Arweave: The Permanent Information Storage Protocol

Sam Williams
sam@arweave.org

Abhav Kedia
abhavkedia@gmail.com

Lev Berman
lev@arweave.org

Sebastian Campos-Groth
sebastian@arweave.org

DRAFT 17
December 26, 2023

Abstract

In this paper we describe Arweave: a protocol for the disintermediated transmission of information across space and time. Arweave aims to serve as an institution offering an arbitrarily scalable permanent ledger of data and speech in cyberspace, without centralized authority and control. The network achieves this by utilizing a novel blockchain consensus mechanism powered by succinct proofs of replicated storage. This proof system has low verification overhead: minimizing compute and bandwidth requirements, while maximizing decentralization. This construction is paired with a storage endowment that incentivizes data replication, allowing the network to operate in a fully autonomous, transparent and predictable fashion. Finally, we describe a new system of fork-based evolution that allows for protocol flexibility while preserving user rights over time.

1 Introduction

The rate of information flow in society has increased precipitously since the invention of the printing press, culminating with the internet in the late 20^{th} century. As human society has progressed along this information distribution efficiency curve, the level to which individuals are informed about the world around them has increased. This increase has been productive for societies as it has fostered greater transparency and awareness within communities. In its current form, the internet is still practically subject to subversion by centralized points of control, resulting in information distribution inefficiencies: it can be "forgetful", losing a certain proportion of its useful content every year, and these centralized points of control and aggregation have the power to arbitrarily censor and manipulate the flow of information. Arweave is built to address these problems by allowing for the disintermediated transfer of information across extremely long distances of time and space. While further discourse on the uses and potential implications of the Arweave network has taken place in numerous other forums, this paper aims to provide a precise and succinct description of the system's technical functions and the techniques used to accomplish them.

Arweave is a permanent information storage system. The word "permanent" has a number of definitions: the Oxford Dictionary of English defines it as "lasting or intended to last or remain unchanged indefinitely", while the Merriam-Webster Dictionary defines the word as "continuing or enduring without fundamental or marked change". In this paper, we use a definition of permanence that encompasses both of these ideas: the Arweave network is engineered to store data for the *maximum possible term, without change.*

In order to achieve its goals, the Arweave protocol is constructed of three core components:

- **Cryptographic Proofs of Storage:** A succinct cryptographic proof system for verifying even replication and accessibility of data.
- **Storage Endowment:** A predictable, self-executing endowment that pays for storage, utilizing the deflationary effect of technology improvements over time.
- **Incentivized Evolution:** A mechanism for allowing protocol adaptability by generating and rewarding non-coercive upgrades to the network.

The Arweave protocol is precisely codified, immutable, and entirely transparent. As a result, the network is able to offer its users predictable rights and guarantees that would not be possible in a comparable system operated by centralized institutions. This paper precisely documents each component of the fully implemented protocol as of the current version at the time of writing (v2.7.0). Users are encouraged to read this paper and audit the protocol themselves in order to gain a first-hand understanding of the services and guarantees that the protocol offers. In order to facilitate verification of the protocol, we provide references to the codebase at various points in this paper.

2 Protocol Design

A core component of Arweave's construction is a decentralized consensus system for appending to and verifying data in the network. Decentralized consensus is a subfield of distributed computing that encompasses significant research on protocols that enable participants in a network to reach agreement on a state, even in the presence of adversarial nodes [12, 13, 15]. The field gained prominence with Bitcoin's innovative Nakamoto consensus which allowed, for the first time, agreement in an adversarial *and* permissionless environment [24]. As a consequence of this innovation, Bitcoin was able to create the first digital currency that did

not depend on centralized human actors for administering monetary policy. Arweave draws inspiration from Bitcoin's proof-of-work structure for achieving consensus and adapts its implementation to incentivize the permanent storage of information within its network.

Arweave is a global network of computers, referred to as "nodes", which collectively store multiple copies of all data uploaded to the system. Users who wish to store information on Arweave provide a one-time upfront contribution to the network's storage endowment and upload the corresponding data by transmitting it to nodes within the network. Nodes periodically reach consensus on new data to be added to the network's globally distributed repository each time one of the nodes successfully *mines* (confirms) a *block*. A *block* is a list of transactions, each containing either new data to be added to the network, or transfers of its currency AR, or both. *Mining* is a process run by every participating node that serves the dual purpose of accepting new data into the network and validating storage of previously uploaded data. Nodes "pull" data that they would like to replicate for mining from other peers in the network after a block containing the transaction has been confirmed.

2.1 Design principles

The two primary principles that underpin the design of the protocol are as follows:

- **Minimalism:** The protocol aims to be straightforward and minimally opinionated in its design, in order to foster the widest possible social consensus. Arweave only uses well-tested cryptographic primitives in composing its data structures and algorithms.
- **Optimization through incentives:** Rather than explicitly prescribing desired behaviour, the protocol incentivizes participants to achieve desirable outcomes. The specific mechanisms for achieving these outcomes arise organically and will evolve over time.

By nature of its conservative design, the Arweave protocol is focused on concisely solving one problem – permanent and scalable data storage. This leaves the application layer on top of the protocol extensible and composable, enabling wide and varied uses of the network [35]. This has led to the emergence of many different decentralized smart contracting platforms, databases and applications, built on top of Arweave [31, 19, 25, 38]. Further, Arweave's highly efficient proof system imposes very low hardware and bandwidth requirements – comparable with Bitcoin, regardless of dataset size – in order to maximize participation and decentralization in the network.

In order to illustrate the effectiveness of incentives in protocol design, we can examine its effects in the Bitcoin network. Bitcoin rewards miners for discovering a *nonce* that, taken together with a candidate *block*, produces a hash that is lower than a certain value. The uniformly random distribution of hashes obtained from a $\{nonce, block\}$ pair incentivizes miners to find ways to compute the largest number of hashes at the lowest possible cost. This has led to the development of application-specific mining hardware that has

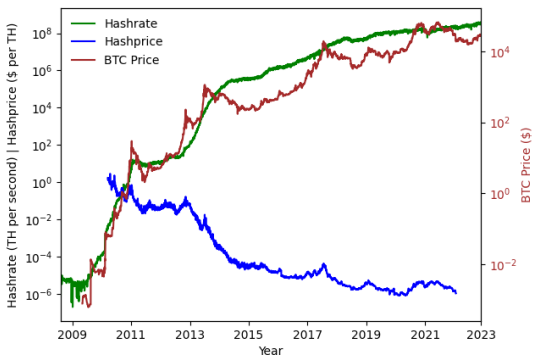

Figure 1: Incentives for efficient mining in Bitcoin have contributed to significant improvements in speed and costs of computing hashes (Y-Axis is log scale) [9].

increased the number of hashes that can be computed per second by a factor of 10^{13} since 2011. Notably, this increase has outperformed Moore's law [28] by a factor of nearly 10^{10} in the same period, up to the time of writing. The same incentives have also led to a decrease in the cost per hash in the Bitcoin network by a factor of 1,000,000 in the same period [9]. In Arweave, we incentivize participants to solve a different set of problems, modifying Bitcoin's mechanism to incentivize optimization of useful proofs of storage and routing of data.

3 Cryptographic Proofs of Storage

The mining mechanism of Arweave composes many distinct algorithms and data structures into a non-interactive and succinct proof of replicated storage of data in the network. In this section, we will initially examine these primitives in the abstract and then describe their use in the process of Arweave mining.

3.1 Block Index

The top-level data structure in Arweave is called the Block Index [2]. It is a merkelized list of 3-tuples, each containing a *block hash*, a *weave size*, and a *transaction root*. This list is represented as a hashed merkle tree [23] with the top-level merkle root representing the most recent state of the network. This root is composed of the hash of two elements – the most recent tuple (representing the latest block) and the previous block index's merkle root. While creating a block, miners embed the previous merkle root of the block index within the new block.

By constructing and embedding a merkle root of the block index in each block, a node in the Arweave network that has performed an SPV proof [24] of the latest blocks is capable of fully validating any prior block. This stands in constrast to traditional blockchains in which individual validation of transactions in older blocks requires full verification of the chain back to that block. After performing an SPV proof on the tip of the network, the Block Index verification mechanism allows the user to validate older blocks in the weave

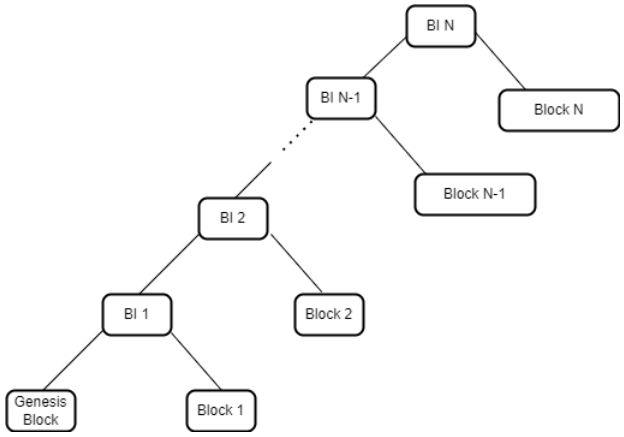

Figure 2: The block index represents a merkle tree of blocks in the Arweave network.

without needing to download or validate intermediate block headers.

3.2 Merkelization of data

When information is to be added to Arweave, the user splits the data into regularized 256 KiB chunks. After the chunks are prepared, a merkle tree is constructed and its root (called the *data root*) is committed into a transaction. This transaction is signed and dispatched to the network by the uploader. Each transaction so constructed is represented within the *transaction root* [3] of the block it was accepted in. A block's transaction root is therefore a merkle root of data roots of transactions.

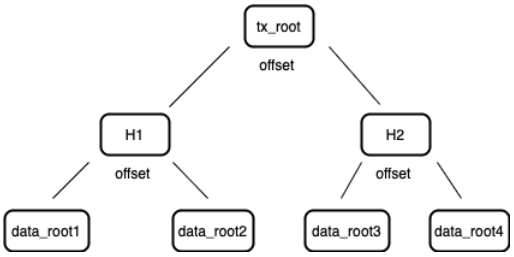

Figure 3: The transaction root is a merkle root of data roots present in each transaction inside a block.

Miners in the Arweave network combine such transactions into a block, compute the transaction root using the data roots inside the transactions, and include this transaction root in the new block. This transaction root eventually ends up in the top-level block index as described above. The output of this entire process is the creation of an expanding merkle tree with all of the data in the protocol sequentially ordered into chunks.

3.3 Succinct Proofs of Access

In the Arweave network, nodes in a merkle tree are labelled with the offset of the data that can be found to the "left" or "right" of the node [8]. This labeling of nodes enables the creation of succinct merkle proofs that allow verification of

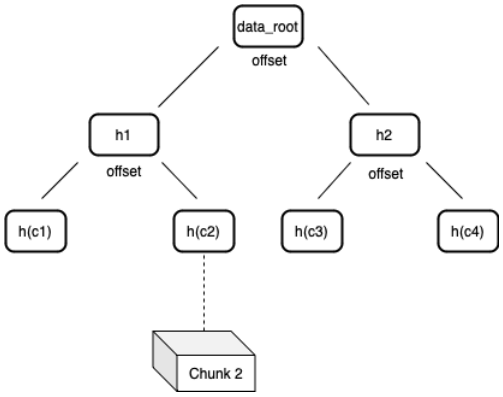

Figure 4: The data root is a merkle root of all hashed chunks in a single transaction.

the existence of chunks at specific locations in the dataset. Such a merkle tree is called "unbalanced" as the number of leaves on either side of a node may not be equal. This structure can be used to construct a succinct non-interactive proving game about the availability of data in a miner's hard drives.

The Succint Proof of Access (SPoA) game begins when Bob wants to verify that Alice is storing data at a specific location of a merkelized dataset. In order to play the game, Bob and Alice should have the same merkle root for the dataset. The game works in three phases: challenge, proof construction, and verification.

Challenge: To start the game, Bob sends Alice a *challenge offset* – the index of the byte that Alice should prove access for.

Proof Construction: Upon receiving this challenge, Alice begins to construct a proof. She searches her merkle tree to find the chunk corresponding to the offset.

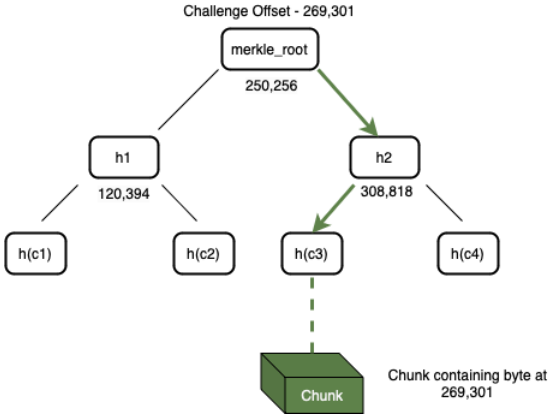

Figure 5: Alice can search for the requested chunk by traversing the merkle tree labelled with offsets.

In the first stage of proof construction, Alice retrieves the entire chunk from storage. She then constructs a merkle proof using the chunk and the merkle tree. She hashes the entire chunk to obtain a 32-byte identifier, and locates the parent of this leaf node in the tree. The parent is a node containing its offset and the two hashes of its child nodes, each of which is 32 bytes in length. This parent is added

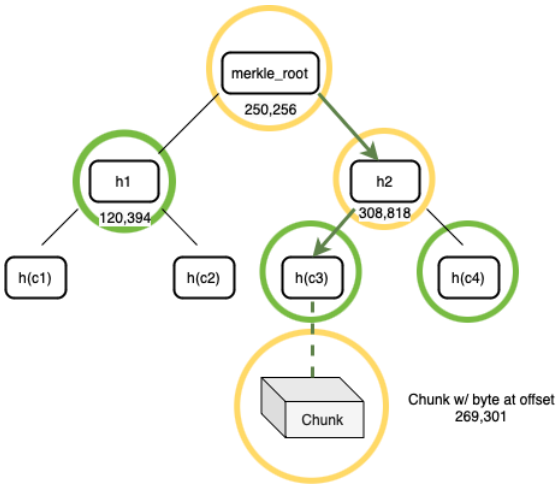

Figure 6: The proof for a challenge offset includes the chunk stored at the offset and its merkle path.

to the proof. Each parent is then recursively added to the proof until Alice reaches the merkle root. This series of nodes forms the merkle proof that, along with the chunk, Alice sends to Bob [8].

Verification: Bob receives the merkle proof and the chunk from Alice. He verifies the nodes in the merkle proof by recursively validating the hashes in the tree from the root to the two leaf nodes. He then hashes the chunk included in Alice's proof and verifies that this chunk matches the leaf with the correct offset. If the hashes match, the proof is successful. If at any stage, however, the hashes that Bob is verifying are incongruent, Bob stops the computation and rejects Alice's proof without needing to compute any remaining hashes. The output of this verification function is a boolean value, indicating the success or failure of the proof.

3.3.1 SPoA Complexity

The complexity of constructing, transmitting and validating SPoA proofs is $O(\lceil logn \rceil)$, where n represents the size, in bytes, of the dataset. In practice, this means that the proof of the location and correctness of a single byte in a dataset of size 2^{256} bytes can be transmitted in just $256 * 96B$ (maximum path size) + 256KiB (maximum chunk size) $\approx$ 280 KiB.

3.4 Replica Uniqueness

Mining power in the Arweave network is engineered to be proportional to the number of replicas of the weave accessible by a miner (or a group of cooperating miners). In the absence of a replica uniqueness scheme, copies of a piece of data are identical to each other in their content. In order to incentivize and measure multiple unique replicas of a single piece of data, Arweave deploys a system of packing.

3.4.1 Packing

Packing ensures that miners cannot forge SPoA proofs claiming to represent multiple unique replicas from the same stored chunk. Arweave's packing scheme uses RandomX [34] to convert regular data chunks into "packed" chunks in a process that incurs a compute cost for the miner.

Packing in Arweave works as follows [4, 5, 6]:

- The *chunk offset*, *transaction root* and *mining address* are combined in a SHA-256 hash to generate a packing key.
- The packing key is used as entropy for an algorithm that generates a 2 MiB scratchpad over several successive rounds of RandomX execution. This component of the packing mechanism incurs significant cost for non-compliant miners.
- The first 256 KiB of the resulting entropy is used to symmetrically encrypt the chunk using a Feistel cypher [22]. This computation results in a packed chunk.

If the overall cost of packing chunks is greater than the cost of storing packed chunks in the time period between consecutive reads, the miner is incentivized to store unique chunks as opposed to packing chunks "on-demand" in order to maximize mining efficiency. In order for the correct behaviour to be incentivized, the following invariant must hold:

$$c_s(\text{chunk}) * t < c_p(chunk)$$

where:

c_s = Cost of storing a chunk for unit time
c_p = Cost of packing a chunk
t = Mean time between chunk reads

Total packing costs incorporate the price of electricity for computation, price of specialized hardware and its mean lifetime. The cost of storage can similarly be computed from the cost of hard disks, mean-time to hard disk failure, electricity costs of operating the disk as well as other costs associated with operating storage hardware. We can compute the ratio of the costs of a high-efficiency on-demand miner versus an honest miner that stores packed chunks to estimate the safety margin for incentivizing packed chunks. In practice, Arweave uses a *safety ratio* – the ratio of the cost of creation of chunks on-demand to the cost of storage of the chunk for mean time between access – of approximately 19 at the time of writing, benchmarked on commercially available hardware.

3.4.2 RandomX

In order to ensure packing remains resistant to hardware acceleration, Arweave runs a carefully chosen hashing algorithm called RandomX over the packing key. RandomX is optimized for general-purpose CPUs as it uses randomized code execution during the hashing process, and several memory-hard techniques that minimize the efficiency of specialized hardware [34]. RandomX's use of randomly generated programs that closely match the hardware of modern CPUs implies that the only way to accelerate RandomX hashing is to create faster general-purpose CPUs. While the

Arweave protocol theoretically could incentivize the creation of faster CPUs, we note that the effect of this incentive relative to the existing incentives for improving CPU speeds would be negligible.

3.5 Verifiable Delay Function

A *verifiable delay function* (VDF) allows computational verification of the passage of time between events. A VDF requires a specified number of sequential steps to evaluate, yet produces a unique output that can be efficiently and publicly verified [10, 41]. There are several techniques for constructing VDFs. Arweave uses a chained hashing technique, where the VDF is defined recursively as follows:

$$V(n, seed) = hash(V(n-1, seed))$$

for $n > 1$, and:

$$V(1, seed) = hash(seed)$$

The hash function used in Arweave's VDF is SHA2-256 [7]. Using this construction, if the fastest available processor can produce only k sequential SHA2-256 hashes every second, then the correct computation of $V(k, seed)$ using just the seed requires at least 1 second. This is because the hash of a hash is impossible to compute without computing the first hash. A correctly generated $V(k, seed)$, called a *checkpoint*, implies the passage of at least 1 second between the prover receiving the seed and generating this checkpoint.

VDF construction takes $O(n)$ time for n steps. However, if intermediate checkpoints are transferred to the verifier, the verification of correctly generated VDF outputs can be completed in $O(\frac{n}{p})$ time using p parallel threads. We use the chained-hash VDF rather than other methods of VDF construction due to the simplicity of its assumptions and its singular reliance on the robustness of the hash function.

3.6 Succinct Proofs of Replications

The SPoA game described earlier can be used by any participant to prove that they are storing some data at a particular location in the dataset. This system can be used to create a second game, called **S**uccinct **P**roofs of **Re**plication**s** (SPoRes), that will allow a prover to convince a verifier that the prover stores a certain number of data replicas with extremely minimal data transfer and verification overhead, regardless of dataset size.

3.6.1 SPoRes Game

The SPoRes game works as follows. Alice claims that she stores n copies of a merkelized dataset. Bob wants to verify Alice's claim. In order to do this, he provides Alice with a difficulty parameter d, and a random seed. Alice uses the seed to generate a VDF chain that emits a *checkpoint* every second that can be used to unlock a maximum of k SPoA challenges within the dataset. Whenever she has packed chunks corresponding to any of these challenges, Alice can construct corresponding SPoA proofs. Each of these proofs is then hashed and compared to Bob's difficulty parameter d. If the proof hash is greater than d, then Alice has found a *valid solution* and sends the corresponding proof to Bob. Bob records the time it takes between delivering the random seed and receiving a valid proof from Alice.

Based on the difficulty d, the probability of finding a valid solution on a single trial is given by:

$$p = \frac{2^{256} - d}{2^{256}}$$

If Alice has N replicas and can perform k trials per replica every second, her probability of finding a solution in any given one-second time period is:

$$p_2 = 1 - (1-p)^{kN}$$

The time taken for Alice to deliver the proof can be modeled with a geometric random variable $X \sim geom(p_2)$, with probability p_2 of success. This random variable depends on d, k and N. In order to verify Alice's claim, Bob sends over a difficulty d such that she can deliver proofs to him once every 120 seconds, given that she has N replicas. This is equivalent to sending a difficulty parameter such that:

$$p_2 = \frac{1}{120}$$

or equivalently:

$$p_2 = 1 - (\frac{d}{2^{256}})^{kN}$$

If Alice can deliver the proof in the required time frame, she is likely to have the right number of replicas. A single proof, however, will not be sufficient for Bob to be convinced that Alice isn't lying – after all, she might get lucky and find a proof quickly even with fewer stored replicas. But if Alice can consistently deliver proofs on average every 120 seconds over a very long time period, Bob can be reasonably assured that Alice is storing the desired amount of data.

We shall attempt to quantify the certainty that Bob has about Alice's storage. Let us say that Alice claims to store 20 replicas, and has consistently delivered proofs at an average of 120 seconds over a 2-week time period (for a total of 10,080 proofs). Bob is interested in knowing the probability that Alice managed these proofs despite storing only 19 (or fewer) replicas. This represents a different one-second proof probability:

$$p_2^* = 1 - (1-p)^{19k}$$

Which we can simplify to:

$$p_2^* = 1 - (\frac{d}{2^{256}})^{19k} \quad (1)$$

This probability can be calculated using the value of d that Bob generates for a truth-telling Alice. Her one-second proof probability if she stores 19 or fewer replicas is given by the series of inferences given below. Here, p_2 represents the probability of producing a proof in one-second when Alice is truthfully storing 20 replicas, and p_2^* represents the one-second proof probability if she is lying (storing ≤ 19 replicas):

$$p_2 = 1 - (\frac{d}{2^{256}})^{20k} = \frac{1}{120}$$

$$\Rightarrow (\frac{d}{2^{256}})^{20k} = \frac{119}{120}$$

$$\Rightarrow (\frac{d}{2^{256}})^{19k} = ((\frac{d}{2^{256}})^{20k})^{\frac{19}{20}} = (\frac{119}{120})^{\frac{19}{20}} = 0.99208167919$$

$$\Rightarrow p_2^* = 1 - (\frac{d}{2^{256}})^{19k} = 0.00791832081$$

Therefore $p_2^* = 0.00791832081$, corresponding to an expected proof production time of 126.2894 seconds. Let X^* be the random variable obtained from p_2^*, i.e. $X^* \sim geom(p_2^*)$. This represents Alice's distribution if she is lying – storing only 19 replicas. The expected value (mean) of X^* is:

$$\mathbb{E}[X^*] = \frac{1}{p_2^*} = 126.289402008$$

We can use the Central Limit Theorem [17] to estimate the probability that the sample mean is below 120, i.e., different from the mean $\mathbb{E}[X^*]$ obtained above. This is represented by the probability:

$$\mathbb{P}(\frac{\sum X_i^*}{10080} \leq 120) = \mathbb{P}(\frac{\sum X_i^*}{10080} \leq 126.2894 - 6.2894) \quad (2)$$

For a large number of samples, the left hand side in this inequality tends to a normal distribution with mean $\mathbb{E}[X^*]$ and variance $\frac{\sigma_{X^*}^2}{\sqrt{n}}$, where σ_{X^*} is the variance of X_d and n (=10,080 here) is the sample size [17]. Therefore (2) yields:

$$\mathbb{P}(\mathcal{N}(\mathbb{E}[X^*], \frac{\sigma_{X^*}}{\sqrt{10080}}) \leq \mathbb{E}[X^*] - 6.2894)$$

In the equation above, $\mathcal{N}$ represents a normal distribution. We can convert this distribution to its standard normal form to obtain the equivalent probability:

$$\mathbb{P}(\mathcal{N}(0,1) \leq -6.2894 * \frac{\sqrt{10080}}{\sigma_{X^*}})$$

Finally, using standard normal distribution tables, we can obtain the probability that Alice has lied over the 2-week time period:

$$\mathbb{P}(\mathcal{N}(0,1) \leq -5.019943) = 2.584 * 10^{-7}$$

Therefore, in this example, Bob can ascertain with ≈99.99997416% certainty that during the 2-week period Alice has stored over 19 replicas of the dataset. We note that this certainty would be even higher if Alice was storing fewer than 19 replicas, owing to an expected proof-production time longer than 126.3 seconds. We also note that this entire proof system requires the transfer of just one proof every 120 seconds. This is an average data transfer rate between Alice and Bob of 2.389 KiB per second (280 KiB/120 seconds), comparable with the synchronization overhead of the Bitcoin [24] network.

Finally, we note that these probabilities do not change with arbitrary increases in the size of the dataset, while increasing the sampling period continues to improve the accuracy of Bob's certainty of Alice's replica count at a superlinear rate (Fig 7).

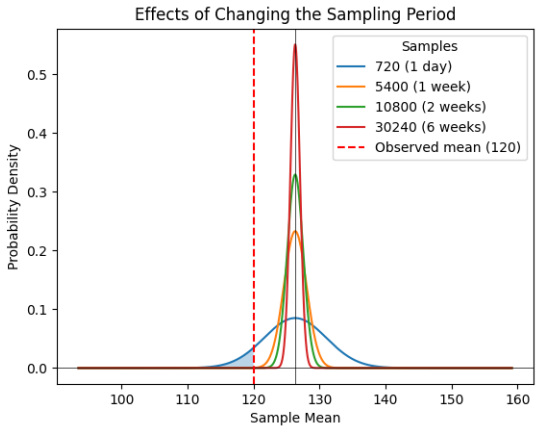

Figure 7: Certainty increases superlinearly to sample duration in the SPoRes game. After two weeks of sampling, the probability that less than 19 replicas are maintained is negligible ($P < 0.0000002584$).

Formally, a SPoRes game is defined by its parameters:

$$SPoRes(r, k, d)$$

where:

r = The merkle root of the dataset to be used for the game.

k = The maximum number of challenges unlocked per second per replica.

d = The difficulty parameter that determines the probability of success on each try.

4 Protocol Construction

In Arweave mining, a modification of the SPoRes game outlined in the previous section is deployed. During mining, the protocol acts as Bob and all miners in the network together play the role of Alice. Each proof of a SPoRes game is used to create the next block in Arweave. Specifically, Arweave block construction uses the following parameters:

$$SPoRes(BI, 800 * n_p, d)$$

where:

BI = The block index of the Arweave network.

n_p = The number of 3.6 TiB partitions of the weave stored by miners.

d = The difficulty of the network.

These parameters allow a maximum of 800 hashes per partition per checkpoint. A successful proof is one that is greater than the difficulty, which is adjusted over time to ensure that blocks are mined every 120 seconds on average. If the time difference between blocks i and $(i + 10)$ is t, the difficulty adjustment from the old difficulty d_i to the new difficulty d_{i+10} is computed as follows:

$$d_{i+10} = 2^{256} - (2^{256} - d_i)r$$

where:

$$r = \frac{t}{120 * 10}$$

The newly computed difficulty implies a probability of block-mining success on each generated SPoA proof given by:

$$p_{i+10} = \frac{(2^{256} - d_i)r}{2^{256}} = p_i r$$

The VDF difficulty is similarly recomputed to maintain single-second checkpoint periods over time.

4.1 Incentives for Replica Completeness

Arweave's block construction using SPoRes includes an assumption of rational agent behavior under incentives for maintaining full replicas, either by a lone miner or by cooperation between miners. This section explores how these incentives are provided, and how their effectiveness can be validated in the Arweave network. In the abstract SPoRes game as previously presented, the net effect of storing two copies of the same half of the dataset would yield the same number of SPoA hashes as one full replica spanning the entire dataset. The deployed version of the game in Arweave includes a modification that incentivizes miners to store and maintain all parts of the dataset – either cooperatively or independently.

The protocol provides incentives for accessing full replicas by splitting the number of SPoA challenges unlocked per second into two halves – one half requires just one partition of the dataset, and the other spans all partitions. To understand this, we will examine how the VDF construction described in the previous section is used to unlock SPoA challenges [27].

Algorithm 1 Unlocking SPoA Challenges

inputs: $seed, M$
$k \leftarrow 0$
while true **do**
  $check \leftarrow V(k + M, seed)$
  for $p \in partitions$ **do**
    $H_0 \leftarrow RandomX(check, addr, index(p), seed)$
    $C_1 \leftarrow H_0 \bmod size(p)$
    $C_2 \leftarrow H_0 \bmod (\sum_{q \in partitions} size(q))$
    $i = 0$
    while $i < 400$ **do**
      $chunk_1 \leftarrow chunkAtOffset(C_1)$
      $H_1 \leftarrow spoa\_first(chunk_1, C_1)$
      $chunk_2 \leftarrow chunkAtOffset(C_2)$
      $H_2 \leftarrow spoa\_second(chunk_2, C_2, H_1)$
      $C_1 \leftarrow C_1 + 262144$
      $C_2 \leftarrow C_2 + 262144$
      $i \leftarrow i + 1$
    end while
  end for
end while

Approximately once per second, the VDF chain outputs a checkpoint [7]. For every partition of the weave that the miner stores, this checkpoint is mixed with the mining address, the partition index, and the original VDF seed to obtain an intermediate RandomX hash. This 256-bit hash is treated as a number and divided by the size of the partition, yielding a remainder that is used as a recall offset. This offset unlocks 400 256-KiB challenges over a contiguous 100 MiB recall range starting from this offset. The mechanism that incentivizes full replicas is that each such recall range within a partition also unlocks another, *global*, set of 400 challenges in a second recall range with the constraint that solutions in the second range require a solution at the corresponding location in the first range.

4.1.1 Performance Per Packed Partition

The performance per packed partition represents the number of SPoA challenges that each partition yields for every VDF checkpoint. This number is greater when the miner stores unique partitions than when the miner stores multiple copies of the same data.

If the miner owns only unique data, each packed partition will yield all first-range challenges along with the few second-range challenges that fall within this partition. With n unique partitions stored out of a total m in the weave, this yields a *performance per packed partition* of:

$$perf_{unique}(n, m) = \frac{1}{2}(1 + \frac{n}{m})$$

When a miner stores partitions that are copies of the same data, each packed partition still yields all first-range challenges. Only on $\frac{1}{m}$ occasions, however, the second recall-range will be inside the same partition. This leads to a highly significant performance penalty, yielding a rate of just:

$$perf_{copied}(n, m) = \frac{1}{2}(1 + \frac{1}{m})$$

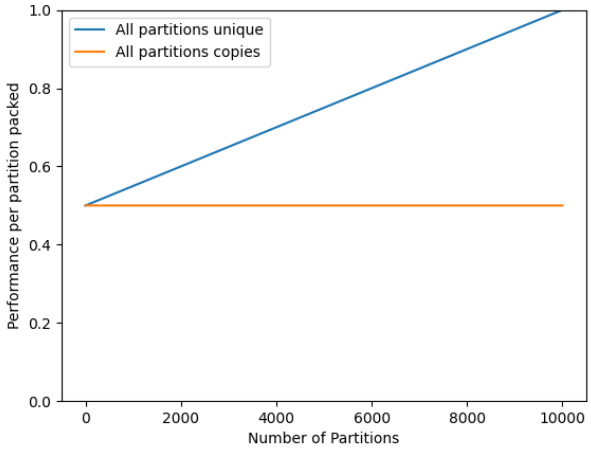

Figure 8: The performance for a given partition increases as a miner (or group of collaborating miners) completes their dataset.

The blue line in Figure 8 displaying $\{perf_{unique}(n, m)\}$ shows the efficiency per partition when storing n partitions

out of a maximum m. It demonstrates that the mining efficiency per partition is only 50% when the miner stores very few partitions from a replica. The mining efficiency is maximized when storing and maintaining all parts of the dataset, i.e., when $n = m$.

4.1.2 Total Hashrate

The total hashrate (shown in Figure 9) is given by the following equations, obtained by multiplying the *per partition* values by n:

$$tperf_{unique} = \frac{1}{2}n(1 + \frac{n}{m})$$

$$tperf_{copied} = \frac{1}{2}n(1 + \frac{1}{m})$$

The equations above show that as the size of the weave grows, the penalty function (for not storing unique data) increases quadratically with the number of stored partitions.

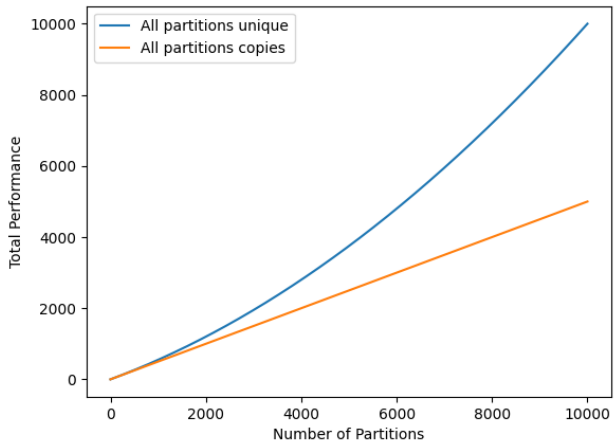

Figure 9: The total mining hashrate for unique and copied datasets.

4.1.3 Marginal Partition Efficiency

Given this structure, let us consider the decision problem a miner faces if they were to add a new partition and are deciding between replicating a partition that they already own, or fetching new data from other miners. When they already store n unique partitions out of a maximum m, their mining hash rate is proportional to:

$$tperf_{unique} \propto (n + \frac{n^2}{m})$$

The additional benefit of adding one new partition is:

$$(n+1) + \frac{(n+1)^2}{m} - n - \frac{n^2}{m} = 1 + \frac{2n+1}{m}$$

The (smaller) benefit of replicating an already packed partition is:

$$(n+1) + \frac{n^2}{m} + \frac{1}{m} - n - \frac{n^2}{m} = 1 + \frac{1}{m}$$

Dividing the first quantity by the second, we get the miner's marginal efficiency from acquiring a new partition relative to the additional efficiency of duplicating an existing partition. We call this the *relative marginal partition efficiency*:

$$rmpe(n, m) = \frac{2n + m + 1}{m + 1} = 1 + \frac{2n}{m + 1}$$

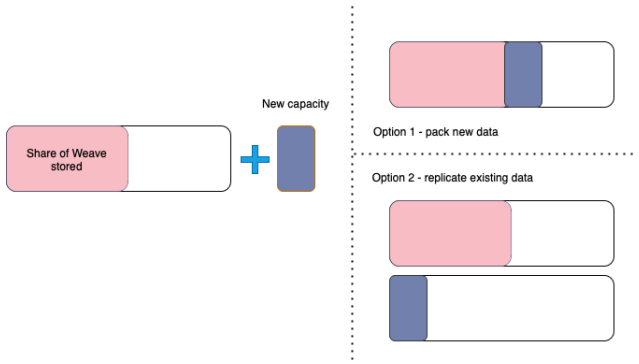

Figure 10: Miners are incentivized to organize into complete replicas (option 1), rather than making additional replicas of data that they already have (option 2.

The *rmpe* value can be seen as a penalty for copying existing partitions when adding new capacity as a miner. Consider the efficiency trade-off at different values of n when m is large:

- The reward for completing a replica is highest when a miner has close to a full copy of the data set. Indeed, if $n \to m$ and $m \to \infty$, we get an *rmpe* value of 3. This means that close to a full replica, the efficiency of seeking out new data is 3 times the efficiency of repacking existing data.
- When the miner stores half the weave, i.e. when $n = \frac{1}{2}m$, *rmpe* is 2. This means that the benefit to the miner of seeking out new data is twice the benefit of copying existing data.
- For lower values of n, the *rmpe* value tends towards but is always greater than 1. This configuration infers the greatest performance penalty upon the miner.

As the network grows ($m \to \infty$), there will be a strong incentive for miners to organize into full replicas. This promotes the creation of cooperative mining groups that together store at least one full replica of the dataset.

4.2 Network Metrics

Using the equations described in the previous section and some information from the network, we can determine the number of replicas that the network is storing. With each

block that miners create, we can identify whether the solution hash is generated from a first or second-range SPoA proof. In a network that only stores full replicas, we would expect this ratio to be 1:1. However, when miners mine with less than a full partition or duplicated partitions (and therefore gain an efficiency penalty), the ratio would be lower than 1.

We can calculate the average hashes per partition by calculating the ratio of the observed SPoAs. Suppose that, over the last 1,000 blocks there are n_1 first-range SPoAs and n_2 second-range SPoAs. This means that the average replica completeness is $\frac{n_2}{n_1}$ and, as a result, the mining efficiency per partition is:

$$e_m = \frac{1}{2}(1 + \frac{n_2}{n_1})$$

Using the above expression, we can accurately estimate the total number of partitions in the network. The expected number of hashes tried when the difficulty parameter is d is given by:

$$\mathbb{E}[trials] = \frac{1}{p} = \frac{2^{256}}{2^{256} - d}$$

The expected number of partitions that are needed to generate these many trials in a 120-second time period when each partition is only e_m efficient is:

$$\mathbb{E}[partitions] = \frac{\mathbb{E}[trials]}{800 e_m * 120} \tag{3}$$

With a partition size of 3.6TiB, we can derive the deployed storage capacity of the network:

$$Storage = \mathbb{E}[partitions] * 3.6TiB$$

All of these metrics regarding the stored dataset and average replica completeness can be computed from the observed values in the network without additional coordination overhead.

4.3 Incentives to Optimize Data Routing

Encouraging miners to organize into full replicas for efficient mining yields a number of useful downstream incentives. One such notable incentive is the impetus for miners to create optimized solutions for fast data routing within a peer-to-peer network, thereby addressing an otherwise complex and critical distributed systems challenge [21, 18, 29]. This necessity arises as nodes are required to be able to swiftly transmit any data chunk within the network to the location of any other chunk as part of mining, subsequently demanding the maintenance of routing capabilities that can be reused to facilitate data access for users and other miners.

The introduction of this new incentive for miners to optimize data routing is likely to stimulate a competitive landscape reminiscent of the race for optimized hashing hardware among Bitcoin miners. Such competition will drive innovation in routing infrastructure, ultimately fostering a more efficient and robust distributed network.

4.4 Bandwidth Sharing Incentives

Another downstream effect of Arweave's mining incentives for storage replication is the inherent imperative for miners to gain access to data in the network. This creates a wide variety of market structures for data access, including:

- **Karma and Optimistic Tit-for-Tat:** Some nodes in the Arweave network engage in a BitTorrent-like game [11] for bandwidth sharing. In this game, nodes share data reciprocally with other miners that share data with them. Additionally, nodes occasionally share data at random, optimistically expecting future reciprocation. Each node maintains their own rankings of peers, with no obligation to report how or why these rankings have been determined. Such a mechanism has been very successful and highly adaptive in BitTorrent, a data sharing platform that was responsible at one point for approximately 27% of the world's internet traffic [20].

- **Payments for physical distribution of disks:** Node operators may directly buy or sell disks loaded with data from the weave, in exchange for monetary or other forms of payment. This may be a preferable option for bandwidth-limited miners because of the amount of data required in order to operate Arweave nodes. This mode of transmission bypasses traditional packet filters and firewalls.

- **Payment protocols:** Nodes may also participate in protocols and markets that allow them to pay for data upon access. One such implementation is provided by the Permaweb Payment Protocol (P3) [40], that uses payment channels to incentivize a variety of services (including simple data access) within Arweave.

4.5 Scalability

Arweave blocks are created on average every 2 minutes, each containing a maximum of 1,000 transactions. This limit ensures that block validation and synchronization stays extremely lightweight, allowing the network to remain widely decentralized. However, this transaction limit does not impose any restriction on the size or quantity of data items that can be stored in a given block due to the network's "bundling" mechanism [1]. Bundling is a network-wide standard built on top of the core protocol for consolidating many different data entries into a single transaction. These data entries are functionally equivalent to top-level data storage transactions on the network as, upon retrieval, bundled transactions can be "unbundled" into their constituent items.

Arweave's maximum transaction size is $2^{256} - 1$ bytes, which can be subdivided into arbitrary numbers of individual data entries inside potentially recursive bundles. This allows the throughput of the network to be scaled without practically applicable limits. This optimization is possible because data uploads on Arweave are not parameterized – every byte on the network is part of the same global merkelized dataset, and is funded by a shared endowment. One

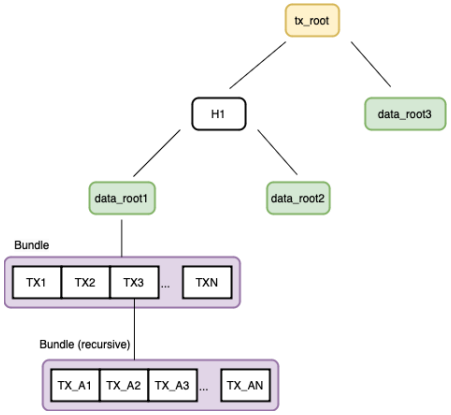

Figure 11: Bundling allows recursively stacking data uploads into a single top-level transaction.

of the components of this design is the aggregation of payments from individual data entries to the uploaded bundle. Users either aggregate payments for their data items within a single bundled transaction, or move their payments entirely off-chain to a bundling service which groups their data entries with those of other users.

In Arweave, transactions are selected for inclusion inside the 1,000 slots in each block according to their total value, as miners earn an inclusion fee proportional to the transaction fee. In the presence of block space scarcity, this incentivizes bundling services to recursively group transactions, adding to the scalability of the network. As a consequence, any number of bundlers and users can write to the network at any given time without leading to the typical block space auction mechanics of other blockchains. Further, competition between bundlers to build larger transactions creates downward pressure on the fees that they will impose on end users. This is in contrast to other blockchain networks, where competition for limited block space increases the fees imposed upon users until some users are priced out of utilizing the network.

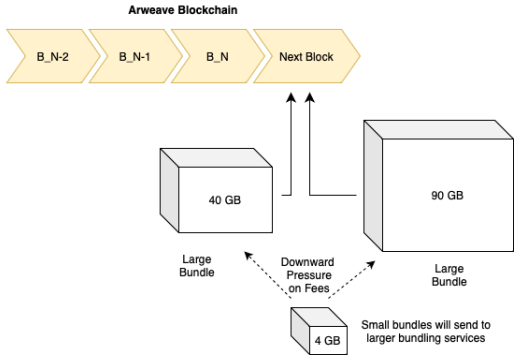

Figure 12: Preference for larger bundles creates incentives to recursively bundle data, minimizing fees.

One additional consequence of moving transaction negotiation for data uploads to off-chain bundling services is that users can pay for Arweave storage in any payment mode supported by the providers, with grouped data entries settled in AR by the bundlers. At the time of writing, the Arweave network supports at least 18 different payment modes through bundling services [14, 37].

5 Storage Endowment

Arweave's incentive model requires uploaders of data to pay a small transaction placement fee and provide an upfront contribution, denominated in AR, to the network's *storage endowment*. This endowment serves as a faucet through which miners are paid out over time, as they collectively provide proofs of replication of the dataset. The necessary payout from the endowment to maintain a piece of data decreases as the cost of storage declines.

5.1 Storage Pricing

Users pay upfront for the storage of 20 replicas for 200 years, at present costs. The trustless mechanism that the protocol employs to determine the price of storage acquisition from miners is described below. In this section, we sequentially construct the complete calculation of storage price that is used by the protocol.

During the period of a single block B with difficulty d_B, the estimated number of partitions in the network is given by equation (2) from §4.2:

$$\mathbb{E}[partitions] = \frac{2^{256}}{800 * 120 * (2^{256} - d_B)}$$

This expression multiplied by the partition size calculates the total amount of storage currently in the network at the time of block B:

$$\mathbb{E}[storage] = \frac{2^{256} * 3.6TiB}{800 * 120 * (2^{256} - d_B)}$$

The amount of AR given out as rewards to miners and the difficulty for the block could be used to estimate the storage acquisition cost – the price of servicing 1 GiB for 1 minute at the time of block B:

$$P^*_m(B) = \frac{r_B}{2 * 1024 * E_{d_B}[storage]}$$

where:

$P^*_m(B)$ = The estimated cost of storing 1 GiB for one minute at the time of block B.
r_B = Total reward for block B.

Using a single block period this estimation of storage price has high variance, owing to differences in the collected transaction placement fees and the difficulty adjustment algorithm. Therefore in practice, the network records the difficulty and released rewards over a large number of blocks. These recordings are used by the network to accurately calculate the storage acquisition cost from miners over a 6-week period preceding a block:

$$P_m(B) = \frac{\sum_{i=h_B-n}^{h_B} P^*_m(B_i)}{n}$$

where:

$P_m(B)$ = The average storage acquisition cost for 1 GiB-minute, taken over a 6-week period.

h_B = Height at block B.

n = Number of blocks in a 6-week period ($30 * 24 * 7 * 6 = 30,240$).

Using these calculations, the network can accurately estimate the acquisition cost of storage for one GiB for a 1-block period (~2 minutes):

$$P_b(B) = 2 * P_m(B)$$

Given this formula, the protocol calculates the present price of 20 replicas of any piece of data D for 200 years as follows:

$$P(D) = 20 * 200 * 365 * 30 * 24 * size(D) * P_b(B)$$

This is the price charged to the user as an upfront contribution to the storage endowment. Miners are paid out from the endowment over time as they prove storage of the network's dataset, according to the following calculation:

$$r_e(B) = 20 * P_b(B) * size(W) - (r_i(B) + r_f(B))$$

where:

$r_e(B)$ = The withdrawal from the endowment in block B.

$r_i(B)$ = The inflation reward released in block B.

$r_f(B)$ = The transaction placement fee for transactions accepted in block B.

$P_B(B)$ = The estimated cost of storing 1 GiB for one block at the time of block B.

W = The total dataset stored on Arweave at the time of block B.

5.2 Competition for Storage Efficiency

As a result of the storage pricing mechanism described above, Arweave miners are incentivized to compete to increase the efficiency of their mining operations. This has similar effects to the competition observed in the Bitcoin network, where miners have competed to minimize their cost of computing hashes by building optimized software and hardware. As a consequence of Arweave's storage pricing mechanism, an efficient market is created for storage acquisition without the need for parameterized "on-chain order books" that would hinder protocol scalability.

5.3 Deflation and Endowment Value

The Arweave network's endowment removes tokens from circulation every time data is uploaded, creating a reserve to pay for data storage over time. The storage purchasing power of the endowment is elastic, changing with the volume of data committed, the cost of data storage, and token value over time. One of the main drivers of change in the value of the endowment is that a decreasing cost of storage creates a corresponding proportional increase in storage purchasing power, leading to fewer tokens needing to be released from the endowment in the future. We call the rate of decline in overall costs for storing a unit of data for a fixed period of time the *Kryder+* rate. This rate incorporates the change in price of hardware, electricity, and operational costs surrounding data storage.

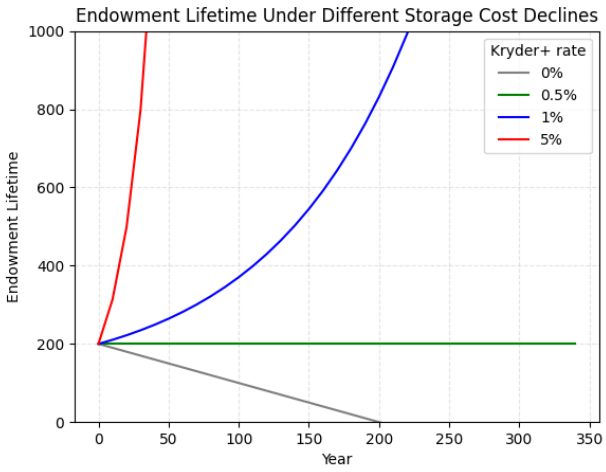

Figure 13: Endowment health is affected by the difference between the protocolized Kryder+ rate (0.5%) and actual Kryder+ rates, as well as token price changes.

Users pay for 200 years worth of replicated storage at present prices, such that only a 0.5% Kryder+ rate is sufficient to sustain the endowment for an indefinite term, in the absence of token price changes. Under these conditions, the storage purchasing power of the endowment at the end of each year would be equal to that at the beginning. The actual declining cost of storage over the last 50 years has been, on average, ≈38.5% per year [16]. We note that this pattern is a continuation of the diminishing real cost of information encoding and preservation that can be observed throughout history. Further, this trend appears likely to continue given the strong incentives and significant scope for improvement. The delta between the observed and protocol-specified Kryder+ rates is tuned to provide a wide safety margin for token price volatility, with excess leading to deflation in the token supply over time.

6 Decentralized Content Policies

The Arweave network employs a disintermediated, layered system of content policies without centralized points of control or moderation. The underlying principle of this system is voluntarism: every participant is able to choose what data they would like to store and serve, without any mandates from the protocol. This system allows each participant in the network to create and operate their own content policy without requiring consensus from others. The consequence of this is a diverse selection of content policies, applied at various stages of the infrastructure supporting Arweave-based applications:

- **Miners:** Miners in the network can run arbitrary compute (various forms of text, image, video, etc. analysis)

upon the data they store to filter content they deem unlawful or inappropriate. As miners store and publicly serve their content, they are subject to the local laws and regulations of their state. This disincentivizes them from storing illicit content that does not conform to local regulation.

- **Gateways:** Users often access content on Arweave through a gateway [30]. Gateways act as portals, allowing users and developers access to data in the Arweave network without running their own nodes. Just as miners can choose their own content policy with regard to the data that they store, gateways are also able to independently decide which content they index and serve. Further, the interoperability of gateways in the network allows users to choose gateways that conform to their personal beliefs and values.
- **Applications:** The final layer of content moderation that may affect Arweave users is at the application level. Each application built on data from Arweave may enforce an additional filter on content served by their interfaces as their developers have programmed. These application-level content policies can be embedded in the source code of the applications themselves and stored immutably on Arweave — giving users the ability to trust how the app will moderate content, permanently [26].

7 Protocol Evolution

Over time, it is inevitable that the circumstances surrounding the Arweave network will change. Truly permanent storage requires a system capable of adapting to such changes as they arise. We note that existing methods of blockchain protocol governance have fallen short of simultaneously allowing protocol adaptability and preservation of user guarantees [39]. In order to address these concerns, the governance mechanism deployed in the Arweave network improves upon the traditional blockchain forking model in order to provide a framework for incentivized evolution of the protocol. The governance procedures of the Arweave network are purely constitutional in nature. They are a social protocol agreed upon by community participants, allowing the technical implementation to mutate as necessary. The full Framework and the Principles that it references can be found on the Arweave network itself [32, 33]. In the remainder of this section, we describe the operation of this mechanism and its emergent incentives.

7.1 Mechanism Overview

At its core, the framework is a system that enables and rewards permissionless innovation, via forks, on top of the Arweave protocol. This mechanism gives rise to a form of evolution: creating powerful incentives to generate, test and select preferable mutations of the protocol.

By allowing anyone to offer a mutation of the protocol with an associated reward, a market is created for improvements. Broadly, the mechanism functions as follows:

1. Innovators choose a prior iteration of the protocol on which to base their new work. They select the parent for their innovation based on its perceived adoption and fitness.
2. Innovators then create a mutation of the protocol with new features and offer it to the community, minting a new quantity of tokens that they believe represents reasonable reward for their contribution.
3. Finally, community members assess the new mutation, opting to use it above alternatives if it offers sufficient improvements at a reasonable dilution.

The outcome of this process is a protocol that evolves to adapt to its environment, maximizing 'fitness' – its utility and robustness – while minimizing dilution. As can be inferred, successful innovators will need to satisfy the two primary market participants: users of the protocol and other innovators. If they achieve this, they will receive a market-determined reward for their efforts.

7.2 Incentives for Cooperation

The permissionless nature of this mechanism invites all market participants to cooperate rather than compete with prior versions of the protocol. Specifically, anyone attempting to build a permanent data storage system is incentivized to take part in this mechanism for three primary reasons:

- Faithful participation in the mechanism allows them to bootstrap adoption by easily inheriting an activated, engaged userbase. This maximizes the reward for their innovation, while minimizing expenses.
- By participating in the mechanism faithfully, innovators can expect that their iteration will become part of the lineage of permanent data storage systems. Consequently, others will advance their protocol – carrying their data and tokens forward – without further action on their behalf.
- Any builder attempting to create a permanent information storage system will need a flexible and adaptive mechanism to respond to an environment that is certain to change over time. Arweave's evolutionary framework provides a solution to this problem that all can participate in without cost or limitation.

Considering that these advantages are available to innovators with very little true cost – after all, they may choose to dilute as much as they consider appropriate – participation in the mechanism represents a preferable proposition rather than bootstrapping a competing service.

7.3 Incentives for Dataset Unification

Innovators are exposed to an additional powerful incentive: to maximize the value of their new evolution by merging the data of diverged lineages. This incentive emerges from the dynamic that a unified protocol will generally be more valuable than an evolution of a single strand of a diverged

lineage. The most basic strategy for achieving this unification has two components: addressing the technical concerns of the divergent communities, and including the data from diverging forks into the new evolution. If executed correctly, this strategy of unification leaves little reason for community members to continue participating in the divergent fork, encouraging them to migrate their usage and token holdings to the new evolution.

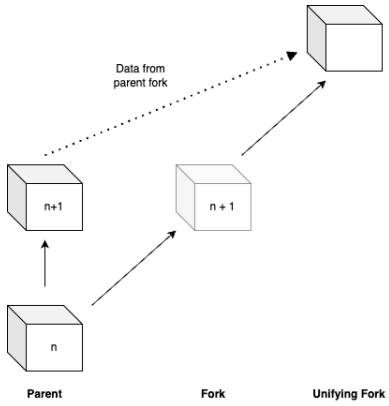

Figure 14: Data from divergent forks can be uploaded to accepted evolutions of the protocol.

Approximately, an innovator is incentivized to unify data from divergent forks in the evolutionary tree if the result of the following expression is positive:

$$V = F_M C_D - F_M C_N - D_C + D_V$$

where:

V = Net value gain or loss as a result of including the data from the divergent fork.

F_M = The market capitalization of the divergent fork.

C_D = Proportion of market capitalization that will move from an old fork to a new one if the old data is included.

C_N = Proportion of market capitalization that will move from an old fork to a new one regardless of whether the data is included.

D_C = The cost – in terms of necessary dilution – of copying the old data into the new evolution.

D_V = The intrinsic social value of having data from the divergent fork included in the new evolution.

Given that the D_C term in this equation is typically small compared to F_M, innovators are often incentivized to include data and unify the most prominent forks of the tree. Consequently, users can have confidence that their data will be included in the canonical version of the system as long as they upload to a network with appropriate prominence and usage.

When assessed in aggregate, the effect of these incentives is that token holders will accrue a basket of assets across different evolutions, while Arweave users can expect their data to be maintained by an ever-improving set of protocols as the network evolves. For a deeper analysis of the components of the framework, we refer readers to the companion to the Framework for Evolving Arweave [36].

8 Conclusion

In this paper, we have presented the Arweave protocol: a permanent information storage system that allows for disintermediated and peer-to-peer communication over time and space. We have presented the technical and economic mechanisms of this protocol and explored their emergent dynamics, as currently deployed in the network. As a component of this, a novel evolutionary system has also been described, which will allow the protocol to adapt to changing circumstances over time.

9 Acknowledgements

Arweave is the result of the combined efforts of many thousands of contributors and teams that have built infrastructure, services and products at various stages in the development of the network and its ecosystem. The Arweave protocol also enjoys a vibrant community of hundreds of thousands that support the network as it grows. The authors would like to thank them all for helping make this network and the fulfilment of its mission possible.

References

[1] *ANS-104: Bundled Data v2.0 - Binary Serialization.* URL: https://github.com/ArweaveTeam/arweave-standards/blob/master/ans/ANS-104.md.

[2] Lev Berman. https://github.com/ArweaveTeam/arweave/blob/master/apps/arweave/src/ar_block_index.erl.

[3] Lev Berman. https://github.com/ArweaveTeam/arweave/blob/master/apps/arweave/src/ar_tx.erl.

[4] Lev Berman and Sergii Glushkovskyi. URL: https://github.com/ArweaveTeam/arweave/blob/master/apps/arweave/src/ar_packing_server.erl.

[5] Lev Berman and Sergii Glushkovskyi. URL: https://github.com/ArweaveTeam/arweave/blob/4f5d69a810317bf529a4e7b2ca133fbcbc8c532f/apps/arweave/c_src/ar_mine_randomx.c.

[6] Lev Berman and Sergii Glushkovskyi. URL: https://github.com/ArweaveTeam/arweave/blob/master/apps/arweave/c_src/randomx_long_with_entropy.cpp.

[7] Lev Berman and Sergii Glushkovskyi. URL: https://github.com/ArweaveTeam/arweave/blob/master/apps/arweave/src/ar_vdf.erl.

[8] Lev Berman and Sam Williams. https://github.com/ArweaveTeam/arweave/blob/master/apps/arweave/src/ar_unbalanced_merkle.erl.

[9] *Blockchain.com*. https://www.blockchain.com/explorer/charts/hash-rate. [Accessed April 28, 2023].

[10] Dan Boneh et al. *Verifiable Delay Functions*. Cryptology ePrint Archive, Paper 2018/601. https://eprint.iacr.org/2018/601. 2018. URL: https://eprint.iacr.org/2018/601.

[11] Bram Cohen. *Incentives Build Robustness in BitTorrent*. 2003. URL: http://www.bittorrent.org/bittorrentecon.pdf.

[12] Danny Dolev and H. Raymond Strong. *Authenticated Algorithms for Byzantine Agreement*. Nov. 1983.

[13] Danny Dolev et al. "An efficient algorithm for byzantine agreement without authentication". In: *Information and Control* 52.3 (1982), pp. 257–274. ISSN: 0019-9958. DOI: https://doi.org/10.1016/S0019-9958(82)90776-8. URL: https://www.sciencedirect.com/science/article/pii/S0019995882907768.

[14] *Everpay Docs*. https://docs.everpay.io/en/docs/guide/dive/deposit.

[15] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. "Impossibility of Distributed Consensus with One Faulty Process". In: *J. ACM* 32.2 (Apr. 1985), pp. 374–382. ISSN: 0004-5411. DOI: 10.1145/3149.214121. URL: https://doi.org/10.1145/3149.214121.

[16] *Hard Disk Prices Over Time*. (Permalink).

[17] Robert V. Hogg, Joseph W. McKean, and Allen T. Craig. *Introduction to Mathematical Statistics*. 8th ed. Pearson, 2018.

[18] et. al Hun J. Kang. *Why Kad Lookup Fails*. URL: https://www-users.cse.umn.edu/~hoppernj/kad.pdf.

[19] Brennan Lamey. *KwilDB: The Decentralized SQL Database*. URL: https://uploads-ssl.webflow.com/632df6381908134a8b796288/63373a98cfa42878438c5449_KwilDB%20White%20Paper.pdf.

[20] Fabrizio Marozzo, Domenico Talia, and Paolo Trunfio. "A Sleep-and-Wake technique for reducing energy consumption in BitTorrent networks". In: *Concurrency and Computation Practice and Experience* 32 (Feb. 2020). DOI: 10.1002/cpe.5723.

[21] Petar Maymounkov and David Mazières. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". In: *Revised Papers from the First International Workshop on Peer-to-Peer Systems*. IPTPS '01. Berlin, Heidelberg: Springer-Verlag, 2002, pp. 53–65. ISBN: 3540441794.

[22] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. 5th. CRC Press, 2001, p. 251. ISBN: 978-0849385230. URL: https://archive.org/details/handbookofapplie0000mene/page/250/mode/2up.

[23] Ralph Merkle. "Secrecy, authentication, and public key systems". PhD thesis. 1979. URL: https://link.springer.com/content/pdf/10.1007/0-387-34805-0_21.pdf.

[24] Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System". In: *Cryptography Mailing list at https://metzdowd.com* (Mar. 2009).

[25] Outprog. *Storage-based Consensus Paradigm*. URL: https://mirror.xyz/0xDc19464589c1cfdD10AEdcC1d09336622b282652/KCYNKCIhFvTZ1DmD7IpXr3p8di31ecC283HgMDqasmU.

[26] India Raybould. *What is the permaweb?* (Permalink).

[27] Lev Berman Sam Williams and Sergii Glushkovskyi. URL: https://github.com/ArweaveTeam/arweave/commits/master/apps/arweave/src/ar_block.erl.

[28] R.R. Schaller. "Moore's law: past, present and future". In: *IEEE Spectrum* 34.6 (1997), pp. 52–59. DOI: 10.1109/6.591665.

[29] Jiajie Shen et al. "Understanding I/O Performance of IPFS Storage: A Client's Perspective". In: *Proceedings of the International Symposium on Quality of Service*. IWQoS '19. Phoenix, Arizona: Association for Computing Machinery, 2019. ISBN: 9781450367783. DOI: 10.1145/3326285.3329052. URL: https://doi.org/10.1145/3326285.3329052.

[30] Jack Smith. *Gateways*. https://cookbook.arweave.dev/concepts/gateways.html.

[31] Arweave Team. *SmartWeave*. https://github.com/ArweaveTeam/SmartWeave. June 2022.

[32] Arweave Core Team. *Framework For Evolving Arweave*. (Permalink).

[33] Arweave Core Team. *Principles of the Arweave network*. (Permalink).

[34] tevador. *RandomX*. https://github.com/tevador/RandomX. Accessed: April 29, 2023. Dec. 2022.

[35] D. Thaler and B. Aboba. *RFC 5218 - What Makes for a Successful Protocol?* https://www.rfc-editor.org/info/rfc5218. Online; accessed 2 May 2023. July 2008.

[36] *Understanding the Framework For Evolving Arweave*. (Permalink).

[37] *Using Other Currencies*. Online; accessed 2 May 2023. URL: https://docs.bundlr.network/sdk/using-other-currencies.

[38] Sam Williams. *Public Square Protocol*. URL: https://github.com/luckyr13/public-square.

[39] Sam Williams and Abhav Kedia. "Fair Forks: Towards Incentivized Protocol Governance". In: (Oct. 2022). (Permalink).

[40] Sam Williams and Dan MacDonald. *Permaweb Payment Protocol*. URL: https://blog.ar-io.dev.

[41] Anatoly Yakovenko. *Solana: A new architecture for a high-performance blockchain*. Tech. rep. 2020. URL: https://solana.com/solana-whitepaper.pdf.