

极客的互联网计算机


(v1.0)

DFINITY 团队¹

2022 年 1 月 21 日

本文档通过 DeepL 翻译

校对：kk 德米安

当前校对进度： 

摘要

智能合约是一种新的软件形式，将彻底改变软件的编写、IT 系统的维护以及应用程序和整个企业的构建方式。智能合约是可组合的、自主的软件片断，在去中心化的区块链上运行，这使得它们是不可篡改和不可阻挡的。在本文中，我们描述了互联网计算机（IC），它是区块链的一个激进的新设计，释放了智能合约的全部潜力，克服了传统区块链上智能合约在速度、存储成本和计算能力方面的限制。这使得智能合约首次实现了完全去中心化的应用，并在区块链上进行点到点的托管。IC 由一套加密协议组成，将独立运行的连接成一个区块链的集合。这些区块链承载并执行 "Canister" 合约，是 IC 的智能合约形式。Canister 可以存储数据，对这些数据进行非常一般的计算，并提供一个完整的技术栈，直接为终端用户提供网页。计算和存储成本由一个 "反向气体模型" 来支付，Canister 的开发者以 cycle 的形式预付成本，这些成本从 ICP--IC 的原始代币中获得。ICP 代币也被用于管理：IC 由一个去中心化的自治组织，或称 DAO 管理 (NNS)，除其他事项外，它决定网络拓扑结构的变化和协议的升级。

1 引言

1.1 释放智能合约

由于其独特的功能，智能合约是 Web3 的关键推动者，Web3 是一种新的网络方法，其中

¹IC 允许智能合约有一系列的可变性政策，从纯粹的不可变到单方面的可升级，还有其他介于两者之间的选项。

应用程序完全由其用户控制，并在去中心化的区块链平台上运行。这种去中心化的应用程序（dapp）通常是代币化的，意味着代币被分配给用户，作为参与 dapp 的奖励。参与可以有許多不同的形式，从主持和提供内容到管理一个 dapp，再到创建和维护一个 dapp。通常情况下，代币也可以在交易所购买；事实上，出售代币是资助 dapp 开发的一种常见方式。最后，代币也被用来作为 dapp 提供的服务或内容的一种支付方式。在今天的区块链平台上运行的智能合约，包括所有流行的区块链平台（如以太坊），受到许多限制，如交易和存储成本高，计算速度慢，无法为用户提供前台。因此，许多流行的区块链应用程序不是完全去中心化的，而是混合型的，其中大部分应用程序托管在传统的云平台上，并为其整体功能的一小部分调用区块链上的智能合约。不幸的是，这使得此类应用程序非去中心化，并使其面临传统云托管应用程序的许多缺点，如受云供应商的控制，并容易受到许多单点故障的影响。

互联网计算机（IC）是一个执行**智能合约**的新平台。在这里，我们在非常广泛的意义上使用“智能合约”一词：一个*通用的、不可改变的、防篡改的*计算机程序，其执行是在一个*去中心化的公共网络上自主进行*。

- 所谓*通用*，我们的意思是，智能合约程序的类别是图灵完全的（即任何可计算的东西都可以由智能合约计算）。
- 所谓*不可变*，我们的意思是，一旦部署，智能合约的代码不能被一方单方面改变。¹
- 我们所说的*防篡改*，是指程序的指令被忠实地执行，中间和最终结果被准确地存储和或传输。
- 我们所说的*自主*，是指智能合约由网络自动执行，不需要任何个人的行为。
- 我们所说的*去中心化的公共网络*，是指可以公开访问的计算机网络，在地理上分布，并且不受少数个人或组织的控制。

此外，智能合约

- 是*可组合的*，这意味着它们可以相互作用，并且
- 支持*代币化*，意味着他们可以使用和交易数字代币。

与现有的智能合约平台相比，IC 的优势：

- *更具成本效益*，特别是允许应用程序以以前平台的一小部分成本来计算和存储数据。
- 为处理智能合约交易提供*更高的吞吐量和更低的延时*。
- *更具可扩展性*，特别是，IC 可以处理无限制的智能合约数据和计算量，因为它可以通过向网络添加更多的节点（子网）来增长容量。

除了提供一个智能合约平台外，IC 还被设计为一个*完整的技术栈*，这样就可以建立完全在 IC 上运行的系统和服务。特别是，IC 上的智能合约可以为终端用户创建的 HTTP 请求提供服务，因此，智能合约可以直接为交互式网络体验提供服务。这意味着可以在不依赖企

业云主机服务或私人服务器的情况下创建系统和服务，从而以真正的点对点方式提供智能合约的所有好处。

实现 Web3 的愿景。对于终端用户来说，访问基于 IC 的服务基本上是透明的。他们的个人数据比在公共或私有云上访问应用程序时更安全，但与应用程序的互动体验是相同的。

然而，对于创建和管理这些基于 IC 的服务的人来说，IC 消除了许多与开发和部署现代应用程序和微服务相关的成本、风险和复杂性。例如，IC 平台为大型技术公司所推动的垄断互联网的整合提供了一个替代方案。此外，它的安全协议保证了可靠的消息传递、透明的问责制和弹性，而无需依赖防火墙、备份设施、负载均衡服务或故障转移协调。

建设 IC 是为了恢复互联网的开放性、创新性和创造性的根基，换句话说，是为了实现 Web3 的愿景。

- 支持互操作性、共享功能、永久 API 和自主应用，所有这些都能降低平台风险，鼓励创新和协作。
- 将数据自动保存在内存中，这消除了对数据库服务器和存储管理的需求，提高了计算效率，并简化了软件开发。
- 简化了 IT 组织需要集成和管理的技术栈，从而提高了运营效率

1.2 互联网计算机的高层视图

初步估计，IC 是一个相互作用的**复制状态机**的网络。复制的状态机的概念在分布式系统中是一个相当标准的概念。[Sch90]但我们在此简单介绍一下，从**状态机**的概念开始。

状态机是一种特殊的计算模型。这种机器保持一种**状态**，相当于普通计算机中的主存储器或其他形式的数据存储。这样的机器以离散的**轮次**执行：在每一轮中，它接受一个**输入**，对输入和**当前状态**应用一个**状态转换函数**，得到一个**输出**和一个**新的状态**。**新的状态**在下一轮中成为**当前状态**。

IC 的状态转换函数是一个**通用函数**，也就是说，存储在状态中的一些输入和数据可能是任意的**程序**，这些**程序**作用于其他输入和数据。因此，这样的状态机代表了一个通用的（即图灵完全）计算模型。

为了实现**容错**，状态机可以被**复制**。**复制的状态机**包括一个子网的**副本**，每个子网都在运行同一个状态机的副本。一个子网应该继续运行--并且正确地运行--即使一些副本有**问题**。

一个子网中的每个副本以相同的顺序处理相同的输入是至关重要的。为了实现这一点，子网中的副本必须运行一个**共识协议**。[Fis83]这将确保子网中的所有副本以相同的顺序处理输入。因此，每个副本的内部状态将以完全相同的方式随时间演变，每个副本将产生完全相同的输出序列。请注意，IC 上的复制状态机的输入可能是由外部用户产生的输入，或者是由另一个复制状态机产生的输出。同样地，一个复制的状态机的输出可以是指向外部用户的输出，也可以是另一个复制的状态机的输入。

1.3 故障模型

在计算机科学的这个领域，人们通常考虑两种类型的复制故障：**崩溃故障**和**拜占庭容错**。崩

崩溃故障发生在一个复制体突然停止并且没有恢复的情况下。**拜占庭故障**是指副本可能以任意的方​​式偏离其规定的协议的故障。此外，在拜占庭故障中，一个或可能几个副本可能直接在恶意对手的控制之下，而恶意对手可能协调这些副本的行为。在这两种类型的故障中，拜占庭故障的潜在破坏性要大得多。

共识的协议和实现复制状态机的协议通常会对**多少个**复制体可能出现故障以及故障的**程度**（崩溃或拜占庭）做出假设。在 **IC** 中，假设如果一个给定的子网有 n 个副本，那么少于 $n/3$ 的这些副本是有问题的，这些故障可能是拜占庭式的。（请注意，**IC** 中的不同子网可能有不同的规模）。

1.4 沟通模式

共识的协议和实现复制状态机的协议通常也会对**通信模型**做出假设，该模型描述了对手在复制体之间延迟传递信息的能力。在光谱的两端，我们有以下的模型。

- 在**同步模型**中，存在一些已知的有限时间界限 Δ ，这样，对于任何发送的信息，它将在少于 Δ 时间的情况下被送达。
- 在**异步模型**中，对于发送的任何信息，对手可以通过任何有限的时间来延迟其回传，因此对交付信息的时间没有约束；然而，每个信息**最终**都必须被回传。

由于 **IC** 子网中的复制体通常分布在全球各地，同步通信模型将是非常不现实的。事实上，一个攻击者可以

通过延迟诚实副本或它们之间的通信来破坏协议的正确行为。这种攻击通常比控制和破坏一个诚实的副本更容易发动。

在全球分布式子网的设置中，最现实和稳健的模型是异步模型。不幸的是，在这个模型中没有已知的真正实用的共识协议（最近的异步共识协议，如在 **[MXC+16]**，达到了合理的吞吐量，但延迟不是很好）。因此，像其他大多数不依赖同步通信的实用拜占庭容错系统（如 **PBFT [CL99, BKM18, AMN+20]**），**IC** 选择了一个折衷方案：**部分同步通信模型 [DLS88]**。这种部分同步模型可以用不同的方式制定。**IC** 使用的部分同步假设大致上说，对于每个子网，该子网中的副本之间的通信在短时间内是 **Cycle** 性同步的；此外，同步约束 d 不需要事先知道。这种部分同步性假设只是为了确保共识协议取得进展（所谓的有效性属性）。部分同步假设（也不需要最终交付假设）不需要确保共识的正确行为（所谓的安全性属性），也不需要 **IC** 协议栈中的其他地方。

在部分同步和拜占庭故障的假设下，已知我们对故障数量的约束 $f < n/3$ 是最优的。

1.5 许可模式

最早的共识协议（例如，**PBFT [CL99]**）是**许可的**，也就是说，组成复制状态机的副本由一个中心化的组织来管理，该组织决定哪些实体提供副本，网络的拓扑结构，可能还会实施某种中心化的公钥基础设施。许可的共识协议通常是最有效的，虽然它们确实避免了单点故障，但集中式的治理对于某些应用来说是不可取的，而且它与正在兴起的 **Web3** 时代的精神背道而驰。

最近，我们看到了**无许可**共识协议的兴起，如比特币 **[Nak08]**、以太坊 **[But13]**，以及

Algorand [GHM+17].这类协议基于**区块链**和工作证明（**PoW**）（如比特币、V2.0 之前的以太坊）或**股权证明（PoS）**（如 Algorand、Ethereum v2.0）。虽然这类协议是完全去中心化的，但它们的效率却远低于许可协议。我们还指出，正如在 [PSS17]，基于 PoW 的共识协议，如比特币，在异步通信网络中不能保证正确性（即安全性）。

IC 的许可模型是一个**混合模型**，获得了许可协议的效率，同时提供了许多去中心化 PoS 协议的好处。这种混合模型被称为 **DAO 控制的网络**，（大致上）工作原理如下：每个子网运行一个许可的共识协议，但一个**分散的自治组织（DAO）** 决定哪些实体提供副本，配置网络的拓扑结构，提供一个公钥基础设施，并控制哪个版本的协议被部署到副本。IC 的 DAO 被称为**网络神经系统（NNS）**，基于 PoS，因此 NNS 的所有决定都是由社区成员做出的，他们的投票权由他们在 NNS 中投入的 IC 的原生治理代币的数量决定（见 Section 1.8 更多关于此代币的信息）。通过这个基于 PoS 的治理系统，可以创建新的子网，可以在现有的子网中添加或删除副本，可以部署软件更新，也可以对 IC 进行其他修改。NNS 本身是一个复制的状态机，它（像任何其他）运行在一个特定的子网上，其成员是通过相同的基于 PoS 的治理系统确定的。NNS 维护着一个叫做**注册表**的数据库，它记录着 IC 的拓扑结构：哪些副本属于哪些子网，副本的公钥，等等。（参见 Section 1.10 了解更多关于 NNS 的细节）。

因此，人们看到，IC DAO 控制的网络允许 IC 实现许可网络的许多实际好处（在更有效的共识方面），同时保持去中心化网络的许多好处（由 DAO 控制的治理）。

运行 IC 协议的复制体被托管在地理上分布的、独立运行的数据中心的服务器上。这也加强了 IC 的安全性和分散性。

1.6 链式钥匙加密法

事实上，IC 的共识协议确实使用了区块链，但它也使用了公钥密码学，特别是数字签名：由 NNS 维护的注册表被用来将公钥绑定到副本和子网成为一个整体。这实现了一个独特而强大的技术集合，我们称之为**链式密码学**，它有几个组成部分。

1.6.1 阈值签名

链式密码学的第一个组成部分是**阈值签名**：这是一种成熟的密码技术，它允许子网拥有一个公开的签名验证密钥，其对应的秘密签名密钥被分成**若干份**，这些份在子网的所有副本中分配，其方式是腐败的副本持有的份不能让他们伪造任何签名，而诚实的副本持有的份允许子网产生符合 IC 政策和协议的签名。

这些阈值签名的一个关键应用是

一个子网的单个输出可以由另一个子网或外部用户通过简单地验证与（第一）子网的公共签名验证密钥有关的数字签名来进行验证。

请注意，子网的公共签名验证密钥可以从 NNS 获得--这个公共签名验证密钥在子网的生命 Cycle 内保持不变（即使子网的成员在该生命 Cycle 内可能发生变化）。这与许多不可扩展的基于区块链的协议形成对比，后者需要验证整个区块链才能验证任何单一的输出。

正如我们将看到的，这些阈值签名在 IC 内有许多其他的应用。其中一个应用是让子网中的每个副本获得不可预测的伪随机位（来自于这种签名）。这是共识中使用的**随机信标**和

执行中使用的**随机磁带**的基础。

为了安全地部署阈值签名，IC 使用了一个创新的**分布式密钥生成 (DKG)** 协议，该协议构建了一个公开的签名验证密钥，并为每个副本提供了相应的秘密签名密钥的份额，并在我们的故障和通信模型中工作。

1.6.2 链式进化技术

链式密码学还包括一个复杂的技术集合，用于随着时间的推移稳健、安全地维护基于区块链的复制状态机，这些技术共同构成了我们所说的**链式演进技术**。每个子网在许多轮的**纪元**中运行（通常在几百轮的数量级上）。使用阈值签名和其他一些技术，链式进化技术实现了许多基本的维护活动，这些活动定期执行，其节奏与纪元相联系。

垃圾收集。在每个纪元结束时，所有已经处理过的输入，以及为这些输入排序所需的所有共识级协议消息，都可以安全地从每个副本的内存中清除掉。这对于保持副本的存储需求不受限制地增长至关重要。这与许多不可扩展的基于区块链的协议形成对比，在这些协议中，必须存储从创世区块开始的整个区块链。

快速转发。如果一个子网中的副本落后于它的同伴非常多（因为它长期停机或与网络断开连接），或者一个新的副本被添加到子网中，它可以被快速进到最近的纪元的开始，而不必运行共识协议和处理到这一点的所有输入。这与许多不可扩展的基于区块链的协议相反，在这些协议中，必须处理从创世区块开始的整个区块链。

子网成员的变化。子网的成员（由 NNS 决定，见 Section 1.5）可能会随着时间的推移而改变。这只能发生在一个纪元边界，需要谨慎行事，以确保一致和正确的行为。

积极主动地重新分享秘密。我们在上面提到 Section 1.6.1 IC 如何使用链式密码学--特别是阈值签名--进行输出验证。这是基于**秘密共享**，它通过将秘密（在这种情况下，秘密签名密钥）分割成储存在副本中的份额，来避免任何单点故障。在每个纪元的开始，这些份额被**主动地重新分享**。这实现了两个目标。

- 当一个子网的成员发生变化时，重新共享将确保任何新成员拥有适当的秘密份额，而任何不再是成员的副本不再拥有秘密份额。
- 如果在任何一个纪元，甚至在**每一个纪元**都有少量的股份被泄露给攻击者，这些股份对攻击者没有任何好处。

协议升级。当 IC 协议本身需要升级，以修复错误或增加新的功能时，这可以在一个纪元的开始使用特殊的协议自动完成。

1.7 执行模式

如前所述，IC 中的复制状态机可以执行任意的程序。IC 中的基本计算单元被称为**Canister**，它与**进程**的概念大致相同，因为它包括一个**程序**和其**状态**（随时间变化）。

Canister 程序是用 **WebAssembly** 编码的，简称 **Wasm**，这是一种基于堆栈的虚拟机的

二进制指令格式。**Wasm** 是一个开放的标准。² 虽然它最初是为了实现网页上的高性能应用而设计的，但它实际上非常适用于通用计算。

IC 提供了一个运行时环境，用于在一个 **Canister** 中执行 **Wasm** 程序，并与其他 **Canister** 和外部用户（通过消息传递）进行通信。虽然在原则上，人们可以用任何可以编译为 **Wasm** 的语言来编写一个 **Canister** 程序，但已经设计了一种叫做 **Motoko** 的语言，它与 **IC** 的操作语义非常一致。**Motoko** 是一种强类型的、以行为体为基础的编程语言。³ **d³** 编程语言，内置了对正交持久性和异步消息传递的支持。⁴ **e⁴** 和异步消息传递。正交持久化只是意味着由 **Canister** 维护的所有内存都是自动持久化的（也就是说，它不需要被写入文件）。**Motoko** 有许多生产力和安全特性，包括自动内存管理、泛型、类型推理、模式匹配，以及任意和固定精度的算术。

除了 **Motoko** 之外，**IC** 还提供了一种称为 **Candid** 的消息接口定义语言和线格式，用于类型化、高级和跨语言的互操作性。这允许任何两个 **Canister**，即使是用不同的高级语言编写的，也可以很容易地相互通信。

为了完全支持任何特定编程语言的 **Canister** 开发，除了该语言的 **Wasm** 编译器，还必须提供某些运行时支持。目前，除了 **Motoko** 之外，**IC** 还完全支持 **Rust** 编程语言的 **Canister** 开发。

1.8 公用事业令牌

该 **IC** 利用了一个名为 **ICP** 的实用令牌。该令牌用于以下功能。

在 NNS 中的押注：正如在第 1.5 节中所讨论的，**ICP** 代币可以在 **NNS** 中押注以获得投票权，从而参与控制 **IC** 网络的 **DAO**。Section 1.5 中已经讨论过，**ICP** 代币可以在 **NNS** 中被抵押，以获得投票权，从而参与控制 **IC** 网络的 **DAO**。在 **NNS** 中持有 **ICP** 代币并参与 **IC** 网络的用户可以获得投票权。

NNS 的治理者也会收到新铸造的 **ICP** 代币作为投票奖励。奖励的金额由 **NNS** 制定和执行的决策决定。

转换为 Cycle。**ICP** 被用来支付 **IC** 的使用费。更具体地说，**ICP** 代币可以转换为 **Cycle**（即烧毁），这些 **Cycle** 用于支付创建 **Canister**（见 Section 1.7）以及 **Canister** 使用的资源（存储、**CPU** 和带宽）。**ICP** 转换为 **Cycle** 的速度由 **NNS** 决定。

对节点提供者的支付。**ICP** 代币用于支付节点提供者--这些实体拥有并运营承载构成 **IC** 的副本的计算节点。在定期（目前是每月），**NNS** 决定每个节点提供者应该收到的新铸造的代币数量，并将代币发送到节点提供者的账户。支付代币的条件是，根据 **NNS** 制定和执行的决策，向 **IC** 提供可靠的服务。

²见 <https://webassembly.org/>.

³见 https://en.wikipedia.org/wiki/Actor_model.

⁴见 [https://en.wikipedia.org/wiki/Persistence_\(computer_science\)#Orthogonal_or_transparent_persistence](https://en.wikipedia.org/wiki/Persistence_(computer_science)#Orthogonal_or_transparent_persistence).

1.9 边界节点

边界节点提供 IC 的网络边缘服务。特别是，它们提供

- 明确规定了进入 IC 的入口。
- 为 IC 提供拒绝服务保护。
- 从传统的客户端（如网络浏览器）无缝访问 IC。

为了促进从传统客户端无缝访问 IC，边界节点提供的功能是将用户的标准 HTTPS 请求翻译成指向 IC 上的 Canister 的入口信息，然后将这个入口信息路由到这个 Canister 所在的子网的特定副本上。此外，边界节点提供额外的服务，以改善用户体验：缓存、负载平衡、速率限制，以及让传统客户验证来自 IC 的响应的能力。

一个 Canister 由 `icO.app` 域名上的一个 URL 识别。最初，传统客户端查找该 URL 的相应 DNS 记录，获得一个边界节点的 IP 地址，然后向该地址发送一个初始 HTTPS 请求。边界节点返回一个基于 javascript 的 "服务工作者 (Service workers)"，将在传统客户端中执行。在此之后，传统客户端和边界节点之间的所有互动都将通过这个服务工作者完成。

服务工作者执行的基本任务之一是使用链式密钥加密法对来自 IC 的响应进行验证（见 Section 1.6）。为了做到这一点，NNS 的公共验证密钥被硬编码在服务工作者中。

边界节点本身负责将请求路由到指定 Canister 所在的子网上的副本。执行这种路由所需的信息是由边界节点从 NNS 获得的。边界节点保持一个有提供及时回复的副本的列表，并从这个列表中随机选择一个副本。

传统客户和边界节点之间以及边界节点和副本之间的所有通信都是通过 TLS 来保障的。

5

除了传统的客户端，还可以使用 "IC 原生" 客户端与边界节点进行交互，这些客户端已经包括服务工作者逻辑，不需要从边界节点检索服务工作者程序。

就像复制一样，边界节点的部署和配置是由 NNS 控制的。

1.10 NNS 的更多细节

已经提到 Section 1.5 中已经提到，网络神经系统（NNS）是一个控制 IC 的算法治理系统。它由特殊系统子网中的一组 Canister 实现。这个子网与其他子网一样，但配置上有些不同（例如，系统子网上的 Canister 不为它们使用的 Cycle 收费）。

一些最相关的 NNS Canister 是

- **注册表 Canister**，存储 IC 的配置，即哪些副本属于哪个子网，与子网和单个副本相关的公钥，等等。
- **治理机构**，负责管理关于 IC 如何发展的决策和投票，以及

⁵见 https://en.wikipedia.org/wiki/Transport_Layer_Security.

- **账簿 Canister**，它记录了用户的 ICP 公用代币账户和它们之间的交易。

1.10.1 关于 NNS 的决策

任何人都可以通过在所谓的**神经元**中押注 ICP 代币来参与 NNS 治理。然后，神经元持有者可以对**提案**进行建议和投票，这些提案是关于 IC 应该如何改变的提议，例如，子网拓扑结构或协议应该如何改变。神经元的决策投票权是基于股权证明的 (Pos)。直观地讲，拥有更多赌注 (Staking) 的 ICP 代币的神经元有更多的投票权。然而，投票权也取决于其他一些神经元的特征，例如，更多的投票权是给那些承诺在较长时间内保持其代币抵押的神经元持有人。

每个提案都有一个确定的投票期。如果在表决期结束时，总投票权的简单多数投票赞成该提案，并且这些赞成票构成总投票权的特定法定人数（目前为 3%），则该提案被**通过**。否则，该提案被**否决**。此外，在任何时候，如果有绝对多数（超过总投票权的一半）赞成或反对该提案，则该提案被通过或否决。

如果一项建议被采纳，治理罐会自动执行该决定。例如，如果一项提案建议改变网络拓扑结构并被采纳，治理罐就会自动用新的配置更新注册表。

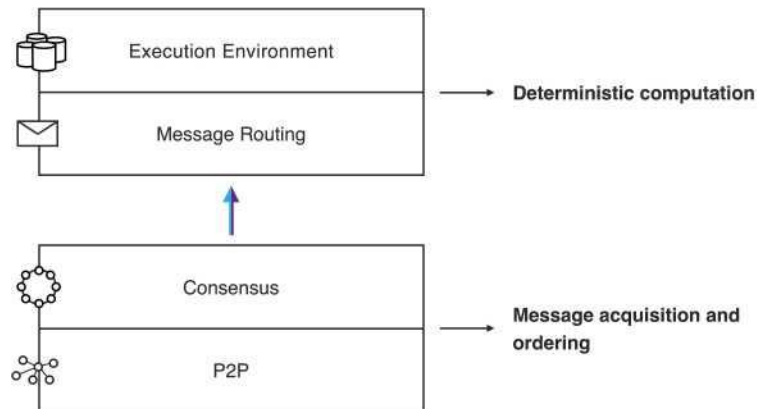


图 1：互联网计算机协议的层次

1.11 正在进行的工作

IC 的架构仍在不断发展和扩大。以下是即将部署的几个新功能。

DAO 控制的 Canister。正如 IC 的整体配置由 NNS 控制一样，任何 Canister 也可以由它自己的 DAO 控制，称为**服务神经系统 (SNS)**。控制一个 Canister 的 DAO 可以控制对 Canister 逻辑的更新，以及发布由 Canister 执行的特权命令。

阈值 ECDSA。ECDSA 签名 [JMV01]被用于加密货币，如比特币和以太坊，以及许多其他应用中。虽然阈值签名已经是 IC 中的一个基本成分，但这些并不是阈值 ECDSA 签名。这个新功能将允许单个 Canister 控制一个 ECDSA 签名密钥，该密钥在承载 Canister 的子网中的所有副本中安全地分发。

比特币和以太坊整合。在新的阈值 ECDSA 功能的基础上，该功能将允许 Canister 与比特币和以太坊区块链互动，包括签署交易的能力。

HTTP 集成。这个功能将允许 Canister 读取任意的网页（在 IC 外部）。

2 架构概述

如图所示 Figure 1 所示，互联网计算机协议由四层组成。

- 点对点层（见 Section 4);
- 共识层（见 Section 5);
- 路由层（见 Section 6);
- 执行层（见 Section 7).

链式钥匙加密法影响到几个层次，并在第 3 章中详细讨论。Sections 3(阈值签名)和 8 (链式进化技术)中详细讨论。

2.1 点对点层

对等层的任务是在子网中的副本之间传输协议消息。这些协议消息包括

- 用来实现共识的信息。
- 由外部用户产生的输入信息。

基本上，点对点提供的服务是一个 "尽力而为 "的广播频道。

如果一个诚实的复制体广播了一个消息，那么这个消息最终会被子网中所有诚实的复制体收到。

设计目标包括以下内容。

- **有界限的资源。**所有的算法都必须在有限的资源（内存、带宽、CPU）下工作。
- **优先次序。**根据某些属性（如类型、大小、轮次），不同的信息可能会有不同的优先级处理，而且这些优先级可能会随着时间的推移而改变。
- **效率。**高吞吐量比低延时更重要。
- **DOS/SPAM 的弹性。**腐败的复制体不应该阻止诚实的复制体相互之间的通信。

2.2 共识层

IC 的共识层的工作是对输入进行排序，以便子网中的所有副本以相同的顺序处理这些输入。对于这个问题，文献中有许多协议。IC 使用了一种新的共识协议，这里对其进行了高度描述。

任何安全的共识协议都应该保证两个属性，这两个属性（大致上说）是。

- **安全性：**所有的复制体实际上都同意相同的输入顺序，并且

- **有效性**：所有的复制体都应该取得稳定的进展。

IC 共识协议被设计为

- 非常简单，而且
- **稳健**：当一些复制体是恶意的時候，性能会优雅地下降。

如上所述，我们假设 $f < n/3$ 故障（即拜占庭）副本。另外，在部分同步的假设下，有效性是成立的，而安全性是有保证的，即使是在一个完全异步的网络中。

像一些共识协议一样，IC 共识协议是基于区块链的。随着协议的进行，一棵区块树被生长出来，从一个特殊的**创世区块**开始，它是该树的根。树上的每个非创世区块都包含（除其他外）一个**有效载荷**，由一连串组成，以及该区块在树上的父方的哈希值。诚实的副本对这棵树有一个一致的看法：虽然每个副本可能对这棵树有不同的、部分的看法，但所有的副本对同一棵树都有看法。此外，随着协议的进展，这棵树上总是有一条**最终完成的**区块的路径。同样，诚实的复制体对这一路径有一致的看法：虽然每个复制体对这一路径可能有不同的、部分的看法，但所有复制体对同一路径都有看法。沿着这个路径的块的有效载荷中的输入是有序的输入，将由互联网计算机的执行层处理。

该协议以**轮次进行**。在协议的**第 h 轮**，一个或多个**高度为 h** 的块被添加到树上。也就是说，在**第 h 轮**中加入的区块与根的距离总是恰好为 h 。在每一轮中，一个伪随机过程被用来给每个副本分配一个唯一的**等级**，这个等级是 $0, \dots, n-1$ 范围内的一个整数。这个伪随机过程是用一个**随机信标**来实现的（这利用了阈值签名，在上面提到的 **Section 1.6.1** 中提到的，并在 **Section 3**）.排名最低的副本是该轮的**领导者**。当领导者是诚实的并且网络是同步的，领导者将提出一个区块，该区块将被添加到树上；此外，这将是本轮中添加到树上的**唯一**区块，它将扩展最终的路径。如果领导者不诚实或网络不同步，其他一些等级较高的副本也可能提出区块，并将其区块添加到树上。在任何情况下，协议的逻辑都会给领导者提议的区块以最高的优先权，**一些或一些**区块会在这一轮被添加到这棵树上。即使协议进行了几轮而没有扩展最终确定的路径，树的高度也会随着每一轮的进行而继续增长，所以当最终确定的路径在**第 h 轮**被扩展时，最终确定的路径的长度将是 h 。

共识协议依靠数字签名来验证副本之间发送的信息。为了实现该协议，每个副本都与一个签名方案的公共验证密钥相关联。复制体与公共密钥的关联是从 **NNS** 维护的注册表中获得的。

2.3 信息路由

正如在第 1 章中所讨论的那样 **Section 1.7** 中所讨论的，IC 中的基本计算单元被称为 **"Canister"**。IC 提供了一个运行时环境来执行 Canister 里的程序，并与其他 Canister 和外部用户（通过消息传递）进行通信。

共识层将输入捆绑成**有效载荷**，这些有效载荷被放入**区块**中，随着区块的最终完成，相应的有效载荷被送到**消息路由**层，然后由**执行环境**处理，**执行环境**更新复制状态机上的 Canister 的状态并产生输出，而这些输出由**消息路由**层处理。

区分两种类型的输入是有用的。

入口信息：这些是来自外部用户的信息。

跨子网信息：这些是来自其他子网的 Canister 的信息。

我们还可以区分两种类型的产出。

ingress 消息/回应：这些是对 ingress 消息的回应（可能由外部用户检索）。

跨子网信息：这些是给其他子网的 Canister 的信息。

在收到来自共识的有效载荷后，该有效载荷中的输入被放置到各种**输入队列**中。对于在子网中运行的每个 Canister **C** 来说，有几个输入队列：一个用于向 **C** 输入消息，对于与 **C** 通信的每个其他 Canister **C'** 来说，一个用于从 **C'** 到 **C** 的跨子网消息。

在每一轮中，执行层将消耗这些队列中的一些输入，更新相关 Canister 的复制状态，并将输出放在各种**输出队列**中。对于在子网中运行的每个 Canister **C** 来说，有几个输出队列：对于每个与 **C** 通信的其他 Canister **C'** 来说，有一个用于从 **C** 到 **C'** 的跨子网消息。消息路由层将把这些输出队列中的消息，放入**子网到子网的流**中，由**跨网传输协议**处理，其工作是将这些消息实际传输到其他子网中。

除了这些输出队列外，还有一个**进站历史**数据结构。一旦一个入口信息被一个 Canister 处理，对该入口信息的**响应**将被记录在这个数据结构中。这时，提供入口信息的外部用户将能够检索到相应的响应。（注意，入口信息**历史**并不保持所有入口信息的完整历史。）

请注意，复制的状态包括 Canister 的状态，以及“系统状态”，包括上述的队列和流，以及入口的**历史**数据结构。因此，消息路由层和执行层都参与更新和维护子网的复制状态。至关重要的是，所有这些状态都是以完全**确定**的方式更新的，因此所有的复制都保持完全相同的状态。

还要注意的，共识层与消息路由和执行层是脱钩的，也就是说，共识区块链中的任何分叉都会在其有效载荷传递给消息路由之前得到解决，事实上，共识不必与消息路由和共识保持同步，允许提前一点运行。

2.3.1 每轮认证状态

在每一轮中，一个子网的**部分**状态将被**认证**。**每轮的认证状态**是使用链式钥匙加密法认证的。在其他方面，某一轮的认证状态包括

- 最近添加到子网至子网流中的**跨子网信息**。
- 其他元数据，包括**进站历史**数据结构。

每轮认证的状态是使用阈值签名认证的（见 Section 1.6.1）。每轮认证状态在 IC 中以几种方式使用。

- **输出认证**。跨子网消息和对入口消息的响应使用每轮认证状态进行认证。
- **防止和检测非决定性**。共识保证每个副本以相同的顺序处理输入。由于每个副本都以确定的方式处理这些输入，所以每个副本应该获得相同的状态。然而，IC 被设计成具有额外的稳健性，以防止和检测任何（意外的）非决定性计算，如果它出现的话。每轮认证的状态是用来做到这一点的机制之一。

- *与共识的协调*。每轮认证状态也被用来协调执行层和共识层，有两种不同的方式。
 - 如果共识运行在执行之前（其进度由最后一轮的状态得到认证的回合决定），共识将被“扼杀”。
 - 共识的输入必须通过某些有效性检查，而这些有效性检查可能取决于认证状态，所有副本都必须同意。

2.3.2 查询调用与更新调用

正如我们到目前为止所描述的那样，进站信息必须通过共识，以便它们被子网的所有复制体以相同的顺序处理。然而，对于那些处理过程不修改子网的复制状态的进站消息，有一个重要的优化。这些被称为**查询调用**--与其他入口消息相反，它们被称为**更新调用**。查询调用被允许执行读取和可能更新 Canister 状态的计算，但对 Canister 状态的任何更新都不会被提交到复制的状态。因此，一个查询调用可以直接由一个副本处理，而不需要通过共识，这大大降低了从查询调用中获得响应的延迟。

一般来说，对查询调用的响应不记录在入口历史数据结构中，因此不能使用上述的每轮认证状态机制进行认证。然而，IC 使 Canister 有可能将数据（在处理更新调用时）存储在特殊的认证变量中，这可以通过该机制进行认证；因此，返回认证变量作为其值的查询调用仍然可以被认证。

2.3.3 外部用户认证

进站信息和跨子网信息的主要区别之一是用于验证这些信息的机制。虽然链式密码学被用来验证跨子网消息，但一个不同的机制被用来验证来自外部用户的进站消息。

没有外部用户的中央登记处。相反，外部用户使用一个**用户标识符**来识别自己的身份，该标识符是一个公共签名验证密钥的哈希值。该用户持有一个相应的秘密签名密钥，用于签署进入的信息。这样的签名，以及相应的公钥，与入口信息一起发送。IC 自动验证签名，并将用户标识符传递给适当的 Canister。然后，Canister 可以根据用户识别码和进入信息中指定的操作的其他参数，授权所要求的操作。

首次使用的用户在与 IC 的第一次互动中产生一对密钥，并从公钥中得出他们的用户标识。返回的用户使用由用户代理存储的秘密密钥进行认证。一个用户可以使用签名授权，将几个密钥对与一个用户身份联系起来。这很有用，因为它允许一个用户使用同一个用户身份从几个设备访问 IC。

2.4 执行层

执行层一次处理一个输入。这个输入来自其中一个输入队列，并被引向一个 Canister。基于这个输入和 Canister 的状态，执行环境更新 Canister 的状态，另外还可以向输出队列添加消息，并更新入口历史（可能是对早期入口消息的回应）。

每个子网都可以访问一个**分布式伪随机发生器（PRG）**。伪随机位来自一个种子，该种子本身是一个阈值签名，称为**随机磁带**（见 Section 1.6.1 和更多的细节在 Section 3）。共识协议的每一轮都有一个不同的随机带。

随机带的基本属性是。

1. 在高度为 h 的区块被任何诚实的复制体最终确定之前，高度为 $h+1$ 的随机磁带被保证是不可预测的。
2. 当高度为 $h+1$ 的区块被任何一个诚实的副本最终确定时，该副本通常会拥有它所需要的所有股份来构建高度为 $h+1$ 的随机磁带。

为了获得伪随机位，一个子网必须在某一轮，比如说 h ，通过执行层的 "系统调用" 提出对这些位的请求，然后系统将在稍后使用高度为 $h+1$ 的随机磁带对该请求做出响应。根据上面的属性 (1)，可以保证请求的伪随机位在提出请求时是不可预测的。根据上面的属性 (2)，所请求的随机位通常在下一个区块被最终确定时是可用的。

2.5 把它放在一起

我们通过典型的流程来追踪处理 IC 上的用户请求。

1. 用户对 Canister C 的请求 M 由用户的客户端发送到一个边界节点（见 Section 1.9）边界节点将 M 发送到承载 Canister C 的子网中的一个副本。
2. 在收到 M 后，这个副本将利用对等层将 M 广播给子网上的所有其他副本（见 Section 2.1）。
3. 收到 M 后，下一轮共识的领导者（见 Section 2.2）将 M 与其他输入捆绑在一起，形成领导者提议的 B 块的有效载荷。
4. 一段时间后， B 块被最终确定，有效载荷被发送到消息路由层（见 Section 2.3）进行处理。请注意，对等层也是通过共识来最终确定这个块。
5. 消息路由层将把 M 放在 Canister C 的输入队列中。
6. 一段时间后，执行层（见 Section 2.4）将处理 M ，更新 Canister C 的内部状态。

在某些情况下，Canister C 将能够立即计算出对请求 M 的响应 R ，在这种情况下， R 被放在入口历史数据结构中。

在其他情况下，处理请求 M 可能需要向另一个筒子发出请求。在这个例子中，我们假设为了处理这个特定的请求 M ，Canister C 必须向居住在另一个子网的另一个 Canister C' 发出请求 M' 。这第二个请求 M' 将被放在 C 的输出队列中，然后执行以下步骤。

7. 一段时间后，消息路由将把 M' 移到一个适当的跨子网流中，这将最终被传送到承载 C' 的子网。
8. 在第二个子网，请求 M' 将从第一个子网获得，并最终通过第二个子网的共识和消息路由，然后被执行层处理。执行层将更新 Canister C 的内部状态，并生成对请求 M' 的响应 R' 。响应 R' 将进入 Canister C 的输出队列，最终被置于跨子网的流中，并传送回第一子网。
9. 回到第一子网，响应 R' 将从第二子网获得，并最终通过第一子网的共识和消息路由，然后被执行层处理。执行层将更新 Canister C 的内部状态，并产生对原始请求消息

M 的响应 R。这个响应将被记录在入口历史数据结构中。

无论采取哪种执行路径，对请求 M 的响应 R 最终将被记录在承载 Canister C 的子网的入口历史数据结构中。为了获得这个响应，用户的客户端必须执行一种 "查询调用" (见 Section 2.3.2)。正如在 Section 2.3.1 所述，该响应将通过链式密码学 (特别是使用阈值签名) 进行认证。认证逻辑本身 (即阈值签名验证) 将由客户使用客户最初从边界节点获得的服务工作者进行。

3 链式密码学 I：阈值签名

IC 的链式密码学的一个重要组成部分是阈值签名方案 [Des87]。IC 使用阈值签名的目的有很多。让 n 是子网中的副本数量，让 f 是对腐败副本数量的约束。

- 共识层利用一个 $(f+1)$ - n 个阈值的签名来实现 *随机信标* (见 Section 5.5)。
- 执行层利用一个 $(f+1)$ -out-of- n 的阈值签名来实现一个 *随机磁带*，它被用来向 Canister 提供不可预测的伪随机数 (见 Section 7.1)。
- 执行层利用一个 $(n - f)$ -out-of- n 阈值签名来 *认证复制的状态*。这既被用来验证子网的输出 (见 Section 6.1) 和实现 IC 的 *链式进化技术* 的 *快进功能* (见 Section 8.2)。

对于前两个应用 (随机信标和随机磁带)，阈值签名必须是 *唯一的*，也就是说，对于一个给定的公钥和信息，只有一个有效的签名。这是必须的，因为我们使用签名作为伪随机发生器的种子，所有计算这种阈值签名的副本必须同意同一种子。

3.1 阈值 BLS 签名

我们在 BLS 签名方案的基础上实现阈值签名。[BLS01] 的方式来实现，这对于适应阈值设置来说是很简单的。

普通的 (即非阈值的) BLS 签名方案使用两个组， G 和 G' ，都是质数 q 的。我们还假设有一个哈希函数 H_G 将其输入映射到 G' (并被模拟成一个随机神谕)。秘密签署密钥是一个元素 $x \in G$ ，公共验证密钥是 $V := g^x \in G'$ 。

在非阈值设置中，为了对信息 m 进行签名，签名者计算 $h^f \cdot H_G(m) \in G$ ，然后计算签名 $a := (h^f \cdot H_G(m))^x \in G$ 。为了验证这样的签名是否有效，我们必须测试 $\log_{\#} a = \log_{\#} V$ 。我们不能在这里讨论配对和椭圆曲线的细节。详情见 [BLS01] 以了解更多细节。BLS 签名有一个很好的特性 (上面提到的)，即签名是唯一的。

在 t -out-of- n 阈值设置中，我们有 n 个副本，其中任何一个 t 都可以用来生成一个消息的签名。在一些细节上，每个副本 p_j 持有秘密签名密钥 $x \in G$ 的一个份额 $x_j \in G$ ，它由 p_j 私人持有，而组元素 $v_j := g^{x_j}$ 是公开的。共享 (x_1, \dots, x_n) 是 x 的 t -out-of- n 秘密共享 (见 Section 3.4)。

给定一个信息 m ，副本 p_j 可以生成一个 **签名共享**

$$a_j := (h^f \cdot H_G(m))^{x_j} \in G',$$

其中 $h^f = H_G(m)$ 如前。为了验证这样的签名共享是有效的，我们必须测试 $\log_{\#} a_j = \log_{\#} v_j$ 。这可以通过配对来完成，如上所述--事实上，这与普通的 BLS 签名与公共密钥 V_j 的有效性

测试完全相同。

这个方案满足以下**重建属性**。

给予消息 m 上的任何 t 个有效签名份额 σ_j 的集合（由不同的副本贡献），我们可以在公共验证密钥下有效地计算出 m 上的有效 **BLS** 签名 σ 。

事实上， σ 可以被计算为

$$\sigma \leftarrow \prod_j \sigma_j$$

其中， σ_j 的计算可以有效地从 t 个贡献副本的索引中获得。

在 G 的合理难行性假设下，并将 H 建模为一个随机神谕，该方案满足以下**安全属性**。

假设最多只有 f 个复制体可能被对手破坏。那么，除非对手从至少 $t-f$ 个诚实的副本中获得该消息的签名份额，否则要计算出该消息的有效签名是不可行的。

3.2 分布式密钥分发

为了实现**阈值 BLS**，我们需要一种方法来分配秘密签名钥匙的份额给各个副本。一种方法是让**受信任的一方**直接计算所有这些份额并将其分配给所有的副本。不幸的是，这将产生一个单点故障。相反，我们使用**分布式密钥生成 (DKG)** 协议，它允许副本基本上使用安全的分布式协议来执行这样一个受信任方的逻辑。

我们简要介绍一下目前实施的协议的高层次想法。我们请读者参阅 [Gro21] 以了解更多细节。所使用的 **DKG** 协议本质上是**非交互式的**。它使用两个基本要素。

- 一个可公开验证的**秘密共享 (PVSS)** 方案，以及
- 一个**共识协议**。

尽管可以使用任何共识协议，但毫不奇怪的是，我们使用的是在 Section 5 (另见 Section 8)。

3.3 假设

所做的基本假设与《中国社会科学》中概述的相同。Section 1:

- 异步通信，以及
- $f < n/3$ 。

我们只是间接地利用了部分同步假设（如在 Section 5.1）来确保共识协议达到失效性。

我们还假设，对于一个 t -out- n 的阈值签名方案，我们有

$$f < t < n - f。$$

这（除其他外）确保（1）腐败的副本不能自己全部签名，以及（2）诚实的副本可以自己全部签名。

我们还假设每个副本都与一些公钥有关，其中每个副本也持有相应的私钥。其中一个公钥是签署密钥（与在 Section 5.4). 另一个公钥是实现 PVSS 方案所需的特定公钥加密方案的公开加密密钥（详情见下文）。

3.4 PVSS 计划

让 G 是上面介绍的由 $g \in G$ 生成的质数 q 的群。让 $s \in \mathbb{Z}_q$ 是一个秘密。回顾一下， s 的 t -out-of- n Shamir 秘密共享是一个向量 (s_i, \dots, s_n) ，其中

$$s_j := a(j) \quad (j = 1, \dots, n)。$$

和

$$a(x) := a_0 + a_1x + \dots + a_{t-1}x^{t-1} \in \mathbb{Z}_q[x]。$$

是一个度数小于 t 的多项式， $a_0 := s$ 。这种秘密共享的关键属性是

- 从任何 t 个 s_j 的集合中，我们可以有效地计算（通过多项式插值）秘密 $s = a_0 = a(0)$ ，并且
- 如果 a_1, \dots, a_{t-1} 是在 \mathbb{Z}_q 上统一独立选择的，那么任何少于 t 的 s_j 的集合都不会泄露关于秘密 s 的信息。

在高水平上，PVSS 方案允许一个副本， P_i ，称为**庄家**，采取这样的共享，并计算出一个称为**交易**的对象，其中包括

- 一个群组元素的向量 (A_0, \dots, A_{t-1}) ，其中 $A_k := g^{a_k}$ 为 $k = 0, \dots, t-1$ ，
- 一个密码文本的向量 (c_1, \dots, c_n) ，其中 c_j 是根据 P_j 的公开加密密钥对 s_j 进行的加密。
- 一个非交互式的零知识证明 π ，证明每个 c_j 确实加密了这样一个份额--更确切地说，每个 c_j 都解密了满足以下条件的值 s_j

$$s_{gj} = \pi \prod_{k=0}^{t-1} A_k = g^{a(j)}。 \quad (2)$$

我们注意到，为了建立我们的 DKG 协议的整体安全性，PVSS 方案必须提供适当级别的选择密码文安全。具体来说，庄家必须将其身份作为**关联数据**嵌入交易中，而加密的股份必须

即使在选择的密码文本攻击下，即允许对手解密任意的交易，这些交易在与创建交易的相关数据不同的情况下被解密。

如果不太关心效率，实现 PVSS 方案是很容易的。我们的想法是使用类似 ElGamal 的加密方案对每个 s_j 进行逐位加密，然后使用一个标准的非交互式零知识证明来证明关系 (2) 这将是基于菲亚特-沙米尔变换的标准应用（见 [FS86]）到一个适当的 Sigma 协议（见 [CDS94]）。虽然这产生了一个多项式时间方案，但它并不那么实用。然而，有许多可能的方法来优化这种类型的方案。参见 [Gro21] 关于 IC 中使用的高度优化的 PVSS 方案的细节。

3.5 基本的 DKG 协议

使用 PVSS 方案和一个共识协议，基本的 DKG 协议非常简单。

1. 每个复制体向所有其他复制体广播一个随机秘密的**签名处理**。

这样的签名交易包括一个交易，以及交易员的身份和在交易员的公共签名密钥下的签名。

如果一个有签名的交易具有正确的语法形式，并且签名和非交互式零知识证明都是有效的，那么这样的交易就被称为**有效的**。

2. 利用共识，各副本就 $t+1$ 个有效的签名交易（来自不同的交易商）的集合 S 达成一致。

3. 假设集合 S 中的第 i 个交易包含组元素的向量 $(A_{i,0}, A_{i,t-i})$ 和密码文本的向量 $(c_{i,j})$ 。

那么阈值签名方案的公共验证密钥为

$$V := \prod_{i=0}^t A_{i,0}^{A_{i,t-i}}$$

请注意，秘密签署密钥被隐含地定义为

$$X := \log_g V。$$

p_j 对秘密签名密钥 x 的份额是

$$x_j =$$

其中 $s_{i,j}$ 是根据 p_j 的秘密解密密钥对 $c_{i,j}$ 进行的解密。复制体 P 的公开验证密钥 V_j 是

$$V_j := \prod_{i=0}^{t-i} A_{i,0}^{A_{i,t-i}} \prod_{k=0}^{n-j} c_{i,k}^{s_{i,k}}$$

请注意，股份 x_j 包括 x 的 t -out-of- n Shamir 秘密分享。(1)中出现的 X_j 值只是拉格朗日插值的系数。这就确立了在以下内容中所述的**重建属性** Section 3.1.至于中所述的安全属性 Section 3.1 中所述的安全属性，可以证明将 $H(G)$ 作为随机信使，并假设 PVSS 方案是安全的，并且群组 G 和（有配对）满足某种类型的一更**Diffie-Hellman** 硬度假设，这可以说是没

有有效对手可以以不可忽略的概率赢得以下游戏。

挑战者随机选择 $p_1, \dots, p_k \in \mathbb{Z}_q$ 和 $\{g^{p_i} \}_{i=1}^k$ 给对手。

$\in \mathbb{Z}_q$ 的随机性，并给出

对手向挑战者发出一连串的询问，每一个询问都是一个 $\{K_{i,j}\}_{i,j}$ 形式的向量，挑战者的回答是

$$H(g^{p_i})^{x_{i,j}}。$$

为了结束游戏，对手输出一个向量 $\{X_{i,j}\}_{i,j}$ 和一个组元素 $h \in G$ ，并在以下情况下赢得游戏

$$h = \prod_{i,j} \left((g^{p_i})^{x_{i,j}} \right)^{\lambda_{i,j}}$$

而输出向量 $\{X_{i,j}\}_{i,j}$ 不是查询向量的线性组合。

虽然在 $t > f+1$ 的情况下需要这种更多的 Diffie-Hellman 假设，但当 $t = f+1$ 时，我们可以用一个更弱的假设来解决（所谓的 co-CDH 假设，普通 BLS 方案的安全性就是基于此）。

3.6 一个重新分享的协议

基本的 DKG 协议可以很容易地被修改，以便它不是创建一个新的随机秘密 x 的共享，而是创建一个新的、随机的先前共享的共享。

- 基本协议的第 1 步被修改，以便每个副本广播其现有份额的签名交易。
- 步骤 2 被修改，以便商定一组有效的签名交易。同时，每个交易被验证，以确保它确实是适当的现有股份的交易（这意味着第 1 次交易的价值应等于旧价值 oE ）。
- 在步骤 3 中，新的 x_i （和 v_j ）值的计算对 i 个拉格朗日插值系数的和（和积）进行加权。

4 点对点层

对等层的任务是在子网中的副本之间传输协议消息。这些协议消息包括

- 用于实现共识的信息，例如，区块建议、公证等（见 Section 5）；
- 入境信息（见 Section 6）。

基本上，点对点提供的服务是一个“尽力而为”的广播渠道：如果一个诚实的副本广播了一个消息，那么这个消息最终会被子网中所有诚实的副本收到。

设计目标包括以下内容。

- **有界限的资源。**所有的算法都必须在有限的资源（内存、带宽、CPU）下工作。
- **优先次序。**根据某些属性（如类型、大小、轮次），不同的信息可能会有不同的优先级

处理，而且这些优先级可能会随着时间的推移而改变。

- **效率。**高吞吐量比低延时更重要。
- **DOS/SPAM 的弹性。**腐败的复制体应该阻止诚实的复制体相互之间的通信。

请注意，在共识协议中，一些消息，特别是块建议（可能相当大），将被所有副本重新广播。这对于确保该协议的正确行为是必要的。然而，如果天真地实施，这将对资源的巨大浪费。为了避免所有复制体广播相同的消息，点对点层利用了**广告-请求-交付**机制。它不直接广播一个（大）消息，而是为该消息广播一个（小）**广告**：如果一个副本收到这样的广告，还没有收到，并且认为该消息是重要的，它将**请求**该消息被**传递**。这种策略以更高的延迟为代价降低了带宽利用率。对于小的消息，这种权衡是不值得的，直接发送消息而不是广告更有意义。

对于相对较小的子网，希望广播消息的复制体将向子网中的所有复制体发送广告，然后每个复制体都可以请求消息被传递。对于较大的子网，这种广告-请求-传递的机制可以在一个覆盖**网络**上运行。覆盖网络是一个连接的无向图，其顶点由子网中的复制体组成。如果在这个图中有一条连接它们的边，那么两个复制体就是**对等体**，复制体只与它的对等体进行通信。因此，当一个复制体希望广播一个消息时，它向其对等体发送该消息的广告。这些对等体可以请求传递该消息，在收到该消息后，如果满足某些条件，这些对等体将向其对等体宣传该消息。这实质上是一个八卦**网络**。这种策略再次降低了带宽利用率，但代价是更高的延迟。

5 共识层

IC 的共识层的工作是对输入进行排序，以使子网中的所有副本都以相同的顺序处理这些输入。对于这个问题，文献中有许多协议。IC 使用了一种新的共识协议，这里对其进行了高度描述。更多的细节，请参见该论文 [CDH+21](特别是该论文中的 ICC1 协议)。

任何安全的共识协议都应该保证两个属性，这两个属性（大致上说）是。

- **安全性**：所有的复制体实际上都同意相同的输入顺序，并且
- **有效性**：所有的复制体都应该取得稳定的进展。

这篇论文 [CDH+21]证明了 IC 共识协议满足上述两个特性

IC 共识协议被设计为

- 非常简单，而且稳健：当一些复制体是恶意的時候，性能会逐渐地下降。

5.1 假设

正如介绍中所讨论的，我们假设

- 一个有 n 个副本的子网，和
- 最多只有 $f < n/3$ 的复制是有缺陷的。

有问题的复制体可能会表现出任意的、恶意的（即拜占庭式的）行为。

我们假设通信是异步的，对复制体之间发送消息的延迟没有先验的约束。事实上，消息

传递的时间安排可能完全在对抗者的控制之下。IC 共识协议在这种非常弱的通信假设下保证了安全性。然而，为了保证有效性，我们需要假设一种部分同步的形式，（粗略地说）网络将在短时间内 Cycle 性地同步。在这样的同步间隔中，所有未交付的信息将在少于时间 5Δ 的时间内交付，对于一些固定的边界 5Δ 。边界 5Δ 不必事先知道（协议初始化时有一个合理的边界，但如果这个边界太小，将动态地适应和增加这个边界）。无论我们假设的是异步网络还是部分同步网络，我们都假设每个从一个诚实的副本发送到另一个的消息最终都会被传递。

5.2 协议概述

像一些共识协议一样，IC 共识协议是基于区块链的。随着协议的进行，一棵区块树被生长出来，从一个特殊的创世区块开始，它是该树的根。树上的每个非创世区块都包含（除其他外）一个有效载荷，由一连串输入组成，以及该区块在树上的父方的哈希值。诚实的复制体对这棵树有一个一致的看法：虽然每个复制体可以

有一个不同的、部分的树的视图，所有的副本有一个相同的树的视图。此外，随着协议的进展，在这棵树上总有一条最终完成的区块路径。同样，诚实的复制体对这一路径有一致的看法：虽然每个复制体对这一路径可能有不同的、部分的看法，但所有的复制体对同一路径都有看法。沿着这个路径的区块有效载荷中的输入是有序的输入，将由互联网计算机的执行层处理（见 Section 7）。

该协议以轮次进行。在协议的第 h 轮，一个或多个高度为 h 的区块被添加到树上。也就是说，在第 h 轮中加入的区块与根的距离总是恰好为 h 。在每一轮中，一个伪随机过程被用来给每个副本分配一个独特的等级，它是一个范围在 1 的整数。这伪随机过程是用一个随机信标来实现的（见 Section 5.5 见下文）。排名最低的副本是该轮的领导者。当领导者是诚实的并且网络是同步的，领导者将提出一个区块，这个区块将被添加到树上；此外，这将是本轮中添加到树上的唯一区块，它将扩展最终的路径。如果领导者不诚实或网络不同步，其他一些等级较高的副本也可能提出区块，并将其区块添加到树上。在任何情况下，协议的逻辑都会给领导者提议的区块以最高的优先权，一些或一些区块会在这一轮被添加到这棵树上。即使协议进行了几轮而没有扩展最终确定的路径，树的高度也会随着每一轮的进行而继续增长，所以当最终确定的路径在第 h 轮被扩展时，最终确定的路径的长度将是 h 。

5.3 额外属性

IC 共识协议还享有一个额外的属性（就像 PBFT [CL99] 和 HotStuff [AMN+20] 不同于其他协议，如 Tendermint [BKM18]）的另一个特性是乐观的响应性 [PS18] 这意味着，当领导者是诚实的，协议可以按照实际网络延迟的速度进行，而不是按照网络延迟的某个上限进行。

我们注意到，IC 共识协议的简单设计也确保了在拜占庭故障实际发生时，其性能会相当优雅地下降。如同在 [CWA+09] 中指出的，最近关于共识的许多工作都集中在改善没有故障的“乐观情况”下的性能，以至于所产生的协议非常脆弱，当故障发生时可能变得几乎无法使用。例如，[CWA+09] 本文表明，在某些类型的（相当简单的）拜占庭行为下，PBFT 的现有实现的吞吐量下降到零。这篇论文 [CWA+09] 主张采用稳健的共识，即在最佳条件下的峰值性能被部分牺牲，以确保在某些当事方实际腐败时的合理性能（但仍假设网络是同步的）。IC 共识协议在以下意义上确实是稳健的 [CWA+09]：在领导者腐败的任何一轮中（这种情况本身发生的概率小于 $1/3$ ），该协议将有效地允许另一方接替该轮的领导者，几

乎不费吹灰之力，使协议及时地进入下一轮。

5.4 公钥

为了实现该协议，每个副本都与 **BLS** 签名方案的公钥相关联。[BLS01]，每个副本也持有相应的秘密签名密钥。复制体与公钥的关联是由 **NNS** 维护的注册表获得的（见 Section 1.5）。这些 **BLS** 签名将被用来验证副本发送的消息。

该协议还使用了 **BLS** 签名的签名聚合功能。[BGLS03]的签名聚合功能，它允许将同一消息上的许多签名聚合成一个紧凑的多重签名。该协议将使用这些多重签名进行公证（见 Section 5.7）和最终确认（见 Section 5.8），这是某种形式的信息上的 $n-f$ 个签名的聚合。

5.5 随机信标

除了上面讨论的 **BLS** 签名和多重签名外，该协议还利用 **BLS** 阈值签名方案来实现上述的随机信标。高度为 h 的随机信标是高度为 h 的唯一消息上的 $(f+1)$ 阈值签名。在协议的每一轮中，每个副本为下一轮广播它的信标份额，所以当下一轮开始时，所有副本应该有足够的份额来重构该轮的信标。如上所述，高度为 h 的随机信标被用来给每个副本分配一个伪随机等级，该等级将在协议的 h 轮中使用。由于阈值签名的安全特性，对手将无法提前一轮以上预测副本的排名，这些排名将有效地与随机一样好。参见 Section 3 了解更多关于 **BLS** 阈值签名的信息。

5.6 砌块制作

每个副本都可以在不同的时间点上扮演一个**区块制造者**的角色。作为第 h 轮的区块制造者，副本提出一个高度为 h 的区块 B ，作为区块树中高度为 $h-1$ 的区块 fB 的孩子。为此，区块制造者首先收集了一个由它所知道的所有输入组成的**有效载荷**（但不包括那些已经包含在以 B 为终点的区块树路径中的有效载荷）。区块 B 由以下部分组成的：

- 有效载荷。
- B 的哈希值。
- 挡板制作者的等级。
- 区块的高度 h 。

在形成区块 B 之后，区块制作者形成了一个**区块提案**，由以下内容组成

- 区块 B 。
- 区块制造者的身份，以及
- 区块制造者在 B 上的签名。

一个区块制造者将向所有其他副本广播其区块提议。

5.7 公证

当一个区块被公证时，它被有效地添加到区块树中。一个区块要成为公证对象，必须有 $n-f$ 个不同的副本支持其公证。

给定一个高度为 h 的提议区块 B ，副本将确定该提议是否有效，这意味着 B 具有上述的句法形式。特别是， B 应该包含高度为 h 的区块 B 的哈希值，¹ 该区块已经在区块树中（即已经被公证）。此外， B 的有效载荷必须满足某些条件（特别是，有效载荷中的所有输入必须满足各种约束，但这些约束通常与共识协议无关）。另外，区块制造者的等级（如记录在区块 B 中）必须与随机信标在第 h 轮分配给提议该区块的副本的等级（如记录在区块提议中）一致。

如果区块是有效的，并且某些其他约束条件成立，副本将通过广播 B 的公证份额来支持区块的公证，其中包括

- B 的哈希值。
- B 的高度为 h 。
- 支持复制品的身份，以及
- 支持复制品在包含 B 的哈希值和高度 h 的信息上的签名。

任何一组关于 B 的 $n-f$ 个公证份额都可以汇总在一起，形成对 B 的公证，其中包括

- B 的哈希值。
- B 的高度为 h 。
- 是指 $n-f$ 个支持性复制的身份集合。
- 由 B 的哈希值和高度 h 组成的信息上的 $n-f$ 签名的集合。

一旦一个副本获得了高度为 h 的公证区块，它将完成 h 轮，随后将不支持任何其他高度为 h 的区块的公证。在这个时候，这样的副本也将把这个公证转达给所有其他副本。请注意，这个副本可能是通过（1）从另一个副本那里获得的公证，或者（2）聚合它所收到的 $n-f$ 个公证份额。

增长不变量指出，每个诚实的副本最终将完成每一轮并开始下一轮，因此，公证区块的树继续增长（这只假设异步的最终交付，而不是部分同步）。我们在下面证明增长不变性（见 Section 5.11.4）。

5.8 定稿

然而，如果一个区块被最终确定，那么我们可以肯定，在高度为 h 的地方没有其他被公证的区块。让我们把这称为安全不变量。

对于一个区块来说，必须有 $n-f$ 个不同的副本支持它的最终确定。回顾一下，当一个副本获得高度为 h 的公证区块 B 时， h 轮结束。在那个时间点上，这样的副本将检查它是否支持高度为 h 的任何区块的公证，而不是区块 B （它可能支持也可能不支持 B 本身的公证）。如果不支持，复制体将通过广播 B 的终结份额来支持 B 的终结。终结份额的格式与公证份额完全相同（但其标记方式是公证份额和终结份额不能相互混淆）。 B 上的任何 $n-f$ 个最终确认份额的集合可以被聚集在一起，形成 B 的最终确认，其格式与公证完全相同（但同样，被适当地标记）。任何获得定稿区块的副本将向所有其他副本广播该定稿。

我们在下面证明安全不变量（见 Section 5.11.5）。安全不变量的一个结果是下面的。假设两个区块 B 和 B' 被最终确定，其中 B 的高度为 h ， B' 的高度为 $h' < h$ ，那么安全不变式意味着在公证区块树中以 B' 为终点的路径是以 B 为终点的路径的前缀（如果不是，那么就会有两个高度为 h 的公证区块，与最终确定不变式相矛盾）。因此，每当一个副本看到一个终结的区块 B 时，它可以把 B 的所有祖先看作是隐性终结的，而且由于安全不变性，安全属性被保证对这些（显性和隐性）终结的区块是成立的--也就是说，所有副本对这些终结的区块的排序是一致的。

5.9 延迟功能

该协议使用了两个延迟函数 A_m 和 A_n ，它们控制了区块制作和公证活动的时间。这两个函数都将提议副本的等级 r 映射为非负的延迟量，并且假定每个函数在 r 中是单调增加的，并且 $A_m(r) < A_n(r)$ ，对于所有 $r=0, \dots, n-1$ 。这些函数的推荐定义是 $A_m(r)=25r$ 和 $A_n(r)=25r+e$ ，其中 $A_m(r)$ 是 5 将信息从一个诚实副本传递到另一个诚实副本的时间的上限， $e>0$ 是一个 "治理者"，用于防止协议运行过快。有了这些定义，在那些(1)领导者是诚实的，和(2)消息真的在时间 5 之内在诚实的副本之间传递的回合中，将确保有效性。事实上，如果(1)和(2)在某一轮中同时成立，那么领导者在该轮中提出的区块将被最终确定。我们把这称为 "有效性不变"。我们在下面证明这一点（见 Section 5.11.6）。

5.10 一个例子

Figure 2 说明了一个区块树。每个区块都标有其高度（30, 31, 32, ...）和是其区块制造者的等级。图中还显示，树上的每个区块都经过了公证，如 \textcircled{R} 符号所示。这至少要有 $n-f$ 个不同的副本支持其公证。正如人们所看到的，可以有更多的意味着，对于树上的每一个经过公证的块，在

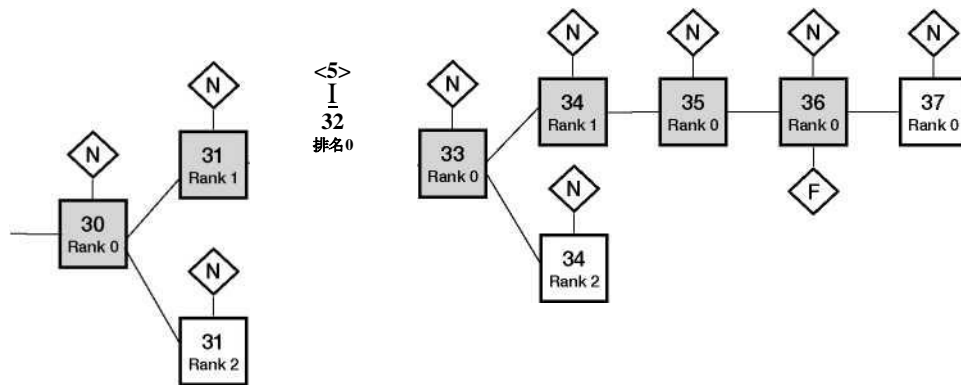


图 2：一个区块树的例子

在一个特定的高度，树上有一个以上的公证区块。例如，在高度 32，我们看到有两个经过公证的区块，一个由等级 1 和 2 的区块制造者提出。同样的事情也发生在高度为 34 的地方。我们还可以看到，高度为 36 的区块也被明确地最终确定了，如◆符号所示。这意味着有 $n-f$ 个不同的副本支持这个区块的最终确定，这意味着这些副本（或者至少是这些副本中的诚实副本）并不支持任何其他区块的公证。这个区块的所有祖先，也就是灰色阴影部分，都被认为是隐含的终结。

5.11 将所有的东西放在一起

我们现在更详细地描述协议是如何工作的；具体来说，我们更精确地描述一个副本何时提出一个区块，以及一个副本何时支持一个区块的公证。一个给定的副本 P 将记录它进入一个给定回合 h 的时间，这发生在它获得（1）高度为 $h-1$ 的区块的一些公证，以及（2）回合 h 的随机信标的时候。

5.11.1 随机信标细节

一旦一个副本收到 h 轮的随机信标，或者有足够份额来构建 h 轮的随机信标，它将把 h 轮的随机信标转发给所有其他副本。一旦一个副本进入第 h 轮，它将在第 $h+1$ 轮生成并广播其份额的随机信标。

5.11.2 制块细节

复制体 P 只会提出自己的区块 B_p ，前提是：（1）自本轮开始以来，至少已经过了 $A_{(mp)}$ 个时间单位；（2） P 目前还没有看到有效的低等级区块。

请注意，由于 P 在进入第 h 轮时保证有一个高度为 $h-1$ 的公证区块，它可以将其提议的区块作为这个公证区块（或它可能有的任何其他高度为 $h-1$ 的公证区块）的子块。还要注意，当 p 广播它对 B_p 的提议时，它还必须确保它也将 B_p 的父代的公证转达给所有的副本。

假设一个复制体 Q 从一个等级为 $rp < TQ$ 的复制体 P 那里看到一个有效的区块提议，并且

(1)从本轮开始到现在至少已经过了 $A_m(r_p)$ 个时间单位, 并且(2)Q 目前没有看到等级小于 r_p 的区块。那么在这个时间点, 如果它还没有这样做, Q 将把这个区块提议 (连同提议区块的父代的公证) 转发给所有其他复制体。

5.11.3 公证细节

复制体 P 将支持等级为 τ_Q 的复制体 Q 提出的有效区块 B_Q 的公证, 条件是: (1)自本轮开始以来, 至少经过了 $A_H(\tau_Q)$ 个时间单位; (2)P 目前没有看到等级小于 τ_Q 的区块。

5.11.4 增长不变性的证明

增长不变性指出, 每个诚实的副本最终将完成每一轮并开始下一轮。假设所有的诚实副本都开始了第 h 轮。让 r^* 为第 h 轮中排名最低的诚实副本 P^* 的排名。最终, P^* 将 (1) 提出自己的区块, 或者 (2) 转述一个由排名较低的副本提出的有效区块。在任何一种情况下, 一些区块最终必须得到所有诚实副本的支持, 这意味着一些区块将被公证, 所有诚实副本将完成 h 轮。1.

5.11.5 安全不变性的证明

安全不变性指出, 如果一个区块在某一轮中被最终确定, 那么在该轮中就没有其他区块可以被公证。这里有一个安全不变性的证明。

1. 假设损坏的复制数量正好是 $f^* < f < n/3$ 。
2. 如果一个区块 B 被最终确定, 那么它的最终确定必须得到至少由 $n - f - f^*$ 诚实副本组成的集合 S 的支持 (根据集合签名的安全属性)。
3. 假设 (通过矛盾的方式) 另一个区块 $B' \neq B$ 被公证了。那么它的公证一定是由一组至少有 $n - f - f^*$ 诚实副本的 S' 支持的 (同样, 根据集合签名的安全属性)。
4. 集合 S 和 S' 是不相交的 (通过最终化逻辑)。
5. 因此, $n - f^* > |S \cup S'| = |S| + |S'| > 2(n - f - f^*)$, 这意味着 $n < 3f$, 这是一个矛盾。

5.11.6 有效性不变量的证明

如果所有在时间 t 或之前由诚实副本发送的消息在时间 t 之前到达目的地, 我们就说网络在时间 t 是 Δ 同步的。

有效性不变量可以表述如下。假设 $A_n(1) > A_m(0) + 25$ 。还假设在一个给定的回合 h 中, 我们有

- h 轮中的领导者 P 是诚实的。
- 第一个进入 h 轮的诚实复制品 Q 在时间上是这样的, 并且
- 该网络在 t 和 $t + 5A_m(0)$ 时是 5 同步的。

然后, P 在第 h 轮提出的区块将被最终确定。

这里有一个有效性不变量的证明。

1. 在时间 t 的部分同步下，所有诚实的副本将在时间 $t+5$ 之前进入 h 轮（对 Q 来说，结束 $h-1$ 轮的公证以及 h 轮随机的随机信标将在这个时间之前到达所有诚实的副本）。
2. h 轮的领导者 P 将在时间 $t+5+A_m(0)$ 之前提出一个区块 B ，同样通过部分同步，这个区块提议将在时间 $t+25+A_m(0)$ 之前传递给所有其他副本。
3. 由于 $A_n(1) > A_m(0) + 25$ ，协议逻辑保证每个诚实的副本都支持 B 区块的公证，而不支持其他区块的公证，因此 B 将成为公证和最终确定的。

5.12 其他问题

5.12.1 增长延时

在部分同步的假设下，我们也可以制定和证明一个定量版本的增长不变性。为简单起见，假设延迟函数的定义如上文所建议。最后，假设网络在 $[t, t + (3r + 2)5]$ 区间的所有时间都是 5 同步的。那么，所有诚实的副本将在时间 $t+3(r+1)5$ 之前开始 $h+1$ 轮。

5.12.2 局部调整的延迟功能

当一个副本在几轮中没有看到任何最终确定的区块时，它将开始增加自己的延迟函数 A_n 用于公证。复制体不需要就这些本地调整的公证延迟函数达成一致。

另外，虽然复制体没有明确地调整延迟函数 A_p ，但我们可以通过局部调整两个延迟函数来建立局部时钟漂移的数学模型。

因此，有许多延迟函数，以副本和回合为参数。那么，灵活性所需的关键条件 $A_n(1) > A_m(0) + 25$ 就变成了 $\max A_n(1) > \min A_m(0) + 25$ ，其中 \max 和 \min 是在给定回合的所有诚实副本中取值。因此，如果最终确认在足够多的回合中失败，所有诚实的副本最终会增加他们的公证延迟，直到这一点成立，然后最终确认将恢复。如果一些诚实的复制体比其他复制体更多地增加他们的公证延迟函数，那么在有效性方面没有惩罚（但在增长延迟方面可能有惩罚）。

5.12.3 公平性

另一个在共识协议中很重要的属性是**公平性**。我们没有给出一个一般的定义，而是简单地观察到，有效性不变性也意味着一个有用的公平性属性。回顾一下，有效性不变性基本上是说，在领导者是诚实的、网络是同步的任何一轮中，领导者提出的区块将被最终确定。在那些发生这种情况的回合中，领导者是诚实的这一事实确保它将在其块的有效载荷中包括它所知道的所有输入（对有效载荷大小的限制）。因此，非常粗略地讲，任何被传播到足够多的副本的输入都会在合理的时间内以高概率的方式被包含在最终的区块中。

6 信息路由层

正如在第 1.7 节中所讨论的 Section 1.7, IC 中的基本计算单元被称为“程序罐 (canister)”，这与**进程**的概念大致相同，因为它包括一个**程序**和它的**状态**。IC 提供了一个运行时环境来执行 Canister 中的程序，并与其他 Canister 和外部用户（通过消息传递）进行通信。

共识层（见 Section 5）将输入捆绑成有效载荷，这些有效载荷被放置在区块中，随着区块的最终完成，相应的有效载荷被传递到消息路由层，然后由执行环境处理，执行环境更新复制状态机上的 Canister 的状态并产生输出，这些输出由消息路由层处理。

区分两种类型的输入是有用的。

入口信息：这些是来自外部用户的信息。

跨子网信息：这些是来自其他子网的 Canister 的信息。

我们还可以区分两种类型的产出。

ingress 消息回应：这些是对 ingress 消息的回应（可能由外部用户检索）。

跨子网信息：这些是给其他子网的 Canister 的信息。

在收到来自共识的有效载荷后，该有效载荷中的输入被放置到各种**输入队列**中。对于在子网中运行的每个 Canister **C** 来说，有几个输入队列：一个用于向 **C** 输入消息，对于与 **C** 通信的每个其他 Canister **C** 来说，一个用于从 **CL 到 C** 的跨子网消息

正如下面详细描述的那样，在每一轮中，执行层将消耗这些队列中的一些输入，更新相关 Canister 的复制状态，并且

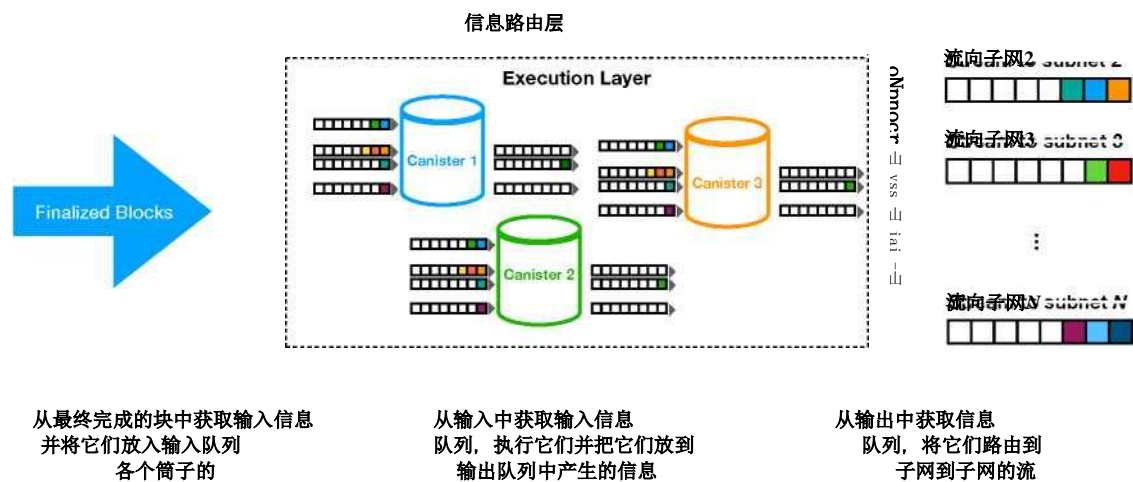


图 3：消息路由和执行层

将输出放在各种输出队列中。对于在子网中运行的每个 Canister C 来说，有几个输出队列：对于与 C 通信的每个其他 Canister C' 来说，一个用于从 C 到 C' 的跨子网消息。消息路由层将把这些输出队列中的消息，放到子网到子网的流中，由跨网传输协议来处理，其工作是将这些消息实际传输到其他子网。

除了这些输出队列外，还有一个进站历史数据结构。一旦一个入口信息被一个 Canister 处理，对该入口信息的响应将被记录在这个数据结构中。这时，提供入口信息的外部用户将能够检索到相应的响应。(注意，入口信息历史并不保持所有入口信息的完整历史。)

我们还应该提到，除了跨子网消息，还有子网内消息，即从一个 Canister 到同一子网的另一个 Canister 的消息。消息路由层将这类消息直接从输出队列转移到相应的输入队列。

Figure 3 说明了消息路由和执行层的基本功能。

请注意，复制的状态包括 Canister 的状态，以及 "系统状态"，包括上述的队列和流，以及入口的历史数据结构。因此，消息路由层和执行层都参与更新和维护子网的复制状态。至关重要的是，所有这些状态都是以完全确定的方式更新的，因此所有的复制都保持完全相同的状态。

还要注意的是，共识层与消息路由和执行层是脱钩的，也就是说，共识区块链中的任何分叉在其有效载荷被传递到消息路由之前就被解决了，事实上，共识不需要保持在

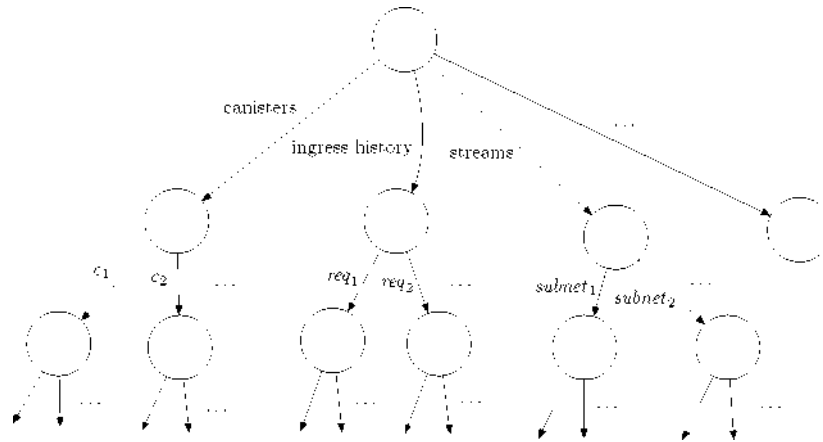


图 4：以树状方式组织的每轮认证状态

锁定消息路由和共识，并允许提前一点运行。

6.1 每轮认证状态

在每一轮中，一个子网的 *部分* 状态将被 *认证*。每轮认证的状态是使用链式钥匙加密法认证的（见 Section 1.6）具体来说，就是使用在第 6 节中提到的 $(n-f)-n$ 中的阈值签名方案。Section 3 更详细地说，在每个副本为给定回合生成每轮认证状态后，它将生成相应阈值签名的份额，并将其广播给其子网中的所有其他副本。在收集了 $n-f$ 个这样的份额后，每个副本可以构建出结果的阈值签名，作为该轮的每轮认证状态的 *证书*。请注意，在签名之前，每轮认证状态被散列为 **Merkle 树** [Mer87]。

在某一回合中，每一回合的认证状态包括

1. 最近添加到子网至子网流中的 *跨子网信息*。
2. 其他元数据，包括 *进站历史* 数据结构。
3. 是前一轮的每轮认证状态的 *Merkle 树根哈希值*。

请注意，每轮认证的状态 *不包括* 子网的整个复制状态，因为这通常是相当巨大的，在每轮认证 *所有的* 状态是不现实的。⁶

Figure 4 说明了如何将每轮认证状态组织成一棵树。树的第一个分支存储关于每个 Canister 的各种元数据（但不是 Canister 的整个复制状态）。第二个分支存储入口 *历史* 数据结构。第三个分支存储关于子网到子网流的信息，包括每个流的最近添加的跨子网消息的“窗口”。其他分支存储其他类型的元数据，这里不讨论。然后，这个树状结构可以被哈希成 Merkle 树，它的大小和形状与这个树基本相同。

每轮认证状态在 IC 中以几种方式使用。

- **输出认证**。跨子网消息和对入口消息的响应是使用每轮认证状态进行认证的。使用 Merkle 树结构，一个单独的输出（跨子网消息或入口消息响应）可以通过提供

⁶但见 Section 8.2

Merkle 树根部的阈值签名，以及 Merkle 中从根部到代表该输出的叶子的路径上（和邻近）的哈希值，对任何一方进行认证。因此，验证单个输出所需的哈希值数量与 Merkle 树的深度成正比，即使 Merkle 树的大小非常大，其深度通常也是相当小的。因此，一个阈值签名可以用来有效地验证许多单独的输出。

- **防止和检测非决定性。** 共识保证每个副本以相同的顺序处理输入。由于每个副本都以确定的方式处理这些输入，所以每个副本应该获得相同的状态。然而，IC 被设计成具有额外的稳健性，以防止和检测任何（意外的）非决定性计算，如果它出现。每轮认证的状态是用来做到这一点的机制之一。由于我们使用一个 $(n - f)$ -out-of- n 的阈值签名进行认证，并且由于 $f < n/3$ ，所以只能有一个状态序列被认证。

为了理解为什么状态链很重要，请考虑下面的例子。假设我们有 4 个副本， P_1, P_2, P_3, P_4 ，其中一个腐败的，比如说 P_4 ，每个副本 P_1, P_2, P_3 开始都是相同的状态。

- 在第一轮中，由于非确定性的计算， P_1, P_2 计算出一个消息 m_1 发送到子网 A，而 P_2 计算出一个消息发送到子网 A。
- 在第 2 轮中， P_1, P_3 计算出一个消息 m_2 发送到子网 B，而 P_2 计算出一个消息 m_2 发送到子网 B。
- 在第 3 轮中， P_2, P_3 计算出一个消息 m_3 发送到子网 C，而 P_2 计算出一个消息 m_3 发送到子网 C。

这在下表中有所说明。

貔貅	m_1	$T A$	m_2	$T B$	m_3	$T C$
P_2	m_1	$T A$	m_2	$T B$	m_3	$T C$
P_3	m_1	$T A$	m_2	$T B$	m_3	$T C$

我们假设复制体 P_1, P_2 和 P_3 各自执行一个有效的计算序列，但是由于非确定性，这些序列并不完全相同。（尽管不应该有任何非决定性，在这个例子中，我们假设有非决定性）。

现在，假设我们没有把这些状态连在一起。因为 P_4 是腐败的，可以签署任何东西，他可以在说 " $m_1 T A$ " 的第 1 轮状态上创建一个 4 选 3 的签名，同样，在说 " $m_2 T B$ " 的第 2 轮状态上，以及在说 " $m_3 T C$ " 的第 3 轮状态上，创建一个 3 选 4 的签名，即使相应的序列

$m_1 T A, m_2 T B, m_3 T C$

可能与任何有效的计算序列不兼容。更糟糕的是，这样一个无效的计算序列可能会导致其他子网的状态不一致。

通过连锁，我们确保即使存在一些非确定性，任何认证状态的序列都对应于一些有效的计算序列，而这些计算是由诚实的副本实际进行的。

- **与共识的协调。** 每轮认证状态也被用来协调执行层和共识层，有两种不同的方式。
 - **共识节流。** 每个副本将跟踪其拥有认证状态的最新一轮--这被称为**认证高度**。它还

将跟踪它拥有公证区块的最新一轮--这被称为**公证高度**。如果公证的高度明显大于认证的高度，这就是一个信号，表明执行滞后于共识，而共识需要被**节流**。这种滞后可能是由于非确定性的计算，也可能是由于各层之间更良性的性能不匹配。共识的节流方式是通过在第 2 章中讨论的**延迟函数**来实现的。Section 5.9 - 具体来说，每个副本会随着公证高度和认证高度之间的差距增加 "治理者" 值 e （这利用了 "局部调整的延迟函数" 的概念，如在 Section 5.12.2).

- **特定状态的有效载荷验证**。正如在 Section 5.7 所述，有效载荷中的输入必须通过某些有效性检查。事实上，这些有效性检查可能在一定程度上取决于状态。我们跳过的一个细节是，每个区块都包括一个圆周率，据了解，这些有效性检查应该是针对该圆周率的认证状态进行的。一个需要执行这种验证的副本将等待，直到该轮数的状态被认证，然后使用该轮的认证状态来执行验证。这确保了即使在非确定性的计算中，所有的副本都在执行相同的有效性测试（因为否则，共识可能会被卡住）。

6.2 查询调用与更新调用

正如我们到目前为止所描述的那样，进站信息必须通过共识，以便它们被子网的所有复制体以相同的顺序处理。然而，对于那些处理过程不修改子网的复制状态的进站消息，有一个重要的优化。这些被称为**查询调用**--与其他入口消息相反，它们被称为**更新调用**。查询调用被允许执行读取和可能更新 Canister 状态的计算，但对 Canister 状态的任何更新都不会被提交到复制的状态。因此，一个查询调用可以被处理

这大大减少了从查询调用中获得响应的延迟。

请注意，对查询调用的响应并不记录在入口历史数据结构中。因此，我们不能直接使用每轮认证的状态机制来认证对查询调用的响应。然而，我们提供了一个单独的机制来验证这种响应：**认证变量**。作为每轮认证状态的一部分，子网中的每个 Canister 都被分配了少量的字节，这是该 Canister 的**认证变量**，其值可以通过更新调用进行更新，并可以使用每轮认证状态机制进行验证。此外，一个 Canister 可以使用其认证变量来存储 Merkle 树的根。这样一来，只要响应是默克尔树中以该 Canister 的认证变量为根的叶子，那么对该 Canister 的查询调用的响应就可以得到认证。

6.3 外部用户认证

入口报文和跨子网报文的主要区别之一是用于验证这些报文的机制。我们已经在上面看到（见 Section 6.1）**阈值签名**是如何用于认证跨子网消息的。**NNS 注册处**（见 Section 1.5）持有用于验证跨子网消息的**阈值签名的公共验证密钥**。

没有外部用户的中央登记处。相反，外部用户使用一个**用户标识符**来识别自己的身份，该**标识符**是一个公共签名验证密钥的哈希值。该用户持有一个相应的秘密签名密钥，用于签署进入的信息。这样的签名，以及相应的公钥，与入口信息一起发送。**IC** 自动验证签名，并将用户标识符传递给适当的 Canister。然后，Canister 可以根据用户识别码和进入信息中指定的操作的其他参数，授权所要求的操作。

首次使用的用户在与 **IC** 的第一次互动中产生一对密钥，并从公钥中得出他们的用户标

识。返回的用户使用由用户代理存储的秘密密钥进行认证。一个用户可以使用签名授权，将几个密钥对与一个用户身份联系起来。这很有用，因为它允许一个用户使用同一个用户身份从几个设备访问 IC。

7 执行层

执行环境一次处理一个输入。这个输入来自其中一个输入队列，并被引向一个 Canister。基于这个输入和 Canister 的状态，执行环境更新 Canister 的状态，另外还可以向输出队列添加消息并更新输入历史（可能是对早期输入消息的回应）。

在一个特定的回合中，执行环境将处理几个输入。一个调度器决定了哪些输入在某一轮中被执行，以及以何种顺序执行。在不深入了解调度器的所有细节的情况下，我们强调了其中的一些目标：

- 它必须是确定性的，也就是说，只取决于给定的数据。
- 它应该在 Canister 之间公平地分配工作负载（但要优化吞吐量而不是延迟）。
- 每一轮所做的工作总量，以 *Cycle* 为单位衡量（见 Section 1.8）循环，应该接近某个预先确定的数量。

执行环境（与消息路由器一起）必须处理的另一项任务是，一个子网的 Canister 产生跨子网消息的速度超过了另一个子网的 Canister 的消耗速度。为此，实施了一个自我调节机制，对产生信息的 Canister 进行节制。

还有许多其他的资源管理和记账任务是由执行环境来处理的。然而，所有这些任务都必须以确定性的方式处理。

7.1 随机磁条

每个子网都可以访问一个分布式伪随机发生器（PRG）。如前所述 Section 3，伪随机位来自于一个种子，该种子本身是一个 $(f+1)$ -out-of- n BLS 签名，称为随机带（Random tape）。共识协议的每一轮都有一个不同的随机带。虽然这个 BLS 签名与共识中使用的随机信标相似（见 Section 5.5），但其机制却有些不同。

在共识协议中，一旦高度为 h 的区块被最终确定，每个诚实的副本将释放其高度为 $h+1$ 的随机磁带份额。这有两个含义。

1. 在高度为 h 的区块被任何诚实的复制体最终确定之前，高度为 $h+1$ 的随机磁带被保证是不可预测的。
2. 当高度为 $h+1$ 的区块被任何一个诚实的复制体最终确定时，该复制体通常会拥有它所需要的所有股份来构建高度为 $h+1$ 的随机磁带。

为了获得伪随机比特，一个子网必须对这些比特提出请求。这样的伪随机位请求将作为执行层的“系统调用”在某个回合中提出，比如说 h ，然后系统将在稍后回应该请求，此时高度为 $h+1$ 的随机磁带是可用的。根据上面的属性（1），可以保证请求的伪随机位在提出请求时是不可预测的。根据上面的属性（2），所请求的随机位通常在下一个块被最终确定

时是可用的。事实上，在目前的实现中，在一个高度为 h 的块被最终确定时，共识层（见 Section 5）将同时把高度为 h 的块的（有效载荷）和高度为 $h+1$ 的随机磁带送到消息路由层进行处理。

8 链式钥匙密码学 II：链式进化技术

中提到的。Section 1.6.2 中提到，链式密码学包括一系列技术，用于随着时间的推移稳健安全地维护基于区块链的复制状态机，这些技术共同构成了所谓的**链式演进技术**。每个子网在许多轮的**纪元**中运行（通常在几百轮的数量级上）。链式进化技术实现了许多基本的维护活动，这些活动定期执行，其节奏与纪元挂钩：**垃圾收集**、**快进**、**子网成员变化**、**主动重新分享秘密**和**协议升级**。

链式进化技术有两个基本成分：**摘要块**和**追赶包（CUPs）**。

8.1 摘要块

每个纪元的第一个块是一个**摘要块**。摘要块包含特殊数据，将用于管理各种阈值签名方案的份额（见 Section 3）。有两种阈值方案。

- 一个 $(f+1)$ - n 的方案，每一个纪元都会产生一个新的签名密钥。
- 一个 $(n-f)$ - n 的方案，对于该方案，签署密钥每隔一段时间就会重新分配一次。

低门槛方案用于**随机信标**和**随机磁条**，而高门槛方案则用于认证子网的复制状态。

回想一下，DKG 协议（见 Section 3.5）要求对于每个签名密钥，我们有一组交易，并且每个副本可以从这组交易中非交互地获得它的签名密钥份额。

还记得 NNS 维护着一个**注册表**，除其他外，它决定了一个子网的成员资格（见 Section 1.5）。注册表（以及子网成员）可能会随着时间的推移而改变。因此，子网必须同意他们在不同时期为不同目的使用哪个**注册表版本**。这一信息也被存储在摘要块中。

纪元 i 的摘要块包含以下数据字段。

- **currentRegistryVersion**。这个注册表版本将决定整个第 i 纪元所使用的**共识委员会**--共识层执行的所有任务（制块、公证、定稿）都将由这个委员会执行。
- **nextRegistryVersion**。在每一轮共识中，区块制造者将在其提议中包括它所知道的最新注册表版本（该版本必须不早于提议区块所扩展的区块）。这确保了第 i 个纪元的摘要块中的 **nextRegistryVersion** 的值是相当最新的。

纪元 i 中的 **currentRegistryVersion** 的值被设置为纪元 i 中的 **nextRegistryVersion** 的值。1.

- **currentDealingSets**。这些是决定阈值签名密钥的交易集，这些密钥将在第 i 纪元用于签名信息。

正如我们将看到的，第 i 个纪元的**阈值签署委员会**（即持有相应阈值签署密钥份额的副本）是第 $i-1$ 个纪元的**共识委员会**。

- *nextDealingSets*。这是在纪元 *i-1* 期间收集的交易被收集和存储的地方。⁷ 第 *i* 个纪元的 *currentDealingSets* 的值将被设置为第 *i-1* 个纪元的 *nextDealingSets* 的值（它本身由第 *i-2* 个纪元收集的交易组成）。

- *collectDealingParams*。这描述了定义在纪元 *i* 期间要收集的交易集的参数。在纪元 *i* 期间，区块制造者将在他们提议的区块中包括相对于这些参数验证的交易。

这些交易的接收委员会是基于纪元 *i* 的摘要块的 *nextRegistryVersion* 值。

对于低门槛方案，交易委员会是第 *i* 个纪元的共识委员会。

对于高门槛方案，要重新分享的份额是基于第 *i* 个纪元的 *nextDealingSets* 的值。因此，交易委员会是第 *i-1* 个纪元的接收委员会，它也是第 *i* 个纪元的共识委员会。

还可以观察到，纪元 *i* 的阈值签署委员会是纪元 *i-2* 的接收委员会，它是纪元 *i-1* 的共识委员会。

第 *i* 纪元的共识依赖于第 *i* 纪元的值 *currentRegistryVersion* 和 *currentDealingSets*--特别是，共识委员会本身的组成是基于 *currentRegistryVersion* 的，共识中使用的随机信标是基于 *currentDealingSets* 的。此外，就像其他区块一样，在第 *i* 纪元开始时可能有不止一个总结区块被公证，这种模糊性需要通过第 *i* 纪元的共识来解决。这种看似循环的问题是通过坚持第 *i-1* 纪元开始时的总结区块在第 *i* 纪元开始前已经被最终确定，因为较新的总结区块中的相关值是直接从那个较老的总结区块中复制的。这实际上是一个隐含的同步性假设，但这是一个相当学术性的假设。事实上，由于在“共识节流”中讨论的 Section 5.12.2 中讨论的“共识节流”以确保有效性，而且一个纪元的长度相当大，这在实践中基本上不可能发生：在共识到达纪元 *i-1* 的末尾而没有最终确定纪元 *i-1* 的摘要块之前，公证延迟函数将增长到天文数字般大，因此最终确定所需的部分同步假设将（基本上）肯定地得到满足（就所有实际目的而言）。⁸

8.2 CUPs

在描述 CUP 之前，我们首先指出随机信标的一个细节：每一轮的随机信标取决于前一轮的随机信标。这不是一个基本的特征，但它影响了 CUP 的设计。

CUP 是一个特殊的消息（不在区块链上），它拥有（大部分）副本在特定纪元中开始工作所需的一切，而不知道以前纪元的任何信息。它由以下数据字段组成。

- 整个复制状态的 Merkle 哈希树的根（相对于部分的、每轮的认证状态，如在 Section 6.1）。
- 纪元的摘要块。

⁷ 我们省略的一个细节是，如果我们未能在第 *i-1* 个纪元收集到所有需要的交易，那么作为后备措施，第 *i* 个纪元的 *nextDealingSets* 的值实际上将被设置为第 *i* 个纪元的 *currentDealingSets* 的值。

⁸ 还要注意的，在第 *i* 纪元收集的交易取决于第 *i* 纪元的摘要块中的数据，特别是 *nextDealingSets* 和 *nextRegistryVersion* 的值。因此，这些交易不应该被生成，也不能被验证，直到第 *i* 个纪元的摘要块被最终确定。

- 纪元第一轮的随机信标。
- 在子网的 $(n - f) \cdot n$ 个阈值的签名密钥下对上述字段进行签名。

为了生成一个给定纪元的 CUP，一个副本必须等到该纪元的摘要块被最终确定，并且相应的每轮状态被认证。正如已经提到的，整个复制的状态必须被哈希为 Merkle 树--即使有一些技术被用来加速这个过程，这仍然是相当昂贵的，这就是为什么它在每个纪元只做一次。由于 CUP 只包含 Merkle 树的根，所以使用了一个特殊的状态同步子协议，允许副本从其对等体中提取它需要的任何状态--同样，使用了一些技术来加速这一过程，但它仍然相当昂贵。由于我们对 CUP 使用高阈值签名，我们可以确保在任何时代只有一个有效的 CUP，此外，将有许多对等体可能被拉出状态。

8.3 实施链式进化技术

垃圾收集。由于在一个给定纪元的 CUP 中包含的信息，对于每个副本来说，在该纪元之前清除所有已经处理过的输入，以及为这些输入排序所需的所有共识级协议信息是安全的。

快速转发。如果一个子网中的副本落后于它的同伴非常多（因为它已经停机或与网络断开了很长一段时间），或者一个新的副本被添加到一个子网中，它可以被快速转发到最近的纪元的开始，而不必运行共识协议和处理所有的输入到这一点。这样的副本可以通过获得最新的 CUP 来实现。使用 CUP 中包含的摘要块和随机信标，以及来自其他副本（尚未被清除）的协议消息，该副本可以从相应的历时开始向前运行共识协议。该副本还将使用状态同步子协议来获得对应于纪元开始的复制状态，这样它也可以处理由共识产生的输入。

Figure 5 说明了快进的情况。在这里，我们假设一个需要追赶的复制体在一个纪元的开始有一个 CUP，它开始（比如）在高度 101。CUP 包含高度为 101 的复制状态的 Merkle 树的根，高度为 101 的摘要块（以绿色显示），以及高度为 101 的随机信标。这个复制体将使用状态同步子协议从其对等体那里获得高度为 101 的完整复制状态，使用 CUP 中的 Merkle 树的根来验证这个状态。

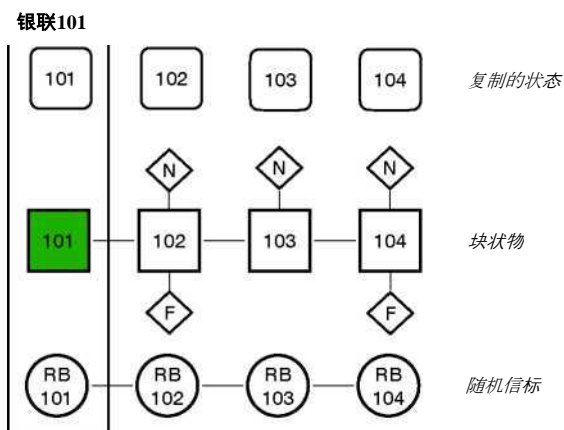


图5：快进

在获得这个状态后，复制体就可以参与协议，从其对等人那里获得 102、103 等高度

的区块（以及与共识相关的其他消息），并更新其复制状态的副本。如果它的对等体已经在更高的高度上完成了区块，这个副本将以最快的速度处理这些已完成的区块，因为它可以从对等体那里获得这些区块（以及它们的公证和最终确定）（并且以执行层允许的速度）。

子网成员的变化。我们已经讨论了摘要块是如何被用来编码哪个版本的注册表在一个给定的纪元中是有效的，以及如何用它来决定子网成员，更具体地说，各种任务的成员委员会。请注意，即使在一个副本从子网中删除后，它也应该（如果可能的话）再参加一个纪元的指定委员会职责。

积极主动地重新分享秘密。我们已经讨论了如何使用摘要块来生成和重新分享签名钥匙。如果有必要，可以从银联中获得所需的摘要块。

协议升级。银联也被用来实施协议升级。协议升级是由 **NNS** 发起的（见 **Section 1.5**）。在不涉及所有细节的情况下，其基本思想是这样的。

- 当需要安装一个新版本的协议时，在一个纪元开始的摘要块将表明这一点。
- 运行旧版协议的复制体将继续运行共识，足以最终完成摘要块并创建相应的 **CUP**；但是，它们将只创建空块，不向消息路由和执行传递任何有效载荷。
- 新版本的协议将被安装，而运行新版本协议的副本将恢复运行上述 **CUP** 的完整协议。

参考文献

- [AMN+20] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin. Sync HotStuff: Simple and Practical Synchronous State Machine Replication. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 106-118. IEEE, 2020.
- [BGLS03] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In E. Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 416-432. Springer, 2003.
- [BKM18] E. Buchman, J. Kwon, and Z. Milosevic. The latest gossip on BFT consensus, 2018. arXiv:1807.04938, <http://arxiv.org/abs/1807.04938>.
- [BLS01] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. In C. Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 514-532. Springer, 2001.
- [But13] V. Buterin. Ethereum whitepaper, 2013. <https://ethereum.org/en/whitepaper/>.

- [CDH+21] J. Camenisch, M. Drijvers, T. Hanke, Y.-A. Pignolet, V. Shoup, and D. Williams. Internet Computer Consensus. Cryptology ePrint Archive, Report 2021/632, 2021. <https://ia.cr/2021/632>.
- [CDS94] R. Cramer, I. Damgard, and B. Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *Advances in Cryptology – CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, volume 839 of *Lecture Notes in Computer Science*, pages 174-187. Springer, 1994.
- [CL99] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In M. I. Seltzer and P. J. Leach, editors, *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173-186. USENIX Association, 1999.
- [CWA+09] A. Clement, E. L. Wong, L. Alvisi, M. Dahlin, and M. Marchetti. Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults. In J. Rexford and

- E. G. Sirer, editors, *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2009, April 22-24, 2009, Boston, MA, USA*, pages 153-168. USENIX Association, 2009. http://www.usenix.org/events/nsdi09/tech/full_papers/clement/clement.pdf.
- [Des87] Y. Desmedt. Society and Group Oriented Cryptography: A New Concept. In C. Pomerance, editor, *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, volume 293 of *Lecture Notes in Computer Science*, pages 120-127. Springer, 1987.
- [DLS88] C. Dwork, N. A. Lynch, and L. J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288-323, 1988.
- [Fis83] M. J. Fischer. The Consensus Problem in Unreliable Distributed Systems (A Brief Survey). In *Fundamentals of Computation Theory, Proceedings of the 1983 International FCT-Conference, Borgholm, Sweden, August 21-27, 1983*, volume 158 of *Lecture Notes in Computer Science*, pages 127-140. Springer, 1983.
- [FS86] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186-194. Springer, 1986.
- [GHM⁺17] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. *Cryptology ePrint Archive*, Report 2017/454, 2017. <https://eprint.iacr.org/2017/454>.
- [Gro21] J. Groth. Non-interactive distributed key generation and key resharing. *Cryptology ePrint Archive*, Report 2021/339, 2021. <https://ia.cr/2021/339>.
- [JMV01] D. Johnson, A. Menezes, and S. A. Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA). *Int. J. Inf. Sec.*, 1(1):36-63, 2001.
- [Mer87] R. C. Merkle. A Digital Signature Based on a Conventional Encryption Function. In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, volume 293 of *Lecture Notes in Computer Science*, pages 369-378. Springer, 1987.
- [MXC⁺16] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The Honey Badger of BFT Protocols. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 31-42. ACM, 2016.
- [Nak08] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. <https://bitcoin.org/bitcoin.pdf>.

- [PS18] R. Pass and E. Shi. Thunderella: Blockchains with Optimistic Instant Confirmation. In J. B. Nielsen and V. Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 3-33. Springer, 2018.
- [PSS17] R. Pass, L. Seeman, and A. Shelat. Analysis of the Blockchain Protocol in Asynchronous Networks. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 643-673, 2017.
- [Sch90] F. B. Schneider. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Comput. Surv.*, 22(4):299-319, 1990.
-

*<https://dfinity.org/foundation/>