Dfinity白皮书

——The Internet Computer for Geeks(v1.2)

Dfinity团队

2022年2月14日

译者：方俊伟，袁博南

Github Repo

摘要

智能合约是一种新的软件形式，它将彻底改变软件的编写方式、IT 系统的维护方式，以及应用程序及整体业务的构建方式。智能合约是在去中心化区块链上运行的可组合且自治的软件部件，这使得它们无法被篡改和停止。在本文中，我们将介绍互联网计算机(以下简称IC)，作为一种全新的区块链设计，IC摆脱了智能合约在传统区块链上的速度、存储成本和计算能力方面的限制，使智能合约的全部潜力得以释放。IC首次允许智能合约实现完全的去中心化，使得前端到后端都能被托管在区块链上。IC 由一组加密协议组成，这些协议将独立运行的节点相互连接以组成一个区块链的集合。这些区块链托管并执行"储罐(Canister)"，即 IC上的智能合约。储罐可以存储数据，对数据进行通用计算，并提供完整的技术栈，从而直接为终端用户提供网络服务。计算和存储开销采用“反向GAS模型”，这里需要储罐开发人员将IC 的原生代币ICP兑换成cycles进行预付。ICP代币同时也用于治理：IC由去中心化自治组织（以下简称DAO）进行管理，DAO决定变更IC的网络拓扑结构和升级IC协议。

Dfinity白皮书

摘要

1 引言

1.1 释放智能合约

1.2 高阶视角下的IC

1.3 故障模型

1.4 通信模型

1.5 权限模型

1.6 链钥密码学（Chain-key cryptography）

1.6.1 阈值签名

1.6.2 链演进技术（Chain-evolution technology）

1.7 执行模型

1.8 功能代币

1.9 边界节点

1.10 NNS的更多细节

1.10.1 NNS的决策制定

1.11 当前工作

2 架构概述

2.1 P2P层

2.2 共识层

2.3 消息路由

2.3.1 每轮认证状态(Per-round certified state)

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

2.3.3 外部用户验证(External user authentication)

2.4 执行层

2.5 组合阐述

3 链钥密码学I：阈值签名

3.1 BLS阈值签名

3.2 分布式密钥分发

3.3 假设

3.4 PVSS方案

3.5 基础DKG协议

3.6 再共享协议

4 P2P层

5 共识层
5.1 假设
5.2 协议概述
5.3 额外特性
5.4 公钥
5.5 随机信标
5.6 区块生成
5.7 公证
5.8 最终确认
5.9 延迟函数
5.10 例证
5.11 组合详述
5.11.1 随机信标详述
5.11.2 区块生成详述
5.11.3 公证详述
5.11.4 生长不变性（growth invariant）证明
5.11.5 安全不变性（safety invariant）证明
5.11.6 活性不变性（liveness invariant）证明
5.12 其他问题
5.12.1 生长延迟（Growth latency）
5.12.2 本地化调整的延迟函数（Locally adjusted delay functions）
5.12.3 公正性
6 消息路由层
6.1 每轮认证状态（Per-round certified state）
6.2 查询调用和更新调用
6.3 外部用户验证
7 执行层
7.1 随机磁带
8 链钥密码学II：链演进技术
8.1 摘要块
8.2 CUPs
8.3 链演进技术实现
参考文献

1 引言

1.1 释放智能合约

因其独具的特性，智能合约是 Web3 的关键推动要素，Web3是一种获取网络服务的新途径，其应用程序完全由用户控制并在去中心化的区块链之上运行。这种去中心化应用程序（以下简称dapps/dapp）通常是代币化的，这意味着项目方本身的代币被分发给用户作为参与dapps的奖励。参与dapps的方式有很多种，包括审核和提供内容来治理dapp，以及创建和维护dapp。通常来说，用户可以在交易所购买代币。实际上，常见的是项目方是通过出售代币为dapp 开发进行融资。最后代币还可以被用于支付dapp提供的服务或内容。在目前的区块链平台上，其中包括最主流的这些（诸如以太坊），智能合约的运行都会遇到诸多限制，包括较高的交易手续费和存储成本、较慢的运算速度和无法向用户提供前端服务。因此，很多受欢迎的区块链并不完全去中心化，而是中心化和去中心化混合的状态。大多数方案是将程序托管在传统的云平台中，其中链上的智能合约的调用仅为其功能中极小的一部分。不幸的是，这使得这些应用变得不再去中心化，令它们暴露在许多传统云平台托管的程序的缺点之中，例如受制于云平台的服务商和容易发生单点故障。

IC是一个运行智能合约的新平台。在这里，我们使用“智能合约”术语中广义上的定义：一种*通用的*，*不可变的*，*防篡改*的计算机程序，其在*分布式公共网络*中*自治地*执行。

- *通用的*，是指智能合约这类程序是图灵完备的（即任何可计算的运算都可以用智能合约完成）。
- *防篡改的*，是指程序的指令被可信地执行，并且计算的中间结果和最终结果被准确地存储和/或传输。
- *自治地*，是指智能合约被网络自动执行，不需要任何人采取任何行动。
- *分布式公共网络*，是指计算机网络可被公开地访问的、地理上分布式的，并且不受少数人或组织控制。

此外，智能合约

- 是*可组合的*，意味着他们可以彼此交互。
- 支持*代币化*，意味着他们可以使用和交易代币。

对比现有的智能合约平台，IC在设计上：

- 更具*成本优势*，特别是应用程序计算和存储数据的成本，仅为先前平台成本的很小一部分；
- 为智能合约的交易处理提供*更高的吞吐量*和*更低的延迟*；
- 更具扩展性，特别是IC可以原生地处理无限量的智能合约数据和计算，因为它可以通过向网络中添加节点来提升容量。

智能合约可能具备的另一个特性是不可变性，意味着其一旦部署，智能合约的代码就不能由一方单独更改。虽然此特性是某些应用程序所必备的，但是并不是所有的应用程序都需要具备该特性，在智能合约有漏洞需要修复时，不可变性也是一个问题。IC允许智能合约有

除了提供智能合约平台之外，IC设计上是一个完整的技术堆栈，构建的系统和服务可以完全在IC上运行。特别的是，IC上的智能合约可以处理终端用户的HTTP请求，因此智能合约可以直接提供交互式的网络体验。这意味着，系统和服务的创建不需要依赖于公司的云托管服务或者私人服务器，从而以真正的端到端的方式提供智能合约的所有优势。

实现Web3的愿景。对于终端用户而言，访问基于IC的服务在很大程度上是透明的。他们的个人数据比在访问公有云或私有云的应用时更安全，但是与应用程序的交互体验是一样的。

然而，对于创建和管理基于IC的服务的人来说，IC消除了许多在开发和部署当前的应用程序和微服务时的成本、风险和复杂性。例如，在当前垄断互联网的科技巨头们所推动的整合下，IC平台提供了另一种选择。此外，IC安全协议可以确保消息的可靠传递、透明可追溯，以及不需要依赖于防火墙、备份设施、负载均衡服务器和故障编排就可以实现的网络弹性。IC允许智能合约有一系列的可变性策略，从完全不可变到单方面可升级，以及介于两者之间的其他选项。

构建IC就是要互联网回归其开放，创新和创造性的本源——换而言之即*实现Web3的愿景*。针对一些特别的示例，IC做了下述的事：

- 提供互操作性，共享函数，持久化APIs和无主应用程序，上述的所有特点减少了平台的风险，并鼓励创新和协作。
- 数据自动持久化于内存中，无需数据库服务器和存储管理，提升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堆栈，提升了运营效率。

1.2 高阶视角下的IC

大致上，IC是一个交互**复制状态机(replicated state machines)**的网络。复制状态机在分布式系统[Sch90]中是一个相当标准的概念，但是我们在这里仍然简单介绍下，从*状态机*的概念开始。

状态机是一种特定的计算模型。此类机器维护着一个**状态**，即对应普通计算机中的主存储或是其他形式的数据存储。此类机器按离散的**轮次**进行执行：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**，获得一个**输出**和一个**新的状态**。这一*新状态*将变成下一轮次的*当前状态*。

IC中的状态转换函数是一个**通用函数**，意味着一些存储在状态中的输入和数据可能是任意的**程序**，这些程序会作用于其他的输入和数据。因此，这样的一个状态机代表了一个通用（即图灵完备）的计算模型。

为了实现**容错性**，状态机可以被**复制**。**复制状态机**包含由一个**节点副本(replicas)**组成的**子网**，其中每一个节点副本运行相同状态机的副本。即使某些节点副本发生故障，子网也应当继续——并且正常运转。

子网中的每个节点副本都必须按照相同的顺序处理相同的输入。为此，子网中的节点副本必须运行**共识协议**[Fis83]，来确保子网中的所有节点副本按照相同的顺序处理输入。因此，每一个节点副本的内部状态将按照相同的方式随时间演变，并且每个节点副本会生成完全相同序列的输出。需要注意的是，IC上复制状态机的输入可以是由外部用户生成的输入，也可以是另一台复制状态机生成的输出。类似地，复制状态机的输出可以作为输出导向外部用户，也可以作为输入导向另一台复制状态机。

1.3 故障模型

在计算机科学的分布式系统领域中，通常会考虑两种类型的节点副本故障：**宕机故障**和**拜占庭故障**。**宕机故障**发生在节点副本突然停机并且无法恢复时。**拜占庭故障**是节点副本可能以任意的方式偏离规定的协议。而且，在拜占庭故障下，一个或多个节点副本可能直接处于恶意对手方的操控之中，其可以操纵这些节点副本的行为。在这两种故障类型中，拜占庭故障具有更大的潜在破坏性。

共识协议和实现复制状态机的协议通常会假设**多少**节点副本可能发生故障以及发生**何种程度**的故障（宕机或拜占庭）。IC中假设一个给定的子网若有n个节点副本，那么发生故障的节点副本少于$n/3$，并且这些故障可能是拜占庭式。（需要注意的是，IC中的不同子网可能有不同的大小。）

1.4 通信模型

共识协议和执行复制状态机的协议通常也会对**通信模型**作出假设，描述了敌对方延迟节点副本间消息传递的能力。在两个对立端下，我们有如下的模型：

- 在**同步模型**中，存在已知的有限时间限制δ，因此对于发送的任意消息，它会在小于δ的时间内递达。
- 在**异步模型**中，对于发送的任意消息，敌对方可以延迟其传递任意有限时间，因此对于传递消息没有时间限制。

由于IC子网中的节点副本通常分布在全球，同步通信模型非常不切实际。事实上，攻击者可以延迟诚实节点副本或是延迟诚实节点副本间的通信，来破坏协议的正确行为。这种攻击通常比控制和破坏诚实节点副本更容易实施。

在全球分布的子网的设定下，最可行和健壮的模型是异步模型。不幸的是，目前没有已知的共识模型在异步模型下是真正可行的（最近的异步共识协议，如[MXC&+^16]，可以达到可观的吞吐量，但是延迟不太好）。所以同其他大多数不依赖于同步通信的实用拜占庭容错系统（例如PBFT[CL99], [BKM18], [AMN+20]）一样，IC选择了一种折衷的方案：**部分同步**通信模型[DLS88]。这样的部分同步模型可以有多种构建方式。IC使用的部分同步模型假设，大致上讲，每个子网中节点副本的通信在很短的时间间隔内是周期

性同步的；此外，同步时间限制δ不需要被提前知晓。建立这种部分同步假设仅仅是为了确保共识协议的进行（所谓的活性）。确保共识的正确性（所谓的安全性）并不需要这种部分同步假设，同样在IC协议栈的其他任何地方也不需要。

在部分同步和拜占庭故障的假设下，众所周知的是，我们对于故障节点数量$f < n/3$的限制是最优解。

1.5 权限模型

最早的共识协议（例如PBFT[CL99]））是**有许可的**，也就是说构成复制状态机的节点副本是受一个中央组织管理，其决定了哪些实体可以提供节点副本，网络的拓扑结构，并且可能还实现了某种程度中心化的公钥基础设施。许可共识机制通常效率是最高的，尽管它们避免了单点故障，但是中心化管理对于特定的应用程序是不可取的，并且这违背了蓬勃发展的Web3时代的精神。

最近，我们看到了**无许可的**的共识协议的兴起，例如Bitcoin[Nak08]，Ethereum[But13]和Algorand[GHM+17]。这些协议基于**区块链**，采用**工作量证明（以下简称PoW）**（例如Bitcoin，Ethereum2.0之前）或是**权益证明机制（以下简称PoS）**（例如Algorand，Ethereum2.0）。尽管这些协议是完全去中心化的，但是他们比有许可的协议效率更低。我们也需要指出，正如[PSS17]所观察到的，基于PoW机制的共识协议例如Bitcoin，在异步通信网络下并不能保证正确性（即安全性）。

IC的权限模型是一个**混合模型**，在拥有有许可协议的效率的同时提供去中心化PoS协议的许多好处。这个混合模型叫做**DAO控制网络（DAO-controlled network）**，（大致上讲）按如下机制工作：每个子网运行一个有许可的共识协议，但是由一个**DAO**决定哪些实体可以提供节点副本，配置网络的拓扑结构，提供公钥基础设施，并且控制节点副本部署的协议版本。IC的DAO被称为**网络神经系统（以下简称NNS）**，基于Pos，因此所有NNS的决策都由社区成员决定，社区成员的投票权由其在NNS中质押的IC原生治理代币（关于该代币的细节详见章节1.8）数量决定。通过这个基于PoS创建的治理系统，可以创建新的子网，可以在现有子网中增加或者移除节点副本，可以部署软件更新并且可以对IC进行其他调整。NNS本身是一个复制状态机，（和其他状态机一样）运行在特定的子网上，其成员资格由同一套基于PoS的治理系统所决定。NNS维护一个称为**注册表**的数据库，跟踪IC的拓扑结构：哪些节点副本属于哪个子网，节点副本的公钥等等。（关于NNS的更多细节详见章节1.10。）

因此，人们看到IC的DAO控制网络既允许IC获得许可网络的很多实用优势（在更有效的共识方面），也保留了去中心化网络的许多优势（在DAO治理下）。

运行IC协议的节点副本托管在地理上分布式的、独立运行的数据中心之上。这也增强了IC的安全性和去中心化性。

1.6 链钥密码学（Chain-key cryptography）

IC的共识协议确实使用了区块链，但它也采用了公钥加密技术，特别是电子签名：NNS维护的注册表用于将公钥绑定至节点副本及子网形成一个整体。这实现了一个独一无二的强大技术集合，我们称之为**链钥密码学**，它有几个组成部分。

1.6.1 阈值签名

链钥密码学的第一个组成部分是**阈值签名**：阈值签名是一个成熟的加密技术，它允许子网拥有一个公共的验证签名密钥，对应的签名私钥分成**多份**分配给子网中的节点副本，而分配保证作恶节点无法伪造任何签名，而诚实节点拥有的私钥片段可以允许子网生成符合IC原则和协议的签名。

这些阈值签名的一个关键应用在于

一个子网的单独输出可以由另一个子网或是外部用户进行验证，验证可以简单地利用该子网（第一个子网）的公共验证签名密钥来验证电子签名实现。

需要注意的是，子网的公共验证签名密钥可以从NNS中获取——该公共验证签名密钥在子网的生命周期中保持不变（即使子网的成员在该生命周期中可能发生变化）。这与许多不可扩展的区块链协议形成鲜明的对比，其需要验证整个区块链来验证单个输出。

正如我们所看到的，这些阈值签名在IC中还有许多其他应用。一个应用于让子网中的每个节点副本可以访问无法预测的伪随机数位（衍生于此类阈值签名）。这是共识层使用的**随机信标**和执行层使用的**随机磁带**的基础。

为了安全地部署阈值签名，IC采用了创新性的**分布式密钥生成**（以下简称DKG）协议，来构建公共签名验证密钥，并为每个节点副本提供对应签名私钥的一个片段，用于我们的故障和通信模型。

1.6.2 链演进技术（Chain-evolution technology）

链钥密码学也包括一系列复杂的技术，用于随时间推移健壮和安全地维护基于区块链的复制状态机，其合起来我们称之为**链演进技术**。每个子网在包含多轮（通常大约是几百轮）的**时期**（Epoch）内运行。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：在每一周期的末尾，所有已经被处理的输入以及所有排序这些输入所需要的共识层消息，可以安全地从每个节点副本的内存中清除。这对防止对于节点副本的存储需求的无限增长至关重要。这也与许多不可扩展的区块链协议形成对比，它们必须存储从创世区块开始的整个区块链。

快速转发：如果一个子网中的节点副本大幅落后于其同步节点（因为其宕机或是网络断连很长时间），或是一个新的节点副本被添加入子网，他们可以通过*快速转发*至最新Epoch的起始点，不需要运行共识协议并处理该点之前的所有输入。这也与许多不可扩展的区块链协议形成对比，它们必须处理从创世区块开始的整个区块链。

子网成员变更：子网的成员（由NNS决定，详见章节1.5）可能会随着时间变化。这仅可能发生在**Epoch**的边缘，需要小心操作以确保一致且正确的行为。

主动秘密再共享：我们在上面的章节1.6.1中提到IC是如何使用链钥密码学——具体来说，阈值签名——来进行输出验证。它基于的就是**秘密共享**，通过将一个秘密（在这里就是签名私钥）拆分成片段分别存储在节点副本中，从而避免了任何单点故障。在每个时期开始时，这些片段都会被**主动再共享**。这实现了两个目标：

- 当一个子网的成员发生变动时，再共享可以确保任何新成员都有适当的秘密片段，而任何不再是成员的节点副本就不再会拥有秘密片段。
- 如果在任意一个时期，甚至每个时期都有少量的秘密片段被泄露给攻击者，这些片段也不会帮助到攻击者。

协议升级：如果IC协议本身需要升级，修复漏洞或是增加新功能，可以在时期开始时通过特殊协议自动完成。

1.7 执行模型

如前所述，IC的复制状态机可以执行任意的程序。IC中的基本计算单元叫做**储罐**，它和*进程*的概念大致相同，包含了*程序*和其*状态*（随时间变化）。

储罐的程序用**WebAssembly（以下简称Wasm）**进行编码，是一种基于堆栈的虚拟机的二机制指令格式。Wasm是一种开源标准[1]。尽管它最初设计是为了实现网页端的高性能应用，但它也非常适合用于通用计算

IC提供了一个运行时环境，用于储罐内执行Wasm程序，并与其他储罐和外部用户通信（通过消息传递）。虽然原则上可以用任何能编译成Wasm的语言去编写储罐程序，但我们设计了一个叫做**Motoko**的语言，它与IC的操作语义十分一致。Motoko是一种强类型，*基于actor*[2]的编程程序，内置支持*正交持久性*[3]和*异步消息传递*。正交持久性意味着储罐维护的内存会自动持久化（即不必写入文件）。Motoko具有许多生产力和安全特性，包括自动内存管理，泛型，类型推断，模式匹配，以及任意和固定精度的算术。

除了Motoko之外，IC还提供了一个消息接口定义语言和数据格式称为**Candid**，用于固定类型、高级语言及跨语言互操作性。这使得任何两个储罐，即使是用不同的高级语言编写，也可以轻松地相互通信。

为了提供全面支持任何给定编程语言的储罐开发，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编程语言的储罐开发。

1.8 功能代币

IC使用称为ICP的功能代币。该代币有如下功能：

NNS质押：如章节1.5所述，ICP可以用于在NNS中*质押*获得投票权，从而参与控制IC网络的DAO。在NNS中质押代币并参与NNS治理的用户还会收到*新铸造*的ICP代币作为*投票奖励*。奖励数量由NNS制定和执行的政策所决定。

兑换Cycles：ICP用于支付IC的使用费用。更具体来说，ICP可以兑换成**cycles**（即销毁），这些cycles可以用于支付创建储罐（详见章节1.7）和储罐所使用的资源（存储，CPU和带宽）费用。ICP兑换成cycle的比例由NNS决定。

支付节点提供者：ICP用于支付节点提供者——这些实体拥有和运营着计算节点，来托管构成IC的节点副本。NNS定期（当前每月一次）决定每个节点提供者应收到的*新铸造*代币并发放至其账户。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。特别是，他们提供了

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

为了可以从传统客户端无缝访问IC，边界节点提供了对应的功能，将用户的标准HTTPS请求转换成指向IC容器的输入消息，随后将该输入消息路由至储罐所在子网的特定节点副本。而且，边界节点提供了改善用户体验的额外服务：缓存，负载均衡，速率限制以及传统客户端验证来自IC响应的能力。

储罐通过ic0.app域名上的URL链接进行标识。初始条件下，传统客户端会寻找URL链接对应的DNS记录，获取边界节点的IP地址，随后发送一个初始的HTTPS请求至该地址。边界节点返回一个基于JavaScript的“服务工作机（service worker）”，以运行于传统客户端。在此之后，传统客户端和边界节点的所有交互都通过这个service worker完成。

Service worker的一项基本任务是利用链钥密码学（详见章节1.6）验证来自IC的响应。为此，NNS公共的验证密钥被硬编码在service worker内。

边界节点本身负责将请求路由至托管特定储罐子网的节点副本。边界节点执行路由所需要的信息从NNS中获取。边界节点保管着一个可以及时响应的节点副本列表并从中随机选择一个。

传统客户端与边界节点间，边界节点与节点副本之间的通信安全都由TLS[4]保证。

除了传统客户端外，还可以使用“IC原生”客户端与边界节点交互，其已经包含了service worker的逻辑，不需要向边界节点取回service worker程序。

和节点副本一样，边界节点的部署和配置由NNS控制。

1.10 NNS的更多细节

如章节1.5所述，NNS是一个控制IC的算法治理系统。它通过特殊的**系统子网**中的一组储罐实现。该子网和其他子网类似，但配置有所不同（例如，系统子网中的储罐不收取cycles费用）。

其中最重要的一些NNS储罐是

- **注册表储罐**，存储着IC的配置信息，即哪些节点副本属于哪个子网，子网和节点副本的公钥等等。
- **治理储罐**，管理着IC协议如何演进的决策制定和投票。
- **账本储罐**，记录用户的ICP代币账户和相互间的交易。

1.10.1 NNS的决策制定

任何人通过在所谓的**神经元（neurons）**中质押ICP代币参与NNS治理。神经元的持有者可以提议关于IC应该如何改变的**提案**并投票，例如子网的拓扑结构或者协议该如何改变。神经元的投票权利是基于PoS的。直观地说，质押更多ICP的神经元投票权利更大。但是，投票权也取决于神经元的其他特征，例如愿意质押代币更长时间的神经元持有人，被赋予了更大的投票权。

每一个提案有确定的投票期限。如果投票期结束时，参与投票的简单多数赞成该提案，并且赞成票数超过了给定的总投票权法定人数要求（现在是3%），则该提案被采纳。否则，该提案被否决。除此之外，任何时候只要绝对多数（超过总投票一半）赞成或反对该提案，该提案相应被采纳或否决。

如果提案被采纳，治理储罐会自动执行决策。例如，如果一个提案提议改变网络的拓扑结构并且被采纳，治理储罐将自动使用新配置来更新注册表储罐。

1.11 当前工作

IC的架构仍在不断演进和扩展。以下是一些即将部署的新功能：

DAO控制储罐。就像IC的整体配置是由NNS控制一样，任意的储罐也可以由其自身的DAO控制，称为**服务神经系统（以下简称SNS）**。控制储罐的DAO可以更新储罐逻辑，也可以下达专属权限命令让储罐执行。

ECDSA阈值签名。ECDSA签名[JVM01]被用于加密货币，如Bitcoin和Ethereum，以及许多其他应用程序。虽然阈值签名已经是IC中的重要组成部分，但并不是ECSDA阈值签名。而这项新特性将允许单个储罐控制ECDSA签名密钥，这些签名密钥安全地分布在托管该储罐子网的节点副本中。

Bitcoin和Ethereum集成。基于新的ECDSA阈值签名，这一特性将允许储罐与Bitcoin和Ethereum链交互，包括直接签名链上交易。

HTTP集成。此功能将允许储罐读取任意网页（IC外部的）。

2 架构概述

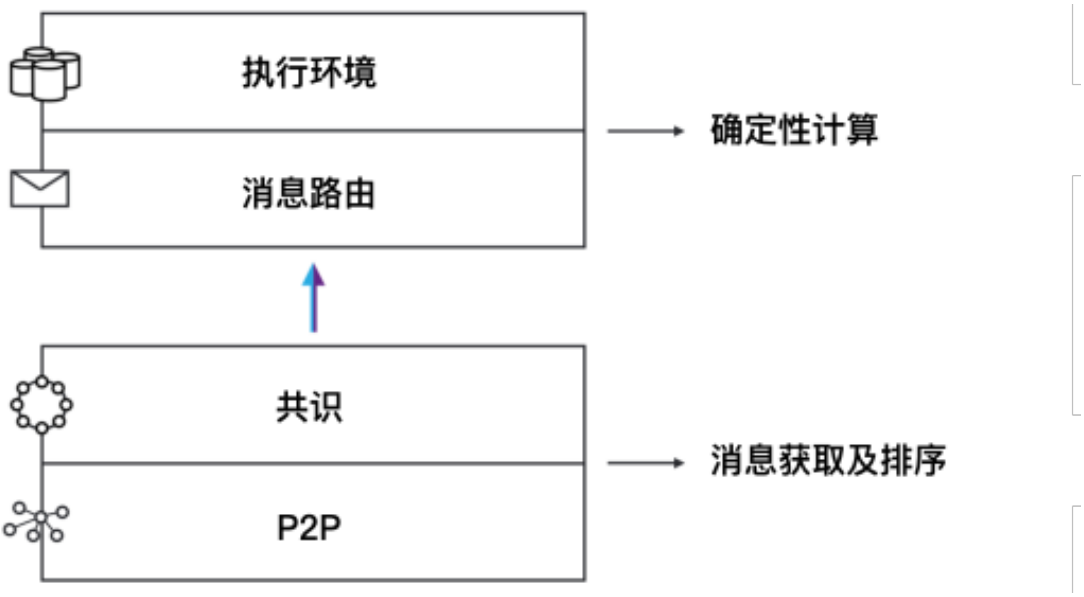

图1：互联网计算机协议层

如图1所示，IC协议包括四层：

- P2P层（详见第4章）
- 共识层（详见第5章）
- 路由层（详见第6章）
- 执行层（详见第7章）

链钥密码学在多个层中都有应用，将分别在第3章（阈值签名）和第8章中（链演进技术）详细介绍。

2.1 P2P层

P2P层的任务在子网的节点副本中传递协议消息。协议消息包括

- 用于实现共识的消息
- 外部用户发起的输入消息

P2P层基本上提供的是一个“最大努力（best effort）”广播通道

如果一个诚实的节点副本广播了一条消息，那么这条消息最终将会被子网中的所有诚实节点所接收。

P2P层的设计目标如下：

- **有限资源.** 所有的算法都在有限的资源(内存，带宽，CPU)下运转。
- **优先级.** 根据特定的属性(例如类型，大小和轮次)，不同的消息将按照不同的优先级进行排序。并且这些优先级的规则可能随着时间将会改变。
- **高效.** 高吞吐量比低延迟更重要。
- **抗DOS/SPAM.** 故障节点将不会影响诚实节点副本间的相互通信

2.2 共识层

IC共识层的任务是对输入消息进行排序，以确保所有的节点副本按照相同的顺序处理输入消息。现在已有很多文献中的协议是为了解决这一问题。IC采用了一种全新的共识协议，本文将用概括性的语言对其进行阐述。

任何安全的共识协议都应当确保两个属性，大体上就是：

- **安全性：** 所有的节点副本都事实上同意相同的输入顺序，和
- **活性：**所有的节点副本都应当逐一更新状态。

IC共识层的设计目标

- 极其简单，和
- 健壮：当存在个别恶意节点时，性能会柔性下降。

如上所述，我们假设作恶节点$f < n/3$(即拜占庭容错)。同时，IC在部分同步网络的假设下可以确保协议的活性，而协议的安全性甚至在完全异步的网络下依然可以保证。

像许多的共识协议一样，IC共识协议是基于区块链的。伴随着协议的推进，以创世区块为根节点的区块树将不断生长。每一个非创世区块都包含一个荷载(payload)，由一系列输入和父区块的哈希组成。诚实节点副本对这个区块树有一致的视角：尽管每个节点副本可能对这个区块树有不同的局部视角，但是所有的节点副本看到的都是这一相同的区块树。此外，伴随着协议的推进，区块树中总会有一条最终确认区块的路径。同样地，诚实节点副本对这一路径有一致的视角：尽管每个节点副本可能对这条路径有不同的局部视角，但是所有的节点副本看到的都是这一相同的路径。沿着这条路径的区块的荷载中的输入，是已经排序好的输入并将由IC的执行层进行处理。

IC的共识协议按照轮次进行处理。在**轮次**h中，一个或多个**块高**h的区块被添加到区块树中。也就是说，在轮次h添加的区块，距离根节点的距离都是h。在每一轮中，将通过伪随机过程给每一个节点副本分配一个唯一的**排位**，范围是$0, \ldots, n-1$的整数。这一伪随机过程使用了**随机信标(Random Beacon**，使用了阈值签名技术，在章节1.6.1中已经提及并将在第3章进行详述)来实现。排位最低的节点副本是该轮次的主节点。当主节点是诚实的并且网络同步时，主节点会提议一个新区块，并加入

到区块树中；此外，这将会是该轮次唯一添加到区块树中的区块，并延伸最终确认的路径。如果主节点不诚实或者网络不同步状态，其他排位更高的节点副本也可以提议新区块，并将其添加到区块树中。在任何情况下，协议逻辑给予主节点提议的区块最高优先级，并且某些区块会在该轮次被加入到区块树中。即使协议在没有延伸最终确认路径的情形下继续执行，区块树的高度也会在每轮继续增长，这样最终确认路径会在轮次h继续延伸，使其长度达到h。上述做法下，即使*故障节点*或不可预测的高网络延迟导致延迟增加，整体协议的*吞吐量*大体上依旧维持稳定。

共识协议依赖于电子签名在节点副本中去验证消息。为实现这一点，每一个节点副本都与签名协议的一个公共验证密钥相关联。而节点副本和公钥之间的关联性可以从NNS维护的注册表中获取。

2.3 消息路由

如章节1.7中所述，IC中的基本计算单元叫做储罐。IC提供了运行环境，使得储罐中可以执行程序，并可以（通过消息）与其他储罐和外部用户通信。

共识层将输入打包进区块的**荷载**中，并随着区块被最红确认，相应的荷载会被传递给**消息路由层**并由**执行环境**处理。执行层将随之更新复制状态机中相应储罐中的状态，并将输出交由 **消息路由层**处理。

有必要区分两种输入类型：

入口消息：来自外部用户的消息

跨子网消息：来自其他子网的储罐的消息

我们同样可以区分两种输出类型：

入口消息响应：对于入口消息的响应（可被外部用户取回）

跨子网消息：传输给其他子网储罐的消息

当收到来自共识的负载后，这些负载中的输入会被置入不同的**输入队列**。对于一个子网下的每一个储罐 C，都存在多个输入队列：用于发给C的的入口消息，用于和C通信的每一个别的储罐C'，用于C到C'的跨子网消息。

在每一轮中，执行层都会消耗这些队列中的一些输入，更新相应储罐中的复制状态，并将输出置于不同的队列中。对于一个子网下的每一个储罐C，都存在多个输出队列：对于每一个与C通信的储罐C'，都有一个用于C到C'的跨子网消息的队列。消息的路由层将取得消息队列中的消息并置入**子网-到-子网的数据流**，以被跨子网传输协议处理，该协议的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，同时存在一种入口消息的历史数据结构。一旦一条入口消息已被储罐处理，对该条入口消息的响应将被记录在该数据结构中。此刻，提供该条入口消息的外部用户将能够获取相关的响应。（注意入口历史并不保留所有入口消息的完整历史）

需要注意的是，节点副本的状态包括储罐的状态以及“系统状态”。“系统状态”包括上述提及的队列，数据流以及*入口历史*的数据结构。因此，消息路由层和执行层同时参与更新和维护子网的副本状态。此状态应全部在完全**确定性**的原则下被更新，这样所有的节点都会维护完全相同的状态。

另外需要注意的是，共识层解耦于消息路由层以及执行层，也就是说传入荷载之前，共识区块链中的任何分叉都已经被解决了。事实上，共识层允许提前运行，并不需要和消息路由层保持完全一致的进度。

2.3.1 每轮认证状态(Per-round certified state)

在每一轮中，子网的一部分状态会被验证。每轮认证状态采用链钥密码学进行验证。除了别的之外，给定轮次中的验证状态包括

- 最近添加到子网间数据流中的跨子网消息
- 其他元数据，包括入口历史的数据结构

每轮认证状态利用了阈值签名技术验证（详见章节1.6.1）。在IC中每轮认证状态被用于如下几个方面：

- **输出验证.** 跨子网消息和入口消息响应都使用每轮认证状态进行验证。
- **防止和识别非确定性.**共识层保证了每个节点副本按相同的顺序处理输入消息。因为每个节点副本确定性得处理消息，应当会得到相同的状态。但是IC额外设计了一层确保健壮性，来防止和识别任何意外的非确定性计算的发生。每轮认证状态是该机制中的一环。
- **与共识层协作.** 每轮认证状态还通过两种方式和执行层和共识层进行协作：
  - 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”。
  - 共识层的输入必须经过特定的有效性验证，这些验证取决于所有节点副本已达成共识的验证状态。

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

正如我们之前所阐述的，所有的入口消息必须经过共识，才能被子网的所有节点副本按相同的顺序进行处理。但是，针对那些处理时不会变更状态的入口消息，可以进行一项重要的优化。他们被称为查询调用——相对于其他入口消息，被称为**更新调用**。查询调用被允许进行只读或是可能改变储罐状态的计算，但是任何对节点副本状态的更新都不会被提交给复制状态。正因如此，查询调用可以被单个节点副本直接处理而不需要经过共识，这极大的降低了从查询调用获得响应的延迟。

一般而言，对查询调用的响应不会记录在*入口历史*的数据结构中，因此也不可以用之前的每轮认证状

态进行验证。然而，IC使储罐可以存储数据（在处理更新调用时）在特殊的经认证变量中(certified variables)，可以利用这一机制验证数据的有效性；这样的话，查询调用可以返回值并存储在经认证变量中，仍然可以被验证。

2.3.3 外部用户验证(External user authentication)

入口消息和跨子网消息的一个主要区别在于用于验证消息的机制。链钥密码学用于验证跨子网消息，另一个不同的机制用于来自外部用户的入口消息。

IC中没有外部用户的中央注册表。相反，外部用户通过一串公钥哈希作为**用户标识**（又称*principal*）来向储罐来标识自己。用户自己持有对应的签名密钥，用来签署入口消息。签名和公钥会随着入口消息一起发送。IC将自动验证签名并传递用户标识给到对应的储罐。随后该储罐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。

新用户在首次与IC交互时会生成一对密钥对并从公钥中衍生出他们的用户标识。老用户根据存储在用户代理中的私钥完成验证。用户还可以用签名委托的方式，将多个密钥对关联到一个用户身份上。该特性非常有用，因为它允许了一个用户在多个设备上通过相同的用户身份证明来访问IC。

2.4 执行层

执行层一次处理一个输入消息。这一输入消息取自输入消息队列，并导向到一个储罐。根据此输入消息和该储罐的状态，执行层环境将更新储罐的状态，另外将消息加入输出队列并更新*入口历史*（可能包括对更早的入口消息的响应）。

每个子网都可以访问**分布式伪随机生成器(PRG)**。二进制伪随机数的种子被称为随机磁带(Random Tape)(参见章节1.6.1，详见第3章)的阈值签名。共识协议的每一轮都会有一个不同的随机磁带。

随机磁带的基本特性有

1. 在块高h的区块被任意诚实节点副本确认之前，块高h+1的随机磁带是不可预测的。
2. 在块高h+1的区块被任意诚实节点副本确认的时间点前，节点副本已经有构建块高h+1的随机磁带的所有片段

例如在轮次h时，子网为获得二进制伪随机数，需要向执行层发起“系统调用”。系统随后将响应块高h+1的随机磁带。根据上述的特性(1)，协议保证在子网发送请求时，请求的二进制伪随机数不可预测。事实上，共识层会将随机磁带和荷载都传递给消息路由层；根据上述的特性(2)，一般不会引起任何额外的延迟。

2.5 组合阐述

我们追踪一个用户请求在IC上的典型流程。

查询调用

1. 用户通过用户端向边界节点（详见章节1.9）发送对于容器C的请求消息M，边界节点将消息M发送给托管着容器C的子网节点副本。在收到M后，节点副本将计算响应并通过边界节点发回给用户。

更新调用

1. 用户通过用户端向边界节点（详见章节1.9）发送对于储罐C的请求消息M，边界节点将消息M发送给托管着容器C的子网节点副本。
2. 接收到消息M后，该节点副本通过P2P层（详见章节2.1）向子网中的所有节点副本广播消息M。
3. 在接收到消息M的情况下，共识层下一轮的主节点（详见章节2.2）会将消息M和其他输入一同打包进其提议的区块B。
4. 一段时间之后，区块B被最终确认，其荷载被发送至消息路由层（详见章节2.3）进行处理。需要注意的是，P2P层同样用于共识层去确认区块。
5. 消息路由层会将消息放置在储罐C的输入消息队列中。
6. 一段时间之后，执行层（详见章节2.4）会处理消息M，并更新储罐C的内部状态。

   在一些情况下，储罐C能够计算对于请求消息M的响应R。在这种情况下，响应R会被记录在入口历史的数据结构中。

   在其它情况下，处理请求消息M需要向别的储罐发起请求。在这个例子中，我们假设为处理请求消息M，储罐C需要向另一个子网的另一个储罐C'发起请求M'。这第二个请求M'会被放置在储罐C的输出队列中，然后接下去的几步将被执行。
7. 一段时间之后，消息路由层会将调用请求M'移动到合适的跨子网数据流中，最终将会被传输到托管储罐C'的子网。
8. 在第二个子网中，获取来自第一个子网的请求M'后，其会通过共识层和消息路由，最终由执行层进行处理。第二个子网的执行层会更新储罐C'的内部状态，然后生成对于请求M'的响应R'。响应R'会进入储罐C'的输出队列，最终放置在跨子网数据流中被传输回第一个子网。
9. 回到第一个子网，当其获取来自第二个子网的响应R'后，响应R'经由共识层和消息路由层，并最终由执行层进行处理。第一个子网的执行层会更新储罐C的内部状态，然后生成对于原始请求M的响应R。这一响应R会被记录在入口历史的数据结构中。

无论是哪条执行路径，对于请求消息M的响应R，最终会被记录在托管储罐C的子网的入口历史数据结构中。为了获取这一响应结构，用户端必须执行“查询调用”（详见章节2.3.2）。就像在章节2.3.1中所阐述的，这一响应可以被链钥密码学（具体来说，使用的是阈值签名技术）进行验证。这一验证逻辑本身（即阈值签名认证）可以被用户端执行，其使用了最初从边界节点获取的service worker。

3 链钥密码学I：阈值签名

IC的链钥密码学的一个关键组成部分是阈值签名方案[Des87]。IC在多处应用了阈值签名。假设子网中的节点副本数量为n，故障节点数量上限为f。

- 共识层使用$(f+1)/n$的阈值签名来实现*随机信标*（详见章节5.5）。
- 执行层使用$(f+1)/n$的阈值签名来实现*随机磁带*，用于向储罐提供不可预测的伪随机数（详见章节7.1）。
- 执行层使用$(n-f)/n$的阈值签名来*认证复制状态*。这既用于验证子网的输出（详见章节6.1），也用于实现IC*链演进技术*中的*快速转发*功能。（详见章节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，即给定公钥和消息，仅有一个有效签名。因为我们使用签名作为随机数生成器的种子，所有计算该阈值签名的节点副本必须对相同的种子达成共识。

3.1 BLS阈值签名

我们基于BLS签名方案实现了阈值签名[BLS01]，使得调整阈值设定十分简单。

普通的BLS签名方案（即非阈值）利用两个质数阶均为q的群$\mathbb{G}$和$\mathbb{G}'$。我们假设$\mathbb{G}$通过基准点$g \in \mathbb{G}$生成，$\mathbb{G}'$通过基准点$g' \in \mathbb{G}'$生成。我们同样假设一个哈希函数$H_{\mathbb{G}'}$可以将其输入映射到$\mathbb{G}'$（一个随机预言机模型）。签名私钥元素$x \in \mathbb{Z}_q$，公共验证密钥$V := g^x \in \mathbb{G}$。

非阈值的设定下，为对消息m签名，签名人计算$h' \leftarrow H_{\mathbb{G}'}(m) \in \mathbb{G}'$，然后计算签名$\sigma := (\mathrm{h}')^x \in \mathbb{G}$。为验证签名是否有效，必须检验$log_{\$h\$'}\sigma$是否等于$log_g V$。为了高效地执行检验，BLS方案在群$\mathbb{G}$和$\mathbb{G}'$上使用了**配对**的概念，这是一种特殊的代数工具，当$\mathbb{G}$和$\mathbb{G}'$是特殊类型的**椭圆曲线**时可用。我们无法在这里详细介绍配对和椭圆曲线。更多细节详见[BLS01]。BLS签名具有签名唯一这一良好的特性（如上所述）。

在t/n的阈值设定下，我们有n个节点副本，其中任意t个都可以用于生成消息的签名。更具体来看，每个节点副本P_j私下持有签名密钥$x \in \mathbb{Z}_q$的一个片段$x_j \in \mathbb{Z}_q$，而群元素$V_j := g^{x_j}$公开可用。密钥片段$(x_1, \ldots, x_n)$是x的t/n秘密共享。

给定消息m，节点副本p_j可以生成**签名片段**

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

其中同之前一样，$h' := H_{\mathbb{G}'}(m)$。为了验证这样的签名片段是否有效，必须检验$log_{\$h\$'}\sigma_j$是否等于$log_g V_j$。如上所述，这可以通过配对来完成——事实上，这和用公钥V_j验证普通BLS签名有效性完全一样。

我们的阈值签名方案满足如下的**重构特性**：

给定消息m的任意t个有效签名片段σ_j的集合（由不同的节点副本提供），我们可以高效得计算出公共验证密钥下，消息m的有效BLS签名σ。

事实上，σ可以按如下公式计算

$$\sigma \leftarrow \prod_j \sigma_j^{\lambda_j} \$ \ (1)$$

其中，λ_j可由签名该消息m的t个节点副本的索引，高效地计算出来。

此方案在$\mathbb{G}$的合理难度假设下，将$H_{\mathbb{G}'}$建模成随机预言机，可以满足如下的**安全性**：

假设最多有f个节点副本被敌对方破坏。它也无法计算出消息的有效签名，除非他获取了至少$t - f$个诚实节点副本对该消息的签名片段。

3.2 分布式密钥分发

为了实现BLS阈值签名，我们需要一种分发签名私钥给节点副本的方法。一种实现方法是让一个**可信方**直接计算所有这些私钥片段并分发给所有的节点副本。不幸的是，这可能会造成单点故障。相反，我们使用了**分布式密钥生成（以下简称DKG）协议**，它本质上允许节点副本在安全的分布式协议下，执行此类可信方的逻辑。

我们概述了当前实现的协议的顶层思想。推荐读者阅读[Gro21]了解更多细节。我们使用的DKG本质上是非交互式的。它有两个基本组成部分：

- **公开可验证秘密共享（以下简称PVSS）**方案，和
- **共识**协议。

尽管任何共识协议都适用，但毫无疑问我们使用了第5章（也可见第八章）中的协议。

3.3 假设

所做的基本假设和第一章中所概述的一致：

- 异步通信，和
- $f < n/3$。

我们仅仅（在章节5.1中）间接地使用了*部分*同步假设，来确保共识协议的活性。

对于一个t/n的阈值签名方案，我们还假设

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

这（除了其他方面外）确保了（1）故障节点副本们无法独立地签名，和（2）诚实的节点副本们可以独立地签名。

我们还假设每个节点副本和一些公钥相关联，其中每个节点持有这些公钥的对应私钥。一个公钥是签名密钥（同章节5.4中一样）。另一个公钥是公共加密密钥，用于实现PVSS方案的特殊公钥加密方案（详细方案如下）。

3.4 PVSS方案

如上所述，假设$\mathbb{G}$通过基准点$g \in \mathbb{G}$生成的质数阶为q的群。假设$s \in \mathbb{Z}_q$是私钥。回想一下，阈值结构为t/n情况下，密钥s的Shamir密钥共享的是向量$(s_1, \ldots, s_n) \in \mathbb{Z}_q^n$，其中

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

而

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

是次数小于t的多项式，其中$a_0 := s$。该类密钥共享方案的关键属性是

- 根据t个s_j的集合，我们可以高效地计算密钥$s = a_0 = a(0)$，和
- 如果从$\mathbb{Z}_q$中一致且独立得抽取$a_1, \ldots, a_{t-1}$，任意少于t个元素的集合都无法披露密钥s的任何信息。

在更高层面上，PVSS方案允许节点副本P_i，被称为**dealer**，进行密钥共享并计算称为**dealing**的对象，其中包括

- 群元素的向量$(A_0, \ldots, A_{t-1})$，其中$A_k := g^{a_k}$对于$k = 0, \ldots, t-1$，

- 密文向量$(c_1, \ldots, c_n)$，其中c_j是P_j的公共验证密钥下对s_j的加密，
- 非交互式的零知识证明π，证明每个节点副本c_j确实加密了密钥片段——更准确地说，每个节点副本c_j解密的值满足

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

我们注意到，为了确保DKG协议的整体安全性，PVSS方案必须提供适当级别的所选密文安全性。具体来说，dealer必须将其身份作为关联数据嵌入dealing，并且加密的共享片段必须要保持隐匿，即使是在选择密文攻击中，敌对方可以用与创建dealing无关的关联数据，解密任意dealing。

如果不考虑效率问题，实现PVSS方案很容易。方案中利用类似ElGamal的加密方案，对每个s_j逐位加密，然后对公式（2）使用标准的非交互式零知识证明，该证明基于对正确的Sigma协议(详见[CDS94])应用标准的Fiat-Shamir转换(详见[FS86])实现。尽管这产生了一个多项式时间的方案，但是并不实用。然而，有很多可能的方法来优化这个类型的方案。关于IC中使用的高度优化的PVSS方案，详见[Gro21]。

3.5 基础DKG协议

使用PVSS方案和共识协议，基础的DKG协议十分简单。

1. 每个节点副本向其他节点副本广播关于随机密钥的**已签名dealing**。

   这样的已签名dealing包括一个dealing主体，dealer身份以及dealer公钥下对该dealing的签名。

   如果语法格式正确，签名和非交互式的零知识证明有效，那么这样的签名dealing是有效的。
2. 利用共识协议，节点副本对$f+1$的有效的已签名dealing集合S达成共识。
3. 假设集合S中的第i个dealing包含群元素向量$(A_{i,0}, \ldots, A_{i,t-1})$和密文向量$(c_{i,1}, \ldots, c_{i,n})$。

   然后阈值签名方案的公共验证密钥是

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

   注意，签名私钥被隐式定义

   $$x := log_g V$$

   节点副本P_j的签名私钥x的片段为

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

   其中$s_{i,j}$是p_j的解密私钥下对$c_{i,j}$的解密。

   节点副本P_j的公共验证密钥是

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

需要注意的是，密钥片段x_j包含x的阈值结构为t/n的Shami密钥共享。因此，公式（1）中出现的只是拉格朗日插值系数。这证实了章节3.1中提到的*重构特性*。至于中章节3.1提及的安全性，可被如下证明，在$H_{\mathbb{G}'}$建模成随机预言机的情况下，假设PVSS方案是安全的，群$\mathbb{G}$和$\mathbb{G}'$（通过配对）满足**one-more Diffie-Hellman**的特定类型的难度假设，即不存在有效的对手方，有概率赢得下面的博弈：

挑战者随机选择$\mu_1, \ldots, \mu_k \in \mathbb{Z}_q$*和*$v_1, \ldots, v_l \in \mathbb{Z}_q$*，将给到对手方*$\{g^{\mu_i}\}_{i=1}^k$*和*$\{(g')^{v_j}\}_{j=1}^l$

对手方向挑战者发起一系列查询请求，每一个请求都是$\{k_{i,j}\}_{i,j}$*形式的向量，挑战者回应对应的*

$$\prod_{i,j}((g')^{\mu_i v_i})^{k_{i,j}}$$

为结束这场博弈，对手方输出向量$\{\lambda_{i,j}\}_{i,j}$*和群元素*$h' \in \mathbb{G}'$*，并将赢得博弈如果*

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

并且输出向量$\{\lambda_{i,j}\}_{i,j}$*不是请求向量的线性组合。*

尽管当$t > f + 1$的情况下，需要这类one-more Diffie-Hellman假设，但是当$t = f + 1$的情况下，可以采用较弱的假设(即所谓的co-CDH假设，普通BLS签名方案的安全性即基于此)。

3.6 再共享协议

基础DKG协议可以被很容易地修改，因此不需要创建一个新的随机密钥x的共享，而是创建一个先前共享密钥的新随机共享。

- 修改基础协议的第1步，以便每个节点可以广播已有共享片段的已签名dealing
- 修改第2步，以便就t个有效的已签名dealing集合达成共识。此外，每个dealing都经验证以确保是现有共享片段的dealing（这意味着在第i个dealing中$A_{i,0}$的值应当等于V_i的旧值）。
- 第3步中，通过计算i个拉格朗日插值系数的和（和乘积），等于新的x_j（和V_j）。

4 P2P层

P2P层的任务是在子网的节点副本中传递协议消息。这些协议消息包括

- 用于实现共识的消息，例如，区块提案（block proposals），公证（notarizations）等（详见第5章）
- 入口消息（详见第6章）

本质上，P2P协议提供的服务是“最大努力（best effort）”的广播通道：

如果一个诚实节点副本广播一条消息，那么这条消息最终会被子网中的所有诚实节点副本接受到。

协议的设计目标包括以下内容：：

- **有限资源.** 所有的算法都在有限的资源(内存，带宽，CPU)下运转。
- **优先级.** 根据特定的属性(例如类型，大小和轮次)，不同的消息将按照不同的优先级进行排序。并且这些优先级的规则可能随着时间将会改变。
- **高效** 高吞吐量比低延迟更重要。
- **抗DOS/SPAM.** 故障节点将不会影响诚实节点副本间的相互通信

注意到，在共识协议中，一些消息，尤其是区块提案（会非常大），将被所有节点副本反复广播。这对确保协议的正确运作是必须的。但是，如果单纯这样实现，将会是巨大的资源浪费。为了避免节点副本广播相同的消息，P2P层使用了**公告-请求-传递**的机制。它不会直接传递（数据量大的）消息，而是广播该消息的（数据量小的）**公告**：如果节点副本收到这样的公告并且没有收到过对应的消息，将认定该消息是重要的，节点副本将**请求**该消息的**传递**。这个策略以更高延迟的代价，降低了带宽使用。对于小消息而言，牺牲延迟追求带宽是不值得的，可以直接发送消息而不是公告。

对于相对较小的子网，希望广播消息的节点副本，会将公告发送至子网中的所有节点副本。对于较大的子网，**公告-请求-传递**机制可以在一个**覆盖网络（overlay network）**上运行。覆盖网络是一个连接的无向图，子网内的节点副本组成其顶点。如果图中有一条边连接两个节点副本，那他们是**对等节点（peers）**，节点副本仅和它的对等节点通信。因此，当节点副本希望广播消息时，他将该消息的公告发送给它的对等节点。这些对等节点在收到公告后，可能会请求消息的传递，如果满足特定条件，这些对等节点会将该消息的公告发送给他们的对等节点。这本质上是一个**gossip network**。这个策略用比先前更高的延迟代价，再次降低了带宽使用。

5 共识层

IC共识层的工作是为输入排序以使一个子网下所有节点副本按照相同的顺序处理输入。 目前有许多协议来解决这个问题，其中IC使用了一种新的共识协议，将在这一章进行简明扼要的阐述。更多的细节，参考[CDH+21] (特别是该论文中的 Protocol ICC1). 任何安全的共识协议应当确保以下两种属性，（粗略地）表述为：

- **安全性**：全部节点副本事实上同意相同的输入顺序
- **活性**：全部节点副本都应取得持续的进展

该论文[CDH+21] 证实了IC共识协议同时满足以上两种属性。IC的共识协议在设计上：

- 极其简单，且
- 健壮的：当存在部分恶意节点副本，性能也能优雅地下降。

5.1 假设

如引言所述，我们假设

- 一个子网包含n个节点副本，和
- 其中最多$f < n/3$的节点副本存在故障

故障节点副本可能表现出随意、恶意（即拜占庭）的行为。

我们假设通信为**异步**且不存在*先验*对节点副本间消息延迟的限制。事实上，消息传递的调度可能完全处于敌对状态。在此弱通信假设下，IC共识协议可以确保安全性。但是为确保活性，我们需要假设一定形态的**部分同步**，粗略着意味着网络在较短的间隔中保持周期性的同步。在此同步间隔下，所有未递交信息将在δ时间完成递交，即固定的时间限制δ。时间限制δ不会被事先得知（协议会初始化合理的边界值但动态的调节该值，且当时间限制过小时提高限制值）。无论网络为异步或是部分同步，我们假设诚实节点副本发送给另一个节点副本的消息**最终**都会递交。

5.2 协议概述

像许多的共识协议一样，IC共识协议是基于区块链的。伴随着协议的推进，以**创世区块（genesis block）**为根节点的区块树将不断生长。每一个非创世区块都包含一个**荷载(payload)**，由一系列输入和父区块的哈希组成。诚实节点副本对这个区块树有一致的视角：尽管每个节点副本可能对这个区块树有不同的局部视角，但是所有的节点副本看到的都是这一相同的区块树。此外，伴随着协议的推进，区块树中总会有一条最终确认区块的路径。同样地，诚实节点副本对这一路径有一致的视角：尽管每个节点副本可能对这条路径有不同的局部视角，但是所有的节点副本看到的都是这一相同的路径。沿着这条路径的区块的荷载中的输入，是已经排序好的输入并将由IC的执行层进行处理（详见第7章）。

IC的共识协议按照轮次进行处理。在**轮次**h中，一个或多个**块高**h的区块被添加到区块树中。也就是说，在轮次h添加的区块，距离根节点的距离都是h。在每一轮中，将通过伪随机过程给每一个节点副本分配一个唯一的**排位**，范围是$0, \ldots, n-1$的整数。这一伪随机过程使用了**随机信标(Random Beacon)**（详见如下章节5.5）来实现。排位最低的节点副本是该轮次的**主节点**。当主节点是诚实的并且网络同步时，主节点会提议一个新区块，并加入到区块树中；此外，这将会是该轮次唯一添加到区块树中的区块，并延伸最终确认的路径。如果主节点不诚实或者网络不同步状态，其他排位更高的节点副本也可以提议新区块，并将其添加到区块树中。在任何情况下，协议逻辑给予主节点提议的区块最高优先级，并且某些区块会在该轮次被加入到区块树中。即使协议在没有延伸最终确认路径的情形下继续执行，区块树的高度也会在每轮继续增长，这样最终确认路径会在轮次h继续延伸，使其长度达到h。上述做法下，即使*故障节点副本*或不可预测的高网络延迟导致延迟增加，整体协议的*吞吐量*大体上依旧维持稳定。

5.3 额外特性

一个由IC共识协议带来的额外属性(如同 PBFT [CL99]和HotStuff [AMN+20], 而不像别的协议, 诸如 Tendermint [BKM18])是*乐观响应*[PS18]。这意味着，当主节点诚实时，协议会以实际网络延迟而非网络延迟上限继续执行。

我们注意到一个IC共识协议下的简单设计也确保了当拜占庭故障实际发生时，IC的性能也能相当优雅的下降。如同[CWA+09]中指出的，目前大量的工作都专注于无故障、乐观情况下的性能提升，这导致协议变得危险且脆弱，以至于故障真的发生时协议无法使用。如[CWA+09]展示的，现存PBFT事先的吞吐量在特定(及其简单)的拜占庭行为下下降至0. 该论文[CWA+09]倡导*健壮*的共识，意味着 最优状态下的*最高*吞吐会被部分牺牲以确保部分故障下的*合理*吞吐(但依然假设网络为同步)。在论文[CWA+09]的定义下，IC共识协议确实是是健壮的:当任何一轮主节点发生故障时（发生概率小于$1/3$），协议有效的允许别的节点副本无障碍的取代之并成为主节点，使得协议能在规定时间内进行到下一轮。

5.4 公钥

为实现该协议，每一个节点副本都关联了一个用于BLS签名的公钥，并且每一个节点副本都拥有对应的私钥。节点副本与公钥的关联关系可从NNS维护的注册表中获得。这些BLS签名将用于验证节点副本发送的消息。 协议同时使用BLS签名[BGLS03]的**签名聚合**特性，该特性允许同一条消息的许多签名聚合为一个紧凑的多签。协议将这些多签用于**公证（notarization）**（见[章节5.7）and **最终确认（finalizations）**（见 章节 5.8），指特定形态消息上$n-f$个签名的聚合。

5.5 随机信标

除了上述的BLS签名与多签外，协议使用BLS阈值签名方案来实现上文提及的随机信标。块高h的随机信标是位于一个专属于块高h中消息的一个$(f+1)$阈值 的签名。在协议的每一轮中，每一个节点副本广播其在下一轮中信标的片段，这样当下一轮开始时，每一个节点副本都有足够的片段来重构此轮需要的信标。如上所述，在轮次h，块高h的随机信标被用来给每个节点副本分配伪随机排位。因

为阈值签名的安全特性，敌对方将无法提前超过一轮去预测节点副本的排位，因此这些排位拥有有效的随机性。阈值签名参考第3章。

5.6 区块生成

每一个节点副本可能在不同的时间点成为区块生成者。作为轮次h的区块生成者，此节点副本提议一个块高h的区块B，此区块为区块树中$h-1$区块B'的子区块。做法为该区块生成者首先合并一个包含已知所有输入的荷载(但并不包括B之前路径上的已有区块中荷载)。区块B由以下组成：

- 荷载，
- B'的哈希，
- 区块生成者的排位，
- 区块的高度 h。

  当组成区块*B*后，区块生成者会生成一份**区块提案**，组成如下：
- 区块B，
- 区块生成者的身份标识，以及
- B上的区块生成者的签名。

区块生成者将其区块提案广播给其他所有的节点副本。

5.7 公证

当区块完成公证时，该区块才可被有效的纳入区块树中。为使区块实现公证，$n-f$个不同的节点副本必须**支持**公证。

假定一个在块高h提议的区块B，节点副本会根据上文提及的区块组成来判断该区块提案是否有效。细节上，B应该包括区块树中块高为 h' 区块 *B'* 的哈希(即已被公证)。另外，B的荷载必须满足特定的条件(细节为所有荷载中的输入必须满足多条限制，但是这些限制整体上独立于共识协议)。同时，区块生成者的节点副本排位(记录于区块B)必须对应随机信标在h轮中分配的，负责提议区块的节点副本排位(记录在区块提案中)。 如果该区块有效且满足别的特定限制，此节点副本将广播B的**公证片段**以**支持**该区块的公证，**公证片段** 组成如下：

- B的哈希，
- B 的块高，
- 支持节点副本的身份标识，以及
- 支持节点副本B的签名的消息，包含B的哈希以及块高 h

任意包含$n-f$公证片段的集合可被合并以形成B的**公证**，组成如下：

- B的哈希
- B的块高
- 由$n-f$个支持节点副本的身份证明组成的集合
- 一条消息上的$n-f$的签名聚合，包含B的哈希以及块高h

一旦一个节点副本获得了公证后块高为h的区块，它会完成轮次h并随即不支持其它块高h的区块，同时*中继（relay）该公证给其他的所有节点副本*。注意该节点副本有2种方式获得公证，（1）从其它节点副本收到 或（2）收到的$n-f$部分公证的聚合。

生长不变性阐明了每个诚实节点副本最终都会完成每一轮且开始下一轮次，因此经过公证的区块树会继续增长(并且该理论仅假设异步的最终递交且不是部分同步)。我们在下方证明了生长不变性（详见章节5.11.4）。

5.8 最终确认

在给定块高h可能存在多个经公证的区块，但是一旦区块被**最终确认**，则我们可以确定在块高h没有其它经公证的区块。让我们称之为**安全不变性**。 为使一个区块实现最终确认，$n-f$个不同的节点副本必须支持其最终确认。回想在轮次h结束时，当一个节点副本获得一个经公证的块高h的区块B，*此时，此节点副本将检查其是否支持过*B以外块高h下其它区块的公证。如果不，则该节点副本会广播**最终确认片段**以支持B*的最终确认。最终确认片段的格式与公证片段的格式完全相同(但是通过特定方式标注以防止混淆)。任何*B集合的$n-f$最终确认片段都可以被聚合形成B的**最终确认**，其拥有与公证相同的格式。任何收到最终确认区块的节点副本都将广播最终确认给其它的节点副本。

我们在下方证明了**安全不变性**（详见章节5.11.5）。 安全不变性的结果如下：假设两个区块B和B'都被最终确认，其中*B*的高度为h，B'的高度为$h' \leq h$。则安全不变性意味着结束于B'且经公证的区块树的路径为结束于B的路径的前缀（如果不，则在块高h'将会有2个公证后的区块，这违背了最终确认不变性）。因此，任何时候当一个节点副本看见最终确认的区块*B*，它可将*B*所有的祖先视作隐式的最终确认，同时又因为安全不变性，这些已经最终确认的区块安全属性可以被确保，这意味着所有的节点副本都同意最终确认区块的排序。

5.9 延迟函数

该协议应用了2种**延迟函数**，Δ_m与Δ_n，他们控制了区块的生成与公正。这两个函数都映射区块生成的排位*r*到一个非负的延迟数值，且假设每个函数都单调的增长至r，且对于所有$r=0,\ldots,n-1$，$\Delta_m(r) \leq \Delta_n(r)$。这两个函数推荐的定义是$\Delta_m(r)=2\delta r$和$\Delta_m(r)=2\delta r+\epsilon$，其中$\delta$是由一个诚实节点副本递交给另一个的时间上限，而$\epsilon \geq 0$是一个防止协议运行过快的监督者。通过这些定义，活性在以下轮次可被保证 （1）主节点是诚实的（2）诚信节点副本间的消息确实传递了。事实上，如果（1）和（2）在给定轮次种同时满足，则本轮主节点提议的区块会被最终确认。让我们把其称之为**活性不变性**。我们在下文证明该理论(见 章节 5.11.6).

5.10 例证

图2展示了一颗区块树。每一个区块都被标记了块高$(30，31，32，...)$和区块生成者的排位，该图同时展示了区块树中的每一个区块经过了公证，并以符号N标记。这意味着每一个区块树里的经公证的区块至少得到了$n-f$个不同节点副本的公证支持。可以发现，在指定的块高可以存在超过一个经公证的区块。举例说明，在块高32，我们可以看到2个经公证的区块，一个由排位为1的区块生成者提议，而另一个由排位2的提议，同样的事情也是发生在块高34。我们也可以看到块高36的区块也明确的被最终确认，就如符号F所标识。这意味着$n-f$不同个节点副本支持过该区块的最终确认，意味着这些节点副本(或至少这些中诚实节点副本)不支持其它任何区块的公证。该区块被灰色填充的所有祖先都被认为得到了隐性的最终确认。

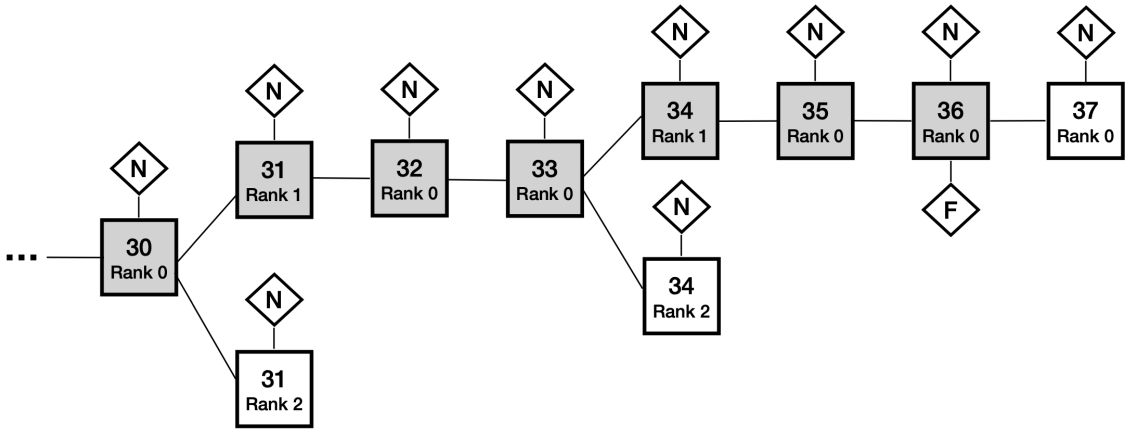

图2: 区块树的一个例子

5.11 组合详述

我们现在更详细地描述该协议的工作原理；具体上，我们更准确地描述一个节点副本何时会提议一个块及何时支持一个区块的公证。给定的节点副本P将记录轮次h的时间，这发生在当它得到（1）一些块高h-1的的公证（2）轮次h的随机信标。因为轮次h的随机信标已被决定，P可以判定其自身的排位r_P以及h轮每个其它节点副本的排位r_Q。

5.11.1 随机信标详述

一旦一个节点副本接受到了轮次h的随机信标或是足够的片段以构建轮次h的随机信标，它将在轮次h中继该随机信标给所有的其它节点副本。一旦一个节点副本进入了轮次h，它将生成并广播它在轮次h+1中随机信标的片段。

5.11.2 区块生成详述

节点副本P只会在2种情况下提议区块B_p（1）自轮次开始已经过去至少$\Delta_m(R_P)$的时间单位（2）无有效的更低排位的区块被P发现。

注意因为当进入h轮次时P保证了拥有经公证且位于块高h-1的的区块，它可将其提议的区块作为该经公证的区块的字区块（或是它拥有的其它经公证的块高h-1的任意区块）。同时注意当p广播其B_p的区块提案时，它必须同时确保它已经将B_p父区块的公证中继给所有节点副本。

假使节点副本Q看见了一个来自排位$r_P < r_Q$节点副本P的有效区块提案，且（1）自轮次开始至少已经过去$\Delta_m(R_P)$个时间单位（2）Q当前没有看见没有其它更低排位的区块。则在该时间点，如果Q尚未中继，Q将中继该区块提案(携带提议区块的父区块的公证)。

5.11.3 公证详述

节点副本P会支持由排位r_Q的节点副本Q提出的有效区块B_Q的公证,当（1）自轮次开始已经过去至少$\Delta_n(R_P)$个时间单位（2）当前没有其它排位小于r_Q的区块被P发现

5.11.4 生长不变性（growth invariant）证明

生长不变性阐述了每一个诚实节点副本最终都会完成当下轮次并开始下一轮次。假设所有诚实节点副本都开始了轮次h，假定r为轮次h最低排位节点副本P的排位，最终P会（1）自己提议区块 或（2）中继一个由更低排位节点副本提议的有效区块。两种情形中的任何一种，某个区块都必将被所有诚信节点副本支持，这意味着某个区块将会被公证且所有诚实节点副本将会完成轮次h。所有诚实节点副本将会接收到足以构建h+1随机信标的片段并开始轮次h+1。

5.11.5 安全不变性（safety invariant）证明

安全不变性阐述了给定轮次中如果一个区块被最终确认，则没有其它区块可以在该轮被公证。以下为安全不变性的证明：

1. 假设故障节点副本的数量恰好为$f \leq f < n/3$。
2. 如果区块B被最终确认，则其最终确认必须被由$n - f - f^*$个诚实节点副本组成的集合S支持（基于聚合签名的安全性）。
3. 假定（矛盾的情况下）另一个区块B'且$B' \neq B$被公证，则该公证必须被由至少$n - f - f^*$个诚实节点副本组成的集合支持（再次基于聚合签名的安全性）。
4. 集合S与集合S'是不相交的（基于最终确认的逻辑）。
5. 因此 $n - f^* \geq |S \cup S'| = |S| + |S'| \geq 2(n - f - f^*)$ 且意味着$n \leq 3f$为悖论。

5.11.6 活性不变性（liveness invariant）证明

我们认为，假使所有所有诚实节点副本间在时间点*t*时或前发送的的消息都在时间*t*时或前到底到达终点，则该网络在**此时为*δ*-同步**。

活性不变性可被阐述这样阐述。假设 $\Delta_n(1) \geq \Delta_m(0) + 2\delta$，且假定在给定轮次$h$，我们有

- 轮次h的主节点副本P为诚实的，
- 即将进入轮次h的第一个诚实Q节点在轮次h同样如此，和
- 该网络在时间t以及$t + \delta + \Delta_m(0)$时为δ-同步。

则在P提议的区块在轮次h将被最终确认。下面为活性不变性的证明：

1. 在时间t的部分同步下，所有的诚实节点副本都会在时间$t + \delta$前进入h轮（Q节点用以结束轮次h - 1的公证以及轮次h的随机信标在该时间前到达所有诚实节点副本）。
2. 轮次h的主节点P在时间$t + \delta + \Delta_m(0)$将提议区块B，且再次基于部分同步，该区块提案将在$t + 2\delta + \Delta_m(0)$前递达所有其它节点副本。
3. 因为$\Delta_n(1) \geq \Delta_m(0) + 2\delta$，该协议逻辑确保了每一个诚实节点副本仅支持区块$B$而非其他区块的公证，因此$B$将被公证和最终确认。

5.12 其他问题

5.12.1 生长延迟（Growth latency）

在部分同步的假设下，我们可以构想并证明生长不变性的定性版本。简单来说，假设延迟函数按照上文推荐的定义为：$\Delta_m(r) = 2\delta r$和$\Delta_m(r) = 2\delta r + \epsilon$，且进一步假设$\epsilon \leq \delta$。假设在时间t，被任何节点副本进入的最高轮次为$h$。使 r为低排位的诚实节点P在$h$轮次的排位。最终，假设该网络在间隔为$[t, t + (3r^* + 2)\delta]$的全部时间下为$\delta$-同步。那么所有诚实节点副本都会在时间$t + 3(r^* + 1)\delta$前开启轮次$h + 1$。

5.12.2 本地化调整的延迟函数（Locally adjusted delay functions）

当一个节点副本在多个轮次后都没有见到最终确认的区块，它将开始增加自身用于公证的延迟函数Δ_n。节点副本间无需同意这些本地化调整的公证延迟函数。

另外，当节点副本不直接调整他们的延迟函数Δ_p时，我们可以用数学的方式来建模本地的时钟偏移，来本地化的同时调整这2个延迟函数。

因此，会存在许多由节点副本和轮次为参数的延迟函数。因此对于活性的关键条件$\Delta_m(1) > \Delta_m(0) + 2\delta$变成$max\Delta_m(1) > min\Delta_m(0) + 2\delta$，且max和 min 值来自给定轮次的全部诚实节点副本。因此，如果最终确认失败了足够轮次，所有的诚实节点副本都会最终提升他们的公证延迟，直到条件满足，接着最终确认将继续。如果一些诚实节点副本比其他节点副本更多的增加

他们的公证延迟函数，就活性而言并不存在惩罚（但就生长延迟而言可能存在）。

5.12.3 公正性

在共识协议中另一个重要的属性是**公平性**。不同于设定一个普遍的定义，我们简单的观察到生长不变性也暗示着一个有用的公平性属性。回顾一下，活性不变性本质上意味着在任何一轮中，当主节点为诚实且网络为同步时，则主节点提议的区块也将被最终确认。在该情形发生的轮次中，诚实的主节点事实上确保了，它区块的荷载中包含它所知道的所有输入（取决于荷载大小的模块限制）。因此，粗略的来说，任何传播给足够节点副本的输入，都将大概率在合理时间内被包含入最终确认的区块中。

6 消息路由层

如同章节 1.7中讨论的，IC中的基础计算单元称为**储罐**，可以粗略的等同于*进程*的概念，其同时包含了一个*程序*和它的*状态*。IC提供了储罐中执行程序的运行时环境并使其能够于其他储罐和外部用户通信（通过消息传递）。

共识层（见第5章）将输入打包为**荷载**并置于**区块**中，且当区块被最终确认时，相应的荷载会被传递给消息路由层并由**执行环境**处理，此举将更新复制状态机中相应的储罐中的状态，并将输出交由 **消息路由层**处理。

有必要区分两种输入类型：：

- **入口消息**：来自外部用户的消息
- **跨子网消息**：来自其他子网的储罐的消息

我们也可以区分两种不同的输出：

- **入口消息响应**：对于入口消息的响应（可被外部用户取回）
- **跨子网消息**：传输给其他子网储罐的消息

当收到来自共识的负载后，这些负载中的输入会被置入不同的**输入队列**。对于一个子网下的每一个储罐 C，都存在多个输入队列：用于发给C的的入口消息，用于和C通信的每一个别的储罐C'，用于C到C'的跨子网消息。

如下详述，在每一轮中，执行层都会消耗这些队列中的一些输入，更新相应储罐中的复制状态，并将输出置于不同的队列中。对于一个子网下的每一个储罐C，都存在多个输出队列：对于每一个与C通信的储罐C'，都有一个用于C到C'的跨子网消息的队列。消息的路由层将取得消息队列中的消息并置入**子网-到-子网的数据流**，以被跨子网传输协议处理，该协议的工作是将这些消息实际传输到其他子网。

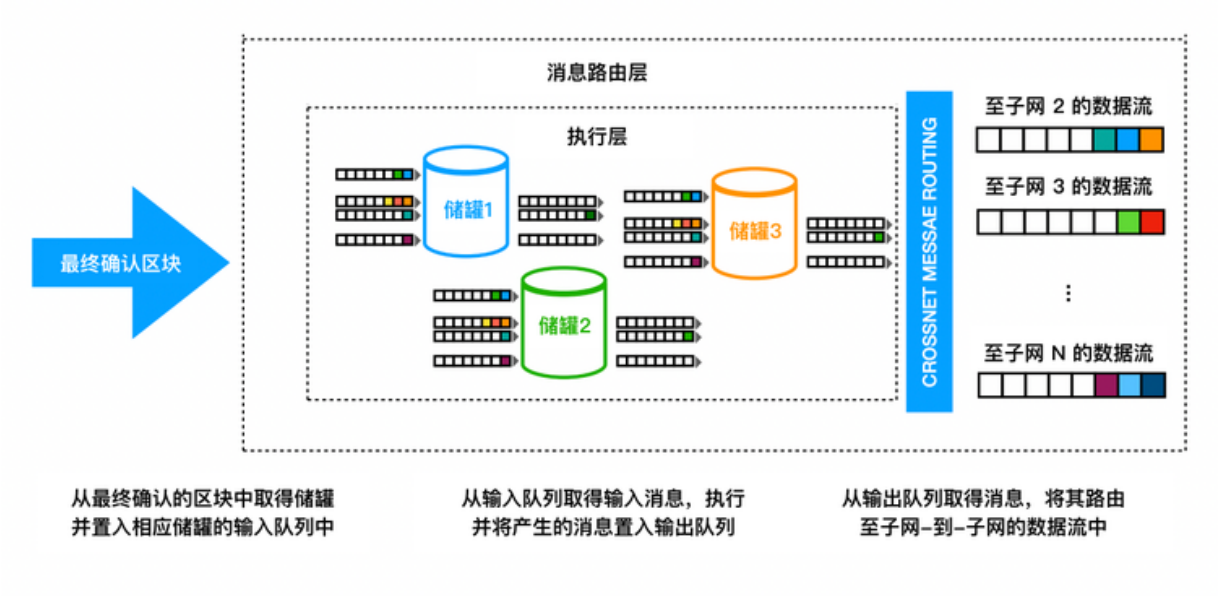

图 3: 消息路由层和执行层

除了这些输出队列外，同时存在一种入口消息的历史数据结构。一旦一条入口消息已被储罐处理，对该条入口消息的响应将被记录在该数据结构中。此刻，提供该条入口消息的外部用户将能够获取相关的响应。（注意入口历史并不保留所有入口消息的完整历史）

我们也应提及除了跨子网消息外，也存在着**子网内消息（intra- subnet messages）**，这些是*同一子网下*由一个储罐发送给另一储罐的消息。消息路由层会将这类消息由输出队列直接移至对应的输入队列中。

图3展示了消息路由层以及执行层的基础功能。

需要注意的是，节点副本的状态包括储罐的状态以及“系统状态”。“系统状态”包括上述提及的队列，数据流以及*入口历史*的数据结构。因此，消息路由层和执行层同时参与更新和维护子网的副本状态。此状态应全部在完全**确定性**的原则下被更新，这样所有的节点都会维护完全相同的状态。 另外需要注意的是，共识层解耦于消息路由层以及执行层，也就是说传入荷载之前，共识区块链中的任何分叉都已经被解决了。

6.1 每轮认证状态（Per-round certified state）

在每一轮，一个子网下的*某些*状态会被认证。**每轮认证状态**通过链钥密码学（详见章节1.6）进行验证，具体为使用了第3章提及的$(n-f)/n$阈值签名。细节上，给定轮次下当每个副本生成了生成了每轮认证状态，副本将生成相应阈值签名的片段并将其广播给所有其同一子网下的其他节点副本。一旦收集了$n-f$个此类的片段，每一个节点副本就可以构建出最终的阈值签名，该签名作用于该轮下每轮认证状态的**证书**。请注意签名前，每轮认证状态会被哈希计算为**默克尔树（Merkle Tree）**[Mer87]。 给定轮次的每轮认证状态包含

1. 最近加入子网-到-子网数据流的跨子网消息；
2. 其他元数据，包括入口历史的数据结构；
3. 来自前一轮的每轮认证状态的默克尔树的根节点哈希。

请注意每轮认证状态并不包括一个子网的完整副本状态，因为这么做整体上将非常巨大且认证每一轮的全部状态也并不实际[^5]。

图4展示了每轮认证状态是如何被组织入树中的。该树的第一个分支存储了多种关于每一个储罐的元数据（但不是储罐的完整副本状态）。第二个分支储存了入口历史的数据结构。第三个分支储存了关于子网-到-子网数据流的信息，包括了对每个数据流新加入的跨子网消息的“视图”。其他的分支存储了其他类型的元数据，这里不做讨论。该树状结构之后可被哈希计算为一棵默克尔树，其本质上拥有与该树相同的尺寸和形状。

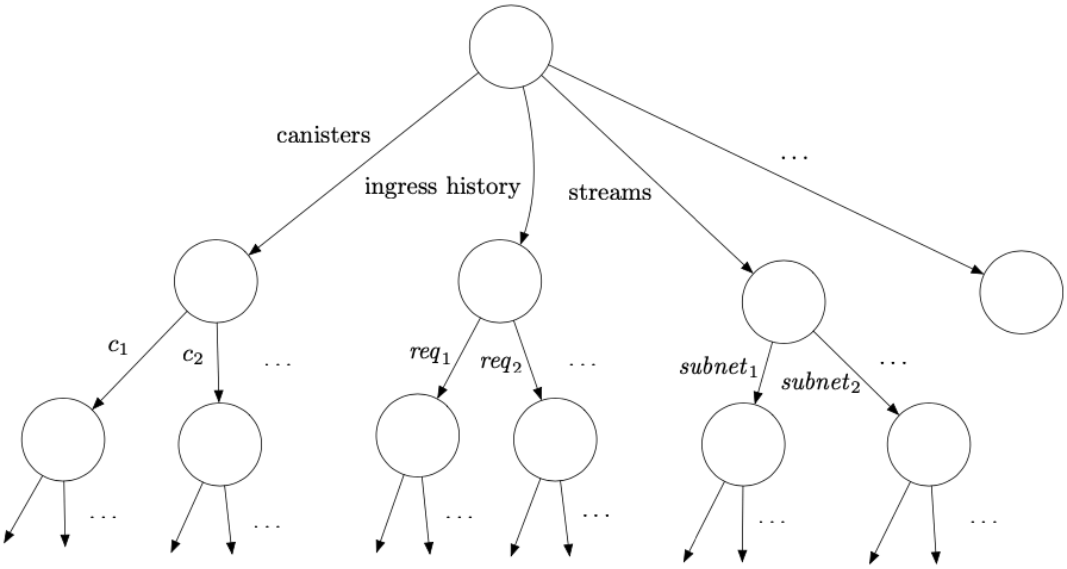

图4：每轮认证状态被组织成树

每轮认证状态在IC中有一下用例：

- **输出验证**。入口消息的跨子网消息和响应通过每轮认证状态进行验证。使用默克尔树的结构，一个单独的输出（跨子网消息或是入口消息响应）可以被任一方通过提供根节点的阈值签名来验证，以及默克尔树中从根节点至叶节点的路径的哈希值组（以及相邻的）。需验证一个单独输出的哈希值的数值因此成比例与*默克尔*树的深度，即使哈希树的尺寸非常大，该数值也通常非常小。因此，单个阈值签名可被有效的用于验证许多单独的输出。
- **防止并识别非确定性**。共识确保了每个节点在相同的顺序下处理输入。因为每个节点副本都确定性处理这些输入，每个节点副本都应取得相同的状态。但是在此应被提及，IC在设计上增加了额外的一层健壮性来阻止并发现任何（意外的）非确定性计算。其中每轮认证状态是该机制中用的一部分，因为我们使用了$(n-f)/n$的阈值签名来认证，且$f < n/3$，所以仅有单个状

态序列可被认证。

要理解为何状态成链如此重要，考虑下面的例子。假定我们有P_1, P_2, P_3, P_4四个节点副本，且其中一个节点副本P_4为恶意节点。P_1, P_2, P_3均开始与相同的状态。

- 在轮次1，由于非确定性计算，P_1, P_2均开始计算一条消息m_1以发送至子网A，而$P2$则计算$m'1$以发送至子网A
- 在轮次2，P_1, P_3均开始计算消息m_2以发送至子网B，而P_2则计算消息m'_2以发送至子网B
- 在轮次3，P_2, P_3均开始计算消息m3以发送至子网C，而P2则计算消息m'_3以发送至子网C

如下表所示：

$$P_1 \;\; m_1 \to A \;\; m_2 \to B \;\; m'_3 \to C$$

$$P_2 \;\; m_1 \to A \;\; m'_2 \to B \;\; m_3 \to C$$

$$P_3 \;\; m'_1 \to A \;\; m_2 \to B \;\; m_3 \to C$$

我们假设节点P_1, P_2, P_3分别进行了一串有效序列的计算，但是因为非确定性，这些序列并不相同（即使不应当存在任何的非确定性，我们在这个例子中假设非确定性的存在）。

现在假设我们没有将这些状态成链。因为P_4为恶意且可能签名任何内容，它可以在轮次1的状态中创建一个申明“$m_1 \to A$”的$3/4$的签名，在轮次2状态中相似申明“$m2 \to B$”并在轮次3状态中申明“$m3 \to C$”，尽管相应的序列

$$m_1 \to A, m_2 \to B, m_3 \to C$$

可能不与**任何**有效的序列兼容。更糟糕的是，此无效的计算序列可能导致了其他子网的非一致性状态。

通过成链，我们确保了即使存在一定程度的非确定性，任何经认证状态的序列都对应了*一些*由诚实节点完成计算的有效序列。

- **与共识协作**。每轮认证状态也被用于与执行层和共识层协作，有如下两种方式：
  - *共识降速（Consensus throttling）*。每个节点都会跟踪存在认证的状态的最新轮次，这被称之为**认证高度**。它也将跟踪持有一个经公证区块的最新轮次——这被称之为**公证高度**。如果公证高度明显大于认证高度，这是执行延迟于共识的信号，则共识需被降速。该延迟可能由非确定性计算导致或者由不同层次中无害的性能不匹配导致。共识将通过章节5.9中讨论的*延迟函数*进行降速-具体上，每一个节点都会增加其*治理者*的数值ϵ来作为公证高度与认证高度生长间的间隙（这使用了章节5.12.2中提及的“本地化调整的延迟函数”的概

念）。

- *状态-特定的荷载有效性验证（State-specific payload validation）*。如章节5.7所述，一个负载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取决于该状态的某些程度。我们省略的一个细节是每个区块都包含一个轮次数，这些有效性检查应当在顾及该轮次数下的验证后状态后作出。一个需要执行验证的节点需要等待直至该轮次的状态被认证，然后使用该轮经认证的状态来进行验证。这么做确保了即使存在非确定性计算，所有的节点都在进行相同的有效性测试（否则共识可能会卡住）。

6.2 查询调用和更新调用

综上所述，一条入口消息必须通过共识以使他们可以被同一子网下的所有节点按相同顺序处理。但是对于这些不改变子网下节点中副本状态的入口消息存在一条重要优化。他们被称之为**查询调用**——相对于其他入口消息，被称为**更新调用**。查询调用被允许进行只读或是可能改变储罐状态的计算，但是任何对节点副本状态的更新都不会被提交给复制状态。正因如此，查询调用可以被单个节点副本直接处理而不需要经过共识，这极大的降低了从查询调用获得响应的延迟。

需要注意的是查询调用的相应是不会被记录在入口历史的数据结构中的。正因如此，我们无法直接使用每轮认证状态来验证查询调用的响应。但是，我们提供了一种单独的用于验证该类响应的机制：认证的变量。作为每轮认证状态中的一部分，每个子网下的储罐都被分配了一小段字节，这便是 *该储罐认证的变量*，它的数值可被更新调用更新也可被每轮认证状态机制来进行验证。另外，储罐也可使用其认证的变量来存储默克尔树的的根节点。以此为由，查询调用的相应可被验证，只要响应作为该储罐中以认证的变量为根的默克尔树的叶片即可。

6.3 外部用户验证

入口消息与跨子网消息的一个主要区别便是用于验证这些消息的机制。参见上文，我们已经知道了聚合签名如何被用于验证跨子网消息。NNS的注册表（见章节1.5）持有用于验证跨子网消息的阈值签名的公共验证钥匙。

IC中没有外部用户的中央注册表。相反，外部用户通过一串公钥哈希作为**用户标识**（又称*principal*）来向储罐来标识自己。用户自己持有对应的签名密钥，用来签署入口消息。签名和公钥会随着入口消息一起发送。IC将自动验证签名并传递用户标识给到对应的储罐。随后该储罐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。

新用户在首次与IC交互时会生成一对密钥对并从公钥中衍生出他们的用户标识。老用户根据存储在用户代理中的私钥完成验证。用户还可以用签名委托的方式，将多个密钥对关联到一个用户身份上。该特性非常有用，因为它允许了一个用户在多个设备上通过相同的用户身份证明来访问IC。

7 执行层

执行环境每次处理一条输入，该输入取自输入队列中的一个并被导向一个储罐。取决于该条输入以及储罐的状态，执行环境更新储罐的状态并可以额外添加消息至输出队列，并且更新入口历史（可能连同此前入口消息的响应）。 在给定轮次，执行环境将处理多条输入。**定时管理器（scheduler）**会判断哪些输入将在指定轮次被何种顺序下执行。我们这里不深入讨论过多scheduler的细节，而是突出强调一些目标：

- 它必须是*确定性的*，即仅仅取决于给定数据；
- 它应当*公平*的在储罐间分布工作量（但是对*吞吐*而不是*延迟*进行优化）。
- 每轮的完成的工作数量使用*cycles*（详见章节1.8）为计量单位，且应当接近一些前置判定的数量。

另一个执行环境必须处理的任务是该情形，即当某子网的一个储罐的生产跨子网消息的速度比其他子网能够消耗的速度更快。针对该情况，我们实现了一种自我监管的机制用于给生产储罐降速。

这里存在的很多需被运行环境处理的其他资源管理以及簿记任务，但是，全部的这些任务均须被确定性的处理。

7.1 随机磁带

每个子网都持有**分布式伪随机生成器（以下简称PRG）**的访问权限。如第3章提及的，随机字节可从一个由$(f+1)/n$构成的BLS签名衍生得出，此被称为**随机磁带**。对于共识协议的每一轮都有一个不同的随机字节。虽然此BLS签名与用于共识中使用的的随机信标的签名相似（见章节5.5），其机理却略有不同。 在共识协议中，一旦一个k块高的区块被最终确认，每个诚实节点将释放其在h+1块高下随机磁带中的片段。此处暗指两点：

1. 在块高h的区块被任何诚实节点副本最终确认前，块高h+1的随机磁带是被确保为不可预测的。
2. 在块高h的区块被任何诚实节点副本确认时，该节点副本一般将拥有所有所需的片段以构建块高$h+1$的随机磁带。

为获取随机字节，一个子网需要发起对这些字节的请求。此随机字节请求将会作为某轮次执行层的“系统请求”，且假定该轮次为h。当块高h+1的随机磁带可被获取时，系统之后则会响应该请求。基于上文的属性（1），我们可以确保在请求发起时，被请求的随机字节是不可预测的。基于上文的属性（2），被请求的随机字节一般将在下个区块被最终确认时可被获取。事实上，在当前的实现下，在块高h的区块被最终确认时，共识层（见章节5）将会同时递交块高h的区块（其荷载）以及h+1的随机磁带来给消息路由层处理。

8 链钥密码学II：链演进技术

如章节1.6.2所述，链钥密码学包括一系列复杂的技术，用于随时间推移健壮和安全地维护基于区块链的复制状态机，其合起来我们称之为**链演进技术**。每个子网在包含多轮（通常大约是几百轮）的**时期**（Epoch）内运行。链演技术实现了许多按epoch定期执行的基本维护工作：*垃圾回收，快速转发，子网成员变更，主动秘密转发和协议升级。*

链演进技术包含两个基本组成部分：**摘要块（summary blocks）**和**追赶包（catch-up packages，以下简称CUPs）**。

8.1 摘要块

每个epoch 的第一个区块是摘要块。摘要块包含特殊数据，用于管理不同阈值签名方案的密钥片段（详见第3章）。其中有两种阈值签名方案：

- 阈值结构为$f+1/n$的方案中，每个epoch生成新的签名密钥；
- 阈值结构为$n-f/n$的方案中，每个epoch重新共享一次签名密钥。

阈值低的方案用于*随机信标*和*随机磁带*，而阈值高的方案用于验证子网的复制状态。

回想一下，DKG协议（详见章节3.5）要求，对于每个签名密钥，有一个dealing的集合，而每个节点副本可以根据这组dealing，非交互式得获取它的签名密钥片段。

再回想一下，除了别的之外，NNS还维护着决定子网成员的**注册表**（详见章节1.5）。注册表（以及子网成员）会随时间改变。因此，子网必须对在不同时间和不同目的下，对使用的**注册表版本**达共识。这一信息也存储在摘要块中。

Epoch i的摘要块包括如下数据字段。

- $currentRegistryVersion$。*这个注册表版本将决定epoch i中的共识委员会（consensus committee）*——所有共识层的任务（生成区块，公证，最终确认）都由这个委员会执行。
- $nextRegistryVersion$。在每一轮共识中，区块生成者会将其知道的最新注册表版本（不得早于提议的区块的构建时间）包含在区块提案内。这确保了epoch i中的*nextRegistryVersion*是最新值。

  epoch i中的currentRegistryVersion值设置为epoch $i-1$中的nextRegistryVersion值。
- $currentDealingSets$。这个dealing集用于决定时期i中签名消息的阈值签名密钥。

  正如我们所看到的，epoch i中的阈值签名委员会是epoch $i-1$中的共识委员会。

- $nextDealingSets$。这个字段收集和存储epoch $i-1$中*收集到*的dealing[5]。epoch i中的$currentDealingSets$的值将被设置为epoch $i-1$中$nextDealingSets$的值（本身包含epoch $i-2$中收集到的dealing）。
- $collectDealingParams$。这个字段描述了时期i中需要*收集*的dealing集的参数。epoch i中，区块生成者会将经这些参数验证有效的dealing，放进提议的区块内。

  这些交易的*接受委员会*基于epoch i的摘要块的$nextRegistryVersion$。

  对于低阈值的签名方案，epoch i中的*交易委员会*是epoch i中的共识委员会。

  对于高阈值的签名方案，密钥片段的共享是基于时期i中的$nextDealingSets$。因此epoch i中的*交易委员会*是epoch $i-1$中的*接受委员会*，也是epoch i中的*共识委员会*。

  还要注意的是，epoch i中的阈值签名委员会是epoch $i-2$中的接受委员会，其是epoch $i-1$中的共识委员会。

epoch i中的共识协议依赖于epoch i中的$currentRegistryVersion$和$currentDealingSets$——具体来说，共识委员会本身基于*currentRegistryVersion*，共识中的随机信标基于$currentDealingSets$。此外，同其他区块一样，在epoch i开始的时候可能会有多个经过公证的摘要块，并且这种歧义需要在epoch i中经过共识来解决。这种看似循环的问题的解决办法是，保证epoch $i-1$开始时的摘要块，在epoch i开始前已经最终确认，因为新的摘要块的相关值，是直接从老的摘要块复制而来。这实际上是一个*隐含的同步假设*，但它也更是一个学术假设。事实上，因为章节5.12.2中讨论的确保活性的"共识降速（consensus throttling）"，也因为epoch 的长度很长，下面的情况本质上不可能发生：早在共识到达epoch $i-1$结束之前，epoch $i-1$的摘要块没有被最终确认，公证延迟函数将增长到非常之大，因此最终确认需要的部分同步假设（实际上）肯定会（基本）满足[6]。

8.2 CUPs

在阐述CUP之前，我们首先需要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。它不是CUP的基本特性，但是影响了CUP的设计。

CUP是一种特殊的（不在区块链上的）消息，它（基本上）拥有一个节点副本在不知道先前epoch 任何信息下，在epoch的起始点工作时所需的一切。它包含如下的数据字段：

- **整个**复制状态的默克尔哈希树的根（与章节1.6中的每轮验证的部分状态不同）。
- epoch的摘要块。
- epoch第一轮的随机信标。
- 子网对上述字段的$(n-f)/n$的阈值签名。

为生成给定epoch 的CUP，节点副本必须等到该epoch 的摘要块已经被最终确认，并且对应的每轮状态经过验证。如前所述，*整个*复制状态必须经哈希函数处理为一个默克尔树——尽管有很多技术用于加快这一过程，这个成本代价仍然非常大，这也是为什么每个epoch 仅处理一次。因为CUP仅包含这个默克尔树的根，因此我们使用了一个**状态同步**子协议，允许节点副本从对等节点中提取它所需要的任何状态——同样地，我们用了很多技术来加快这一过程，它的成本代价仍然很大。因为我们对CUP使用了高阈值签名，因此可以保证在任何epoch 只有一个有效的CUP，而且可以从很多对等节点中提取状态。

8.3 链演进技术实现

垃圾回收：因为CUP包含特定epoch 的信息，因此每个节点副本可以安全地清除该epoch 前所有已处理的输入，以及对这些输入排序的共识层消息。

快速转发：如果一个子网中的节点副本大幅落后于其同步节点（因为其宕机或是网络断连很长时间），或是一个新的节点副本被添加入子网，他们可以通过快速转发至最新epoch 的起始点，不需要运行共识协议并处理该点之前的所有输入。该节点副本可以通过获取最新CUP做到。利用从CUP中包含的摘要块和随机信标，以及来自其他节点副本的（还没有被清除的）共识消息，该节点副本可以从相应epoch 的起始点开始，向前运行共识协议。该节点也可以使用状态同步子协议来获取对应epoch 开始时的复制状态，这样它也可以开始处理共识层产生的输入。

图5描绘了快速转发。此处，我们假设需要一个需要追赶的节点副本处于epoch 起始点，（比方说）块高为101，有一个CUP。这个CUP包含了块高101的复制状态的默克尔树的根，块高101的摘要块和块高101的随机信标。该节点会使用状态同步子协议，从它的对等节点中获取块高101的所有复制状态，并用CUP中的默克尔树来验证此状态。在获取到该状态后，节点副本可以参与到协议之中，从对等节点中获取块高102，103等等的区块（以及其他和共识相关的消息），并更新其复制状态的副本。如果其对等节点已经确认了更高高度的区块，该节点副本将尽快处理（以及公证和最终确认）这些从对等节点获取的已最终确认区块（以执行层所允许的最快速度）。

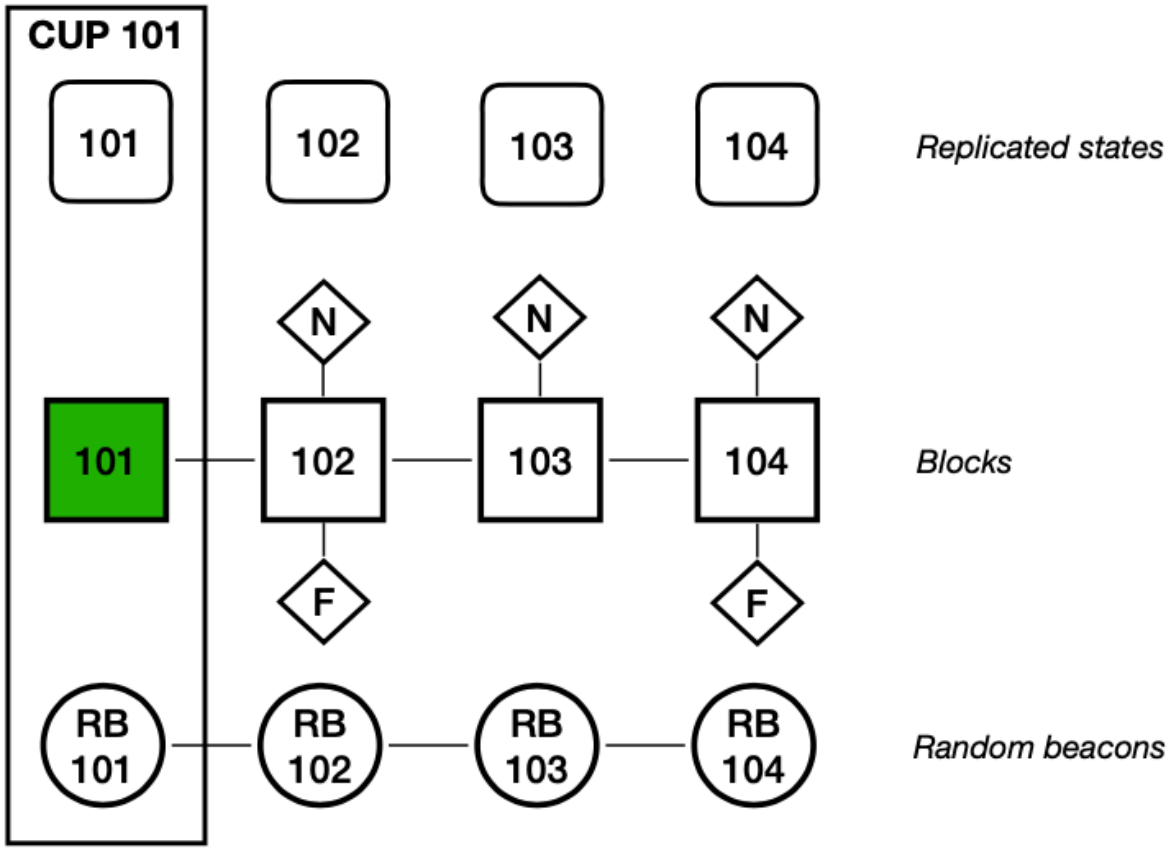

图5：快速转发

子网成员变更：我们已经讨论过特定epoch 内，如何使用摘要块来加密，使用哪个版本的注册表以及它如何决定子网成员，更具体来说，是各种任务的委员会成员。需要注意的是，即使一个节点副本从一个子网中移除，（如果可能的话）它应当多履行一个epoch 的分配到的委员会职责。

主动秘密再共享：我们已经讨论过了，如何使用摘要块生成和转发签名密钥。如有必要，需要的摘要块也可以从CUP中获取。

协议升级：CUP也用于协议升级。协议升级由NNS发起（详见章节1.5）。不考虑所有细节，基本细节如下：

- 当需要安装新版本的协议时，epoch 开始时的摘要块会做出指示；
- 所有运行老版本协议的节点副本，将继续运行共识协议直到最终确认摘要块并创建对应的CUP；然而，他们只会创建空区块，并不会将任何荷载传递给消息路由层和执行层；
- 安装新版本的协议后，运行新版本共识协议的节点副本，将从上述的CUP开始继续运行完整的协议。

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 Op- erating 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 Identifica- tion 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. Cryp- tology ePrint Archive, Report 2021/339, 2021. https://ia.cr/2021/339.

[JMV01] D. Johnson, A. Menezes, and S. A. Vanstone. The Elliptic Curve Digital Sig- nature Algorithm (ECDSA). Int. J. Inf. Sec., 1(1):36–63, 2001.

[Mer87] R. C. Merkle. A Digital Signature Based on a Conventional Encryption Func- tion. 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.

---

1. 详见https://webassembly.org/. ↩

2. 详见https://en.wikipedia.org/wiki/Actor_model. ↩

3. 详见https://en.wikipedia.org/wiki/Persistence_(computer_science)#Orthogonal_or_transparent_persistence. ↩

4. 详见 https://en.wikipedia.org/wiki/Transport_Layer_Security. ↩

5. 但是详见章节8.2 ↩

6. 另外需要注意的是，epoch i中收集的交易取决于epoch i的摘要块，具体来说，就是$nextDealingSets$和$nextRegistryVersion$。因此在epoch i的摘要块被最终确定之前，这些交易不应当生成且不能被验证。 ↩