互联网计算机白皮书（中文）

作　　者：　Dfinity Teams

翻　　译：　Mixlabs 团队

2022 年 1 月

摘　要

智能合约是一种新型软件形式，它将彻底改变软件的编写方式、IT 系统的维护方式、应用程序和商业的构建方式。智能合约是建立在去中心化区块链上的可组合、自动化软件，具有防篡改性和不可停止性。在本文中，我们将介绍互联网计算机 (Internet Computer，简称 IC)。这是区块链上一种不同凡响的全新设计，足以释放智能合约的全部潜力，打破传统区块链智能合约在速度、存储成本和计算能力方面的不足，让智能合约第一次在区块链上实现端到端托管的完全去中心化应用程序。互联网计算机（IC）由一组加密协议组成，这些协议将独立操作的节点整合连接到一组区块链中。这些区块链存储和执行 IC 的智能合约，即 Canister。Canisters 可以存储数据，对数据进行一般的计算，提供完整的技术栈，并直接为终端用户提供网页服务。计算和存储成本由“反向 gas 费模型（reverse-gas model)”来提供，其中 Canister 开发者需要用从 ICP (IC 的原生代币) 中获得的收入来预付相关费用。ICP 代币也用于社区治理：IC 由一个去中心化自治组织 (Decentralized autonomous organization，简称 DAO) 管理。DAO 决定着网络拓扑的更改和协议的升级。

目　录

第 1 章 简介

1.1 释放智能合约潜力

由于其独特的属性，智能合约成为了 Web3 的关键赋能者。Web3 是一种新的 Web 方式：应用程序运行在去中心化的区块链平台上并完全由用户控制。这样的去中心化应用程序 (Decentralized Applications，简称 dapps) 通常是代币化的，这意味着代币将作为奖励分发给参与 dapps 的用户。用户可以以多种形式参与 dapp，比如说运营、贡献内容、管理 dapp，或者是创建和维护 dapp 等。代币也可以在交易所购买; 实际来说，出售代币是为 dapp 发展融资的一种常见方式。最后，代币也是为 dapp 提供的服务或内容付费的一种支付方式。目前运行在区块链平台上的智能合约，包括所有流行的平台 (如以太坊)，都存在许多不足，比如交易和存储成本高，计算速度慢，无法为用户提供前端服务等。因此，许多流行的区块链应用程序并不是完全去中心化的，而是混合的。其中大多数应用程序托管在传统的云平台上，并调用区块链上的智能合约来实现其整体功能的一小部分。不过这导致了应用程序去中心化不彻底，无可避免地携带了许多传统云托管应用程序的缺点，例如受云提供商的控制，并且容易受到众多单点故障的影响。

互联网计算机 (IC) 是一个执行智能合约的新平台。此处，我们使用的是广义上的" 智能合约"：一个通用的、不可变的、防篡改的计算机程序，可以在去中心化公共网络上自动执行。

- 所谓通用，是指智能合约程序的类别是图灵完备的 (即任何可计算的东西都可以通过智能合约计算)。
- 所谓不可变，是指一旦部署，智能合约的代码就不能由一方单方面更改。[1]
- 所谓防篡改，是指程序的指令被如实执行，中间结果和最终结果被准确地存储和/或传输。IC 为智能合约提供了一系列可变策略，从完全不可变到单方面可升级，其间还有其他可变选择。
- 所谓自动化，是指智能合约由网络自动执行，无需个人手动操作。
- 所谓去中心化的公共网络，是指公共可访问的、地理上分散的、不受少数个人或组织控制的计算机网络。

此外，智能合约还有以下特点：

- 可组合，这意味着它们可以相互作用；

[1] IC 允许智能合约的一系列可变性策略，从纯粹的不可变到单方面可升级，以及介于两者之间的其他选项。

- 支持代币化，这意味着它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 致力于：

- 提高成本效益，特别是保证应用程序计算和存储数据成本仅为以往平台的极小部分;
- 为处理智能合约交易提供更高的吞吐量和更低的延迟;
- 具有更强的可扩展性。IC 可以处理无限数量的智能合约数据和计算，因为它可以通过为网络增加节点来增加容量。

除了是一个智能合约平台外,IC 也作为一个完整的技术栈，因此工作系统和服务可以完全基于 IC 平台。在 IC 上的智能合同可以满足终端用户的 HTTP 请求, 进而直接提供交互式 web 服务体验。这意味着可以在不依赖企业云托管服务或私有服务器的情况下创建系统和服务，从而以真正的端到端方式实现所有智能合约的优势。

实现 Web3 的愿景。对于终端用户来说，访问基于 IC 的服务在很大程度上是透明的。与访问公共或私有云上的应用程序相比，他们的个人数据更安全，但与应用程序交互的体验是相同的。

然而，对于创建和管理这些基于 IC 的服务的人员来说，IC 消除了与开发和部署现代应用程序和微服务相关的许多成本、风险和复杂性。例如，垄断互联网的大型科技公司推动了应用的合并，而 IC 提供了另一种选择。此外，在不依赖于防火墙、备份设施、负载均衡服务或故障转移编排的情况下，它的安全协议保证了可靠的信息传达、透明的问责制度和恢复力。

构建 IC 就需要恢复互联网开放、创新和创意的本质，换句话说，就是要实现 Web3 的愿景。为了集中讨论几个具体的例子，IC 做了以下几点:

- 支持互操作性、共享功能、永久 APIs 和无主应用程序，从而降低平台风险，鼓励创新和协作。
- 将数据自动保存在存储器中，从而消除对数据库服务器和存储管理的需求，提高计算效率，简化软件开发。
- 简化了 IT 组织需要集成和管理的技术栈，从而提高操作效率。

1.2 高层次概览互联网计算机

大致说来，IC 是一个相互作用的**复制状态机**网络。复制状态机在分布式系统中是一个相当通用的概念 [Sch90]，从介绍状态机开始，我们在这里做一个简短的阐释。

状态机是一种特殊的计算模型，维持着一种状态，这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。该机器是**分轮**工作的: 每一轮，其接受**输入**，对输入内容和当前状态应用**状态转换函数**，获得一个**输出**和一个新状态。新状态将在下一轮中成为当前状态。

IC 的状态转换函数是一种**通用函数**，即存储在状态中的一些输入和数据可以是任意**程序**，它们可以作用于其他输入和数据。因此，一个状态机代表了一个通用的 (即图灵完备的) 计算模型。

为了实现**容错**，状态机可以被复制。复制的状态机由多个同一子网的 replicass 组成，每个 replica 都运行同一状态机的 replicas。即使某些 replicas 出现故障，子网也应继续且能正确工作。

子网中的每个 replica 必须以相同的顺序处理相同的输入。为了实现这一点，子网中的 replicas 必须运行一个共识协议 [Fis83]，该协议确保子网进程中所有 replicas 的输入顺序相同。因此，每个 replica 的内部状态将以完全相同的方式随时间改变，并且将产生完全相同的输出序列。请注意，IC 上复制状态机的输入可能是由外部用户生成的输入，也可能是由另一个复制状态机输出的输入。类似地，复制状态机的输出可以是指向外部用户的输出，也可以是指向另一个复制状态机的输入。

1.3 故障模型

在计算机科学领域中，通常考虑两种类型的 replica 故障: **崩溃故障和拜占庭故障**。当 replicas 突然停止且不恢复时，就是**崩溃故障**。**拜占庭故障**是指 replicas 可能以任意方式偏离其指定协议的故障。此外，由于拜占庭故障，恶意对手可能会直接控制一个或几个 replicas 并协调这些 replicas 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。

共识协议和实现复制状态机的协议通常会假设出错 replicas 的数量，以及它们可能出错的程度 (崩溃或拜占庭)。在 IC 中，假设一般为：一个给定的子网有 n 个 replicas，这些 replicas 中有少于 $n/3$ 个是故障的，而且这些故障可能是拜占庭类。(请注意，IC 中的不同子网可能有不同的大小。)

1.4 通信模型

共识协议和实现复制状态机的协议通常也会对**通信模型**进行假设，假设描述了敌手延迟 replicas 之间的消息传递能力。在两个极端情况下，有以下两种模型:

- 在**同步模型**中，存在已知的有限时间上界 δ，任何发送的消息，它将在小于 δ 的时间内到达。(译者注：消息传递延迟已知且有上界 δ)

- 在**异步模型**中，对于发送的任何消息，敌手可以延迟其发送的任何时间，因此没有发送消息的时间限制; 然而，每条消息最终都必须送达。（译者注：消息传递延迟无上界，但最终会到达。）

由于 IC 子网中的 replicas 通常分布在全球各地，因此同步通信模型是非常不现实的。实际上，攻击者可以通过延迟正确 replicas 或它们之间的通信来损害协议的正常运行。这样的攻击通常比控制和破坏一个正确 replicas 更容易。

在全球分布式子网的设置中，最实用、最稳固的模型是非同步模型。不幸的是，在这个模型中还没有已知的真正实用的共识协议 (最近的非同步共识协议，如 [MXC+16]，可以达到可观的吞吐量，但延迟方面表现不佳)。因此，就像大多数不依赖同步通信而切实可行的拜占庭容错系统 (如 PBFT[CL99][BKM18][AMN+20]) 一样，IC 选择了一种折衷方案: **部分同步**通信模型 [DLS88]。这种部分同步模型可以用多种方式表示。IC 使用的部分同步假设大致上来说就是：每个子网中 replicas 之间的通信在每一小段短时间内周期性同步; 此外，延迟时间上界 δ 不需要提前知道。只有在保证共识协议持续进行时（即 liveness）才需要使用到部分同步假设。共识协议的安全性（即 safety）不需要部分同步假设，甚至不需要消息最终到达。IC 协议栈中也没有部分同步假设的用武之地。

在部分同步和拜占庭故障的假设下，我们已知将故障数量限制为 $f<n/3$ 是最优的设置。

1.5 权限模型

最早的共识协议 (如 PBFT [CL99]) 是**许可协议**，即复制状态机组成的 replicas 由一个集中的组织管理，该组织决定哪些单位提供 replicas 和网络的拓扑结构，并可能实现某种集中的公钥基础设施（译者注：PKI）。许可共识协议通常是最高效的。它们确实避免了单点故障，但对于某些应用程序来说，集中式治理是不合需求的，而且这与蓬勃发展的 Web3 时代的精神背道而驰。

最近，我们看到了**无需许可**共识协议的兴起，如比特币 [Nak08]、以太坊 [But13] 和 Algorand[GHM+17]。这样的协议基于区块链和工作证明 (PoW)(例如，比特币和 2.0 版本之前的以太坊) 或权益证明 (PoS)(例如，Algorand 和以太坊 2.0 版本)。虽然这样的协议是完全去中心化的，但它们比许可协议的效率低得多。我们也要指出，正如在 [PSS17] 中看到的，基于工作证明的共识协议，如比特币，在异步通信网络中不能保证正确性 (即安全)。

IC 的权限模型是一种**混合模型**，在保留去中心化 PoS 协议优势的同时，获得了受许可协议的效率。这种混合模型称为 **DAO 控制网络**，其工作原理大致如下:

每个子网运行一个许可共识协议，但是由去中心化自治组织 (DAO) 决定哪些单位提供 replicas，配置网络的拓扑结构，提供公钥基础设施，并控制部署到 replicas 上的协议版本。IC 的 DAO 基于 PoS 协议，被称为网络神经系统 (network nervous system，NNS)。NNS 所有的决策都由社区成员的投票决定，而成员们的投票权大小是由其在 IC 抵押的原生治理代币数量决定 (见 1.8 节)。通过这个基于 PoS 的治理系统，创建新的子网、向现有的子网添加或删除 replicas、部署软件更新、对 IC 进行其他修改等都得以实现。NNS 本身就是一个复制状态机，它 (与其他任何状态机一样) 运行在特定的子网上，该子网的成员是通过相同的基于 PoS 的治理系统确定的。NNS 拥有一个称为**注册表**的数据库，该数据库跟踪 IC 的拓扑结构: 哪个 replicas 属于哪个子网、replicas 的公钥等等。(关于 NNS 的更多细节，请参阅第 1.10 节。)

因此，可以看到 IC 的 DAO 控制网络允许 IC 实现许可网络的许多实际优势 (就更有效的共识而言)，同时保证了去中心化网络的许多好处 (治理由 DAO 控制)。运行 IC 协议的 replicas 托管分布在不同地方、独立操作的数据中心的服务器上。这也巩固了 IC 的安全性和去中心化本质。

1.6 chain-key cryptography

IC 的共识协议确实运用了区块链，同时也使用了公钥密码学，具体来说就是数字签名: 由 NNS 维护的注册表将公钥绑定到 replicas 和整个子网。这就形成了一种独特而有力的技术集合，我们称之为 chain-key cryptography 。它由几个部分组成。

1.6.1 门限签名（Threshold signatures）

chain-key cryptography 的第一个组成部分是**门限签名**: 这是一种成熟的加密技术, 允许子网把公共签名验证相应的签名密钥分成多个部分，并分布在该子网的所有 replicas 中。使得恶意的 replicas 持有的份额不能伪造签名，而诚实的 replicas 所持有的份额可以生成与 IC 的策略和协议一致的签名。

这些门限签名的一个关键应用是：一个子网和外部用户只需要验证另一个子网的公钥数字签名，就完成了对它的输出的验证。

一个子网的单个输出只需由另一个子网或外部用户对 (第一) 子网的签名验证公钥的数字签名进行验证即可。

注意，子网的签名验证公钥可以从 NSS 获得，该签名验证公钥在子网的有效期内保持不变 (即使子网的成员可能在有效期内发生变化)。这与许多不可扩展的

基于区块链的协议有所不同。那些协议为了验证任何一个输出，都需要验证整个区块链。

这些阈值签名在 IC 中还有许多其他应用。其中一个应用是给子网中的每个 replica 提供不可预测的伪随机比特的访问权限。这是共识中使用 Random Beacon 以及执行中使用 Random Tape 的基础。

授予子网中的每个副本对不可预测的伪随机位（派生自此类签名）的访问权限。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。

为了安全地部署门限签名，IC 使用了一种创新的分布式密钥生成 (distributed key generation ，DKG) 协议。该协议构建了签名验证公钥，为每个 replica 提供相应的签名私钥，并在上述的故障和通信模型中生效。

1.6.2 链式演进技术

Chain-key cryptography 还包括一系列复杂的技术，用于随着时间的推移稳健和安全地维护基于区块链的复制状态机（replicasted state machine），这些技术共同构成了我们所说的链演进技术（chain-evolution technology）。每个子网以多轮次的 Epoch 运行 (通常按几百轮的次序)。利用阈值签名和其他一些技术，链演技术实现了许多按时期定期执行的基本维护工作：

- **垃圾回收:** 在每个时代结束时，所有已经处理的输入以及所有需要对这些输入进行排序的共识级协议消息都可以安全地从每个 replica 的内存中清除。这对于保证 replicas 的存储需求不会无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了鲜明的对比，这些不可扩展的协议必须存储创世块以来的整个区块链。
- **快速参与:** 如果子网中的一个 replicas 远远落后于它的同伴 (因为它长时间处于停机状态或与网络断开连接)，或者有一个新的 replicas 被添加到一个子网中，那么就会把它快速转发到最近一个 Epoch 的开始，而不必运行共识协议并处理到那个时候为止的所有输入。这与许多不可扩展的基于区块链的协议形成了鲜明对比。在这些不可扩展的协议中，来自创世块的整个区块链都必须进行处理。
- **子网成员变更:** 子网的成员资格 (由 NNS 决定，参见 1.5 节) 可能随着时间的推移而变化。这只在一个 Epoch 的边界处发生，并且需要谨慎操作，以确保一致和正确。
- **主动秘密再共享:** 我们在上面的 1.6.1 节中提到了 IC 如何使用链式密钥加密技术（尤其是阈值签名）来进行输出验证。这是基于秘密共享的，它通过将秘密 (在本例中为秘密签名密钥) 拆分为存在 replicas 之间的份额来避免任何

单点故障。在每个新 Epoch 的开始，这些共享被主动地重新分享。这实现了两个目标:

  - 当子网的成员更改时，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 replicas 不再拥有秘密份额。
  - 如果在任何一个 Epoch，甚至每个 Epoch 都有少量的份额泄露给攻击者，那么这些份额对攻击者没有任何帮助。

- **协议升级:** 当 IC 协议本身需要升级以修复漏洞或添加新特性时，这可以在新 Epoch 开始时使用特殊协议自动完成。

1.7 执行模型

如前所述，IC 中的复制状态机可以执行任意程序。IC 中的基本计算单元称为“canister”，它与进程（process）的概念大致相同，因为它包括程序及其状态 (随着时间的推移而变化)。

Canister 程序被编码在 WebAssembly 中，简称 Wasm，是一种基于堆栈的虚拟机的二进制指令格式。Wasm 是一个开放标准[2]。虽然它最初是为了在网页上实现高性能应用程序而设计的，但实际上它非常适合通用计算。

IC 提供了一个运行环境，用于在一个 Canister 中执行 Wasm 程序，并与其他 Canister 和外部用户通信 (通过消息传递)。虽然在原则上，可以用任何语言编写一个可以编译到 Wasm 的 canister 程序，但我们设计了一种名为 Motoko 的语言，它与 IC 的操作语义非常一致。Motoko 是一种强类型的，基于 actor 的 [3] 的编程语言，内置对正交持久性 [4]和异步消息传递的支持。正交持久性仅仅意味着由一个 Canister 维护的所有内存被自动持久化 (即，它不必写入一个文件)。Motoko 具有自动内存管理、泛型、类型推理、模式匹配以及任意和固定精度算术等生产性和安全性特点。

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数据格式，用于类型化、高级别和跨语言的互操作性。这允许任何两个 Canister，即使使用不同的高级语言，也可以很容易地相互沟通。

为了提供全面支持任何给定编程语言的 Canister 开发，除了该语言的 Wasm 编译器外，还必须提供特定的运行时支持。当前除了 Motoko 之外，IC 还全面支持了 Rust 编程语言的 Canister 开发。

[2]参见 https://webassembly.org/.

[3]参见 https://en.wikipedia.org/wiki/Actor_model/.

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

1.8 实用型通证

IC 使用一个名为 **ICP** 的**实用通证**。这个通证用于以下功能:

- **在 NNS 中质押:** 正如在第 1.5 节中已经讨论过的，ICP 令牌可以被质押在 NNS 中以获得投票权，从而参与控制 IC 网络的 DAO。拥有 ICP 令牌并参与 NNS 治理的用户也会收到新铸造的 ICP 令牌作为投票奖励。奖励的金额由 NNS 制定和执行的政策决定。
- **转换为 Cycles:** ICP 用于支付 IC 的使用费。更具体地说，ICP 通证可以转换为 Cycles (即燃烧 ICP)，这些 Cycles 用于支付创建 canisters (参见 1.7 节) 和 Canister 使用的资源 (存储、CPU 和带宽) 的费用。ICP 转换为 Cycles 的费率由 NNS 决定。
- **向节点提供商付款:** ICP 通证用于向节点提供商付款 – 那些拥有并操作承载组成 IC 的 replicas 的计算节点实体。NNS 定期 (当前是每月一次) 决定每个节点提供者应该接收的新生成的通证的数量，并将通证发送到节点提供者的帐户。根据 NNS 制定和执行的政策，支付代币的前提是节点提供者为 IC 提供可靠的服务。

1.9 边界节点

边界节点提供 IC 的网络边缘服务，尤其提供

- 明确定义的 IC 入口,
- IC 的拒绝服务保护,
- 传统客户端 (例如网页浏览器) 无缝访问 IC。

为了方便传统客户端对 IC 的无缝访问，边界节点将用户的标准 HTTPS 请求转换为指向 IC 上一个 canister 的输入消息，然后将这个输入消息路由到 canister 所在的子网上的指定 replicas。此外，边界节点或其他服务可以改善用户体验，例如缓存、负载平衡、速率限制，以及传统客户端验证 IC 响应的能力。

Canister 是通过 ic0.app 域上的 URL 来识别的。最初，传统客户端查找相应的 DNS 记录以获取 URL，获取边界节点的 IP 地址，然后向该地址发送初始 HTTPS 请求。边界节点返回一个基于 javascript "service worker"，这将在传统客户端中执行。在这之后，传统客户端和边界节点之间的所有交互都将通过这个 service worker 完成。

Service worker 执行的一项基本任务是使用链式密钥加密法对 IC 的响应进行身份验证 (参见第 1.6 节)。为了做到这一点，NNS 的公共验证密钥在 Service worker

中是硬编码的。

边界节点本身负责将请求路由到托管指定 canister 的子网上的 replicas。执行此路由所需的信息由边界节点从 NNS 获得。边界节点保持一个包含 replicas 的列表，这些 replicas 可以提供及时的回复，并从这个列表中选择一个随机的 replicas。

传统客户端和边界节点之间以及边界节点和 replicas 之间的所有通信都是通过 TLS [5]保护的。

除了传统客户端之外，还可以使用"IC native" 客户端与边界节点交互，这些客户端已经包含 service worker 逻辑，并且不需要从边界节点检索 service worker 程序。

就像 replicas 一样，边界节点的部署和配置是由 NNS 控制的。

1.10 更多关于 NNS 的资料

正如在 1.5 节中已经提到的，网络神经系统 (NNS) 是一个控制 IC 的算法治理系统。它是通过在一个特殊的系统子网上的一组 canisters 来实现的。这个子网和其他子网一样，但是它的设计有些不同 (例如，系统子网上的 canisters 使用是不需要支付 Cycles)。

NNS canisters 中最重要的一些:

- **Registry canister**，它负责存储 IC 配置，即哪个 replicas 属于哪个子网、与子网关联的公钥和单个 replicas，等等。
- **Governance canister**, 负责管理关于 IC 应如何发展的决策和投票。
- **Ledger canister**, 用于跟踪用户的 ICP 代币账户和账户之间的交易。

1.10.1 NNS 的决策

任何人都可以通过在所谓的神经元中质押 ICP 代币来参与 NNS 的治理。然后，神经元持有者可以提交并投票表决提案。这些提案内容是有关如何更改 IC 的建议，例如，如何更改子网拓扑或协议。神经元对决策的投票权基于 PoS。直观地说，质押更多的 ICP 代币的神经元有更多的投票权。然而，投票权也取决于一些其他神经元特征，例如，质押更长时间神经元持有者用户更多的投票权。

每个提案都有一个确定的投票期。如果在投票期结束时，参与投票的总投票权的简单多数投票赞成该提案，且这些赞成票构成总投票权的特定法定比例 (目前为 3 %)，则通过该提案。否则，提案将被否决。此外，当绝对多数 (占总投票权的一半以上) 分别赞成或反对某项提案，则在任何时候该提案均获得通过或否决。（译

[5]参见 https://en.wikipedia.org/wiki/Transport_Layer_Security.

者注：假设社区总人数为 1000 人，有两个提案分别是提案 A 和提案 B: 1) 参与 A 提案的投票人数为 100 人，其中投赞同票的人数为 51 人，超过了参与投票人数 100 人的 50%，即简单多数。2) 参与 B 提案的投票人数为 600 人，其中投赞同票的人数为 501 人，已经超过了社区总人数 1000 人的 50%，即绝对多数。在绝对多数的情况下，即使提案 B 的投票截止时间还没有到，我们也可以以绝对多数认可的结果直接通过提案。）

如果一个提案被采纳，Governance Canister 将自动执行该决定。例如，如果一建议更改网络拓扑的提案被采纳，那么 Governance Canister 将根据新的配置自动更新注册表。

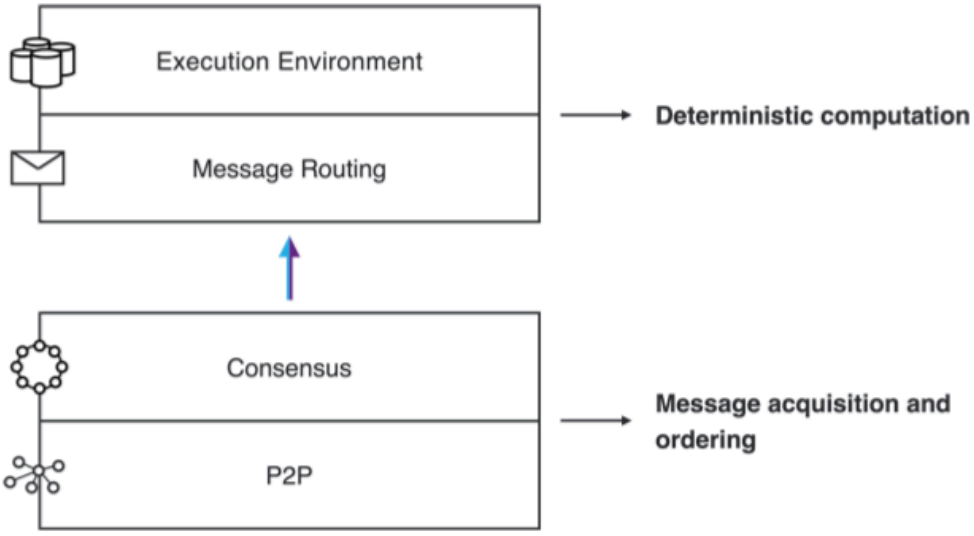

图 1: IC 协议层

1.11 正在进行的工作

IC 的体系结构仍在改进和扩展。以下是一些即将部署的新功能:

- **Dao 控制的 canisters**。就像 IC 的整体结构由 NNS 控制一样，任何 Canister 也可能由其自身的 DAO 控制, 又名服务神经系统 (SNS)。控制一个 canister 的 DAO 可以控制 canister 的逻辑更新，也可以下达专属权限命令让 Canister 执行。
- **阈值 ECDSA**。ECDSA 签名 [JMV01] 用于诸多加密货币，如比特币和以太坊，以及许多其他应用。虽然阈值签名已经是 IC 中的一个重要组成部分，但这

些不是阈值 ECDSA 签名。这个新功能将允许个人的 canister 控制 ECDSA 签名密钥，密钥被安全地分布在托管 canister 子网上的所有 replicas 中。

- **比特币和以太坊的整合**。基于新的 ECDSA 阈值功能，该功能将允许 canisters 与比特币和以太坊交互，包括能够签署交易。
- **HTTP 集成**。这个功能将允许 canisters 读取任意网页 (IC 外部的)。

第 2 章 架构概览

如图 1 所示，互联网计算机协议包含四层。

- P2P 网络层（详细见第 4 节）;
- 共识层（详细见第 5 节）;
- 路由层（详细见第 6 节）;
- 执行层（详细见第 7 节）。

2.1 P2P 层

P2P 层的任务是在同一个子网中的 replicas 之间传输协议消息。这些协议消息包含:

- 用于实现共识的消息;
- 由外部用户生成的输入消息。

基本上, P2P 协议提供的服务是一个最好的广播信道: 如果一个诚实的 replicas 广播一条消息，那么该消息最终将被子网中所有的诚实的 replicas 接收。设计目标包括如下内容:

- 有限资源。所有算法使用的资源（内存、带宽、CPU）必须是有限的。
- 优先级。不同的消息可能具有不同的优先级，这取决于特定属性（比如，类型，大小，取整），而且这些优先级可能会随时间变化。
- 效率。高吞吐量比低延迟更重要。
- DOS/SPAM。恶意的 replicas 不应阻止诚实的 replicas 间的相互通信。

2.2 共识层

互联网计算机的共识层的工作是对输入进行排序，以便子网中的所有 replicas 以相同的顺序处理输入。以往的文献中有很多关于这个问题的协议。互联网计算机使用一个新的共识协议，此处将对它进行一个概括性描述。

任何安全的共识协议都应该保证两个属性,（粗略地说）是:

- **Safety**：所有的 replicas 对输入顺序达成一致，并且,
- **Liveness**：所有的 replicas 应该持续运行。

互联网计算机协议设计的目标是:

- 非常简单

- 具有鲁棒性：当某些 replicas 作恶时，性能下降很平缓。

如上所述，我们假设 $f < n/3$ 个错误的 replica（例如拜占庭错误）。此外，在部分同步网络假设下，Liveness 得到保证。即使在完全异步的网络假设下，Safety 也能得到保证。

与许多共识协议一样，互联网计算机协议也是基于区块链的。随着协议的进展，区块链从一个特殊的创世区块开始，像一棵树一样生长，这个创世区块就像是这棵树的根。树中的每一个非创世区块都包含（除其他信息外）一个有效的信息负载（Payload）。该信息负载包含一系列的输入和该区块在树中的父区块的哈希。诚实的 replicas 对这棵树的视图是一样的：虽然每个 replicas 可能具有这棵树的不同部分的视图，但是所有 replicas 的视图都是同一棵树的视图。此外，随着协议的进展，此树中始终存在 **finalized**(最终确定的) 块的路径。同样，诚实的 replicas 对此路径具有一致的视图：虽然每个 replicas 可能具有此路径的不同部分视图，但所有 replicas 都具有相同路径的视图。沿此路径的区块的有效负载中的输入就是将由互联网计算机的执行层处理的有序输入。

互联网计算机协议按照轮次进行。在协议的第 h 轮中，一个或多个高度为 h 的区块被添加到树中。也就是说，在第 h 轮中添加的区块与根（创世区块）的距离正好是 h。在每一轮中，使用伪随机进程为每个 replicas 分配一个唯一的顺序值，范围为：$0, \ldots, n-1$。这个伪随机过程是使用 Random Beacon 实现的（这利用了阈值签名，在上面的 1.6.1 节中提到了，并会在第 3 节中进行更加详细的讨论）。顺序最靠前的 replicas 是这一轮的领导者。当作为领导者的 replicas 是诚实的，并且网络是同步的时候，领导节点会提出一个会被添加到树中的区块，而且这将是这一轮中唯一被添加到树中的区块，它将扩展最终路径。如果作为领导者的 replicas 不是诚实的，或者网络不同步，其他顺序靠后的 replicas 也可能提交区块，并且他们的区块也会被添加到树中。在任何情况下，协议的逻辑给予作为领导者的 replicas 提交的区块最高的优先级，并且在这轮中，一个或多个区块会被添加到树中。即使协议进行了几轮之后没有扩展最终确定的路径，树的高度依然会随着轮次增长，因此最终路径在第 h 轮扩展时，最终确定的路径的长度将为 h。因此，即使 replicas 作恶或者有意料之外的高网络延迟，也只会导致延迟时间偶尔地增加，但是协议的吞吐量基本保持不变。

互联网计算机的共识协议依赖于数字签名来验证 replicas 之间发送的消息。为了实现该协议，每个 replicas 都与一个签名方案的公共验证密钥相关联。replicas 与公钥的关联是从 NNS 维护的注册表中获取的。

2.3 消息路由

如第 1.7 节所述，互联网计算机（IC）中的基本计算单元称为 Canister。互联网计算机（IC）提供了一个运行时环境，用于在 Canister 中执行程序，以及与其他 Canister 和外部用户进行通信（通过消息传递）。

共识层将输入捆绑到信息负载中，这些信息负载被放入区块中，并且随着区块的最终确定，相应的信息负载被传递到消息路由层，然后由执行环境处理，执行环境更新 replicas 状态机上 Canister 的状态并生成输出，这些输出由消息路由层处理。区分以下两种类型的输入是必要的：

- 外部输入消息：这些是来自外部用户的消息；
- 跨子网消息：这些消息来自其他子网上的 Canister。

我们还可以区分两种类型的输出：

- 外部输入消息响应：这些是对外部输入消息的响应（外部用户可以取回这些消息）；
- 跨子网消息：这些是发给其他子网上的 Canister 的消息。

从共识接收信息负载后，该信息负载中的输入将被放入各种输入队列中。对于子网上运行的每个 Canister$C$，都有数个输入队列：一个用于向 C 输入消息，另一个用于与 C 通信的 CanisterC'，一个用于从 C' 向 C 发送跨子网消息。

在每一轮中，执行层将使用这些队列中的一些输入，更新相关 Canister 的复制状态，并将输出放置在各种输出队列中。对于子网上运行的每个 Canister C，都有多个输出队列：对于每一个与 C 通信的其他 Canister$C$，都有一个用于从 C 到 C 的跨子网消息。消息路由层将获取这些输出队列中的消息，并将它们放入跨子网数据流中，由跨网传输协议处理，跨网传输协议的工作实际上是将这些消息传输到其他子网。

除了这些消息消息输出队列，还有一个外部输入历史记录数据结构。Canister 处理完外部输入消息后，将对外部输入消息的响应记录在此数据结构中。此时，提供外部输入消息的外部用户将能够检索相应的响应。（请注意，外部消息历史记录不会维护所有外部消息的完整历史记录。）

请注意，replicas 状态包括 Canister 的状态，以及" 系统状态"，包括上述队列和流，以及外部消息历史数据结构。因此，消息路由层和执行层都涉及更新和维护子网的 replicas 状态。所有这些状态都必须以完全确定的方式更新，以便所有 replicas 保持完全相同的状态。

还要注意的是，共识层与消息路由层和执行层是分离的，从某种意义上说，区块链中的任何分叉都在其信息负载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，并且可以提前运行。

2.3.1 per-round 认证状态

在每一轮中，子网的某些状态将被验证。per-round 认证状态使用 Chain-Key 加密技术进行认证。除其他事项外，给定一轮的认证状态包括：

- 最近添加到跨子网数据流的跨子网消息；
- 其他元数据，包括外部输入消息历史数据结构。

per-round 认证状态使用阈值签名进行认证（参见第 1.6.1 节）。在 IC 中每轮经认证状态被用于如下几个方面：

- *输出验证*。跨子网消息和对外部输入消息的响应使用 per-round 认证状态进行验证。
- *阻止和发现非确定性*。共识保证每个 replicas 以相同的顺序处理输入。由于每个 replicas 确定性地处理这些输入，因此每个 replicas 都应获得相同的状态。但是，互联网计算机协议在设计上具有额外的鲁棒性，如果出现任何意外的非确定性计算，就可以及时发现和阻止。per-round 认证状态是用于做到这一点的机制之一。
- *协助共识*。per-round 认证状态还用于以两种不同的方式协调执行层和共识层：

  –如果共识层运行快于执行层（进度由上一轮经认证状态决定），共识层会被“降速”。

  –共识的输入必须通过某些有效性检查，这些有效性检查可能依赖于所有 replicas 都达成一致的已验证状态。

2.3.2 查询调用 (query calls) VS 更新调用 (update calls)

正如我们到前面所描述的那样，外部输入消息必须通过共识，以便子网上的所有 replicas 以相同的顺序处理它们。但是，当处理某些外部输入消息而不会修改子网的 replicas 状态的，可以采取一个重要的优化措施。这些消息称为查询调用，相反地，需要修改子网的 replicas 状态的被称为更新调用。允许查询调用读取并执行可能更新 Canister 状态的计算，但对 Canister 状态的任何更新永远不会提交到复制状态。因此，查询调用可以由单个 replicas 直接处理，而无需通过共识，这大大减少了查询调用的响应延迟。

通常，对查询调用的响应不会记录在外部输入消息历史记录数据结构中，因此无法使用如上所述的 per-round 认证状态机制进行验证。但是，互联网计算机协

议使 Canister 能够将数据（在处理更新调用时）存储在特殊的认证变量中，这些变量可以通过这种机制进行验证；因此，返回已认证变量作为其值的查询调用仍然可以进行验证。

2.3.3 外部用户验证

外部输入消息和跨子网消息之间的主要区别之一是用于对这些消息进行验证的机制不同。用 Chain-key 加密技术对跨子网消息进行验证，但使用不同的机制对来自外部用户的外部输入消息进行验证。

外部用户没有中央注册表。相反，外部用户使用用户标识符（也就是 principal），即公共签名验证密钥的哈希值，向 Canister 标识自己。用户持有相应的私有签名密钥，该密钥用于对外部输入消息进行签名。签名以及相应的公钥将与外部输入消息一起发送。互联网计算机会自动验证签名，并将用户标识符传递给相应的 Canister。随后 Canister 根据用户标识和入口消息中指定操作的其他参数，批准请求的操作。

新用户在与互联网计算机的首次交互期间会获得一个密钥对，并从公钥派生出其用户标识符。非新用户使用用户代理存储的密钥进行验证。一个用户可以使用签名委派将多个密钥对与单个用户标识相关联。这很有用，因为它允许单个用户使用相同的用户标识从多个设备访问互联网计算机。

2.4 执行层

执行层一次处理一个输入。此输入取自其中一个输入队列，并定向到一个 Canister 中。根据此输入和 Canister 的状态，执行环境会更新 Canister 的状态，还可以将消息添加到输出队列并更新外部输入消息历史记录（可能带有对较早外部输入消息的响应）。

每个子网都可以访问分布式伪随机生成器（PRG）。伪随机位派生自种子，种子本身就是称为 Random Tape 的阈值签名（请参见第 1.6.1 节和第 3 节中的更多详细信息）。共识过程中的每轮都有一个不同的 Random Tape。

Random Tape 的基本属性包括：

- 1. 在高度为 h 的区块被任何诚实的 replicas 确认之前，高度 h+1 的 Random Tape 是不可预测的。
- 2. 在高度为 h+1 的区块被任何诚实的 replicas 确认时，replicas 通常拥有它构造高度 h+1 的 Random Tape 需要的所有份额。

为了获取伪随机位，子网必须在某个轮次中通过执行层的" 系统调用" 对这些位发出请求，比如轮次 h。系统稍后会使用高度为 h+1 的 Random Tape 响应该请

求。根据上面的属性（1），可以保证请求的伪随机位在发出请求时是不可预测的。事实上，共识层会将 Random Tape 和 payload 在 $h+1$ 轮同时传递给消息路由层；根据上面的属性（2）事实上，共识层会将随机磁带和荷载都传递给消息路由层.

2.5 总结

我们追踪在 IC 上的用户请求的一般处理流程。

- 1. 用户对 Canister C 的请求 M 由用户的客户机发送到边界节点（请参见第 1.9 节），边界节点将 M 发送到托管 Canister C 的子网上的 replicas。
- 2. 收到 M 后，此 replicas 将使用 P2P 层将 M 广播到子网上的所有其他 replicas（请参阅第 2.1 节）。
- 3. 收到 M 后，作为下一轮共识的领导者（leader）（见第 2.2 节）的 replicas 将把 M 与其他输入捆绑在一起，形成领导者提出的区块 B 的信息负载。
- 4. 一段时间后，区块 B 最终确定，信息负载被发送到消息路由层（参见第 2.3 节）进行处理。请注意，P2P 层也被用于确认该区块。
- 5. 消息路由层将 M 放在 Canister C 的输入队列中。
- 6. 一段时间后，执行层（参见第 2.4 节）将处理 M，更新 Canister C 的内部状态。

  在某些情况下，Canister C 将能够立即计算出请求 M 的响应 R。在这种情况下，R 被放置在外部输入消息历史数据结构中。

  在其他情况下，处理请求 M 可能需要向另一个 Canister 发出请求。在此示例中，让我们假设要处理此特定请求 M，Canister C 必须向驻留在另一个子网上的另一个 Canister C 发出请求 M。第二个请求 M 将放在 C 的输出队列中，然后执行以下步骤。
- 7. 一段时间后，消息路由会将 M 移动到适当的跨子网数据流中，并且最终将传输到托管 C 的子网。
- 8. 在第二个子网上，请求 M 将从第一个子网获取，并最终通过第二个子网上的共识和消息路由，然后通过执行进行处理。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，并最终放置在跨子网流中并传输回第一个子网。
- 9. 回到第一个子网，响应 R 将被从第二个子网获取，并最终在第一个子网上通过共识和消息路由，然后通过执行进行处理。执行层将更新 Canister C 的内部状态，并生成对原始请求消息 M 的响应 R。此响应将记录在外部输入消息历史记录数据结构中。

无论采用哪种执行路径，对请求 M 的响应 R 最终都将记录在承载 Canister C 的子网上的外部输入消息历史记录数据结构中。要获得此响应，用户的客户端必须执行" 查询调用"（请参见第 2.3.2 节）。如第 2.3.1 节中所述，此响应将通过 Chain-key 加密技术（即，使用阈值签名）进行验证。验证逻辑操作 (即阈值签名验证) 将由最初从边界节点获得服务线程的客户端充分利用此服务线程来执行。

第 3 章 Chain-key 密码学 1: 门限签名

门限签名算法 [Des87] 是 IC chain-key 密码学中的一个重要内容。门限签名算法在 IC 中有很多用途。假设一个 subnet（子网）中有 n 个 replicas（节点），其中最多有 f 个恶意节点，则门限签名可用于下述几个方面：

- 共识层使用 $(f+1)-out-of-n$ 门限签名来生成一个 random beacon（不断产生随机数）（参见 5.5 小节）。
- 在执行层使用 $(f+1)-out-of-n$ 门限签名来生成一个为 canister 提供伪随机数的 random tape（参见 7.1 小节）。
- 在执行层使用 $(n-f)-out-of-n$ 门限签名来证明链上状态已被共识验证（certify the replicated state）。同时它被用于验证 subnet 的输出（见 6.1 小节），和赋予 IC chain-evolution 技术中的 fast-forwarding 属性。

对于前两个应用（random beacon 和 random tape），我们需要一个确定性的门限签名，给定同一个公钥和消息，只会有一个合法的签名。（译者注：同一私钥对于相同消息生成的签名是唯一的。）因为我们需要使用签名来作为伪随机数生成器的种子，并且所有的 replicas 计算门限签名必须基于相同的种子。

3.1 BLS 门限签名

我们基于 BLS 签名算法 [BLS01] 实现门限签名，该算法对于门限的设置没有特殊的要求。

一般的 BLS 签名（例如，非门限的签名）使用两个阶为素数 q 的群 $\mathbb{G}$ 和 $\mathbb{G}'$。设 $\mathbb{G}$ 的生成元 $g \in \mathbb{G}$，$\mathbb{G}'$ 的生成元 $g' \in \mathbb{G}'$。设哈希函数 $H_{\mathbb{G}'}$ 将输入映射为 $\mathbb{G}'$ 中的一个元素（假设这种映射的结果是一个随机数，这种假设我们可以称之为 random oracle 模型）。签名密钥为 $x \in \mathbb{Z}_q$，验证密钥为 $V := g^x \in \mathbb{G}$。

在非门限情况下，签名者计算 $h' \leftarrow H_{\mathbb{G}'}(m) \in \mathbb{G}'$，然后计算签名 $\delta := (h')^x \in \mathbb{G}'$。验证者检验是否 $log_{h'}\delta = log_g V$。为了提高验证的效率，BLS 签名算法使用了配对的概念，在 $\mathbb{G}$ 和 $\mathbb{G}'$ 是特殊的椭圆曲线时可以使用配对这种代数工具。在此，我们不深究配对和椭圆曲线的细节，详细信息参见 [BLS01]。BLS 签名具有一个非常好的性质：它的签名是唯一的。

在 $t-out-of-n$ 的情况下，我们有 n 个 replicas，其中任意的 t 个 replicas 可以联合起来对一个消息生成签名。具体为：每一个 replica P_j 持有一个整体密钥 $x \in \mathbb{Z}_q$ 的分享密钥 $x_j \in \mathbb{Z}_q$，分享公钥为 $V_j := g^{x_j}$。x_j 由 replicasP_j 持有并保证其

安全性，V_j 为公开信息。分享密钥 $(x_1, \ldots, x_n)$ 是密钥 x 的 $t - out - of - n$ 密钥分享（见 3.4 节）。

给定一个消息 m，replicas（节点）P_i 可以生成一个**分享签名**:

$$\delta_j := (h')^{x_j} \in \mathbb{G}'$$

其中 $h' := H_{\mathbb{G}'}(m)$。为了验证每一个分享签名是否正确，需要验证 $log_{h'}\delta_j = log_g V_j$。这可以通过上文中的配对进行验证，这其实和一般情况的下 BLS 算法在公钥为 V_j 的情况下的验证过程相同。

BLS 算法的优势在于拥有下述的**重构属性：**

给定同一个消息 m 的任意 t 个合法的分享签名 δ_j（由不同的 replica 生成的），我们可以高效地计算出消息 m 在验证密钥下的合法 BLS 签名 δ，即

$$\delta \leftarrow \prod_j \delta_j^{\lambda_j} \tag{3-1}$$

其中，λ_j 可以由 t 个 replicas 的指数，高效计算得出。

在对于 $\mathbb{G}$ 合理假设以及 $H_{\mathbb{G}'}$ 满足 random oracle 模型的条件下，该协议满足下列**安全属性：**

假设最多有 f 个 replicas 被敌手攻陷。除非这则消息拥有来自至少 t-f 个诚实 replicas 的签名分享，敌手才能计算出这个消息的合法签名。

3.2 分布式密钥分发（Distributed key distribution DKG）

为了实现门限签名，我们需要向 replicas 分发签名密钥。一种方式是使用一个可信的第三方直接计算分享密钥并且分发给 replicas。但是这样会存在单点错误。互联网计算机中采用一种分布式密钥分发协议，这种协议可以使 replicas 高效地实现这样的可信第三方的协议。

我们简要说明该协议现阶段的实现方式。详情请参见 [Gro21]。DKG 协议是一种高效的非交互式协议，它由两个基本部分组成:

- 公开可验证密钥分享（publicly verifiable secret sharing PVSS）协议
- 共识协议

尽管任何一个共识协议都具有可行性，但是我们采用了第 5 节中的共识协议（见第 8 节）。

3.3 假设

基础假设和第 1 节中的一样:

- 异步通信
- $f < n/3$

我们只在保证共识协议的 liveness 时使用部分同步假设。

假设存在一个 $t-out-of-n$ 的门限签名算法，

$$f < t \leq n - f$$

有以下结论:

- （1）恶意 replicas 无法独立生成签名。
- （2）诚实 replicas 可以在恶意 replicas 参与的情况下生成签名。

假设，每一个 replica 都和一些公钥关联并且持有公钥相对应的私钥。一个公钥用于验证签名（参见 5.4 节），另一个公钥用于实现 PVSS 算法的特殊公钥加密算法的公开加密密钥。

3.4 PVSS 算法

设 $\mathbb{G}$ 为素数阶 q 的群，其中 $g \in \mathbb{G}$。设 $s \in \mathbb{Z}_q$ 为密钥。在 $t-out-of-n$ Shamir 秘密分享协议中，s 为向量 $(s_1, \ldots, s_n) \in \mathbb{Z}_q^n$。其中,

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

并且，

$$a(x) := a_0 + a_1 x + \cdots + a_{t-1} x^{t-1} \in \mathbb{Z}_q[x]$$

是一个阶小于 t 的多项式，并且零次项 $a_0 := s$。该秘密分享协议的核心特性为:

- 可以从任意 t 个 s_j 中，高效地计算出秘密 $s = a_0 = a(0)$（通过多项式差值）
- 如果 $a_1, \ldots, a_{t-1}$ 是均匀选取并且在 $\mathbb{Z}_q$ 中不相关，那么任意少于 t 个 s_j 的集合不会暴露任何关于秘密 s 的信息。

PVSS 协议允许一个被称为 dealer 的 replicasP_i，做如下秘密分享，并计算得到一个称为 dealing 的结构:

- 一个群元素 $(A_0, \ldots, A_{t-1})$ 的向量，其中 $A_k := g^{a_k}, k = 0, \ldots, t-1$;
- 一个密文 $(c_1, \ldots, c_n)$ 的向量，其中 c_j 是 s_j 在 P_j 的公开加密密钥下的密文;
- 一个非交互式的零知识证明 π，每个 c_j 确实加密了这样的秘密片段—更准确地说，每个 c_j 都加密了 s_j 的值，即满足

$$g^{s_j} = \prod_{k=0}^{t-1} A_k^{j^k} = g^{a(j)} \tag{3-2}$$

为了建立 DKG 协议的整体安全，PVSS 协议必须提供一个合适级别的选择密文安全（CCA 安全）。即 dealer 必须在其 dealing 中，附带能够验证其身份的 associated data，并且被加密的分享必须被隐藏。即使在选择密文攻击的情况下，即攻击者可以解密任意的 dealing，也无法区分用于生成 dealing 的 associated data。

如果不考虑效率，实现一个 PVSS 很容易。主要的思路为，使用 ELGamal 类型的加密算法一个比特一个比特地去加密每一个 s_i，然后使用标准的非交互式零知识证明来证明 relation（3-2），这个零知识证明可以基于一个标准的 Fiat-Shamir 转换 [FS86] 和一个合适的 Sigma 协议 [CDS94]。但是这实现方式拥有多项式时间的复杂度，在实践中非常不可取。然而，有很多可行的方式来优化这类算法，有关 IC 中使用的高度优化的 PVSS 方案的详细信息，请参见 [Gro21]。

3.5 基础 DKG 协议

因为使用了 PVSS 协议和一个共识协议，所以基础 DKG 协议的实现非常简单：

- 1. 每一个 replica 向所有其他 replicaS 广播一个随机秘密的被签名的 dealing。每一个签名过的 dealing 包含一个 dealing、dealer 的身份信息以及一个 dealer 的签名（dealer 公开签名密钥下的签名）。
  这样签名过的 dealing 如果符合语法格式，签名验证通过并且非交互式零知识证明是合法的，那么我们就称之为合法的 dealing。
- 2. 使用协议，replicas 对来自不同的 dealer 的 $f + 1$ 个合法的 dealing 的集合 S 达成共识；
- 3. 假设 S 中，第 i 个 dealing 包含群元素向量 $A_{i,0}, \ldots A_{i,t-1}$ 以及密文向量 $(c_{i,1}), \ldots c_{i,n}$。门限签名的公开验证密钥为

$$V := \prod_i A_{i,0}$$

  注意，签名密钥精准的定义为：

$$x := log_g V$$

  P'_j 关于签名密钥 x 的分享为:

$$x_j := \sum_i s_{i,j}$$

  其中 $s_{i,j}$ 是 $c_{i,j}$ 在 P_j 解密密钥下，解密得到的原文。
  replica P_j 的公开验证密钥为：

$$V_j := \prod_i \prod_{k=0}^{t-1} A_{i,k}^{j^k} = g^{x_j}$$

注意分享密钥 (share)x_j 包含 x 的 $t-out-of-n$Shamir 秘密分享。例如公式（1）中出现的 λ_j 是拉格朗日差值的系数。这构成了 3.1 节中声明的重构特性。在 $H_{\mathbb{G}}$ 为 random oracle 模型下，在 PVSS 算法是安全的，且群 $\mathbb{G}$ 和 $\mathbb{G}'$ 满足一种特殊的 one-more Diffie-Hellman 强假设的情况下，3.1 节中声明的安全性可以被证明。one-more Diffie-Hellman 强假设为：没有有效的敌手可以在下述的游戏中以不可忽略的概率获胜：

挑战者随机选取 $\mu_1, \ldots, \mu_k \in \mathbb{Z}_q$ 和 $v_1, \ldots, v_l \in \mathbb{Z}_q$，并且将 $\{g^{\mu_i}\}_{i=1}^{k}$ 和 $\{(g')^{v_j}\}_{j=1}^{l}$ 发送给敌手。

敌手发送一个请求序列给挑战者，每个请求是一个形如 $\{\kappa_{i,j}\}_{i,j}$。挑战者返回：

$$\prod_{i,j}((g')^{\mu_i,v_j})^{\kappa_{i,j}}$$

敌手输出一个向量 $\{\lambda_{i,j}\}_{i,j}$ 和一个群元素 $h' \in \mathbb{G}'$，并且在输出向量 $\{\lambda_{i,j}\}_{i,j}$ 不是请求向量的线性组合且

$$h' = \prod_{i,j}((g')^{\mu_i,v_j})^{\lambda_{i,j}}$$

时，敌手赢得游戏。

在 $t > f+1$ 时需要这种 one-more Diffie-Hellman 假设，在 $t = f+1$ 时，可以使用一个更弱的假设，我们称之为 co-CDH 假设，普通的 BLS 算法是基于 co-CDH 假设的。

3.6 重新分享协议

基础的 DKG 协议可以轻易地被改造为对一个密钥 x，重新产生新随机分享密钥的协议：

- 步骤一改为每一个 replica 广播已有共享片段的已签名 dealing。
- 步骤二改为对一个 t 个合法的、被签名的 dealing 的集合达成一致。并且每一个 dealing 被验证确实是现有的分享的合法 dealing（即在第 i 个 dealing 中的值 $A_{i,0}$ 等于旧的 V_i 的值）。
- 步骤三新的 x_j（和 V_j）的计算对拉格朗日差值的第 i 个系数进行加权（和相乘）。

第 4 章 P2P 层

P2P 层的任务是在子网中的 replicas 节点之间传输协议消息。这些协议消息包括:

- 用于实现共识的消息，例如区块提案、公证等（请参阅第 5 节）；
- 外部输入消息（请参阅第 6 节）。

P2P 提供的服务是一个" 尽最大努力" 的广播频道：如果诚实 replicas 节点广播一条消息，则子网中的所有诚实 replicas 最终都会收到该消息。

设计目标包括:

- **有限资源**。所有算法都必须使用有限资源（内存、带宽、CPU）。
- **优先级**。不同的消息可能具有不同的优先级，这取决于特定属性（比如，类型，大小，轮次），而且这些优先级可能会随时间变化。
- **效率**。高吞吐量比低延迟更重要.
- **抗 DOS/SPAM**。诚实的 replicas 节点之间的互相通信不应该受到恶意节点存在的影响。

请注意，在共识协议中，某些消息，特别是区块提案（可能相当大），将由所有 replicas 重新广播。这对于确保该协议的正确是必要的。但是，如果只是简单地直接广播，这将是对资源的巨大浪费。为了避免所有 replicas 广播相同的消息，P2P 层使用 “广播-请求-传递” 机制。它不是直接广播（大）消息，而是广播消息的（小）摘要：如果 replicas 收到这样的小摘要，此时尚未收到消息，并认为消息很重要，它将请求传递消息。此策略以更高的延迟为代价降低了带宽使用。但是对于小消息，这种权衡是不值得的，直接发送消息全文而不是消息的小摘要更有意义。

在相对较小的子网中，希望广播消息的 replicas 将向子网中的所有 replicas 发送消息的摘要，然后每个 replicas 都可以请求传递消息。对于较大的子网，此 “播发-请求-传递” 机制可能通过覆盖网络运行。覆盖网络是一个连接的无向图，其顶点组成子网中的 replicas。如果此图中有一条边连接两个 replicas，则它们是相邻 replicas，并且 replica 仅与其相邻 replica 通信。因此，当 replicas 希望广播消息时，它会向其相邻 replicas 发送该消息的摘要。这些相邻 replicas 可能会请求传递消息，并且在收到消息后，如果满足某些条件，这些相邻 replicas 将向其相邻 replica 转发消息。这本质上是一个 gossip 网络。此策略再次降低了带宽的使用，但代价是更高的延迟。

第 5 章 共识层

IC 共识层的工作为，对输入进行排序，使子网中所有 replica 按照相同的顺序处理输入。目前有很多协议可以处理这个问题。本文简要概述了 IC 使用的共识协议，详情参见 [CDH+21]（具体为论文中的 ICC1 协议）。

任何安全的共识协议应该满足下述两个属性：

- Safety（安全性）：所有的 replica 对输入的顺序达成一致；
- Liveness（活性）：所有的 replica 应该稳定的执行；

[CDH+21] 证明了 IC 共识协议满足上述两个性质。

IC 共识协议的特点为：

- 简洁；
- 鲁棒：存在恶意 replica 的情况下，性能下降幅度较小；

5.1 假设

同简介中的讨论的一样，我们假设：

- 子网有 n 个 replica
- 最多有 $f < n/3$ 的 replica 发生故障。

故障可以为任何类型，例如被敌人控制（拜占庭 replica）、replica 宕机等。

我们假设通信为异步的，即没有一个消息在 replica 之间传递的网络延迟的先验上界。事实上，消息传递策略可以完全在敌手的控制下。IC 共识协议在非常弱的通信假设下，能够保证 safety（安全性）。为了保证 liveness（活性），我们需要部分同步假设的通信模型，即网络会周期性的在一个较短时间间隔内同步。在同步的时间间隔内，所有没有被传输的消息会在 δ 时间内被传递。其中 δ 是网络延迟时间的上界，并且它不需要提前知道（协议会初始化一个合适的数值，并且会动态的调整，在其过小的时候调大）。忽略异步和半同步假设，我们假设每一个由诚实 replica 发往其他 replica 的消息最终会被送达。

5.2 协议概览

同很多共识协议一样，IC 共识协议基于区块链。随着协议的执行，一个区块组成的树不断地生长，树根是一个特殊的创世区块。每一个树中的非创世区块包含一个由有序输入构成的 payload（负载）以及一个树中父区块的哈希。诚实 replica

拥有区块树的一致性视图：尽管每个 replica 可能拥有区块树的不同部分视图，但是所有的 replica 的视图是基于相同的区块树。此外，随着协议的进行，树中总会出现由 finalized（确定的）状态的区块组成的路径，并且诚实 replica 会拥有这个路径的一致性视图：尽管每个 replica 可能拥有路径的不同部分，但是所有 replica 的视图是基于相同的路径。路径中区块的 payload 中的输入会在互联网计算机的执行层中（见 7 节）按顺序执行。

该协议会按照轮次执行。在协议的第 h 轮中，一个或者更多的高度为 h 的区块被添加到树中，即在第 h 轮中被添加的区块到根区块的长度等于 h。每轮中，用一个伪随机数来为每一个 replica 确定 rank（产生区块的优先级），该 rank 为 $0,\ldots,n-1$ 中的一个整数。这个伪随机数由 random beacon 产生（见 5.5 节）。每轮中，rank 最低的 replica 称之为 leader。当 leader 为诚实的，并且网络是同步的情况下，leader 提交一个区块，该区块将被加入到区块树中。该轮有且只有这一个区块被加入到树中并且 finalized 的路径将延长。如果 leader 不是诚实的或者网络不是同步的，那么其他拥有更高 rank 的 replica 可以提交区块，并且将这些区块加入到区块树中。在任何情况下，本轮中由 leader 提交的区块拥有加入区块树的最高优先权。就算协议执行了几轮依然没有扩展 finalized 的路径，树的高度在每轮中依然会持续增长，这使得当 finalized 的路径在 h 轮被扩展，那么 finalized 的路径的长度将延长至 h。因此，尽管由于 replica 故障和过高的网络延迟，交易的确认延迟会增大，但是协议的吞吐率将保持基本不变。

5.3 额外的性质

IC 共识协议另一个非常好的性质为 optimistic responsiveness[PS18]（一些共识协议例如，PBFT[CL99]、HostStuff[AMN+20] 拥有这个性质，另一些协议例如 Tendermint[BKM18] 没有这个性质）。该性质表明，当 leader 为诚实时，协议以实际的网络延迟速度执行而非网络延迟的上界速度。

我们注意到 IC 共识协议保证，即使出现拜占庭 replica 时，协议的执行的效率下降会控制在一个合理的范围内。[CWA+09] 中指出，目前很多对于共识协议的研究更多关注，在没有错误的最优情况下协议效率。但是这样的共识协议是十分脆弱的，在实际工作中发生故障时可能会导致系统不可用。例如 [CWA+09] 中指出，一种特定的拜占庭 replica 出现时，现存的 PBFT 协议的吞吐率会下降到 0。论文 [CWA+09] 倡导鲁棒的共识协议，为保证当恶意 replica 出现时（但是依然假设网络是同步的）系统依然可以维持一个合理的吞吐率，牺牲了在最佳情况下一定的峰值效率。IC 共识协议在 [CWA+09] 讨论中的情况下是鲁棒的：在任意 leader 为恶

意 replica 的轮次（这种情况发生的概率小于 1/3），协议可以高效地允许其他的参与者接管该轮 leader 的工作，并且快速地推进协议的下一轮工作。

5.4 公钥

为实现共识协议，每个 replica 都需要和一个 BLS 签名算法 [BLS01] 的公钥关联，并且每个 replica 持有对应的签名密钥。公钥和 replica 的关联关系可以从记录在 NNS（见 1.5）的注册表中获得。这些 BLS 签名用于验证发送自 replica 的消息。

共识协议同时利用了 BLS 签名 [BGLS03] 的可聚合性质，该性质可以将多个对于相同消息的签名聚合成一个压缩的多签。在共识协议的 notarization（见 5.7）和 finalization（见 5.8）阶段，使用这些由 n-f 个对于相同消息的签名聚合而来的多签。

5.5 random beacon

除了上文中讨论过的 BLS 签名和多签，IC 的共识协议使用一种 BLS 门限签名算法来实现上文中提及的 random beacon。高度 h 的 random beacon 是 “高度 h 唯一消息”（这个消息是和高度唯一绑定的）的 $(f+1)$ 门限签名。在每轮中，每个 replica 广播其和下一轮相关的 beacon 的份额（shares），在下一轮开始时，所有 repilcas 将会收到足够的份额来重构该轮的 random beacon。高度 h 的 random beacon 用于为每个 replica 分发一个伪随机的 rank（等级），这个 rank（等级）在协议的第 h 轮中使用。由于门限签名的安全性，敌手无法提前一个以上的轮次预测 replica 的 rank（等级），rank（等级）可以被看作是随机分配的。参见 3 节了解 BLS 门限签名的更多细节。

5.6 区块的生成

每个 replica 在不同的时间节点充当区块生成者的角色。作为第 h 轮的区块生成者（生成区块的 replica），该 replica 提交一个高度为 h 的区块 B，区块 B 为区块构成的树中的一个高度为 h-1 的区块 B’的子节点。区块生成者首先收集它知道的、所有不包含在从根区块到区块 B’的 payload（负载）中的输入，并放入区块 B 的 payload（负载）中。区块 B 包含：

- payload（负载）
- 区块 B’的哈希
- 区块生成者的 rank（等级）

- 区块的高度 h

生成一个区块后，区块生成者生成一个区块的提案，内容如下：

- 区块 B
- 区块生成者的身份信息
- 区块生成者对于区块 B 的签名

区块生成者向所有其他的 replicas 广播这个区块的提案。

5.7 Notarization

一个区块 B 成为 notarized 状态时，B 才会被加入区块树中。必须有 $n - f$ 个不同的 replication 支持这个区块的 notarization，这个区块才会变为 notarized 状态。

给定一个高度为 h 的被提交的区块，replica 会判断这个提案是否合法，即区块满足上述的语义结构。具体为，区块 B 应该包含已经在区块树中，高度为 h' 的区块 B' 的哈希（例如：已经为 notarized 状态的区块）。此外，区块 B 的 payload（负载）必须满足一定的条件（payload 中的所有输入必须满足各种约束，但是这些约束通常和共识协议无关）。并且区块生成者的 rank（记录在区块 B 中）必须和第 h 轮中 random beacon 分配给提交区块的 replica 的 rank（记录在区块的提案中）匹配。

如果区块是合法的并且满足其他的约束，replica 通过广播一个区块 B 的 notarization 的分享来支持这个区块成为 notarized 状态，区块 B 的 notarization 分享的内容为：

- 区块 B 的哈希
- 区块 B 的高度
- 支持者（replica）的身份信息
- 支持者（replica）对于一条包含区块 B 的哈希和区块 B 的高度的消息的签名

任何 $n - f$ 个对于区块 B 的 notarization 份额可以被聚合形成一个 B 的 notarization，其内容为：

- 区块 B 的哈希
- 区块 B 的高度
- $n - f$ 个支持者（replica）的身份信息的集合
- $n - f$ 个支持者（replica）对于一条包含区块 B 的哈希和区块 B 的高度的消息的签名的聚合

只要 replica 获得一个高度为 h 的，notarized 的区块，它将结束第 h 轮，并且不再支持任何其他高度为 h 的区块的 notarization。这个 replica 将这个 notarization 转发

给其他所有的 replica。需要注意的是，这些 replica 可能已经获得了这个 notarization（1）接受自其他 replica；（2）由它自己收到的 $n - f$ 个 notarization 的分享聚合生成。

Growth Invariant（增长不变特性）说明每一个诚实 replica 最终会完成每一轮执行并且开始下一轮协议，以保证区块树中 notarized 的路径持续增长（该性质只需要假设异步的消息最终会被送达到所有的诚实 replica，而不需要网络同步假设）。我们在 5.11.4 节中证明 Growth Invariant。

5.8 Finalization

目前我们的协议，在同一高度可能出现不止一个 notarized 的区块。但是，当区块变为 finalized 状态时，能够确保在高度 h 只有一个 notarized 的区块。我们称之为 safety invariant（安全不变性）。

区块想要变为 finalized 状态，必须有 n-f 个不同的 replica 支持它的 finalization。前文说到，当 replica 包含一个高度为 h 的 notarized 区块 B，它将结束第 h 轮。此时，replica 检查它是否支持其他高度为 h 的区块的 notarization（这个 replica 可能没有支持 B 区块的 notarization）。如果这个 replica 没有支持别的区块的 notarization，那么它将广播 B 的 finalization 份额来支持区块 B 的 finalization。finalization 份额的格式同 notarization 份额的格式相同（通过标签来区分两种份额）。任何 n-f 个区块 B 的 finalization 份额可以被聚合成一个区块 B 的 finalization，其格式和 notarization 格式相同（同样通过标签区分）。任何 replica 包含一个 finalized 状态的区块将广播这个区块的 finalization。

我们将在 5.11.5 节中证明协议的 safety invariant。Safety invariant 的性质如下：假设两个区块 B、B'，其中 B 的高度为 h，B' 的高度 $h' < h$。Safety invariant 表明区块树中以 B' 为结尾的路径是以 B 为结尾的路径的前缀。（否则，在高度 h' 会有两个 notarized 状态的区块，这和 finalization invariant 性质矛盾）。因此，当一个 replica 观测到一个 finalized 状态的区块，那么该区块的所有祖先区块都会立刻变为 finalized 状态。同时因为 safety invariant 性质，这些 finalized 状态的区块的 safety 性质得到满足，即所有的 replica 对这些 finalized 状态的区块的顺序达成一致。

5.9 延迟函数

本文中的共识协议使用了两个延迟函数 Δ_m 和 Δ_n，这两个延迟函数控制了区块的生成和 notarization 行为的触发时间。这两个函数将 replica 的 rank r 映射到一个非负整数的延迟，并且这两个函数在 r 上是单调递增的，且 $\Delta_m(r) \leq \Delta_n(r)$，当

$r = 0, \ldots, n-1$。一种推荐的定义为：$\Delta_m(r) = 2\delta r$，$\Delta_n(r) = 2\delta r + \epsilon$。其中，$\delta$ 是消息在 replica 之间传递的时间上界，$\epsilon > 0$ 是控制协议不要运行过快的 “调节器”。在此定义下，liveness（活性）将在下述的轮次中得到满足：（1）leader 是诚实的，（2）消息在 δ 时间内被传递至诚实 replica。如果（1）（2）得到满足，那么该轮中由 leader 提交的区块将会成为 finalized 状态。我们称之为 liveness invariant。我们在 5.11.6 节中给出证明。

5.10 一个例子

图 2 中，每一个区块用它的高度（30, 31, 32）和区块生成者的 rank（等级）进行标记。该图用符号 N 标记那些状态为 notarized 的区块，即该区块至少有 $n - f$ 个不同的 replica 支持他的 notarization。可以看到，在同一个高度，可以有多个 notarized 状态的区块。例如，在高度 32，可以看到两个 notarized 状态的区块，他们的区块生成者的 rank（等级）分别为 1 和 2。同样的情况也发生在高度 34。可以观察到，在高度为 36 的区块达到了 finalized 状态，我们使用符号 F 来标记。区块成为 finalized 状态表明：有 $n - f$ 个不同的 replica 支持该区块的 finalization，并且（至少他们中诚实的 replica）没有支持其他的区块的 notarization。图中所有灰色的区块都为 finalized 状态。

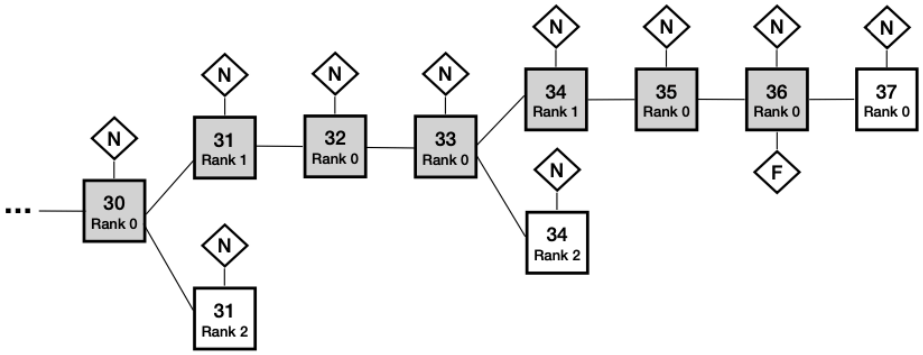

图 2: 区块树举例

5.11 结合起来

我们将介绍更多协议运行的细节：replica 何时会提交一个区块、replica 何时会支持一个区块的 notarization、给定一个 replica P，（1）它何时会进入给定的轮次

h：他获得一个高度为 $h-1$ 的区块的 notarization，（2）第 h 轮的 random beacon 如何操作。由于 h 轮的 random beacon 已经被确定，P 可以知道第 h 轮中自己的 rank（等级）r_p，以及其他的 replicaQ 的等级 r_Q。

5.11.1 random beacon 的细节

当收到 h 轮的 random beacon 或者足够的能够重构 h 轮 random beacon 的份额（shares），replica 会立刻将 h 轮的 random beacon 转发给所有的 replica。当一个 replica 进入 h 轮，它会立刻生成并转发 $h+1$ 轮的 random beacon 的份额（shares）。

5.11.2 区块生成的细节

replica P 只有满足下述条件时才会提交自己的区块 B_p：（1）自该轮开始起，至少过去 $\Delta_m(r_P)$ 的时间（2）没有收到更低 rank（等级）区块生成者生成的区块。

由于 replicaP 在进入 h 轮之前确保有一个 notarized 的区块 B，P 提交的区块以区块 B 作为父区块（如果有多个 notarized 的区块，也可选择其他的）。当 P 广播区块 B_P 的提案时，需要确保 B_P 父节点的 notarization 已经被传递。

假设一个 replicaQ 从 replicaP 收到了一个区块的提案，并且 P 的 rank（等级）高于 Q 的 rank（等级），即 $r_P < r_Q$ 时（1）从该轮开始到收到这个提案至少过去了 $\Delta_m(r_P)$ 时间。（2）目前 Q 没有看到 rank（等级）小于 r_P 的 replica 发出的提案。当满足上述两个条件时，Q 将向所有的 replica 转发这个区块的提案（以及这个区块的父区块的 notarization）。

5.11.3 Notarization 的细节

当 replica Q 提交的区块 B_Q 满足下述条件时，replica$P$ 将支持 B_Q 的 notarization：（1）自区块开始至收到 B_Q，时间间隔至少为 $\Delta_n(r_Q)$。（2）目前 P 没有收到 rank 小于 r_Q 的由 replica 提交的区块。

5.11.4 Growth invariant 的证明

Growth invariant 性质说明每一个诚实 replica 最终会结束一轮协议并且开启下一轮（协议会持续运行）。假设所有诚实 replica 已经开始了 h 轮协议。设 h 轮诚实 replica 中 rank 最低的 replica 为 P*，它的 rank 为 r*。最终，P* 会出现下述行为之一（1）提交它自己的区块。（2）转发一个由更低 rank 的 replica 提交的区块。任何一种情况，都会导致一些区块最终会被所有诚实 replica 支持，这意味着一些区块会成为 notarized 状态并且所有的 replica 都会结束 h 轮。所有诚实的 replica 会收到

h+1 轮所需的 random beacon 的分享并且开启 h+1 轮。

5.11.5 Safety invariant 的证明

Safety invariant 性质说明如果一个区块在一轮协议中成为 finalized 状态，那么在该轮不会有其他的区块成为 notarized 状态。证明如下：

- 1. 假设恶意 replica 的数量 $f^* \leq f \leq n/3$。
- 2. 如果区块 B 成为 finalized 状态，那么他的 finalization 被一个至少由 $n-f-f^*$ 个诚实 replica 所组成的集合 S 支持（可以使用聚合签名的安全特性）。
- 3. 假设（产生冲突），另一个区块 $B' \neq B$ 也成为了 notarized 状态。那么 B' 的 notarization 必须被一个至少由 $n-f-f*$ 个诚实的 replica 的集合 S' 支持（同样可以使用聚合签名的安全特性）。
- 4. S 和 S' 不相交（finalization 的逻辑所决定）
- 5. 因此有 $n-f^* \leq |S \cup S'| = |S| + |S'| \leq 2(n-f-f*)$，得到 $n \leq 3f$，这与我们的假设-恶意 replica 数量小于 replica 总数的 1/3-相矛盾。Safety invariant 性质得到证明。

5.11.6 liveness invariant 的证明

如果在时间 t 或时间 t 之前由诚实 replicas 发送的所有消息在时间 t 之前到达目的地，我们就说网络在时间 t 时为 $\delta-$ **synchronous**。

Liveness invariant 定义如下：假设 $\Delta_n(0) \geq \Delta_m(0) + 2\delta$，假设在给定轮次 h，有：

- leader P 是诚实的
- 第一个进入 h 轮的诚实 replica 在 t 时间进入 h 轮
- 网络在 t 和 $t + \delta + \Delta_m(0)$ 时间间隔内是 δ-同步的

那么在 h 轮中，P 提交的区块将进入 finalized 状态。证明如下：

- 1. 在 t 时间，网络是半同步的，所有诚实 replica 在 $t + \delta$ 时间之前进入 h 轮（在这之前，所有诚实 replica 收到，使 h-1 轮结束的 Q 的 notarization 以及 h 轮的 random beacon）。
- 2. h 轮的 leader P 在 $t + \delta + \Delta_m(0)$ 之前提交一个区块 B。根据部分同步网络模型，这个区块的提案将在 $t + 2\delta + \Delta_m(0)$ 时间之前被传递至所有的 replica。
- 3. 由于 $\Delta_n(1) \geq \Delta_m(0) + 2\delta$，协议确保所有诚实 replica 支持区块 B，而非其他区块的 notarization。因此区块 B 将进入 notarized 和 finalized 状态。

5.12 其他的事项

5.12.1 生长延迟

在部分同步网络假设下，我们可以建立并证明 growth invariant 的一个定量版本。假设延迟函数的定义为：$\Delta_m(r) = 2\delta r$、$\Delta_n(r) = 2\delta r + \epsilon$，且 $\epsilon \leq \delta$。假设在时间 t，诚实的 replica 进入的最高轮次为 h。设 r* 为在 h 轮中，rank 最低的 replica P* 的 rank。最后，假设在时间间隔 $[t, t + (3r^* + 2)\delta]$ 内有 $\delta-$ 同步的网络。那么所有诚实的 replica 将在 $t + 3(r^* + 1)\delta$ 之前进入到 h+1 轮。（译者：这个性质说明该协议的吞吐率是较为稳定的）。

5.12.2 自调整的延迟函数

当 replica 几轮都没有发现 finalized 状态的区块后，它将开始增加自身的 notarization 延迟函数 Δ_n。Replica 无需就这些局部调整的 notarization 延迟函数达成一致。

同样的，尽管 replica 没有明确地调整 Δ_p 延迟函数，但是我们可以在本地调整两个延迟函数来对本地时钟的漂移进行数学建模。

因此，有多种由 replica 和轮次作为参数的延迟函数。在延迟函数中，一个非常重要的条件为 $\Delta_n(1) \geq \Delta_m(0) + 2\delta$。为了实现 liveness，协议中的延迟函数满足条件：$max\Delta_n(1) \geq min\Delta_m(0)$，其中 max 和 min 表示诚实 replica 中在给定的轮次中延迟函数的最大和最小值。因此，如果在多轮中都没有得到 finalized 状态的区块，所有的诚实 replica 将会增加 notarization 延迟，直到上述的条件得到满足并且得到 finalized 状态的区块。一些诚实 replica 的 notarization 延迟增加得比其他的 replica 更多对于 liveness 没有影响（但是可能影响 growth 延迟）。

5.12.3 公平性

共识协议中另一个非常重要的特性是公平性。相比于给出一个通用的定义，我们发现 liveness invariant 蕴含着一种很有用处的公平性。Liveness invariant 说明，在任意 leader 为诚实并且网络是同步的轮次中，由 leader 提交的区块将会进入 finalized 状态。在这种轮次中，诚实的 leader 将其收到的所有输入都会打包进它提交的区块的 payload 中。因此，任何被足够多的 replica 收到的输入在经过一定时间后会以很高的概率被包含进 finalized 状态的区块。（译者：此性质说明优秀的共识协议不太可能发生拒绝服务这种行为）。

第 6 章 消息路由层

如第 1.7 节所述，IC 中的基本计算单元称为 canister，它与进程的概念大致相同，因为它既包括程序又包括其状态。该 IC 提供了一个运行时环境，用于在 canister 中执行程序，以及与其他 canisters 和外部用户进行通信（通过消息传递）。

共识层（见第 5 节）将输入捆绑到 payload(payload) 中，这些 payload 被放入块 (blocks) 中，随着块的最终确定，相应的 payload 被传递到消息路由 (message routing) 层，然后由执行环境 (execution environment) 处理，执行环境更新复制状态机上 Canister 的状态并生成输出，这些输出由消息路由层处理。

有必要区分两种类型的输入：

- **外部消息：**这些是来自外部用户的消息;
- **跨子网消息：**这些是来自其他子网上的 canisters 的消息。

我们也可以区分两种类型的输出：

- **外部消息响应：**这些是对外部消息的响应（可能由外部用户检索）;
- **跨子网消息：**这些是发送到其他子网上的 canisters 的消息。

从共识接收 payload 后，该 payload 中的输入将被放入各种输入队列中。对于在子网上运行的每个 canister C，都有多个输入队列：一个队列用于将消息传入 C，然后对于每一个与 C 通信的其他 CanisterC'，都有一个用于从 C' 到 C 的跨子网消息。

如下所述，在每个轮次中，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，以及将输出放在各种输出队列中。对于每个在子网上运行的 canister C，都有多个输出队列：其中一个队列就是用于从 C 到 C' 的跨子网消息。消息路由层将获取这些输出队列中的消息，并将它们放入子网-子网流中，由跨网传输协议进行处理，其工作是将这些消息真正传输到其他子网。

除了这些输出队列之外，还有一个输入历史数据结构。当 canister 处理输入消息后，将在此数据结构中记录对该输入消息的响应。此时，提供输入消息的外部用户将能够检索相应的响应。（请注意，输入历史记录不会维护所有输入消息的完整历史记录。）

我们还应该提到，除了跨子网消息之外，还有一些子网内部消息，它们是从同一子网上的一个 canister 到另一个 canister 的消息。消息路由层将此类消息直接从输出队列移动到相应的输入队列。

图 3 说明了消息路由和执行层的基本功能。

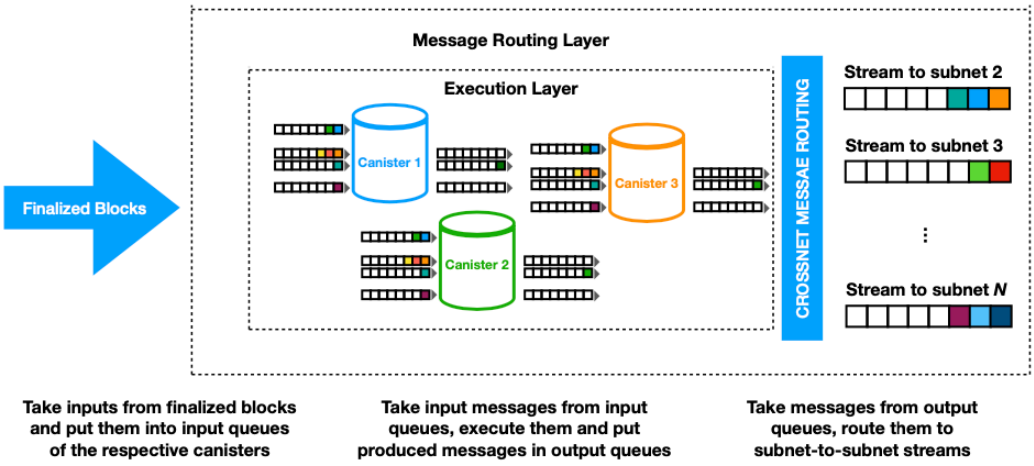

图 3: 消息路由和执行层

请注意，复制状态包括 canister 的状态，以及" 系统状态"，包括上述队列和流，以及输入历史数据结构。因此，消息路由层和执行层都涉及更新和维护子网的复制状态。必须以完全确定的方式更新所有这些状态，以便所有 replicas 保持完全相同的状态。

另请注意，共识层与消息路由和执行层分离，从某种意义上说，共识区块链中的任何分叉都在其 payload 传递给消息路由之前得到解决。并且实际上，共识不需要与消息路由和共识保持同步，并被允许提前运行。

6.1 每轮经认证的状态

在每一轮中，子网的某些状态将经过认证。每轮认证状态使用链-密钥加密技术（参见第 1.6 节）进行认证。具体而言，使用第 3 节中提到的 $(n-f)out-of-n$ 阈值签名方案。更详细地说，是每个 replica 在给定轮次生成每轮认证状态后，它将生成相应阈值签名的份额，并将其广播到其子网中的所有其他 replicas。在收集 $n-f$ 个此类份额时，每个 replica 都可以构造生成的阈值签名，该签名用作该轮次的每轮认证状态的证书。请注意，在签名之前，每轮认证的状态被散列为 Merkle 树 [Mer87]。给定轮次中的每轮认证状态包含

- 1. 最近添加到子网-子网流的跨子网消息;
- 2. 其他元数据，包括输入历史数据结构;
- 3. 上一轮中每轮认证状态的默克尔树根哈希。

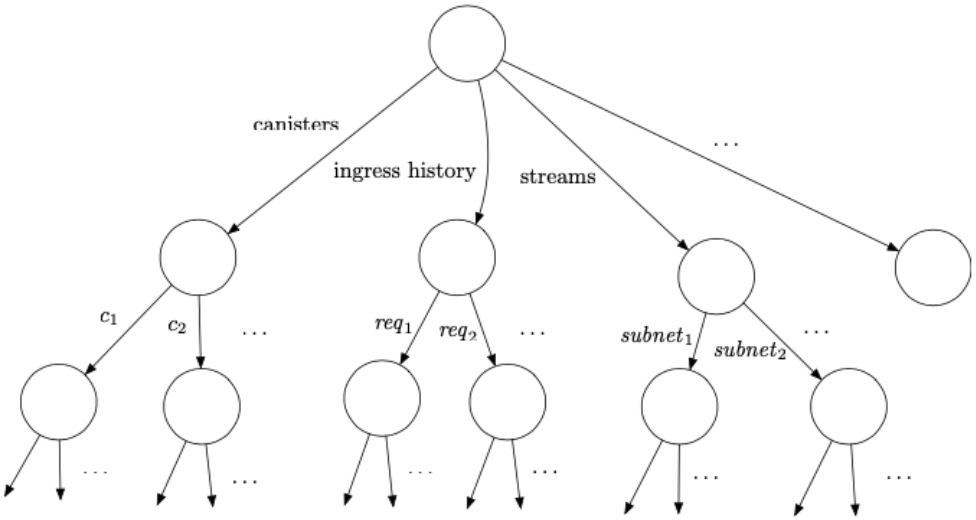

图 4: 每轮认证状态树状图

请注意，每轮认证状态不包括子网的整个复制状态，因为这通常非常大，并且在每轮中认证所有这些状态是不切实际的[1]。

图 4 说明了如何将每轮认证状态组织到树中。树的第一个分支存储有关每个 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记录数据。第三个分支存储有关子网到子网流的信息，包括最近为每个流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，此处不讨论。然后，可以将此树结构散列到 Merkle 树中，该树的大小和形状与此树基本相同。

每轮认证状态在 IC 中以多种方式使用：

- **输出验证**。跨子网消息和对输入消息的响应都会使用每轮经认证的状态进行验证。使用 Merkle 树结构，一个单个输出（跨子网消息或输入消息响应）可以向任何一方进行验证——通过在 Merkle 树的根上提供阈值签名，以及提供 Merkle 中从根到叶子的路径上表示该输出的哈希值（和相邻的哈希值）。因此，验证单个输出所需的哈希值与 Merkle 树的深度成正比，即使 Merkle 树的大小非常大，深度通常也很小。因此，单个阈值签名可用于有效地对许多单独的输出进行验证。
- **预防和检测非确定性**。共识保证每个 replica 以相同的顺序处理输入。由于每个 replica 都确定性地处理这些输入，因此每个 replica 都应获得相同的状态。但是，IC 在设计上具有额外的鲁棒性，以防止和检测任何有可能出现的（意

[1]参见第 8.2 节

外的）非确定性计算。每轮经认证状态是用于执行此操作的机制之一。由于我们使用 $(n-f)-out-of-n$ 阈值签名进行认证，当 $f < n/3$ 时，只能有一个状态序列是经过认证的。

若要了解状态链接为何很重要，请考虑以下示例。假设我们有 4 个 replicas，P_1, P_2, P_3, P_4，一个是故障的，比如 P_4。每个 replicaP_1, P_2, P_3 以相同的状态开始。

- -在第 1 轮中，由于非确定性计算，P_1, P_2 计算消息 m_1 发送到子网 A，而 P_2 计算消息 m_1' 发送到子网 A。
- -在第 2 轮，P_1, P_3 计算要发送到子网 B 的消息 m_2，而 P_2 计算发送到子网 B 的消息 m_2'。
- -在第 3 轮，P_2, P_3 计算要发送到子网 C 的消息 m_3，而 P_2 计算要发送到子网 C 的消息 m_3'。

下表对此进行了说明:

$$\begin{array}{llll} P_1 & m_1->A & m_2->B & m_3'->C \\ p_2 & m_1->A & m_2'->B & m_3->C \\ p_3 & m_1'->A & m_2->B & m_3->C \end{array}$$

我们假设 replicas P_1,P_2 和 P_3 各自单独执行有效的计算序列，但由于非确定性，这些序列并不相同。(尽管不应该有任何非决定论，但在这个例子中，我们假设存在。)

现在假设我们没有链接这些状态。因为 P_4 有错误，可以签署任何内容，所以他可以在第 1 轮状态上创建一个 $3-out-of-4$ 的签名（签名生效的阈值为 3，总共参与的数量为 4），上面写着"$m_1->A$"，以及类似地在第 2 轮状态上显示"$m_2->B$"，并在第 3 轮状态上显示"$m_3->C$"，即使相应的序列可能与任何有效的计算序列不兼容。更糟糕的是，这种无效的计算序列可能导致其他子网络上的状态不一致。

$$m_1->A \quad m_2->B \quad m_3->C$$

通过链接，我们确保即使存在一些非确定性，任何经认证的状态序列都对应于由诚实 replicas 实际执行的某个有效计算序列。

- 协调与共识。每轮认证状态也用于协调执行层和共识层。共有两种不同的协调方式:
  - 共识限流。每个 replica 将跟踪最新一轮。在这一轮中，有一个认证状态-称之为认证高度；还有一个公证区块-称之为公证高度。如果公证高

度明显大于认证高度，那说明执行滞后于共识，此时需要对共识进行限流。这种滞后可能是不确定性的计算，也可能只是层之间的性能不匹配引起的。共识是通过第 5.9 节中讨论的延迟函数来调节的——具体来说，随着认证高度和公证高度之间的差距增大，每个 replica 将增加“调控器”值 (这用到了第 5.12.2 节提到的“局部调整的延迟函数”的概念)。

– 特定状态 payload 验证。正如第 5.7 节所讨论的，payload 中的输入必须通过某些有效性检查。事实上，这些有效性检查一定程度上可能取决于状态。我们前文所省略的一个细节是，每个区块包含一个整数，以及有效性检查应该根据该整数的认证状态进行。执行验证的 replicas 将等待该整数的状态被验证，然后使用其验证状态来完成验证。这确保了即使在非确定性计算中，所有 replicas 都执行相同的有效性测试 (否则，共识可能无法达成)。

6.2 查询请求 vs 更新请求

正如上文所描述，一个外部消息必须通过共识，以便它们被一个子网上的所有 replicas 以相同的顺序处理。然而，对于那些处理不修改子网复制状态的外部消息，IC 推出了一个重要的优化。这些消息称为查询请求——与其他外部消息相反，那些消息称为更新请求。查询请求可以执行读取和（可能）更新 canister 状态的计算，但对 canister 状态的任何更新都不会交付到复制状态机。因此，查询请求可以由单个 replicas 直接处理，而无需经过共识，这大大减少了从查询请求到获得响应的时间消耗。

请注意，对查询请求的响应不会记录在进入历史数据结构中。因此，我们不能直接使用每轮认证状态机制来验证查询请求的响应。但是，我们提供了一种单独的机制来验证这些响应: 认证变量。作为每轮认证状态的一部分，子网上的每个 Canister 被分配少量字节，这是该 Canister 的认证变量，它的值可以通过更新请求进行更新，并且每一轮中可以使用认证状态机制进行认证。此外，canister 可以使用其认证的变量来存储 Merkle 树的根。通过这种方式，可以验证对 canister 的查询请求的响应，只要该响应是 Merkle 树中的一片叶，根位于该 canister 的已认证变量处。（译者注：Merkle 可信树是一棵完全二叉树，应用 Merkle 可信树需要计算并输出 Merkle 可信树的每个叶子节点的认证路径上的节点的哈希函数值，一个叶子节点的认证路径是指从该叶子节点到根节点的路径上经过的节点的兄弟节点。Merkle 可信树是为了解决多重一次签名中的认证问题而产生的，Merkle 可信树结构具有一次签名大量认证的优点，在认证方面具有显著的优势。）

6.3 外部用户认证

外部消息和跨子网消息之间的主要区别之一是用于验证消息的机制。我们已经在上文看到 (见 6.1 节) 如何使用门限签名来验证跨子网消息。NNS 注册表 (参见第 1.5 节) 持有用于验证跨子网消息的门限签名的验证公钥。

IC 中没有外部用户的注册中心。相反地，外部用户使用用户标识将自己标识到一个 canister 中。该标识是一个签名验证公钥的哈希值。用户持有一个相应的签名私钥，用于对进入信息进行签名。此类签名和相应的公钥将随进入信息一起发送。IC 自动验证签名，并将用户标识符传递给相应的 canister。然后，根据用户标识和进入信息中指定的操作的其他参数，canister 可以授权所请求的操作。

初次用户在与 IC 的第一次交互过程中生成密钥对，并从公钥获得用户标识符。老用户使用用户代理处存储的密钥进行验证。用户可以使用签名授权将多个密钥对与单个用户身份关联起来。这是非常有用且便利的方法，其允许单个用户使用相同的用户身份从多个设备访问 IC。

第 7 章 执行层

执行环境每次处理一个输入。该输入来自其中一个输入队列，并定向到一个 canister。基于此输入和 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进入历史 (也可能对较早的进入消息进行回应)。

在给定的一轮中，执行环境将处理数个输入。调度程序决定在给定的一轮中执行哪些输入，以及执行顺序。现在暂不讨论调度程序的所有细节，我们要着重强调了一些目标:

- 它必须是确定性的，也就是说，只依赖于给定的数据;
- 它应该在 canisters 之间公平地分配工作负载 (但吞吐量优化应优先于延迟优化)。
- 每一轮所做的工作总量是以 cycles 来衡量 (见第 1.8 节)，且应该接近某些预先设定的数量。

执行环境与消息路由器必须共同处理的另一个任务是: 一个子网上的 canister 产生跨子网消息的速度比另一个子网上的 canister 消耗消息的速度快。因此，启用了一种对生产 canister 进行限流的自我调节机制。

执行环境还要处理许多其他的资源管理和簿记任务。然而，所有任务都必须确切地得到处理。

7.1 Random Tape

每个子网都可以访问一个分布式伪随机发生器 (distributed pseudorandom generator，PRG)。正如在第 3 节中提到的，伪随机数字来自一个种子（真随机数），它本身是一个 (f +1)-out- n BLS 签名，称为 Random Tape。每一轮共识协议都有一个独特的 Random Tape。虽然这种 BLS 签名类似于共识中使用的随机信标 (见 5.5 节)，但机制有所不同。

在共识协议中，当一个高度为 h 的区块完成，每个正确 replicas 将释放高度 h + 1 的 Random Tape 份额。这有两层含义:

- 1. 在一个高度为 h 的区块被任何正确 replicas 完成之前，高度为 h + 1 的 Random Tape 肯定是不可预测的。
- 2. 在高度 h + 1 的区块被正确 replicas 完成之前，通常该 replicas 将拥有它需要用来构建高度 h + 1Random Tape 的所有份额。

为了获得伪随机数字，子网必须发出伪随机数字请求。这种请求将在某一轮

（例如 h）中作为执行层的“系统调用”发出。当高度为 h+1 的 Random Tape 可用时，系统随后将响应该请求。就上述性质 (1) 来看，可以保证的是：请求的伪随机数字在请求时是不可预测的。就上述性质 (2) 来看，请求的随机数字通常在下一个区块完成时可用。事实上, 在当前的实施中, 当一块高度 h 最终完成时, 共识层 (见第 5 节) 将把高度为 h 的区块的 payload 和高度为 h + 1 的 Random Tape 同时交给消息路由层进行处理。

第 8 章 Chain-key cryptographyII: chain-evolution technology

如第 1.6.2 节所述，链密钥加密技术包括一系列技术，以实现对基于区块链的复制状态机进行长期稳健安全地维护，这些技术共同构成了所谓的链演进技术（chain-evolution technology）。

链演进技术实现了一些周期性的基本行为，这些行为包括：垃圾回收，快速同步，子网成员更换，秘密重新分享的预执行以及协议的更新。chain-evolution technology 中有两个基本组成部分：summary blocks 以及 catch-up package（CUPs）。

8.1 Summary blocks

每个 epoch 中的第一个区块为 summary block。Summary block 包含用于管理各种门限签名（见第 3 章）的分享的特殊数据。其中有两种门限策略：

- $(f+1)-out-of-n$ 策略，每个 epoch 都会为其产生一个新的签名密钥。
- $(n-f)-out-of-n$ 策略，每个 epoch 都会对签名密钥重新分享。

第一种门限策略用于生成 random beacon 以及 random tape，第二种门限策略用于验证子网的状态。

DKG 协议（见第 3.5 节）中对于每个签名密钥需要一个 dealing 的集合，并且每个 replica 无法从这个集合中非交互地获得它关于签名密钥的份额。

NNS 中维护了一个注册表以及其他的一些信息，他们共同确定了子网的身份信息（见 1.5 节）。注册表（子网成员）可以随时间变化。因此子网必须对注册表的版本达成一致，不同的版本对应不同时间为了不同的目的注册表内容。版本信息同样储存于 summary block 中。

epoch i 的 summary block 包含下列数据段：

- currentyRegistryVersion：注册表版本，它确定了 epoch i 中的共识委员会，共识层的所有行为（区块生成，公证、终止）都由该委员会执行。
- nextRegistryVersion：共识的每一轮中，区块生成者在他的提案中包含它知道的最新的注册表版本（该版本不能早于将要提交的区块的父区块）。该策略保证了 nextRegistryVersion 的值及时更新。epoch i 中，currentyRegistryVersion 的值是 epoch $i-1$ 中，nextRegistryVersion 值的集合。
- currentDealingSets.：确定门限签名密钥的 dealing 集合，该值在 epoch i 中对

消息进行签名时使用。epoch i 中签名委员会（例如，持有相关门限签名密钥分享的 replica）是 epoch $i-1$ 中的共识委员会。

- nextDealingSets：该集合在 epoch $i-1$ 中被收集存储[1]。epoch i 中，currentDealingSets 的值是 epoch $i-1$ 中 nextDealingSets 的值（其包含 epcho $i-2$ 中被收集的 dealing）。
- collectDealingParams：该值描述了 epoch i 定义了被收集的 dealing 集合的参数。epoch i 中，区块生成者将符合这些参数的 dealings 包含进提交的区块中。
  接收委员会基于 epoch i 中 summary block 中 nextRegistryVersion 的值构建。
  对于低门限策略，dealing 委员会为 epoch i 中的共识委员会。
  对于高门限策略，将被重新共享的“分享物”基于 epoch i 里 nextDealingSets 的值。因此，dealing 委员会是 epoch $i-1$ 中的接收委员会，也是 epoch i 的共识委员会。
  可以看到，epoch i 的门限签名委员会为 epcho $i-2d$ 接收委员会，同时是 epoch $i-1$ 的共识委员会。

epoch i 的共识依赖 epoch i 的 currentRegistryVersion 以及 currentDealingSets，即共识委员会的建立基于 currentRegistryVersion，共识协议中使用的 random beacon 基于 currentDealingSets。此外，和其他的区块一样，在 epoch i 的开始可能有不止一个经过公证的 summary block。如果假设在 epoch i 之前 epoch $i-1$ 的 summaryblock 已经是 finalized（终止）状态，那么这种看似循环的问题便得到了解决。因为 epoch i 中 summary block 的相关信息可以直接从 epoch $i-1$ 拷贝。这是一种隐含的同步假设，由于 5.12.2 中提及保证 liveness 的共识降速（consensus throttling），并且每个 epoch 都相当长。这种一个 summary block 的循环论证在现实中不大会发生：在 epoch $i-1$ 中如果区块长时间没有终止，那么 notarization 延迟函数将会急剧增大，因此 finalization 的部分同步假设将被满足，因此 epoch $i-1$ 的 summary block 也将成为 finalized 状态。[2]

8.2 CUPs

在描述 CUP 之前，我们先说明 random beacon 的一些细节：每一轮 random beacon 的生成依赖于上一轮的 random beacon。这不是必要条件，但是会影响 CUP

[1]我们省略的一个细节是，如果我们未能在 epoch $i-1$ 中收集所有必需的事务，那么作为回退，epoch i 中 nextDealingSet 的值将有效地设置为 epoch$i$ 中 currentDealingSet 的值。如果发生这种情况，那么协议将酌情利用过去更多的交易委员会和阈值签署委员会。

[2]另请注意，在 epoch$i$ 中收集的交易依赖于 epoch$i$ 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，并且在 epoch i 的摘要块最终确定之前无法被验证。

的设计。

CUP 是一个特殊的消息（不在链上），它几乎包含了一个 replica 在给定 epoch 中开始工作的所有信息，并且该 replica 无需知道之前 epochs 的信息。CUP 包含如下字段：

- 全部状态的 Merkle 树的根（非 6.1 节中的每轮验证状态，每轮验证状态为部分状态）
- 该 epoch 的 summary block
- 该 epoch 第一轮的 random beacon
- 上述字段在该子网密钥下的 $(n-f)-out-of-n$ 门限签名。

为了生成给定 epoch 的 CUP，replica 必须等到该 epoch 的 summary block 成为 finalized 状态并且相关的状态被验证。完成的状态必须被构建成一棵 Merkle 树。尽管使用了很多技术来加速这个过程，但是该行为依旧十分昂贵。这也是为什么每个 epoch 只做一次的原因。由于 CUP 只包含了 Merkle 树的根，replica 使用一种**状态同步 (state sync)** 的子协议向其他状态存储节点拉取状态数据，这个技术流程复杂的过程同样十分昂贵。针对 CUP 使用上文中提及的第二种高门限签名策略，可以确保任意 epoch 中只有一个合法的 CUP，并且存在一些存储状态的节点，以便 replica 拉取状态数据。此外，由于门限签名方案的公钥随时间保持不变，因此可以在不知道子网当前参与者的情况下验证 CUP。

8.3 chain-evolution technology 的实现

垃圾回收：凭借给定 epoch 的 CUP 中包含的信息，replica 可以无风险地清理所有该 epoch 之前执行过的输入，以及该 epoch 前对这些输入进行排序所需的所有共识层消息。

快速同步：如果子网中的 repilca 远远落后于其他的 replica（由于宕机或者长时间网络断开），或者一个新的 replica 被加入到子网中，该 replica 可以快速的同步到最新 epoch 的开始，而无需运行共识协议以及执行该时间点之前的输入。该 replica 通过获得一个最新的 CUP 来完成上述行为。使用 CUP 中的 summary block 以及 random beacon 以及其他 replica 获得的共识消息（此时该时间节点之后的消息还没被抛弃），该 replica 可以从该时间节点（相关 epoch 的开始）执行共识协议。该 replica 同样适用状态同步子协议获得相关 epoch 开始时的状态，并且开始从该时间节点执行共识的产生的输入，直到追上其他的 replica。

图 5 描述了快速同步的过程。假设需要同步的 replica 在一个 epoch 的开始有一个 CUP，该 CUP 在高度 101。CUP 包含在高度 101 状态的 Merkle 树的根、高

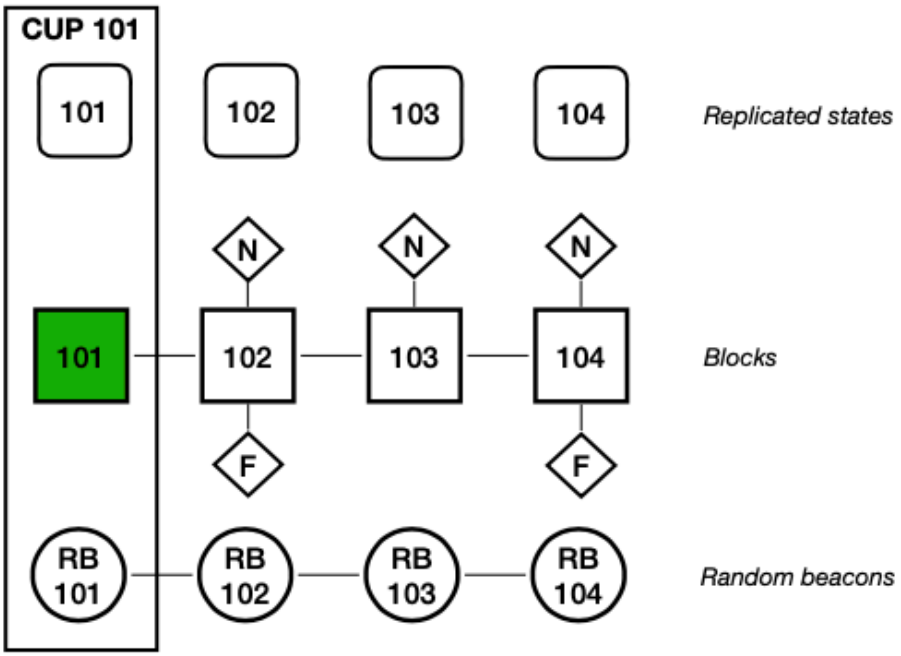

图 5: 快速推进

度 101 的 summary block（绿色区块）以及高度 101 的 random beacon。replica 使用状态同步子协议从其他的节点获得高度 101 的完全复制状态，并使用 Merkle 树的根验证状态的合法性。在获得完整状态之后，replica 可以参与后续协议，即从高度 102，103 等处的其他区块（以及与共识关联的其他消息）中获取并更新其复制状态的 replicas。当收到 finalized 状态的区块，replilca 会立即执行。

子网成员变更：我们已经讨论过，给定 epoch 如何使用 summary block 确定注册表的版本、确定子网的成员、以及完成不同任务的成员委员会。需要注意的是，即使一个 replica 被从子网中移除，它在需要的情况下依旧要参与额外的一个 epoch 完成它所属委员会的任务。

秘密重分享准备：我们已经讨论过了如何使用 summary block 生成以及重新分享签名密钥。在需要的情况下，summary block 可以从 CUP 中获得。

协议更新：同样可以使用 CUPs 来更新协议的实现。NNS（见 1.5 节）初始化协议更新。基本的思路为：

- 当需要下载一个新版本的协议时，在一个 epoch 开始的 summary block 将会表明需要更新协议。
- 运行旧版本协议的 replica 将会继续执行共识协议直到 summary block 变为

finalized 状态并且创建一个对应的 CUP。然而，这些 replica 只会创建空区块并且不会向消息路由和执行层传递任何的 payload（负载）。Replica 下载新版本的协议，

- 将安装该协议的新版本，并且运行该协议新版本的 replicas 将恢复运行上述 CUP 中的完整协议。

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 Sympo- sium 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 Interna- tional 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. Damg ard, 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 Bar- bara, 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 Sci- ence, 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. My- ers, 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 Con- firmation. 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 Cryp- tographic 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.