) = C_{\text{SHS}}(\alpha, \beta) = R_{\text{BHS}}(\alpha, \beta). \tag{24}$$

If neither (19) nor (23) holds, then for any fixed β there exists a sufficiently large α such that

$$C_{\text{DHS}}(\alpha, \beta) \geq C_{\text{SHS}}(\alpha, \beta) > R_{\text{BHS}}(\alpha, \beta). \tag{25}$$

Proposition 3: For any $(n, \hat{k}, d, r)$ values satisfying $r = 1, d = 1$, if either (i) (19) holds, (ii) $\hat{k} = 3$, or (iii)

$$\hat{k} = 4, \ \text{and } n \bmod 3 \neq 0 \tag{26}$$

[9]Arguably, the cases of small r are more interesting from a practical perspective.

holds, then for any (α, β) values,

$$C_{\text{DHS}}(\alpha, \beta) = C_{\text{SHS}}(\alpha, \beta) = R_{\text{BHS}}(\alpha, \beta). \tag{27}$$

If none of (i)-(iii) holds, then for any fixed β there exists a sufficiently large α such that

$$C_{\text{DHS}}(\alpha, \beta) \geq C_{\text{SHS}}(\alpha, \beta) > R_{\text{BHS}}(\alpha, \beta). \tag{28}$$

Proposition 4: For any $(n, \hat{k}, d, r)$ value satisfying $r = 1, d = 2$, if neither (19) nor (21) holds, then for any fixed β, there exists a sufficiently large α such that

$$C_{\text{DHS}}(\alpha, \beta) > C_{\text{SHS}}(\alpha, \beta) = R_{\text{BHS}}(\alpha, \beta). \tag{29}$$

Propositions 2 to 4 close the gap in Theorem 2 and provide tight characterization for the cases of "$r = 0$" and "$r = 1, d \leq 2$." Also see the rows in Table III corresponding to "$r = 0$", "$r = 1, d = 1$", and "$r = 1, d = 2$". Although we do not have a tight characterization when $3 \leq d$, one can still use Theorem 2 to prove the following corollary.

Corollary 1: For any $(n, \hat{k}, d, r)$ satisfying $r \leq 1$, $d \leq 5$, and

$$(n, \hat{k}, d, r) \notin \{(7, 3, 3, 1), (9, 3, 4, 1), (7, 4, 4, 1), (11, 3, 5, 1)\}, \tag{30}$$

we can easily determine whether $C_{\text{DHS}}(\alpha, \beta) = R_{\text{BHS}}(\alpha, \beta)$ or $C_{\text{DHS}}(\alpha, \beta) > R_{\text{BHS}}(\alpha, \beta)$ since for these $(n, \hat{k}, d, r)$ values, either (19) or (21) must be true. Namely, when $r = 1$ and $3 \leq d \leq 5$, only four tuples $(n, \hat{k}, d, r)$ fall into the "unknown case" of Table III.

The proof of Theorem 1 will be presented in Section V. The proofs of the converse and the achievability parts of Theorem 2 will be provided in Sections VI.

Proposition 2 focuses on the special case of $r = 0$ and is a restatement of the existing results in [5, Propositions 1 and 2]. The proof of Proposition 3 is relegated to Appendix B. We close this section by providing the proof of Proposition 4, which reveals a connection between Proposition 4 and Theorem 1.

Proof of Proposition 4: We first show that, when $r = 1$ and $d = 2$, (19) and (21) cover all the range of parameters that satisfy (1) except for the points $(n, \hat{k}, d, r) = (5, 3, 2, 1)$ and $(5, 4, 2, 1)$. When $r = 1$ and $d = 2$, the left-hand side of (19) becomes $\left\lceil \frac{n-1}{n-3} \right\rceil$ and the left-hand side of (21) becomes $\left\lceil \frac{n}{n-3} \right\rceil$. Since $d \leq n - 1 - r$, we must have $n \geq 4$ when $r = 1$ and $d = 2$. In the following we analyze the gap between the two conditions "$\hat{k} \leq \left\lceil \frac{n-1}{n-3} \right\rceil$" and "$\min(d + 1, \hat{k}) > \left\lceil \frac{n}{n-3} \right\rceil$" for different n values.

For $n = 4$, (19) becomes $\hat{k} \leq \lceil \frac{3}{1} \rceil = 3$ and (21) becomes $\min(3, \hat{k}) > \lceil \frac{4}{1} \rceil = 4$. By (1), we must also have $\hat{k} \leq n - 1 = 3$. Therefore, for the scenarios of $n = 4$, $d = 2$, and $r = 1$, all possible $(n, \hat{k}, d, r)$ values satisfy (19) and none of them satisfy (21).

For $n = 5$, (19) becomes $\hat{k} \leq \lceil \frac{4}{2} \rceil = 2$. By (1), we must also have $\hat{k} \leq n - 1 = 4$. For $\hat{k} = 1, 2$, (19) is satisfied. On the other hand, $\hat{k} = 3, 4$ cannot satisfy (19). Since (21) becomes $\min(3, \hat{k}) > \lceil \frac{5}{2} \rceil = 3$, no $\hat{k}$ value can satisfy (21). We thus

TABLE III
A SUMMARY OF THE RESULTS IN THEOREM 2 AND PROPOSITIONS 2, 3 AND 4

$(n, \hat{k}, d, r)$ values	$C_{\rm DHS} \geq C_{\rm SHS} > R_{\rm BHS}$ for the MBR point: i.e., a fixed β but with $\alpha = \infty$	$C_{\rm DHS} > C_{\rm SHS} = R_{\rm BHS}$ for the MBR point: i.e., a fixed β but with $\alpha = \infty$	$C_{\rm DHS} = C_{\rm SHS} = R_{\rm BHS}$ for all possible (α, β)
$r = 0$	If and only if: Neither (19) nor (23) holds	An impossible case	If and only if: Either (19) or (23) holds
$r = 1, d = 1$	If and only if: None of (19), "$\hat{k} = 3$", and (26) holds	An impossible case	If and only if: Either of (19), "$\hat{k} = 3$", or (26) holds
$r = 1, d = 2$	If and only if: (21) holds	If and only if: Neither (19) nor (21) holds	If and only if: (19) holds
General $(n, \hat{k}, d, r)$	If: (21) holds	An unknown/open case	If: (19) holds

have that points $(n, \hat{k}, d, r) = (5, 4, 2, 1)$ and $(5, 3, 2, 1)$ satisfy neither (19) nor (21).

For $n \geq 6$, we first observe that $1 < \frac{n-1}{n-3} < \frac{n}{n-3} \leq 2$ whenever $n \geq 6$. The reason is as follows. The first 2 strict inequalities are straightforward. The last inequality follows from that $\frac{n}{n-3}$ is monotonically decreasing with n and $\frac{6}{6-3} = 2$. The above observation thus ensures that when $n \geq 6$, (19) becomes $\hat{k} \leq \left\lceil \frac{n-1}{n-3} \right\rceil = 2$ and (21) becomes $\min(3, \hat{k}) > \left\lceil \frac{n}{n-3} \right\rceil = 2$. We can see that there is no gap between these two conditions. Therefore, for $n \geq 6$, all possible parameters satisfy either (19) or (21).

We have thus shown that the only points that conditions (19) and (21) do not cover are the points $(n, \hat{k}, d, r) = (5, 4, 2, 1)$ and $(5, 3, 2, 1)$.

At the same time, these two $(n, \hat{k}, d, r)$ points are indeed the focus of Theorem 1, which then implies (29). The proof is thus complete. ■

V. PROOF OF THEOREM 1

The proof of Theorem 1 is provided in Sections V-A and V-B. Jointly, these sections prove that SHS is insufficient in terms of achieving the optimal repair-bandwidth. A byproduct of the result in Theorem 1 is a simple proof showing that functional repair can be better than exact repair, provided that we allow the use of LRRCs with DHS instead of the traditional model of RCs with BHS. The simple proof is provided in Section V-D.

A. *Proof of* (16)

We first consider the case of $(n, \hat{k}, d, r) = (5, 3, 2, 1)$. If BHS is used, the newcomer can access any $d = 2$ out of $3 = n - r - 1$ available nodes and it thus naturally handles node unavailability ($r = 1$). By plugging $(n, \hat{k}, d) = (5, 3, 2)$ in (15), we have that

$$R_{\rm BHS}(\alpha, \beta) = \min(2\beta, \alpha) + \min(\beta, \alpha). \tag{31}$$

A normalized storage-bandwidth tradeoff curve, i.e., for fixed $\mathcal{M} = 1$, of (31) is plotted in Fig. 1. Namely, if each node stores only half of the overall file $\frac{\alpha}{\mathcal{M}} = 0.5$, then the normalized repair-bandwdith is $\frac{d\beta}{\mathcal{M}} = 1$. However, if we are willing to use a larger normalized storage size $\frac{\alpha}{\mathcal{M}} = \frac{2}{3}$ rather than $\frac{1}{2}$, we can reduce the normalized bandwidth $\frac{d\beta}{\mathcal{M}}$ from 1 to $\frac{2}{3}$. Note that when BHS is used, we are essentially analyzing

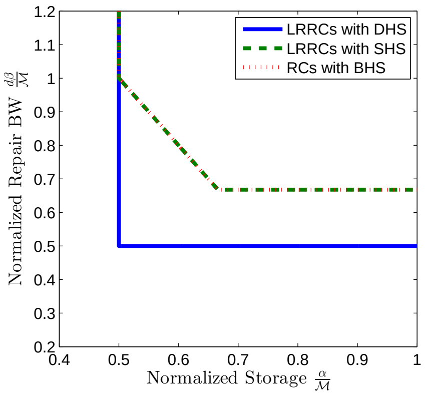

Fig. 1. Storage-bandwidth tradeoff curves of LRRCs with DHS, LRRCs with SHS, and RCs with BHS for $(n, \hat{k}, d, r) = (5, 3, 2, 1)$.

the original RCs in [6] with $(n, \hat{k}, d) = (5, 3, 2)$. Therefore, the actual protection level satisfies $k = d = 2$ which is smaller than the target protection level $\hat{k} = 3$. This means that any code construction with BHS is overprotecting the data.

In the following, we will show that even if one is allowed to choose the helpers in an intelligent way (other than BHS), we are still not able to improve on (31) if we are restricted to using only SHS. One implication of this result is that when $(n, \hat{k}, d, r) = (5, 3, 2, 1)$ any existing/future LRC scheme will have the same performance as the original RC if restricted to using only SHS schemes.

Consider any SHS scheme A with the corresponding stationary helper selection function being $\mathcal{D}(F, U)$ where F is the failed node and $U = \{j\}$ is the set containing the temporarily unavailable node j since we now focus on $r = 1$. For simplicity, we sometimes just say node U is unavailable when it is clear from the context that $|U| = r = 1$. Without loss of generality, we assume $\mathcal{D}(F, U)$ returns only 1 subset of d nodes, even though by our definition the function $\mathcal{D}(F, U)$ may return a collection of several subsets of d nodes. The reason is that since the newcomer will arbitrarily/blindly choose one subset of d nodes from the collection returned by $\mathcal{D}(F, U)$, the resulting performance becomes the "worst case" of the entire returned collection. As a result, we can safely assume $\mathcal{D}(F, U)$ returns only 1 subset of d nodes. Taking it a step further, we can instead use $D(F, U)$ to denote the returned set (i.e., $\mathcal{D}(F, U)$ returns the collection

$\mathcal{D}(F,U) = \{D(F,U)\})$.

Recall that $\mathcal{G}_A$ is the collection of IFGs that are grown according to the helper selection scheme A. The main idea is to prove that no matter how we design the $D(F,U)$, we are bound to have a graph $G^* \in \mathcal{G}_A$ for which $\min_{t\in \mathrm{DC}(G^*)} \mathrm{mincut}_{G^*}(s,t)$ equals to the left-hand side of (31). This implies that

$$R_A(\alpha,\beta) \le \min(2\beta,\alpha) + \min(\beta,\alpha). \tag{32}$$

By Proposition 1, and by (31), we thus have

$$\begin{aligned} R_{\mathrm{BHS}}(\alpha,\beta) &\le R_A(\alpha,\beta) \\ &\le \min(2\beta,\alpha) + \min(\beta,\alpha) \\ &= R_{\mathrm{BHS}}(\alpha,\beta). \end{aligned} \tag{33}$$

As a result, all the inequalities in (33) must be equalities. Therefore, $R_A(\alpha,\beta) = R_{\mathrm{BHS}}(\alpha,\beta)$ for all (α,β).

To that end, we will prove that regardless how we design the helper selection $D(F,U)$ function, we can always find a graph $G^* \in \mathcal{G}_A$ such that there exists 3 active nodes x, y, and z satisfying (i) each node has been repaired at least once, and (ii) x is a helper when repairing y, and (iii) both x and y are the helpers when repairing z.

Once such G^* and nodes x, y, and z are found, consider the cut in G^* that directly separates the source (root) from $\{x,y,z\}$, we can observe that node x will contribute $\min(2\beta,\alpha)$ to the cut value; node y will contribute $\min(\beta,\alpha)$ to the cut value since node x was the helper of node y; and node z will contribute 0 to the cut value since both x and y are the helpers of z. Therefore, the cut that separates the source (root) directly from $\{x,y,z\}$ will have the cut-value being $\min(2\beta,\alpha)+\min(\beta,\alpha)$. As a result, the min-cut value $\mathrm{mincut}_{G^*}(s,\{x,y,z\})$ is no larger than the left-hand-side of (31). (32) is thus proven.

We now prove the existence of such an IFG G^* and nodes x, y, and z. Without loss of generality, suppose that $D(1,\{4\}) = \{2,3\}$ in the SHS scheme A. Namely, if node 1 fails and node 4 is not available, then the newcomer (node 1) will access nodes 2 and 3 as helpers. This assumption can always be made true by relabeling the nodes. We consider the following 2 cases.

Case 1: $D(2,\{1\}) \neq \{4,5\}$ or $D(2,\{4\}) \neq \{1,5\}$. Consider the following three subcases. Case 1.1: $D(2,\{1\}) \neq \{4,5\}$. Since $D(2,\{1\})$, by definition, returns a subset of $\{1,2,3,4,5\}\backslash(\{F\}\cup U) = \{3,4,5\}$, we must have either $D(2,\{1\}) = \{3,4\}$ or $\{3,5\}$. We now fail node 3 first and repair it following scheme A. (The rule that scheme A uses to repair node 3 is irrelevant in our proof.) Then, fail node 2 and suppose node 1 is unavailable. Since $D(2,\{1\}) = \{3,4\}$ or $\{3,5\}$ in Case 1.1, node 3 will definitely be a helper of node 2. Next, fail node 1 and assume node 4 is unavailable. Since $D(1,\{4\}) = \{2,3\}$, node 1 will access nodes 2 and 3 for repair. We can observe that we have constructed such a G^* where nodes $(x,y,z) = (3,2,1)$ satisfy properties (i) to (iii) above. The proof for Case 1.1 is complete.

Case 1.2: $D(2,\{4\}) \neq \{1,5\}$. Since $D(2,\{4\})$, by definition, returns a subset of $\{1,3,5\}$, we must either have $D(2,\{4\}) = \{1,3\}$ or $\{3,5\}$. Similar to Case 1.1, we fail node 3 first. Then, we fail node 2 and assume that node 4 is unavailable. Since $D(2,\{4\}) = \{1,3\}$ or $\{3,5\}$, node 3 must be a helper of node 2. Finally, fail node 1 and assume that node 4 is unavailable. Since $D(1,\{4\}) = \{2,3\}$, node 1 will access nodes 2 and 3 for repair. In the end, nodes $(x,y,z) = (3,2,1)$ satisfy (i) to (iii).

Case 2: $D(2,\{1\}) = \{4,5\}$ and $D(2,\{4\}) = \{1,5\}$. Case 2.1: $D(1,\{3\}) \neq \{4,5\}$. Therefore, we must have $D(1,\{3\}) = \{2,4\}$ (or $\{2,5\}$). For simpler exposition, we say $D(1,\{3\}) = \{2,v\}$ where v is either node 4 or node 5. We now fail node v first and repair it under scheme A. (The rule that scheme A uses to repair node v is irrelevant in our proof.) We then fail node 2 and assume that node 1 is unavailable. Since $D(2,\{1\}) = \{4,5\}$, nodes 4 and 5 are the helpers of node 2. Then, fail node 1 and assume node 3 is unavailable. Since $D(1,\{3\}) = \{2,v\}$, nodes 2 and v are the helpers of node 1. Observe that $(x,y,z) = (v,2,1)$ satisfy (i) to (iii) and we have thus constructed such a G^*.

Case 2.2: $D(1,\{3\}) = \{4,5\}$. We fail node 5 first and repair it under scheme A. (The rule that scheme A uses to repair node v is irrelevant in our proof.) Fail node 1 second, while assuming that node 3 is unavailable. Since $D(1,\{3\}) = \{4,5\}$, nodes 4 and 5 are the helpers of node 1. Then, fail node 2 and assume that node 4 is unavailable. Since in Case 2 we have $D(2,\{4\}) = \{1,5\}$, nodes 5 and 1 are the helpers of node 2. Observe that $(x,y,z) = (5,1,2)$ satisfy (i) to (iii) and we have thus constructed such a G^*.

Thus far, we have completed the proof for the case of $(n,\hat{k},d,r) = (5,3,2,1)$. We now discuss how to prove the case of $(n,\hat{k},d,r) = (5,4,2,1)$.

To prove that the capacity of SHS is (31), we will prove that regardless how we choose the helper selection function $D(F,U)$, there always exists a graph $G^* \in \mathcal{G}_A$ such that there exist 4 active nodes x, y, z, and w satisfying (i) each node has been repaired at least once, and (ii) x is a helper when repairing y, (iii) both x and y are the helpers when repairing z, and (iv) the helper nodes of w are a subset of $\{x,y,z\}$.

By the discussion in the previous proof of $(n,\hat{k},d,r) = (5,3,2,1)$, we can always find a G^* such that there exist three active nodes (x,y,z) satisfying (i) to (iii). Without loss of generality, assume the three active nodes are $(x,y,z) = (1,2,3)$. Then, we fail node 4 and assume node 5 is unavailable. Since there are only 3 remaining nodes $\{1,2,3\}$, regardless how we choose $D(4,\{5\})$, the helpers of node 4 must be a subset of $\{1,2,3\}$. Choose $w = 4$. Then nodes $(x,y,z,w) = (1,2,3,4)$ satisfy (i) to (iv).

By similar arguments, one can easily check that the min-cut separating the root of G^* and the four nodes (x,y,z,w) is at most $\min(2\beta,\alpha)+\min(\beta,\alpha)$. Therefore, $C_{\mathrm{SHS}}(\alpha,\beta) = R_{\mathrm{BHS}}(\alpha,\beta)$. The proof of (16) in Theorem 1 is thus complete.

B. *Proof of* (17)

1) The description of the scheme: We first consider $(n,\hat{k},d,r) = (5,3,2,1)$ and prove that there exists a DHS scheme that can protect any file size $\mathcal{M}$ that satisfies:

$$2\min(2\beta,\alpha) \ge \mathcal{M}, \tag{34}$$

i.e., can achieve a file size larger than $C_{\text{SHS}} = R_{\text{BHS}}$. Our proof is by explicit code construction with $\beta = 1$, $\alpha = 2$, and $\mathcal{M} = 4$, which achieves the corner point of (34), also see Fig. 1. The scheme consists of two parts. **Part I:** How to choose the helper nodes for a newcomer? **Part II:** What is the coded data sent by each helper[10] after the helpers are decided?

To describe **Part I**, we need the following notation. We say node i is the *parent* of node j if (i) node i was the helper of node j, and (ii) node i has not been repaired since the failure of node j. For example, say node 1 fails and accesses nodes 2 and 3 as helpers. Then node 2 fails and accesses nodes 3 and 4. After the above two repairs, node 3 is a parent of node 1 but node 2 is not since node 2 has been repaired. On the other hand, both nodes 3 and 4 are parents of node 2.

The main idea of the proposed DHS scheme is to choose helpers such that no 3 nodes ever form a "triangle". Namely, we carefully choose the helpers of the newcomers so that we can avoid the existence of 3 nodes $\{x, y, z\}$ such that x is the parent of both y and z; and y is the parent of z. We term this DHS scheme *the Triangle-Avoiding (TA) scheme*.

We now prove by induction that TA is always possible. In the beginning, all nodes are intact and no node is the parent of another. Therefore, there does not exist any 3 nodes forming a triangle. Suppose there is no triangle after $(\tau_0 - 1)$ repairs. At time $\tau = \tau_0$, suppose a node fails. For future reference, denote that node as node z. Since the network had no triangle at time $(\tau_0 - 1)$, we only need to ensure that the newcomer z does not participate in any triangle *after* the repair. Denote the helper choice of the TA scheme for time $\tau = \tau_0$ by $\{x, y\}$. Therefore, we only need to carefully choose the helper set $\{x, y\}$ such that neither nodes $\{x, y, z\}$ nor nodes $\{y, x, z\}$ form a triangle after repair.

To prove the existence of such $\{x, y\}$, we observe that out of the $n - 1 = 4$ surviving nodes, at most $r = 1$ node is unavailable. As a result, the newcomer c has 3 nodes to choose $d = 2$ helpers from. Say, the nodes to choose from are $\{i, j, k\}$ and without loss of generality assume node i is the oldest (being repaired the earliest) and node k is the youngest (being repaired the latest) of the three. We argue that among the three pairs $\{(i, j), (j, k), (i, k)\}$ one of them must not form a parent-child pair. If not, i.e., all three are parent-child pairs, then nodes $\{i, j, k\}$ form a triangle in time $(\tau_0 - 1)$, which leads to a contradiction. Without loss of generality, say node i is not a parent of k. Then, we choose nodes i and k to be the helper set $\{x, y\}$. As a result, neither nodes $\{x, y, z\}$ nor nodes $\{y, x, z\}$ form a triangle.[11] By induction TA is always possible.

Note that the TA scheme needs to use the *repair history* to decide which of the three pairs $\{(i, j), (j, k), (i, k)\}$ is not a parent-child pair and then chooses that pair as the helpers. Therefore, the choice of the helper set may vary from time to time even for the same failed node F and the same unavailable node U. This is a significant departure from the principle of associating each newcomer node x with a *fixed helper set*. Because the TA scheme has to *dynamically* select the helpers based on repair history, we can see that the TA scheme is indeed a DHS scheme.

We now describe **Part II**: What is the coded data sent by each helper? Our construction uses only the *binary field* rather than high-order $\text{GF}(q)$. A concrete example will be given after we give a complete description of our coding scheme. **Initialization:** Recall that $\alpha = 2$, $\beta = 1$, and $\mathcal{M} = 4$. Consider a file of 4 packets X_1 to X_4. Initially, we let nodes 1 and 2 store $\{X_1, X_2\}$ and $\{X_3, X_4\}$, respectively. We then let nodes 3 and 4 store packets $\{X_1, X_3\}$ and $\{X_2, X_4\}$, respectively. Finally, let node 5 store coded packets $\{[X_1 + X_2], [X_3 + X_4]\}$. The initialization phase is now complete. See Fig. 2 for the illustration after the initialization phase.

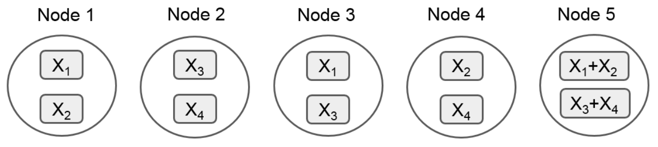

Fig. 2. The code of the TA scheme after initialization.

For easier description of our code construction, immediately after initialization, we artificially define nodes 1 and 2 as the parents of node 3 even though nodes 1 and 2 are not helpers of node 3. The reason is that the packets in node 3 are $\{X_1, X_3\}$ and they can be viewed as if node 3 has failed and got repaired from nodes 1 and 2. Similarly, we artificially define nodes 1 and 2 as the parents of node 4 (resp. node 5) even though nodes 1 and 2 are not helpers of node 4 (resp. node 5).[12]

The regular repair operations: Suppose node a fails and one other node is temporarily unavailable at time τ. We run the aforementioned TA helper selection scheme to find the helpers b and c for node a. Denote the two non-helper nodes by d and e. Each of b and c will send 1 packet to a since $\beta = 1$. The packets are constructed as follows. **Step 1:** Denote the two (potentially coded) packets stored in b by $Y_1^{(b)}$ and $Y_2^{(b)}$. Among the three candidate packets $Y_1^{(b)}$, $Y_2^{(b)}$, and the binary sum $[Y_1^{(b)} + Y_2^{(b)}]$, node b will choose one packet, call it Z_b^*, and send it to a.

Before describing how to choose Z_b^*, we construct two conditions based on the packets currently stored in nodes c, d, and e. If nodes c and d jointly contain 4 *linearly independent* packets, then we construct Condition 1 to be "Z_b^* cannot be expressed as a linear combination of the two packets stored in d." Otherwise, we construct Condition 1 to be "Z_b^* cannot be expressed as a linear combination of the packets stored in c and d." Namely, depending on the coded packets stored in nodes c and d, Condition 1 can be one of the above two different statements. Similarly, if nodes c and e jointly contain 4 linearly independent packets, then we construct Condition 2 to be "Z_b^* cannot be expressed as a linear combination of the two packets stored in e." Otherwise, we construct Condition 2

[10]Since $\alpha = 2 = d\beta$, each node simply stores all the $d\beta$ packets it has received in its local memory.

[11]Recall that node i is older than node k so node k can never be a parent of node i.

[12]Even with the artificially defined parent-child relationship, there is no triangle after initialization. We can thus use the same induction proof to show that TA is always possible after initialization.

to be "Z_b^* cannot be expressed as a linear combination of the packets stored in c and e."

After constructing the two conditions, we require the choice Z_b^* to satisfy simultaneously both Conditions 1 and 2. If there is more than one choice of Z_b^* satisfying both conditions, then an arbitrary one of those choices of Z_b^* will do.

Step 2: Denote the two packets stored in c by $Y_1^{(c)}$ and $Y_2^{(c)}$. Among the three candidate packets $Y_1^{(c)}$, $Y_2^{(c)}$, and the binary sum $[Y_1^{(c)} + Y_2^{(c)}]$, node c will choose one packet, call it Z_c^*, and send it to node a. We require the packet Z_c^* to satisfy simultaneously: (i) Z_c^* cannot be expressed as a linear combination of Z_b^* and the two packets stored in d; and (ii) Z_c^* cannot be expressed as a linear combination of Z_b^* and the two packets stored in e.

Once node a receives Z_b^* and Z_c^*, it stores both packets in its local memory (since $\alpha = 2$).

2) A concrete example illustrating how the proposed scheme operates: Let us use an example to illustrate our construction. Suppose immediately after initialization, node 3 fails and node 2 is unavailable. Newcomer 3 thus has to access two of the nodes $\{1, 4, 5\}$ for repair. Since node 1 is the parent of both nodes 4 and 5, the TA scheme will avoid choosing $\{1, 4\}$ and $\{1, 5\}$ and select helpers $\{4, 5\}$ instead. Specifically, $a = 3$, $b = 4$, and $c = 5$; and $d = 1$ and $e = 2$.

Since node $b = 4$ stores $\{X_2, X_4\}$, see Fig. 2, the three candidates for Z_b^* are X_2, X_4, and $[X_2 + X_4]$.

Since node $c = 5$ stores $\{[X_1 + X_2], [X_3 + X_4]\}$ and node $d = 1$ stores $\{X_1, X_2\}$, these two nodes contain 4 linearly independent packets. Condition 1 becomes "Z_b^* cannot be any linear expression of packets X_1 and X_2, the packets in node d." Similarly, since $e = 2$ stores $\{X_3, X_4\}$ and jointly nodes c and e contain 4 linearly independent packets, Condition 2 becomes "Z_b^* cannot be any linear expression of packets X_3 and X_4, the packets in node e." Out of the three candidates X_2, X_4, and $[X_2 + X_4]$, only the coded packet $[X_2 + X_4]$ can satisfy both conditions simultaneously. Therefore, we choose $Z_b^* = [X_2 + X_4]$.

Since node $c = 5$ stores $\{[X_1 + X_2], [X_3 + X_4]\}$, the three candidates for Z_c^* are $[X_1 + X_2]$, $[X_3 + X_4]$, and $[X_1 + X_2 + X_3 + X_4]$. The choice of Z_c^* thus has to satisfy simultaneously (i) Z_c^* is not a linear combination of $Z_b^* = [X_2 + X_4]$ and the two packets X_1 and X_2 in node d; and (ii) Z_c^* is not a linear combination of $Z_b^* = [X_2 + X_4]$ and the two packets X_3 and X_4 in node e. Out of the three candidates $[X_1+X_2]$, $[X_3+X_4]$, and $[X_1 + X_2 + X_3 + X_4]$, only the coded packet $[X_1 + X_2 + X_3 + X_4]$ can satisfy both conditions simultaneously. Therefore, we choose $Z_c^* = [X_1 + X_2 + X_3 + X_4]$.

In the end, $Z_b^* = [X_2 + X_4]$ will then be sent to node a from node b and $Z_c^* = [X_1 + X_2 + X_3 + X_4]$ will be sent to node a from node c. Newcomer a will then store both packets in its storage. The same repair process can then be repeated and applied to any arbitrary next newcomer.

3) Performance analysis (validity) of the proposed scheme:

Lemma 1 (feasibility of the proposed scheme): We can always find the Z_b^* and Z_c^* satisfying the specified conditions. As a result the code can be iteratively constructed for all times $\tau = 1$ to ∞.

The proof of Lemma 1 is relegated to Appendix C.

Proposition 5: Using the above construction (Parts I and II), for any time τ, any $\hat{k} = 3$ nodes can always reconstruct the original $n = 4$ packets X_1 to X_4. Such a binary code construction $(\alpha, \beta, \mathcal{M}) = (2, 1, 4)$ thus satisfies the reliability requirement $(n, \hat{k}, d, r) = (5, 3, 2, 1)$.

The proof of Proposition 5 is relegated to Appendix C.

The above TA helper selection scheme (Part I) and its code construction (Part II) thus achieve the right-hand-side of (34) for the case of $(n, \hat{k}, d, r) = (5, 3, 2, 1)$.

For the case of $(n, \hat{k}, d, r) = (5, 4, 2, 1)$, we notice that we can use the same code[13] that is constructed for $(n, \hat{k}, d, r) = (5, 3, 2, 1)$ to achieve (34). At the end of Section V-A, we have already proven that $C_{\text{SHS}}(\alpha, \beta) = R_{\text{BHS}}(\alpha, \beta)$ for $(n, \hat{k}, d, r) = (5, 4, 2, 1)$ and for all (α, β). As a result (17) of Theorem 1 is proven for $(n, \hat{k}, d, r) = (5, 3, 2, 1)$ and $(5, 4, 2, 1)$.

C. The Optimal Solution for $(n, \hat{k}, d, r) = (5, 3, 2, 1)$ and $(5, 4, 2, 1)$

In this subsection, we prove that there exists no scheme (DHS or SHS) that can protect a file larger than the right-hand-side of (34). Therefore, the scheme described in Section V-B is universally optimal.

Proposition 6: Suppose $(n, \hat{k}, d, r) = (5, 3, 2, 1)$ or $(5, 4, 2, 1)$. We have that for all (α, β)

$$C_{\text{DHS}}(\alpha, \beta) = 2 \min(2\beta, \alpha), \tag{35}$$

i.e., the TA scheme/code is universally optimal.

Proof of Proposition 6: To prove (35), it is sufficient to prove that for any DHS scheme A, we have

$$\min_{G \in \mathcal{G}_A} \min_{t \in \text{DC}(G)} \text{mincut}_G(s, t) \leq 2 \min(2\beta, \alpha). \tag{36}$$

To that end, we first prove this proposition for $(n, \hat{k}, d, r) = (5, 3, 2, 1)$. Consider an IFG $G^* \in \mathcal{G}_A$ such that all its nodes have been repaired before. Consider the newest node in G^* that we denote by z. Observe that z must be connected to two older active nodes, call them x and y. Now, consider a data collector t that is connected to $\{x, y, z\}$. We can see that node x will contribute $\min(2\beta, \alpha)$ to the value of the cut that directly separates the root from the three nodes $\{x, y, z\}$. Moreover, node y will contribute at most $\min(2\beta, \alpha)$ to the value of the cut that directly separates $\{x, y, z\}$. On the other hand, node z cannot contribute anything to the cut-value since it is connected to both x and y. Therefore, the value of the cut that directly separates the root and $\{x, y, z\}$ is at most $2 \min(2\beta, \alpha)$. As a result, the minimum cut-value $\text{mincut}_{G^*}(s, t)$ for that particular t is upper bounded by $2 \min(2\beta, \alpha)$. Taking the minimum of all possible t and all possible $G \in \mathcal{G}_A$, we get the inequality in (36) and the proof is complete.

For the case of $(n, \hat{k}, d, r) = (5, 4, 2, 1)$, consider an IFG $G^* \in \mathcal{G}_A$ such that all its nodes have been repaired before. Consider the newest node in G^* that we denote by z. Observe

[13]In this way, we are *overprotecting* the data since the actual $k = 3$ but the target $\hat{k} = 4$.

that z must be connected to two older active nodes, call them x and y. Denote the nodes other than nodes x, y, and z, by nodes w and u. We fail node w and make u temporarily unavailable. Repair node w according to the given scheme A, which must access 2 out of the three remaining nodes $\{x, y, z\}$.

Now, consider a data collector that is connected to $\{x, y, z, w\}$. We can see that node x will contribute $\min(2\beta, \alpha)$ to the value of the cut that directly separates the root from $\{x, y, z, w\}$ and node y will contribute at most $\min(2\beta, \alpha)$ to the value of that cut. Nodes z and w will not contribute any amount to the cut value since z is connected to $\{x, y\}$ and w is connected to two of $\{x, y, z\}$. By the verbatim argument as in the case of $(n, \hat{k}, d, r) = (5, 3, 2, 1)$, we have thus proven (35) for the case of $(n, \hat{k}, d, r) = (5, 4, 2, 1)$. ■

D. A Byproduct of Theorem 1

In [16], it was shown that the majority of the interior points on the tradeoff curve of RCs with BHS cannot be achieved by exact repair. The exact repair rate region of the simple case of $(n, \hat{k}, d, r) = (4, 3, 3, 0)$ was characterized in [18] and it was shown that indeed there is a gap between the optimal tradeoff of functional repair and exact repair. Specifically, functional repair is strictly more powerful than exact repair and its benefits should not be overlooked.

The fundamental finding in [18] is proven by a computer-aided proof. It turns out that if we focus on a different $(n, \hat{k}, d, r)$ value other than $(4, 3, 3, 0)$ and if we allow for LRRCs with DHS rather than the simple RCs with BHS, we can easily prove the same statement "functional repair outperforms exact repair" without resorting to the relatively-involved computer-aided-proof approach.

Denote by $C_{\text{funct,DHS}}(\alpha, \beta)$ and $C_{\text{exact,DHS}}(\alpha, \beta)$ the capacity (largest file size that can be protected) of LRRCs with DHS under the functional repair scenario and under the restriction to exact repair, respectively.

Proposition 7: For $(n, \hat{k}, d, r) = (5, 3, 2, 1)$ and $(5, 4, 2, 1)$, there exists at least one pair of (α, β) values such that

$$C_{\text{funct,DHS}}(\alpha, \beta) > C_{\text{exact,DHS}}(\alpha, \beta). \quad (37)$$

Furthermore, for these two $(n, \hat{k}, d, r)$ values, (37) occurs at both the MSR and MBR points, unlike the case of $(n, \hat{k}, d, r) = (4, 3, 3, 0)$ where it only happens at the interior points.[14]

Proof of Proposition 7: The proof is by contradiction. Consider $(n, \hat{k}, d, r) = (5, 3, 2, 1)$ or $(5, 4, 2, 1)$. The following arguments work for both cases. By Theorem 1, we know that when $\beta = 1$ and $\alpha = 2$, we have

$$C_{\text{funct,DHS}} = 4 > C_{\text{funct,SHS}} = R_{\text{funct,BHS}} = 3. \quad (38)$$

Suppose now that there exist exact repair LRRCs with DHS that can achieve $C_{\text{exact,DHS}} = 4$. Since such a code is an exact repair code, the same helper nodes that can repair a failed node at time $\tau = 1$ can be used to repair the same failed node at any other time τ.

[14]It is known that $R_{\text{funct,BHS}}(\alpha, \beta) = R_{\text{exact,BHS}}(\alpha, \beta)$ for both the MBR and MSR points, but not the interior points.

The reason is that in exact repair the packets on the nodes are the same at any time, so we can reuse the helper choice in time 1 and the resulting new code should still meet the reliability requirement. Thus, the considered exact repair LRRC with DHS can be converted to an exact repair LRRC with SHS. Therefore, $C_{\text{exact,DHS}} = 4$ implies $C_{\text{exact,SHS}} = 4$, which contradicts (16) of Theorem 1 that shows that $C_{\text{funct,SHS}} = 3$ since we always have $C_{\text{funct,SHS}} \geq C_{\text{exact,SHS}}$. Therefore, the initial assumption $C_{\text{funct,DHS}} = C_{\text{exact,DHS}}$ is false. (37) is thus proven.

If we compare the tradeoff curve of functional repair and the tradeoff curve of the best possible exact repair, see Fig. 1, it is clear that (37) occurs at both the MSR and MBR points when considering $C_{\text{funct,DHS}}(\alpha, \beta)$. Hence, the proof is complete. ■

VI. Proof of Theorem 2

In this section, we prove Theorem 2, which focuses on answering the question: Given $(n, \hat{k}, d, r)$ values, whether there exists a DHS/SHS scheme that outperforms the baseline BHS scheme.

A. The $(n, \hat{k}, d, r)$ Values for Which BHS is Optimal

For easier reference, we reproduce the converse part of Theorem 2 in the following proposition.

Proposition 8: If $\hat{k} \leq \left\lceil \frac{n-r}{n-d-r} \right\rceil$, then for all (α, β) values

$$C_{\text{DHS}}(\alpha, \beta) = C_{\text{SHS}}(\alpha, \beta) = R_{\text{BHS}}(\alpha, \beta). \quad (39)$$

Proof of Proposition 8: Since the capacity $C_{\text{DHS}}(\alpha, \beta)$ is the worst case analysis among all possible failure and node unavailability patterns, we focus on a special node unavailability pattern. Specifically, we define the unavailable set V as nodes $\{1, 2, \cdots, r\}$, the first r nodes. And for every time instant, we reuse/impose the same unavailable node set V.

In [5, Proposition 1] (and the proof provided in [5, Appendix C-A]), it has been proven that if $r = 0$, (39) holds whenever $\hat{k} \leq \left\lceil \frac{n}{n-d} \right\rceil$. Proposition 8 thus holds when $r = 0$.

If $r > 0$, since in our construction nodes 1 to r are always unavailable, the effective number of nodes thus reduces from n to $n - r$ when we focus only on the last $n - r$ available nodes. Therefore, the situation is as if we are facing a scenario with new parameters $n' = n - r$ and $r' = 0$ since none of the remaining $n - r$ nodes will ever be unavailable. Again by invoking [5, Theorem 1] while replacing n by $n - r$, we have thus proven that if $r > 0$, (39) holds whenever $\hat{k} \leq \left\lceil \frac{n-r}{n-r-d} \right\rceil$. The proof is complete. ■

B. The Achievability Proof

For easier reference, we reproduce the achievability part of Theorem 2 in the following proposition.

Proposition 9: If $\min(d + 1, \hat{k}) > \left\lceil \frac{n}{n-d-r} \right\rceil$, then for any fixed β there exists a sufficiently large α such that

$$C_{\text{DHS}}(\alpha, \beta) \geq C_{\text{SHS}}(\alpha, \beta) > R_{\text{BHS}}(\alpha, \beta). \quad (40)$$

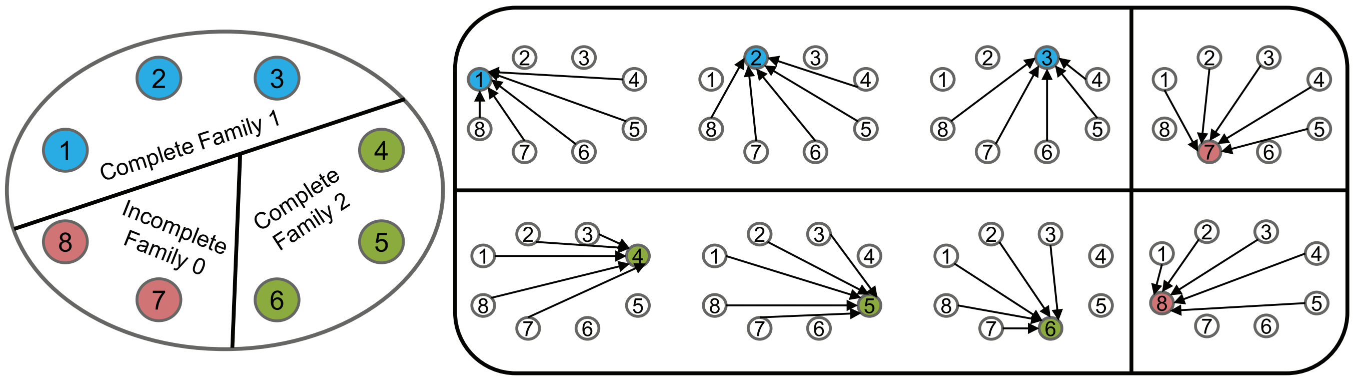

Fig. 3. The MFHS scheme for $(n, d, r) = (8, 4, 1)$ and the illustration of the repair process of each of the 8 nodes. Each newcomer may choose to access $(d + r) = 5$ helpers, as illustrated in the arrows. However, only $d = 4$ of them will be actually accessed since we assume $r = 1$ of the helpers may be temporarily unavailable.

We prove the above result by explicit construction. We describe a helper selection scheme first and then we analyze its performance.

1) Scheme: The proposed scheme is termed the *modified family helper selection (MFHS)* scheme, which is based on the family helper selection scheme in [3]–[5] that was originally devised for the case of $r = 0$, i.e., all helper nodes are always available. The MFHS is a wSHS that can be described as follows. First, we arbitrarily sort all storage nodes and denote them by 1 to n. Then, we define a *complete family* as a group of $(n - d - r)$ physical nodes. The first $(n - d - r)$ nodes are grouped as the first complete family, the second $(n - d - r)$ nodes are grouped as the second complete family and so on. In total, we have $\left\lfloor \frac{n}{n-d-r} \right\rfloor$ complete families. The remaining $n \bmod (n - d - r)$ nodes, if there is any, are grouped as an *incomplete family*. For any node F, if F belongs to a complete family, $\overline{D}(F)$ is defined as the set of nodes outside the family of F (see Section II-B). Since the family of node F has $(n - d - r)$ nodes, $\overline{D}(F)$ contains exactly $n - (n - d - r) = d + r$ nodes. If F belongs to an incomplete family, $\overline{D}(F)$ is defined as the set of nodes from the first node to the $(d + r)$-th node (recall we sorted the nodes in the very beginning). Again $\overline{D}(F)$ contains exactly $(d + r)$ nodes. Now, we define the helper function for the MFHS as $\mathcal{D}(F, U) = g(\overline{D}(F)\backslash U)$, where compared to col_d of the wSHS (see Section II-B) g returns exactly *one* subset containing the first d smallest integers/nodes of its input set while col_d returns *all* subsets of exactly d nodes of its input set.

2) An Example of How the MFHS Works: For example, suppose that $(n, d, r) = (8, 4, 1)$. There are 2 complete families, $\{1, 2, 3\}$ and $\{4, 5, 6\}$, and 1 incomplete family, $\{7, 8\}$. See Fig. 3 for illustration. Then suppose node 4 fails. Since node 4 belongs to a complete family $\{4, 5, 6\}$, $\overline{D}(4) = \{1, 2, 3, 7, 8\}$ since nodes 1, 2, 3, 7, and 8 are outside the family of node 4. Therefore, if node 2 is temporarily unavailable $U = \{2\}$, the newcomer will then access nodes $\overline{D}(4)\backslash U = \{1, 3, 7, 8\}$ for repair. If it is node 8 being unavailable, then the newcomer will access $\overline{D}(4)\backslash\{8\} = \{1, 2, 3, 7\}$ for help. If it is node 5 being unavailable, then the g function will return the first 4 nodes in $\overline{D}(4)\backslash\{5\} = \{1, 2, 3, 7, 8\}$, which is $\{1, 2, 3, 7\}$. If node 7 fails, then since node 7 belongs to an incomplete family $\{7, 8\}$, the corresponding candidate helper set contains the first $(d + r) = 5$ nodes $\overline{D}(7) = \{1, 2, 3, 4, 5\}$. If node 2 is unavailable $(U = \{2\})$, then the helpers become $\overline{D}(7)\backslash U = \{1, 3, 4, 5\}$.

3) Analysis: In the following, we analyze the performance of the MFHS scheme. Before we analyze the performance, we introduce some useful definitions.

The family index vector: Notice that the MFHS scheme has in total $\left\lceil \frac{n}{n-d-r} \right\rceil$ families, which we index from 1 to $\left\lceil \frac{n}{n-d-r} \right\rceil$. However, since the incomplete family has different properties from the complete families, we index the incomplete family by the family index 0. The family indices thus become from 1 to $c \triangleq \left\lfloor \frac{n}{n-d-r} \right\rfloor$ and then 0, where c is the index of the last Complete family. If there is no incomplete family, we omit the index 0. Moreover, notice that any member of the incomplete family has $\overline{D}(F) = \{1, \cdots, d + r\}$. That is, for an incomplete family node F, $\overline{D}(F)$ contains *all* the members of the first $(c-1)$ complete families and only the first $(d+r)-(n-d-r)(c-1) = n \bmod (n-d-r)$ members of the last complete family c. Among the $(n-d-r)$ members in the last complete family, *we add a negative sign to the family indices of those that will "not" be helpers for the incomplete family.*

We use the notation $FI(n_0)$ to denote the family index of node n_0. We can now list the family indices of the n nodes as an n-dimensional family index vector defined as $(FI(1), FI(2), \ldots, FI(n))$. Considering the same example above where $(n, d, r) = (8, 4, 1)$, the *family index vector* is $(1, 1, 1, 2, 2, -2, 0, 0)$.

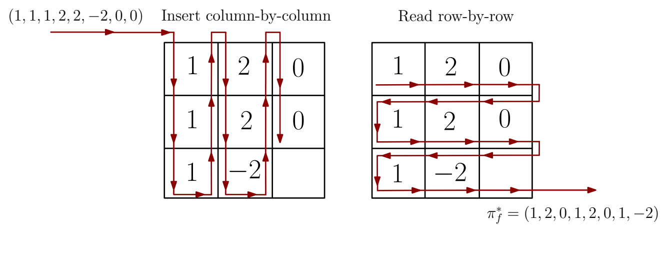

Fig. 4. The construction of the RFIP for $(n, d, r) = (8, 4, 1)$.

The family index permutation and the RFIP: A *family index permutation* is a permutation of the family index vector, which we denote by π_f. Using the previous example, one instance of family index permutations is $\pi_f = (1,1,0,2,0,-2,1,2)$. The *rotating family index permutation (RFIP)* π_f^* is a special family index permutation that puts the family indices of the family index vector in an $(n-d-r) \times \left\lceil \frac{n}{n-d-r} \right\rceil$ table column-by-column and then reads it row-by-row. Fig. 4 illustrates the construction of the RFIP for the case of $(n,d,r) = (8,4,1)$. The input is the family index vector $(1,1,1,2,2,-2,0,0)$ and the output RFIP π_f^* is $(1,2,0,1,2,0,1,-2)$.

We now analyze the performance of the MFHS scheme. Denote by $R_{\text{MFHS}}(\alpha,\beta)$ the rate of the MFHS scheme.

Proposition 10: Consider any $(n,\hat{k},d,r)$, we have that

$$R_{\text{MFHS}}(\alpha,\beta) = \min_{\forall \pi_f} \sum_{i=1}^{\hat{k}} \min((d-y_i(\pi_f))^+\beta, \alpha), \tag{41}$$

where π_f can be any family index permutation and $y_i(\pi_f)$ is computed as follows. If the i-th coordinate of π_f is 0, then $y_i(\pi_f)$ returns the number of j satisfying both (i) $j < i$ and (ii) the j-th coordinate > 0. If the i-th coordinate of π_f is not 0, then $y_i(\pi_f)$ returns the number of j satisfying both (i) $j < i$ and (ii) the absolute value of the j-th coordinate of π_f and the absolute value of the i-th coordinate of π_f are not equal. For example, if $\pi_f = (1,2,-2,1,1,0,0,1,2,-2)$, then $y_6(\pi_f) = 4$ and $y_{10}(\pi_f) = 6$.

The proof of this proposition is provided at the end of this subsection.

We note that computing the right-hand-side of (41) requires searching over all possible permutations π_f. The following proposition shows that when focusing on the case of $\alpha = d\beta$, one can further simplify the expression.

Proposition 11: Consider any $(n,\hat{k},d,r)$, we have that

$$R_{\text{MFHS}}(d\beta,\beta) = \sum_{i=1}^{\hat{k}} (d-y_i(\pi_f^*))^+\beta, \tag{42}$$

where π_f^* is the RFIP.

The proof of Proposition 11 is relegated to Appendix D.

It is worth emphasizing that Propositions 10 and 11 are based on the analysis of the underlying IFGs. Namely, we do not have any explicit code construction[15] for these two propositions and rely exclusively on the min-cut analysis with the assumption of random code construction. We note that for some class of $(n,\hat{k},d,r)$ combinations, it is possible to derive explicit code constructions without relying on this random-linear-network-coding-based assumption. The code constructions are rather involved and for that reason we omit them from this work and provide them in [2].

Proposition 11 directly implies Proposition 9, the achievability part of Theorem 2. The reason is as follows. We first notice that by the definition of $y_i(\cdot)$, we always have $y_i(\pi_f^*) \leq i-1$. Suppose $\min(d+1,\hat{k}) > \left\lceil \frac{n}{n-d-r} \right\rceil$ and consider the MFHS scheme. Since there are exactly $\left\lceil \frac{n}{n-d-r} \right\rceil$ number of families (including both complete and incomplete families), among the first $\min(d+1,\hat{k})$ indices of a family index permutation π_f^* there is at least one family index that is repeated. Say the j_1-th and the j_2-th coordinates of π_f^* are of the same value where $j_1, j_2 \leq \min(d+1,\hat{k})$. Without loss of generality, we assume $j_1 < j_2$. Then, by the definition of $y_i(\cdot)$, we have $y_{j_2}(\pi_f^*) < j_2 - 1$ with a strict inequality since the j_1-th coordinate of π_f^* will not contribute to $y_{j_2}(\pi_f^*)$. Letting $q = \min(d+1,\hat{k})$, we thus have

$$\begin{aligned} R_{\text{MFHS}}(d\beta,\beta) &= \sum_{i=1}^{\hat{k}} (d-y_i(\pi_f^*))^+\beta \qquad (43)\\ &= \sum_{i=1}^{q} (d-y_i(\pi_f^*))\beta + \sum_{i=q+1}^{\hat{k}} (d-y_i(\pi_f^*))^+\beta \qquad (44)\\ &> \sum_{i=1}^{q} (d-(i-1))\beta + \sum_{i=q+1}^{\hat{k}} (d-(i-1))^+\beta \qquad (45)\\ &= \sum_{i=1}^{\hat{k}} (d-(i-1))^+\beta \qquad (46)\\ &= R_{\text{BHS}}(d\beta,\beta), \qquad (47) \end{aligned}$$

where (44) follows from $y_i(\pi_f^*) \leq (i-1)$ so that we can remove the $()^+$ when $i = 1$ to $\min(d+1,\hat{k})$ without changing the value; (45) follows from that $y_{j_2}(\pi_f^*) < (j_2 - 1)$ and that $y_i(\pi_f^*) \leq (i-1)$ for arbitrary i; (46) follows from $(d-(i-1)) \geq 0$ for all $i = 1$ to $\min(d+1,\hat{k})$; and (47) follows by (15).

By (47), we have completed the proof of Proposition 9. Before closing this subsection with the proof of Proposition 10, it is worth mentioning the following result on the MBR point of the MFHS.

Corollary 2: Consider any $(n,\hat{k},d,r)$ values and a fixed file size $\mathcal{M}$. The MBR point of the MFHS scheme is

$$\alpha_{\text{MBR}} = d\beta_{\text{MBR}} = \frac{d\mathcal{M}}{\sum_{i=1}^{\hat{k}} (d-y_i(\pi_f^*))^+}, \tag{48}$$

where π_f^* is the RFIP.

Proof: It is not hard to see that at the MBR point we have that $\alpha_{\text{MBR}} = d\beta_{\text{MBR}}$ since we are minimizing β_{MBR} at fixed file size $\mathcal{M}$. We can then use (42) to get (48). ■

Proof of Proposition 10: Recall that the MFHS scheme specifies the helper candidate set $\overline{D}(i)$ for nodes $i = 1$ to n based on the concepts of complete and incomplete families. In the following discussion, we assume that the helper candidate set $\overline{D}(i)$ is generated by the given MFHS scheme.

Using the same proof technique of [6, Lemma 2], we can get the following lower bound on the smallest possible mincut

[15]In contrast, the code existence result in Section V-B is in the strongest sense since we provide an explicit binary code construction and then directly analyze its performance (see Proposition 6) without using any min-cut analysis.

of an IFG generated by the MFHS scheme denoted by F

$$\min_{G\in\mathcal{G}_F}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}(s,t)\geq$$
$$\min_{\mathbf{p}\in\mathcal{P}}\sum_{i=1}^{\hat{k}}\min((d-z_i(\mathbf{p}))^+\beta,\alpha),\qquad(49)$$

where $\mathbf{p}$ is a $\hat{k}$-dimensional integer-valued vector, $\mathcal{P}=\{(p_1,p_2,\cdots,p_{\hat{k}}):\forall i\in\{1,\cdots,\hat{k}\},1\leq p_i\leq n\}$ and $z_i(\mathbf{p})=|\{p_j:j<i,p_j\in\overline{D}(p_i)\}|$. For example, suppose $(n,\hat{k},d,r)=(6,4,2,1)$, $\overline{D}(3)=\{1,4,5\}$, and $\mathbf{p}=(1,2,1,3)$. Since $p_4=3$, we have $z_4(\mathbf{p})=|\{p_j:j<4,p_j\in\overline{D}(3)\}|=1$. (The double appearances of $p_1=p_3=1$ are only counted as one.)

The main intuition behind (49), is that for any source s and data collector t, we consider the min-cut (V,V^c) separating t from s. That is, V and V^c form a partition of the nodes in the IFG; $s\in V$ and $t\in V^c$; and the edges from V to V^c is the minimum edge cut. Since $t\in V^c$, set V^c contains at least $\hat{k}$ intermediate nodes (the nodes $\neq t$). Denote the $\hat{k}$ oldest intermediate nodes in V^c by u_1 to u_k. We denote the node index of each intermediate node u_i by $NI(u_i)$. Note that some u_i and u_j may have the same index $NI(u_i)=NI(u_j)$ since u_i and u_j are intermediate nodes in the IFG, not the actual physical nodes. We then choose the $\mathbf{p}$ vector by $\mathbf{p}=(NI(u_1),NI(u_2),\cdots,NI(u_k))$.

With the above construction, if we examine the definition of $z_i(\mathbf{p})$ in (49), we can easily see that the function $z_i(\mathbf{p})$ returns an *upper bound* of the number of edges entering $u_{i,\text{in}}$ in the IFG from some $u_{j,\text{out}}$ satisfying $j<i$. Therefore, $(d-z_i(\mathbf{p}))^+$ represents a *lower bound* of the number of edges entering $u_{i,\text{in}}$ that are *not* from $u_{j,\text{out}}$ with $j<i$. Therefore, each u_i will contribute at least $\min((d-z_i(\mathbf{p}))^+\beta,\alpha)$ to the min-cut value. By summing over all u_i, we have (49). Since the analysis is quite standard, see [6, Lemma 2], we omit the detailed proof of (49). Interested readers can find a more detailed discussion in [5, Lemma 8] and the corresponding proof in [5, Appendix G-A].

Next, we will prove that

$$\min_{G\in\mathcal{G}_F}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}_G(s,t)\leq$$
$$\min_{\forall\pi_f}\sum_{i=1}^{\hat{k}}\min((d-y_i(\pi_f))^+\beta,\alpha).\qquad(50)$$

The reason is the following. Denote the smallest IFG in $\mathcal{G}_F(n,\hat{k},d,r,\alpha,\beta)$ by G_0. Specifically, all its nodes are intact, i.e., none of its nodes has failed before. Denote its active nodes arbitrarily by $1,2,\cdots,n$. Consider the family index permutation of the MFHS scheme F that attains the minimal value of the right-hand-side of (50) and call it $\tilde{\pi}_f$. Find a vector of node indices $\tilde{\mathbf{p}}$ such that (i) $FI(\tilde{p}_i)=\tilde{\pi}_f(i)$ for $i=1$ to n and (ii) $\tilde{p}_i\neq\tilde{p}_j$ if $i\neq j$. This is always possible since the family index permutation $\tilde{\pi}_f$ can be viewed as transcribing some node index vector $\tilde{\mathbf{p}}$ to the corresponding family indices.

After constructing $\tilde{\mathbf{p}}$, we fail each active node in $\{1,2,\cdots,n\}$ of G_0 exactly once starting by failing node $\tilde{p}_1$ to node $\tilde{p}_n$. Along this failing process, at each step of repair, say we are now repairing $\tilde{p}_i$, we choose the unavailable nodes U_i as follows. Among the $(d+r)$ nodes in $\overline{D}(\tilde{p}_i)$, we first sort them according to their locations in the node index vector $\tilde{\mathbf{p}}$. Namely, if both nodes $\tilde{p}_{j_1}$ and $\tilde{p}_{j_2}$ belong to $\overline{D}(\tilde{p}_i)$, then we say $\tilde{p}_{j_1}$ is ahead of $\tilde{p}_{j_2}$ if $j_1<j_2$. Once we have sorted the $(d+r)$ nodes in $\overline{D}(\tilde{p}_i)$, we let the *last* r nodes of $\overline{D}(\tilde{p}_i)$ be temporarily unavailable during the repair of node $\tilde{p}_i$. Therefore, the helpers of the newcomer $\tilde{p}_i$ must be, according to the locations in the vector $\tilde{\mathbf{p}}$, the first d nodes of $\overline{D}(\tilde{p}_i)$. After repairing all n nodes according to the above description, we denote the final IFG as graph G'.

We use the following example to demonstrate the above failing/repair process. Let $(n,d,r)=(8,4,1)$ and suppose the minimizing family index permutation is $\tilde{\pi}_f=(1,2,1,-2,0,0,1,2)$. Then, a possible $\tilde{\mathbf{p}}$ is $\tilde{\mathbf{p}}=(1,4,2,6,7,8,3,5)$, which satisfies $(FI(\tilde{p}_1),FI(\tilde{p}_2),\cdots,FI(\tilde{p}_n))=\tilde{\pi}_f$. Using the permutation $\tilde{\mathbf{p}}$, we fail nodes 1, 4, 2, 6, 7, 8, 3, and 5 in this sequence. To illustrate how we choose the unavailable node set U_i when failing node $\tilde{p}_i$, consider the fourth repair operation, for which node $\tilde{p}_4=6$ fails and we want to repair it. Recall that node 6 belongs to the second complete family $\{4,5,6\}$. Therefore, $\overline{D}(6)=\{1,2,3,7,8\}$. We sort $\overline{D}(6)$ according to their locations in $\tilde{\mathbf{p}}$ and we thus have $\overline{D}(6)=\{1,2,7,8,3\}$. Therefore, we assume $U=\{3\}$ is unavailable and the helper nodes of node 6 are $\{1,2,7,8\}$. Another example is when repairing node $\tilde{p}_6=8$, i.e., the sixth repair operation. Since node 8 belongs to an incomplete family, the corresponding helper candidate set is $\overline{D}(8)=\{1,2,3,4,5\}$. After sorting, we have $\overline{D}(8)=\{1,4,2,3,5\}$. Therefore, we make node 5 be temporarily unavailable when repairing node 8, and the actual helpers of node 8 become $\{1,4,2,3\}$. Note that $\tilde{\mathbf{p}}$ may not be unique in our construction. For example, $\tilde{\mathbf{p}}=(3,5,2,6,8,7,1,4)$ is also a possible permutation satisfying $(FI(\tilde{p}_1),FI(\tilde{p}_2),\cdots,FI(\tilde{p}_n))=\tilde{\pi}_f$. Our construction holds for any arbitrary choice of $\tilde{\mathbf{p}}$.

Consider a data collector t in G' that connects to the oldest $\hat{k}$ newcomers, i.e., nodes $\tilde{p}_1$ to $\tilde{p}_{\hat{k}}$. We now analyze the cut-value between the root s and the data collector t using the same arguments as in [6, Lemma 2]. Consider a special cut (V,V^c) between t and s that are constructed as follows. Initially, we set $V^c=\{t\}$ containing only the data collector. Then, for each $i\in\{1,\ldots,\hat{k}\}$, if $\alpha\leq(d-y_i(\tilde{\pi}_f))^+\beta$ then we add $x_{\text{out}}^{\tilde{p}_i}$ to V^c. Namely, the *out half* of node $\tilde{p}_i$ is added to V^c; Otherwise, we include both $x_{\text{out}}^{\tilde{p}_i}$ and $x_{\text{in}}^{\tilde{p}_i}$ in V^c. With the above construction of V^c, it is not hard to see that the cut-value of the cut (V,V^c) is equal to $\sum_{i=1}^{\hat{k}}\min((d-y_i(\tilde{\pi}_f))^+\beta,\alpha)$.

Since the left-hand side of (50) further takes the minimum over $\mathcal{G}_F$ and all data collectors t, we have proven the inequality (50).

Thus far, we have proven that

$$\min_{\mathbf{p}\in\mathcal{P}}\sum_{i=1}^{\hat{k}}\min((d-z_i(\mathbf{p}))^+\beta,\alpha)\leq \min_{G\in\mathcal{G}_F}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}_G(s,t) \leq \min_{\forall\pi_f}\sum_{i=1}^{\hat{k}}\min((d-y_i(\pi_f))^+\beta,\alpha). \quad (51)$$

The remaining step is to prove that

$$\min_{\mathbf{p}\in\mathcal{P}}\sum_{i=1}^{\hat{k}}\min((d-z_i(\mathbf{p}))^+\beta,\alpha)\geq \min_{\forall\pi_f}\sum_{i=1}^{\hat{k}}\min((d-y_i(\pi_f))^+\beta,\alpha). \quad (52)$$

Once we prove (52), we have (41) since (51) is true. We prove (52) by showing that for any $\mathbf{p}\in\mathcal{P}$ we can find a $\hat{\pi}_f$ such that[16]

$$z_i(\mathbf{p})\leq y_i(\hat{\pi}_f),\quad \forall i=1,\cdots,\hat{k}. \quad (53)$$

One can clearly see that the existence of $\hat{\pi}_f$ satisfying (53) for any $\mathbf{p}\in\mathcal{P}$ immediately implies (52).

In our previous work [5], we have proven that (53) holds for the case of $r=0$ and arbitrary $(n,\hat{k},d)$ values. We will now prove that (53) holds for arbitrary $(n,\hat{k},d,r)$ values.

A closer look at the definition of $z_i(\cdot)$ in the proof of Proposition 10 shows that different parameter values $(n,\hat{k},d,r)$ and $(n',\hat{k}',d',r')$ can still lead to the same function $z_i(\cdot)$ for $i=1$ to n, provided we have $n=n'$ and $d+r=d'+r'$. The reason is as follows. Suppose we apply the MFHS scheme to two different scenarios $(n,\hat{k},d,r)$ and $(n',\hat{k}',d',r')$. If we have $n=n'$, then the total number of nodes is the same for both scenarios. Since each complete family contains $n-(d+r)$ nodes, if we also have $(d+r)=(d'+r')$, then MFHS will divide the nodes into families in the same way for both scenarios. Since a newcomer requests help from outside its own family in MFHS, the helper candidate set $\overline{D}(i)$ will again be the same for both scenarios. Notice that the definition of $z_i(\cdot)$ in the proof of Proposition 10 depends only on the helper candidate set $\overline{D}(j)$ and the $z_i(\cdot)$ function will be identical in both scenarios.

We now argue that if two scenarios $(n,\hat{k},d,r)$ and $(n',\hat{k}',d',r')$ satisfy $n=n'$ and $d+r=d'+r'$, then the $y_i(\cdot)$ function in Proposition 10 will again be the same for both scenarios. The reason is as follows. By comparing the definitions of $y_i(\pi_f)$ and $z_i(\mathbf{p})$, one can quickly see that if we choose the family index permutation π_f and the $\mathbf{p}$ that satisfy $\pi_f=(FI(p_1),\cdots,FI(p_n))$, then

$$y_i(\pi_f)=z_i(\mathbf{p}),\quad \forall i=1,\cdots,\hat{k}. \quad (54)$$

[16]We note that $\hat{\pi}_f$ does not necessarily have to satisfy $\hat{\pi}_f=(FI(p_1),FI(p_2),\cdots,FI(p_n))$, and in fact $\hat{\pi}_f=(FI(p_1),FI(p_2),\cdots,FI(p_n))$ is not always possible. For illustration, consider $\mathbf{p}=(1,1,1,1,\cdots,1)$, which is a legitimate choice of $\mathbf{p}\in\mathcal{P}$. However, for such $\mathbf{p}$ it is impossible to find a family index permutation satisfying $\hat{\pi}_f=(FI(p_1),FI(p_2),\cdots,FI(p_n))$ since the vector $(FI(p_1),FI(p_2),\cdots,FI(p_n))$ is not a family index permutation.

Namely, $y_i(\pi_f)$ can be viewed as a *transcribed* version of $z_i(\mathbf{p})$ from the node index $\mathbf{p}$ to a family index π_f if $\mathbf{p}\in\mathcal{P}$. Since we have shown that $z_i(\cdot)$ will be the same for both scenarios and since when the $(d+r)=(d'+r')$ the node index to family index transcription will be identical for both scenarios, $y_i(\cdot)$ will also be identical for both scenarios.

Consider any arbitrarily given $(n,\hat{k},d,r)$ and use it to generate another scenario $(n',\hat{k}',d',r')$ satisfying $n'=n$, $\hat{k}'=\hat{k}$, $d'=d+r$ and $r'=0$. Consider an arbitrarily chosen $\mathbf{p}\in\mathcal{P}$. For the scenario of $(n',\hat{k}',d',r')$, since $r'=0$, our previous results in [5] show that there exists a $\hat{\pi}_f$ satisfying (53). Since the above paragraphs have proven that the functions $z_i(\cdot)$ and $y_i(\cdot)$ are identical for both scenarios (n,k,d,r) and (n',k',d',r'), the $\hat{\pi}_f$ that satisfies (53) for (n',k',d',r') must also satisfy (53) for the given (n,k,d,r) as well. The proof of Proposition 10 is thus complete. ■

VII. Conclusion

We have shown that stationary helper selection (SHS) can be strictly suboptimal by carefully constructing an optimal binary code for $(n,k,d,r)=(5,3,2,1)$ and $(5,4,2,1)$ based on *dynamic helper selection* (DHS), where r represents the maximum number of nodes that can be temporarily unavailable. For general $(n,\hat{k},d,r)$ values, we have answered the question whether SHS/DHS can outperform blind helper selection (BHS) or not, for a vast majority of $(n,\hat{k},d,r)$ values. The results thus provide valuable guidelines for each $(n,\hat{k},d,r)$ whether it is beneficial to spend time and design new SHS/DHS schemes or whether one should simply use the basic BHS.

Appendix A
The Information Flow Graph

We provide in this appendix the description of the information flow graph (IFG) that was first introduced in [6]. This appendix follows the same IFG description as in [5].

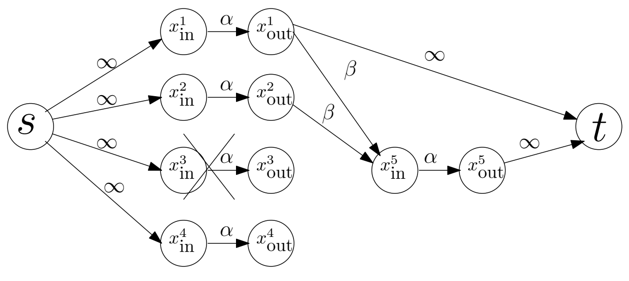

Fig. 5. An example of the information flow graph with $(n,\hat{k},d)=(4,2,2)$.

As shown in Fig 5, 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 $\hat{k}$ active storage nodes, which represents the party that is interested in extracting the original data object initially produced by the source s. Fig. 5 illustrates one such data collector, denoted by t, which connects to $\hat{k} = 2$ storage nodes.

The IFG evolves with time. In the first stage of an IFG, 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. 5 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 storage nodes 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. 5, the newcomer connects to its helpers by edges of capacity β resembling the amount of data communicated from each helper. 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. 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 $\binom{n}{\hat{k}}$ *data collector nodes* in G [6]. Each data collector $t \in \mathrm{DC}(G)$ represents one unique way of choosing $\hat{k}$ out of n active nodes when reconstructing the file.

Appendix B Proof of Proposition 3

The statement that $C_{\mathrm{DHS}}(\alpha, \beta) = R_{\mathrm{BHS}}(\alpha, \beta)$ for all (α, β) if (19) holds is a restatement of Theorem 2. We now prove that under the additional assumption $d = r = 1$, $C_{\mathrm{DHS}}(\alpha, \beta) = R_{\mathrm{BHS}}(\alpha, \beta)$ for all (α, β) if either $\hat{k} = 3$ or if $\hat{k} = 4$ and $n \bmod 3 \neq 0$.

We first give the following definition of an m-tree that will be useful in our proof.

Definition 10: Consider $(n, \hat{k}, d, r)$ such that $d = 1$. Consider an IFG G and a set of m active nodes of G denoted by $x^1, x^2, \ldots, x^m$. The set of m active nodes $\{x^1, \ldots, x^m\}$ is said to be an m-tree if the following two properties hold simultaneously: (a) For $i, j \in \{1, 2 \cdots, m-1\}$ and $j > i$, x^i is repaired before x^j; (b) for any $i = 2$ to m, there exists a node $b \in \{1, \cdots, i-1\}$ such that $(x^b_{\text{out}}, x^i_{\text{in}})$ is an edge in G. The reason that we call the above m nodes an m-tree is because since $d = 1$, there is exactly 1 edge entering each node x^i_{in}. The above condition (b) thus implies that each node x^i_{in} is *connected* to one of the previous nodes x^1_{out} to x^{i-1}_{out}. Therefore, these m nodes form a tree.

We first consider the case of $\hat{k} = 3$, and we state the following claim.

Claim 1: Consider $(n, \hat{k}, d, r)$ parameters that satisfy that $d = r = 1$. For any given DHS scheme A and the corresponding collection of IFGs $\mathcal{G}_A$, we can always find a $G^* \in \mathcal{G}_A$ such that there exists a 3-tree in its set of active nodes.

We now use the above claim to prove $C_{\mathrm{DHS}}(\alpha, \beta) = R_{\mathrm{BHS}}(\alpha, \beta)$ if $\hat{k} = 3$. Suppose the above claim is true. We let t^* denote the data collector that is connected to the 3-tree. By properties (a) and (b) in Definition 10 of a 3-tree, we can see that node x^1 is a vertex-cut separating source s and the data collector t^*. The min-cut value separating s and t^* thus satisfies $\operatorname{mincut}_{G^*}(s, t^*) = \min(\beta, \alpha)$ for the given $G^* \in G_A$ and the specific choice of t^*. Also note that $\min(\beta, \alpha) = \sum_{i=0}^{\hat{k}-1} \min((d-i)^+\beta, \alpha)$ since we assume $d = 1$ and $\hat{k} = 3$. Combining both, we thus have $\operatorname{mincut}_{G^*}(s, t^*) = \sum_{i=0}^{\hat{k}-1} \min((d-i)^+\beta, \alpha)$. Since this is equal to R_{BHS} in (15), we thus have that $C_{\mathrm{DHS}}(\alpha, \beta) = R_{\mathrm{BHS}}(\alpha, \beta)$ for all (α, β) when $\hat{k} = 3$ holds.

Proof of Claim 1: We prove Claim 1 by explicit construction. Start from any $G \in \mathcal{G}_A$ with all n nodes have been repaired at least once. We choose one arbitrary 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 A will choose a 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 $G^{(1)}$. By our construction, $x^{(1)}$ as an existing active node is repaired before the newcomer $y^{(1)}$ and there is an edge $(x^{(1)}_{\text{out}}, y^{(1)}_{\text{in}})$ in $G^{(1)}$.

Now starting from $G^{(1)}$, we choose another $w^{(2)}$, which is not one of $x^{(1)}$ and $y^{(1)}$ and let this node fail. We use $y^{(2)}$ to denote the newcomer that replaces $w^{(2)}$. The helper selection scheme A will again choose a helper node based on the history of the failure pattern. We denote the new IFG (after the helper selection chosen by scheme A) as $G^{(2)}$. If the helper node of $y^{(2)}$ is $x^{(1)}$ or $y^{(1)}$, then the three nodes $(x^{(1)}, y^{(1)}, y^{(2)})$ are the (x^1, x^2, x^3) nodes satisfying properties (a) and (b) of Definition 10 of a 3-tree. In this case, we can stop our construction and let $G^* = G^{(2)}$ and we say that the construction is complete in the second round. Suppose the above case is not 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 $\{x^{(m)}, y^{(m)} : \forall m = 1, 2\}$. Again, let $y^{(3)}$ denote the newcomer that replaces $w^{(3)}$ and the scheme A 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 $x^{(m)}$ or $y^{(m)}$ for some $m = 1, 2$, then the three nodes $(x^{(m)}, y^{(m)}, y^{(3)})$ are the (x^1, x^2, x^3) nodes in Definition 10 satisfying properties (a) and (b). In this case, we can stop our construction and let $G^* = G^{(3)}$ and we say that the construction is complete in the third round. If the above case is not true, then we denote the helper of $y^{(3)}$ by $x^{(3)}$, and repeat this process for the fourth time and so on.

If the construction is not complete in the m-th round for some $m \leq \lceil \frac{n}{2} \rceil - 1$, we can always start the $(m+1)$-th round since out of the n nodes, we can always find another node $w^{(m+1)}$ that is none of $\{x^{(m')}, y^{(m')} : \forall m' = 1, 2, \ldots, m\}$. Now, suppose that n is odd and the construction is not complete after $m_0 = \lceil \frac{n}{2} \rceil - 1$ rounds. In this case, there is only 1 remaining node that is not inside $\{x^{(m)}, y^{(m)} : \forall m = 1, 2, \ldots, m_0\}$. Denote that node as $w^{(m_0+1)}$. Fail $w^{(m_0+1)}$ and replace it by $y^{(m_0+1)}$. Since $\{x^{(m)}, y^{(m)} : \forall m = 1, 2, \ldots, m_0\}$ and $y^{(m_0+1)}$ cover all n nodes, the helper node of $y^{(m_0+1)}$ must be one of the nodes in $\{x^{(m)}, y^{(m)} : \forall m = 1, 2, \ldots, m_0\}$. If the helper node of $y^{(m_0+1)}$ is $x^{(m')}$ or $y^{(m')}$ for some $m' = 1, 2, \ldots, m_0$, then the three nodes $(x^{(m')}, y^{(m')}, y^{(m_0+1)})$ form a 3-tree satisfying properties (a) and (b) of Definition 10.

For the case when n is even and the construction is not complete after $m_0 = \lceil \frac{n}{2} \rceil - 1$ rounds, there are 2 remaining nodes that are not inside $\{x^{(m)}, y^{(m)} : \forall m = 1, 2, \ldots, m_0\}$. Choose arbitrarily one of them and denote that node as $w^{(m_0+1)}$. Fail $w^{(m_0+1)}$ and replace it by $y^{(m_0+1)}$ while having the other remaining node (the one that is not $w^{(m_0+1)}$) temporarily unavailable when repairing $y^{(m_0+1)}$. Therefore, we have forced $y^{(m_0+1)}$ to connect to an $x^{(m')}$ or $y^{(m')}$ node for some $m' = 1, 2, \ldots, m_0$. Similar to the case for which n is odd, the three nodes $(x^{(m')}, y^{(m')}, y^{(m_0+1)})$ form a 3-tree. The proof of Claim 1 is complete. ■

Now, we turn our attention to the case when $\hat{k} = 4$ and $n \bmod 3 \neq 0$. Similarly, we state the following claim.

Claim 2: Consider $(n, \hat{k}, d, r)$ parameters that satisfy that $d = r = 1$ and $n \bmod (3) \neq 0$. For any given DHS scheme A and the corresponding collection of IFGs $\mathcal{G}_A$, we can always find a $G^{**} \in \mathcal{G}_A$ such that there exists a 4-tree in its set of active nodes.

We now use the above claim to prove BHS is optimal if $\hat{k} = 4$ and $n \bmod 3 \neq 0$. Suppose the above Claim 2 is true. As we did above, we let t^{**} denote the data collector that is connected to the 4-tree. By properties (a) and (b) of the definition of a 4-tree we can again see that node x^1 is a vertex-cut separating source s and the data collector t^{**}. The min-cut value separating s and t^{**} thus satisfies $\text{mincut}_{G^{**}}(s, t^{**}) \leq \min(d\beta, \alpha) = \sum_{i=0}^{\hat{k}-1} \min((d-i)^+\beta, \alpha)$ for $G^{**} \in G_A$ and the specific choice of t^{**}, where the inequality, as discussed before, follows from x^1 being a vertex-cut separating s and t^{**} and the equality follows from the assumption that $d = 1$ and $\hat{k} = 4$. We thus have that $C_{\text{DHS}}(\alpha, \beta) = R_{\text{BHS}}(\alpha, \beta)$ for all (α, β) when $\hat{k} = 4$ and $n \bmod 3 \neq 0$.

Proof of Claim 2: We prove this claim by explicit construction. The construction contains 2 phases. The goal of Phase 1 is to convert all nodes to be either part of a 2-set or part of a 3-tree. We start from time 1, when no node has ever been repaired. Let V denote a subset of nodes. Initially, set $V = \emptyset$. We arbitrarily choose one node in $\{1, \cdots, n\} \backslash V$, say node w. We fail node w. The newcomer is denoted by y and we let x denote its helper. If the helper x is already in V, then we add y to V. If x is not in V, we add both $\{x, y\}$ to V. Repeat the above process by choosing a new $w \in \{1, \cdots, n\} \backslash V$ until $\{1, \cdots, n\} \backslash V = \emptyset$.

Consider two possibilities. If the resulting IFG contains a 4-tree, then our construction is complete. If not, then we argue that all the nodes in V (n nodes in total when the construction terminates) must either be in a 2-set or in a 3-tree but cannot be in both. We prove this by induction with the induction assumption being that set V contains only disjoint 2-sets or 3-trees during the construction. Consider our iterative construction, for which we choose a node $w \in \{1, \cdots, n\} \backslash V$ and replace it by a newcomer y with the corresponding helper being x. If $x \notin V$ already, after adding $\{x, y\}$ to V the new pair (x, y) will form a 2-set that is disjoint to all the previous nodes in V. The induction assumption holds. If $x \in V$ already, we claim that x must be part of a 2-set. The reason why x cannot be part of a 3-tree is that if so, then the 3-tree plus the newcomer y will form a 4-tree and we have already ruled out such a possibility by focusing on the case for which the construction does not lead to any 4-tree. Since $x \in V$ is part of a 2-set, after adding node y to V, node y will be part of a 3-set and the induction assumption still holds.

We are now ready for Phase 2. Recall that after Phase 1 all n nodes have been partitioned to be a collection of disjoint 2-sets and 3-trees. Pick arbitrarily a 2-set in the active nodes of $G^{(1)}$. This is always possible since $n \bmod 3 \neq 0$, which implies that the n nodes cannot all be 3-trees. Denote the chosen 2-set by (v, w). Fail node w and during its repair let v be unavailable. Call the newcomer w'. If w' connects to a node that belongs to a 3-tree, then the 3-tree and the newcomer w' form a 4-tree. The construction is thus finished/terminated.

If w' connects to a node that belongs to a 2-set, then the 2-set and w' now form a 3-tree. Namely, we have converted w, part of a 2-set, to a new node w' being part of a 3-tree. We then fail node v and replace it by a newcomer v'. Similarly, if v' connects to a node that belongs to a 3-tree, then the 3-tree and the newcomer v' form a 4-tree. The construction is finished/terminated. If v' connects to a node that belongs to a 2-set, then the 2-set and v' now form a 3-tree. Specifically, we have converted v, part of a 2-set, to a new node v' being part of a 3-tree. One can see that the above procedure removes the 2-set (v, w) from the IFG and replaces it by v' and w' that participate in two different 3-trees.

We then iteratively repeat the above process to convert all 2-sets into 3-trees. This is always possible since $n \bmod 3 \neq 0$, which implies that the n nodes cannot all be 3-trees. Nonetheless, we cannot repeat this process indefinitely since each round will remove one 2-set and we only have finitely many 2-sets. This implies that the process must terminate after some finite rounds. Specifically, either w' or v' must eventually be connected to a 3-tree and we will have a 4-tree at the end of this construction. The proof of Claim 2 is thus complete. ■

Thus far, we have proven the converse part of Proposition 3. Now, we notice that, under the assumption of $d = r = 1$, the statement "none of (i)-(iii) holds" is equivalent to "at least one of the following conditions holds: (a) $n \bmod 3 = 0$ and $\hat{k} = 4$, or (b) $\hat{k} \geq 5$." The reason is by the simple observation that when focusing on $d = r = 1$, condition (i) is equivalent to "$\hat{k} \leq 2$" since $\left\lceil \frac{n-r}{n-d-r} \right\rceil = 2$. Therefore, not satisfying (i) to (iii) is equivalent to satisfying one of conditions (a) and (b).

In the following, we will prove that $C_{\text{SHS}}(\alpha, \beta) > R_{\text{BHS}}(\alpha, \beta)$ for some (α, β) when the $(n, \hat{k}, d, r)$ parameters satisfy at least one of conditions (a) or (b). Suppose condition (a) is satisfied. We prove that by proving the existence of a SHS scheme by explicit construction that can protect a file of size larger than $R_{\text{BHS}}(\alpha, \beta)$ for some (α, β).

Our construction is as follows. Since $n \bmod 3 = 0$, we can divide n nodes into $\frac{n}{3}$ groups of 3 nodes. Suppose the file to be protected has size $\mathcal{M}$. We first divide the file into 2 *packets*, each of size $\mathcal{M}/2$. We call each packet the *systematic* packet, which is analogous to the concept of systematic bits in error control coding. We then use an $(\frac{n}{3}, 2)$ MDS code to protect the systematic packets by adding $\frac{n}{3} - 2$ *parity* packets. Finally, each group of 3 nodes is associated with one distinct packet (can be either systematic or parity packets). Each packet is then duplicated 3 times and all 3 nodes in the same group will store an identical copy of the packet of that group.

We argue that such a system can satisfy $(n, \hat{k}, d, r, \alpha, \beta)$ with $\hat{k} = 4$, $d = r = 1$, $\alpha = \beta = \frac{\mathcal{M}}{2}$. The reason is as follows. $\alpha = \frac{\mathcal{M}}{2}$ since each node only stores 1 packet of size $\frac{\mathcal{M}}{2}$. Since $\hat{k} = 4 > 3$, any $\hat{k}$ nodes must belong to at least 2 different groups and the nodes in these ≥ 2 groups must collectively contain ≥ 2 distinct packets. Because we use an $(\frac{n}{3}, 2)$ MDS code to protect the file, one can reconstruct the original file by accessing any $\hat{k} = 4$ nodes. We now consider the repair operation. Suppose a node fails and we consider the other 2 nodes of the same group. Since $r = 1$, at least one of the other two nodes must still be available. The newcomer can thus ask the remaining available node to transfer the packet it stores to the newcomer. Therefore, exact-repair can be achieved with $d = r = 1$ and $\beta = \frac{\mathcal{M}}{2}$. Since the capacity of *all* SHS schemes must be no smaller than the rate of this particular SHS scheme, we have $C_{\text{SHS}}(\alpha, \beta) \geq \mathcal{M}$ when $\alpha = \beta = \frac{\mathcal{M}}{2}$.

We now compare the performance to the BHS scheme for the same $(n, \hat{k}, d, r)$ parameter values. By (15), when $d = r = 1$ and $\hat{k} = 4$, we have that

$$R_{\text{BHS}}(\alpha, \beta) = \sum_{i=0}^{\hat{k}-1} \min((d-i)^+\beta, \alpha) = \min(\beta, \alpha). \quad (55)$$

One can clearly see that our parameter values $\alpha = \beta = \frac{\mathcal{M}}{2}$ lead to $R_{\text{BHS}}(\alpha, \beta) = \frac{\mathcal{M}}{2}$. As a result, $C_{\text{SHS}}(\alpha, \beta) > R_{\text{BHS}}(\alpha, \beta)$ for $(\alpha, \beta) = (\frac{\mathcal{M}}{2}, \frac{\mathcal{M}}{2})$.

Suppose now that condition (b) is satisfied, i.e., $\hat{k} \geq 5$. Our construction is almost identical to the scheme we described for condition (a). That is, depending on the value of $n \bmod 3$, we can either divide the n nodes into $\frac{n}{3}$ group of 3 nodes; or $\frac{n-1}{3} - 1$ groups of 3 nodes plus 1 group of 4 nodes; or $\frac{n-2}{3} - 2$ groups of 3 nodes plus 2 groups of 4 nodes. Regardless of which case we are in, we again divide the file of size $\mathcal{M}$ into 2 packets, each of size $\frac{\mathcal{M}}{2}$. Then we protect the two packets by an $(\lfloor \frac{n}{3} \rfloor, 2)$ MDS code. Associate each group with one coded packet and let the nodes of each group store an identical copy of that packet. Since every group has at most 4 nodes, any $\hat{k} \geq 5$ nodes must belong to at least two different groups. Since any 2 packets can be used to recover the original file, the proposed scheme can reconstruct the original file from any $\hat{k}$ nodes. By similar reasons as before, exact-repair can also be achieved with $d = r = 1$ and $\beta = \frac{\mathcal{M}}{2}$. We now compare the performance to the BHS scheme for the same $(n, \hat{k}, d, r)$ parameter values. By (15), $R_{\text{BHS}}(\alpha, \beta)$ when $d = r = 1$ and $\hat{k} = 5$ is again (55). The above scheme with $\alpha = \beta = \frac{\mathcal{M}}{2}$ thus implies that $C_{\text{SHS}}(\alpha, \beta) > R_{\text{BHS}}(\alpha, \beta)$ for $(\alpha, \beta) = (\frac{\mathcal{M}}{2}, \frac{\mathcal{M}}{2})$. The proof of Proposition 3 is hence complete.

Appendix C
Proofs of Lemma 1 and Proposition 5

In this appendix, we prove both Lemma 1 and Proposition 5 simultaneously.

The proof is by induction with the following two induction conditions: (A) if node i is the parent of node j, then jointly nodes i and j contain 3 linearly independent packets, (B) if neither i is a parent of j nor j is a parent of i, then jointly nodes i and j contain 4 linearly independent packets. Let time $\tau = 0$ be the stage where the storage network is still intact, i.e., no node has failed before. By checking all $\binom{5}{2}$ pairs of nodes, we can easily see that the initial code in Fig. 2 satisfies the induction conditions (A) and (B) at $\tau = 0$. In the following, we first show that conditions (A) and (B) guarantee that we can always find packets (Z_b^*, Z_c^*) using the regular repair operations and that the induction conditions (A) and (B) will remain satisfied for the new code.

Now, let us assume that the induction conditions (A) and (B) are satisfied until time $\tau = \tau_0 - 1$ and, using the same notation as above, node a is the failed node with nodes $\{b, c\}$ selected as helpers by the TA scheme. We also use the same notation to denote the two packets in node i by $(Y_1^{(i)}, Y_2^{(i)})$. We introduce some vector notation to aid in this proof. We first let $\mathbf{m}$ be a 4×1 column vector holding the 4 packets of the file such that $\mathbf{m}^T = \begin{pmatrix} X_1 & X_2 & X_3 & X_4 \end{pmatrix}$. Since the file size is 4, we can express each coded packet by a 4×1 column vector $\mathbf{v}$ over the binary field. Specifically, we denote the vectors of the packets in node i by $(\mathbf{v}_1^{(i)}, \mathbf{v}_2^{(i)})$, which means that $Y_1^{(i)} = \mathbf{m}^T \mathbf{v}_1^{(i)}$ and $Y_2^{(i)} = \mathbf{m}^T \mathbf{v}_2^{(i)}$.

With the above vector notation, consider node b and denote the linear span of the vectors $(\mathbf{v}_1^{(b)}, \mathbf{v}_2^{(b)})$ of node b by $\mathbf{B}$. Specifically, $\mathbf{B} = \text{span}(\{\mathbf{v}_1^{(b)}, \mathbf{v}_2^{(b)}\}) = \{\mathbf{0}, \mathbf{v}_1^{(b)}, \mathbf{v}_2^{(b)}, \mathbf{v}_1^{(b)} + \mathbf{v}_2^{(b)}\}$, where $\mathbf{0}$ is the zero vector. Similarly, denote by $\mathbf{C}$, $\mathbf{D}$, and $\mathbf{E}$ the span of the vectors of nodes c, d, and e, respectively. In the following, we give an equivalent mathematical presentation of the regular repair operations that choose the (Z_b^*, Z_c^*) packets based on the linear spans $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$, and $\mathbf{E}$.

Choosing the Z_b^* packet: Using the above vector notation, how we choose Z_b^* can be rewritten as follows. If $\text{rank}(\mathbf{C} \oplus \mathbf{D}) = 4$ where $\oplus$ is the *sum-space* operator, then construct set $\mathbf{S_1} = \mathbf{D}$. If $\text{rank}(\mathbf{C} \oplus \mathbf{D}) \leq 3$, then construct set $\mathbf{S_1} = \mathbf{C} \oplus \mathbf{D}$. If $\text{rank}(\mathbf{C} \oplus \mathbf{E}) = 4$, then construct set $\mathbf{S_2} = \mathbf{E}$. If $\text{rank}(\mathbf{C} \oplus \mathbf{E}) \leq 3$, then construct set $\mathbf{S_2} = \mathbf{C} \oplus \mathbf{E}$. Then, we choose arbitrarily a vector $\mathbf{v}_b \in \mathbf{B} \backslash (\mathbf{S_1} \cup \mathbf{S_2})$ and then send $Z_b^* = \mathbf{m}^T \mathbf{v}_b$.

We now explain the reason why the above new code construction method is equivalent to the previous description of the repair operations. To that end, we notice that whenever $\mathbf{v}_b \notin \mathbf{S_1}$, then the coded packet Z_b^* will satisfy Condition 1 in our construction. Similarly, whenever $\mathbf{v}_b \notin \mathbf{S_2}$, the coded packet Z_b^* will satisfy Condition 2 in our construction. Since we choose $\mathbf{v} \in \mathbf{B} \backslash (\mathbf{S_1} \cup \mathbf{S_2})$, the Z_b^* will simultaneously satisfy both conditions.

We will argue now that we can always find such a vector $\mathbf{v}_b$. To that end, we will first prove that regardless how we construct $\mathbf{S_i}$, $i = 1, 2$, we always have $\text{rank}(\mathbf{B} \cap \mathbf{S_i}) \leq 1$. Since the constructions of $\mathbf{S_i}$ are symmetric (one focusing on spaces $\mathbf{C}$ and $\mathbf{D}$, and the other on spaces $\mathbf{C}$ and $\mathbf{E}$), we prove only $\text{rank}(\mathbf{B} \cap \mathbf{S_1}) \leq 1$. Consider two cases. Case 1:

$\mathrm{rank}(\mathbf{C} \oplus \mathbf{D}) = 4$. In this case, $\mathbf{S_1} = \mathbf{D}$, then we have

$$\begin{aligned}\mathrm{rank}(\mathbf{B} \cap \mathbf{S_1}) &= \mathrm{rank}(\mathbf{B}) + \mathrm{rank}(\mathbf{D}) - \mathrm{rank}(\mathbf{B} \oplus \mathbf{D}) \\ &\leq 2 + 2 - \mathrm{rank}(\mathbf{B} \oplus \mathbf{D}) \qquad (56) \\ &\leq 1, \qquad (57)\end{aligned}$$

where (56) follows from that the rank of the space of each node is at most 2, and (57) follows from that the induction conditions (A) and (B) imply that the rank of the sum space of two nodes is either 3 or 4, depending on whether one is the parent of the other.

Case 2: $\mathrm{rank}(\mathbf{C} \oplus \mathbf{D}) \leq 3$. In this case $\mathbf{S_1} = \mathbf{C} \oplus \mathbf{D}$ and we have

$$\begin{aligned}\mathrm{rank}(\mathbf{B} \cap \mathbf{S_1}) &= \mathrm{rank}(\mathbf{B} \cap (\mathbf{C} \oplus \mathbf{D})) \\ &= \mathrm{rank}(\mathbf{B}) + \mathrm{rank}(\mathbf{C} \oplus \mathbf{D}) - \\ &\quad \mathrm{rank}(\mathbf{B} \oplus (\mathbf{C} \oplus \mathbf{D})) \\ &\leq 2 + 3 - \mathrm{rank}(\mathbf{B} \oplus (\mathbf{C} \oplus \mathbf{D})) \qquad (58) \\ &= 2 + 3 - 4 \qquad (59) \\ &= 1,\end{aligned}$$

where (58) follows from that $\mathrm{rank}(\mathbf{B}) \leq 2$ and $\mathrm{rank}(\mathbf{C} \oplus \mathbf{D}) \leq 3$, and (59) follows from the fact that because we use the TA algorithm, among the three nodes b, c, and d, at least two of them do not have the parent-child relationship. By the induction assumption (B), we must have $\mathrm{rank}(\mathbf{B} \oplus \mathbf{C} \oplus \mathbf{D}) = 4$.

The above arguments prove that $\mathrm{rank}(\mathbf{B} \cap \mathbf{S_i}) \leq 1$ for $i = 1, 2$. If we count the number of elements in $\mathbf{B} \cap \mathbf{S_1}$ and $\mathbf{B} \cap \mathbf{S_2}$, then we must have $|\mathbf{B} \cap \mathbf{S_i}| \leq 2^1 = 2$ for $i = 1, 2$. Therefore, the size of $((\mathbf{B} \cap \mathbf{S_1}) \cup (\mathbf{B} \cap \mathbf{S_2}))$ is at most 3 since both $\mathbf{B} \cap \mathbf{S_1}$ and $\mathbf{B} \cap \mathbf{S_2}$ are linear subspaces and both thus contain the zero vector as a common element. As a result,

$$\begin{aligned}|\mathbf{B} \backslash (\mathbf{S_1} \cup \mathbf{S_2})| &= |\mathbf{B}| - |((\mathbf{B} \cap \mathbf{S_1}) \cup (\mathbf{B} \cap \mathbf{S_2}))| \\ &\geq 2^2 - 3 \\ &= 1.\end{aligned}$$

Therefore, there exists at least one non-zero vector $\mathbf{v}_b \in \mathbf{B} \backslash (\mathbf{S_1} \cup \mathbf{S_2})$.

Choosing the Z_c^* packet: Using the above notation, how we choose Z_c^* can be rewritten as follows. Recall that $\mathbf{v}_b$ is the vector for the coded packet Z_b^*. We argue that the construction of Z_c^* in Section V-B is equivalent to the following construction. That is, we choose arbitrarily a vector $\mathbf{v}_c \in \mathbf{C} \backslash ((\mathbf{v}_b \oplus \mathbf{D}) \cup (\mathbf{v}_b \oplus \mathbf{E}))$ and then send $Z_c^* = \mathbf{m}^T \mathbf{v}_c$. The reason that the above new code construction is equivalent to the previous description of the repair operations is as follows. Whenever, $\mathbf{v}_c \notin (\mathbf{v}_b \oplus \mathbf{D})$, then the coded packet Z_c^* will not be a linear combination of Z_b^* and the two packets in node d. Similarly, whenever $\mathbf{v}_c \notin (\mathbf{v}_b \oplus \mathbf{E})$, then the coded packet Z_c^* will not be a linear combination of Z_b^* and the two packets in node e. The choice of $\mathbf{v}_c \in \mathbf{C} \backslash ((\mathbf{v}_b \oplus \mathbf{D}) \cup (\mathbf{v}_b \oplus \mathbf{E}))$ thus satisfies both conditions simultaneously.

We argue now that we can always find such a vector $\mathbf{v}_c$. Specifically, we have that

$$\begin{aligned}\mathrm{rank}(\mathbf{C} \cap (\mathbf{v}_b \oplus \mathbf{D})) &= \mathrm{rank}(\mathbf{C}) + \mathrm{rank}(\mathbf{v}_b \oplus \mathbf{D}) - \\ &\quad \mathrm{rank}(\mathbf{C} \oplus (\mathbf{v}_b \oplus \mathbf{D})) \\ &\leq 2 + 3 - \mathrm{rank}(\mathbf{C} \oplus (\mathbf{v}_b \oplus \mathbf{D})) \qquad (60) \\ &= 2 + 3 - 4 \qquad (61) \\ &= 1,\end{aligned}$$

where (60) follows from $\mathrm{rank}(\mathbf{C}) \leq 2$ and $\mathrm{rank}(\mathbf{v}_b \oplus \mathbf{D}) \leq 3$. Equation (61) is due to the following facts. If $\mathrm{rank}(\mathbf{C} \oplus \mathbf{D}) = 4$, then obviously we have $\mathrm{rank}(\mathbf{C} \oplus \mathbf{v}_b \oplus \mathbf{D}) = 4$. If $\mathrm{rank}(\mathbf{C} \oplus \mathbf{D}) \leq 3$, then by the induction assumptions (A) and (B) we must have $\mathrm{rank}(\mathbf{C} \oplus \mathbf{D}) = 3$. In this case, we have $\mathbf{S_1} = \mathbf{C} \oplus \mathbf{D}$ when we construct $\mathbf{v}_b$. Therefore, $\mathrm{rank}(\mathbf{C} \oplus \mathbf{v}_b \oplus \mathbf{D}) = \mathrm{rank}(\mathbf{C} \oplus \mathbf{D}) + 1 = 4$. The above argument shows that $\mathrm{rank}(\mathbf{C} \cap (\mathbf{v}_b \oplus \mathbf{D})) \leq 1$. Symmetrically, we also have $\mathrm{rank}(\mathbf{C} \cap (\mathbf{v}_b \oplus \mathbf{E})) \leq 1$. By a verbatim argument as used in proving $|\mathbf{B} \backslash (\mathbf{S_1} \cup \mathbf{S_2})| \geq 1$, this implies that $|\mathbf{C} \backslash ((\mathbf{v}_b \oplus \mathbf{D}) \cup (\mathbf{v}_b \oplus \mathbf{E}))| \geq 1$. Therefore, we can always find such a $\mathbf{v}_c$.

Thus far, we have proven that whenever the induction conditions (A) and (B) hold for time $\tau = \tau_0 - 1$, we can always carry out the code construction for time $\tau = \tau_0$. We now argue that the induction conditions (A) and (B) also hold *after* we finish the repair operation in time $\tau = \tau_0$. Since only node a is repaired, we only need to check the node pairs (a, b), (a, c) to make sure they satisfy induction condition (A) and check node pairs (a, d) and (a, e) to make sure they satisfy induction condition (B).

The newcomer a now has packets (Z_b^*, Z_c^*) and the span of the vectors in a is now $\mathbf{A} = \mathrm{span}(\{\mathbf{v}_b, \mathbf{v}_c\})$. By our TA helper selection algorithm, the helper nodes b and c do not form a parent-child relationship. By induction condition (B), $\mathrm{rank}(\mathbf{B} \oplus \mathbf{C}) = 4$. Thus, any non-zero vector $\mathbf{v}_c \in \mathbf{C}$ is independent of the linear space $\mathbf{B}$. Therefore, $\mathrm{rank}(\mathbf{A} \oplus \mathbf{B}) = \mathrm{rank}(\mathbf{v}_c \oplus \mathbf{B}) = 3$. Symmetrically, $\mathrm{rank}(\mathbf{A} \oplus \mathbf{C}) = 3$.

To prove that $\mathrm{rank}(\mathbf{A} \oplus \mathbf{D}) = 4$, we notice that since $\mathbf{v}_b \notin \mathbf{S_1}$ and $\mathbf{S_1} \supseteq \mathbf{D}$, we must have $\mathbf{v}_b \notin \mathbf{D}$. Therefore, $\mathrm{rank}(\mathbf{v}_b \oplus \mathbf{D}) = 3$. Since $\mathbf{v}_c \notin (\mathbf{v}_b \oplus \mathbf{D})$, we have $\mathrm{rank}(\mathbf{A} \oplus \mathbf{D}) = \mathrm{rank}(\mathbf{v}_c \oplus (\mathbf{v}_b \oplus \mathbf{D})) = 4$. Symmetrically, we have $\mathrm{rank}(\mathbf{A} \oplus \mathbf{E}) = 4$. We can thus see that the nodes satisfy the induction conditions (A) and (B) after the repair operations in $\tau = \tau_0$.

We have shown thus far by induction that we can always repair the network/code at any time using the regular repair operations. The above also shows that we can maintain the induction conditions (A) and (B) at any time. We are thus only left with showing that we can construct the whole file from any $\hat{k} = 3$ nodes. Pick any three nodes in the network. By the TA scheme, these three nodes do not form a triangle, i.e., at least one pair of nodes in these three nodes does not form a parent-child relationship. By induction condition (B), we have that the 4 packets on these two nodes are linearly independent and we can use those packets to construct the file. The proof is hence complete.

Appendix D
Proof of Proposition 11

To prove Proposition 11, we need to show that

$$\min_{\forall \pi_f} \sum_{i=1}^{\hat{k}} (d - y_i(\pi_f))^+ = \sum_{i=1}^{\hat{k}} (d - y_i(\pi_f^*))^+, \tag{62}$$

where π_f^* is the RFIP defined in Section VI-B3.

In our previous work [5, Proposition 6], we have proven the following statements. For any $(n, \hat{k}, d, r)$ value satisfying $r = 0$, we have

$$\sum_{i=1}^{\hat{k}} y_i(\pi_f) \leq \sum_{i=1}^{\hat{k}} y_i(\pi_f^*) \tag{63}$$

$$\text{and } y_{i_1}(\pi_f^*) \geq y_{i_2}(\pi_f^*) \text{ if } i_1 < i_2. \tag{64}$$

Namely, the RFIP π_f^* maximizes the cumulative sum of $y_1(\pi_f^*)$ to $y_{\hat{k}}(\pi_f^*)$ and the $y_i(\pi_f^*)$ value is non-decreasing with respect to i.

We now argue that (63) and (64) hold for arbitrary $(n, \hat{k}, d, r)$ value with $r > 0$ as well. In the proof of Proposition 10, the paragraph right before (54), we have established that the $y_i(\cdot)$ function defined for one scenario $(n, \hat{k}, d, r)$ is identical to the $y_i(\cdot)$ function defined for another scenario $(n', \hat{k}', d', r')$ if we have $n = n'$, $d + r = d' + r'$. For any given $(n, \hat{k}, d, r)$ value, consider another set of parameters $(n', \hat{k}', d', r')$ such that $n' = n$, $\hat{k}' = \hat{k}$, $d' = d + r$, and $r' = 0$. Since (63) and (64) hold for any parameter values with $r = 0$, they must hold for the case of $(n', \hat{k}', d', r')$ since by our construction we have $r' = 0$. On the other hand, by the arguments in the proof of Proposition 10, the $y_i(\cdot)$ functions for both $(n, \hat{k}, d, r)$ and $(n', \hat{k}', d', r')$ must be identical. Therefore, (63) and (64) hold for the arbitrarily given $(n, \hat{k}, d, r)$ as well.

We now use (63) and (64) to prove (62). Given any $(n, \hat{k}, d, r)$ value, we construct the corresponding $y_i(\cdot)$ function and the RFIP π_f^*. Then, we define $\hat{k}_0 = \max\{x \in \{1, 2, \cdots, \hat{k}\} : y_x(\pi_f^*) \leq d\}$. Namely, $\hat{k}_0$ is the largest index $x \leq \hat{k}$ such that $y_x(\pi_f^*) \leq d$. Since by (64) $y_i(\pi_f^*)$ is non-decreasing, we must have $y_i(\pi_f^*) \leq d$ for all $1 \leq i \leq \hat{k}_0$ and $y_i(\pi_f^*) > d$ for all $\hat{k}_0 < i \leq \hat{k}$.

Consider any family permutation π_f, we now have

$$\sum_{i=1}^{\hat{k}} (d - y_i(\pi_f))^+ \geq \sum_{i=1}^{\hat{k}_0} (d - y_i(\pi_f))^+ \tag{65}$$

$$\geq \sum_{i=1}^{\hat{k}_0} (d - y_i(\pi_f)) \tag{66}$$

$$\geq \sum_{i=1}^{\hat{k}_0} (d - y_i(\pi_f^*)) \tag{67}$$

$$= \sum_{i=1}^{\hat{k}} (d - y_i(\pi_f^*))^+, \tag{68}$$

where (65) follows from that each $(d - y_i(\pi_f))^+$ is non-negative and we sum over $i = 1$ to $\hat{k}_0$ for some $\hat{k}_0 \leq \hat{k}$; (66) follows from removing the projection $(\cdot)^+$ operator; (67) follows from (63); and (68) follows by the definition of $\hat{k}_0$, which ensures $y_i(\pi_f^*) > d$ for all $\hat{k}_0 < i \leq \hat{k}$. By (68), we get (62). Hence, the proof is complete.

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 code constructions," *[Online]. Available: arXiv:[cs.IT].*

[3] ——, "When can helper node selection improve regenerating codes? Part I: Graph-based analysis," *[Online]. Available: arXiv:1604.08231 [cs.IT].*

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

[5] ——, "When can intelligent helper node selection improve the performance of distributed storage networks?" *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 2142–2171, 2018.

[6] 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.

[7] R. Dougherty, C. Freiling, and K. Zeger, "Insufficiency of linear coding in network information flow," *IEEE Trans. Inf. Theory*, vol. 51, no. 8, pp. 2745–2759, 2005.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

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

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

[19] D. B. West *et al.*, *Introduction to graph theory*. Prentice hall Upper Saddle River, 2001, vol. 2.

[20] 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.

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 (M'06–SM'15) 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.