Communication-Efficient LDPC Code Design for Data Availability Oracle in Side Blockchains

Debarnab Mitra, Lev Tauz and Lara Dolecek

Department of Electrical and Computer Engineering, University of California, Los Angeles, USA

email: debarnabucla@ucla.edu, levtauz@ucla.edu, dolecek@ee.ucla.edu

Abstract—A popular method of improving the throughput of blockchain systems is by running smaller side blockchains that push the hashes of their blocks onto a trusted blockchain. Side blockchains are vulnerable to *stalling attacks* where a side blockchain node pushes the hash of a block to the trusted blockchain but makes the block unavailable to other side blockchain nodes. Recently, Sheng *et al.* proposed a data availability oracle based on LDPC codes and a data dispersal protocol as a solution to the above problem. While showing improvements, the codes and dispersal protocol were designed disjointly which may not be optimal in terms of the communication cost associated with the oracle. In this paper, we provide a tailored dispersal protocol and specialized LDPC code construction based on the Progressive Edge Growth (PEG) algorithm, called the *dispersal-efficient PEG (DE-PEG)* algorithm, aimed to reduce the communication cost associated with the new dispersal protocol. Our new code construction reduces the communication cost and, additionally, is less restrictive in terms of system design.

I. INTRODUCTION

Side blockchains, e.g., [1]–[4], are a popular method of improving the transaction throughput of blockchain systems where a single trusted blockchain supports a large number of side (smaller) blockchains by storing the block hashes of the side blockchains in their ledger [5]. Systems that run side blockchains are vulnerable to a form of data availability attack [6], [7] called a *stalling attack*, where a side blockchain node commits the hash of a block to the trusted blockchain but makes the block itself unavailable to other side blockchain nodes. Authors in [5] proposed a scalable solution to the above attack by introducing a data availability oracle between the trusted blockchain and side blockchains. The oracle consists of nodes whose goal is to collectively ensure that the block is available, even if some of the oracles nodes are malicious. Nodes in the oracle layer accept the block from a side blockchain node (who wishes to commit its hash to the trusted blockchain), and push the hash commitment only if the block is available to the system. The goal is to share (*disperse*) the block among the oracle nodes in a storage and communication efficient way to check the block availability. The solution in [5] involves using a Low-Density Parity-Check (LDPC) code to generate coded chunks from the block such that each oracle node receives different coded chunks, and using *incorrect-coding proofs* [6], [7] to ensure that the block is correctly coded. A dispersal protocol ensures that the oracle nodes receive sufficient coded chunks that guarantee that the original block can always be decoded by a peeling decoder using the coded chunks sent to the oracle nodes, (i.e., the block is available), even in the presence of malicious oracle nodes.

Stopping sets are a set of variable nodes (VNs) of an LDPC code that if erased prevent a peeling decoder from decoding the block. Formally, a set of VNs of an LDPC code is called a stopping set if every check node (CN) connected to this set of VNs is connected to it at least twice. To guarantee block availability in the presence of malicious oracle nodes, the dispersal protocol defined in [5] requires every subset of oracle nodes of a particular size to receive at least $M - M_{\min} + 1$ distinct coded chunks, where M and $M_{\min}$ are the blocklength and minimum stopping set size of the LDPC code, respectively. As a result, the communication cost associated with the dispersal is inversely proportional to the minimum stopping set size of the LDPC code. Thus, authors in [5] focused on LDPC code constructions with large minimum stopping set size for their dispersal protocol. This combination of dispersal protocol and LDPC construction may not necessarily be optimal in terms of communication costs. In this paper, we design a new dispersal protocol that considers the multiplicity of small stopping sets and provide a specialized LDPC code construction based on the Progressive Edge Growth (PEG) algorithm [18], which we call the *dispersal-efficient* PEG (DE-PEG) algorithm, that aims at minimizing the communication cost within our protocol. We demonstrate a significantly lower communication cost using our specialized LDPC construction and dispersal protocol in comparison to [5]. Our techniques support a wider range of system parameters allowing for more flexibility in system design such as scaling the number of oracle nodes while still allowing scalability of the block size as [5] thereby providing a much more scalable solution to the stalling attack problem. Previously, channel coding has been extensively used to mitigate issues such as data availability, storage, and communication in blockchain systems [6]–[17].

The rest of this paper is organized as follows. In Section II, we describe the preliminaries and system model. In Section III, we provide our new dispersal protocol and motivate our LDPC design criterion. The DE-PEG algorithm is described in Section IV. Finally, the simulation results and concluding remarks are presented in Section V.

II. PRELIMINARIES AND SYSTEM MODEL

In this paper, we assume the blockchain and data availability oracle model of [5] and is summarized in Fig. 1. Suppose that there are N oracle nodes and an adversary is able to corrupt a fraction β of them, where $\beta < \frac{1}{2}$, such that the maximum number of malicious oracles nodes is $f = \lceil \beta N \rceil$. When a client proposes a block of size b, it first generates a special Coded Merkle Tree [7], called a Coded Interleaving Tree (CIT) introduced in [5], with the data chunks of the block as leaf nodes of the CIT. A CIT is a coded version of a regular Merkle

Fig. 1: System Model. The network consists of oracle nodes and side blockchain nodes (called *clients*), where clients propose blocks to the oracle nodes to commit to the trusted blockchain. Oracle nodes verify the correctness of each received block and submit the block commitment to the trusted blockchain if the block is available.

tree [19] and is constructed by applying a rate-R systematic LDPC code to each layer of the Merkle tree before hashing the layer to generate its parent layer. Details regarding the CIT construction can be found in [5] and in Appendix A. In particular, the CIT has M base layer coded chunks (symbols), each with an associated Proof of Membership (POM), which consists of a systematic (data) symbol and a parity symbol from each CIT layer. The CIT has a root with t hashes and in each layer q hashes are batched together into a data chunk for the layer. The data availability oracle functions in the following way as shown in Fig. 1:

1) When a client proposes a block of size b, it constructs its CIT, which generates a set of base layer coded symbols, each with an associated POM, and a CIT root.

2) The client then uses a dispersal protocol to disperse the base layer coded symbols, their associated POMs, and the CIT root to the N oracle nodes. The dispersal protocol specifies the base layer coded chunks each oracle node should receive, each receiving k of them (with their POMs) and the CIT root.

3) Each of the oracle nodes, on receiving the specified k coded chunks check their correctness (i.e., whether they satisfy the associated POM with the root). The dispersal is accepted if $\gamma + \beta$ fraction of the nodes vote that they individually received all correct coded chunks, for a parameter $\gamma \leq 1 - 2\beta$ defined in the dispersal protocol. In this case, the CIT root is committed to the trusted blockchain and each of the oracle nodes store the k coded chunks they received to allow for future block retrieval. The CIT prevents clients from performing incorrect coding of the block via an incorrect-coding proof [5].

The focus of this paper is to design a dispersal protocol and an associated LDPC code to reduce the communication cost of the dispersal process. The dispersal protocol must satisfy the *availability* condition: whenever the root of the CIT is committed to the trusted blockchain, an honest client must be able to decode each CIT layer using a peeling decoder by requesting for the coded chunks stored at the oracle nodes. Each CIT layer in [5] is constructed using random LDPC codes that with high probability have a stopping ratio (minimum stopping set size divided by the blocklength) α^*. The dispersal protocol in [5] is designed such that every γ fraction of the oracle nodes receive more than $1 - \alpha^*$ fraction of distinct base

layer coded chunks. Moreover, it was shown in [5] that the POMs of any η fraction of distinct base coded chunks have at least η fraction of distinct coded chunks from each CIT layer (we call this the *repetition property*). Thus, the dispersal protocol ensures that every γ fraction of nodes also have more than $1 - \alpha^*$ fraction of distinct coded chunks from each CIT layer. Hence, when a root is committed, due to 3), there is a γ fraction of honest oracle nodes who have more than $1 - \alpha^*$ fraction of coded symbols from each CIT layer, allowing a peeling decoder to decode each layer ensuring *availability*.

Let the CIT have l layers and n_j coded chunks in layer j, $1 \leq j \leq l$, where $n_l = M$. Let H_j denote the parity check matrix of the LPDC code used in layer j which has n_j columns $\{v_1^j, v_2^j, \ldots, v_{n_j}^j\}$ (we drop the superscript j based on context). Let $\mathcal{G}_j$ denote the Tanner graph (TG) representation of H_j, where we also refer to v_i as the ith VN in $\mathcal{G}_j$ and rows of H_j as CNs in $\mathcal{G}$. A cycle of length g is called a g-cycle. For a set T, let $|T|$ denote its cardinality. Let the dispersal protocol be defined by the set $\mathcal{C} = \{A_1, A_2, \ldots, A_N\}$, where A_i denotes the set of base layer coded chunks sent to oracle node i and $|A_i| = k$. For a set S of VNs, let $\text{neigh}(S)$ be the set of oracle nodes who have at least one coded chunk corresponding to the VNs of S. Let the hashes of each coded block be of size y. Let $H_e(p) = -p \ln(p) - (1-p) \ln(1-p)$.

Definition 1. *Protocol $\mathcal{C}$ is called η-valid for layer j if every γ fraction of oracle nodes have $> \eta$ fraction of distinct layer j coded chunks. Similarly, $\mathcal{C}$ is μ-SS-valid for layer j if every γ fraction of oracle nodes have $> n_j - \mu$ distinct layer j coded chunks. If no layer is specified, we refer to the base layer.*

Note that a protocol that is μ-SS-valid for layer j is also $\left(\frac{n_j - \mu}{n_j}\right)$-valid for layer j. Due to the *repetition* property, if the base layer is μ-SS-valid, we can determine $\tilde{\mu}$ such that the protocol is $\tilde{\mu}$-SS-valid for layer j. Thus, we majorly talk about the base layer and drop the specification of the layer according to Definition 1. In [5], the dispersal protocol is required to be $(1 - \alpha^*)$-valid for all layers. A protocol which is μ-SS-valid for layer j can guarantee that a client will be able to decode layer j of the CIT using a peeling decoder when the block is committed and stopping sets of size $< \mu$ do not exist in H_j.

In [5], elements of A_i are randomly chosen with replacement from the set of M base layer coded chunks. For such a design, it was shown in [5] that for $k > \frac{M}{N\gamma} \ln \frac{1}{1-\eta}$, $\text{Prob}(\mathcal{C} \text{ is not } \eta\text{-valid}) \leq \exp(N H_e(\gamma) - M f(\eta, \rho)) := P^{\text{UB}}(\eta, N, M, k, \gamma)$, where $\rho = \frac{\gamma N k}{M}$ and $f(\eta, \rho) = \frac{(e^\rho (1-\eta) - 1)^2}{e^\rho (e^\rho (1-\eta) + 1)}$ is a positive function. It is clear that M can be made sufficiently large to make $P^{\text{UB}}(\eta, N, M, k, \gamma)$ arbitrarily small. This principle was used in [5] to randomly design the dispersal protocol. However, as we show next, to make $P^{\text{UB}}(\eta, N, M, k, \gamma)$ smaller than a given threshold probability p_{th}, for a fixed M, there is a limit on the number of oracle nodes the system can support.

Lemma 1. *Let $N^{UB} = \frac{M(1-\eta) + \ln(p_{th})}{H_e(\gamma)}$ and $\bar{\eta} = 1 - \eta$. If $N \geq N^{UB}$, $P^{\text{UB}}(\eta, N, M, k, \gamma) > p_{th} \ \forall k > \frac{M}{N\gamma} \ln \frac{1}{\eta}$. If $N < N^{UB}$, then $P^{\text{UB}}(\eta, N, M, k, \gamma) \leq p_{th} \text{ for } k \geq k_{\min}^f :=$*

$\frac{M}{N\gamma} \ln\left(\frac{-(2\bar{\eta}+v)-\sqrt{8\bar{\eta}v+v^2}}{2\bar{\eta}(v-\bar{\eta})}\right)$, *where* $v = \frac{NH_e(\gamma)-\ln(p_{th})}{M}$.

Proof. The proof relies on algebraic manipulation of $P^{\text{UB}}()$ and can be found in Appendix B. $\square$

Thus, the dispersal protocol used in [5] cannot guarantee with high probability to be $(1-\alpha^*)$-valid for all (N, M) pairs. This feature is undesirable and we would like for a given M, any number of oracle nodes to be supported by the protocol. The problem is alleviated if each A_i gets k distinct coded chunks chosen uniformly at random from the M base layer coded chunks which we consider in our design idea.

III. DESIGN IDEA: SECURE STOPPING SET DISPERSAL

Definition 2. *A protocol* $\mathcal{C} = \{A_1, \dots, A_N\}$ *is called a k-dispersal if each A_i is a k element subset chosen uniformly at random with replacement from all the k element subsets of the M base layer coded chunks.*

We analyze the minimum number of distinct coded chunks k to disperse to each oracle node so that the protocol $\mathcal{C}$ is μ-SS-valid with probability at least $1 - p_{th}$.

Lemma 2. *For a k-dispersal protocol, for the base layer,*

$$\text{Prob}(\mathcal{C} \text{ is not } \mu\text{-SS-valid}) \leq e^{NH_e(\gamma)}P_f,$$

where $P_f = \sum_{j=0}^{M-\mu}(-1)^{M-\mu-j}\binom{M}{j}\binom{M-j-1}{\mu-1}\left[\frac{\binom{j}{k}}{\binom{M}{k}}\right]^{\gamma N}$

Proof. The proof utilizes the fact that the process of a given γN nodes sampling with replacement the k element subsets of the M base layer chunks is known as the coupon collector's problem with group drawings [20]. The proof is provided in Appendix C. $\square$

Now, $e^{NH_e(\gamma)}P_f$ can be made smaller than an arbitrary threshold p_{th} by choosing a sufficiently large k. Let $k^*(\mu, N, M, \gamma, p_{th})$ be the smallest k such that $e^{NH_e(\gamma)}P_f \leq p_{th}$. Thus, a $k^*(\mu, N, M, \gamma, p_{th})$-dispersal will be μ-SS-valid for the base layer with probability $\geq 1 - p_{th}$. The associated communication cost is $NXk^*(\mu, N, M, \gamma)$, where X is the total size of one base layer coded chunk along with its POM. To ensure availability, if we set $\mu = M_{\min}$, we see that the communication cost is directly affected by the minimum stopping set size. Thus, for a $k^*(M_{\min}, N, M, \gamma, p_{th})$-dispersal protocol, the best code design strategy is to design LDPC codes with large minimum stopping set sizes which is considered to be a hard problem [21], [22]. In this work, we modify the above dispersal protocol to reduce the communication cost. We then provide a specialized LDPC code construction aimed at minimizing the communication cost associated with the modified dispersal protocol.

Definition 3. *A stopping set S is said to be securely dispersed by a dispersal protocol $\mathcal{C}$ if $|\text{neigh}(S)| \geq f + 1$.*

Since f is the maximum number of malicious oracle nodes, for a stopping set that is securely dispersed, at least one honest oracle node will have a coded chunk corresponding to a VN of S and hence the peeling decoder will not fail due to S and can continue the decoding process. Based on this principle, we consider the following dispersal protocol:

Dispersal Protocol 1. *(k^*-secure dispersal) For $\mu \geq M_{\min}$, let $\mathcal{S}_j$ be the set of stopping sets of layer j (i.e., of H_j) of size less than $n_j - \left\lceil\left(\frac{M-\mu+1}{M}\right)n_j\right\rceil + 1$. Our dispersal protocol consists of two dispersal phases. In the first phase (called the secure phase), all stopping sets in $\mathcal{S}_j$, $1 \leq j \leq l$, are securely dispersed. This is followed by a $k^*(\mu, N, M, \gamma, p_{th})$-dispersal protocol (called the valid phase).*

Lemma 3. *Dispersal protocol 1 guarantees availability with probability $\geq 1 - p_{th}$.*

Proof. Proof is provided in Appendix D. For $d_j = (n_j - \left\lceil\left(\frac{M-\mu+1}{M}\right)n_j\right\rceil + 1)$, we use the *repetition* property to show that with probability $\geq 1 - p_{th}$, the protocol is d_j-SS-valid for each layer j, and all stopping sets of sizes $< d_j$ are securely dispersed. These conditions guarantee availability. $\square$

We use the following greedy procedure to securely disperse all stopping sets in $\mathcal{S}_j$, $1 \leq j \leq l$. Let $\mathcal{V}_j^{gr}$ be a set of VNs of H_j with the property that for all $S \in \mathcal{S}_j$, $\exists v \in \mathcal{V}_j^{gr}$ such that v is part of S. Note that if each VN in $\mathcal{V}_j^{gr}$ is sent to $(f+1)$ oracle nodes, all $S \in \mathcal{S}_j$ will be securely dispersed. We obtain $\mathcal{V}_j^{gr}$ in the following greedy manner: Initialize $\mathcal{V}_j^{gr} = \emptyset$. Find a VN v that is part of the maximum number of stopping sets in $\mathcal{S}_j$, add the VN to $\mathcal{V}_j^{gr}$ and remove all stopping sets in $\mathcal{S}_j$ that have v. We repeat the process until $\mathcal{S}_j$ is empty. Let the VNs in each set $\mathcal{V}_j^{gr}$ be ordered according to the order they were added to $\mathcal{V}_j^{gr}$. For each j, we permute the columns of H_j such that the VNs in $\mathcal{V}_j^{gr}$ appear as columns $1, 2, \dots, |\mathcal{V}_j^{gr}|$, the rest of the columns are randomly ordered. Note that the H_j's after the column permutation are used to build the CIT. Now, our secure phase is designed as follows: the design starts from layer l and moves iteratively up the tree till layer 1. For each layer j, if all VNs corresponding to the first $|\mathcal{V}_j^{gr}|$ columns of H_j are marked as *dispersed*, we mark layer j as *complete* and move to layer $j - 1$, else, we disperse the remaining coded chunks corresponding to the first $|\mathcal{V}_j^{gr}|$ columns of H_j that are not marked *dispersed* by randomly selecting $f+1$ oracle nodes to send each of the coded chunk with its POMs. For each layer i above layer j, coded chunks that were sent to $(f+1)$ nodes as part of POMs of the coded chunks of layer j in the previous step are marked as *dispersed*. We mark layer j as *complete* and proceed to layer $j - 1$. We continue until layer 1 is *complete*.

Note that by initially permuting the columns of H_j's, we have ensured that when a coded chunk of a particular layer j with its POMs are sent to $(f+1)$ nodes, the systematic symbol of the POMs from each layer i above layer j are exactly the VNs in the first $|\mathcal{V}_i^{gr}|$ columns of H_i that we require to send to $(f+1)$ nodes to securely disperse $\mathcal{S}_i$. If the POM for layer i is outside the first $|\mathcal{V}_i^{gr}|$ columns (happens if $|\mathcal{V}_j^{gr}| > |\mathcal{V}_i^{gr}|$), this would imply that layer i is already *complete*.

Let X_j be the size of one coded chunk of layer j along with its POMs which involve a data and parity symbol from each layer above layer j. Also, let $t_j = \max_{i \in \{j+1,\dots,l\}} |\mathcal{V}_i^{gr}|$. As such, $X_l = \frac{b}{RM} + y(2q-1)(l-1)$ and $X_j = qy + y(2q-1)(j-1)$, $1 \leq j < l$, [5] (details can also be found in Appendix A). The total communication cost C^T for Dispersal Protocol 1 is $\text{C}^T = Nty + \text{C}^s + \text{C}^v$, where C^s and C^v

are the costs associated with the secure and valid phases, respectively, and Nty is the cost of dispersing the CIT root. Now, $\mathrm{C}^v = Nk^*(\mu, N, M, \gamma, p_{th})X_l$ and C^s can be calculated as $\mathrm{C}^s = (f+1)\left[|\mathcal{V}_l^{gr}|X_l + \sum_{j=1}^{l-1} \max\left((|\mathcal{V}_j^{gr}| - t_j), 0\right)X_j\right]$, where we have made the assumption that each $|\mathcal{V}_j^{gr}|$ is smaller than Rn_j which is true for small μ and since Rn_j is the total number of systematic variable nodes. The communication cost of the secure phase depends strongly on $|\mathcal{V}_l^{gr}|$ as the base layer involves data chunks whose sizes are larger than the chunks of the higher layers which are concatenations of hashes. Thus, we can reduce the total cost by designing LDPC codes that have small $|\mathcal{V}_l^{gr}|$. We provide the construction in the next section.

IV. DISPERSAL-EFFICIENT PEG ALGORITHM

Algorithm 1 presents our DE-PEG algorithm that constructs a TG $\widehat{\mathcal{G}}$ with M VNs, J CNs, and VN degree d_v that results in a small size of $\mathcal{V}_l^{gr}$. Note that the same algorithm is used for all layers to reduce the sizes of $\mathcal{V}_j^{gr}$. Since stopping sets in LDPC codes are made up of cycles [23], the DE-PEG algorithm focuses on cycles as they are easier to optimize. In the algorithm, all ties are broken randomly.

Algorithm 1 DE-PEG Algorithm

1: **Inputs:** M, J, d_v, $g_{\max}$, T_{th} **Output:** $\widehat{\mathcal{G}}$
2: **Initialize** $\widehat{\mathcal{G}}$ to M VNs, J CNs and no edges, $\mathcal{L} = \emptyset$
3: **for** $j = 1$ to M **do**
4:   **for** $e = 1$ to d_v **do**
5:     $[\mathcal{K}, g] = \text{PEG}(\widehat{\mathcal{G}}, v_j)$
6:     **if** $g > g_{\max}$ **then** c^s = uniformly random CN in $\mathcal{K}$
7:     **else**   ▷ *(g-cycles, $g \leq g_{\max}$, are created)*
8:       **for** each CN c in $\mathcal{K}$ **do**
9:         $\mathcal{L}_{cycles}$ = g-cycles formed due to c
10:         $\bar{s}[c]$ = $\text{greedy-size}(\mathcal{L} \cup \mathcal{L}_{cycles}, v_j)$
11:     c^s = CN in $\mathcal{K}$ with minimum $\bar{s}[c]$
12:     $\mathcal{L}^s$ = g-cycles formed due to c^s with EMD $\leq T_{th}$
13:     $\mathcal{L} = \mathcal{L} \cup \mathcal{L}^s$
14:     $\widehat{\mathcal{G}} = \widehat{\mathcal{G}} \cup \text{edge}\{c^s, v_j\}$

The algorithm uses the concept of the extrinsic message degree (EMD) of a set of VNs which is the number of CN neighbours singly connected to the set of VNs [24] and is calculated using the method described in [25]. EMD of a cycle is the EMD of the VNs involved in the cycle. Cycles with low EMD are more likely to form a stopping set and we consider them as *bad* cycles. The algorithm also uses a procedure $\text{greedy-size}(\tilde{\mathcal{L}}, v)$ which takes as input a list $\tilde{\mathcal{L}}$ of cycles, and outputs $|\bar{S}|$, where $\bar{S}$ is a set of VNs with the property that for every cycle C in $\tilde{\mathcal{L}}$, $\exists$ a VN in $\bar{S}$ that is part of C, and $\bar{S}$ is obtained in a manner similar to that of obtaining $\mathcal{V}_j^{gr}$ from $\mathcal{S}_j$ described in Section III, however, by ignoring the VN v during the greedy selection procedure.

The PEG algorithm [18] builds a TG by iterating over the set of VNs and for each VN v_j, establishing d_v edges to it. For establishing the eth edge, there are two situations that the algorithm encounters: i) addition of an edge is possible without creating cycles; ii) addition of an edge creates cycles. In both

the situations, the PEG algorithm finds a set of *candidate* CNs to connect v_j to, that maximises the girth of the cycles formed. We do not go into the detailed procedure followed by [18] to find the set of candidate CNs, but assume a procedure $\text{PEG}(\mathcal{G}, v_j)$ that provides us with the set of candidate CNs $\mathcal{K}$ for establishing a new edge to VN v_j under the TG setting $\mathcal{G}$ according to the PEG algorithm. We assume that the set $\mathcal{K}$ only contains CNs with the minimum degree under the TG setting $\mathcal{G}$. For situation ii), the procedure returns the cycle length g of the smallest cycles formed when an edge is established between any CN in $\mathcal{K}$ and v_j. For situation i) $g = \infty$ is returned. When $g > g_{\max}$ is returned, we follow the PEG algorithm and select a CN randomly from $\mathcal{K}$.

During the course of the DE-PEG algorithm, we maintain a list $\mathcal{L}$ of *bad* cycles of lengths $\leq g_{\max}$ that had EMD less than or equal to some threshold T_{th} when they were formed. In the algorithm, when cycle length $g \leq g_{\max}$ is returned by the $\text{PEG}()$ procedure, for each CN $c \in \mathcal{K}$, g-cycles are formed when an edge is added between c and v_j. These cycles are listed in $\mathcal{L}_{cycles}$ (line 9). We use $\text{greedy-size}(\mathcal{L} \cup \mathcal{L}_{cycles}, v_j)$ to get $\bar{s}[c]$ (line 10), for each CN c in $\mathcal{K}$. Our CN selection procedure is to select a CN from $\mathcal{K}$ that has the minimum $\bar{s}$ (line 10). Once this CN is selected, we update the list of *bad* cycles as follows: of all the g-cycles formed due to the addition of an edge between c^s and v_j, we find the list of g-cycles $\mathcal{L}^s$ that have EMD $\leq T_{th}$ (line 12) and add them to $\mathcal{L}$ (line 13). We then update the TG $\widehat{\mathcal{G}}$ (line 14). The intuition is that since we want the stopping sets in $\mathcal{S}_l$ to produce a small $\mathcal{V}_l^{gr}$ by a greedy procedure, we select CNs such that a similar greedy procedure produces small $|\bar{S}|$ on the bad cycles which are more likely to form stopping sets. Note that a similar PEG algorithm was provided in [8] but it had a different design objective compared to this paper.

Remark 1. *In the DE-PEG algorithm, $\text{greedy-size}(\mathcal{L} \cup \mathcal{L}_{cycles}, v_j)$ ignores the VN v_j while forming the greedy set of VNs to find $|\bar{S}|$ as v_j is part of all the cycles formed by all CNs c in $\mathcal{K}$ and ignoring v_j allows to better distinguish between the CNs in terms of set sizes $\bar{s}$. While the DE-PEG algorithm is based on cycles, Dispersal Protocol 1 uses stopping sets $\mathcal{S}_j$ to find $\mathcal{V}_j^{gr}$ for the secure phase.*

V. SIMULATIONS AND CONCLUSION

In this section, we present the performance of the codes designed using the DE-PEG algorithm when using the k^*-secure dispersal protocol. To demonstrate the benefits, we consider a baseline system that uses codes constructed using the original PEG algorithm and uses k-dispersal with k chosen such that for all layers j, $1 \leq j \leq l$, the k-dispersal is $M_{\min}^j$-SS-valid, where $M_{\min}^j$ is the minimum stopping set size of layer j (and $M_{\min} = M_{\min}^l$). To compute a lower bound on the total communication cost using Dispersal Protocol 1, we consider a code that has $\mathcal{S}_j = \emptyset$, $1 \leq j \leq l$, i.e., for the given μ, has costs only due to $k^*(\mu, N, M, \gamma, p_{th})$-dispersal and the root, and no cost due to the secure phase. This is equivalent to designing codes having larger minimum stopping set sizes which is considered hard. We use the following parameters for simulations: $b = 1$MB, $y = 32$ Bytes, $t = 32$, $q = 4$, $l = 4$,

TABLE I: Communication costs achieved by k^*-secure dispersal for various choices of μ using the PEG and DE-PEG algorithm for $N = 9000$, $\beta = 0.49$. Lower bound on C^T for $\mu = 20$ is 4.438GB.

μ	k^*	C^v	$(|\mathcal{V}_1^{gr}|,|\mathcal{V}_2^{gr}|,|\mathcal{V}_3^{gr}|,|\mathcal{V}_4^{gr}|)$		C^s		C^T	
			PEG	DE-PEG	PEG	DE-PEG	PEG	DE-PEG
17	67	5.116	(0,0,0,0)	(0,0,0,0)	0	0	5.125	5.125
18	64	4.887	(0,0,0,1)	(0,0,0,0)	0.037	0	4.933	4.896
19	61	4.658	(0,0,1,3)	(0,0,0,1)	0.112	0.037	4.779	4.704
20	58	4.428	(0,0,1,7)	(0,0,0,4)	0.262	0.149	4.700	4.587
21	56	4.276	(0,1,2,14)	(0,1,0,13)	0.524	0.486	4.809	4.771

TABLE II: Comparison of total communication cost of our work with [5] for $\beta = 0.49$, $M = 256$. For each p_{th}, N is the maximum no. of oracle nodes permissible for the oracle of [5] and $k_{\min}$ is the minimum number of chunks at each oracle node for $\eta = 1 - \alpha^*$ and are computed using Lemma 1 (Note that $k_{\min} = \lceil k_{\min}^f \rceil$).

		Oracle [5]			Our Work			
		$\alpha^* = 0.125$			$\mu = 17$		$\mu = 20$	
p_{th}	N	$k_{\min}$	C^T_{full}	$C^T_{distinct}$	k^*	$C^T_{baseline}$	k^*	C^T_{PEG}	C^T_{DE-PEG}
10^{-8}	138	895	1.048	0.2909	207	0.2425	199	0.2372	0.2355
10^{-6}	185	671	1.053	0.3729	184	0.2890	175	0.2803	0.2780
10^{-4}	232	539	1.061	0.4430	164	0.3231	155	0.3122	0.3092

Fig. 2: C^T for various coding schemes and dispersal protocols vs. (a) N at $\beta = 0.49$ (solid plots, top axis) and (b) β at $N = 20000$ (dotted plots, bottom axis). For k^*-secure dispersal $\mu = 20$ is used.

$M = 256$, $R = 0.5$, $\gamma = 1 - 2\beta$, $p_{th} = 10^{-8}$ (specified if otherwise). The CIT thus has 4 layers with $n_4 = M = 256$, $n_3 = 128$, $n_2 = 64$ and $n_1 = 32$. For the LDPC codes constructed using the DE-PEG algorithm, we use $g_{\max} = 8$ for layer 3 and 4 and $g_{\max} = 6$ for layer 1 and 2, $d_v = 4$, and $T_{th} = 5$ (provides the best results from a range of thresholds tested). For the base layer, the PEG and DE-PEG codes constructed have $M_{\min} = 17$ and 18 respectively. All communication costs are in GB. Costs C^s, C^v and C^T are calculated using equations described in Section III.

Table I compares the communication cost achieved by the PEG and DE-PEG algorithm with the k^*-secure dispersal protocol as the value of μ is varied. We see that as μ is increased, the value of k^* decreases and C^v decreases. The table next shows the 4-tuple $(|\mathcal{V}_1^{gr}|,|\mathcal{V}_2^{gr}|,|\mathcal{V}_3^{gr}|,|\mathcal{V}_4^{gr}|)$ for the PEG and DE-PEG algorithm and we see that the DE-PEG algorithm always results in lower values thus resulting in a lower cost C^s during the secure phase compared to the PEG algorithm. Note that, as μ is increased, C^s increases. Finally, we look at the total cost C^T, which is lowest for $\mu = 20$ for both the PEG and DE-PEG algorithms, and are 0.425GB and 0.528GB lower, respectively, compared to the baseline ($C^T = 5.125$GB) at $\mu = 17$. Interestingly, C^T does not monotonically decrease with μ. Note that the lower bound on C^T at $\mu = 20$ is 0.687GB lower than the baseline.

Table II compares the performance of the k^*-secure dispersal protocol and DE-PEG algorithm with the performance achieved by the data availability oracle of [5]. C^T_{full} is the communication cost associated with $k_{\min}$ (considering that each node gets the same chunk multiple times and similar to the computation carried out in [5]). $C^T_{distinct}$ is the total cost by considering only distinct chunks (out of the $k_{\min}$ chunks) at each node (calculated using Monte-Carlo simulations). For our work, we present the baseline and the k^*-secure dispersal protocol with $\mu = 20$ (best results in Table I) with the PEG and the DE-PEG algorithms. From Table II, we see three levels of cost reduction; 1) $C^T_{distinct}$ to $C^T_{baseline}$ is due to sampling with replacement, 2) $C^T_{baseline}$ to C^T_{PEG} is due to

using the k^*-secure dispersal protocol for $\mu = 20$, and 3) C^T_{PEG} to C^T_{DE-PEG} is due to using DE-PEG LDPC codes designed to reduce the total communication cost. Note that these reductions are for a single 1MB block. Also, for the Oracle of [5], we use $\alpha^* = 0.125$ which is for a rate $\frac{1}{4}$ code but assume the data chunk sizes are the same as a rate $\frac{1}{2}$ code to demonstrate that even in this disadvantageous situation we have a better communication cost.

In Fig. 2, the total communication cost C^T is plotted as a function of the number of oracle nodes and adversary fraction. The solid plots show C^T vs. N at $\beta = 0.49$. We see that at $N = 15000$, compared to the baseline, there is around 7% reduction in C^T by using the k^*-secure dispersal protocol with $\mu = 20$ and the PEG algorithm, and a 9.3% reduction by using the protocol with $\mu = 20$ and the DE-PEG algorithm. The yellow plot corresponds to the lower bound on C^T for $\mu = 20$ and is tantamount to a maximum of 13% reduction in C^T from the baseline. The dotted plots show C^T vs. β at $N = 20000$. We see a sharp increase in C^T for higher β and as β is increased from 0.4 to 0.49, C^T increases by around 5.1GB for the baseline, whereas for the PEG and DE-PEG algorithms using the k^*-secure dispersal protocol (with $\mu = 20$), C^T increases by around 4.52GB and 4.47GB, respectively. The result indicates that using our methods, the system has to pay less in terms of the total communication cost in order to handle a higher adversary fraction.

In conclusion, we provided a new dispersal protocol and a modification of the PEG algorithm, called the DE-PEG algorithm, that when combined provide a much lower communication cost in the data availability oracle of [5] compared to previous schemes. Simulation results confirm significant improvement in the communication cost. Additionally, our new constructions are more flexible in terms of system parameters. We are currently investigating other coding techniques, such as Polar codes, in the context of this application.

References

[1] H. Saleh, S. Avdoshin, and A. Dzhonov, *"Platform for Tracking Donations of Charitable Foundations Based on Blockchain Technology,"* Actual Problems of Systems and Software Engineering (APSSE), Nov. 2019.

[2] A. Foti, and D. Marino, *"Blockchain and charities: A systemic opportunity to create social value,"* Economic and Policy Implications of Artifcial Intelligence, pp. 145-148, Springer, 2020.

[3] M. Jirgensons, and J. Kapenieks, *"Blockchain and the future of digital learning credential assessment and management,"* Journal of Teacher Education for Sustainability, vol. 20, no. 1, 2018.

[4] R. Zambrano, A. Young, and S. Velhurst, *"Connecting Refugees to Aid through Blockchain Enabled Id Management: World Food Programme's Building Blocks,"* GovLab, Oct. 2018.

[5] P. Sheng, B. Xue, S. Kannan, and P. Viswanath,*"ACeD: Scalable Data Availability Oracle,"* arXiv preprint arXiv:2011.00102, Oct. 2020.

[6] M. Al-Bassam, A. Sonnino, and V. Buterin, *"Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities,"* arXiv preprint arXiv:1809.09044, Sept. 2018.

[7] M. Yu, S. Sahraei, S. Li, S. Avestimehr, S. Kannan, and P. Viswanath, *"Coded Merkle Tree: Solving Data Availability Attacks in Blockchains,"* International Conference on Financial Cryptography and Data Security, pp. 114-134, Springer, Cham, Feb. 2020.

[8] D. Mitra, L. Tauz, and L. Dolecek, *"Concentrated Stopping Set Design for Coded Merkle Tree: Improving Security against Data Availability Attacks in Blockchain Systems,"* 2020 IEEE Information Theory Workshop (ITW), pp. 1-5, Apr. 2021.

[9] S. Cao, S. Kadhe, and K. Ramchandran, *"CoVer: Collaborative Light-Node-Only Verification and Data Availability for Blockchains,"* IEEE International Conference on Blockchain (Blockchain), pp. 45-52, Nov. 2020.

[10] D. Perard, J. Lacan, Y. Bachy, and J. Detchart, *"Erasure Code-Based Low Storage Blockchain Node,"* IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), pp. 1622-1627, Jul. 2018.

[11] D. Mitra and L. Dolecek, *"Patterned Erasure Correcting Codes for Low Storage-Overhead Blockchain Systems,"* Asilomar Conference on Signals, Systems, and Computers pp. 1734-1738, Nov. 2019.

[12] S. Kadhe, J. Chung, and K. Ramchandran, *"SeF: A Secure Fountain Architecture for Slashing Storage Costs in Blockchains,"* arXiv preprint arXiv:1906.12140, Jun. 2019.

[13] A. Tiwari, and V. Lalitha, *"Secure Raptor Encoder and Decoder for Low Storage Blockchain,"* International Conference on COMmunication Systems & NETworkS (COMSNETS), pp. 161-165, Jan. 2021.

[14] D. S. Gadiraju, V. Lalitha, and V. Aggarwal, *"Secure Regenerating Codes for Reducing Storage and Bootstrap Costs in Sharded Blockchains,"* IEEE International Conference on Blockchain (Blockchain), pp. 229-236, Nov. 2020.

[15] B. Choi, J. -y. Sohn, D. -J. Han, and J. Moon, *"Scalable Network-Coded PBFT Consensus Algorithm,"* IEEE International Symposium on Information Theory (ISIT), pp. 857-861, Jul. 2019.

[16] S. Li, M. Yu, C. Yang, A. S. Avestimehr, S. Kannan, and P. Viswanath, *"PolyShard: Coded Sharding Achieves Linearly Scaling Efficiency and Security Simultaneously,"* IEEE Transactions on Information Forensics and Security, vol. 16, Jul. 2020.

[17] C. Wang, and N. Raviv, *"Low Latency Cross-Shard Transactions in Coded Blockchain,"* arXiv preprint arXiv:2011.00087, Oct. 2020.

[18] X.Y. Hu, E. Eleftheriou, and D.M. Arnold,*"Regular and irregular progressive edge-growth tanner graphs,"* IEEE Transactions on Information Theory, vol. 51, no. 1, Jan. 2005.

[19] S. Nakamoto, *"Bitcoin: A Peer to Peer Electronic Cash System,"* 2008. [Online] Available: https://bitcoin.org/bitcoin.pdf.

[20] W. Stadje,*"The Collector's Problem with Group Drawings,"* Advances in Applied Probability, vol. 22, no. 4, JSTOR, 1990.

[21] Y. He, J. Yang, and J. Song, *"A survey of error floor of LDPC codes,"* International ICST Conference on Communications and Networking in China (CHINACOM), Aug. 2011.

[22] X. Jiao, J. Mu, J. Song, and L. Zhou, *"Eliminating small stopping sets in irregular low-density parity-check codes,"* IEEE Communications Letters, vol. 13, no. 6, Jun. 2009.

[23] T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, *"Construction of irregular LDPC codes with low error floors,"* IEEE International Conference on Communications, May 2003.

[24] T. Tian, C. R. Jones, J. D. Villasenor, and R. D. Wesel,*"Selective avoidance of cycles in irregular LDPC code construction,"* IEEE Transactions on Communications, vol. 52, no. 8, Aug. 2004.

[25] S. Kim, J. Kim, D. Kim, and H. Song, *"LDPC Code Construction with Low Error Floor Based on the IPEG Algorithm,"* IEEE Communications Letters, vol. 11, no. 7, Jul. 2007.

Appendix A
Construction of Coded Interleaving Tree

Let the CIT have l layers (except the root), $L_1, L_2, \ldots, L_l$, where L_l is the base layer. For $1 \leq j \leq l$, let L_j have n_j coded symbols, where we use $n_l = M$ in this paper. Let $N_j[i]$, $1 \leq i \leq n_j$, be the ith symbol of the jth layer, where $S_j = \{N_j[i], 1 \leq i \leq Rn_j\}$ and $P_j = \{N_j[i], Rn_j + 1 \leq i \leq n_j\}$ are the set of systematic (data) and parity symbols of L_j, respectively, where we also write $S_j[i] = N_j[i], 1 \leq i \leq Rn_j$. P_j is obtained from S_j using a rate R systematic LDPC code H_j. In the above CIT, hashes of q coded symbols of every layer are batched (concatenated) together to form a data symbol of its parent layer, where the n_j's satisfy $n_j = \frac{M}{(qR)^{l-j}}$, $j = 1, 2, \ldots, l$.

Let $s_j = Rn_j$ and $p_j = (1 - R)n_j$ denote the number of systematic and parity symbols in L_j. Also define $x \bmod p := (x)_p$. The data symbols of L_{j-1} are formed from the coded symbols of L_j as follows (for $1 < j \leq l$):

$$S_{j-1}[i] = N_{j-1}[i] = \mathtt{concat}(\{\mathtt{Hash}(N_j[x]) \,|\, 1 \leq x \leq n_j,$$
$$i = 1 + (x-1)_{s_{j-1}}\}), \quad 1 \leq i \leq s_{j-1},$$

where $\mathtt{Hash}$ is a hash function (whose output size is y) and $\mathtt{concat}$ represents the string concatenation function. The CIT has a root which consists of t hashes. The CIT allows to create a Proof of Membership (POM) for each base layer coded symbol (which consists of a data and a parity symbol from each intermediate layer of the tree). In particular, the POM of symbol $N_l[i]$ is the set of symbols $\{N_j[1 + (i-1)_{s_j}], N_j[1 + s_j + (i-1)_{p_j}] \,|\, 1 \leq j \leq l - 1\}$. The POMs have the *sibling* property [5], i.e., for each layer j, $1 \leq j < l - 1$, the data part of the POM from layer j is the parent of the two symbols of the POM from layer $j + 1$. In other words, $N_j[1 + (i - 1)_{s_j}]$ is the parent of $N_{j+1}[1 + (i - 1)_{s_{j+1}}]$ and $N_{j+1}[1 + s_{j+1} + (i - 1)_{p_{j+1}}]$. By parent, we mean that $N_j[1 + (i-1)_{s_j}]$ contains the hashes of $N_{j+1}[1 + (i-1)_{s_{j+1}}]$ and $N_{j+1}[1 + s_{j+1} + (i-1)_{p_{j+1}}]$. The POM of a symbol from any intermediate layer j of the tree $1 < j < l$ similarly consists of a data and a parity symbol from each layer above layer j. In particular, POM of the symbol $N_j[i]$ is the set of symbols $\{N_{j'}[1 + (i-1)_{s_{j'}}], N_{j'}[1 + s_{j'} + (i-1)_{p_{j'}}] \,|\, 1 \leq j' < j\}$ and they also satisfy the *sibling* property. The POM of a coded symbol is its Merkle proof [19] and is used to check the inclusion of the coded symbol in the tree (w.r.t to the CIT root). Note that with the described POM for each symbol, the process of checking the Merkle proof is same as for regular Merkle trees in [19].

Let X_j be the size of one coded chunk of layer j along with its POMs which involves a data and parity symbol from each layer above layer j as defined in Section III. Note that

the size of each base layer coded chunk is $\frac{b}{RM}$, where b is the block size. Thus,

$$X_l = \frac{b}{RM} + [(q-1)y + qy](l-1) = \frac{b}{RM} + y(2q-1)(l-1)$$

where the term $(q-1)$ arises due to the fact that of the q hashes present in the data symbol of the POM from layer j, $1 \leq j < l-1$, the hash corresponding to the data symbol of the POM from layer $(j+1)$ is not communicated in the POM (since, due to the *sibling* property, it can be calculated by taking a hash of the data symbol of the POM from layer $(j+1)$). Similarly, of the q hashes present in the data symbol of the POM from layer $l-1$, the the hash corresponding to the actual base layer data chunk is not communicated. Thus we only get $(q-1)$ hashes from each layer for the data part in the POMs. Similarly, $X_j = qy + y(2q-1)(j-1)$, $1 \leq j < l$.

APPENDIX B
PROOF OF LEMMA 1

Let $\rho = \frac{\gamma N k}{M}$, and $x = e^\rho$. The condition $k > \frac{M}{N\gamma} \ln \frac{1}{1-\eta}$ is equivalent to $x > \frac{1}{\eta}$ and the condition $P^{\mathrm{UB}}(\eta, N, M, k, \gamma) \leq p_{th}$ can be simplified to $x^2(v\bar{\eta} - \bar{\eta}^2) + (2\bar{\eta} + v)x - 1 \leq 0$. For $N = \frac{M(1-\eta) + \ln(p_{th})}{H_e(\gamma)}$, $v = \bar{\eta}$ and hence we need $x \leq \frac{1}{3\bar{\eta}}$ which is not possible for $x > \frac{1}{\eta}$ (note that $\bar{\eta} > 0$). For $N > \frac{M(1-\eta) + \ln(p_{th})}{H_e(\gamma)}$, $v > \bar{\eta}$ and hence $x^2(v\bar{\eta} - \bar{\eta}^2) + (2\bar{\eta} + v)x - 1$ is an upward facing quadratic equation with roots of opposite sign. Since x is always positive, $x^2(v\bar{\eta} - \bar{\eta}^2) + (2\bar{\eta} + v)x - 1 \leq 0$ iff $x \leq x_{\max} = \frac{-(2\bar{\eta} + v) + \sqrt{8\bar{\eta}v + v^2}}{2\bar{\eta}(v - \bar{\eta})}$, where $x_{\max}$ is the positive root of the quadratic equation. However, a quick algebraic check would reveal that $x_{\max} < \frac{1}{\eta}$ for $v > \bar{\eta}$ and hence there is no feasible x which satisfies $x^2(v\bar{\eta} - \bar{\eta}^2) + (2\bar{\eta} + v)x - 1 \leq 0$. Thus, for $N \geq \frac{M(1-\eta) + \ln(p_{th})}{H_e(\gamma)}$, $P^{\mathrm{UB}}(\eta, N, M, k, \gamma) > p_{th}$ $\forall k > \frac{M}{N\gamma} \ln \frac{1}{1-\eta}$.

For $N < \frac{M(1-\eta) + \ln(p_{th})}{H_e(\gamma)}$, $v < \bar{\eta}$. In this case $x^2(v\bar{\eta} - \bar{\eta}^2) + (2\bar{\eta} + v)x - 1$ is a downward facing quadratic equation with roots $x_{\max} = \frac{-(2\bar{\eta} + v) - \sqrt{8\bar{\eta}v + v^2}}{2\bar{\eta}(v - \bar{\eta})}$ and $x_{\min} = \frac{-(2\bar{\eta} + v) + \sqrt{8\bar{\eta}v + v^2}}{2\bar{\eta}(v - \bar{\eta})}$ satisfying $x_{\min} < \frac{1}{\eta} < x_{\max}$. In this situation, $x^2(v\bar{\eta} - \bar{\eta}^2) + (2\bar{\eta} + v)x - 1 \leq 0$ iff $x \geq x_{\max}$ which is equivalent to $k \geq \frac{M}{N\gamma} \ln \left(\frac{-(2\bar{\eta} + v) - \sqrt{8\bar{\eta}v + v^2}}{2\bar{\eta}(v - \bar{\eta})} \right)$.

APPENDIX C
PROOF OF LEMMA 2

We use the following result from [20].

Lemma 4. *([20]) Let S be a set of s elements and let $A \subseteq S$, $|A| = l$. From S, let T subsets of size m be drawn with replacement, each subset drawn uniformly at random from all subsets of size m of S. Let $X_T(A)$ be the number of distinct elements of the set A contained in the above T drawings. Then*

$$\mathrm{Prob}(X_T(A) \leq n) := \chi(n, l, s, T, m)$$

$$= \sum_{j=0}^n (-1)^{n-j} \binom{l}{j} \binom{l-j-1}{l-n-1} \left[\frac{\binom{s-l+j}{m}}{\binom{s}{m}} \right]^T.$$

Now, following in a manner similar to [5, Appendix A]

$\mathrm{Prob}(\mathcal{C}$ is not μ-SS-valid$)$

$= \mathrm{Prob}(\exists S$ such that $|S| = \gamma N, | \cup_{i \in S} A_i| \leq M - \mu)$

$\leq \sum_{S \subseteq [M]: |S| = \gamma N} \mathrm{Prob}(| \cup_{i \in S} A_i| \leq M - \mu)$

$= \sum_{S \subseteq [M]: |S| = \gamma N} \chi(M - \mu, M, M, \gamma N, k)$

$= \binom{N}{\gamma N} \chi(M - \mu, M, M, \gamma N, k)$

$\leq e^{NH_e(\gamma)} \chi(M - \mu, M, M, \gamma N, k)$

$= e^{NH_e(\gamma)} P_f$

where similar to [5], we have used the fact that $\binom{N}{\gamma N} \leq e^{NH_e(\gamma)}$ and $\mathrm{Prob}(| \cup_{i \in S} A_i| \leq M - \mu) = \mathrm{Prob}(X_T(A) \leq n)$ when $S = A$, $l = s = M$, $n = M - \mu$, $m = k$ and $T = \gamma N$.

APPENDIX D
PROOF OF LEMMA 3

In the secure phase of Dispersal protocol 1, for each layer j, $1 \leq j \leq l$, all stopping sets (of H_j) of sizes $< (n_j - \left\lceil \left(\frac{M-\mu+1}{M} \right) n_j \right\rceil + 1)$ are securely dispersed. Hence a peeling decoder will never fail to decode layer j, $1 \leq j \leq l$ due to these stopping sets.

Furthermore, the valid phase of Dispersal protocol 1 ensures that every γ fraction of the oracle nodes have at least $M - \mu + 1$ distinct base layer coded chunks with probability $\geq 1 - p_{th}$. Thus, due to the repetition property described in Section II, this ensures that for a given layer j, $1 \leq j < l$, every γ fraction of the oracle nodes have has at least $\frac{M-\mu+1}{M}$ fraction of distinct coded chunk, or at least $\left\lceil \left(\frac{M-\mu+1}{M} \right) n_j \right\rceil$ distinct coded chunks. Thus, with probability $\geq 1 - p_{th}$, the dispersal protocol is $(n_j - \left\lceil \left(\frac{M-\mu+1}{M} \right) n_j \right\rceil + 1)$-SS-valid for each layer j, $1 \leq j \leq l$.

Since the CIT root is committed only when $\gamma + \beta$ fraction of the oracle nodes vote that they received correct coded chunks, this implies that at least γ fraction of honest oracle nodes have received correct coded chunks. Now, since with probability $\geq 1 - p_{th}$, the dispersal protocol is $(n_j - \left\lceil \left(\frac{M-\mu+1}{M} \right) n_j \right\rceil + 1)$-SS-valid for each layer j, a peeling decoder can successfully decoder layer j for all stopping sets of size $\geq (n_j - \left\lceil \left(\frac{M-\mu+1}{M} \right) n_j \right\rceil + 1)$ by downloading the coded chunks from the above honest γ fraction of oracle nodes who voted that they received correct coded chunks.

Combining the above two situations, the decoder can decode the entire CIT if the block is committed. Hence Dispersal Protocol 1 guarantees availability with probability $\geq 1 - p_{th}$.