面向动态物联网网络的无速率编码区块链

Changlin Yang，电气电子工程师学会会员，Alexei Ashikhmin，电气电子工程师学会会士，Xiaodong Wang，电气电子工程师学会会士，Zibin Zheng，电气电子工程师学会会士

arXiv:2305.03895v1 [cs.DC] 6 May 2023

摘要——限制区块链在物联网（IoT）中应用的一个关键因素是其巨大的存储需求，这是因为每个区块链节点都必须存储完整的区块链。这不仅增加了区块链节点的负担，还增加了新节点加入网络时的通信开销，因为它们必须复制整个区块链。为了在不损害系统安全性和完整性的前提下降低存储需求，近期已有研究提出了基于固定码率和固定长度纠错码的编码区块链。然而，这种方法并不适用于节点频繁离开和加入的动态物联网网络。在此类动态区块链中，现有的编码区块链方法会导致新加入节点产生较高的通信开销，并且可能具有较高的解码失败概率。本文提出了一种无速率编码区块链，其编码参数可根据网络状况进行调整。我们的目标是在满足目标解码失败概率约束的前提下，最大限度地降低每个区块链节点的存储需求以及每个新加入节点的通信开销。我们基于真实世界的比特币区块链对所提出的方案进行了评估，结果表明，在最大解码失败概率为 10^{-12} 的情况下，存储开销和通信开销均降低了 99.6%。

索引词——编码区块链、动态物联网网络、存储缩减、解码失败概率、无速率码。

I. 引言

区块链技术凭借其分布式、安全和集成等特性，已在物联网（IoT） [1] 中得到广泛应用 [2]。限制区块链在物联网设备中应用的一项关键挑战，是用户需要承担高昂的存储需求 [3]。例如，2022年[4] 的比特币区块链需要占用 380 GB 的硬盘空间，而要将其下载到一个功能完备的 Bitcoin Core 客户端则需要数天时间 [5]。因此，智能手机和笔记本电脑等便携式设备难以作为区块链中的全节点，这削弱了区块链的分布式特性。

减少区块链存储需求 [6]的方法有几种。第一种是简化支付验证（SPV）客户端，即轻节点 [7]。轻节点只存储区块头，并依靠全节点验证交易。因此，如果它依赖于某些恶意节点 [2]，就可能遭到破坏。第二种方法是剪除验证新区块时不需要的旧信息，即使用比特币时，可以删除已花费的未花费交易输出（UTXO） [5]。这能减少所需的存储空间，同时保留“挖掘”新区块的能力。文献[8] 中的工作进一步降低了剪枝节点的存储需求，只存储其持有的 UTXO 的证明。然而，如果所有节点都被剪枝，最终将无法访问已删除的信息。减少存储的最后一种方法是将区块链节点“分片” [9] [10] 为多个组。每组节点存储并运行整个区块链的一部分，从而减少每个节点的存储需求。不过，“分片”方法的安全级别与组的规模直接相关 [11]。因此，这些方案都会以牺牲区块链安全性和／或完整性为代价。

最近，人们提出了编码区块链，以在不损失安全性和完整性的情况下减少区块链的存储需求 [12] [13] [14] [15] [16]。它们还能提高区块传播效率 [17] [18] [19] [20]。除了降低存储和传输成本，编码区块链还可以通过欺诈证明 [21] [22] 或编码默克尔树 [23] [24] [25], 提升现有 SPV 客户端的安全性，并通过编码分片 [26][27] [28], 等方式提升分片解决方案的安全性。在这类区块链中，节点使用固定码率纠错码对数据块数据进行编码，例如 Reed-Solomon 码 [29], 、低密度奇偶校验码（LDPC）[30] 和极化码 [31], ，并分布式存储编码区块。具体而言，将一组 k 个数据块通过一种 $[n, k]$ 码编码为n 个编码区块。随后，节点的存储量可降至复制式区块链的大约 $1/k$ 。当节点需要某个特定数据块的数据时，可以从其他节点收集略多于 k 个编码区块，并运行纠错码解码器来恢复该数据块。节点也可以仅对挖矿不再需要的旧区块进行编码，这与旧区块剪枝方法相似，但不会损失任何安全性和完整性。

实际上，物联网区块链网络是动态的，节点会频繁加入或离开网络。尽管在大多数时间里，加入和离开网络的节点数量保持稳定，但有时也可能发生剧烈变化。例如，2020年9月，比特币节点的数量下降了24%[32]。 [12] - [28] 中的编码方案是为节点数量固定的网络而设计的，但其中一些节点可能会暂时不可用。由于以下原因，这些编码方案并不适用于动态网络。

- 在动态网络中，节点数量可能会迅速变化。因此，使用固定长度的 $[n, k]$ 码 n

C. Yang 和 Z. Zheng 任职于中国广东省珠海市 519000 的中山大学软件工程学院。（电子邮箱：{yangchlin6, zhzibin}@mail.sysu.edu.cn）。A. Ashikhmin 任职于美国新泽西州默里山的诺基亚贝尔实验室通信与统计科学研究部（电子邮箱：alexei.ashikhmin@nokia-bell-labs.com）。X. Wang 任职于美国纽约州纽约市的哥伦比亚大学电气工程系（电子邮箱：wangx@ee.columbia.edu）。

会非常不方便。相比之下，无速率码是解决这种情况的自然方案。

- 动态网络中的节点通常会永久离开网络。因此，如果我们使用固定长度的 $[n,k]$ 码，而在编码时可用的许多 n 节点随后永久离开，那么其他节点将无法重构这些 k 区块。
- 在动态网络中使用固定长度码的一种可能方式，是使用长度N 显著大于网络节点数量n的 $[N,k]$ 码。因此，当一个新节点（例如节点 $n+1$）加入网络时，它可以计算自己的校验符号。然而，在这种方法中，新节点首先必须从其他节点下载多于 k 个编码块，才能解码 k原始区块，这意味着非常大的通信开销和很高的解码复杂度。在所提出的无速率编码区块链中，不存在这一问题。

无速率Luby变换（LT）码 [33] 被用于对 [34] [35] [36] [37]中的区块链进行编码。此外， [38]中还讨论了一种Raptor码的尝试。然而，这些工作中的参数 k 是固定的，因此这些方案并不适用于动态区块链。此外，其中大多数工作使用非系统式LT码。这迫使新加入节点首先解码所有先前生成的码字。由于这些码字包含几乎整个区块链，其数量可能非常庞大，而即使解码单个码字也需要显著的计算资源。因此，这种方法导致新节点需要非常多的计算资源和通信资源。

在这项工作中，我们使用无速率Raptor码 [39]。与 LT 码 [39] 相比，这些编码的解码失败概率更低。我们采用系统无码率码，使原始区块也存储在网络中的某些节点上。因此，当节点加入网络时，只需收集少量原始区块，并为每组先前编码的区块编码其自身的奇偶校验块（或原始块）。我们根据网络状况调整参数 k，从而在存储效率与区块链可靠性之间找到良好的权衡。我们提出使用新颖的增强区块来存储每组编码区块的编码参数。我们在比特币网络中评估了所提出的无速率编码区块链。结果表明，在确保解码失败概率低于 10^{-12} 的情况下，与复制式区块链相比，新加入节点所需的节点存储和通信开销均降至 0.4%。

本文探讨了以往编码区块链研究尚未考虑的一项关键挑战：当网络中的节点数量发生变化时，如何确保成功解码。具体而言，所有现有的编码区块链研究都使用预先确定的信息符号长度 k。如果参与某个特定码字编码的节点中有足够多的节点最终离开网络——而这确实很有可能发生——那么就无法恢复这些 k 信息符号。因此，许多区块将会丢失，而这在大多数区块链应用中都是不可接受的。现有研究没有更新 k的机制。为此，本文提出一种新方法，用于更新 k、为新加入节点提供编码流程，并采用增强块结构，以确保编码参数在网络中的所有节点之间达成共识。据我们所知，迄今为止，文献尚未探讨过这些重要问题。

本文其余部分安排如下。第 II 节介绍编码区块链和无速率码的背景。第 III 节概述所提出的无速率编码区块链。第 IV 节介绍我们提出的新型增强区块的结构。第 V 节和第 VI 节分别介绍新节点加入网络的协议以及我们选择参数 k 的算法。仿真结果见第 VII 节。最后，第 VIII 节总结本文。

II. 编码区块链和无速率码背景

A. 编码区块链背景介绍

在本小节中，我们简要介绍编码区块链在 [13] 和 [14] 中的组织方式。随后，我们将在本文中针对使用无速率码的情况对这一方法进行推广。

设 W 为最后一个数据块的索引， B^W 为该数据块的数据（即不含区块头的第 W 个数据块）。稍微滥用一下记法，以下我们将用数据块同时表示整个数据块（数据 + 区块头）及其数据部分 B^w。设 α 为一个足够大的整数，使得以较高概率，数据块 0、1、.....、$W-\alpha$ 不会包含可能扩展的分叉。这些数据块称为已确认区块。在编码区块链中，已确认区块被划分为 M组，每组都已通过固定速率码进行编码，如图 1 所示。每组包含 k 个数据块，每个数据块 B^w 被划分为有限域$\mathbb{F}_q$上的 s 个符号，记为 $\mathbf{b}_w$。假设第 m 组包含索引为 $\{w_1, w_2, \ldots, w_k\}$的数据块，则该组可表示为一个具有 k 列的矩阵，记为 $\mathbf{b}_{w_i}$，每一列构成该组中对应的原始块：

$$\begin{bmatrix} b_{w_1,1} & b_{w_2,1} & \ldots & b_{w_k,1} \\ b_{w_1,2} & b_{w_2,2} & \ldots & b_{w_k,2} \\ \vdots & \vdots & \vdots & \vdots \\ b_{w_1,s} & b_{w_2,s} & \ldots & b_{w_k,s} \end{bmatrix}. \tag{1}$$

(1)中每一行的 k 项均为信息符号。因此，使用$[n,k]_q$ 上的线性码将(1)中的每一行编码为 n 个编码符号，其中 $q=2^p$ 和 p 表示每个符号中的比特数$b_{w_i,\ j}$，并配有 $k\times n$ 生成矩阵 G。码长 n 通常等于编码时网络中的节点数。编码通过将(1)中的每一行与 G 相乘来完成。由此可得到向量

$$[u_{1,h}, \ldots, u_{n,h}] = [b_{w_1,h}, \ldots b_{w_k,h}]\mathbf{G}. \tag{2}$$

因此，对式（1）的所有行进行编码后，我们得到矩阵

$$\begin{bmatrix} u_{1,1} & u_{2,1} & \ldots & u_{k,1} & \ldots & u_{n,1} \\ u_{1,2} & u_{2,2} & \ldots & u_{k,2} & \ldots & u_{n,2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_{1,s} & u_{2,s} & \ldots & u_{k,s} & \ldots & u_{n,s} \end{bmatrix}. \tag{3}$$

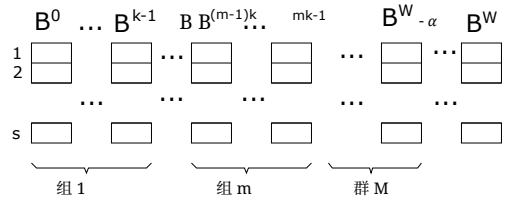


图 1. 区块由组进行编码。每组包含 k 个数据块。每个数据块分为 s 个符号。

每个节点仅存储式（3）的一列，这大大减少了所需的存储空间。我们将式（3）的第 i 列记为 $\mathbf{u}_i$。

使用系统生成矩阵 G 很方便，这样 G 的前 k 列构成单位矩阵 $\mathbf{I}_k$。在这种情况下，我们有 $\mathbf{u}_i = \mathbf{b}_{w_i}, i = 1, ..., k$。如果某个节点（例如节点 j）需要数据块 $\mathbf{u}_i$，例如用于验证交易，它会广播请求并获取 $\mathbf{u}_i$。如果存储 $\mathbf{u}_i$ 的节点不可用，那么节点 j 会收集式 (3) 的其他几列，并使用 $[n, k]_q$ 码的解码器修复 $\mathbf{u}_i$ [14]。另一方面，当在 [14] 中修复失败，或者 G 是非系统式生成矩阵时，如在 [13], 中，节点 j 需要从其他节点收集式 (3) 的 $(1+\epsilon)k$ 列，并解码所有信息符号 $\mathbf{b}_{w_1} = \mathbf{u}_1, ..., \mathbf{b}_{w_k} = \mathbf{u}_k$。这里 ϵ 是解码开销，不同编码的解码开销各不相同。

B. 无码率码的背景

我们现在简要回顾无速率码及其变体。使用无速率码时，发送端利用 n 信息符号生成数量可能无限的编码符号，并将其发送给一个或多个接收器。接收器一旦收集到任意 $(1+\epsilon)n$ 个编码符号，就能以很高的成功概率解码出 n 信息符号。

1) LT 码 *[33]*：设 $\{u_1, \ldots, u_n\}$ 为 n 个信息符号，取自 $\mathbb{F}_q$，$q = 2^p$。我们使用与上一小节中编码符号相同的记号u_i，是因为稍后我们将结合线性 $[k, n]$ 码和 $[N, n]$ LT 码，为我们的区块链构造一个 Raptor码。此类符号的求和 $\oplus$ 通过按位异或运算完成。注意，由于 $q = 2^p$，减法 $\ominus$与 $\oplus$一致。使用 LT 码时，每个编码符号 v_j 的生成过程如下：首先，从生成器度分布 $\Omega(d)$ 中选择一个度 d 。然后，从 $\{u_1, \ldots, u_n\}$ 中以相同概率随机选择d 个信息符号 $\{u_{i_1}, \ldots, u_{i_d}\}$。这些 d 个信息符号是 v_j的邻居。我们用以下符号表示这些邻居的索引集合：

$$\mathcal{N}(v_j) = \{i_1, \ldots, i_d\}. \tag{4}$$

最后，通过异或运算计算编码符号

$$v_j = \sum_{i \in \mathcal{N}(v_j)} u_i = u_{i_1} \oplus \ldots \oplus u_{i_d}. \tag{5}$$

编码符号 v_j 是信息符号$\{u_{i_1}, \ldots, u_{i_d}\}$的边。 u_i的所有边组成的集合记为

$$\mathcal{E}(u_i) = \{j : i \in \mathcal{N}(v_j)\}. \tag{6}$$

信息符号 u_i 可通过一个边编码符号及其其他相邻信息符号计算得出，如下所示

$$u_i = v_j \oplus \sum_{h \in \mathcal{N}(v_j) \backslash \{i\}} u_h, \text{ for any } j \in \mathcal{E}(u_i). \tag{7}$$

图 2 展示了将四个信息符号编码为五个 LT 编码符号的示例。在此示例中，我们有$\mathcal{N}(v_1) = \{u_1\}, \mathcal{N}(v_2) = \{u_1, u_3\}, \mathcal{N}(v_3) = \{u_2, u_3, u_4\}$ 等等；以及 $\mathcal{E}(u_1) = \{v_1\}, \mathcal{E}(u_2) = \{v_3, v_4\}$，等等。

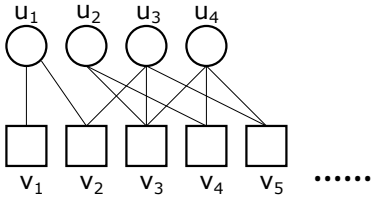


图 2. LT 编码：4 个信息符号（圆形）分别以度数 $d = 1$、2、3、2、2 编码为 5 个编码符号（正方形）。

接收器收到 $(1+\epsilon)n$个编码符号后即可开始解码。LT 码的解码是一个“剥离”过程。首先，解码器找到一个度为 1 的编码符号，即 v_j，其中有 $|\mathcal{N}(v_j)| = 1$。事实上，这个编码符号是索引为 $\{i\} = \mathcal{N}(v_j)$ 的信息符号。解码出信息符号 u_i 后，解码器会从 $\mathcal{E}(u_i)$ 中所有编码符号的邻居集合中移除 u_i，更新这些编码符号，并将其度数也减 1。例如，如果 $j' \in \mathcal{E}(u_i)$，那么 $v_{j'}$ 及其邻居集合将按如下方式更新

$$v_{j'} \leftarrow v_{j'} \oplus u_i, \quad \mathcal{N}(v_{j'}) \leftarrow \mathcal{N}(v_{j'}) \backslash \{i\}. \tag{8}$$

解码器持续查找度为 1 的编码符号，并“剥离”信息符号，直到没有度为 1 的编码符号，或所有信息符号均成功解码。如果仍有尚未解码的信息符号，接收器会收集更多编码符号，并重新启动“剥离”过程。例如，如图 3 所示，对于图 2 中的 LT 码，解码器在收到 $\{v_1, v_2, v_3, v_4\}$ 后可以成功解码信息符号 $\{u_1, u_3\}$ 。不过，它还需要再收集一个编码符号 v_5 ，才能成功解码 $\{u_2, u_4\}$。

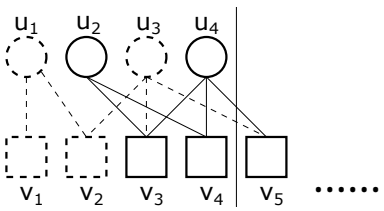


图 3. LT 解码：虚线圆圈表示已解码的信息符号。虚线方框表示度为零的编码符号。虚线表示被减去的度。

LT 码的一个问题是存在误码平层 [40] [41]。误码平层是指随着接收到的编码符号增多，解码失败率并未足够快速地下降。这是因为编码符号的邻居是

从 $\{1,\ldots,n\}$ 中均匀选取的，因此，信息符号有非零概率不会成为任何已接收编码符号的邻居，也就是说，它的边集为空。

2) Raptor码 *[39]*：为了解决误码平层问题并降低 LT 码的编码/解码复杂度，Raptor码首先使用固定码率的外码（如低密度奇偶校验码、Reed-Solomon 码或其他线性码）对信息符号进行预编码。预编码后的符号称为中间符号，随后将其编码为 LT 编码符号。Raptor码的解码失败概率很低，因此非常符合区块链严格的安全要求。此外，Raptor码的编码和解码复杂度为 $\mathcal{O}(k\log(\frac{1}{\epsilon}))$ [39]，这大约是关于 k 的线性函数。下文中，我们将使用 Raptor码构建编码区块链。

III. 所提出的无速率编码区块链概述

我们首先将动态物联网区块链网络定义如下。时间被划分为多个时间周期，$\mathcal{V}_t$ 表示网络在第 t 个时间周期内可用的网络节点集合。网络中的每个节点都可以递归发送“获取地址”消息，以查找网络中所有可达节点并获取 $\mathcal{V}_t$ [42]。此外，只有持有有效 UTXO 的地址才被视为有效，以防止女巫攻击。我们假设区块链以每个时间周期增长 β 个数据块的固定速率。我们进一步假设，在每个时间周期 t，有 l_t 个节点离开网络，e_t 个节点加入网络，其中 l_t 和 e_t 分别服从均值为 λ_l 和 λ_e 的泊松分布。接下来，我们将介绍所提出的无速率编码区块链的编码和解码流程。

A. 编码流程

我们将每个数据块 B^w 划分为 s 个符号（取自 $\mathbb{F}_q$），并按照图 1 的方式组成各组。因此，对于每个组，我们都可以得到一个形式如式 (1) 的矩阵。对于每个组，我们使用相同的 Raptor 码，将式 (1) 的 s 行编码为 s 个码向量。每组中的数据块数量会根据网络状况而变化（详见第 VI 节）。我们用 $\mathcal{G}_m$ 表示第 m组中的数据块集合，并且 $k_m = |\mathcal{G}_m|$。一般而言，如果网络中的活动节点数量较多，且节点不会主动离开网络，则 k_m 较大。我们使用带有生成矩阵 $\mathbf{G}_m$的 $[n_m, k_m]$ 预编码，对每组中的数据块 $\mathcal{G}_m$ 采用系统化 Raptor码 [39] 进行编码。我们假设所有组和时间周期的预编码码率 $r = k_m/n_m$ 相同。所有组的编码和解码流程相同，因此有时会省略下标 m 以简化记号。

由于（1）的每一行以及因此（3）的每一行都采用相同的编码和解码方式，我们将分别用$b_i, i = 1,..., k$和 $u_i, i = 1,..., n$表示（1）和（3）的任意一行。类似地，对于下文给出的矩阵（9），我们将用 $v_i, i = 1, ..., N$表示（9）的任意一行。信息符号 $\{b_i\}$ 首先通过（1）至（3）编码为中间符号 $\{u_i\}$。我们将（1）的每一列（记为 $\mathbf{b}_i$）称为一个原始块；而每一列

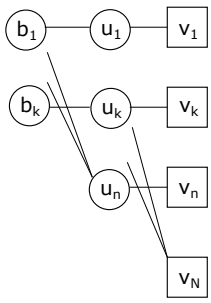


图 4. 使用系统化Raptor码的区块编码图。

（3）的每一列（记为 $\mathbf{u}_i$）称为一个中间块。我们假设 $\mathbf{G}_m$ 为系统式形式，因此$\{\mathbf{u}_1, \ldots, \mathbf{u}_k\} = \{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$。

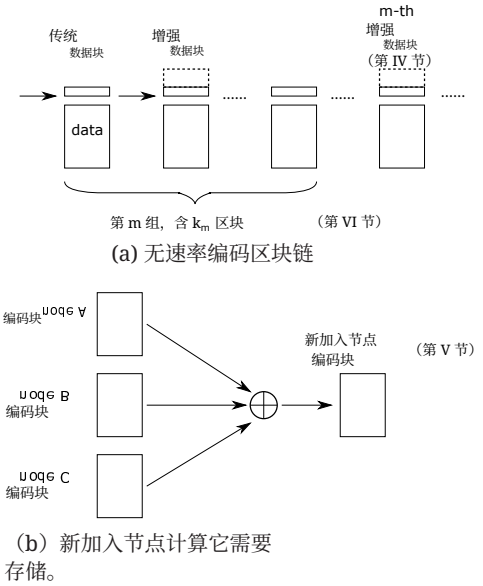
随后，使用度分布为 $\Omega(d)$ 的系统式 LT 码，将式 (3) 每一行的中间符号编码为编码符号。由此得到矩阵

$$\begin{bmatrix} v_{1,1} & v_{2,1} & \cdots & v_{k,1} & \cdots & v_{n,1} & \cdots & v_{N,1} \\ v_{1,2} & v_{2,2} & \cdots & v_{k,2} & \cdots & v_{n,2} & \cdots & v_{N,2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ v_{1,s} & v_{2,s} & \cdots & v_{k,s} & \cdots & v_{n,s} & \cdots & v_{N,s} \end{bmatrix}. \quad (9)$$

这里，N 是编码时网络节点的数量。式 (9) 的每一列 j 都是一个编码块，记为 $\mathbf{v}_j$。我们分别用 $\mathcal{E}(\mathbf{u}_i)$ 表示符号 $\{u_i\}$ 的边集，用 $\mathcal{N}(\mathbf{v}_i)$ 表示符号 $\{v_i\}$ 的邻居集。我们假设采用系统式编码，因此得到系统编码块 $\{\mathbf{v}_1、\ldots、\mathbf{v}_n\} = \{\mathbf{u}_1、\ldots、\mathbf{u}_n\}$，以及非系统编码块 $\{\mathbf{v}_{n+1}、\ldots、\mathbf{v}_N\}$。从 k 原始符号 $\{b_i\}$ 到 n 中间符号$\{u_i\}$，再到 N 编码符号 $\{v_j\}$ 的编码图如图 4 所示。每个节点为每个组仅存储一个编码块。请注意，节点为不同组存储的编码块索引可能不同。例如，某个节点可能为第 m 组存储 $\mathbf{v}_a$，为第 m' 组存储 $\mathbf{v}_b$，其中$a \neq b$。不过，为了简化描述，我们假设所有节点均以整数编号，且节点 j 为所有组 $m = 1、....、M$存储 $\mathbf{v}_j^m$，其中 M是组的总数。在所提出的无速率编码区块链中，我们使用度分布 $\Omega(d)$，其中 $\Omega(1) = 0$。第 VII 节将给出一个示例。

B. 解码过程

当节点 j需要组 m 中的原始数据块 $\mathbf{b}_i$ 时，节点 j 会向网络中的所有节点广播请求。随后，节点 i 会将组 m 中的 $\mathbf{v}_i$（注意，$\mathbf{v}_i = \mathbf{b}_i$）发送给节点 j。然而，区块链网络具有动态性，节点 i 可能会离开网络或断开连接。发生这种情况时，节点 j 可以使用第 V-B 节中介绍的流程来“修复”所需的数据块。简而言之，节点 j 通过从活跃节点获取其他编码块，利用式（7）“修复”所需的数据块。如果修复失败，节点 j会解码组 m中的所有数据块。为此，节点 j 从其他节点收集 $(1+\epsilon)k$ 个互不相同的编码块，这些编码块是式（9）的各列。由于这些非系统编码块



(a) 无速率编码区块链

（b）新加入节点计算它需要存储。

图 5. 所提出的无速率编码区块链概览。

$\{v_{n+1}, \ldots\ldots, v_N\}$ 拥有随机数量的邻居，且邻居是随机选取的，因此以极高的概率，任意 $(1+\epsilon)k$ 个编码块都互不相同。解码器首先按照第 II-B1 节所述进行 LT 解码，以解码中间块。然后，它运行预编码器的解码器，以获得原始数据块。由于式（1）的每一行都是按照预编码的同一生成矩阵和相同的 LT 编码过程编码为式（9）中对应的行，因此，只要能够解码式（1）的第一行，解码器就能成功解码所有原始数据块。

值得注意的是，由于我们使用 LT 码，即使节点数 N 持续增加，所有节点存储不同编码块的概率仍然极高。这降低了多个节点存储相同编码块的可能性。此外，LT 编码过程比为常规（非无速率）线性码设计所需的 $k \times N$ 生成矩阵简单得多，尤其是在 N 规模较大且事先未知时。新节点加入系统后，对于每个组，它平均只需收集 $\mathbb{E}[\Omega(d)]$ 个系统编码块，就能编码出自身要存储的编码块，详情见第 V 节。与 [34]相比，这显著降低了通信开销。码长 k_m 根据网络状况计算得出。如果我们预计网络状况将会恶化（例如，预计许多节点可能离开网络），那么可以使用较小的值 k_m对所有先前编码的组进行解码并进一步重新编码。网络中的所有节点必须就组 m 中包含的数据块以及预编码的生成矩阵 G_m 达成共识。为实现这种共识，我们在第 IV 节提出了一种新颖的增强区块，可轻松嵌入当前区块链中。

所提出的编码区块链概览如图 5(a) 和 5(b) 所示。在图 5(a) 中，展示了构成第 m 组的数据块及其区块头，以及存储第 m 组编码参数的第 m 个增强块。第 m 个增强块的生成时间始终晚于第 m 组中的任何区块。关于增强块的组织方式以及如何确定组大小 k_m 的详细信息，分别见第 IV 节和第 VI 节。在图 5(b) 中，当新节点加入网络时，只需从其他节点收集少量编码块，即可计算出要存储的编码块。新节点加入流程的详细信息将在第 V 节中给出。

备注 1。所提出的编码方案与标准 Raptor 码编码明显不同，例如 [39] 中介绍的方案。在标准场景下，只需生成一个半无限长的校验符号序列，因为系统（信息）符号始终可用，且不会被擦除。然而，在我们的情形中，如果某些系统节点离开网络，系统符号可能会变得不可用。这导致编码流程发生重大变化，其中考虑了某些系统符号可能缺失的情况（见第 V 节）。这也意味着，新加入的节点会按照不同的度分布生成码符号，这与首次对特定码字进行编码时所采用的度分布不同（见引理1）。据我们所知，尚未有人研究过这种工作模式下的 Raptor 码。由于我们的编码方式和实际擦除信道与标准方案有很大不同，标准的码性能分析（见 [39], ）不再适用。因此，我们提出了一种不同的方法来估计码性能，见第 VI 节。

IV. 增强区块

在现有的编码区块链系统中，例如参见[14], ，所有组的 k 值都是固定的。然而，在我们的方案中，组大小会根据网络状况变化，且所有节点必须就特定组所使用的编码参数达成共识。因此，本节介绍一种增强区块，它由一个传统区块和区块头中的若干额外字段组成，用于存储编码参数。增强区块的区块结构、挖矿和验证方式与传统区块略有不同。

A. 增强块结构

增强区块的数据组织方式与常规（非增强）区块相同。然而，与常规区块的区块头相比，增强区块的区块头包含四个额外字段，见图 6。这些新字段为

- 组序号：此字段是增强区块的序列号，独立于区块序号。
- 组索引：此字段包含待编码区块的区块序号，即 $\mathcal{G}_m = \{w_1, \ldots, w_k\}$。请注意，$k_m = |\mathcal{G}_m|$ 和 $n_m = k_m/r$。
- 生成矩阵：此字段包含生成矩阵 G_m。需要注意的是，如果区块链系统使用预定义的生成矩阵计算算法，则可以省略此字段。例如，如果我们使用低密度奇偶校验码对中间块进行编码，则可以通过渐进边增长（PEG）算法生成该编码。

[43] 因此，只需将 k_m 和n_m 的值存储在组索引字段中即可。

- 哈希值：此字段包含非系统式中间块 $\{u_{k+1}$、.....、$u_n\}$ 的哈希值。当新节点加入网络时，需要系统式中间块，有时还需要非系统式中间块，以生成自己的编码块。系统式中间块实际上就是原始区块，其完整性可通过相应区块头中的默克尔树根进行验证。哈希值字段确保新节点也能够验证非系统式中间块的完整性。

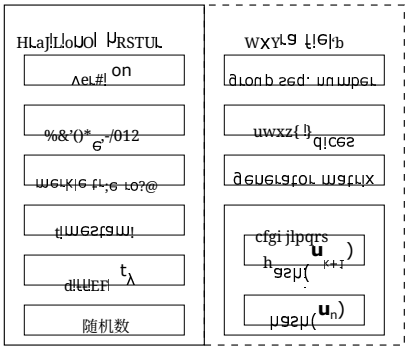


图 6. 增强区块头结构。

B. 挖矿

回顾一下，我们采用工作量证明协议，所有节点通过竞争来“挖掘”新区块。传统区块的“挖矿”过程与现有区块链中的相同。然而，“挖掘”增强区块需要付出更多努力。设 M 为已“挖掘”的增强区块总数。网络中的所有节点使用相同的算法计算 k_{M+1}的值，即下一个组的大小（见第 V 节）。每个节点 j 还维护一个区块池 $\mathcal{P}_j$，其中包含已确认但尚未编码的区块，见图 7。当$|\mathcal{P}_j| \geq k_{M+1}$时，节点 j 开始“挖掘”第（$M+1$）个增强区块。该过程包括以下步骤。

1) “矿工”首先选择要纳入增强区块的交易，并计算默克尔树根。然后填写版本、前一区块哈希、时间戳和难度字段。因此，这部分与“挖掘”传统区块相似。2) “矿工”构造集合 $\mathcal{G}_{M+1} \subset \mathcal{P}_j$ ，其大小为$|\mathcal{G}_{M+1}| = k_{M+1}$。

3) “矿工”随后构造一个 $k_{M+1} \times n_{M+1}$ 生成矩阵 G_{M+1}。

4) 接下来，“矿工”使用（2）计算中间块$\{u_1$，...，$u_{n_{M+1}}\}$及其哈希值。5) 最后，与“挖掘”传统区块类似，“矿工”寻找随机数，使增强区块头的哈希值小于难度级别。

经过上述流程后，“矿工”会生成一个增强区块，其区块头结构如第 IV-A 节所述。我们可以看到，传统区块的挖矿过程中并不包含步骤 2–4。不过，这些步骤的计算复杂度与寻找随机数相比可以忽略不计。因此，挖掘增强区块的激励与挖掘传统区块的激励相同。

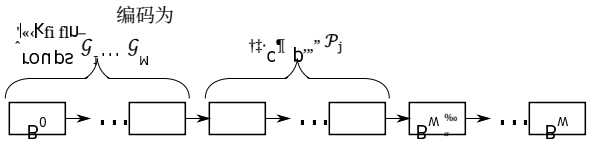


图7. 带有区块池的编码区块链。

C. 验证

当节点收到一个序列号为 $M+1$ 的新“已挖掘”增强区块时，该节点会验证此区块。如果新的增强区块有效，节点便将其附加到自己的链上，并终止其自身对第 $(M+1)$ 个增强区块的“挖矿”过程。否则，它将继续自身的“挖矿”过程。具体而言，节点 j' 的验证应包括但不限于以下步骤：

- 增强区块已正确构建，包含区块数据和区块头结构。此步骤可以包含与传统区块相同的验证过程，例如检查随机数和目标值、各字段的大小、与前一区块的链式关联等。
- 验证节点 j' 的区块池大小大于或等于 k_{M+1}，即，

$$|\mathcal{P}_{j'}| \geq k_{M+1}. \tag{10}$$

- 编码组中的所有区块都位于区块池中，即，

$$\mathcal{G}_{M+1} \subset \mathcal{P}_{j'}. \tag{11}$$

- 使用公式 (2) 和 G_{M+1}计算 $\{u^{k_{M+1}+1}$，...，$u_{n_{M+1}}\}$，并验证其哈希值。

我们回顾一下，挖掘增强区块时，节点需要稍多一些资源，因为它必须找到 k_{M+1} 和 G_{M+1}。虽然这些额外资源远少于寻找随机数所需的资源，但仍有一些节点可能会继续挖掘传统区块，而不是增强区块。为防止这种情况，我们提出采用惩罚机制。例如，节点可以遵循这样的规则：如果某节点的区块池大小满足 (10)，则当且仅当新区块是增强区块时，才接受该区块。

D. 区块池更新

节点 j “已挖掘”或验证了新的第 $(M+1)$ 个增强区块后，其区块池更新为

$$\mathcal{P}_j \leftarrow \mathcal{P}_j \backslash \mathcal{G}_{M+1}. \tag{12}$$

然而，应注意，来自 $\mathcal{G}_{M+1}$ 的区块尚未准备好进行编码，因为第 $(M+1)$ 个增强区块的深度小于 α，即该增强区块尚未确认。区块链可能会分裂成多个分支，并且在不止一个分支中“挖掘”出不同的候选第 $(M+1)$ 个增强区块。在此

在这种情况下，节点 j 将为每个包含候选第 $M+1$ 个增强区块的分支 f 维护区块池 $\mathcal{P}_j^f$。一旦其中一个候选增强区块得到确认，例如在分支 f^* 中，节点 j 就会分配 $\mathcal{P}_j = \mathcal{P}_j^{f^*}$，并丢弃其他分支中的区块池。通过这种方式，节点保留与最长分支相关联的可用区块池。

E. 去中心化系统 Raptor 码编码

在第 m 个增强区块得到确认后，所有节点便按照 III-A节 中所述的过程，以分布式方式开始对 $\mathcal{G}_m$ 中的区块进行编码。需要注意的是，由于网络传播延迟和处理速度各不相同，在给定时刻 t，不同节点 $\hat{}$ 通常具有不同的 W 值。令 $B(m)$ 表示第 m 个增强区块的区块序号。假设任意节点 j 都有 W，使得 $W-\alpha = \hat{}B(m)$。记

$$\mathcal{C}_j^m = \{i : \exists j', \mathcal{N}(\mathbf{v}_{j'}^m) = \{i\}\}, \tag{13}$$

为某个节点在给定时刻 t 所存储的中间块集合。本小节稍后将说明节点 j 如何获取 $\mathcal{C}_j^m$。注意，$\mathcal{C}_j^m = \emptyset$ 是可能的。不同节点对应的集合 $\mathcal{C}_j^m$ 也可能不同。再定义

$$\mathcal{S}_j^m = \{1, \ldots, n\} \backslash \mathcal{C}_j^m, \tag{14}$$

为任何节点都未存储的中间块集合。

一旦 $B(m)$ 时，α 节点 j 便开始为第 m 组编码其自己的编码块，具体如下。

- 节点 j 首先检查 $\mathcal{S}_j^m$。如果 $\mathcal{S}_j^m = \emptyset$，则节点从生成器度分布 $\Omega(d)$ 生成 d_j，以及一组 $\mathcal{N}(\mathbf{v}_j^m)$，并根据 (5) 计算 $\mathbf{v}_j^m$。如果 $\mathcal{S}_j^m \neq \emptyset$，节点获取任意一个 $i_j \in \mathcal{S}_j^m$，将 $\mathcal{N}(\mathbf{v}_j^m) = \{i_j\}$，并在网络中广播消息“我存储中间块 $\mathbf{u}_{i_j}^m$，时间戳为 t_j”。

- 收到此消息后，每个节点，例如节点 j'，都会更新其 $\mathcal{C}_{j'}^m \leftarrow \mathcal{C}_{j'}^m \cup \{i\}$ 和$\mathcal{S}_{j'}^m$。但是，如果节点 j 从节点 j' 接收到带有时间戳 $t_{j'} < t_j$的 $i_{j'} = i_j$，这意味着节点 j' 决定比节点 j更早存储 $\mathbf{u}_{i_j}^m$。在这种情况下，节点 j 会更新 $\mathcal{S}_j^m$和 $\mathcal{C}_j^m$，并从头开始编码过程。

所有网络节点都遵循上述过程。因此，该过程确保每个系统编码块 $\{\mathbf{v}_1^m$、.....、$\mathbf{v}_n^m\} = \{\mathbf{u}_1^m$、.....、$\mathbf{u}_n^m\}$ 均存储在且仅存储在一个节点上。此外，该过程还确保非系统编码块$\{\mathbf{v}_{n+1}^m$、.....、$\mathbf{v}_{n'}^m\}$ 的度分布遵循给定的度分布 $\Omega(d)$。存储系统编码块的某些节点可能会离开网络。此外，恶意节点可能会假装拥有此类系统编码块，却无法将其提供给诚实节点。下一节将介绍一种修复过程，以动态维护系统性。

V. 网络维护算法

本节介绍我们的网络维护算法（NMA）。当新节点 j 加入网络时，它首先尝试通过从网络中的其他节点获取中间块来编码一个新的非系统编码块。如果失败，即网络中没有任何节点存储一个或多个中间块，则节点 j 随后尝试修复并存储缺失的中间块。

A. 非系统编码块的编码

令 $\mathcal{M}$ 为增强块集合，使得所有块组 $\mathcal{G}_m$、$\forall m \in \mathcal{M}$ 均已编码。当新节点 j 加入网络时，它首先从其他节点复制原始未编码块。随后，它为每个 $m \in \mathcal{M}$计算要存储的编码块 $\mathbf{v}_j^m$。具体而言，对于每个组 $m \in \mathcal{M}$，节点 j 从 $\Omega(d)$ 中生成一个度 d_j^m，并构造 $\mathcal{N}(\mathbf{v}_j^m)$。然后，它向全网广播请求，索引范围为 $\mathcal{N}(\mathbf{v}_j^m)$ 的区块。收到此请求后，如果对于任意 (v) 都有 (v)，节点 j'就会向节点 j i $\mathcal{N}_j^m = \{i\}$ $i \in$发送一个索引，以告知该节点中间块 u 。随后，节点 j 将 $\phi(\mathbf{u}_i^m) = 1$ 赋值为表示 $\mathbf{u}_i^m$可用，否则赋值为 $\phi(\mathbf{u}_i^m) = 0$。因此，对于每个组 $m \in \mathcal{M}$，节点 j 能够构造一个由缺失中间块组成的集合 $\mathcal{R}_j^m$

$$\mathcal{R}_j^m = \{i : \phi(\mathbf{u}_i^m) = 0, i \in \mathcal{N}(\mathbf{v}_j^m)\} \tag{15}$$

如果 $\mathcal{R}_j^m = \emptyset$，那么节点 j 可以从其他节点获取 $\mathcal{N}(\mathbf{v}_j^m)$ 的所有区块，然后根据 (5) 计算并存储编码块 $\mathbf{v}_j^m$。否则，如果 $\mathcal{R}_j^m \neq \emptyset$ 是因为存储系统式编码块的节点离开网络，或网络连接不佳所致，节点 j 将尝试修复 $\mathcal{R}_j^m$中的一个数据块。首先，它会尝试修复 $\mathcal{R}_j^m$中的一个数据块。在修复过程中，集合 $\mathcal{R}_j^m$可以扩展为包含其他缺失中间块，具体如下一节所述。

B. 中间块修复

节点 j 随机选择 $i \in \mathcal{R}_j^m$，并尝试使用边编码块 $\mathbf{v}_{j'}^m$ 修复 $\mathbf{u}_i^m$，其中 $j' \in \mathcal{E}(\mathbf{u}_i^m)$，且 $\mathbf{v}_{j'}^m$的其他相邻中间块 $\mathcal{N}(\mathbf{v}_{j'}^m)\backslash\{i\}$ 与 (7) 类似，即

$$\mathbf{u}_i^m = \mathbf{v}_{j'}^m \oplus \sum_{h \in \mathcal{N}(\mathbf{v}_{j'}^m)\backslash\{i\}} \mathbf{u}_h^m, \text{ for any } j' \in \mathcal{E}(\mathbf{u}_i^m). \tag{16}$$

然而，节点 j 并不知道边集 $\mathcal{E}(\mathbf{u}_i^m)$ 以及任意 $j' \in \mathcal{E}(\mathbf{u}_i^m)$ 的邻居集合 $\mathcal{N}(\mathbf{v}_{j'}^m)$。因此，节点 j 尝试构建一个子集 $\hat{}\mathcal{E}(\mathbf{u}_i^m) \subseteq \mathcal{E}(\mathbf{u}_i^m)$。首先，它建立 $\hat{}\mathcal{E}(\mathbf{u}_i^m) = \emptyset$，并将 i 广播到整个网络中。当另一个节点 x 收到此消息时，如果满足条件 $i \in \mathcal{N}(\mathbf{v}_x^m)$，它将向节点 j 回复 $\mathcal{N}(\mathbf{v}_x^m)$。随后，节点 j 将 $\hat{}x$ 添加到集合 $\mathcal{E}(\mathbf{u}_i^m)$ 中。请注意，如果某个节点 $j' \in \mathcal{E}(\mathbf{u}_i^m)$ 的网络连接状况不佳，无法向 j 回复，那么节点 j' 将被视为缺失节点，并且有 $j' \in \mathcal{E}(\mathbf{u}_i^m)\backslash\hat{}\mathcal{E}(\mathbf{u}_i^m)$。在获得集合 $\mathcal{E}(\mathbf{u}_i^m)$ 后，节点 j 会检查每个 $j' \in \hat{}\mathcal{E}(\mathbf{u}_i^m)$ 对应的中间块 $\mathcal{N}(\mathbf{v}_{j'}^m)\backslash\{i\}$ 的可用性。如果存在 $j' \in \hat{}\mathcal{E}(\mathbf{u}_i^m)$，使得来自 $\mathcal{N}(\mathbf{v}_{j'}^m)\backslash\{i\}$ 的所有数据块都.....

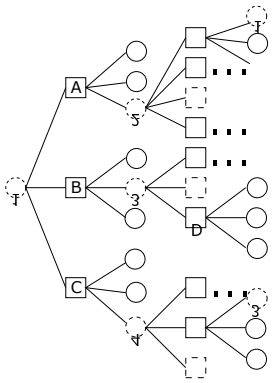


图8. 以 u_1^m 为根节点的扩展Tanner图示例，其中 $d=4$。实线/虚线圆圈分别表示可用/缺失的信息符号，实线/虚线方框分别表示可用/缺失的校验符号。

可用时，可根据 (16) 修复 u_i^m。否则，节点 j 将缺失中间块添加到 $\mathcal{R}_j^m$，即，

$$\mathcal{R}_j^m \leftarrow \mathcal{R}_j^m \cup \{i' : \phi(\mathbf{u}_{i'}^m)=0, i' \in \mathcal{N}(\mathbf{v}_{j'}^m)\backslash\{i\}, j' \in \hat{\mathcal{E}}(\mathbf{u}_i^m)\}, \quad (17)$$

并将 u_i^m 的索引 i 也添加到集合 $\mathcal{Q}_j^m$ 中，表示这些符号不可修复。接下来，节点 j 尝试从 $\mathcal{R}_j^m \backslash \mathcal{Q}_j^m$ 修复一个数据块。

图8通过一个示例说明了中间块修复过程。假设节点 j 生成$\mathcal{N}(\mathbf{v}_j^m)$，并注意到 $\{1\} \in \mathcal{N}(\mathbf{v}_j^m)$，但 u_1^m 不可用，而来自 $\mathcal{N}(\mathbf{v}_j^m)$ 的其他数据块均可用。随后，节点j 构建 $\mathcal{R}_j^m = \{1\}$。接下来，它联系其他节点并ˆ构建集合 $\mathcal{E}(\mathrm{u}_1^m) = \{A, B, C\}$。然而，索引为 $\{2\} \in \mathcal{N}(\mathbf{v}_A^m)$、$\{3\} \in \mathcal{N}(\mathbf{v}_B^m)$ 和 $\{4\} \in \mathcal{N}(\mathbf{v}_C^m)$ 的中间块不可用。因此，节点 j 更新$\mathcal{R}_j^m \leftarrow \{1\} \cup \{2,3,4\}$ 并构建集合 $\mathcal{Q}_j^m = \{1\}$。接下来，节点 j 尝试修复中间块 u_2^m、u_3^m 和 u_4^m 中的任意一个。在此示例中，它成功修复了 u_3^m，并存储编码块 $\mathbf{v}_j^m = \mathrm{u}_3^m$。因此，后续加入节点可以从节点 j 获取 u_3^m，以修复 u_4^m。

如果 $\mathcal{Q}_j^m = \mathcal{R}_j^m$，这意味着节点 j 无法在当前网络中修复一个中间块。然后，节点 j 从$(1+\epsilon)k_m$ 个不同节点收集编码块，并使用 III 节中描述的过程解码原始区块。之后，对于任意$i \in \mathcal{R}_j^m$，它会存储一个中间块 $\mathbf{v}_j^m = \mathrm{u}_i^m$。

需要注意的是，在上述提出的 NMA 算法中，新节点按照分布 $\Omega^*(d) \neq \Omega(d)$ 生成度 d。分布 $\Omega^*(d)$ 取决于从编码组m 时刻到新节点加入时刻的网络变化历史。回想一下，n_m 是在编码组 m 时刻的中间块数量（见 III-A节）。设 n_m^* 为新节点 j 加入网络时，第 m 组中存储唯一中间块的节点数量。则 $\Omega^*(d)$ 由以下引理给出。

引理1。由此产生的实际分布 $\Omega^*(d)$ 来自于 NMA算法是

$$\Omega^*(d) = \begin{cases} \Omega(d) g(n_m^*, d), & \text{for } d \geq 2, \\ \sum_{d'=2}^{d'=n_m} \Omega(d')(1 - g(n_m^*, d')), & \text{for } d = 1, \end{cases} \quad (18)$$

其中

$$g(n_m^*, d) = \binom{n_m^*}{d} \Big/ \binom{n_m}{d}. \quad (19)$$

证明。新节点 j 以概率 $\Omega(d)$ 生成度 d，并根据均匀分布从 n_m 个节点中选择 d 个节点。这些 d 个节点均属于可用的 n_m^* 个节点的概率为

$$\Pr(d \text{ nodes are available}) = \binom{n_m^*}{d} \Big/ \binom{n_m}{d}. \quad (20)$$

由此得到式 (18) 的第一行。

所选 d 个节点中至少有一个不可用的概率为

$$\Pr(\text{some of } d \text{ nodes are not available}) = 1 - \binom{n_m^*}{d} \Big/ \binom{n_m}{d}. \quad (21)$$

在这种情况下，节点 j 开始修复/解码，并进一步存储一个中间块，这意味着它存储了一个度为 1 的数据块。因此，我们得到式 (18) 的第二行。

□

现在我们来考察编码、修复和解码过程的计算复杂度。如果新加入节点成功获得所需的 $d \sim \Omega(d)$ 个中间块，则预期编码复杂度为 $\mathcal{O}(\mathbb{E}[\Omega(d)])$。如果新节点需要使用式 (16) 和编码块 $\mathbf{v}_j^m$ 来修复缺失的中间块，则计算复杂度为 $\mathcal{O}(|\mathcal{N}(\mathbf{v}_j^m)|)$。邻居数量 $|\mathcal{N}(\mathbf{v}_j^m)|$，即 $\mathbf{v}_j^m$ 的度，服从实际分布 $\Omega^*(d)$。根据引理1，我们可知

$$\Omega^*(d) \leq \Omega(d), \quad \forall d \geq 2. \quad (22)$$

因此，修复的预期计算复杂度为 $\mathcal{O}(\mathbb{E}[\Omega^*(d)]) \leq \mathcal{O}(\mathbb{E}[\Omega(d)])$。当新节点需要解码第 m 组中的所有数据块时，计算复杂度为 $\mathcal{O}(k_m)$ [39]。请注意，解码复杂度通常远高于编码和修复的复杂度。例如，在我们的仿真中，编码和修复通常涉及几十个数据块，而解码涉及数千个数据块，具体数量取决于组大小。因此，如果所有新加入节点都必须解码所有组（如 [34] 中的情况），计算开销会很大。另一方面，根据比特币区块链网络中的仿真结果（见第 VII-C 节），采用我们的方法时，新加入节点需要解码一组（而非仅进行编码或修复）的概率低于 6.3×10^{-5}。

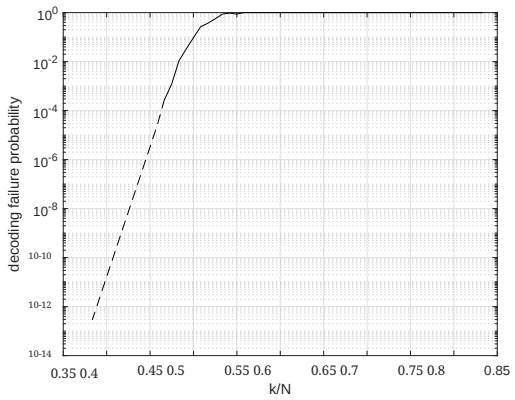


图 9. $f(k,3000;\Omega(d),12,4,100)$ 的解码失败概率。实线表示仿真得到的失败概率，虚线表示预测的失败概率。

VI. 确定组大小 k_m

正如后文将要解释的，我们的目标是尽可能增大 k_m，同时确保经过 γ 个时间周期后，能够以低于目标失败概率 ζ的解码失败概率恢复来自 $\mathcal{G}_m$ 的原始区块。可靠的系统通常要求失败概率低于 10^{-9} [44]。在我们提出的无速率编码区块链中，包含一组 $\mathcal{M}$个编码组，每个节点仅存储 $|\mathcal{M}|$ 个编码块以及尚未进行编码的区块。增强区块头的额外存储开销，即非系统式中间块的哈希值，微乎其微[1]。因此，我们将存储缩减系数定义如下。

定义 1. 存储缩减系数 R_s 定义为所提出的无速率编码区块链中单个节点所需空间与复制式区块链中所需空间之比，即

$$R_s = \frac{W - \sum_{m\in\mathcal{M}} k_m + |\mathcal{M}|}{W}. \tag{23}$$

编码区块总数为 $\sum_{m\in\mathcal{M}} k_m$。因此，$k_m$ 越大，$|\mathcal{M}|$ 就越小，存储缩减系数也就越小。因此，我们希望 k_m 越大越好。另一方面，如果 k_m 过大且有些节点离开网络，我们可能无法恢复（解码）某些组。为了找到合适的折衷方案，我们将解码失败概率定义如下。

定义 2. 给定度分布 $\Omega(d)$、节点离开和加入网络的速率 λ_l 和 λ_e，以及初始节点数 N，解码失败概率 $f(k,N;\Omega(d)\ \lambda_l,\lambda_e,\gamma)$ 表示自组编码后经过 γ 个时间周期，节点无法解码一组数据块的概率。

令 $p(\mathbf{G},\Omega^*(d)\ \epsilon)$ 表示在预编码矩阵 G 和编码符号度分布 $\Omega^*(d)$ 下，Raptor码无法解码至少一个信息符号的概率，并开销 ϵ。假设我们已知 $p(\mathbf{G},\Omega^*(d)\ \epsilon)$，例如可以通过类似于 [39]，中的分析推导得出，也可以通过仿真获得。那么我们有以下结果。

引理 2. 解码失败概率由式 (26) 给出。

证明. 不难看出

$$f(k,N;\Omega(d),\lambda_l,\lambda_e,\gamma) = \int_0^\infty p(\mathbf{G},\Omega^*(d),\epsilon)\Pr(N+\Delta_\gamma=(1+\epsilon)k)\,d\epsilon, \tag{24}$$

其中，Δ_γ 是两个服从斯凯拉姆分布 [45], 的泊松过程之和，即

$$\Pr(\Delta_\gamma=s) = e^{-\gamma(\lambda_l+\lambda_e)}\binom{\lambda_l}{\lambda_e}^{\frac{s}{2}}\sum_{X=0}^{\infty}\frac{1}{X!(X+s)!}(\gamma\sqrt{\lambda_l\lambda_e})^{2X+s} \tag{25}$$

因此，将式 (25) 代入式 (24)，可得式 (26)。

$$f(k,N;\Omega(d),\lambda_l,\lambda_e,\gamma) = \int_0^\infty p(\mathbf{G},\Omega^*(d),\epsilon)e^{-\gamma(\lambda_l+\lambda_e)}\binom{\lambda_l}{\lambda_e}^{\frac{(1+\epsilon)k-N}{2}}\times \sum_{X=0}^{\infty}\frac{1}{X!(X+(1+\epsilon)k-N)!}(\gamma\sqrt{\lambda_l\lambda_e})^{2X+(1+\epsilon)k-N}\,d\epsilon. \tag{26}$$

□

如下文所述，为了找到合适的 k_m值，我们使用函数 $f(k,N;\Omega(d)\ \lambda_l,\lambda_e,\gamma)$，该函数可以通过几种方式获得。一种方式是采用与 [39] 中所述类似的分析，先获得 $p(\mathbf{G},\Omega^*(d)\ \epsilon)$ 的值，再使用引理 2。另一种方式是通过仿真获得 $f(k,N;\Omega(d)\ \lambda_l,\lambda_e,\gamma)$。这需要付出一些努力，因为获得较小的 $f(k,N;\Omega(d)\ \lambda_l,\lambda_e,\gamma)$ 值可能需要过长的仿真时间。编码理论领域的研究者已提出多种解决这一问题的方法，例如参见 [46] 及其中的参考文献。不过，我们工作的主要目标并非编码理论，而是设计新型区块链。因此，在本文中，我们简单地假设可以获得 $f(k,N;\Omega(d)\ \lambda_l,\lambda_e,\gamma)$。对于下一节中的数值示例，我们通过仿真生成 $f(k,N;\Omega(d)\ \lambda_l,\lambda_e,\gamma)$，仿真范围截至 10^{-4}，然后将其外推至更小的值。例如，当 $\lambda_l=12,\lambda_e=4,N=3000,\gamma=100$，且 $\Omega(d)$ 遵循第 VII 节中的生成器度分布时，图 9 展示了解码失败概率。

回顾一下，我们的目标是在时间周期t_m 确定 k_m，此时网络中的节点数量为 $|\mathcal{V}_{t_m}|$，使得在经过 γ 个时间周期后，由于节点离开和加入网络导致网络变化为 $|\mathcal{V}_{t_m+\gamma}|$ 时，解码失败概率不高于 ζ。需要注意的是，当$N\to\infty$时，$f(k,N;\Omega(d)\ \lambda_l,\lambda_e,\gamma)$将成为 $\frac{k}{N}$的函数。对于有限的 N，概率 $f(k,N;\Omega(d)\ \lambda_l,\lambda_e,\gamma)$

[1]例如，当一组包含 $k=10000$ 个区块且 $r=0.8$ 时，增强区块头中需要存储 2500 个哈希值，占用 0.076 MB；与所有区块大小均为 1Mb 时节省的 9.76 GB 以上空间相比，这可以忽略不计。

随着 N 以固定比例 $\frac{k}{N}$ 减小而增加，参见图 10(b) 中的仿真结果。还应注意，组 $\mathcal{G}_m$ 中的数据块将在第 m 个增强区块确认后进行编码。回想一下，β 是每个时间间隔内生成的数据块数量，其平均值为常数，而 α 是确认数据块所需的深度。因此，我们根据以下内容通过查表计算 k_m：

$$f(k_m, |\mathcal{V}_{t_m}|; \Omega(d), \lambda_l, \lambda_e, \gamma + \frac{\alpha}{\beta}) = \zeta \tag{27}$$

其中，$\frac{\alpha}{\beta}$ 表示确认第 m个增强块所需的时间周期数，ζ 表示目标失败概率。

如果在某一时刻 $t' \geq t_m + \gamma$ 我们有 $|\mathcal{V}_{t'}| < |\mathcal{V}_{t_m}|$，那么我们无法保证在时间 $t' + \gamma$ 解码失败概率会小于 ζ。在这种情况下，每个节点 j 都会将 $\mathcal{G}_m$ 中的区块放回区块池 $\mathcal{P}_j$，然后以标准方式继续运行，但有一个例外：如果节点 j “挖出”下一个增强块，则为其分配序列号 m，而不是 $M + 1$。这个新的第 m个增强块会覆盖旧的增强块，其“挖矿”和验证过程与新“挖出”的增强块相同。因此，如果多个增强块具有相同的序列号，则只有最新的一个有效。如果恶意节点故意“重新挖掘”序列号为 m^* 且不需要“重新挖掘”的增强块，所有诚实节点都不会接受这个增强块，因为区块$\mathcal{G}_{m^*}$ 不在其区块池中。新的加入节点沿反向运行NMA算法，即从最长区块链分支中的最新增强块向创世区块运行。如果新节点遇到一个序列号为 m的增强块，而它在此前的反向过程中已经遇到过该序列号，那么新节点会直接忽略这个增强块。

需要指出的是，当增强区块“矿工”需要估算网络中的节点数量时，我们的方案会增加通信开销。为了获取节点数量，我们可以采用现有区块链中用于检查节点是否存活的“ping”和“pong”机制 [47]。为避免虚假连接，我们只能统计最近广播过诚实消息（例如有效区块或交易）的节点。每个节点每 24 小时可以广播一条存活消息，这应该足够了，因为在现代区块链中，节点数量通常变化得相当缓慢。例如，在比特币网络中，每天加入和离开的节点平均分别为 42.18 个和 43.16 个，参见第 VII.C 节。因此，从百分比来看，估算网络规模所增加的通信开销远小于典型的网络负载[2]。

VII. 仿真结果

我们的仿真使用 Matlab 2020b 在一台配备 Intel Core i7 八核 CPU（2.2 GHz）的笔记本电脑上进行。

[2]一条典型的“ping”或“pong”消息大小为 32 字节，这会使网络通信开销每小时增加 $32 \times N^2/24$ 字节。比特币区块大小为 10^6 字节，每 10 分钟生成一个，因此典型网络负载为每小时 $10^6 \times N \times 6$ 字节。因此，当 $N = 10$ 为 000 时，通信开销增加了 0.22%。

在我们的仿真中，我们使用 Reed-Solomon码作为预编码，实现了所提出的无速率编码区块链。Reed-Soloman码的生成矩阵取自 Matlab 2020b通信工具箱。生成器度分布 $\Omega(d)$ 如下。令 $S = c\ln(k/\delta)\sqrt{k}$，其中$c$ 和 δ 为常数，并定义

$$\tau(d) = \begin{cases} \frac{S}{ik}, & \text{for } d = 1, \ldots, \frac{k}{S} - 1, \\ \frac{S \ln(S/\delta)}{k}, & \text{for } d = \frac{k}{S}, \\ 0, & \text{for } d = \frac{k}{S} + 1, \ldots, k, \end{cases} \tag{28}$$

$$\rho(d) = \begin{cases} \frac{1}{k}, & \text{for } d = 1, \\ \frac{1}{d(d-1)}, & \text{for } d = 2, \ldots, k. \end{cases} \tag{29}$$

在 [33] 中，提出了以下度分布以获得 LT 码。

$$\mu(d) = \frac{\tau(d) + \rho(d)}{\sum_{j=1}^{k} \Big(\tau(j) + \rho(j)\Big)}. \tag{30}$$

在我们的仿真中，我们设置 $c = 0.1$ 和 $\delta = 0.5$，众所周知，这能带来良好且稳定的性能 [40]。如 III-A节所述，新节点应采用具有 $\Omega(1) = 0$ 的度分布。为此，我们稍微修改$\mu(d)$ 中的 (30)，并设置

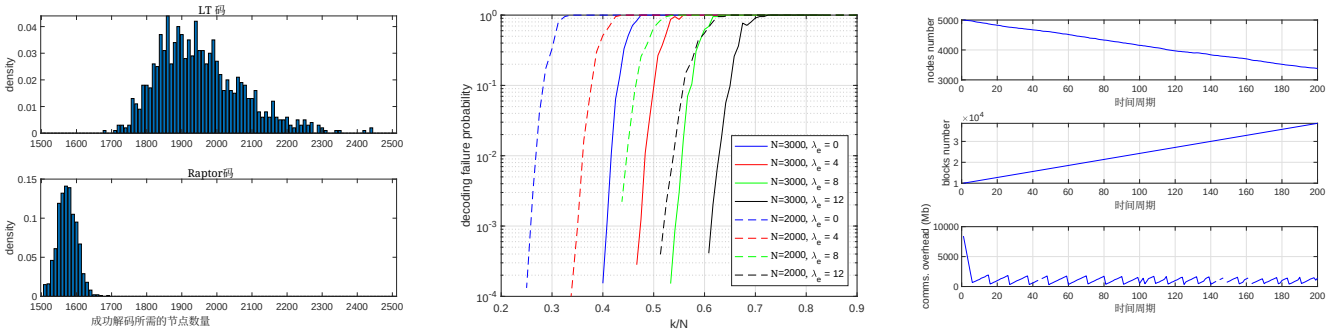
$$\Omega(d) = \begin{cases} 0, & \text{for } d = 1, \\ \mu(d) + \frac{\mu(1)}{k-1}, & \text{for } d = 2, \ldots, k. \end{cases} \tag{31}$$

我们首先比较动态网络中的 LT 码和 Raptor 码，并展示若干网络场景下的解码失败概率$f(k, N; \Omega(d)\ \lambda_l, \lambda_e, \gamma)$。接下来，我们在一个高度动态的区块链网络中评估所提出的无速率编码区块链，其中离开网络的节点数量远高于加入网络的节点数量。最后，我们展示所提出方案在现实世界中的比特币网络中的性能。

A. 编码性能

我们首先分别使用 LT 码和 Raptor码，研究成功解码所需的最少节点数。我们设置一个固定长度区块链，其中只有一个组，包含 $k = 1000$，网络初始有 $N = 3000$ 个节点，并且每个时间周期有 $\lambda_l = 50$ 个和 $\lambda_e = 40$ 个节点。我们将预编码的码率设为 $r = 0.8$。网络使用 NMA算法动态运行。我们生成了 1000 条区块链，并持续运行，直到新节点解码失败。我们记录每个网络在失败发生前最近一次成功解码所需的节点数，并在图 10(a) 中展示其密度。可以看出，Raptor码明显优于 LT 码，因为它成功解码所需的节点数更少。我们还可以看到，有时 LT 码解码需要超过 2400 个节点，这意味着有时它需要非常大的解码开销 $\epsilon = 1.4$。这是由 LT码 [40] [41]中存在的误码平层问题导致的。

接下来，我们在图 10(b) 中展示解码失败概率$f(k, N; \Omega(d)\ \lambda_l, \lambda_e, \gamma)$。我们固定 $\lambda_l = 12$、$\gamma = 100$，并取 $\lambda_e = \{0, 4, 8, 12\}$ 和 $N = \{2000, 3000\}$。



(a) (b) (c)

图 10。（a）动态区块链网络中 LT 码和 Raptor码的性能比较；（b）当 $\gamma = 100$, $\lambda_l = 12$, $\lambda_e = \{0, 4, 8, 12\}$ 时的解码失败概率；（c）随着时间周期增长，新加入节点的节点数量、区块数量及通信开销（Mb）。

我们将预编码的码率保持为 $r = 0.8$。注意，当 $\lambda_e = 0$ 时，在动态网络中对数据块进行解码等价于通过二元删除信道（BEC）进行解码，因为节点（符号）以均匀概率离开网络，即被删除，并且不会发生修复。从结果可以看出，就给定的解码失败概率而言，BEC 是最差的情形，因为它导致 k 的最小值。这为使用已知方法获得 $f(.)$提供了可能性；例如，参见[39], 对 BEC中的Raptor码的分析。我们将在未来的工作中探讨这一可能性。结果还表明，随着 N 减小，$f(.)$会增大。

B. 快速缩减网络

考察快速缩减网络很有启发意义。我们设定 $\lambda_l = 12$, $\lambda_e = 4$, $N = 5000$, $\gamma = 98$, $\beta = 144$,$\alpha = 244$ ，并将解码失败概率目标值设为 $\zeta = 10^{-12}$。我们假设，在采用所提出的无速率编码区块链方法时，区块链最初包含 5,000 个节点和 10,000 个未编码区块。我们假设所有区块的比特大小均为 1 Mb。为清楚展示组大小的变化，我们假设每个时间周期最多生成 1 个新的增强区块。网络运行 200 个时间周期。

图 10(c) 显示了给定时间周期内网络节点数量和区块链数据块数量，以及新节点为先前编码组生成新校验块或修复中间块所需的通信开销。通信开销以新加入节点所需的总比特数衡量。传统区块链的通信开销等于所有数据块的总比特大小，并随区块链长度增加。相比之下，当区块链总比特大小超过 38,000 Mb 时，无速率编码区块链的通信开销低于 800 Mb。通信开销交替增减，这是因为数据块持续生成，而只有在涉及的增强区块得到确认后才会对其进行编码。需要注意的是，通信开销随编码组数量 $|\mathcal{M}|$增加，因而也随区块链大小 W 增加。然而，由于 $|\mathcal{M}|$ 增长非常缓慢，在所展示的 200 个时间周期内，这种增幅并不明显。

表I 增强区块生成和 k_m 值。

seq.	1	2	3 ∼ 13	14 ∼ 26
第1次“挖掘”	1	2	3 ∼ 95	102 ∼ 198
k_m	1910	1906	1904 ∼ 1613	1593 ∼ 1287
第2次“挖掘”	100	101	102 ∼ 194	-
k_m	1600	1595	1593 ∼ 1301	-
第3次“挖掘”	199	200	-	-
k_m	1284	1279	-	-

在本实验中，26个增强区块被“挖掘”。具体而言，第3∼13个增强区块被“挖掘”了两次，而第1∼2个增强区块被“挖掘”了三次。原因是节点数量持续减少，因此，这些组中的区块需要重新分组并重新编码，以确保解码失败概率满足要求。表I列出了这些区块被“挖掘”的时间周期以及它们的 k_m 值。从表I可以看出，在初始阶段（当时有大量可用区块池）之后，增强区块以近乎恒定的间隔每隔12个时间周期被“挖掘”一次。表I还显示，增强区块会在 γ 个时间周期后立即重新生成或被“重新挖掘”，且其 k_m 值随时间周期而降低。这是因为网络中的节点数量正在迅速减少。

图 11 显示了新加入节点为了给定的组生成自己的奇偶块或中间块，需要从其他节点收集的编码块数量的累积分布函数；这些编码块可以是系统式的，也可以是非系统式的。可以看到，新加入节点以超过 90% 的概率只需收集 10 个数据块。这显著少于文献[13], [14] 和 [34]中提出的其他编码区块链。在这些研究中，新节点需要为组 G_m 收集超过 k_m 个数据块，而正如我们所见， k_m 通常远大于 10。

应当注意，本节模拟的是节点持续且快速离开网络的极端场景。在这种情况下，现有节点需要重新编码，以确保区块链可解码，从而增加整体通信开销。然而，在实践中，典型的区块链

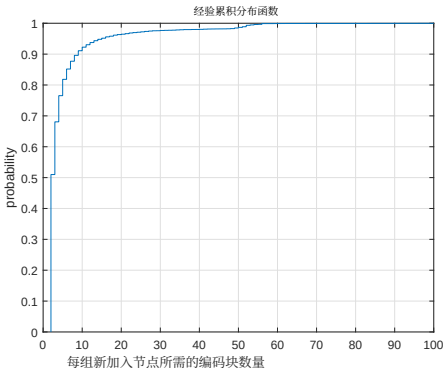


图 11. 新加入节点为生成给定组的校验块或中间块而需要从其他节点收集的编码块数量的累积分布函数。

像比特币这样的网络中的节点数量相对稳定，例如，多年来比特币网络中的节点数量约为 10,000，参见第 VII.C 节。此外，在实践中，我们可以采用一些简单的策略来降低重新编码的频率。例如，可以在 (27) 中引入系数$r < 1$，作为 $f(k_m, r|\mathcal{V}_{t_m}|; .) = \zeta$。这样，只有当 $|\mathcal{V}_{t'}| < r|\mathcal{V}_{t_m}|$时才会对增强区块进行“重新挖矿”。在比特币网络中，我们可以设置 $r = 0.7$，这意味着只有当节点数量减少 30% 时，才会对组进行“重新编码”。据我们所知，自 2015 [32]，以来，比特币历史上从未发生过这种情况，未来也预计不会发生。

C. 比特币网络中的性能

我们使用 [42] 中获取的真实比特币数据，评估所提出的无速率编码区块链。数据时间范围为 2018年 11 月 18 日至 2020年 11 月 13 日。我们假设该方法于 2018年 11 月 18 日应用于比特币区块链。区块链的初始长度和最新长度分别为 551,685 个区块和 656,805 个区块。我们假设每个时间周期为一天，即 24 小时，因此有 $\beta = 144$，因为平均每 10 分钟生成一个区块 [7]。接着，我们假设所有区块均已完整加载，且比特大小为 1 Mb。根据 [42]，节点以泊松过程离开和加入网络，速率分别为 $\lambda_l = 42.18$ 和 $\lambda_e = 43.16$，节点平均数量为 10,000。我们设定$\alpha = 144$ [48]、$\zeta = 10^{-12}$ 和 $\gamma = 98$ （约 3 个月）。我们忽略区块头的比特大小（最新约为 50 Mb），因为与区块数据总量相比可以忽略不计。我们将通信缩减系数定义为所提出的无速率编码区块链与传统区块链的通信开销之比。

我们在图 12(a) 中展示了式 (23) 定义的月平均存储缩减系数以及通信缩减系数。可以看到，存储量减少和通信量减少系数均降至约 0.4%，并在 2018年12月之后保持恒定。这是因为在实施初期，有大量原始区块需要编码为编码块。因此，初期的存储缩减系数和通信缩减系数都较高。

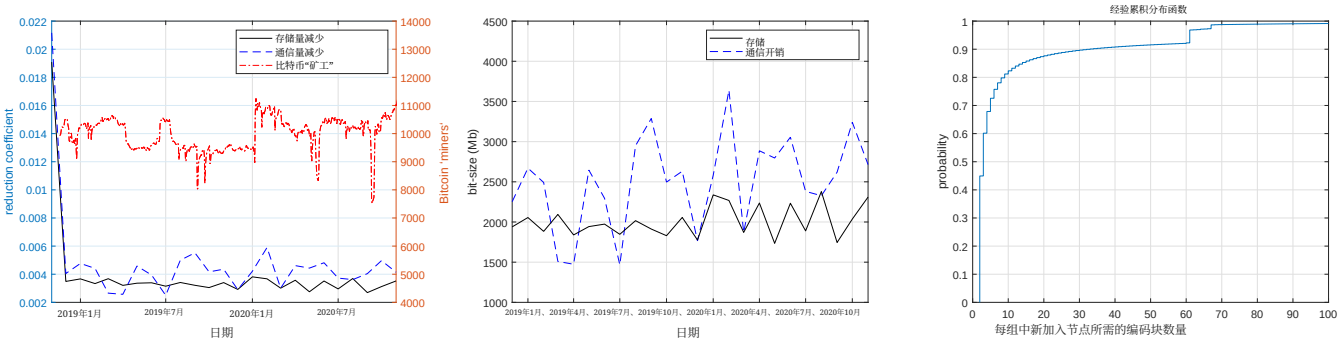
在图 12(b) 中，我们展示了每个节点的月平均存储量（以 Mb 为单位），以及每个新加入节点每月的平均通信开销。回想一下，对于传统的复制式区块链，存储和通信开销都会从 540 Gb 增至 640 Gb。因此，可以看到采用无速率编码区块链带来的显著缩减。尽管从 2018年12月到 2020年11月，区块链大小增加了 100 Gb，但平均存储量和通信开销分别仅增加了 400 Mb 和 700 Mb。图 12(b) 中存储和通信开销的变化趋势与图 12(a) 中的相似，因为比特币区块链长度以每天 144 个区块的恒定速率增长。

图 12(c)展示了新节点收集的编码块（无论是系统式还是非系统式）的数量的累积分布函数。以极高的概率（即 99%），新加入节点每组只需从其他节点获取 70 个编码块，而每组平均有 $10000 \times 0.337 = 3370$个数据块。新节点收集的数据块数量略高于第 VII-B 节图 11 中的数量。这是因为比特币网络中的节点更多——有 10,000 个，而第 VII-B 节假设为 5,000 个。因此，比特币网络中的组大小和预期生成器度均高于第 VII-B 节。尽管难以在图 12(c) 中观察到这一点，我们发现，新节点需要解码一个组 $\mathcal{G}_m$而非直接编码或修复的概率低于 6.3×10^{-5}。解码的复杂度相对较高，因此最好将这一概率保持在较低水平。

总之，如果所有比特币区块链节点都存储验证新交易所需的原始区块（约 5 GB [5]），并使用所提出的无速率编码区块链对所有其他旧区块进行编码，那么每个节点总共需要约 7 GB 的存储空间。这样，所有旧区块都可以恢复，而不是简单地删除。此外，新比特币节点加入网络时，只需下载 8 GB 的数据。这些数值远低于当前比特币所需的存储量；当前比特币需要下载并存储 300 Gb 以上的数据（如果完整加载所有区块，则需要 600 Gb）。

VIII. 结论

我们提出了一种无速率编码区块链，以降低实际动态物联网区块链网络中节点的存储需求，以及新加入节点的通信开销。该方案会根据网络状况调整编码参数，从而确保所有编码区块在需要时都能以极高概率成功解码。我们还提出了新型增强区块，并将其嵌入区块链中以存储编码参数，使这些参数不可篡改，并能借助区块链的集成和分布式特性在所有节点之间共享。我们使用合成数据和真实世界比特币数据进行了大量仿真。结果表明，与传统未编码区块链相比，每个节点的存储需求和每个新加入节点的通信开销均减少了 99.6%。未来一项值得研究的工作是确定编码



(a) (b) (c)

图 12. (a) 实际比特币区块链中的存储和通信缩减系数；(b) 实际比特币区块链中的存储和通信开销比特大小；(c) 实际比特币区块链中新加入节点所需编码块数量的累积分布函数。

区块链场景下 Raptor 码的参数，例如预编码率和度分布，以降低解码失败概率。

参考文献

[1] H.-N. Dai、Z. Zheng 和 Y. Zhang，《物联网区块链：综述》，电气电子工程师学会《物联网期刊》，第 6 卷，8094，2019年，第 5 期，8076 页。[2] G. O. Karame 和 E. Androulaki，《比特币与区块链安全》。Artech House，2016年。[3] X. Fan、B. Niu 和 Z. Liu，《可扩展区块链存储系统：研究进展与模型》，《计算》，第 104 卷，第 6 期，第 1497–1524 页，2022年。[4] Blockchain，《比特币区块链大小》，2022年。[在线]。网址：https://www.blockchain.com/charts/blocks-size。[5] 《Bitcoin Core》，2020年。 [在线]。网址：https://bitcoincore.org/。[6] Q. Zhou、H. Huang、Z. Zheng 和 J. Bian，《区块链可扩展性解决方案：综述》，《IEEE Access》，第 8 卷，第 16440–16455 页，2020年。[7] S. Nakamoto 等，《比特币：一种点对点电子现金系统》，2008年。 [白皮书]。[8] T. Dryja，《Utreexo：一种针对比特币UTXO集合优化的动态哈希累加器》，《IACR密码学电子印本档案》，第 2019 卷，611，2019年。[9] M. Zamani、M. Movahedi 和 M. Raykova，《RapidChain：通过全分片实现区块链扩展》，载于 ACM 计算机与通信安全会议论文集，第 931–948 页，2018年。[10] H. Huang、X. Peng、J. Zhan、S. Zhang、Y. Lin、Z. Zheng 和 S. Guo，《BrokerChain：一种用于账户/余额型状态分片的跨分片区块链协议》，载于电气电子工程师学会 INFOCOM，2022年。[11] S. Das、A. Kolluri、P. Saxena 和 H. Yu，《区块链共识协议的安全性》，载于国际信息系统安全会议论文集，第 465–480 页，2018年。[12] M. Dai、S. Zhang、H. Wang 和 S. Jin，《面向区块链分布式账本的低存储空间需求框架》，《IEEE Access》，第 6 卷，第 22970–22975 页，2018年。[13] D. Perard、J. Lacan、Y. Bachy 和 J. Detchart，《基于纠删码的低存储区块链节点》，载于电气电子工程师学会国际物联网会议（iThings）、电气电子工程师学会绿色计算与通信会议（GreenCom）、电气电子工程师学会网络、物理与社会计算会议（CPSCom）及电气电子工程师学会智能数据会议（SmartData）论文集，第 1622–1627 页，2018年。[14] H. Wu、A. Ashikhmin、X. Wang、C. Li、S. Yang 和 L. Zhang，《面向低存储区块链系统的分布式纠错编码方案》，电气电子工程师学会《物联网期刊》，第 7 卷，7071，2020年，第 8 期，7054 页。[15] R. K. Raman 和 L. R. Varshney，《通过动态分布式存储实现可扩展区块链编码》，《IEEE/ACM 网络学报》，第 29 卷，第 2588–2601 页，2021年12月。[16] C. Yang、X. Wang 和 A. Ashikhmin，《无线编码区块链的存储与通信权衡》，《IEEE 系统期刊》，第 16 卷，第 2911–2922 页，2022年6月。[17] M. Cebe、B. Kaplan 和 K. Akkaya，《一种基于网络编码的信息传播方法：面向物联网环境中的许可型区块链》，载于 ACM Mobiquitous，（纽约，美国），第 470–475 页，2018年11月。

[18] N. Chawla、H. W. Behrens、D. Tapp、D. Boscovic 和 K. S. Candan，“Velocity：通过无速率删除码提升区块传播的可扩展性”，载于 IEEE国际区块链与加密货币会议（ICBC），（韩国首尔），第 447–454 页，2019年5月。[19] M. Braun、A. Wiesmaier、N. Alnahawi 和 J. ßeibler，“ 关于基于消息的共识与网络编码 ”，载于 第 12 届未来网络国际会议（NoF），（葡萄牙科英布拉），第 1–9 页，2021年。[20] L. Zhang、T. Wang 和 S. C. Liew，“加速比特币网络中的区块传播：未编码与编码设计”，《计算机网络》，第 206 卷，第 108791 页，2022年4月。[21] M. Al-Bassam、A. Sonnino 和 V. Buterin，“欺诈证明与数据可用性证明：通过不诚实多数最大化轻客户端安全性并扩展区块链”，arXiv 预印本 arXiv:1809.09044，2018年。[22] P. Santini、G. Rafaiani、M. Battaglioni、F. Chiaraluce 和 M. Baldi，“针对区块链数据可用性攻击的基于里德-所罗门码协议的优化”，arXiv 预印本 arXiv:2201.08261，2022年。[23] M. Yu、S. Sahraei、S. Li、S. Avestimehr、S. Kannan 和 P. Viswanath，“编码默克尔树：解决区块链中的数据可用性攻击”，载于金融密码学与数据安全国际会议，（马来西亚亚庇），第 114–134 页，2020年2月。[24] D. Mitra、L. Tauz 和 L. Dolecek，“克服区块链系统中的数据可用性攻击：编码默克尔树的低密度奇偶校验码设计”，arXiv 预印本 arXiv:2108.13332，2021年。[25] D. Mitra、L. Tauz 和 L. Dolecek，“极化编码默克尔树：改进区块链系统中数据可用性攻击的检测”，载于 IEEE国际信息论研讨会（ISIT），（芬兰埃斯波），2022年6月。[26] S. Li、M. Yu、C.-S. Yang、A. S. Avestimehr、S. Kannan 和 P. Viswanath，“Polyshard：编码分片同时实现线性扩展效率与安全性”，《IEEE信息取证与安全汇刊》，第 16 卷，第 249–261 页，2020年。[27] B. Sasidharan 和 E. Viterbo，“采用编码分片的区块链系统中的私有数据访问”，载于 IEEE国际信息论研讨会（ISIT），（澳大利亚墨尔本），第 2684–2689 页，2021年7月。[28] D. S. Gadiraju、V. Lalitha 和 V. Aggarwal，“用于降低分片区块链存储成本和引导成本的安全再生码”，载于 IEEE国际区块链会议（Blockchain），（美国罗得岛），第 229–236 页，2020年11月。[29] S. B. Wicker 和 V. K. Bhargava，《里德-所罗门码及其应用》。Wiley-IEEE出版社，1994年。[30] G. Cancellieri，“低密度奇偶校验码”，Wiley-IEEE出版社，2002年。[31] P. Trifonov，“极化码的高效设计与解码”，IEEE 通信汇刊，第 60 卷，第 11 期，第 3221–3227 页，2012年。[32] “比特币节点数量（自2015年以来）”，2020年。[在线]。可访问：https://coin.dance/nodes。[33] M. Luby，“LT码”，载于第43届计算机科学基础研讨会论文集，第 271–280 页，2002年。[34] S. Kadhe、J. Chung 和 K. Ramchandran，“SeF：一种用于大幅降低区块链存储成本的安全喷泉架构”，arXiv:1906 12140，2019年。。[35] L. Quan 和 Q. Huang，“透明编码区块链”，载于会议论文集

第 15 届新兴网络实验与技术国际会议论文集，第 12–13 页，2019 年。[36] R. Pal，“用于区块链引导的喷泉码”，载于国际通信系统与网络会议（COMSNETS），（印度班加罗尔），第 1–5 页，2020年 1 月。[37] Q. Huang、L. Quan 和 S. Zhang，“区块链的下采样与透明编码”，《电气电子工程师学会网络科学与工程汇刊》，第 1–1 页，2022年。[38] A. Tiwari 和 V. Lalitha，“用于低存储区块链的安全 Raptor 编码器和解码器”，载于电气电子工程师学会 COMSNET，（印度班加罗尔），第 161–165 页，2021年 1 月。[39] A. Shokrollahi，“Raptor 码”，《电气电子工程师学会信息论汇刊》，第 52 卷，第 6 期，第 2551–2567 页，2006 年。[40] D. J. MacKay，“喷泉码”，《IEE 通信学报》，第 152 卷，第 6 期，第 1062–1068 页，2005 年。[41] I. Hussain、M. Xiao 和 L. K. Rasmussen，“加性白高斯噪声信道上 LT 码的误码平层分析”，载于电气电子工程师学会全球电信会议（GLOBECOM）论文集，第 1–5 页，2011 年。[42] “比特币节点数量”，2022年。[在线]。可访问：https://bitnodes.io/dashboard/?days=730。[43] X.-Y. Hu、E. Eleftheriou 和 D.-M. Arnold，“规则与不规则渐进边增长 Tanner 图”，《电气电子工程师学会信息论汇刊》，第 51 卷，第 1 期，第 386–398 页，2005 年。[44] X. Li、M. C. Huang、K. Shen 和 L. Chu，“对内存硬件错误及软件系统易受影响程度的现实评估”，载于 USENIX 年度技术会议论文集，第 75–88 页，2010 年。[45] D. Karlis 和 I. Ntzoufras，“计数数据差异的贝叶斯分析”，《医学统计学》，第 25 卷，第 11 期，第 1885–1905 页，2006 年。[46] T. Richardson，“LDPC 码的误码平层”，载于第 41 届 Allerton 通信、控制与计算年度会议论文集，第 1426–1435 页，2003 年。[47] “比特币点对点网络”，2020年。[在线]。可访问：https://developer.bitcoin.org/reference/p2p networking.html。[48] “比特币维基”，2020年。 [在线]。可访问：https://en.bitcoin.it/wiki/。