Filecoin Proof of Useful Space - Technical Report

CryptoNet - Protocol Labs
{irene, luca} at protocol.ai

Last edited: 30th Aug 2023

What to expect from this document:

- A simple formal definition of Proof of Space (taken from the academic literature), an informal definition of persistent and useful space (needed for Filecoin);
- Construction details and security proof for the *Stacked-DRGs* proof of space (aka "SDR");
- A short introduction to the Filecoin protocol;
- The complete description of how SDR is used in Filecoin. In particular, description and analysis for the following Filecoin sub-protocols: *PoRep*, *WindowPoSt* and *WinningPoSt*.

Contents

1 Proofs of Persistent and Useful Space

1.1 Proof of Space Definition

We start this report recalling a simple definition of *Proof of Space* (PoS). This concept was introduced by Dziembowski et al. in [DFKP15]. Informally, a PoS is a 2-party protocol where a verifier V uses a small amount of storage and computation to check that a prover P used some disk space (storage) for some time. More in detail (we use the same definition as in [Fis19]):

Definition 1.1. A Proof of Space (PoS) is an interactive protocol between two random access machines, the prover P and the verifier V, that is executed in two phases:

- *Initialization*: $(V, P)(id, N) \to (\Phi, S)$. That is, the verifier and the prover on common input an identifier id and a storage bound $N \in \mathbb{N}$ run an interactive protocol that outputs Φ of size $polylog(N)$ for the verifier and S (called advice) of size N for the prover.
- *Execution*: $(V(\Phi), P(S)) \to (1/0, \emptyset)$. After the initialization phase is successfully (*i.e.*, with no abort from any party) executed once, the verifier on input Φ and the prover on input S repeatedly run an interactive protocol with no output for P and a binary output for V.

And with the following properties:

- *Efficiency.* All the messages exchanged between the two parties is $polylog(N)$, the running time of V during both phases is $polylog(N)$, and P runs in $polylog(N)$ during an execution. Note that the prover is allowed to run in time $poly(N)$ during the initialization, and the best we can have is $\Omega(N)$.
- *Completeness.* A PoS is complete when if P and V follow the protocol, then initialization succeeds and V outputs 1 during any execution phase with overwhelming probability.
- *Execution Soundness.* A PoS is (N_0, T, p)-sound if after a successful initialization phase, a verifier interacting with a malicious prover $\tilde{P}$ with the following two constrains outputs 1 during an execution with probability less than p. Constraints:
  1. $\tilde{P}$'s persistent storage after the initialization is $\leq N_0$ (*i.e.*, $\tilde{P}$ continuously stores S' that has size $\leq N_0$);
  2. $\tilde{P}$' during the execution runs in $\leq T$ steps.

  The value $\epsilon = (N - N_0)/N$ is called the *spacegap*.

We assume that storage has a basic unit (*e.g.*, the size of the output of an hash function, 32 bytes) and saying that the storage is less than N means less than N units. We also assume a set of elementary operations (*e.g.*, how many calls to an hash function) and "running in X steps" means that the total number of operation for completing a task of is X.

Notice that in the definition considered here, $\tilde{P}$'s transient (*i.e.*, temporary) storage during the execution is not bounded. That is, the adversary can use "spikes of storage", but this does not help passing the execution phase. This is more general than the original definition [DFKP15], and the motivation for this is the focus on persistent space.

In the Filecoin system, we use the name **Proof of Spacetime (PoSt)** to indicate the repetitions of the execution phase according to a precise schedule. The schedule is decided in such a way that we can argue that for a successful prover, running $\geq T$ steps is not possible or too expensive (more details in the next section). Therefore persistent storage (space thorough time = "Spacetime") is guaranteed by each PoSt. We use the name **PoRep** to indicate the initialization phase with a proof of its correct implementation. See Sections 4 and 5 for full details.

1.2 Persistent Space Security Models

In this section we discuss the different security models and trade-off between computation and storage allowed by Definition 1.1.

The constrains for $\tilde{P}$ in Definition 1.1 imply that a prover who wants to pass the PoS execution phase can choose between two allowed strategies:

- Either storing more than N_0 units persistently since the last initialization phase (we often refer to this as *honest strategy*);
- Or storing less and spending some resources during each execution (by running $> T$ operations) to recompute the missing part needed to pass the execution (we refer to this as *regeneration attack*).

In an application of PoS where the goal is forcing the prover to store, we want to avoid the possibility for the prover to choose the second strategy. If we mange to do this, then when the execution always succeeds, we can say that the prover is storing $> N_0$ units of storage persistently with probability $> (1 - p)$.

In the following we briefly describe two possible ways to assure that even a malicious P can not choose the computation over storing. Consider a (N_0, T, p)-sound PoS instantiated in such a way that the execution step is repeated every s seconds:

1. **Latency model**: latency bound on P for the execution.

   In this case, given an estimate of how many seconds an elementary operation requires (we call this the "time heuristic assumption") with the best machine available, choose the parameter s less or equal to the time required to run the T operations. This implies that P does not choose the regeneration attack strategy simply because he can not (ie, with probability $1-p$, P he doesn't have time for doing the required computation).

Note that to properly compute the latency of the regeneration attack is necessary to consider the *parallel running time*: that is elementary operations that are independent and can be run in parallel count as one.

This is the model of Stacked-DRGs and ZigZag PoPS [Fis19]. Filecoin uses this model for the *WinningPoSt* protocol (see Section 5.2).

2. **Cost model**: cost estimation on P for the execution.

   In this case, we need an estimate about the price of storing one storage unit per second and the price of an elementary operation (*e.g.*, hash computation or memory access), we call this the "cost heuristic assumptions". Then the parameter s is such that the expected cost of running $> T$ operations with probability at least p every s seconds is higher that the cost of storing N units for s seconds. If we assume that the prover is a *rational* player (*i.e.*, P always chooses the strategy that allows him to pass a execution phase and that costs less money) then the former condition guarantees persistent storage since P prefers the honest strategy over the regeneration attack.

   Filecoin uses this model for the *WindowPoSt* protocol (see Section 5.1).

While the two models mentioned above are the only ones we consider in the rest of this document, there can be others models that are interesting for other application. For example, a "Restricted-Computation model" can be defined. In this case, we consider a PoS with $T = O(2^\lambda N)$ and λ large enough to say that no efficient (computationally bounded) prover can choose the regeneration attack strategy for the execution. While at the same time the prover can run the initialization because it gets access to specific hardware setup (eg, a distributed network of computers) that makes possible run $O(2^\lambda N)$ steps.

1.3 Useful Space

In Filecoin we are interested in "useful space", that is storage space that is used to keep real-world data. Therefore, we want that the advice S of the PoS to encode some real data D instead of just being a random incompressible sequence of bytes.

A simple way to achieve this goal is the following: add the input D (data) for the prover in the initialization phase and make the *id* used in initialization a function a Comm_d, a commitment to the data known also by the verifier. The "new" advice stored by the prover, called the **replica**, is defined as $R = S + D$.

Later on in this report (see Section 2), we show how to apply this idea to a specific PoS and get a proof of useful space that is actually a **Proof of Replication** [Fis18, Fis19]. Informally, a proof of replication is cryptographic proof that some disk space is used for storing k retrievable replicas of a data file (the proof is efficiently verifiable). Respect to PoS, in addition to the space hardness property (definition 1.1), the replica R has the *extraction* property. This guarantees the existence of an extraction algorithm that can recover the original data D from the interaction with a successful prover during the execution phases. In [Fis18], it is proved that these two properties together, space-hardness plus extraction, gives the ϵ-*rational replication* property. That is, a prover in Filecoin asked to store two of more copies of the same data (*e.g.*, by running multiple instances of an SDR proof), does not save storage if it decides to make the replicas dependent. In other words, storing the data in a replicated format is the rational strategy.

SnapDeal: Informally, a **Proof of Useful Space** (PoUS) is a cryptographic proof that some disk space is used for storing a retrievable replica of a data file. Respect to proof of replication, we do not have the rational replication property, while extractability and space-hardness still hold. The Filecoin Protocol Improvement (FIP) proposed in FIP0019 ("*SnapDeal Protocol*") can be seen as a compiler that can compile any PoS in a PoUS via a simple encoding function for the data. This is the high level idea: let S be the advice created using a PoS and D the data we want to store (assume they have the same size), then define the replica R (ie, encoded data) as $R = S + \mathsf{Hash}(S, D) \cdot D$ (ie, randomize the data and then add it to the advise). It is proven in FIP0019 analysis that it is possible to define $\mathsf{Hash}(S, D)$ and a component-wise multiplication $\cdot$ in such a way that R has space-hardness property with parameters very close to the one of the original string S.

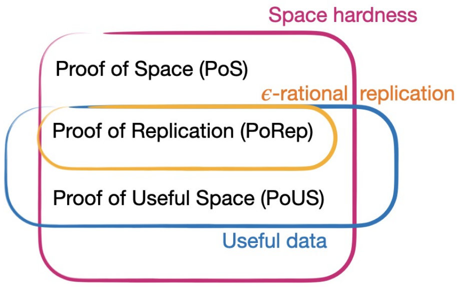

Figure 1: Different proof schemes properties.

2 Stacked-DRGs Proof of Useful Space

In this section we present the proof of useful space that is currently used in Filecoin. We start from some background on graphs, then we present the protocol itself and finally we prove the space hardness property.

2.1 Background on Graphs

1. Numbering for nodes starts from 1. For example, if we have n nodes, the first on the left is node 1, the last on the right is node n. If we have graph with ℓ layer, then the first layer on the top is layer 1 and the last one (*i.e.*, the replica) is layer ℓ.
2. A directed bipartite[1] graph with n nodes in each layer is a (n, α, β) **bipartite expander** if any set of αn sinks (*i.e.*, nodes in the lower layer) are connected to at least βn sources (*i.e.*, nodes in the upper layer). Here $0 < \alpha < \beta < 1$.

[1]A bipartite graph is a graph whose nodes can be divided into two disjoint and independent sets U and V (*e.g.*, upper layer and lower layer) and such that every edge connects a vertex in U to one in V.

3. A directed acyclic graph (DAG) with n nodes is a $(n, 0.80, \beta^*)$ depth-robust graph (**DRG**) if any set of $0.8n$ (or more) nodes contains a path of length $\geq \beta^* n$ (β^* is a constant < 0.8).

   (after removing at most 20% of nodes, there is still a direct path of $\geq \beta^* n$ nodes)

   Note: For a directed path $p = (v_1, v_2, \ldots, v_z)$ in the a graph (v_i are nodes) its length is the number of nodes it traverses $length(p) = z$. The depth $d = depth(G)$ of DAG G is the length of the longest directed path in G.

4. Definition of **Stacked-DRGs graph**: $\mathcal{G}_{\ell,n}$ is a graph with ℓ layers where each layer V_i is a $(n, 0.80, \beta^*)$ DRG of degree 6 constructed via the BucketSample DRG algorithm[2]. Moreover, we add edges in each pair of layers (V_i, V_{i+1}) following the randomized Chung's construction for regular bipartite graphs with degree 8 (edges from layer i to layer $i+1$).

   The number of nodes n has to be large enough in order to give negligible probability of failure for Chung's construction and the DRG construction. In Filecoin we have $n = 2^{30}$ or $n = 2^{31}$.

Chung's construction for degree d. Take a bipartite graph with two layers, each with n nodes and sample a random permutation

$$P : \{1, \ldots, d\} \times \{1, \ldots, n\} \to \{1, \ldots, d\} \times \{1, \ldots, n\}$$

We add an edge from j in the upper level to i in the lower level if there are k, k' such that $(k', j) = P(k, i)$. Equivalently, the parents of nodes in the lower layer are defined by the Algorithm 1.

Algorithm 1 Chung parents

```
1: procedure C.PARENTS($i$)
2:     $Parents(i) = \emptyset$
3:     for $k = 1, \ldots, d$ do
4:         evaluate $P$ on the pair $(k, i)$: $(k', j) = P(k, i)$
5:         $Parents(i) = Parents(i) \cup \{j\}$
       return $Parents(i)$
```

Chung's construction gives a bipartite graph where the in-degree and the out-degree are less or equal to d. For large n, this construction give a d-regular graph. See Section 3.2 in [RD16].

Corollary 2.1. *Let $\mathcal{G}$ be a bipartite graph constructed via Chung's construction with $d = 8$, then with overwhelming probability $\mathcal{G}$ is an $(n, \alpha, 2\alpha)$-expander for any $\alpha \leq 1/3$.*

Corollary 2.2. *Let $\mathcal{G}$ be a bipartite graph constructed via Chung's construction with $d = 8$ and let X be a subset of αn sinks. Then with overwhelming probability X is connected with $0.12n$ sources with distinct indices (i.e. the source index is not in X).*

The proofs of these corollaries can be found in Section 2.7 of the eprint version for [Fis19].

2.2 Stacked-DRGs Proof: Initialization and Execution

Consider the graph $\mathcal{G}_{\ell,n}$ defined in Section 2.1 (with degree $d = 8 + 6 = 14$). Define $DRG.Ind(i)$ as the vector of the indices of DRG parents of the node with index i, and similarly define $Exp.Ind(i)$

[2]See Section 4.1 in the paper "Scaling Proof-of-Replication for Filecoin Mining".

as the vector of the indices of expander parents of the node with index i.

Let $H : \{0,1\}^{dm} \to \{0,1\}^m$ be a collision-resistant hash function and $\mathbb{F}$ a finite field. Currently in Filecoin, H is implemented with SHA-256, $m = 256$ and $\mathbb{F}$ is the arithmetic field of BLS12-381. More details here. We indicate with $+$ the addition operation in $\mathbb{F}$.

The Stacked-DRGs proof of useful space is defined as follows[3]:

- **Initialization** (or offline) phase:

  1. (*Input Prep*) The prover collects a vector D of n data blocks $D_i \in \mathbb{F}$, $D = (D_1, D_2, \dots, D_n)$, and computes Comm_D, a vector commitment to it (*i.e.*, Comm_D is the root of the binary Merkle Tree (MT) with data blocks as leaves, SHA-256 is the hash function used in this MT).

     Note: we assume Comm_D to be a public input of the protocol (ie, anyone participating can compute it or it is available on the blockchain).

  2. (*Labeling*) The prover computes the value $e_i^{(j)}$ (*i.e.*, label of node i at level j) for all nodes in every layer (i goes from 0 to $n-1$ and j from 1 to ℓ) as follow:

     $$\begin{aligned} e_i^{(1)} &= H(\tau||i) \quad \text{if node } i \text{ in layer 1 is a source} \\ e_i^{(j)} &= H(\tau||i||(Parents(i,j)) \quad \text{otherwise} \end{aligned}$$

     with $\tau = \mathsf{H}(\mathsf{Tags}||\mathsf{Comm}_D)$ (τ is called $ReplicaID$ in the Filecoin protocol and the Tags collects other useful identifiers and blockchain anchors).

     $$\begin{aligned} Parents(i,1) &= \left((e_k^{(j)})_{k \in DRG.Ind(\chi_i)}\right) \\ Parents(i,j) &= \left((e_k^{(j)})_{k \in DRG.Ind(\chi_i)}||(e_k^{(j-1)})_{k \in Exp.Ind(\chi_i)}\right) \quad \text{for } j > 1 \end{aligned}$$

  3. (*Sealing*) For all $i = 0, \dots, n-1$, the prover converts $e_i^{(\ell)}$ from bit string to value in $\mathbb{F}$ and computes

     $$R_i = D_i + e_i^{(\ell)} \tag{1}$$

     The vector $R = (R_1, R_2, \dots, R_n) \in \mathbb{F}^n$ is called **Replica**.

  4. (*Commitments*) The prover commits to the replica computing $\mathsf{Comm}_{R_{\mathsf{LAST}}}$ as vector commitment to it (*i.e.*, $\mathsf{Comm}_{R_{\mathsf{LAST}}}$ is the root of the octal Merkle tree constructed on the values $R_1, \dots, R_n$ using the Poseidon hash function).

     The prover commits to the labels using what we call *column commitment*: For $i = 1, \dots, n$, compute

     $$C_i = \mathsf{H}_p(e_i^{(1)}||e_i^{(2)}||\dots||e_i^{(\ell)})$$

     (hash of the labels of node i in all the layers except the last one) and then compute Comm_C be the root of the octal Merkle tree constructed on the values $C_1, \dots, C_n$ using H_p as well in the MT, here H_p is the Poseidon hash function.

[3]This is almost the same protocol as described in [Fis19], here we add important implementation details and a new efficient way to commit to labels ("column commitments"). Note also, that we describe the Stacked-DRGs proof in the case where the prover publishes its output on a ledger (ie, a blockchain) and the verifier can be anyone reading it. We assume the existence of a publicly verifiable random beacon. Filecoin uses drand.

Finally, the prover computes $\mathsf{Comm}_R = \mathsf{H}_p(\mathsf{Comm}_C || \mathsf{Comm}_{R_{\mathsf{LAST}}})$.

The value Comm_R is made public (ie, posted on the blockchain).

5. (Openings, aka *Vanilla Proofs*) The prover computes c challenges. The i-th challenge χ_i is computed as the first n bits of $H(\tau, \mathsf{ticket}||i)$, with $i = 1, 2, \ldots, c$, where ticket is the output of a publicly verifiable random beacon at time $x+$ the time when Comm_R is made public (x is a protocol parameter).

   Then, the prover collects and makes public Comm_C, $\mathsf{Comm}_{R_{\mathsf{LAST}}}$ and for each $i = 1, 2, \ldots, c$:

   – the labels $\{e^{(j)}_{\chi_i}\}_{j=1,\ldots,\ell}$ and the path p_{χ_i} for C_{χ_i} to Comm_C;

   – the data block value D_{χ_i} and its path p_{D_i} to Comm_D;

   – the replica block R_{χ_i} and its path p_{R_i} to $\mathsf{Comm}_{R_{\mathsf{LAST}}}$;

   – for each $k \in DRG.Ind(\chi_i)$: $\{e^{(j)}_k\}_{j=1,2,\ldots,\ell}$ and the path p_k for C_k to Comm_C;

   – for each $k \in Exp.Ind(\chi_i)$: $\{e^{(j)}_k\}_{j=1,2,\ldots,\ell}$ and the path p'_k for C_k to Comm_C;

6. (Verification) Anyone that wants to verify the initialization:

   – verify $\mathsf{Comm}_R = \mathsf{H}_p(\mathsf{Comm}_C || \mathsf{Comm}_{R_{\mathsf{LAST}}})$;

   – compute the challenges from ticket;

   – for any challenge,

     * compute $\mathsf{H}_p(e^{(1)}_i || \cdots || e^{(\ell)}_{\chi_i})$ and for this value verify p_{χ_i} respect to Comm_C;
     * verify p_{D_i} respect to Comm_D;
     * verify p_{R_i} respect to $\mathsf{Comm}_{R_{\mathsf{LAST}}}$;
     * for each $k \in DRG.Ind(\chi_i)$: compute $\mathsf{H}_p(e^{(1)}_k || \cdots || e^{(\ell)}_k)$ and for this value verify p_k respect to Comm_C;
     * for each $k \in Exp.Ind(\chi_i)$: compute $\mathsf{H}_p(e^{(1)}_k || \cdots || e^{(\ell)}_k)$ and for this value verify p_k respect to Comm_C;
     * verify that $R_{\chi_i} = D_{\chi_i} + e^{(\ell)}_{\chi_i}$;
     * verify that for all $j = 1, 2, \ldots, \ell$

$$\begin{aligned} e^{(1)}_{\chi_i} &= H(\tau || \chi_i) \quad \text{if node } \chi_i \text{ in layer 1 is a source} \\ e^{(j)}_{\chi_i} &= H(\tau || \chi_i || Parents(\chi_i, j)) \quad \text{otherwise.} \end{aligned}$$

After the initialization phase, the prover stores the replica R (and $\mathsf{Comm}_C, \mathsf{Comm}_{R_{\mathsf{LAST}}}$ and the relative opening information).

- **The jth execution** (or online) phase:

1. (Openings, aka *Vanilla Proofs*) The prover computes k challenges. The i-th challenge χ_i is computed as the first n bits of $H(\mathsf{ticket_j}||i)$, with $i = 1, 2, \ldots, k$, where $\mathsf{ticket_j}$ is the output of a publicly verifiable random beacon at time x_j (x_j is a protocol parameter).

   Then, the prover collects and makes public $\mathsf{Comm}_C, \mathsf{Comm}_{R_{\mathsf{LAST}}}$ and the replica block R_{χ_i} and its path $p_{R_{chi_i}}$ to $\mathsf{Comm}_{R_{\mathsf{LAST}}}$ for all $i = 1, 2, \ldots, k$.

2. (Verification) Anyone that wants to verify the execution:
   - verify $\mathsf{Comm}_R = \mathsf{H}_p(\mathsf{Comm}_C||\mathsf{Comm}_{R_{\mathsf{LAST}}})$;
   - compute the challenges from $\mathsf{ticket_j}$;
   - for any challenge, verify p_{R_i} respect to $\mathsf{Comm}_{R_{\mathsf{LAST}}}$, for each $i = 1, 2, \ldots, k$.

Repeat according the proving schedule (public protocol parameter).

2.3 Stacked-DRGs Security: Pebbling Analysis

In this section we prove the space hardness property of the Stacked-DRGs protocol. We follow the arguments and the analysis structure originally proposed in [Fis19]. However, the final formulas are different since some typos found in the proofs of [Fis19] are corrected in the updated version proposed here.

Recall the definition of Stacked-DRGs graph $\mathcal{G}_{\ell,n}$ in Section 2.1 and consider the following definitions:

Definition 2.1. 1. Let $\mathcal{G}_{\ell,n}[\epsilon, \vec{\delta}]$ indicate the Stacked-DRGs graph $\mathcal{G}_{\ell,n}$ with the following pebble configuration: $(1-\epsilon)n$ (or less) black pebbles overall and $\delta_i n$ (or less) red pebbles in layer V_i for all $i = 1, 2, \ldots, \ell$ and $\vec{\delta} = (\delta_1, \delta_2, \ldots, \delta_\ell)$.

2. Given this initial pebble configuration, a move in the *pebbling game* on $\mathcal{G}_{\ell,n}[\epsilon, \vec{\delta}]$ consists in placing a pebble in a node. This can be done if and only if all parents currently contain pebbles previously placed. A *round* is placing pebbles simultaneously on all available nodes. Removing pebbles is for free.

3. We say that $\mathcal{G}_{\ell,n}[\epsilon, \vec{\delta}]$ is (r, σ)**-parallel hard** if for any set S of σn of nodes in the last layer no player can pebble S in r or less rounds ($\sigma > 1 - \epsilon + \delta$). Note that the former is equivalent to the following: the probability that a node sampled uniformly at random from the last layer can be pebbled in $\leq r$ rounds is $< \sigma$.

   Proof sketch of deterministic hardness $\Rightarrow$ randomized hardness: if $\mathcal{G}_{\ell,n}[\epsilon, \vec{\delta}]$ is (r, σ) parallel hard then less than σn nodes in the last layer can be pebbled individually in t or less rounds. Otherwise, these nodes form a subset S of size $\geq \sigma n$ and they can be pebbled r rounds. This means that the probability a randomly sampled node from the last layer can be pebbled in r or less rounds is less than σ.

Further notation used in the following for the pebble configuration in $\mathcal{G}_{\ell,n}[\epsilon, \vec{\delta}]$:

- $\gamma = 1 - \epsilon$;
- ρ_i is the fraction of nodes with black pebbles in layer V_i;

- $\gamma_i = \sum_{j=1}^{i-1} \rho_j$ (*i.e.*, $\gamma_i n$ is the number of black pebbles placed before level i);
- U_i is defined as the set of unpebbled nodes in the layer V_i.

Claim 2.3 (Claim 2 in ePrint 2018/702)**.** *If $\mathcal{G}_{\ell,n}[\epsilon, \vec{\delta}]$ is (r, σ)-parallel hard, then the Stacked-DRG proof described in Section 2.2 is $((1-\epsilon)nm, r, p)$-sound, with $p = \max\{(1-\delta_i)^{c_i}, \sigma^k\}$. See Definition 1.1 with one step corresponding to a pebbling round.*

Proof. (Sketch) Consider the labels to which the prover committed to (assuming a perfectly binding commitment) and let x_i be the number of wrong labels in layer i in this labeling. We have two cases:

- Case 1: there exists an index j such that $x_j > \delta_j n$. Then, the probability that V accepts the PoS is less of equal of the probability that the initialization phase succeeds and this is

$$\leq \Pr[\,\text{check at level j are ok}\,] < (1-\delta_j)^{c_j} \leq \max_i\{(1-\delta_i)^{c_i}\}$$

- Case 2: for all i it holds that $x_i \leq \delta_i n$. In this case we are in the configuration in which $\mathcal{G}_\ell[\epsilon, \vec{\delta}]$ is (t, σ)-hard and therefore the the probability that V accepts the PoS is bounded by σ^k.

□

Claim 2.4 (Claim 3 in ePrint 2018/702)**.** *$\mathcal{G}_\ell[\epsilon, \vec{\delta}]$ is $(r, 1)$-parallel hard if and only if any set of size αn of unpebbled nodes in the last layer with $\alpha - \gamma_\ell \geq \epsilon - \delta_\ell$ requires $r+1$ rounds to be pebbled (can't be pebble in r rounds).*

Proof sketch: ($\Leftarrow$) in the last layer there are $\geq (1-\rho_\ell-\delta_\ell)n$ unpebbled nodes. Set $\alpha = 1-\rho_\ell-\delta_\ell$. ($\Rightarrow$) if $\mathcal{G}_{\ell,n}[\epsilon, \vec{\delta}]$ is $(r, 1)$ parallel hard then pebbling all the last layer where there are $\geq (1-\rho_\ell-\delta_\ell)n$ requires $r+1$ rounds.

Claim 2.5 (Claim 4 in ePrint 2018/702, corrected)**.** *If $\mathcal{G}_{\ell-1}[\epsilon', \vec{\delta}]$ with $\epsilon' = \epsilon - 2\delta_{\ell-1}$ is $(r, 1)$ parallel hard and $0 < \epsilon - 2\delta_{\ell-1} \leq 0.24$, then $\mathcal{G}_\ell[\epsilon, \vec{\delta}^*]$ is $(r^*, 1-\epsilon/2)$ parallel hard with $r^* = \min(\beta^* n - 1, r+1)$ and $\vec{\delta}^* = \vec{\delta}$ on all common indices and $\delta^*_\ell = \delta_{\ell-1}$.*

Proof. Let S be a subset of nodes from the last layer in $\mathcal{G}_\ell[\epsilon, \vec{\delta}^*]$ with size $(1-\epsilon/2)n$, we need to show that r^*+1 rounds are required to pebble S. Let X be the subset of S of unpebbled nodes, it is enough to show that X requires r^*+1 rounds to be pebbled. Let $|X| = \alpha n$ and notice that $\alpha \geq \alpha_\ell - \epsilon/2 \geq \epsilon/2 - \delta^*_\ell > 0$ (for the 2nd inequality we use that $\alpha_\ell = 1-\delta^*_\ell - \rho_\ell = \geq 1-\delta^*_\ell - \gamma = \epsilon - \delta^*_\ell$). Now, consider three cases:

1. Case $\alpha \leq 1/3$. Because of Corollary 2.1, we have that the nodes in X are connected to $2\alpha n$ nodes in layer $V_{\ell-1}$. Among these nodes, at least a fraction $\alpha' \geq 2\alpha - \rho_{\ell-1} - \delta^*_{\ell-1}$ are unpebbled. From this and $\vec{\delta}^* = \vec{\delta}$, we have that

$$\alpha' \geq 2\alpha_\ell - \epsilon - \rho_{\ell-1} - \delta_{\ell-1}$$

using $\gamma_{\ell-1} = \gamma - \rho_\ell - \rho_{\ell-1}$:

$$= 2\alpha_\ell - \epsilon + (\gamma_{\ell-1} - \gamma + \rho_\ell) - \delta_{\ell-1}$$

using $\alpha_\ell = 1 - \rho_\ell - \delta^*_\ell$ and $\delta^*_\ell = \delta_{\ell-1}$:

$$= \alpha_\ell - \epsilon + \gamma_{\ell-1} - \gamma + 1 - 2\delta_{\ell-1}$$

using $\epsilon = 1 - \gamma$:

$$= \alpha_\ell + \gamma_{\ell-1} - 2\delta_{\ell-1}$$

And therefore

$$\alpha' - \gamma_{\ell-1} \geq \alpha_\ell - 2\delta_{\ell-1} \geq \epsilon - 3\delta_{\ell-1} \tag{2}$$

For the last inequality we use $\alpha_\ell \geq \epsilon - \delta_{\ell-1}$. Now, consider the graph $\mathcal{G}_{\ell-1}[\epsilon', \vec{\delta}]$ with $\epsilon' = \epsilon - 2\delta_{\ell-1}$ and the following constrain in its pebble configuration: the number of black pebbles from layer 1 to layer $\ell - 2$ is $\gamma_{\ell-1}n$ (the same number as in $\mathcal{G}_\ell[\epsilon, \vec{\delta}^*]$). Then, (2) says that we can apply Claim 2.4, and therefore the fact that $\mathcal{G}_{\ell-1}[\epsilon', \vec{\delta}]$ is $(r, 1)$ parallel hard implies that at least $r+1$ rounds are required to pebble the unpebbled nodes among the βn dependency of X. Finally, X needs $r+2$ rounds to be pebbled in in $\mathcal{G}_\ell[\epsilon, \vec{\delta}^*]$. So in this case $r^* + 1 = r + 2$.

2. Case $1/3 < \alpha < 0.8$. Let β be the fraction of nodes in layer $V_{\ell-1}$ to which the nodes in X are connected. Among these nodes, at least a fraction $\alpha' \geq \beta - \rho_{\ell-1} - \delta^*_{\ell-1}$ are unpebbled. From this and $\vec{\delta}^* = \vec{\delta}$, we have that

$$\alpha' \geq \beta - \rho_{\ell-1} - \delta_{\ell-1} = \beta + (\gamma_{\ell-1} - \gamma + \rho_\ell) - \delta_{\ell-1}\,.$$

And therefore:

$$\alpha' - \gamma_{\ell-1} \geq \beta - \gamma + \rho_\ell - \delta_{\ell-1}$$

using $\alpha_\ell = 1 - \rho_\ell - \delta_{\ell-1}$:

$$\geq \beta - \gamma + (1 - \alpha_\ell - \delta_{\ell-1}) - \delta_{\ell-1}$$

using $\alpha \geq \alpha_\ell - \epsilon/2$:

$$\begin{aligned} &\geq \beta - \gamma + 1 + (-\alpha - \epsilon/2) - 2\delta_{\ell-1} \\ &\geq \beta - \alpha + \epsilon/2 - 2\delta_{\ell-1} \\ &\geq 0.12 + \epsilon/2 - 2\delta_{\ell-1} \end{aligned}$$

The last inequality is because of Corollary 2.2. Moreover if $\epsilon - 2\delta_{\ell-1} \leq 0.24$, then we have that $0.12 + \epsilon/2 - 2\delta_{\ell-1} \geq \epsilon - 3\delta_{\ell-1}$. Therefore it holds again that $\alpha' - \gamma_{\ell-1} \geq \epsilon - 3\delta_{\ell-1}$. Now, we can again consider the graph $\mathcal{G}_{\ell-1}[\epsilon', \vec{\delta}]$ with $\epsilon' = \epsilon - 2\delta_{\ell-1}$ and conclude using Claim 2.4 as we did in the case before.

3. Case $\alpha \geq 0.8$. Because of the depth-robustness, we know that there are $\geq \beta^* n$ nodes that are unpebbled and form a direct path. These implies that $\geq \beta^* n$ rounds are required to pebble X. So in this case $r^* + 1 = \beta^* n$.

□

From now on, we consider $\delta_i = \delta$ for all $i = 1, 2, \ldots, \ell$ (*i.e.*, $(1-\epsilon)$ black pebbles overall and δ red pebbles in each layer) and we indicate this configuration with the notation $\mathcal{G}_{\ell,n}[\epsilon, \delta]$.

Claim 2.6 (Claim 5 in ePrint 2018/702)**.** *Consider the graph $\mathcal{G}_\ell[\epsilon, \delta]$ with $0 < \epsilon - 2\delta$ with*

$$\ell \geq \left\lceil \log_2 \left(\frac{1}{3(\epsilon - 2\delta)} \right) \right\rceil + 1.$$

Then the unpebbled nodes of the last layer have unpebbled paths to at least $n/3$ unpebbled nodes in some layer V_i.

Proof. For $i = 1, 2, \ldots, \ell - 1$, let α_i be the fraction of nodes in V_i that are unpebbled dependencies of U_ℓ (*i.e.*, fraction of nodes in U_i that have unpebbled paths to U_ℓ) and $|U_\ell| = \alpha_\ell n$. Let $k = \left\lceil \log_2 \left(\frac{1}{3(\epsilon - 2\delta)} \right) \right\rceil$ and suppose that $\alpha_i < 1/3$ for $i = \ell, \ell - 1, \ldots, \ell - k$. In the following, we will prove that

$$\alpha_{\ell-k} \geq 2^k (\epsilon - 2\delta) \tag{3}$$

Since $k \geq \log_2 \left(\frac{1}{3(\epsilon - 2\delta)} \right)$, (3) implies that $\alpha_{\ell-k} \geq 1/3$. This contradicts the original hypothesis and implies that there exists an index $i \in \{\ell, \ell - 1, \ldots, \ell - k\}$ such that $\alpha_i \geq 1/3$.

To prove (3), recall that Corollary 2.1 implies that if $\alpha_i < 1/3$ then $\alpha_{i-1} \geq 2\alpha_i - \rho_{i-1} - \delta_{i-1}$. Let $\beta_i = \rho_i + \delta_i$, in the following we prove that for any $j = 0, 1, \ldots, k$, it holds that:

$$\alpha_{\ell-j} \geq 2^j \alpha_\ell - \sum_{i=0}^{j-1} 2^i \beta_{\ell-j+i} \tag{4}$$

The proof is done by induction on j: the base case $j = 0$ is trivial. Assuming this hold for j, then for $j + 1$:

$$\begin{aligned}
\alpha_{\ell-(j+1)} &\geq 2\alpha_{\ell-j} - \beta_{\ell-(j+1)} \\
&\geq 2(2^j \alpha_\ell - \sum_{i=0}^{j-1} 2^i \beta_{\ell-j+i}) - \beta_{\ell-(j+1)} \\
&= 2^{j+1} \alpha_\ell - \sum_{i=-1}^{j-1} 2^{i+1} \beta_{\ell-j+i} \\
&= 2^{j+1} \alpha_\ell - \sum_{i=0}^{j} 2^i \beta_{\ell-(j+1)+i}
\end{aligned}$$

From (4), assuming $\delta_i = \delta$ for all i and using that $\sum_{i=0}^{j-1} 2^i = 2^j - 1$, it follows that for all $j = 0, 1, \ldots, k$ it holds the following:

$$\begin{aligned}
\alpha_{\ell-j} &\geq 2^j \alpha_\ell - \sum_{i=0}^{j-1} 2^i \rho_{\ell-j+i} - (2^j - 1)\delta \\
&\geq 2^j \alpha_\ell - 2^{j-1} \sum_{i=0}^{k-1} \rho_{\ell-k+i} - (2^j - 1)\delta \\
&\geq 2^j \alpha_\ell - 2^{j-1} \gamma_\ell - (2^j - 1)\delta \\
&\geq 2^j (\alpha_\ell - \gamma_\ell/2 - \delta)
\end{aligned}$$

using $\gamma_\ell = \gamma - \rho_\ell = \gamma - (1 - \alpha_\ell - \delta_\ell)$

$$= 2^j(\alpha_\ell - (\delta - \epsilon + \alpha_\ell)/2 - \delta)$$

$$= 2^j \frac{\alpha_\ell + \epsilon - 3\delta}{2} \tag{5}$$

using $\alpha_\ell \geq \epsilon - \delta_\ell$

$$\geq 2^{j-1}(\epsilon - \delta + \epsilon - 3\delta)$$

$$= 2^j(\epsilon - 2\delta)$$

Considering the last relation for $j = k$ concludes the proof. □

Claim 2.7 (Claim 6 in ePrint 2018/702, modified)**.** *Consider the graph $\mathcal{G}_\ell[\epsilon, \delta]$ with $\delta < 0.12, 0 < \epsilon - 2\delta$, $\epsilon - 3\delta < 1/3$, and $\ell = \max\{\ell_1, \ell_2, \ell_3\}$ with*

$$\ell_1 = \left\lceil \frac{0.68 - \epsilon + 2\delta}{0.12 - \delta} \right\rceil + 2$$

$$\ell_2 = \left\lceil \log_2\left(\frac{1}{3(\epsilon - 2\delta)}\right) \right\rceil + \left\lceil \frac{0.24}{0.12 - \delta} \right\rceil + 1$$

$$\ell_3 = \left\lceil \log_2\left(\frac{1}{1 + 3(\epsilon - 3\delta)}\right) \right\rceil + \left\lceil \frac{0.57 - \epsilon + \delta}{0.12 - \delta} \right\rceil + 1$$

Then the unpebbled nodes of the last layer have unpebbled paths to at least $0.8n$ unpebbled nodes in some layer V_i.

Proof. Consider the same notation used in the proof of Claim 2.6. Moreover, define $\beta(\alpha)$ as the minimum bipartite expansion factor for a set of size αn (*i.e.*, every set of αn sinks is connected to at least $\beta(\alpha)n$ sources), and $\hat{\beta}(\alpha) = \beta(\alpha) - \alpha$. Using the relation $\alpha_{i-1} \geq \beta(\alpha_i) - \beta_{i-1} = \hat{\beta}(\alpha_i) + \alpha_i - \beta_{i-1}$ (with $\beta_i = \rho_i + \delta_i$), we prove the following bounds:

- Bound 1: $\alpha_i - \gamma_i \geq \epsilon - \delta$ for all $i = \ell, \ell-1, \ldots, 1$ (assuming $\delta_i = \delta$ for all i). We prove this by induction on i (decreasing index). The base case ($i = \ell$) is true because $\alpha_\ell - \gamma_\ell = 1 - \rho_\ell - \delta_\ell - \gamma + \rho_\ell = \epsilon - \delta_\ell$. Assuming this hold for j, then for $j - 1$

$$\alpha_{j-1} \geq \beta(\alpha_j) - \rho_{j-1} - \delta$$

see the comment box below

$$\geq \alpha_j + \delta - \rho_{j-1} - \delta$$

using the inductive hypothesis

$$\geq \gamma_j + \epsilon - \delta - \rho_{j-1}$$

$$\geq \epsilon + \gamma_{j-1} - \delta$$

The inductive hypothesis says that $\alpha_j \geq \gamma_j + \epsilon - \delta$. From the condition $0 < \epsilon - 2\delta$, it follows that $\alpha_j > \gamma_j + \delta$. Then we divide the proof in two cases. Case 1: if $\alpha_i \leq 1/3$, then $\beta(\alpha_i) \geq 2\alpha_i > \alpha_i + \delta$. Case 2: $\alpha_i < 1/3$, then $\beta(\alpha_i) > \alpha_i + 0.12 > \alpha_i + \delta$ when $\delta < 0.12$.

- Bound 2: For any i:

$$\alpha_{i-k} \geq \beta(\alpha_i) + \sum_{j=1}^{k-1} \hat{\beta}(\alpha_{i-j}) - \sum_{j=1}^{k} \beta_{i-j} \quad \text{for all } k = 1, 2, \ldots, i-1 \tag{6}$$

  (by induction on k, using $\alpha_{i-1} \geq \hat{\beta}(\alpha_i) + \alpha_i - \beta_{i-1}$).

  From (6), assuming $\delta_i = \delta$ for all i, it follows that

$$\begin{aligned}\alpha_{i-k} &\geq (k-1)(\min_{j<k} \hat{\beta}(\alpha_{i-j}) - \delta) + \beta(\alpha_i) - \sum_{j=1}^{k} \rho_{i-j} - \delta \\ &\geq (k-1)(\min_{j<k} \hat{\beta}(\alpha_{i-j}) - \delta) + \beta(\alpha_i) - \gamma_i - \delta \qquad (7)\end{aligned}$$

  Using Bound 1:

$$\begin{aligned}&\geq (k-1)(\min_{j<k} \hat{\beta}(\alpha_{i-j}) - \delta) + \beta(\alpha_i) - \alpha_i + \epsilon - 2\delta \\ &\geq (k-1)(\min_{j<k} \hat{\beta}(\alpha_{i-j}) - \delta) + \hat{\beta}(\alpha_i) + \epsilon - 2\delta \qquad (8)\end{aligned}$$

- Bound 3: If there exists i^* such that $\alpha_{i^*} \geq 1/3$, then $\alpha_i > 0.12$ for all $i = i^*, i^* - 1, \ldots, 1$.

  We prove this by induction on $i = i^*, i^* - 1, \ldots, 1$. The base case $i = i^*$ is true by hypothesis. If this hold for i, then for $i - 1$ we use (8) with $k = 1$ and we get

$$\begin{aligned}\alpha_{i-1} &\geq \hat{\beta}(\alpha_i) + \epsilon - 2\delta \\ &\geq 0.12 + \epsilon - 2\delta > 0.12\end{aligned}$$

  In the last inequality we use Corollary 2.2 and $\epsilon - 2\delta > 0$.

Then, we divide the proof in three cases:

1. Case $0.8 \leq \alpha_\ell$. This is trivial.
2. Case $1/3 \leq \alpha_\ell < 0.8$. Because of Bound 3, for all $i = \ell, \ell - 1, \ldots, 1$ it holds that $\alpha_i > 0.12$. Then, we have that for any k, $\min_{j<k} \hat{\beta}(\alpha_{\ell-j}) \geq 0.12$, and (8) becomes

$$\alpha_{\ell-(k+1)} \geq k(0.12 - \delta) + 0.12 + \epsilon - 2\delta$$

   If $\ell \geq k + 2$ with $k = \left\lceil \frac{0.68 - \epsilon + 2\delta}{0.12 - \delta} \right\rceil$, then $\alpha_{\ell-(k+1)} \geq 0.8$.
3. Case $\alpha_\ell < 1/3$. We have already proved in the proof of Claim 2.6 that if $\ell \geq k + 1$ with $k = \left\lceil \log_2 \left(\frac{2}{3(\alpha_\ell + \epsilon - 3\delta)} \right) \right\rceil$ then there exists an index $i^* \in \{\ell, \ell - 1, \ldots, \ell - k\}$ such that $\alpha_{i^*} \geq 1/3$ (see (5)).

Now, using Bound 3 we have that for $i = i^*, i^*-1, \ldots, 1$ it holds that $\alpha_i > 0.12$. Given this we have that for any k', $\min_{j<k'} \hat{\beta}(\alpha_{i^*-j}) \geq 0.12$, and from (7) we get

$$\begin{aligned}
\alpha_{i^*-k'} &\geq (k'-1)(0.12-\delta) + \beta(\alpha_{i^*}) - \gamma_{i^*} - \delta \\
&\geq (k'-1)(0.12-\delta) + \beta(\alpha_{i^*}) - \gamma_\ell - \delta \\
&\geq (k'-1)(0.12-\delta) + \beta(\alpha_{i^*}) - (\alpha_\ell - \epsilon + \delta) - \delta \\
&\geq (k'-1)(0.12-\delta) + \beta(\alpha_{i^*}) - \alpha_\ell + \epsilon - 2\delta \\
&\geq (k'-1)(0.12-\delta) + 0.68 - \alpha_\ell + \epsilon - 2\delta
\end{aligned}$$

we used the fact that $\beta(\alpha_{i^*}) \geq \beta(0.33) \geq 0.68$.

If $i^* \geq k'+1$ with $k' = \left\lceil \dfrac{0.24 + \alpha_\ell - \epsilon + \delta}{0.12 - \delta} \right\rceil$, then $\alpha_{i^*-k'} \geq 0.8$.

This is implied by choosing $\ell = \left\lceil \log_2 \left(\dfrac{2}{3(\alpha_\ell + \epsilon - 3\delta)} \right) \right\rceil + \left\lceil \dfrac{0.24 + \alpha_\ell - \epsilon + \delta}{0.12 - \delta} \right\rceil + 1.$

The derivative wrt α_ℓ is $\frac{1}{0.12-\delta} - \frac{1}{\log_2(\alpha_\ell + \epsilon - 3\delta)}$. Since the derivative is negative for $\log_2(\alpha_\ell + \epsilon - 3\delta) < 0.12 - \delta$ and positive otherwise, the two maximums are on the extreme of the interval $[\epsilon - \delta, 1/3]$:

- $(\alpha_\ell = \epsilon - \delta)$ $\ell = \log_2(\frac{1}{3(\epsilon - 2\delta)}) + \frac{0.24}{0.12-\delta} + 1$
- $(\alpha_\ell = 1/3)$ $\ell = \log_2(\frac{2}{1+3(\epsilon-3\delta)}) + \frac{0.24+0.33-\epsilon+\delta}{0.12-\delta} + 1$

□

Theorem 2.8. *Consider $\mathcal{G}_{\ell,n}[\epsilon + 2\delta, \delta]$ with $\delta < 0.12$, $\epsilon \leq 0.24$, and $0 < \epsilon - 2\delta < 1/3$, and $\ell = \max\{\ell_1, \ell_2, \ell_3\} + 1$ with*

$$\ell_1 = \left\lceil \frac{0.68 - \epsilon + 2\delta}{0.12 - \delta} \right\rceil + 2, \quad \ell_2 = \left\lceil \log_2 \left(\frac{1}{3(\epsilon - 2\delta)} \right) \right\rceil + \left\lceil \frac{0.24}{0.12 - \delta} \right\rceil + 1$$

$$\ell_3 = \left\lceil \log_2 \left(\frac{2}{1 + 3(\epsilon - 3\delta)} \right) \right\rceil + \left\lceil \frac{0.57 - \epsilon + \delta}{0.12 - \delta} \right\rceil + 1.$$

Then the Stacked-DRG PoS associated at this graph (see Section 2.2) with $c = \frac{-\lambda}{\log_2(1-\delta)}$ and $k = \frac{-\lambda}{\log_2(1-\epsilon/2-\delta)}$ is a $((1-\epsilon-2\delta)nm, \beta^* n - 1, 2^{-\lambda})$-sound PoS.

Proof. First we apply Claim 2.7 to $\mathcal{G}_{\ell-1,n}[\epsilon, \delta]$. Since all the conditions are satisfied, we can conclude that $\mathcal{G}_{\ell-1,n}[\epsilon, \delta]$ is $(\beta^* n - 1, 1)$ parallel hard. (Motivation: the unpebbled nodes in the last layer of $\mathcal{G}_{\ell-1,n}[\epsilon, \delta]$ have unpebbled paths to $0.8n$ unpebbled nodes a previous layer. These dependencies requires $\geq \beta^* n$ rounds to be pebbled.) Then we can use Claim 2.5 to conclude that $\mathcal{G}_{\ell,n}[\epsilon + 2\delta, \delta]$ is $(\beta^* n - 1, 1 - \frac{\epsilon}{2} - \delta)$ parallel hard. Finally, since $\mathcal{G}_{\ell,n}[\epsilon + 2\delta, \delta]$ is (r, σ) parallel hard to pebble, with $\sigma = 1 - \frac{\epsilon}{2} - \delta$ and $r = \beta^* n - 1$, then the statement of the theorem is proved just using Claim 2.3.

□

2.4 Stacked-DRGs Proof Formulas Recap

In this last Section, we rewrite in an easier to read way the formulas proved in Theorem 2.8.

Consider the Stacked-DRGs graph $\mathcal{G}_{\ell,n}$ based on a $(n, 0.8, \beta^*)$ DRG as described in Section 2.1 (expander degree is 8, DRG degree is 6).

Parameters:

- ϵ = spacegap;
- δ = fraction of wrong labels that are allowed in a layer of the SDR graph;
- Initialization step soundness: $p = 2^{-\lambda}$;

  Meaning: a malicious prover who puts more that $\geq \delta n$ nodes is able to pass initialization with probability $< 2^{-\lambda}$.
- Execution step soundness: $p' = 2^{-\lambda'}$;

  Meaning: a malicious prover who is storing $\leq (1-\epsilon)n$ nodes is able to pass to execution in $\leq (\beta^* n - 1)$ rounds with probability $< 2^{-\lambda'}$.

Formulas Recap: For any ϵ with $0 < \epsilon \leq 0.48$ and any positive δ such that $(\epsilon/2 - 0.12) \leq \delta < \epsilon/4$, we have that[4]

- Number of offline (initialization) challenges per layer: $c = \frac{-\lambda}{\log_2(1-\delta)}$. Or equivalently, $p = (1-\delta)^c$.
- Number of online (execution) challenges: $k = \frac{-\lambda'}{\log_2(1-\epsilon/2)}$. Or equivalently, $p' = (1-\epsilon/2)^k$.
- Number of layers: $\ell = \max\{\ell_1, \ell_2, \ell_3\} + 1$ with

$$\ell_1 = \left\lceil \frac{0.68 - \epsilon + 4\delta}{0.12 - \delta} \right\rceil + 2, \quad \ell_2 = \left\lceil \log_2\left(\frac{1}{3(\epsilon - 4\delta)}\right) \right\rceil + \left\lceil \frac{0.24}{0.12 - \delta} \right\rceil + 1$$

$$\ell_3 = \left\lceil \log_2\left(\frac{2}{1 + 3(\epsilon - 5\delta)}\right) \right\rceil + \left\lceil \frac{0.57 - \epsilon + 3\delta}{0.12 - \delta} \right\rceil + 1$$

3 Filecoin Protocol Overview

For the rest of this report, we dive deep in the Filecoin protocol and how the proof of useful space presented in the previous section is used there.

Filecoin is a decentralized storage network that turns cloud storage into an algorithmic market. The market runs on a blockchain with a native protocol token (also called "Filecoin"), which miners (called "Storage Providers", SPs) earn by providing storage to clients and mining blocks.

- Time is divided into discrete units we call **epochs**. Storage Providers have semi-synchronised clocks that indicates the current epoch.

[4]Comment: small δ means less layers but more challenges per layer in the off-line phase. See Figure 2 and Table 1.

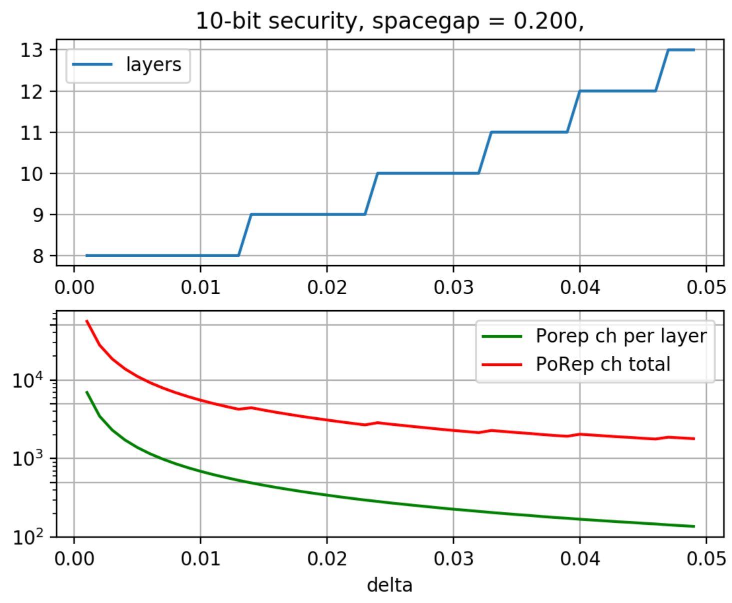

Figure 2: Number of layers and of offline (PoRep) challenges in function of delta.

- A **Storage Provider** (SP) is a node on the network that participates in the protocol and is identified via a providerID.
- A **sector** is the default unit of storage in Filecoin and the "container" for client data. SPs add sectors via the *PoRep* mechanism (see Section 4) and gain units of consensus powers. Each sector has a size (SectorSize in bytes, also called Raw-Byte Power). Currently, we allowed only for 32GiB or 64GiB as sector size. Moreover, each sector has a duration chosen by the SP at the moment of the creation, the SectorDuration in epochs. A sector duration can be extended by the SP before the sector expires, up to a maximum (currently set to 540 days). There is also a minimum set to 180 days. Sectors can be filled with client data or with zeros. In the second case, we call them Committed Capacity (CC) sectors.

  Each SP needs to periodically (every 24h) provide a *WindowPoSt* (see Section 5.1) to prove possession of each of the sectors they added to the network. If the SP fails to do so, they pay penalties and the sectors are marked as faulty. At each epoch e, each active[5] sector is assigned power for that epoch:

$$\mathsf{QualityAdjustedPower}(e) = \mathsf{SectorSize} \times \mathsf{SectorQualityMultiplier}(e).$$

  SectorQualityMultiplier is a value between 1 and 10 that is based on the proportion and kind of deals covering the sector over both (ie, size) and time (ie, duration). Moreover, a faulted

[5]We say a sector is active if the is no fault declarations concerning this sector. See Section 5.1.

Spacegap ϵ	δ	ℓ	tot offline challenges	online challenges
0.025	0.00525	12	15,804	552
	0.001	9	62,361	552
0.25	0.0615	15	1,650	52
	0.005	7	9,681	52
0.20	0.049	13	1,794	66
	0.001	8	55,432	66

Table 1: Example of parameters for 10-bit security ($\lambda = \lambda' = 10$) for SDR. For each spacegap row: in the first row $\delta = \epsilon/4 - 0.001$ (maximum value) and in the second row $\delta = \epsilon/2 - 0.12$ (minimum).

sector is assigned a SectorQualityMultiplier of 0 (*i.e.*, faulted sector do not count for consensus power).

- The NetworkPower at epoch e is the sum of the power at epoch e of all the sectors in the network.

4 Adding Sectors To Filecoin Network

4.1 SDR PoRep Protocol Description

In Filecoin, we use the name "PoRep" for the concrete protocol that implements the initialization step of the Stacked-DRGs proof of useful space (see Section 2.2). This is the method that allows a Storage Provider (SP) to add sectors and later on gain Consensus power in the Filecoin Network.

From now on, we will used "SDR" as short form of "Stacked-DRGs".

In order to register a sector with the Filecoin network, the Storage Provider (SP) runs the PoRep mechanism which ties together: i) the data itself, ii) the actor that performs the action and iii) the time when the specific data has been sealed by the specific SP. In other words, if the same provider attempts to replicate the same data at a later time, then this will result in a different PoRep.

The PoRep protocol is divided in two sequential methods PreCommit and ProveCommit:

- **PreCommit**: Composed by
  - **PC1**: The SP runs steps 1,2 and 3 of the SDR initialization (ie, input preparation, labeling and sealing as described in Section 2.2); the Tags used in the labeling step contains $Prover_{id}$, $Sector_{id}$, SealRandomness (the blockchain height when the action took place) and $PoRep_{id}$.

  - **PC2**: The SP runs step 4 of the SDR initialization (ie, computing column commitments and Merkle Trees root using the Poseidon hashing algorithm, see Section 2.2).

  At the end of PC2, the SP submits on chain Comm_R. Moreover, at this time an ad-hoc collateral called *PCD* (*PreCommit Deposit*) is locked down (as detailed below).

  After PreCommit is completed, the SP has to wait PreCommitChallengeDelay epochs to then read the beacon ticket that is used in the next phase, ProveCommit.

- **ProveCommit**: Composed by
  - **C1**: The SP runs step 5 of the SDR initialization (ie, computing challenges from the beacon ticket and computing the corresponding vanilla proofs);
  - **C2**: The SP runs the prover algorithm of a SNARK proving system[6] to compress in a short SNARK proof all the vanilla proofs. This is the proof submitted on chain.

  Note that there is a deadline for submitting on chain the SNARK proof. If the deadline passes or the proof is invalid (even if submitted in time), PCD is burnt.

  If the SNARK proof is valid, this is considered as a certification that the SP has indeed replicated a copy of the data they agreed to store in R (which is well formed). The sector is added to the proving schedule of WindowPoSt (see Section 5.1) and will give units of consensus power to the SP as soon as the first WindowPoSt is successfully executed and submitted on chain.

The waiting time PreCommitChallengeDelay between the two steps is necessary to avoid the following "short fork attack": a malicious provider reads the beacon ticket for epoch e, computes the challenges for ProveCommit and drafts a replica R not well-formed (ie, with more than δn wrongly-labeled nodes in a layer) such that all wrongly-labelled nodes are not challenged. Then the attacker does a short fork to include its PreCommit message on chain. If the fork succeeds, then it sends also the ProveCommit message and gets to pass PoRep with an not well-formed replica. By running simulations, we know that if PreCommitChallengeDelay ≥ 150 epochs, then for an adversary controlling at most a third of the Network QAP the probability of succeeding in the fork is negligible.

4.2 PreCommit Deposit Analysis

In Filecoin Mainnet, we use the following concrete parameters for SDR:

- $\delta = 0.039$ (fraction of allowed wrongly-labeled nodes per layer);
- $c = 176$ (number of challenges per layer);

which gives a soundness error for the vanilla proofs that is $< 2^{-10}$. In other words, an SP creating a replica R not well-formed (ie, with more than δn wrongly-labeled nodes in a layer) has the probability of passing the verification phase (ie, be able to produce valid vanilla proofs with probability $< 2^{-10}$. This is a small probability, but not negligible. Moreover note that such malicious SP

[6]Currently, in Filecoin the SNARK proving system is Groth16 and when possible the aggregation on top of it given by SnarkPack.

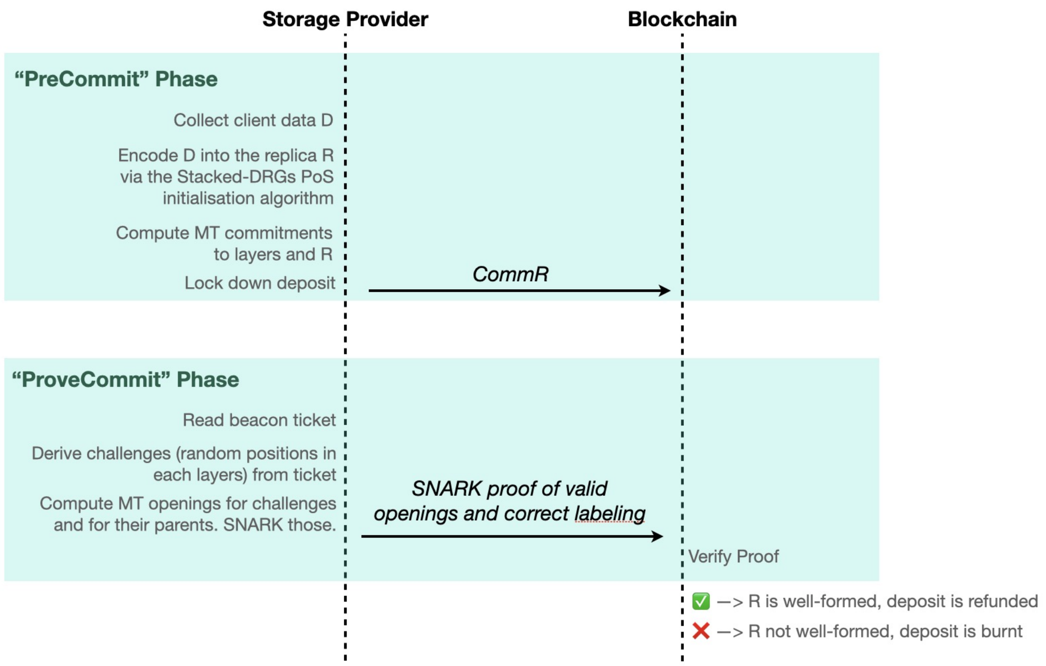
Storage Provider
Blockchain
"PreCommit" Phase
Collect client data D
Encode D into the replica R via the Stacked-DRGs PoS initialisation algorithm
Compute MT commitments to layers and R
Lock down deposit
CommR
"ProveCommit" Phase
Read beacon ticket
Derive challenges (random positions in each layers) from ticket
Compute MT openings for challenges and for their parents. SNARK those.
SNARK proof of valid openings and correct labeling
Verify Proof
✅ —> R is well-formed, deposit is refunded
❌ —> R not well-formed, deposit is burnt

Figure 3: Visual recap of the PoRep mechanism in Filecoin.

(passing verification with more than allowed wrongly-labeled nodes) can be able to "break" space-hardness for the specific sector and gains consensus power for the entire sector duration without storing[7].

To avoid this scenario, we introduced PCD. This is a deposit locked at the end of the PreCommit phase (when the commitment to the replica is published on chain) and burned in case of late or invalid SNARK proof sent in the ProveCommit phase.

Proposition (PCD formula):

Let BR be the expected block reward share for a sector per day[a]. If

$$PCD > \frac{BR \cdot \mathsf{SectorMaxDuration}_{\mathsf{days}}}{2^{10} - 1}$$

then creating a not-well formed replica has negative profit.

[a]Note that BR depends on $\mathsf{QuallityAdjustedPower}$ of the sector.

Proof. Consider an SP that submits a commitment to a not well-formed encoding, its expected profit P is $P = p \cdot BR \cdot \mathsf{SectorMaxDuration}_{\mathsf{days}} - (1-p) \cdot PCD$,with $p = 2^{-10}$. So $P < 0$ if and only

[7]Note we do not know any explicit attack of this form, but we can not prove this is impossible.

if

$$PCD > \frac{p \cdot BR \cdot \mathsf{SectorMaxDuration}_{\mathsf{days}}}{1-p} = \frac{BR \cdot \mathsf{SectorMaxDuration}_{\mathsf{days}}}{2^{10}-1}.$$

□

5 Proving Sectors Over Time In Filecoin

Currently in Filecoin we have two different and independent protocols for auditing storage: WindowPoSt and WinningPoSt. Both of them consist in running the execution step of Stacked-DRGs proof (see Section 2.2) and assume rational players (ie, the SP always chooses the strategy with highest profit), but they differ in the schedule, the security model of the PoS and number of challenges.

WindowPoSt is a periodical audit (ie, the execution step is repeated every 24h) that is run on every sector owned by the SP (ie, the number of execution challenges is linear in the number of sectors). The analysis about honest strategy (ie, storing) being the most profitable is based on the cost of the regeneration attack (ie, the cost model is used, see Section 1.2).

WinningPoSt happens only at leader election time and is run on a randomly chosen sector. This implies a constant number of challenges for auditing. The rationality analysis is based on the latency bound of the regeneration attack (ie, the latency model is used, see Section 1.2).

Full details for both protocols are given in the rest of this chapter.

5.1 Rationality of Storage based on WindowPoSt

WindowPoSt is the name of the proof created to show correctness of an execution step of SDR done on set of replicas with 10 challenges per replica and it is repeated every 24h.

5.1.1 Protocol Description

Storage Provider adds sectors via the PoRep mechanism, see Section 4. At this point, proving proceeds as follows:

- All sectors added by a provider are proved via WindowPoSt in a period of 24h (*proving period*).
- A WindowPoSt is repeated at the end of each period (ie, a proof every 24h per sector).

The way different sectors are organized and proved in Filecoin:

- The sectors owned by an SP are organized in sets called a *partition*: once added the sector is assigned to a partition, and each partition has a fixed maximum number of sectors (for example for 32GiB sector size we have at most 2349 sectors).
- Each partition is assigned to a *deadline*: a specific window of time during which PoSts must be submitted. The proving period (ie, 24h) is broken up into 48 individual deadlines of 30 minutes each.

- Every 24h, 20 epochs (ie, 20×30 seconds $= 10$ minutes) prior to the deadline opening the SP pulls randomness from a random beacon and uses it to create 10 challenges per sector.
- Challenges are used to create the WindowPoSt partition proof (ie, a SNARK compressing the 2349 execution steps, one per replica).
- A commitment to the proof has to be submitted on the chain before the deadline closes.
- If the SP fails to do so or submits a commitment to an invalid proof, sectors are marked as faulty, the SP pays penalties and temporarily loses consensus power for those sectors.

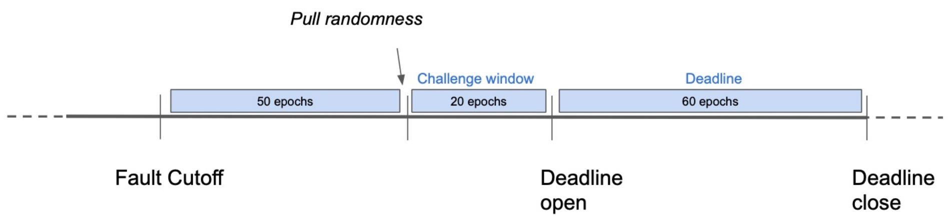

Figure 4: Visual description of a proving period.

More precisely, penalisation works in the following way:

- Each sector has 4 possible states: *unproven, active, faulty, recovering.* Only sectors in the active state give consensus power.
- A sector that is committed (via the PoRep mechanism) but not yet proved by a windowPoSt is called *unproven.*
- A sector becomes active when it is first WindowPoSted.
- The faulty state is obtained for an active or recovering sector as an effect of one of the following situations: declared, skipped and detected faults (see details below). Note that the corresponding power is immediately decremented when a sector is marked as faulty. Power will be restored when its declared recovery is proven.

Declared faults: If an SP knows that is not able to include all the active sectors of the partition (*e.g.*, hard-disk failure, internet connection failure, short power outage) in the proof, he/she can announces the faults at most FaultCutoff (= 50) epochs before the challenges are received. Later fault declarations are not accepted. The sectors are marked as faulty and the corresponding consensus power is removed from the epoch of declaration until completed recovery (see below for recovery details). At the deadline:

- If no recovery message[8] was received in time: the provider pays FF (*fault fee*) per sector;
- If a recovery message is received in time, see the recovery faults paragraph.

[8]In Filecoin, messages are the content of the blocks (ie, “transactions” in Ethereum) and are executed by any node running the blockchain. So, when we write something like “when message x is sent, SP pays fee y”, we mean that the execution of message x triggers the update of the balance of SP by removing y tokens.

Skipped fault: If the SP is not able to announce the faults in time, he/she can notify the missing sectors at proof submission time. At the deadline: For each faulty sector, the provider pays FF[9], the sector is marked as faulty and the corresponding consensus power is removed from now until completed recovery.

Note that it is irrational to declare fault for sectors that can not be recovered before the deadline (skipped faults are less expensive in that case, that is $P_{fault} \leq P_{skip}$[10]).

Detected fault: The provider does not report faulty sectors and the fault gets noticed at the deadline time (ie, no proof message submitted) or later on (ie, the proof is proved not valid by a dispute message).

At the deadline, if no proof message submitted:

- The provider pays FF for every already-faulty (including recovering) sector in the partition; the active sectors are marked as faulty and the corresponding consensus power is removed from now until completed recovery. No fee (FF or other) is paid for sectors that weren't already marked faulty (note that this implies that missing a WindowPost deadline on 100% healthy partitions incurs no fee and therefore it is not rational to declare faults for a whole partition of active sectors).

If the proof message is submitted:

- If a partition has recovering sectors, then verify the partition proof (on-chain verification). If the proof is valid, maintain power for active sectors and restore it for the recovering ones and follow the rules expressed before in case of faulty sectors (skipped or declared) for the message execution.
- If a partition has no recovering sectors, optimistically accept the proof as valid[11]. Note that the SP can still declare "skipped sectors" which are marked as faulty.

  The proof may be disputed by anyone for 1800 epochs (2× finality) after the deadline window ends. If a dispute successfully refutes the partition proof, the provider pays one IPF (*invalid proof fee*) per active sector in the partition plus a *flat fee* of 20FIL; all active sectors are marked faulty from now until complete recovery, and the disputer (address that submitted the dispute) is rewarded a fixed DisputeReward.

Fault recovery: Regardless of how a fault first becomes known (declared, skipped, detected), the sector stays faulty and is excluded from future proofs until the provider sends a recovery message. This message must be received within FaultCutoff epochs before the challenges to be valid. If this is the case, the sector is marked as recovering and at the next deadline (ie, the first after the recovery message) if the sector is proved, then the power is restored and the recovery is complete (ie, sector is marked as active).

[9] Originally, at launch time the per sector fee for skipped faults was SP, sector penalization, it was changed to be the same as declared faults fee FF by FIP002 *"Free Faults on Newly Faulted Sectors of a Missed WindowPoSt"*.

[10] $P_{skip} - P_{invalid} = (1 + p - X)BR + p \cdot FF \geq p(BR + FF) \geq 0$.

[11] Optimistic acceptance was introduced by FIP0010 *"Off-Chain Window PoSt Verification"*.

Termination: If a sector is faulty for 14 consecutive deadlines, it is terminated at the 14th faulty deadline. Moreover, at any time a provider can terminate his/her sector sending a termination message. If a sector is terminated, at the next deadline the providers pays TF (*termination fee*) (remaining initial pledge is refunded). The sector is excluded from the next deadline and power is removed. Note that need $TF \geq FF$, otherwise an SP could terminate the sector to avoid paying FF for detected faults.

5.1.2 Analysis

We consider a single sector with fixed $\mathsf{SectorSize}$ and $\mathsf{QuallityAdjustedPower}$. The different strategies that an SP can follow in the 24h before a deadline closes are:

1. *honest*: storing the replica and including the sector in a valid proof message;
2. *regeneration*: running the regeneration attack (see Section 1.2) and including the sector in a valid proof message;
3. (a) *declare fault recover*: declaring a fault after $X \cdot 24$h since the last deadline, recovering it before the next deadline and including the sector in a valid proof message;
   (b) *declare fault*: declaring a fault and recovering in the following deadline;
4. *skipped fault*: declaring a fault at submission time (when submitting a valid proof message for the partition);
5. *no proof*: no proof message submitted (for the entire partition);
6. *invalid proof*: sending a commitment to an invalid proof (for the entire partition);

We use the following notation:

- SC (Storage Cost) is the cost of storing $\mathsf{SectorSize}$ GiB for 24h;
- RC (Regeneration Cost) is the expected cost of recomputing part of the replica to pass WindowPoSt assuming the SP stores $\leq (1-\epsilon)\mathsf{SectorSize} = 0.80 \cdot \mathsf{SectorSize}$). That is, this is the expected cost of the regeneration attack (see Section 1.2);
- BR (Block Reward) is expected block reward share per sector with $\mathsf{QuallityAdjustedPower}$ per day.

Profit Formulas: We compute the profit of each strategy defined above considering a period of 48h (24h before the deadline opens and 24h after the deadline closes). See Figure 5.

1. Profit for the honest strategy $P_{honest} = 2BR - 2SC$;
2. Profit for the regeneration strategy $P_{regen} = 2BR - 2RC$;
3. (a) Profit for declare fault recover $P_{faultR} = (X+1)BR - FF$;
   (b) Profit for declare fault $P_{fault} = X \cdot BR - FF$ (note $P_{faultR} \geq P_{fault}$ always);

   Comment: we do not consider the possible cost of storing the sector for the X-fraction of the day. Doing this means overestimating the profit for declaring faults, which is okay for a security goal (ie, set fees to have negative profit).

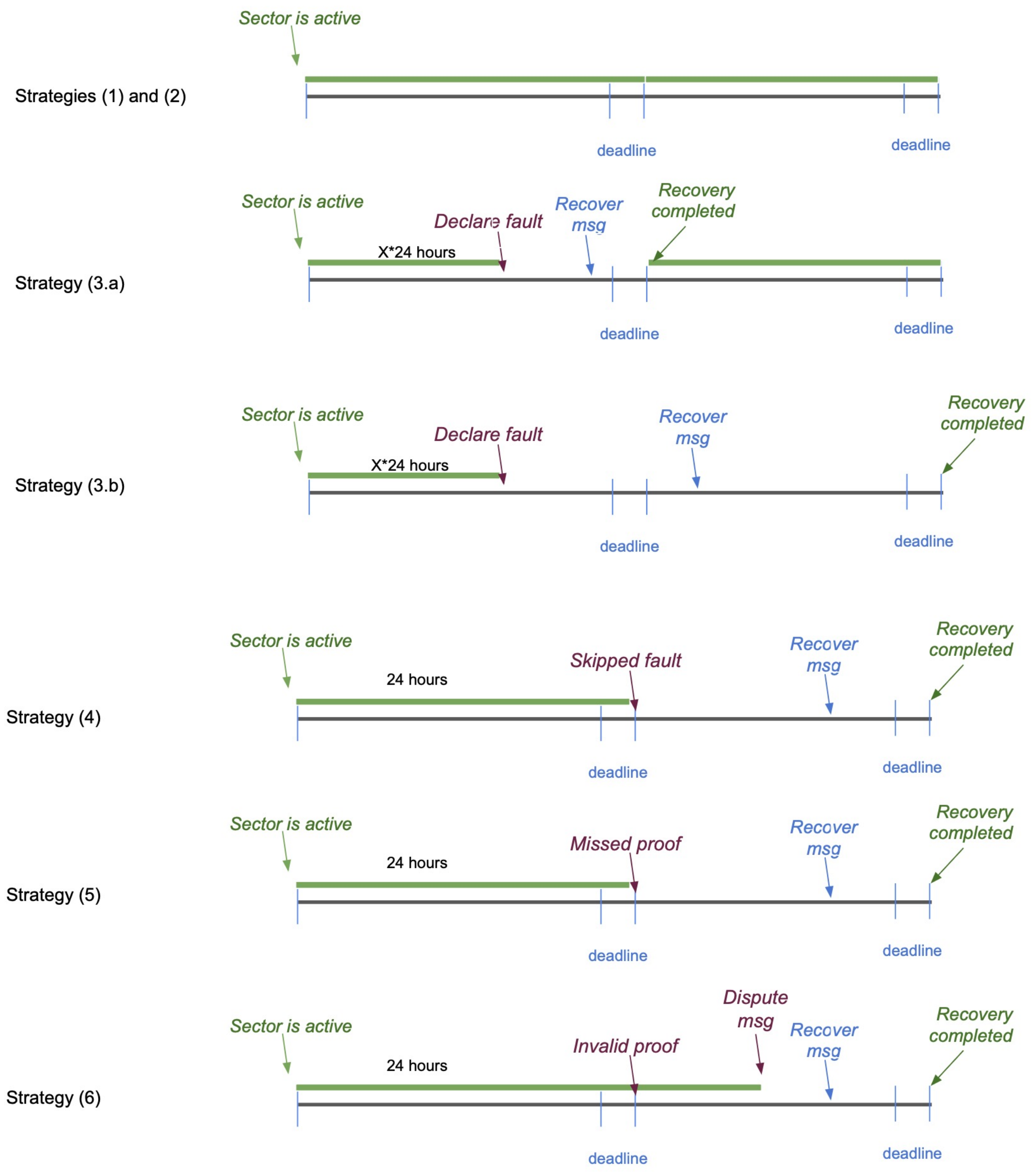

Figure 5: Visual comparison among the different strategies described in Section 5.1.2 (Window-PoSt). The green line corresponds to the sector being consider active and counting for consensus power.

4. Profit for skipped fault $P_{skip} = (1-p)(BR-FF)+2p\cdot BR$, where $p = 0.3487$ (ie, $p = (1-\epsilon/2)^c$) is the soundness error of the execution phase of SDR with $c = 10$ and $\epsilon = 0.2$, see Section 2.4);
5. Profit for the no proof strategy $P_{noProof} = BR$;
6. Profit for invalid proof strategy $P_{invalid} = p'[(1+Y)BR - IPF] + 2(1-p')BR$ (assuming that with probability p' the disputed message arrives $Y\cdot 24$ hours after the deadline closes and ignoring the flat fee for the sake of simplicity).

Proposition 1 (storage fault key results):

(a) If $FF > 2BR$, then declared faults have negative profit ($P_{faultR} < 0$ and $P_{fault} < 0$);

(b) If $FF > 2.07BR$, the skipped faults have negative expected profit ($P_{skip} < 0$);

(c) If $FF > BR$, detected faults for 2 or more consecutive deadlines (ie, submitting no proof) has negative profit;

(d) If $IPF > 3.65BR$, then as long as invalid proofs are successfully disputed with probability 1/2 (ie, one every two proof is disputed) submitting an invalid proof has negative (expected) profit.

Proof. (a) If $FF > 2BR$, then $FF > (X+1)BR$ for all $X \in (0,1)$. And clearly, this is a sufficient condition in order to have that P_{faultR} and P_{fault} are negative.

(b) Since $p = 0.3487$, $FF > 2.07BR$ implies that $FF > [(1+p)/(1-p)]BR$, which is equivalent to $(1-p)(BR-FF)+2pBR < 0$.

(c) Note that $P_{noProof}$ is never negative (for an active sector). However, consider the case of missing 2 consecutive proofs: $P_{no2proof} = BR - FF$. Therefore if $FF > BR$ then the profit for missing a proof for a faulty sector is negative.

(d) One proving period corresponds to 2880 epochs (each epoch is 30 seconds), since the maximum accepted arrival time for a dispute message is 1800 epochs, we have that $Y \leq 1800/2880 = 0.625$. Therefore we have $P_{invalid} \leq (2-0.375p')BR - p'\cdot IPF$. And $(2-0.375p')BR - p'\cdot IPF$ if and only if $IPF \geq \frac{(2-0.375p')}{p'}BR$ (assuming $p' \neq 0$ always), so the condition on IPF depends on the minimum p'. If we assume that at least one every two proof are disputed in time (ie, $p' \geq 1/2$), we have a sufficient $IPF \geq 3.65BR$ is a sufficient condition for $P_{invalid} \leq 0$.

□

Proposition 1 shows that by setting the storage fault fees to appropriate values, we can guarantee that any strategy that implies dropping the sector (ie, stopping proving possession of the replica every day) is not profitable. In other words, strategies from 3 to 6 have negative profit.

It remains to analyse the first two strategies and show that the honest strategy has an higher profit that the regeneration strategy. The next proposition gives the formula for comparing these two cases.

Proposition 2 (honest vs regeneration formula):

Defining $SM = RC/SC$ (**Security Margin**), we have that $P_{honest} - P_{regen} = 2(SM-1)SC$. Therefore, it holds that if $SM \geq 1$, then $P_{store} \geq P_{regen}$.

Now, we can conclude that in Filecoin, since storage fault fees respect the conditions of Proposition 1 (see Appendix A) and $SM \geq 1$ (see next section), for each sector added by an SP, persistently storing the replica associated to the sector is the strategy with highest profit.

5.1.3 Cost Heuristic Assumptions

In the following we show an example about how to argue that $SM \geq 1$ considering approximate, but realistic SC and RC values.

Storage Cost. After market research, we consider a HDD manufacture deal of \$18 per TB (2y lifetime). Considering a redundancy factor of 1.25, this gives:

$$SC = (\$18 \text{ / 1TB in GiB}) \cdot 32\text{GiB} / (2{\cdot}365) = \$0.000429$$

Regeneration Cost. From SDR security (Theorem 2.8), we know that regeneration requires to sequentially label at least $\beta^* n = 0.2n$ nodes (for a 32GiB sector, $n = 2^{30}$) with probability $\geq (1-p)$ and $p = (1-\epsilon/2)^k$ (see Section 2.4). With $k = 10$ challenges and $\epsilon = 0.2$, we have $p = 0.3487$. This give the expected cost of:

$$RC = (1-p) \cdot 0.2 \cdot 2^{30} \cdot (\text{cost of labeling a node}) = (1-p) \cdot 0.2 \cdot (\text{cost of labeling a replica/11})$$

To estimate the cost of labeling a node, we considered existing software that process 15 sectors in parallel in 3h (PC1 running time, see Section 4 and run on a HW consisting of two CPUs (for example AMD Epyc 7713p, \$7000) and fifteen 64GB-memory (\$3000). Assuming an HW lifetime of 2 years, the cost of labeling (PC1) per sector (32GiB) is: [(\$10000 / (2 · 365 · 24)) · 3] / 15 =\$0.114. Therefore $RC = (1 - 0.3487) \cdot 0.2 \cdot \$0.114/11 = \$0.00540$. And, in conclusion $SM = 3.15$.

5.2 Rationality of Storage based on WinningPoSt

WinningPoSt is the name of the proof created to show correctness of an execution step of SDR with 66 challenges for a single replica.

5.2.1 Protocol Description

Every time an SP is elected to produce a block, the SP needs to include in the block a valid SNARK proof for an execution step run on an active sector chosen at random among all the ones that the SP added to the network. If the SP fails to do so, the block is not valid and the SP loses the entire block reward for that epoch.

More in details:

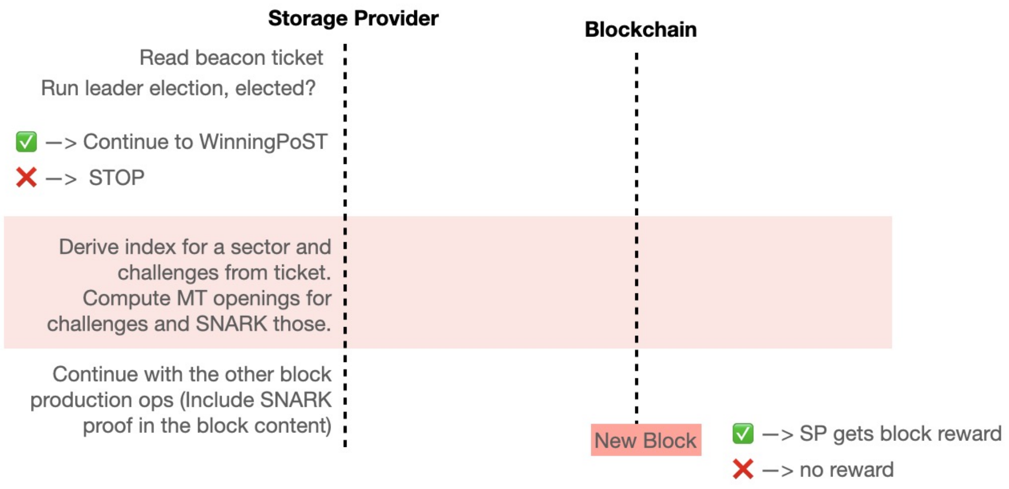
Storage Provider
Blockchain
Read beacon ticket
Run leader election, elected?
✅ —> Continue to WinningPoST
❌ —> STOP
Derive index for a sector and challenges from ticket. Compute MT openings for challenges and SNARK those.
Continue with the other block production ops (Include SNARK proof in the block content)
New Block
✅ —> SP gets block reward
❌ —> no reward

Figure 6: Visual recap of the WinningPoSt protocol in Filecoin.

- At each epoch (ie, every 30 seconds), the SP reads a new random beacon ticket and run the leader election protocol to check if is a leader (ie, allowed to propose a block) for the epoch;
- If the SP is the leader, the ticket is used to select an active sector among all the ones owned by the SP and to generate 66 challenges (ie, replica node indices) for the execution step of SDR (see Section 2.2);
- SP computes the corresponding vanilla proofs and runs the SNARK prover algorithm to compress those in a short proof that is added to the block content;
- The SNARK proof is verified by the others nodes as part of the rules for accepting a new block.

5.2.2 Analysis

We consider an SP that added r sectors and compare the following two strategies:

1. *honest*: storing persistently store each sector;
2. *delete*: deleting d replicas at epoch e_1; note that in this case at epoch e if the SP gets elected as leader and one of the deleted replicas is chosen for the WinningPoSt execution, then the regeneration attack is not possible (see Section 5.2.3) and the probability of be able to generate a valid proof is bounded by the SDR execution soundness error.

We use the following notation:

- sc = cost of storing 1 replica for 1 epoch;
- br = expected block reward share with r replicas for a fixed epoch.

Profit formulas: We compute and compare the profit of case 1 *(honest)* vs the profit of case 2 *(delete)* during a given epoch e after the deletion ($e > e_1$).

1. $P_{honest} = br - r \cdot sc$
2. $P_{delete} = \frac{(r-d)}{r}[br - (r-d)sc] + \frac{d}{r}[p(br - (r-d)sc) + (1-p)(0 - (r-d)sc]$, where p is the soundness error of the SDR execution phase with 66 challenges. That is $p = (1 - \epsilon/2)^c = (1 - 0.1)^6 6 = 0.00095 \leq 2^{-10}$.

Proposition 3:

If $p \approx 0$, then $P_{delete} \approx \left(1 - \frac{d}{r}\right) P_{honest}$.

Proof.

$$\begin{aligned}
P_{honest} - P_{delete} &= br - r \cdot sc - \frac{(r-d)}{r} br + \frac{(r-d)^2}{r} sc + \frac{d(r-d)}{r} sc - \frac{d \cdot p}{r} br \\
&= br - r \cdot sc - \frac{(r-d)}{r}[br - (r-d)sc - d \cdot sc] - \frac{d \cdot p}{r} br \\
&= br - r \cdot sc - \frac{(r-d)}{r}[br - r \cdot sc] - \frac{d \cdot p}{r} br \\
&= (br - r \cdot sc)\left(1 - \frac{r-d}{r}\right) - \frac{d \cdot p}{r} br \\
&= (br - r \cdot sc)\frac{d}{r} - \frac{d \cdot p}{r} br \\
&= P_{honest} \cdot \frac{d}{r} - \frac{d \cdot p}{r} br
\end{aligned}$$

And therefore $P_{delete} = \left(1 - \frac{d}{r}\right) P_{honest} + \frac{d \cdot p}{r} br.$ □

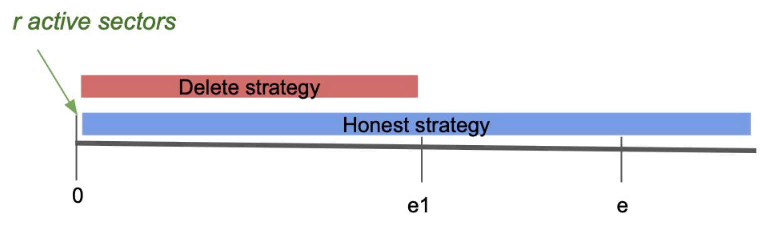

Figure 7: Visual comparison among the different strategies for Winning PoSt (Section 5.2.2.

5.2.3 Latency Assumption

Why regeneration is not possible for WinningPoSt? In short, the security of SDR and the latency assumption about SHA256 imply that regeneration takes more than 30 seconds which is the window that an SP has to produce and propagate a block. More in details:

- Latency Assumption: We consider a 7.5GB/s theoretical sequential ASIC rate, which gives 4.25 ns to hash 32 bytes.
- From SDR security we know that (Theorem 2.8 and Section 2.4) regeneration requires to sequentially label at least $\beta^* n = 0.2n$ nodes (for a 32GiB sector, $n = 2^{30}$). Note that each node's label is hashing at least 37$\cdot$ 32 bytes.
- So, putting the former two points together we have that regeneration takes at least: $0.2 \cdot 2^{30} \cdot 37 \cdot (4.5 \cdot 10^{-9})$ seconds $= 35.7$ seconds, which is larger that the time allowed to build a block in Filecoin (ie, 30 seconds).

References

[DFKP15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 585–605, 2015.

[Fis18] Ben Fisch. Poreps: Proofs of space on useful data. *IACR Cryptol. ePrint Arch.*, 2018:678, 2018.

[Fis19] Ben Fisch. Tight proofs of space and replication. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, pages 324–348, 2019.

[RD16] Ling Ren and Srinivas Devadas. Proof of space from stacked expanders. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part I*, pages 262–285, 2016.

A Filecoin Concrete Parameters

Stacked-DRGs graph:

- Number of layers, ℓ: 11
- Number of nodes per layer: $n = 2^{30}$ or $n = 2^{31}$
- DRG degree: 6
- DRG chain length, β^*: 0.2 (see definition in Section 2.1)
- Expander graph degree: 8

SDR proof:

- $\epsilon = 0.2$ (spacegap);
- $\delta = 0.039$;
- SHA256 is used for labeling a node, each label has 37 inputs of 32 bytes each (we repeat the 14 parents);
- For the PoRep protocol we have $c = 176$ challenges per layer[12]. This implies that for the PoRep protocol (initialization phase) we have a soundness error of 2^{-10}.
- The PreCommit Deposit value depends on the sector power and is set to $PCD = 20 \cdot BR$ (recall that BR is the expected block reward share for the sector per day):
- $\mathsf{PreCommitChallengeDelay} = 150$ epochs;
- $k = 10$ challenges in WindowPoSt execution, which implies a soundness error equal to 0.3487.

[12]In practice, due to SNARK software constraints, we have that $c = 180$ rather than 176, so we actually get better $\delta = 0.0378$. However, the following analysis in this doc is written for the original value $c = 176$.

- $k = 66$ challenges in WinningPoSt execution. In this case, we have a soundness error of 2^{-10}.

Fault Fees

Storage fee values depend on the sector QuallityAdjustedPower and are set to:

- $FF = 3.51 \cdot BR$ (fault fee);
- $IPF = 5.51 \cdot BR$ (invalid proof fee);
- $TF =$ how much BR the sector earned so far, capped at 90 days (termination fee).