Arweave: 永久信息存储协议

Sam Williams sam@arweave.org
Abhav Kedia abhavkedia@gmail.com
Lev Berman lev@arweave.org
Sebastian Campos-Groth sebastian@arweave.org

Translated into Simplified Chinese by Gerry Wang (gerry@arweave.org)

DRAFT 17
2024 年 5 月 29 日

摘要

在本文中，我们对 Arweave 的描述是：一个在去中介化的空间与时间中传输信息的协议。Arweave 在网络空间中提供无需中心化权威控制的任意可扩展的永久数据和言论账本。该网络通过利用简洁的复制存储证明这一全新的区块链共识机制来实现目标。这种证明系统的优势在于极大降低了验证所带来的成本，以此在最大化去中心化的同时，最小化计算和带宽成本。再与一个激励数据复制的存储基金相配合，让网络可以完全以自主、可预测的方式运行。最后，本文还描述了一种基于分叉（fork-based）的新演化系统，它能够在保证协议灵活性的同时还能保护用户的权利不受影响。

1 介绍

自从印刷术发明以来，社会中信息流动的速度急剧增加，直至 20^{th} 世纪末互联网的出现让其达到了顶峰。这让人类个体对周围世界的了解程度也不断增加。但目前，互联网实际上仍然受到中心化控制点的控制，导致信息分发效率低下：每年都会丢失一定比例的有用内容，而这些中心化的控制和汇聚点有权任意审查和操纵信息流。Arweave 旨在通过允许信息在极长的时间和空间距离上的去中介化传输来解决这些问题。虽然关于 Arweave 网络的讨论已在许多社群中进行着，但本文将会为你提供更为精确的介绍与描述。

Arweave 是一个永久性信息存储系统。“永久性”这个词有多种定义：《牛津英语词典》将其定义为“持续存在或无限期保持不变”，而《韦氏词典》则将该词定义为“继续或持久存在，无基本或显著变化”。在这篇文章中，我们使用一个包含这两种思想的永久性定义：Arweave 网络旨在以最大可能的期限存储数据，且数据不发生变化。为了实现其目标，Arweave 协议由三个核心组成部分构成：

- **存储的加密证明：**一个简洁的加密证明系统，用于验证数据的复制和可访问性。
- **存储基金：**一个可预测的、自执行的基金，利用技术进步随时间带来的通货紧缩效应来支付存储费用。
- **激励演化：**通过生成和奖励非强制性的网络升级，允许协议适应性的机制。

Arweave 协议被精确地编码、不可篡改且完全透明。本文准确地描述了编写时（v2.7.0 版本）协议已实现的每个组成部分。我们鼓励用户阅读本文并自行审查协议，以便直接了解协议提供的服务和保障。为了便于验证协议，我们在本文的各个部分都提供了相关代码库的引用。

2 协议设计

Arweave 构建的一个核心组成部分是用于在网络中添加和验证数据的去中心化共识系统。去中心化共识是分布式计算的一个子领域，涵盖了对允许网络中的参与者即便在有敌对节点存在的情况下，也能就某一状态达成一致共识的重要研究 [12, 13, 15]。这个子领域随着比特币“中本聪共识机制”的出现而获得了广泛地关注，该机制首次允许在敌对和无需许可的环境中达成一致共识 [24]。由于这一创新，比特币创建了第一个不依赖于中心化管理货币政策的数字货币。Arweave 从中获得灵感，并对其进行了优化调整，来激励网络中信息的永久性存储。

Arweave 是一个由“节点”组成的全球计算机网络，这些节点共同存储网络中所有数据的多个副本。希望在

Arweave 上存储信息的用户，会向网络的存储基金支付一次性前期费用，并通过将数据传输给网络内的节点来上传相应的数据。每当某个节点成功挖掘（确认）一个区块时，节点们就会定期对新进入全球分布式数据库网络中的数据达成共识。一个区块是一个交易列表，而每个交易中包含了要添加到网络的新数据，或者是其数字货币 AR 的转账交易，或者两者都有。挖矿是指每个节点参与运行的一个过程，它既接受新进入网络的数据，也会对已上传数据的存储情况进行验证。节点在确认包含了交易的区块后，便会从网络中的其他节点那里“拉取”想要复制存储的数据，以便日后进行更加高效地挖矿。

2.1 设计原则

设计该协议的两个主要原则是：

- **极简主义：**协议的设计理念是保持直接且最小程度的主观判断，以促进尽可能广泛的社会共识。Arweave 仅在构建其数据结构和算法时使用了经过良好测试的加密原语。

- **通过激励进行优化：**协议并不是用于对期望的行为进行规范，而是激励参与者去实现理想的结果。实现这些结果的具体机制将自然产生，并随时间而自行演化。

由于其保守的设计原则，Arweave 协议专注于简洁地解决一个问题——永久且可扩展的数据存储。这使得建立在协议之上的应用层可扩展且可组合，使网络的使用能够更加广泛且多样化 [35]。这催生了许多去中心化智能合约平台、数据库和应用程序的出现 [31, 19, 25, 38]，它们都是建立在 Arweave 之上的。此外，Arweave 的高效证明系统对硬件和带宽的要求极低——与比特币相当，且不受数据集大小影响——这能最大化网络中的参与度和去中心化程度。

为了说明激励在协议设计中的有效性，我们可以检视其在比特币网络中的影响。比特币奖励矿工发现一个随机数（*nonce*），该随机数与一个候选区块一起，产生一个低于某个特定值的哈希值。从 $\{nonce, block\}$ 对得到的哈希值的均匀随机分布激励矿工寻找以最低成本计算最大数量哈希值的方法。这导致了专用矿机的出现与发展，自 2011 年以来，每秒可计算的哈希数增加了 10^{13} 倍。值得注意的是，到本文撰写时，这一增长速度在同一时期内比摩尔定律 [28] 高出近 10^{10} 。同样的激励也导致了同期比特币网络中每个哈希的成本下降了 100 万倍 [9]。在

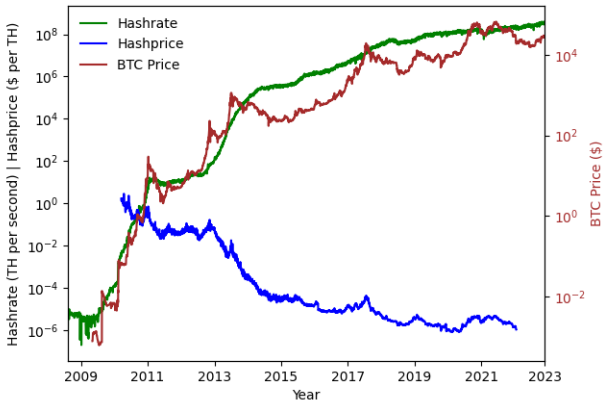

图 1: 比特币中高效挖矿的激励措施已促成了计算哈希速度和成本的显著提升（Y 轴为对数刻度）。[9]

Arweave 中，我们调整了比特币的激励机制，以激励参与者优化存储证明和数据传输的问题解决方案。

3 加密存储证明

Arweave 的挖矿机制将许多不同的算法和数据结构组合成一个非交互式且简洁的数据复制存储证明。在本节中，我们将首先抽象地检视这些概念，然后描述它们在 Arweave 挖矿过程中是如何运作的。

3.1 区块索引

Arweave 中最高级别的数据结构称为区块索引 [2]。它是一个由 3 元组组成的默克尔化列表，每个元组包含了一个区块哈希（*block hash*）、一个 weave 大小（*weave size*）和一个交易根（*transaction root*）。这个列表以一个哈希默克尔树 [23] 的形式表示，顶层的默克尔根代表网络中的最新状态。这个根是由两个元素的哈希组成：最新的元组（代表最新区块）和前一个区块索引的默克尔根。在创建区块时，矿工将前一个区块索引的默克尔根嵌入在新区块中。

通过在每个区块中构建并嵌入一个区块索引的默克尔根，一个已对最新区块执行了 SPV 证明 [24] 的 Arweave 网络节点能够完全验证任何之前的区块。这与传统的区块链形成对比，在传统区块链中，对旧区块中的交易进行单独验证时，需要对该区块回溯至整个链的完整验证。在对网络顶端进行 SPV 验证之后，区块索引验证机制使用户能够在不需要下载或验证中间区块头的情况下，验证 weave 中的旧区块。

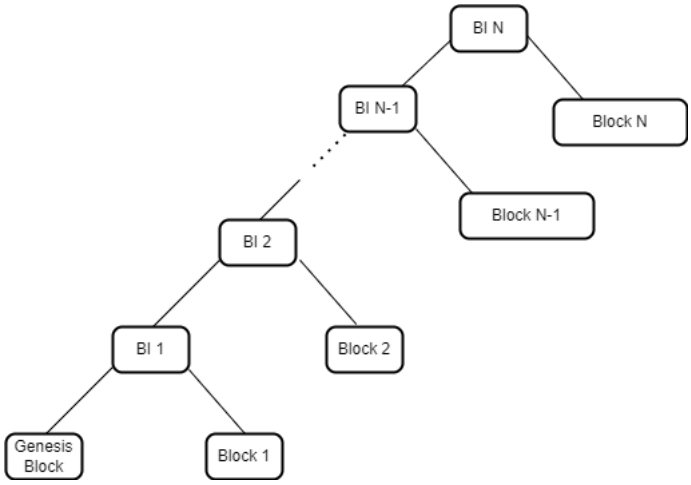

图 2: 区块索引代表了 Arweave 网络中区块的一个默克尔树

3.2 默克尔化数据

当信息需要被上传到 Arweave 网络中时，用户会将数据分割成 256 KiB 大小的数据块。准备好这些块之后，再将其构建成一个默克尔树，并将它的根 (称为 *data root*) 提交到一个交易中。该交易被签名并由上传者发送到网络。每个这样构建的交易都被记录在其所属区块的交易根 *transaction root* [3] 里。因此，一个区块的交易根实际上是该区块包含的所有交易数据根的默克尔根。

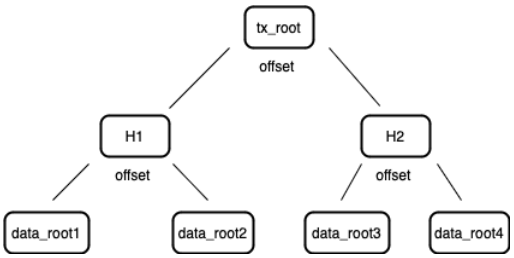

图 3: 交易根是区块内每个交易中的数据根的默克尔根。

Arweave 网络中的矿工将这样的交易组合成一个区块，使用交易内的数据根计算交易根，并将这个交易根包含在新区块中。这个交易根最终会如上所述出现在顶层的区块索引中。这整个过程创建了一个不断扩展的默克尔树，其中包含了协议中按顺序排列成块的的所有数据。

3.3 简洁的访问证明

在 Arweave 网络里，默克尔树的节点通过标记数据在节点 “左侧” 或 “右侧” 的偏移位置进行标识 [8]。这种标记方法能够生成简洁的默克尔证明，使验证数据集中特定位置上的数据块的存在性变得更加简单。这类默克尔树

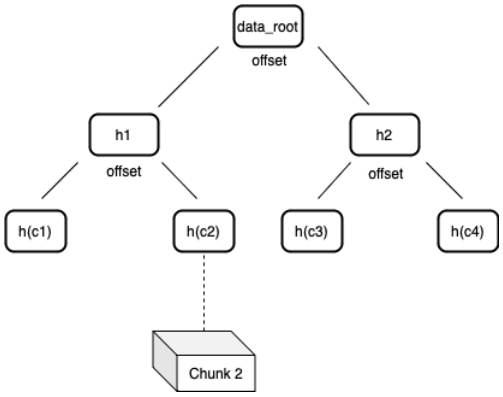

图 4: 数据根（data root）是单个交易中所有经过哈希处理的数据块（Chunks）的默克尔根。

被称作 “不平衡树”，因为节点一边的叶节点数可能不等于另一边。这种结构可以被矿工用来创建一个数据可用性的简洁且非互动的证明过程，我们也称其为简洁的访问证明。

简洁的访问证明（Succint Proof of Access SPoA）游戏从 Bob 想要验证 Alice 是否在一个默克尔化数据集的特定位置存储了数据开始。为了参与游戏，Bob 和 Alice 应该有相同的数据集的默克尔根。游戏分为三个阶段：挑战、证明构建和验证。

挑战： 游戏开始时，Bob 向 Alice 发送一个数据块偏移量的挑战（*challenge offset* ）———一个 Alice 需要证明可以访问的索引字节。

构建证明： 收到这个挑战后，Alice 开始构建证明。她搜索自己的默克尔树，找到与偏移量对应的数据块。

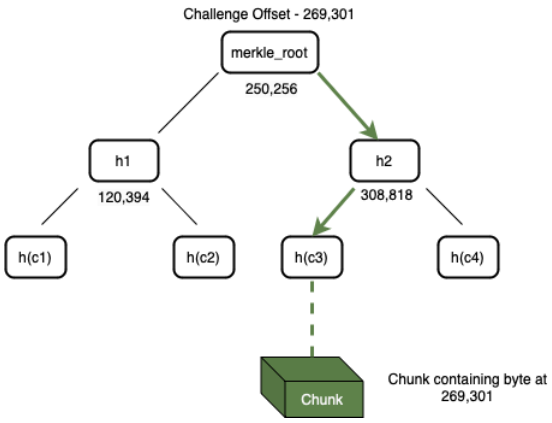

图 5: Alice 可以通过遍历带有偏移量标签的默克尔树来搜索挑战的数据块（Chunk）

在构建证明的第一阶段，Alice 从存储中检索整个数据块。然后，她使用该数据块和默克尔树构建一个默克尔证明。她对整个数据块进行哈希处理，从而获得一个 32

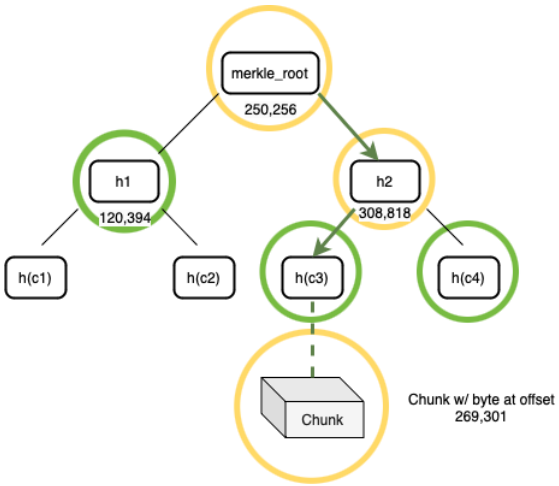

图 6: 一个挑战偏移量的证明包括了存储在偏移量处的数据块及其默克尔路径

字节的字符串作为标识符，并在树中找到这个叶节点的父节点，并将其添加到证明中。父节点是一个包含其偏移量和两个子节点的哈希值的节点，每个哈希值都是 32 字节长度。然后递归地将每个父节点添加到证明中，直到 Alice 达到默克尔根，让这一系列节点形成一个默克尔证明。随后，Alice 将这个证明连同数据块一起发送给 Bob [8]。

验证： Bob 接收到了来自 Alice 的默克尔证明以及数据块。他通过一步步检查从树顶端到底部叶子的哈希值来核实默克尔证明的正确性。接着，他会对 Alice 提供的证明中的数据块进行哈希运算，确认这个数据块是否正是位于正确位置的那个数据块。如果这些哈希值都对得上，就说明证明是有效的。但是，如果在验证过程中发现任何哈希值不符，Bob 会立即停止检查，并判定 Alice 的证明无效。这个检验过程的结果是个简单的是或否，表明证明是否通过。

3.3.1 SPoA 复杂性

构造、传输和验证简洁访问证明（SPoA）的复杂度是 $O(\lceil logn \rceil)$，其中 n 代表数据集的大小，以字节计。实际上，这意味着在一个大小为 2^{256} 字节的数据集里，一个单字节的位置和正确性的证明可以仅通过 $256 * 96B$（最大路径大小）+ 256KiB（最大数据块大小）≈ 280 KiB 来传输。

3.4 副本的唯一性

Arweave 网络将矿工的挖掘能力设计为与其（或一组合作的矿工）可以访问的 weave 副本数量成正比。在没有副本唯一性方案的情况下，一份数据的副本在内容上彼此是完全相同的。为了能够激励和衡量单个数据的多个唯一副本，Arweave 使用了一套打包（Packing）机制来实现。

3.4.1 打包

打包机制能够确保矿工无法使用完全相同的一个数据块备份来伪造出多个不同副本的简洁访问证明（SPoA）。Arweave 的打包方案使用 RandomX [34] 将常规数据块转换为“打包”的数据块，这个过程会为矿工带来了计算成本。

Arweave 中的打包工作如下 [4, 5, 6]：

- 数据块偏移量 *chunk offset*、交易根 *transaction root* 和挖矿地址 *mining address* 被结合在一个 SHA-256 哈希中以生成一个打包密钥。
- 打包密钥被用作算法的熵，该算法在几轮 RandomX 执行后生成一个 2 MiB 的暂存器。这个打包机制的组成部分对于不遵守规则的矿工来说带来了重大成本。
- 结果熵的前 256 KiB 被用来通过费斯妥密码（Feistel cypher）[22] 对数据块进行对称加密。这一计算产生了一个打包块。

如果在连续读取之间的时间内，打包数据块的总成本超过了存储打包数据块的成本，那么矿工将被激励去存储唯一的数据块，而不是按需打包数据块，以此来提高挖矿效率。为了激励这种正确的行为，必须满足以下条件：

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

其中：

c_s = 存储一个数据块的单位时间成本
c_p = 打包一个数据块的成本
t = 数据块读取之间的平均时间

打包总成本包括计算所需的电力价格、专用硬件的价格及其平均使用寿命。存储成本同样可以从硬盘成本、硬盘平均故障时间、运行硬盘的电力成本以及与运行存储硬件相关的其他成本中计算得出。我们可以计算按需高效挖矿者与存储打包数据块的诚实挖矿者的成本比率，以估算激励打包数据块的安全边际。实践中，Arweave 使用一个安全比率 *safety ratio* – 即按需创建数据块的成本与存储数据块的平均访问间隔时间的成本的比率——在撰写本文时，基于商业可获得硬件的基准测试，大约为 19。

3.4.2 RandomX

为了保持对硬件加速的抵抗力，Arweave 采用了一种精心选择的哈希算法 RandomX 对打包密钥进行处理。RandomX 针对通用 CPU 进行了优化，通过在哈希过程中实行随机代码执行和多种内存密集型技巧，降低了专用硬件的效率 [34]。RandomX 利用随机生成的程序，与现代 CPU 的硬件特性高度匹配，这表明提高 RandomX 哈希速度的唯一方法是开发更快的通用 CPU。尽管 Arweave 协议在理论上可能促进了更快 CPU 的开发，但我们认为，相对于已有的提升 CPU 速度的动机，这种激励的影响几乎可以忽略不计。

3.5 可验证延迟函数

可验证的延迟函数（*verifiable delay function* VDF）允许对事件之间时间流逝进行计算验证。VDF 需要执行指定数量的顺序步骤来评估，但产生的唯一输出可以高效且公开地验证 [10, 41]。构建 VDF 有几种技术。Arweave 使用一种链式哈希技术，其中 VDF 如下递归定义：

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

对于 $n > 1$，且：

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

Arweave VDF 使用的哈希函数是 SHA2-256 [7]。通过这种结构，如果最快的可用处理器每秒只能产生 k 个连续的 SHA2-256 哈希值，那么仅使用种子（seed）来对 $V(k, seed)$ 作出正确计算至少需要 1 秒时间。这是因为没有计算第一个哈希，就不可能计算出哈希的哈希。一个正确生成的 $V(k, seed)$，称为检查点（*checkpoint*），意味着证明者接收种子和生成这个检查点之间至少经过了 1 秒时间。

VDF 结构需要 $O(n)$ 时间来完成 n 步。然而，如果中间检查点被传输给了验证者，那么使用 p 个并行线程可以在 $O(\frac{n}{p})$ 时间内完成对正确生成 VDF 输出的验证。我们之所以使用链式哈希 VDF 而不是其他 VDF 结构，是因为它简单的假设，以及完全依赖于哈希函数的稳健性的特性。

3.6 简洁的复制证明

前面描述的简洁访问证明（SPoA）游戏可以被任何参与者用来证明他们在数据集的特定位置存储了一些数据。这个系统还可以用来创建第二个游戏，称为简洁的复制证明（SPoRes），它将允许证明者向验证者证明其存储了一定数量的数据副本。而且无论数据集大小如何，这个验证过程都能以极小的数据传输和验证开销完成。

3.6.1 SPoRes 游戏

简洁的复制证明（SPoRes）游戏是这样进行的。Alice 声称她存储了 n 份默克尔化数据集的副本。Bob 想要验证 Alice 的声明。为了做到这一点，他提供给 Alice 一个难度参数 d 和一个随机种子。Alice 使用这个种子生成一个每秒钟发出一个检查点的 VDF 链，这个检查点可以用来解锁数据集内最多 k 个 SPoA 挑战。每当她有对应于这些挑战的打包数据块时，Alice 可以构建相应的 SPoA 证明。然后将这些证明进行哈希处理，并与 Bob 的难度参数 d 进行对比。如果证明哈希值大于 d，则 Alice 找到了一个有效解决方案，并将相应的证明发送给 Bob。Bob 则将从发送随机种子到从 Alice 收到有效证明之间的时间记录下来。

基于难度 d，单次尝试找到有效解决方案的概率由以下给出：

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

如果 Alice 有 n 个副本，并且每秒每个副本可以进行 k 次尝试，那么她在任何给定的一秒时间内找到解决方案的概率是：

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

Alice 提交证明所需的时间可以用一个几何随机变量 $X \sim geom(p_2)$ 来模拟，成功的概率为 p_2。这个随机变量取决于 d，k 和 n。为了验证 Alice 的声明，Bob 发送一个难度 d，使得她可以每 120 秒向他提交一次证明，前提是她有 n 个副本。这相当于发送了一个难度参数：

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

或者:

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

如果 Alice 可以在要求的时间框架内提交证明，那她很可能拥有正确数量的副本。然而，单一的证明是不足以让 Bob 确信 Alice 没有说谎——毕竟，即使存储的副本数较少，她也可能幸运地快速找到一个证明。但如果 Alice 能在很长一段时间内始终能以平均每 120 秒来提交证明，Bob 就可以合理地确信 Alice 存储了符合期望的数据量。

我们将尝试量化 Bob 对 Alice 存储的确定性。假设 Alice 声称存储了 20 个副本，并且在 2 周的时间内以平均

120 秒的速度一致地提交了证明（总共 10,080 个证明）。此时，Bob 更加感兴趣的是，如果尽管 Alice 只存储了 19 个（或更少）副本，她依然能够提交这些证明的概率是多少。这对应于一个不同的一秒钟内提供证明的概率：

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

经过简化：

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

这个概率可以使用 Bob 为诚实的 Alice 生成的 d 值来计算。如果她存储了 19 个或更少的副本，她一秒钟内提供证明的概率可以通过下面给出的一系列推导得到。其中，p_2 代表了 Alice 诚实存储了 20 个副本时，一秒钟内产生证明的概率，而 p_2^* 代表如果她在说谎（存储 19 个副本）时，一秒钟内提供证明的概率：

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

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

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

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

因此，$p_2^* = 0.00791832081$，对应于预期的证明生成时间为 126.2894 秒。设 X 为从 p_2 得到的随机变量，即 $X \sim geom(p_2)$ 。这代表了 Alice 在说谎情况下的分布——只存储了 19 个副本。X 的期望值（平均值）是：

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

我们可以使用中心极限定理 [17] 来估算样本均值低于 120 的概率，即与上述得到的期望值 $\mathbb{E}[X^*]$ 不同。这个概率表示为：

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

对于大量样本来说，这个不等式左侧趋向于一个均值为 $\mathbb{E}[X^*]$ 和方差为 $\frac{\sigma_{X^*}^2}{n}$ 的正态分布，其中 σ_{X^*} 是 X_d 的标准差，而 n (这里 =10,080) 是样本大小 [17]。因此，公式 (2) 得出：

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

在上述方程中，$\mathcal{N}$ 代表一个正态分布。我们可以将这个分布转换为其标准正态形式，以获得等价的概率：

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

最终，使用标准正态分布表，我们可以获得 Alice 在这 2 周时间周期中说谎的概率：

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

因此，在这个例子中，Bob 可以以约 ≈99.99997416% 的确定性判断，在这 2 周时间里 Alice 存储了超过 19 份数据集的副本。我们注意到，如果 Alice 存储的副本少于 19 份，这种确定性会更高，因为预期的证明生成时间将长于 126.3 秒。我们还注意到，整个证明系统仅需要每 120 秒传输一次证明。这是 Alice 和 Bob 之间的平均数据传输率为每秒 2.389 KiB(280 KiB/120 秒)，与比特币 [24] 网络的同步开销相当。

最后，我们注意到，这些概率不会随着数据集大小的任意增加而改变，而增加采样周期将以超线性的速率持续提高 Bob 对 Alice 副本数量的确定性的准确度（图 7）。

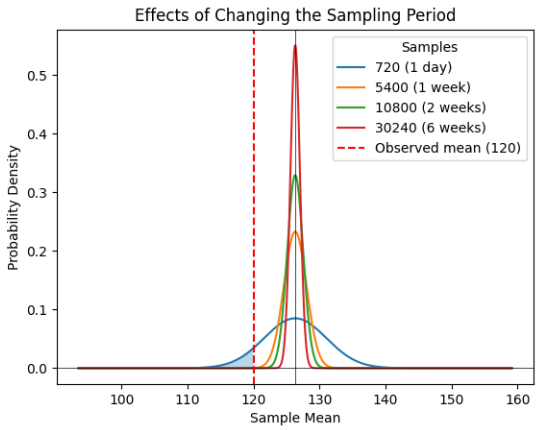

图 7: 在简洁复制证明（SPoRes）游戏中，确定性相对于样本持续时间的增加是超线性的。经过两周的样本收集，维持少于 19 个副本的概率是微不足道的 ($P < 0.0000002584$).

正式地，一个 SPoRes 游戏由其参数定义：

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

其中：

$r =$ 用于游戏的数据集的默克尔根。

$k =$ 每个副本每秒钟解锁的最大的挑战数量。

$d =$ 决定每次尝试时的成功概率的难度参数。

4 协议结构

在 Arweave 挖矿中，部署了在前一节中概述的 SPoRes 游戏的修改版本。在挖矿过程中，协议充当了 Bob 的角色，而网络中的所有矿工共同扮演 Alice 的角色。SPoRes 游戏的每个证明都用于创建 Arweave 的下一个区块。具体来说，Arweave 区块构建使用了以下参数：

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

其中：

BI = Arweave 网络的区块索引。
n_p = 矿工存储在 Weave 中的 3.6 TiB 分区数量。
d = 网络的难度。

这些参数允许每个检查点每个分区最多 800 个哈希。一个成功的证明是那些大于难度值的证明，而这个难度值会随时间被调整，以确保平均每 120 秒挖出一个区块。如果区块 i 与区块 $i+10$ 之间的时间差为 t ，那么从旧难度 d_i 到新难度 d_{i+10} 的调整如下计算：

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

其中：

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

新计算的难度决定了基于每个生成的 SPoA 证明，挖掘区块成功的概率，具体如下：

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

同样，VDF 的难度会重新计算，目的是为了保持检查点周期在时间上每秒发生一次。

4.1 完整副本的激励机制

Arweave 利用 SPoRes 进行区块构建，这包括了一个假设：在保持完整副本的激励下，无论是单独矿工还是矿工之间的合作，都会表现出理性的行为。本节将探讨这些激励是如何提供的，以及它们在 Arweave 网络中的有效性是如何得到验证的。在之前介绍的 SPoRes 游戏中，为数据集的一半数据存储两份备份所产生的 SPoA 挑战哈希数量与存储整个数据集的完整副本是相同的。但在 Arweave 实际部署的版本中又做了改动，其目标就是激励矿工无论是通过合作还是独立运行，都以存储并维护完整数据集为主要策略。

协议通过将每秒解锁的 SPoA 挑战数量分成两半来提供访问完整副本的激励——一半在矿工存储数据集的一个分区中，另一半在所有分区的随机一个中。为了理解这一点，我们将检查前一节中描述的 VDF 构造是如何被用来解锁 SPoA 挑战的 [27]。

Algorithm 1 解锁 SPoA 挑战

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

大约每秒钟，VDF 链会输出一个检查点 [7]。对于矿工存储的织网的每个分区，这个检查点会与挖矿地址、分区索引和原始 VDF 种子混合，以获得一个中间 RandomX 哈希。这个 256 位的哈希被视为一个数字，并除以分区的大小，产生一个余数，该余数被用作回溯偏移量。它解锁了从这个偏移量开始的连续 100 MiB 的回溯范围内的 400 个 256-KiB 挑战。激励完整副本的机制是，分区内的每个此类回溯范围还会解锁第二个回溯范围中的另外 400 个全局挑战，这些挑战的约束是第二范围内的解决方案需要在第一个范围的对应位置有解决方案。

4.1.1 每个已打包分区的性能

每个已打包分区的性能指的是每个分区为每个 VDF 检查点所产生的 SPoA 挑战数量。当矿工存储的分区是唯一副本时，SPoA 挑战数量将大于矿工存储相同数据的多个备份时的数量。

如果矿工存储的是唯一副本数据，每个打包分区将产生所有第一范围的挑战，以及落在该分区内的少数第二

范围挑战。若编织网络中总共有 m 个分区，矿工存储了 n 个唯一分区副本，那么每个打包分区的性能为：

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

当矿工存储的分区是相同数据的备份时，每个打包分区仍然会产生所有第一范围挑战。然而，只有在 $\frac{1}{m}$ 次情况下，第二回调范围会位于同一个分区内。这相当于一个非常明显的性能惩罚，产生的比率仅为：

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

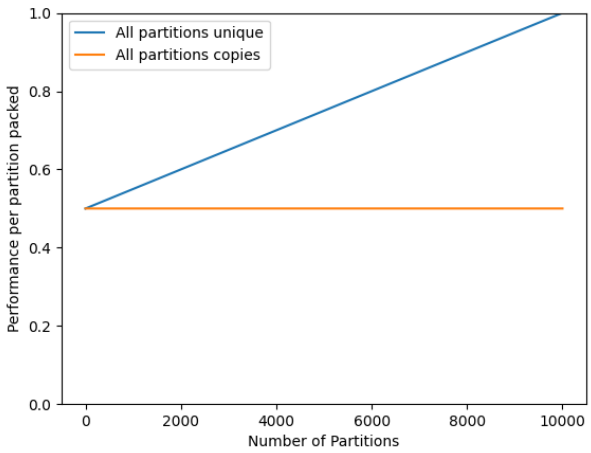

图 8: 当一个矿工（或一组合作的矿工）完成他们的数据集时，给定分区的性能会提高。

图表 8 中的蓝色线为 $\{perf_{unique}(n,m)\}$，说明了当存储最大数量 m 个分区中的 n 个分区时，每个分区的效率。该图直观地表明了，当矿工只存储了很少的分区副本时，每个分区的挖矿效率仅为 50%。当存储和维护所有数据集，即 $n=m$ 时，挖矿效率达到最大化的 1。

4.1.2 总哈希率

总哈希率（见图表 9 所示）由以下方程给出，通过将每个分区（*per partition*）的值乘以 n 得到：

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

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

以上公式表明了随着编织网络（Weave）大小的增长，如果不存储唯一副本数据，惩罚函数（Penalty Function）随着存储分区数量的增加而呈二次方增长。

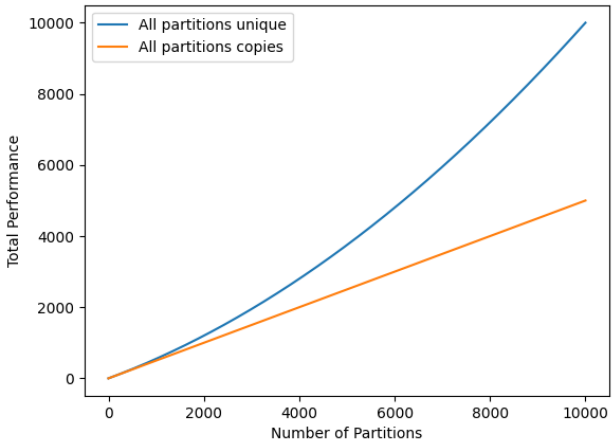

图 9: 唯一副本数据集和备份数据集的总挖矿哈希率

4.1.3 边际分区效率

基于这个框架，我们来探讨矿工在添加新分区时面临的决策问题，即是选择复制一个他们已有的分区，还是从其他矿工那获取新数据。当他们从最大可能的 m 个分区中已经存储了 n 个分区唯一副本时，他们的挖矿哈希率是成比例的：

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

增加一个新分区的额外收益是：

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

复制一个已打包分区的（较小）收益是：

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

将第一个数量除以第二个数量，我们得到矿工从获取一个新分区相对于复制一个现有分区的额外效率获得的边际效率。我们称之为相对边际分区效率（*relative marginal partition efficiency*）：

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

rmpe 值可被视为矿工在添加新容量时复制现有分区的一种惩罚。当 m 很大时，考虑不同 n 值下的效率权衡：

- 当矿工拥有接近完整数据集副本时，完成一个副本的奖励最高。实际上，如果 $n \to m$ 并且 $m \to \infty$，我们得到一个 *rmpe* 的值为 3。这意味着，接近完整副本时，寻找新数据的效率是重新打包现有数据效率的 3 倍。

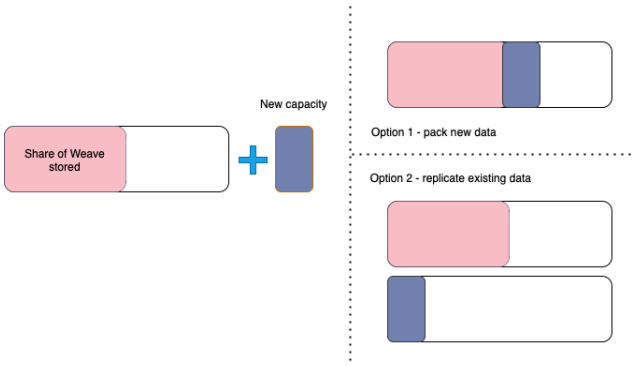

图 10: 矿工被激励去组织成一个完整的副本（选项 1），而不是制作他们已经拥有的数据的额外副本（选项 2）

- 当矿工存储一半编织网络（Weave）时，例如，当 $n = \frac{1}{2}m$，$rmpe$ 是 2。这表示寻找新数据的矿工收益是复制现有数据收益的 2 倍。

- 对于较低的 n 值，$rmpe$ 值趋向于但总是大于 1。这种配置对矿工施加了最大的性能惩罚。

随着网络的增长 ($m \to \infty$)，矿工组织成完整副本的动力将会增强。这促进了合作挖矿小组的创建，这些小组共同存储至少一个数据集的完整副本。

4.2 网络参数

通过使用前一节描述的方程和网络提供的部分信息，我们能够计算出网络正在存储的副本数。每当矿工产生一个新区块时，我们能够判断该区块的解决方案哈希是基于第一范围还是第二范围 SPoA 证明生成的。在一个仅存储完整副本的网络中，这个比例预期为 1:1。但是，如果矿工挖掘的是不完全的分区或者是重复的分区（从而受到效率上的惩罚），那么这个比例会小于 1。

我们可以通过计算观察到的 SPoA 的比例来计算每个分区的平均哈希值。假设在过去的 1,000 个区块中，有 n_1 个第一范围 SPoA 和 n_2 个第二范围 SPoA。这意味着平均副本完整性是 $\frac{n_2}{n_1}$，因此，每个分区的挖矿效率为：

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

使用上述表达式，我们可以准确估计网络中分区的总数。当难度参数为 d 时，尝试的哈希数量的预期值由以下给出：

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

当每个分区的效率仅为 e_m 时，在 120 秒的时间内生成这么多尝试所需的预期分区数量为：

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

考虑到一个分区的大小为 3.6 TiB，我们可以推导出网络的部署存储容量：

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

有关存储数据集和平均副本完整性的所有这些指标都可以从网络中观察到的值计算出来，无需额外的协调成本。

4.3 优化数据路由的激励

激励矿工去构建完整副本来提高挖矿效率将带来一系列正向的激励机制。这些机制中的一个显著例子是，为了在点对点网络中快速传输数据，能促使矿工开发出优化的数据路由解决方案，这是解决去中心化系统中复杂且关键挑战的驱动力 [21, 18, 29]。因为节点必须能够快速地将网络中的任何数据块传输到任何其他块的位置，作为挖矿过程的一部分，这就要求维持可以重复使用的路由能力，以便用户和其他矿工方便地访问数据。

对矿工来说，这个优化数据路由的新激励可能会催生出一个竞争环境，这与比特币矿工竞相开发更高效率的哈希硬件矿机的情况类似。这种竞争将促进路由基础设施的创新，最终促成一个更加高效和强大的分布式网络。

4.4 带宽共享激励

Arweave 挖矿激励对存储复制的另一个衍生效应是，矿工获取网络中数据的内在必要性。这创造了多种数据访问的市场结构，包括：

- **Karma 和乐观的互惠原则：** Arweave 网络中的一些节点参与了一个类似于 BitTorrent 的带宽共享游戏 [11]。在这个游戏中，节点与其他共享数据的矿工互相共享数据。此外，节点偶尔也会随机共享数据，乐观地期待未来的回报。每个节点维护自己的对等方排名，无需报告这些排名是如何或为何确定的。这样的机制在 BitTorrent 这样曾一度占据全球约 27% 互联网流量 [20] 的数据共享平台中获得了非常显著的成功。

- **物理磁盘分发的收益：** 节点运营商可以直接购买或出售存储了编织网络（weave）数据的物理磁盘，以换取

金钱或其他形式的支付。对于带宽受限的矿工来说，鉴于运行 Arweave 节点所需的大量数据，这可能是一个更可取的选项。这种传输方式绕过了传统的数据包过滤器和防火墙。

- **支付协议：**节点还可以参与允许他们在访问数据时进行支付的协议和市场。Permaweb Payment Protocol（P3）[40] 就提供了这样一种方式，它使用支付通道激励 Arweave 内的多种服务（包括简单数据访问）。

4.5 可扩展性

Arweave 创建区块的平均时间为 2 分钟左右，每个区块包含了最多 1,000 笔交易。这一限制确保了区块的验证和同步保持在极为轻量的程度，让整个网络能够广泛地去中心化。但得益于网络的“捆绑（bundling）”机制 [1]，这个交易笔数的限制并不意味着会对存储在给定区块中的数据大小或数量施加任何限制。捆绑是建立在核心协议之上的全网标准，用于将许多不同的数据条目合并到单个交易中。这些数据条目在功能上等同于网络上的顶级数据存储交易，因为在检索时，捆绑的交易可以被“解包”成其组成项。

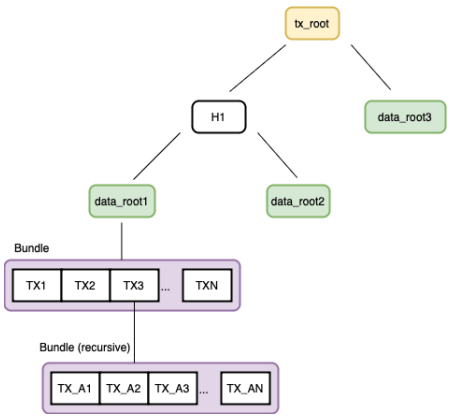

图 11: 捆绑允许将数据上传递归地堆叠到一个顶级交易中。

Arweave 的最大交易大小为 $2^{256}-1$ 字节，这可以在潜在的递归打包中被划分为任意数量的单独数据条目。这允许网络的吞吐量在没有实际适用限制的情况下进行扩展。这种优化是可能的，因为在 Arweave 上的数据上传没有参数化——网络上的每个字节都是同一个全球性默克尔化数据集的一部分，并且由一个共享的基金（endowment）支持。这个设计中的一个要素是从单个数据项到上传捆绑包的支付聚合。用户可以选择在一个捆绑交易中合并他们的数据项支付，或者将支付完全转移到链下，由一个捆绑数据服务提供方将其数据项与其他用户的数据项合并。

在 Arweave 中，所有交易根据其总价值被选中包含在每个区块的 1,000 个槽中，因为矿工赚取的包含费与交易费成正比。这在区块空间稀缺的情况下，激励了打包服务以递归的方式组合交易，增加了网络的可扩展性。因此，不管多少数量的捆绑数据者和用户都可以在任何给定时间内向网络写入数据，而不会导致如其他区块链那样的区块空间竞拍机制。此外，捆绑数据者之间为构建更大的交易而进行的竞争，将对用户最终的费用成本产生向下压力。这与其他区块链形成鲜明对比，那里因为对有限的区块空间的竞争非常激烈，导致用户需要支付的费用不断增加，最终使得部分用户因费用太高而被迫停止使用该网络。

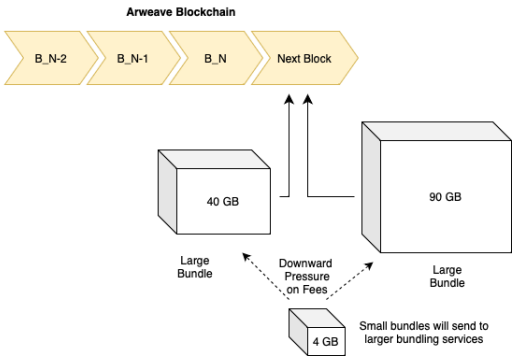

图 12: 对更大数据包的偏好激励了捆绑者对数据进行递归打包，以最小化费用支出。

将数据上传的交易协商转移到链下捆绑数据服务的另一个结果是，用户可以通过捆绑数据服务商支持的任何支付方式为 Arweave 存储支付，而捆绑数据者则用 AR 来结算已分组的数据。截至目前，通过捆绑服务，Arweave 网络支持至少 18 种不同的支付方式。[14, 37]

5 存储基金

Arweave 的激励模型要求数据上传者支付一小笔交易部署费，并向网络的存储基金（*storage endowment*）以 AR 来提供预先的贡献。这个基金充当了一个支付水龙头，随着时间的推移，当矿工共同提供数据集的复制证明时，他们将通过这个水龙头获得支付。随着存储成本的下降，维护一条数据所需从基金中获得支付的金额也会随之减少。

5.1 存储价格

用户提前为存储 20 份副本 200 年而支付的费用，是基于当前的成本计算的。该协议使用了一个无需信任的机制来确定从矿工获取存储空间的价格，具体方法将在后文中说明。本节我们将逐步介绍协议计算存储价格的完整过程。

在具有特定难度 d_B 的单个区块 B 的时期内，网络中的分区估计数量由 §4.2 节中的公式 (3) 计算得出：

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

将这个表达式乘以分区（Partition）大小，可以计算出在区块 B 时刻网络中当前的总存储量：

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

作为奖励发放给矿工的 AR 数量和区块的难度可以用来估计存储获取成本——在区块 B 时刻，为 1 GiB 服务 1 分钟的价格：

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

其中：

$P_m^*(B)$ = 在区块 B 时刻，存储 1 GiB 数据 1 分钟的估计成本。

r_B = 区块 B 的所有回报。

使用单个区块周期来估计存储价格会有很高的变异性，这是由于收集到的交易部署费和难度调整算法之间的差异所致。因此，在实际操作中，网络会记录大量区块上的难度和释放的奖励。这些记录被网络用来准确计算在区块前 6 周内从矿工那里获得的存储获取成本：

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

其中：

$P_m(B)$ = 在 6 周时间内计算的 1 GiB 1 分钟的平均存储获取成本。

h_B = 区块 B 的高度。

$n = 6$ 周内区块的数量（$30 * 24 * 7 * 6 = 30,240$ ）。

利用这些计算，网络可以准确估算出一个区块周期（$\sim$ 2 分钟）内 1 GiB 存储的获取成本：

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

根据这个公式，协议计算任何数据 D 的 20 份副本在 200 年内的当前价格如下：

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

这是作为预先贡献向用户收取的价格，用于存储基金。随着时间的推移，矿工在证明了他们存储了网络数据集后，将从基金中获得支付，支付计算如下：

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

其中：

$r_e(B)$ = 在区块 B 从基金中的提取金额。

$r_i(B)$ = 在区块 B 中释放的通货膨胀奖励。

$r_f(B)$ = 在区块 B 中接受的交易的交易部署费。

$P_B(B)$ = 在区块 B 时刻，一个区块周期存储 1 GiB 数据的估计成本。

W = 在区块 B 时刻，存储在 Arweave 上的所有数据集。

5.2 存储效率的竞争

根据上述存储定价机制，Arweave 的矿工被激励去竞争以提高挖矿的效率。这与比特币网络中的竞争有类似的效果，那里的矿工通过更加优化的软件与更强性能的硬件来竞争最小化计算哈希的成本。使用 Arweave 存储定价机制的一个结果，是创建了一个高效的存储获取市场，它无需依赖于可能会限制协议可扩展性的参数化 “链上订单簿”。

5.3 通货紧缩与基金价值

每当数据被上传时，Arweave 网络的基金（endowment）就会将流通中相应数量的代币，移至这个用于支付随时间而累积的数据存储费用的基金中。基金的存储购买力是有弹性的，它随着提交的数据量、数据存储成本和代币价值的变化而变化。基金价值变化的主要驱动因素之一是，存储成本的降低会导致存储购买力相应的比例增加，从而导致未来需要从基金中释放的代币数量减少。我们将固定时间段中存储一单位数据的总成本下降速率称为 *Kryder+* 。这个速率包括硬件价格、电力成本和数据存储相关运营成本的变化。

用户支付当前价格下 200 年的复制存储费用，这样只需 0.5% 的 *Kryder+* 速率就足以在没有代币价格变化的情况下无限期地维持基金。在这些条件下，基金每年末的存储购买力将等于年初的购买力。过去 50 年里，存储

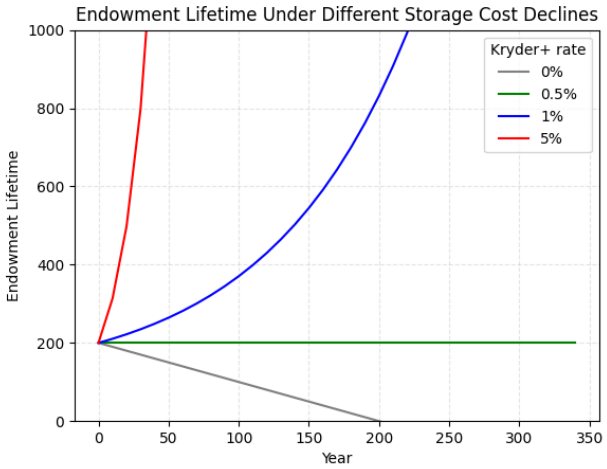

图 13: 基金的健康状况受到协议化 *Kryder+* 速率（0.5%）与实际 *Kryder+* 速率之间的差异以及代币价格变化的影响。

成本的实际下降速率平均为每年 $\approx 38.5\%$ [16]。我们注意到，这一模式延续了历史上可见的信息编码和保存的实际成本逐渐减少的趋势。此外，鉴于存在强烈的激励措施和显著的改进空间，这一趋势很可能会持续下去。观测到的与协议规定的 *Kryder+* 速率之间的差值被调整以提供代币价格波动的广阔安全边际，任何过剩将随时间导致代币供应的通货紧缩。

6 去中心化内容政策

Arweave 网络采用了一个没有中心化控制与审核的去中心化分层内容政策系统。这个系统的基本原则是自愿性：每个参与者都可以自由选择他们想要存储和提供的数据，协议不设任何强制要求。这一系统允许网络中的每个参与者创建并运营自己的内容政策，无需其他人的共识。这一做法的结果是形成了内容政策的多样化选择，具体体现在三个不同层面：

- **矿工：**网络中的矿工可以对他们存储的数据运行任意计算（包括各种形式的文本、图像、视频等分析），以筛选出他们认为非法或不当的内容。由于矿工存储并公开提供他们的内容，他们受到所在国家或地区法律和规定的约束。这使他们避免存储不符合当地法规的非法内容。

- **网关：**用户通常通过网关 [30] 访问 Arweave 上的内容。网关充当门户，允许用户和开发者在不运行自己节点的情况下访问 Arweave 网络中的数据。就像矿工可以选择自己对存储数据的内容政策一样，网关也能独立决定它们索引和提供哪些内容。此外，网络中网关的互操作性允许用户选择符合个人信仰和价值观的网关。

- **应用程序：**可能影响 Arweave 用户的最后一层内容审核在应用程序层面。基于 Arweave 数据构建的每个应用程序可能会在其接口提供的内容上使用额外的过滤器，这取决于开发者的代码实现。这些应用程序层面的内容政策可以嵌入到应用程序本身的源代码中，并不可变地存储在 Arweave 上——这使用户能够永久信赖应用程序将如何进行内容审核 [26]。

7 协议进化

随着时间的推移，围绕 Arweave 网络的环境必然会发生变化。真正的永久存储需要一个能够适应这些变化的系统。我们发现现有的区块链协议治理方法未能同时实现协议的适应性和用户保障的维护 [39]。为了解决这些问题，Arweave 网络中部署的治理机制在传统的区块链分叉模型的基础上进行了改进，提供了一个激励协议进化的框架。Arweave 网络的治理程序完全是宪法性质的。它们是社区参与者之间达成一致的社会协议，允许技术实现根据需要而变化。完整的框架和它引用的原则可以在 Arweave 网络本身找到 [32, 33]。在本节我们将描述这一机制的操作及其产生的激励。

7.1 机制概览

该框架的核心是一个系统，它通过在 Arweave 协议上进行分叉来激活和奖励无需许可的创新。这种机制促成了一种进化形式：为生成、测试和挑选出更优的协议改进版本创造了强大的动力。通过允许任何人提出带有奖励的协议改进版本，形成了一个针对改进的市场。大体上，这个机制功能如下：

1. 创新者选择基于先前版本的协议来开展他们的新工作。他们基于对其采纳率和适用性的认识来选择他们创新的父版本。

2. 然后，创新者创建一个带有新特性的协议改进版本，并将其提供给社区，铸造他们认为代表对其贡献合理奖励的新代币数量。

3. 最后，社区成员评估这个新的改进版本，如果它在合理稀释的情况下提供了足够的改进，则选择使用它而

非其他选择。

这一过程造就了一个随环境演变而进化的协议，它在最大化自身的实用性和稳健性——即“适应性”——的同时，尽量减少了稀释。据此可见，要取得成功，创新者需要同时满足协议用户和其他创新者——两大主要市场参与群体的需求。一旦实现这一目标，他们将因为这些贡献获得市场所认定的奖赏。

7.2 合作动机

这种无需许可的机制鼓励所有市场参与者合作，而不是与协议的早期版本竞争。具体来说，有三大理由激励了任何试图构建永久数据存储系统的人参与这一机制：

- 忠实地参与这一机制能够让他们通过一个已激活且参与度高的用户基础来轻松推动改进版本的采纳。这最大化了他们的创新奖励，同时也最小化了成本开销。
- 通过忠实地参与机制，创新者可以期待他们的改进版本将成为永久数据存储系统血统的一部分。因此，其他人将推进他们的协议（他们的数据和代币）向前发展。
- 任何试图创建永久信息存储系统的构建者都需要一个灵活且可适应的机制，以应对一个肯定会随时间变化的环境。Arweave 的进化框架为这个问题提供了一个所有人都可以无成本或限制地参与的解决方案。

鉴于创新者所面临的真实成本极低——他们可以根据自己认为合适的程度进行稀释——参与这种机制比自行创建一个竞争性服务更为理想。

7.3 数据集统一的激励

创新者还受到另一个强力的激励：通过合并分叉谱系的数据来最大化他们新进化版本的价值。这种激励来源于一种现象：一个统一的协议通常比仅是某个分叉谱系的演化版本更具价值。达到这种统一的基本策略包括两个要素：解决不同社区的技术问题，并将分支的数据纳入新的演化中。如果正确执行，这种统一策略几乎没有理由让社区成员继续参与分叉，鼓励他们将他们的使用和代币持有迁移到新的进化。

如果以下表达式的结果为正，创新者就有动力在演化树上合并不同分叉的数据：

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

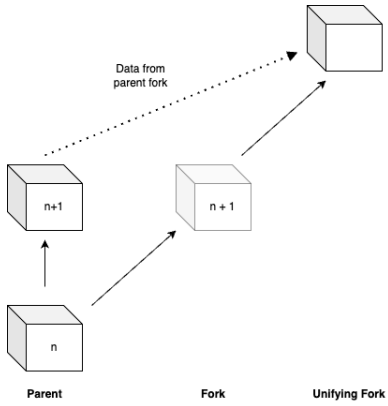

图 14: 来自分叉的数据可以被上传到协议接受的演化版本中。

其中：

V = 涵盖来自分叉的数据所导致的净价值增益或损失。

F_M = 分叉的市场资本化。

C_D = 如果涵盖了旧数据，将会有多少市场资本化比例从旧分叉转移到新分叉。

C_N = 不管是否涵盖数据，市场资本化的比例将会从旧分叉转移到新分叉。

D_C = 复制旧数据到新演化版本所需的成本——以必要的稀释为代价。

D_V = 在新演化中包含来自分叉的数据的内在社会价值。

鉴于这个方程中的 D_C 项通常与 F_M 相比较小，创新者通常会被激励去包含数据并统一树上最突出的分叉。因此，只要用户上传到具有适当突出性和使用量的网络中，他们就可以有信心他们的数据将被包含在系统的权威版本中。

综合评估这些激励的效果是，代币持有者将在不同演化版本中积累一系列资产，而 Arweave 用户可以期待他们的数据会被一个不断改进的协议集合维护，随着网络的演化而演化。为了更深入分析框架的各个组成部分，我们推荐读者参考《演化 Arweave 框架》的伴随文档 [36]。

8 总结

在这篇论文中，我们介绍了 Arweave 协议：一个永久信息存储系统，它允许随时间和空间进行去中心化和点

对点通信。我们展示了这一协议的技术和经济机制，并探索了它们当前在网络中部署的涌现动态。作为其组成部分，还描述了一个新颖的演化系统，将允许协议随着时间变化而适应不同环境。

9 鸣谢

Arweave 是成千上万名贡献者和团队共同努力的结果，他们在网络及其生态系统发展的各个阶段建立了基础设施、服务和产品。Arweave 协议还拥有一个由数十万人组成的活跃社区，支持网络的成长。作者们想要感谢他们所有人，感谢他们帮助实现了这个网络及其使命。

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.