Telegram Open Network （TON）

中文版白皮书 v1.0 （修订版）

作者：Dr. Nikolai Durov

译者：吴泰、lucas3s、Inkiy、CryptoRose、Erica

摘要

本文主要目的是介绍Telegram Open Network（TON）和相关区块链、点对点、分布式存储和托管服务技术。篇幅有限，我们将重点关注TON平台的独特性和定义性特质，这对实现TON的既定目标非常重要。

简介

Telegram Open Newtwork（TON）是一个快速、安全且可扩展的区块链和网络项目，在必要的情况下，它可达百万级别TPS[1]，对普通用户与服务供应商而言都是友好的。TON的目标是能够容纳当前已提出和构想中的所有应用。我们可以认为TON是一个巨大的分布式超级计算机，或者更确切地说是一个巨大的“超级服务器”，旨在托管和提供各种服务。本文不构成各种实施细节的最终参考文献。在开发和测试阶段， 一些细节可能会发生变化。

2019年6月15日

[1] TPS: Transactions Per Second，即服务器每秒处理的事务数

1 TON 概述

*Telegram Open Network（TON）*由以下几个组件组成：

- **灵活的多链平台**（*TON Blockchain*或称*TON区块链*；参见第 2 章），能够达到百万级别 TPS，带有图灵完备智能合约、可升级的正式区块链规范、多类型加密货币价值交换，支持小额支付通道和链下支付网络。TON区块链提出了一些新颖和独特的功能，例如 “自我修复”的垂直区块链机制（参见 2.1.17）和即时超立方体路由（Instant Hypercube Routing;参见 2.4.20），使其同时兼具快速、可靠、可扩展和一致性的特性。

- **点对点网络**（*TON P2P Network*，或称*TON Network*，亦称*TON网络*；参见第 3 章），不仅可以用于访问TON区块链、发送交易申请，以及接收用户感兴趣的区块链更新内容（例如与客户账户或智能合约相关的更新），也可以支持任意分布式服务，无论服务是否与区块链相关。

- **分布式文件存储技术**（*TON Storage*或称*TON 存储技术*；参见 4.1.8），可通过 TON 网络进行访问，被 TON 区块链用于存储区块和状态数据（快照）的存档副本，还可以通过类似流（Torrent）的访问技术，为用户储存任意文件，或是储存运行于平台的其他服务项目。

- **网络代理/ 匿名层**（*TON Proxy*或称*TON代理*；参见 4.1.11 和 3.1.6），类似于I^2P（Invisible Internet Project；隐形网络计划），用于在必要时隐藏*TON网络*节点的身份和IP地址（例如包含大量加密货币的账户进行交易所用的节点，或者高额质押的区块链验证人节点，后者希望隐藏其确切的 IP 地址和地理位置，防止 DDoS 攻击）。

- **类似Kademlia的分布式哈希表**（*TON DHT*；参见 3.2），用作TON存储的 “流追踪器”（torrent tracker；参见 3.2.10）或者用作*TON 代理*的 “输入通道定位器”（input tunnel locator；参见 3.2.14 ），并作为 *TON 服务*的服务定位器（参见 3.2.12）。

- **支持任意服务的平台**（*TON Services*或称*TON服务*；参见第 4 章），内置于*TON 网络*和 *TON 代理*，并通过 *TON 网络*和 *TON 代理*访问，带有形式化接口 （参见 4.3.14 ），驱动着类似浏览器或智能手机应用程序之间交互。这些形式化接口和永久性服务入口点可以发布在TON区块链上（参见 4.3.17）；从信息发布在TON区块链上开始，实际节点在任意时刻提供的服务都可以通过*TON DHT*查找（参见 3.2.12）。服务可以通过在TON区块链中创建智能合约的方式向客户进行保证（参见 4.1.7）。

- **TON DNS**（参见 4.3.1），一种服务，用于为帐户、智能合约、服务和网络节点分配可读性强的名称。

- **TON Payments**（参见第 5 章），小额支付、小额支付通道和小额支付通道网络的平台。它可用于快速的链下价值交换，也可用于支付由 TON Services 支持的服务。

- TON 将允许与第三方的通信和社交网络应用进行简易集成，从而使区块链技术和分布式服务普及到普通用户（参见 4.3.24），而不仅仅供少数早期的加密货币持有者使用。我们将在另一个项目Telegram Messenger中提供这种集成的示例（参见 4.3.19）。

虽然 TON Blockchain是TON Project的核心，而其他组件可能被视为TON Blockchain的支持角色，但它们本身就具有有用且有趣的功能。结合起来，它们允许平台承载更多样化的应用程序，而不仅仅靠TON Blockchain来实现（参见 2.9.13 和 4.1）。

2 TON Blockchain

我们首先介绍Telegram Open Network（TON）区块链，这是项目的核心组成部分。白皮书打算“自上而下”：首先给出整体项目的描述，然后提供每个组件的更多细节。
为简单起见，我们在这里谈论TON Blockchain，尽管原则上该区块链协议的几个实例可以独立运行。

2.1 TON Blockchain作为2-Blockchains的集合及定义

TON Blockchain实际上是区块链的集合（甚至是“区块链中的区块链”的集合，或者说是“二维区块链”（*2-blockchains*）——这一概念将在后面的 2.1.17 中阐明），因为没有哪个单一的区块链项目可以实现百万 TPS 交易吞吐量，对比而言，单链区块链仅能达到每秒数十次的交易速度。

2.1.1 TON 区块链类型列表。包含以下这几类区块链：

- 独特的*master blockchain*，或简称主链（*masterchain*），用来存储一些基础信息，这些信息包括协议、参数的当前价值；验证人和它们的抵押记录组；当前活跃的工作链（workchains）和它们的 “分片”（shards）组；及最重要的是所有工作链（workchains）和分片链（shardchains）里的最近生成的哈希值组。

  - 多个working blockchains（最多可达2^{32}），或简称工作链（*workchains*），是名副其实的“苦力”，功能包含价值交换和智能合约。不同的工作链有不同的“规则”，意味着不同的帐户地址格式、不同的交易格式，不同的虚拟机（VMs）用于智能合约，甚至是不同的工作链由不同的加密货币来处理交易等。但它们都必须满足统一的互操作性标准，以便使不同工作链之间的交互成为可能且相对简单。在这点而言，TON Blockchain是*异构*的（参见 2.8.8），类似于 EOS（参见 2.9.7）和 Polkadot（参见 2.9.8）。

  - 每个工作链（workchains）又依次被细分为最多2^{60}次方个分片链（*shard blockchains*；或简称为*shardchains*），分片链与工作链的生成规则和区块结构相同，一个分片链负责一种类型的帐户子集，具体取决于其帐户地址的头几个字节（这些是最重要的）。换句话说，分片的形式植入系统中的（参见 2.8.12）。因为所有

这些分片链具有统一的生成规则和区块结构，这样说来所以 TON 区块链是*同构*的（参见 2.8.8），这也与之前以太坊的的一个扩容方案[2]中提及的类似。

- 分片链上的每个区块（其实主链上也是）不仅仅只是一个区块，它可能是一个小型区块链。通常情况下，这种 “区块的区块链” 或者 “垂直区块链” 都只包含一个块，然后我们可能会认为这只是分片链的相应区块（在这种情况下也称为 “水平区块链”）。但是，如果有必要修复不正确的分片链区块，则会将新区块提交到 “垂直区块链” 中，其中包含无效 “水平区块链” 区块的替换品或者 “区块的不同性”，包含此区块先前版本部分内容的仅有描述也需要修改。这是一种特定于TON的机制，用于替换检测到的无效区块，而不会涉及所有分片链的分叉问题；它将在 2.1.17 中更详细地解释。现在，我们只是说每个分片链（和主链）不是传统的区块链，而是*链中链*或*区块链中的区块链*，或者只是一个*二维区块链*（2D-blockchain或2-blockchain）。

2.1.2 无限分片范式（Infinite Sharding Paradigm）。

几乎所有的区块链分片提议都是 “自上而下” 的：首先我们想象一个单链区块链，然后讨论如何将其拆分成几个交互的分片链，以提高性能并实现可扩展性。
TON 的分片方法是 “自下而上”，解释如下。

想象一下，分片已经发挥到极限，因此每个分片链中只保留一个帐户或智能合约。然后我们有大量的 “账户链（account- chains）”，每个账户链描述唯一一个账户的状态和状态过渡，并相互发送价值承载信息以传输价值和信息。

当然，想要拥有数以亿计的区块链是不切实际的，并且每个区块链通常很少出现更新（即新区块）。为了更有效地实现它们，我们将这些 “账户链（account- chains）” 分组为 “分片链（shardchains）”，以便分片链的每个区块基本上都是已分配给此分片的帐户链区块的集合。因此，“帐户链” 在 “分片链” 中仅以纯粹的虚拟或逻辑形式存在。

我们将这种观点称为*无限分片范式*。它解释了TON区块链的许多设计初衷。

2.1.3 消息/即时超立方体路由。

即时超立方体路由（Instant Hypercube Routing）。无限分片范式（Sharding Paradigm）引导我们将每个帐户（或智能合约）视为自己的分片链。那么，A帐户可能影响B帐户状态的唯一方法就是向它发送一条消息（这是所谓的Actor模型的特殊实例，带有Actors账户；参见 2.4.2）。因此，帐户之间的消息系统（和分片链，因为从一般意义上说来源和目标帐户通常在不同的分片链中）对于可扩展系统至关重要，如TON区块链。实际上，TON区块链的这个新功能，我们称其为即时超立方体路由（Instant Hypercube Routing；参见 2.4.20），使其能够将一个分片链区块中创建的消息传递和处理至目标分片链的下一个区块中，而不用管系统中分片链的总数。

2.1.4 主链、工作链和分片链的数量。

TON区块链只有一个主链。但该系统最多可容纳2^{32}个工作链，每个工作链最多可细分成2^{60}个分片链。

[2] https://github.com/ethereum/wiki/wiki/Sharding-FAQ

2.1.5 工作链可以是虚拟区块链，而不是真正的区块链。

因为工作链通常被细分为分片链，所以工作链的存在是“虚拟的”，而不是真正的区块链，这意味着它不是后文2.2.1所述的一般意义上的真正区块链，而只是一个分片链的集合。当只有一个分片链对应一个工作链时，这个唯一的分片链可能可以用工作链来识别，在这种情况下，工作链成为“真正的”区块链，至少在一段时间内，它与常规的单链区块链的设计有一定相似性。然而无限分片范式（参见 2.1.2）告诉我们，这种相似性只是表面：重要的是潜在大量的“账户链（account- chains）”可以暂时归类到一个区块链中。

2.1.6 工作链的识别。

每个工作链都有序号或工作链的专属标识符（*workchain_id*：uint_{32}），这是一个无符号的32位整数。工作链由主链中的特殊交易创建，定义（先前未使用过的）工作链的标识符和工作链的正式描述，至少足以使这个工作链与其他工作链交互，并对这个工作链的区块进行表面验证。

2.1.7 创建和激活新的工作链。

基本上任何社区成员都可以创建新的工作链，只需支付发布新工作链所需的（高）主链交易费用，这些费用是用于发布新工作链的正式规范。但为了使新的工作链更积极，它需要得到三分之二验证人的共识，因为他们也需要升级软件来处理新工作链的区块，并通过主链的特殊交易来表明它们已经准好用新的工作链了。对激活新工作链感兴趣的一方可能会为验证人提供一些激励，通过智能合约分发的一些奖励来支持新的工作链。

2.1.8 鉴定分片链。

每个 shardchain 由一对（*w,s*）=（*workchain_id,shard_prefix*）标识，其中 *workchain_id*：uint_{32}标识相应的工作链，*shard_prefix*：$2^{0\ldots60}$是一个长度最多为2的60次方的字符串，来定义此分片链负责的帐户子集。即，所有具有以 *shard_prefix* 开头的 *account_id* 的帐户（即 *shard_prefix* 作为最高有效位）将被分配给该分片链。

2.1.9 识别帐户链。

回想一下，“账户链（*account- chains*）”只是虚拟存在的（参见2.1.2）。但是，它们有一个自然的标识符——即（*workchain_id*，*account_id*），因为任何帐户链里都包含有关状态信息和准确的某一账户的更新信息（无论是简单帐户还是智能合约，区别并不重要）。

2.1.10 分片链的动态拆分和合并（参照2.7）。

不那么复杂的系统可以使用*静态分片*——例如，通过使用 *account_id* 的前八位来选择 256 个预定义分片中的一个。

TON区块链的一个重要特征是它实现了*动态分片*，这意味着分片数量不固定。相反，如果满足某些条件（实际上，如果原始分片上交易负荷负载在很长一段时间内足够高的话），则分片 *shard (w, s)* 可以自动细分为分片*shards (w, s.0)* 和*(w, s.1)* 。反过来如果负载在一段时间内都很低，则分片*shards (w, s.0)* 和*(w, s.1)* 可以自动合并并退回*shard (w, s)* 。

最初，只为工作链 *w* 创建了一个分片*shard (w, ∅)* 。之后，如果有必要，它会细分为更多分片（参见2.7.6和2.7.8）。

2.1.11 基础工作链或初始工作链（Workchain Zero）。

虽然使用特定的规则和交易可以定义多达2^{32}条工作链，但我们最初只定义了一条工作链，*workchain_id = 0*。这个称为基础工作链或初始工作链（Workchain Zero），它是用于处理TON智能合约和转移*TON coins*的工作链，也称为*Grams*（参照附录a）。大多数应用程序只需要初始工作链。基础工作链的分片链将被称为基础分片链（*basic shardchains*）。

2.1.12 出块时间。

我们期望每五秒在每个分片链和主链上生成一个新区块。这可以将交易确认时间缩到非常短。所有分片链的新区块几乎同时生成；而主链将在大约一秒后生成一个区块，因为它包含所有分片链的最新区块的哈希值。

2.1.13 通过主链使工作链和分片链紧密耦合。

一旦分片链生成的某个区块的哈希值被合并到主链的某个区块中，这个分片链区块和它的父块都会被认为是 “经典的”，这意味着之后所有分片链出的区块都可以引用它们，因为它们是固定且不可更改的。实际上，每个新的分片链区块都包含最新主链区块的哈希值，并且引用自主链区块的所有分片链区块被视为新区块，且不可变。

也就是说，这意味着在分片连区块中提交的交易或消息可以安全用于其他分片链的下一系列区块中，且无需等待。比如基于前一笔交易，你在发送消息或采取其他行动之前，需要系统的 20 个确认（即在同一区块链中原始区块之后生成的 20 个区块），这在大多数 “松散耦合” 系统（参见 2.8.14）里是常见的，例如 EOS。但是在我们的 “紧密耦合” 系统中，你提交后仅仅5秒就后能在其他分片链中发送交易和消息，这种能力是我们相信它的原因，这也是第一个能提供这种前所未有性能的系统（参见 2.8.12 和 2.8.14）。

2.1.14主链（Masterchain）区块的哈希作为一个全局状态（global state）。

根据 2.1.13，整体来看，最后一个主链区块的哈希值确定了系统的全局状态。 而无需分别监视其他 shardchains 的状态。

2.1.15 验证人（Validators）生成新区块；参照 2.6。

TON区块链使用 Proof-of-Stake（PoS）在分片链和主链中生成新区块。这意味着会存在几百个*验证人*——那些已经进行存币（大量的TON coins）质押的特殊节点，通过一个特殊的主链交易才有资格成为验证人并生成新区块。

然后，以确定的伪随机方式将小量子集验证人分配给每个分片shard *(w; s)*，大约每 1024 个区块改变一次。这些子集验证人通过从客户端收集合适的拟议中交易给新的有效候选区块，会提出建议并就下一个分片区块的内容达成共识。对于每个区块，验证人都存在伪随机选择的顺序，以确定每一轮中谁的候选区块享有最高出块优先级。

验证人和其他节点会检验被提名的候选区块的有效性；如果验证人签署了无效性，则会受到自动惩罚，丢失部分或全部的质押代币，或者在一段时间内不许继续做验证人。在此之后，验证人应该就下一个区块的选择达成共识，主要通过 BFT共识协议（拜占庭容错；参见 2.8.4）的有效变体，类似于pBFT【4】或者Honey Badger BFT【11】。如果达成共识，则创建新区块，验证人会瓜分该块的交易费用和新造代币（挖矿）奖励。

每个验证人可以选择参与几个验证人子集；这种情况下，所有验证和共识算法都会同时运行。在所有新的分片链生成区块或出块超时后，一个主链区块将会生成，它包括所有分片链最新区块的哈希值。这是由验证人通过BFT共识完成的。[3]

有关TON PoS的方法及其经济模型的更多详细信息，请参见第 2.6 节。

2.1.16 主链（masterchain）分叉。

我们采用紧密耦合导致的复杂性是，转换到主链的一个不同分叉几乎必然要求在一些分片链上转换到另一分叉。另一方面，只要主链中没有分叉，分片链就不可能分叉，因为分片链区块的哈希值都是合并到主链区块中，分片链的备用分叉中没有区块能够成为“经典”。

一般规则是，*如果主链区块B'是区块B的上一个区块，则区块B'包含分片链(w; s)的区块$B'_{w;s}$的哈希值Hash ($B'_{w;s}$)，且区块B包括哈希值Hash ($B_{w;s}$)，那么区块$B'_{w;s}$也是区块$B_{w;s}$的上一个区块。否则，主链区块B无效。*

我们预计主链几乎不会分叉，因为TON区块链采用的BFT共识下，只有在大多数验证人行为不正确的情况下才可能发生这种情况（参见2.6.1和2.6.15），这将意味着违法者会遭受重大损失。因此可以说，分片链中不会有真正的分叉。相反，如果检测到无效的分片链，它会通过2-blockchain的“垂直区块链”机制（参见2.1.17）进行纠正就可以实现这一目标，而无需分叉“水平区块链”（horizontal blockchain，即分片链）。同样的机制也可用于修复主链区块中的非致命错误。

2.1.17 更正无效的分片链区块。

通常情况下，只有有效的分片链区块才能出块，因为分配到分片链的验证人必须有三分之二达到拜占庭共识，新区块才能出块。然而，系统必须先允许检测之前已经出的无效区块并校正。

当然，一旦找到无效的分片链区块——由验证人（不一定是负责这个分片链的）或 “渔夫”（fisherman）；系统内的一种节点，可以存入一定代币然后提出有关区块有效性的问题；参见2.6.4）——当无效性的声明及其证据被提交到主链时，签署区块无效性的验证人将收到惩罚——失去部分质押和/或暂时从验证人阵营中退出（后一种措施面对攻击者窃取其他良性验证人的签名私钥这种情况时很重要）。

然而这还不够，因为系统（TON Blockchain）的整体状态由于先前释放的分片链区块无效而被证明是无效的。必须使用较新的有效区块替换此无效块。

大多数系统都会 “回退” 至分片链无效区块的前一个区块来实现此目的，再之前的其他区块不受其他每个分片链中的无效区块传播的消息影响。并从这些区块开始分叉。这种方法的缺点是许多其他正确的和已提交的交易突然回退，并且不清楚这些回退的交易是否稍后将被包括在内。

TON区块链通过使每个分片链和主链（即“水平区块链”[horizontal blockchains]）的每个 “区块” 自身成为一个小区块链（即“垂直区块链”[vertical blockchain]）来解决这个问题，包含不同

[3] 实际上，三分之二的验证人就足以达成共识，但尽可能多的签名是更好的。

版本的 “区块” 或它们的 “差异””。通常，垂直区块链只包含一个区块，而分片链看起来像一个经典的区块链。然而，一旦确认块的无效性并将其提交到主链区块中，则允许无效区块的 “垂直区块链” 在垂直方向上由新区块增长，从而替换或编辑无效区块。新区块由相关分片链的当前验证人子集来生成。

新的 “垂直” 区块规则的有效性非常严格。特别是如果无效区块中包含着一个虚拟的 “帐户链区块”（参见 2.1.2）本身有效，则必须保持新的垂直块不变。

一旦在无效区块之上提交了一个新的“垂直”区块，其哈希值就会发布在新的主链区块中（或者说在一个新的“垂直”区块中，该区块位于原始主链区块上方，原始主链区块是最初用来发布无效的分片链区块哈希值的地方），这些改变进一步扩散到任何的分片链区块中，这些分片链区块引用该无效区块的先前版本（例如，那些已从不正确的区块接收消息的区块）。这一问题的解决方式是通过在垂直区块链中，为以前引用 “不正确” 区块的所有区块提交新的 “垂直” 区块；新的垂直区块将引用最新（已更正）的版本。同样，严格的规则禁止更改未受影响的帐户链（例如依旧像先前版本那样照常接收信息）。通过这种方式，修复不正确的区块会产生 “连带效应”，最终传播到所有受影响的分片链的最新区块；这些变化反映在新的 “垂直”主链区块也是如此。

一旦 “历史重写” 的波及最新的区块，新的分片链区块仅在一个版本中生成，仅作为最新区块版本的接续。这意味着它们将从一开始就包含对正确（最近）垂直区块的引用。

主链状态隐含地定义了将每个 “垂直” 区块链的第一个区块的哈希值转换为其最新版本哈希值的映射。这使客户能够通过第一个（通常是唯一的）区块哈希值来验证和定位任何 “垂直区块链” 。

2.1.18 TON coins和多货币工作链（multi-currency workchains）。

TON区块链最多支持2^{32}种不同的“加密货币（cryptocurrencies）”、“币（coins）”, 或者 “代币（tokens）”，由 32 个字节的*currency_id*作为区分。可以通过主链中的特殊交易添加新的加密货币。每个工作链都有一个基础的加密货币，并且可以有几个额外的加密货币。

有一种特殊的加密货币，*currency_id = 0*，叫作*TON coin*，也被称为*Gram*（参见 附录A）。它是初始工作链（Workchain Zero）的基础加密货币。它还用作交易费和验证人的抵押资产。

原则上，其他工作链可能会收取其他代币作为交易费用。这种情况下，应该提供一些智能合约，将这些交易费用自动转换为Grams。

2.1.19 消息的传递与价值转移。

属于相同或不同工作链的分片链可以相互发送消息。虽然允许的消息格式取决于接收工作链和接收帐户（智能合约），但有一些共同的字段让工作链间消息传递成为可能。特别是每个事件可以附加一些价值，价值的形式是一定数量的*Grams*（TON coin）和/或其他已注册的加密货币，倘若这些已注册的加密货币是被接收工作链声明为可接受的加密货币。
这种消息传递的最简单形式是从一个（通常不是智能合约）账户到另一个账户的价值转移。

2.1.20 TON虚拟机。

TON虚拟机（也称为*TON VM*或*TVM*）是用于在主链和基础工作链中执行智能合约代码的虚拟机。其他工作链可以同时使用其他虚拟机或者用其他虚拟机代替TVM。
在这里我们列出了它的一些功能。它们将在 2.3.12、2.3.14 中进一步讨论。

- TVM将所有数据表示为（TVM）单元（cell）的集合（参见 2.3.14）。每个单元最多包含128字节的数据，最多包含 4 次对其他单元的引用。由于 “一切都是一组单元” 的理念（参见 2.5.14），这使得TVM能够处理与TON区块链相关的所有数据，如有必要，包括区块和区块链全局状态。
- TVM可以处理任意代数数据类型的值（参见 2.3.12），以TVM细胞里的Merkle树或DAG为代表。但它只对单元有效，对代数数据类型未知。
- TVM内置了对哈希图的支持（参见 2.3.7）。
- TVM是一台堆栈机器。它的堆栈保持着64 位整数或对单元的引用。
- 支持64位、128位和256位算术。所有n位算术运算有三种形式：无符号整数、有符号整数和模数2^n整数（后一种情况下不进行自动溢出检测）。
- TVM具有从n位到m位的无符号和有符号整数转换，对于所有0≤m，n≤256，带有溢出检查。
- 默认情况下，所有算术运算都执行溢出检查，大大简化了智能合约的开发。
- TVM具有 “先乘法后位移运算” 和“先位移后除法运算” 的算术运算方式，其中间值以较大的整数类型计算；这简化了实现定点算术的过程。
- TVM支持二进制字符串和字节字符串。
- 支持对一些预定义曲线（包括Curve25519）的256位椭圆曲线加密算法（ECC）。
- 还存在对某些椭圆曲线上的Weil配对的支持，这有利于快速实现zk-SNARK。
- 支持流行的哈希函数，包括SHA256。
- TVM可以使用Merkle证明（参见 5.1.9）。
- TVM为 “大型” 或 “全局” 的智能合约提供支持。这种智能合约必须意识到分片技术（参见 2.3.18 和 2.3.16）。而本地智能合约不用知道是否分片。
- TVM支持闭包。
- 可在TVM内轻松实现STG（spineless tagless G-machine）【13】。

除了 “TVM组件” 之外，还可以为TVM设计几种高级语言。所有这些语言都将具有静态类型，并将支持代数数据类型。我们设想了以下可能性：

- 类似 Java 的命令式语言，每个智能合约类似于一个单独的类别。
- 一种惰性的函数式语言（设想一下Haskell）。
- 热切的功能语言（设想一下ML）。

2.1.21 可配置的参数。

TON区块链的一个重要特征是它的许多参数都是可配置的。这意味着它们是主链状态的一部分，并且可以通过主链中的某些特殊提案/投票/结果交易记录进行更改，而无需任何硬分叉。改变这些参数需要收集三分之二的验证人投票，以及所有其他参与投票过程中支持该提案的参与者的一半以上的投票。

2.2 区块链概述

2.2.1 一般区块链定义。

一般来说，任何（真）区块链都是一个区块序列，每个区块*B*包含一个引用blk-prev(*B*)，该引用是对对上一个区块（通常是通过将上一个区块的哈希值包含到当前区块的区块头）和交易列表的引用。每笔交易都描述了*全局区块链状态*的转型；在一个区块中列出的交易有序地被引用

来计算新状态，这个新状态是从旧状态开始的，新状态也是前一个区块评测之后出来的结果状态。

2.2.2 TON区块链的相关性。

回想一下，TON区块链不是真正的区块链，而是二维区块链的集合（即区块链的区块链；参见 2.1.1），因此上述内容并不直接适用于它。但我们从真正区块链的一般性开始，将它们用作我们构建更复杂结构的区块。

2.2.3 区块链实例和区块链类型。

人们经常使用*区块链*这个词来表示一般的*区块链类型*及其特定的*区块链实例*，它们被定义为满足某些条件的区块序列。例如，2.2.1指是区块链实例。

以这种方式，区块链类型通常是类型$Block^*$的“子类型”，类型$Block^*$是区块的列表（即，有限序列），由满足某些兼容性和有效性条件的那些区块序列组成：

$$Blockchain \subset Block^* \tag{1}$$

定义区块链的更好方法是说区块链是一个*依赖成对类型（dependent couple type）*，由一对值$(\mathbf{B},v)$组成，第一个组成部分$\mathbf{B}:Block^*$是类型$Block^*$（例如区块的列表），第二个组成部分$v:isValidBc(\mathbf{B})$是$\mathbf{B}$的有效性的证明或见证。表达式是这样，

$$Blockchain \equiv \Sigma_{(\mathbf{B}:Block^*)}\ isValidBc(\mathbf{B}) \tag{2}$$

我们在这里使用的表示法是依赖型求和类型，参考【16】。

2.2.3 依赖类型理论（Dependent type theory），交互式的定理证明器（Coq）和类型语言（TL）。

请注意，我们在这里使用（Martin-Löf）依赖类型理论，类似于Coq[4]证明助手中使用的理论。依赖类型理论的简化版本也用于*类型语言（TL）*[5]，将在TON区块链的正式规范中使用它来描述所有数据结构的序列化结构以及块、交易等的排列方式。

实际上，依赖类型理论（Dependent type theory）给出了一种证明的可用形式，这种形式证明（或者是它们的序列），会在比如需要为某些区块提供无效证明时变得很方便。

2.2.5 类型语言（TL / Type Language）。

由于类型语言（TL）将用于TON区块、交易和网络数据报的正式规范中，因此有必要进行简短的讨论。

类型语言是一种适用于描述依赖型代数类型的语言，允许使用数字（自然）和类型参数。通过几个*构造函数*描述每种类型。每个构造函数都有一个（人类可读的）标识符和一个*名称*，它是一个位字符串（默认情况下为32位整数）。除此之外，构造函数的定义包含字段列表及其类型。

[4] https://coq.inria.fr
[5] https://core.telegram.org/mtproto/TL

构造函数和类型定义的集合称为*TL-scheme*。它通常保存在一个或多个带有后缀 .tl 的文件中。

TL-schemes的一个重要特征是它们确定了一种明确的对已定义的代数类型值（或对象）进行序列化和反序列化的方法。也就是说，当需要将值序列化为字节流时，首先用于此函数值的构造函数的名称会被序列化。递归计算每个字段的序列化如下。

对TL的先前版本的描述，适用于将任意对象序列化为32位整数序列，这可以参考：https://core.Teleggram.org/MTProto/TL。一种TL的新版本TL-B正在研发，用于描述对象的序列化，这个新版本被TON项目采用。这个新版本可以将对象序列化为字节流甚至比特流（不仅仅是32位整数），并且支持将序列化应用到TVM单元树中（参见 2.3.14）。TL-B的描述将是TON区块链的正式规范的一部分。

2.2.6 区块和交易作为状态转换运算符。

通常，任何类型的区块链（类型）*Blockchain*都具有关联的全局状态（类型）*State* 和交易（类型）*Transaction*。 区块链的语义在很大程度上取决于交易应用程序的功能：

$$ev\_trans' : Transaction \times State \to State^{?} \tag{3}$$

这里$X^{?}$表示MAYBE X，是应用MAYBE monad到类型X的结果。这与我们使用X^{*}表示列表X类似。实际上，类型$X^{?}$的值要么是类型X的值，要么表示没有实际值（设想一下null pointer）的特殊值$\perp$。在我们的例子中，我们使用$State^{?}$而不是$State$作为结果类型，因为如果从某些原始状态请求（试想试图从一个帐户取出比账户中实际更多的钱），交易可能无效。

我们可能更喜欢*ev_trans'*的改进版本：

$$ev\_trans : Transaction \to State \to State^{?} \tag{4}$$

因为一个区块本质上是一个交易列表，区块评价函数

$$ev\_block : Block \to State \to State^{?} \tag{5}$$

可以从*ev_trans*派生。它需要一个区块$B : Block$和前一个区块链状态$s : State$（可能包括前一个区块的哈希值）并计算下一个区块链状态$s' = ev_block(B)(s) : State$，它要么是一个真实状态或者是特殊值$\perp$，表示无法计算下一个状态（即如果从给定的起始状态进行评估，则该块无效——例如，该区块包括一笔试图从空账户收款的交易）。

2.2.7 区块序列号（Block sequence numbers）。

区块链中的每个区块*B*可以通过序列号BLK-SEQNO(*B*) 来引用，从第一个区块的0开始，并且每当传递到下一个块时递增1。正式的公式：

$$\text{BLK-SEQNO}(B) = \text{BLK-SEQNO}(\text{ BLK - PREV}(B)\ ) + 1 \tag{6}$$

请注意，在分叉出现时，序列号不是用来唯一识别一个区块的。

2.2.8 区块哈希值（Block hashes）。

引用区块*B*的另一种方式是通过其哈希值BLK-SEQNO(*B*)，实际上是区块*B*区块头的哈希值（然而，区块的区块头通常包含区块*B*的所有内容的哈希值）。假设所使用的哈希函数没有冲突（或者至少它们非常不可能产生冲突），则通过其哈希值可以唯一地标识区块。

2.2.9 哈希假设（Hash assumption）。

在区块链算法的形式分析期间，我们假设使用的k-bit的哈希函数 $HASH : Bytes^* \to 2^k$没有冲突：

$$\text{Hash}(s) = \text{Hash}(s') \Rightarrow s = s' \quad \text{对任意} s,\ s' \in Bytes^* \tag{7}$$

这里 $Bytes = \{0...255\} = 2^8$次方的字节类型，或者是所有字节值的集合，Bytes*是任意（有限）字节列表的类型或集合；而2 = {0,1}是位类型，2^k次方是所有k-bit的序列（即k-bit的编号）的集合（或实际类型）。

当然，（7）在数学上是不可能的，因为从无限集到有限集的映射不能是单射的。一个更严谨的假设是：

$$\forall\ s,s' : s \neq s',\ P(\text{Hash}(s) = \text{Hash}(s')) = 2^{-k} \tag{8}$$

然而，这对于证明来说并不方便。对于某些小值的 ε（例如，$\varepsilon=10^{-18}$），如果（8）在$2^{-k}N < \varepsilon$的证明中最多使用 N 次，倘若我们接受失败概率 ε（即最终结论将会为真的概率至少是1-ε），我们可以推论（7）似乎为真。

最后评论：为了使（8）的概率陈述真正严谨，必须在所有字节序列的集合*Bytes** 中引入概率分布。这样做的一种方法是假设所有相同长度***l***的字节序列是等概率的，并且设置观察长度***l***的序列的概率等于$p^l - p^{l+1}$（对于某些$p \to 1$-）。然后当*p*从更小值趋近于1时，（8）应该被理解为条件概率$P(\text{Hash}(s) = \text{Hash}(s') | s \neq s')$的一种极限状态。

2.2.10 哈希作用于TON区块链。

我们暂时在TON区块链中使用256位的SHA256哈希函数。如果结果比预期的要差，那么将来可以用另一个哈希函数来替换它。哈希函数的选择是协议的可配置参数，因此可以在不进行硬分叉的情况下更改，如 **2.1.21** 中所述。

2.3 区块链状态、账户和哈希表

我们在上文中已经注意到，全局状态由每一个区块链来决定，每个区块和每个交易都定义了全局状态的变化。这里我们描述TON区块链使用的全局状态。

2.3.1 帐户 ID。

TON 区块链使用的基础帐户ID——或者至少是其主链和初始工作链使用的基础帐户ID，是 256 位整数，对于一个特定的椭圆曲线，基础账户ID假定为256位椭圆曲线加密（ECC）的公钥。用这种方式，

$$account\_id : Account = uint_{256} = 2^{256} \tag{9}$$

此处Account是*帐户类型*，而 account_id是帐户类型的特定变量。
其他工作链可以使用其他帐户ID格式，256位或其他。 例如，可以使用比特币类型的帐户ID，等于ECC公钥的SHA256。
但是，在创建工作链（在主链上）期间，帐户ID的位长度***l***必须是固定的，并且它必须至少为64位，因为account_id的前64位是用于分片和按路由发送消息的。

2.3.2 主要组成部分：*Hashmaps*。

TON区块链状态的主要组成部分是*hashmap*。在某些情况下，我们考虑（部分定义）“map”h：2^n ---> 2^m。或者通俗地说，我们可能对哈希图 h 更感兴趣：$h : 2^n$ --->X 表示复合型X。但是源（或索引）类型几乎总是2^n 。
有时，我们的“默认值”为空值：X, 哈希图为 $h : \to X$， “初始化”状态是“默认值” i |→ 空值。

2.3.3 示例：TON帐户余额。

TON账户余额给出了一个重要的例子：

$$balance : Account \to uint_{128} \tag{10}$$

就是这个 hashmap将Account = 2^{256} 映射至类型 $uint_{128} = 2^{128}$ 的Gram (TON coin)余额。该hashmap的默认值为零，这意味着最初（在处理第一个块之前）所有帐户的余额为零。

2.3.4 示例：智能合约的永久存储。

另一个例子是智能合约永久存储，它可以（非常近似地）表示为一个hashmap：

$$storage : 2^{256} \text{ ---> } 2^{256} \tag{11}$$

该hashmap的默认值也为零，意味着假设未初始化的永久存储单元个数为零。

2.3.5 示例：永久存储所有的智能合约。

因为我们有多个智能合约，由account_id来区分，每个都要独立的永久存储，所以必须有一个hashmap

$$Storage : Account \text{ ---> } (2^{256} \text{ ---> } 2^{256}) \tag{12}$$

将智能合约的account_id映射到其永久存储里。

2.3.6 hashmap的类型。

Hashmap不仅仅是一个抽象的（部分定义的）函数2^n ---> X； 它有一个特定的表示法。 因此，我们假设我们有一个特殊的hashmap类型

$$Hashmap(n, X) : Type \tag{13}$$

对应于编码（部分）图2^n ---> X的数据结构。我们也可以写为

$$Hashmap(n : nat)(X : Type) : Type \tag{14}$$

或者

$$Hashmap : nat \to Type \to Type \qquad (15)$$

我们也可以将$h : Hashmap(n, X)$转换成一个map：$hget(h) : 2^n \to X^?$。之后，我们通常写$h[i]$来替代 hget(h)(i)：

$$h[i] :\equiv hget\ (h)(i) : X^? \quad 对于任意\ i : 2^n,\ h : Hashmap\ (n, X) \qquad (16)$$

2.3.7 把hashmap的类型当作Patricia树。

从逻辑上讲，可以将$Hashmap\ (n,\ X)$定义为深度为n的（不完整）的二叉树（binary tree），边缘标签为0和1，叶片中有类型X的值。描述相同结构的另一种方式是（按位）前缀树（*trie*），用于长度等于n的二进制字符串。

在实践中，我们更喜欢使用此前缀树的压缩表示法，通过压缩每个顶点的父节点只带有一个子节点。结果表示为Patricia树或者二进制基数树。每个中间顶点现在有两个子节点，由两个非空二进制字符串标记，左边子节点为0，右边的子节点为1。

换句话说，Patricia树中有两种类型的（非根）节点：

- 叶子LEAF(x), 包含类型X的值x；
- 节点NODE (l,s_l,r,s_r)，其中l是对左边子节点或子树（的引用），s_l是用于标记连接此顶点到左边子节点的边缘的位字符串（总是从0开始），r是右边的子树，而s_r是位字符串用于右边子节点的边缘（总是以1开头）。

第三种类型的节点，仅在Patricia树的根部使用一次，也是必要的：

- 根ROOT(n,s_0,t)，其中n是$Hashmap\ (n,\ X)$的索引位字符串的公共长度，s_0是所有索引位字符串的公共前缀，t是对Leaf或Node的引用。

如果我们想让Patricia树为空值，将使用第四种类型的（根）节点：

- 空根EmptyRoot(n)，其中n是所有索引位串的公共长度。

我们定义 Patricia 树的高度

$$HEIGHT\ (LEAF\ (x)) = 0 \qquad (17)$$

$$HEIGHT\ (NODE\ (l,s_l,r,s_r)) = HEIGHT\ (l) + LEN\ (s_l) = HEIGHT\ (r) + LEN\ (s_r) \qquad (18)$$

$$HEIGHT\ (ROOT\ (n,s_0,t)) = LEN\ (s_0) + HEIGHT\ (t) = n \qquad (19)$$

最后两个公式中每个公式中的最后两个表达式必须相等。我们使用高度为n的Patricia树来表示$Hashmap\ (n\ ,\ X)$类型的值。

如果树中有N叶子（即我们的hashmap包含N个值），则恰好有N-1个中间顶点。插入新值总是涉及通过在中间插入新顶点并添加新分支作为此新顶点的另一个分支来分割现有边界。从哈希表中删除值则相反：删除了一个叶子及其上级，则上级的上级与其他叶子直接连接。

2.3.8 Merkle-Patricia树（MPT树）。

使用区块链时，我们希望能够比较Patricia树（即哈希表）及其子树，方法是将它们减少为单个哈希值。Merkle树给出了实现这一目标的经典方法。本质上，假使是我们知道如何计算对象$x : X$的哈希HASH(x)（例如，通过将哈希函数 Hash应用于对象x的二进制序列化），借助为

二进制字符串定义的哈希函数Hash，这是我们想要描述一种类型哈希图 Hashmap(*n*，*X*)的对象*h*的一种方法。
可以递归地定义HASH (h)，如下所示：

$$\text{HASH (LEAF } (x)) = \text{HASH } (x) \quad (20)$$

$$\text{HASH (NODE } (l,s_l,r,s_r)) = \text{HASH (HASH } (l).\ \text{HASH } (r).\ \text{CODE } (s_l).\ \text{CODE } (s_r)) \quad (21)$$

$$\text{HASH (ROOT } (n,s_0,t)) = \text{HASH (CODE } (n).\ \text{CODE } (s_0).\ \text{HASH } (t)) \quad (22)$$

这里*s.t*表示（位）字符串s和t的关系，并且代码CODE(*s*)是所有位字符串s的前置码。例如，可以用10编码0，用11编码1，在字符串结尾用0编码。[6]
稍后我们将看到（参见2.3.12和2.3.14）这是递归定义的哈希函数的一个（稍微调整过的）版本，用于任意（从属）代数类型的值。

2.3.9 重新计算Merkle树的哈希值。

这种递归定义Hash (h)的方式称为*Merkle树哈希*，其优点是，如果一个人明确地将Hash (*h'*）与每个节点*h'*一起存储（产生一个称为*Merkle树的结构*，或者在我们的例子中，叫作*Merkle-Patricia树*），在hashmap中添加、删除或更改一个元素时，只需要重新计算最多n个哈希值。

用这种方法，如果通过合适的Merkle树哈希表示全局区块链状态，则在每次交易后很容易重新计算此状态哈希值。

2.3.10 Merkle证明。

所选哈希函数Hash在“内射”（injectivity）的假设（7）下，我们可以构造一个证明，对于Hash (h)的给定值*z*，h：Hashmap (n，X)，有一个*hget (h)(i) = x*，对于某些 $i: 2^n$ 和 *x:X*。这样的证明将包括 Merkle-Patricia树中的路径，这种路径从对应于i到叶子到根，由出现在路径上的所有兄弟节点的哈希值来增强。

以这种方式，一些仅知道哈希图h的Hash (h)值的轻节点[7]（例如，智能合约永久存储或全局区块链状态）可以从完整节点[8]请求，请求的内容不仅是值 *x = h [i] = hget (h) (i)*，而且是伴随着从已知值 Hash (*h*)开始的Merkle证明。然后，在假设（7）下，轻节点可以检查自身，x确实是h [*i*]的正确值。

而在一些情况下，客户端可能想要获得值 *y* = Hash（*x*）= Hash (h [*i*])——例如，如果 x 本身非常大（例如，hashmap本身）。然后可以提供(*i , y*)的Merkle证明。如果*x*也是哈希函数，则从 *y* = Hash (*x*)开始的第二个Merkle证明可以从完整节点获取，以提供值 x[*j*] = h [*i*] [*j*] 或仅其哈希值。

---

[6] 可以表明这种编码对于带有随机或连续索引的Patricia树的几乎一半边缘标签是最佳的。剩余的边缘标签可能很长（即几乎256位长度）。因此，边缘标签的近似最佳编码是使用上面的代码，该代码用前缀为0表示“短”位串字符串，并编码1，剩下9位包含位字符串*s*的长度*l* = |*s*|，然后用*s*的*l*表示“长”位字符串（$l \geq 10$）

[7] 轻节点不追踪分片链的完整状态;相反，它保留最少的信息，例如最近几个块的哈希值，并且当需要检查完整状态的某些部分时，依赖于从完整节点获得的信息。

[8] 完整节点用于追踪被提及的分片链的完整最新状态。

2.3.11 对于诸如TON的多链系统，Merkle证明的重要性。

请注意，节点通常不能是存在于TON环境中所有分片链的完整节点。它通常是一种针对一些分片链的全节点——例如，那些包含自己帐户的分片链，对智能合约感兴趣的分片链，或者这个节点被指定为验证人的那些分片链。对于其他的分片链，它必须是一个轻节点——否则存储、计算和网络带宽要求会非常高。这意味着这样的节点不能直接检查关于其他分片链状态的断言（assertion）；对于这些分片链，它必须依赖于从全节点获得的Merkle 证明，这与自身检查一样安全，除非(7)失败（即发现哈希冲突）。

2.3.12 TON VM的特点。

用于在主链和初始工作链中运行智能合约的TON VM或TVM（Telegram虚拟机）与受EVM（以太坊虚拟机）启发的传统设计有很大不同：它不仅用256位整数操作，而且实际上（几乎是）带有任意 “记录”、“结构” 或 “和产品类型”，使其更适合执行用高级（特别是函数编程）语言编写的代码。 从本质上讲，TVM使用标记数据类型，与实现Prolog或Erlang时使用的数据类型还不同。

人们首先可能想到的是，TVM智能合约的状态不仅仅是一个hashmap $2^{256} \to 2^{256}$，或者 *Hashmap* (256 , 2^{256})，但是（作为第一步）*Hashmap* (256 , X)，其中X是具有多个构造函数的类型，使其能够存储除256位整数之外的其他结构的数据，包括其他hashmap，特别是 *Hashmap* (256 , X)。这意味着TVM（永久或临时）存储的单元——或 TVM 智能合约代码中的变量或数组元素——不仅可以包含整数，而且可以包含全新的哈希表。当然，这意味着一个单元不仅包含256位，而且还包含一个8位标签，描述了如何解析这个256位。

实际上，值不需要精确为 256 位。TVM 使用的值的格式包括一系列原始字节和对其他结构的引用，以任意顺序混合，其中一些描述符字节插入适当的位置，以便能够区分指针与原始数据（例如，字符串或整数）；参照 2.3.14。

该原始值格式可用于实现任意和积（SPN）代数类型。在这种情况下，该值首先包含一个原始字节描述正在使用的 “构造函数”（从高级语言的角度来看），然后是其他 “字段” 或 “构造函数参数”，其中包括原始字节和对其他结构的引用，这些结构取决于所选择的构造函数（参见 2.2.5）。然而，TVM对构造函数与其参数之间的对应关系一无所知；字节和引用的混合由某些描述符字节明确描述。[9]

Merkle树哈希扩展到任意这样的结构：为了计算这种结构的哈希，所有引用都提及对象的哈希值递归替换，然后计算得到的结果字节字符串（包括描述符字节）的哈希。

通过这种方式，2.3.8中描述的针对Hashmap的Merkle树哈希，只是对任意（依赖型）代数数据类型进行哈希的特殊情况，应用于具有两个构造函数的*Hashmap (n，X)*类型。[10]

2.3.13 永久存储TON智能合约。

[9] 存在于任何TVM细胞中的这两个描述符字节仅描述引用和原始字节的总数；参考文献在所有原始字节之前或者之后一起保存。

[10] 实际上，Leaf和Node是辅助类型*HashmapAux (n，X)*的构造函数。类型*Hashmap (n，X)*带有构造函数Root和EmptyRoot，其中Root包含类型*HashmapAux (n，X)*的一个值。

对于一个TON智能合约的永久存储本质上由它的 “全局变量”组成，这些“ 全局变量”保存在智能合约的调用之间。因此，它只是一个 “产品”、“元组” 或 “记录” 类型，由正确类型的字段组成，每个字段对应一个全局变量。如果全局变量太多，由于对TON单元体积的全局限制，它们则不能归属于一个TON单元。在这种情况下，它们被分成若干记录并组织成树，本质上成为了 “产品的产品” 或 “产品的产品的产品” 类型，而不仅仅是产品类型。

2.3.14 TVM单元。

最终，TON VM将所有数据保存在（TVM）单元的集合（Collection）中。每个单元首先包含两个描述符字节，指示该单元中存在多少字节的原始数据（最多 128 个），以及存在多少个对其他单元的引用（最多四个）。然后是这些原始数据字节和引用。每个单元只被引用一次，因此我们可能已经在每个单元中包含对其 “父”（引用该单元的唯一单元）的引用。但是，该引用不必明确说明。

以这种方式，TON智能合约的永久数据存储单元被组织成树[11]，其中智能合约描述中保存了对该树的根的引用。如有必要，整个永久存储器的Merkle树哈希值被递归计算了，从树叶开始，然后简单地用被引用单元的递归计算的哈希值代替一个单元中的所有引用，并随后计算由此获得的字节字符串的哈希值。

2.3.15 任意代数类型值的广义Merkle证明。

因为TON VM通过由（TVM）单元组成的树来表示任意代数类型的值，并且每个单元具有明确定义的（递归计算的）Merkle哈希，实际上取决于根据该单元格生成的整个子树，所以我们可以为任意代数类型的（部分）值提供 “广义Merkle 证明”，旨在证明具有已知Merkle哈希的树的某个子树采用特定值或具有特定哈希的值。这概括了 2.3.10的方法，其中只考虑了x[*i*] = *y*的Merkle证明。

2.3.16 支持TON VM数据结构中的分片。

我们刚刚概述了TON VM如何在不过度复杂的情况下支持高级智能合约语言中的任意（依赖型）代数数据类型。但是，大型（或全局）智能合约的分片需要在TON VM层面提供特殊支持。为此，系统中添加了特殊版本的hashmap类型，相当于一个 “map” *Account* ---> *X*。这个 “map” 似乎等同于*Hashmap (m, X)*，其中*Account* = 2^m。但是，当分片一分为二或两个分片合并时，这些哈希图会自动拆分成两个或合并回来，目的是为了保持只有那些属于相应分片的钥。

2.3.17 为永久存储支付。

TON区块链的一个值得注意的特征是从智能合约中提取的支付的用于存储其永久数据的支付（即，用于扩大区块链的总状态）。它的工作原理如下：

每个区块有两种费率，以区块链的主要货币（通常是 Gram）指定：一种是将一个单元保留在永久存储中的价格，另一种是在永久存储的某个单元中保留一个原始字节的价格。每个帐户使用的细胞和字节总数的统计信息存储为其状态的一部分，因此通过将这些数字乘以区块头中的

[11] 从逻辑上讲， 2.5.5中描述的“bag of cells””表述确认所有重复单元，在序列化时将此树转换为有向无环图（dag）。

两种费率，我们可以计算要从帐户余额中扣除的付款以便在前一个区块和当前区块之间保留数据。

但是，为使用永久存储而付款，并不强求每个区块中的每个账户和智能合约；相反，最后要求该支付的区块序列号存储在帐户数据中，当该帐户进行任何动作时（例如，通过智能合约接收和处理价值转移或消息），在执行任何进一步动作之前，会从帐户余额中扣除从上一次此类付款以来的所有区块的存储使用费用。如果此后帐户的余额将变为负数，则该帐户将被销毁。工作链可以声明每个帐户的一些原始数据字节是 “免费” 的（即，不参与永久性存储支付），以便制作 “简单” 帐户，这些帐户仅用一种或两种加密货币保留余额，而不用不断付款。

请注意，如果没有人向帐户发送任何消息，则不会收集其永久存储费用，并且账户可以一直存在。但是，任何人都可以发送一条空消息来销毁这样的帐户。想要销毁帐户余额的部分作为小激励，会提供给这样的消息的发送者。我们预计验证人将乐意销毁此类资不抵债的账户，只是为了减少全局区块链的规模，避免无偿地保留大量数据。

为保持永久性数据而收集的付款分布在分片链或主链的验证人之间（后一种情况下与他们的质押成比例）。

2.3.18 本地和全局智能合约；智能合约实例。

智能合约通常只存在于一个分片中，根据智能合约的*account_id*选择，类似于 “普通” 账户。对于大多数应用来说，这通常就足够了。然而，一些 “高负荷” 的智能合约可能希望在某些工作链的每个分片链中都有一个 “实例”。为了实现这一点，他们必须将他们的创建的交易传播到所有分片链中，例如，通过将此交易提交到工作链的 “root”分片链(w，∅)[12]并支付大额佣金。[13]此操作在每个分片中有效地创建了智能合约的实例，并具有单独的余额。最初，在创建交易中传输的余额简单地通过在shard (*w*，*s*)中给出的实例$2^{-|s|}$作为总余额的一部分来分布。当分片分成两个子分片时，所有全局智能合约实例的余额分成两半；当两个分片合并时，余额会加在一起。

在某些情况下，拆分/合并全局智能合约的实例可能涉及（延迟）执行这些智能合约的特殊方法。默认情况下，拆分和合并余额如上所述，并且一些特殊的 “帐户索引” hashmap也会自动拆分和合并（参见 2.3.16）。

2.3.19 限制智能合约的分割。

全局智能合约可能会在创建时限制其拆分深度*d*，以便使永久存储费用更具可预测性。这意味着，如果shardchain (w，s)（其中$|s| \geq d$）分为两部分，两个新的分片链中只有一个接替了智能合约的一个实例。这种分片链的选择是确定性的：每个全局智能合约都有一些 “account*_id”*，它本质上是创建的交易的哈希值，并且其实例具有相同的*account_id*，其中第一个≤d的位替换为落入正确分片所需的适当值。此*account_id*选择在拆分后由哪个分片接替智能合约实例。

2.3.20 账户 / 智能合约状态。

我们可以总结以上所有内容，得出结论：帐户或智能合约状态包含以下内容：

- 区块链主要货币的余额

[12] 一种更昂贵的替代方案，目的是在主链中发布这种“全局”智能合约。
[13] 这是所有分片的一种“广播”功能，因此它必然是非常昂贵的。

- 区块链的其他货币余额
- 智能合约代码（或其哈希）
- 智能合约永久性数据（或其Merkle哈希）
- 永久存储单元数和使用的原始字节数的数据统计
- 最后一次收集智能合约永久存储付款（实际上是主链区块编号）
- 从此账户传输货币和发送消息所需的公钥（可选；默认情况下等于*account_id*本身）。在某些情况下，可以在此处找到更复杂的用于检查代码的签名，类似于比特币交易输出所做的操作； 然后*account_id* 将等同于此代码的哈希值。

我们还需要在帐户状态或其他一些帐户索引的哈希图中保留以下数据：

- 帐户的输出消息队列（参见 2.4.17）
- 最近发送的消息（哈希）的集合（参见 2.4.23）

并非所有这些都是每个帐户真正需要的；例如，智能合约代码仅适用于智能合约，但不适用于“简单” 账户。 此外，虽然任何账户的主要货币的余额必须非零（例如，基础工作链的主链和分片链的Grams），但其他货币的余额可能为零。为了避免保留未使用的数据，（在工作链的创建期间）我们定义了一个sum-product类型（取决于工作链），它使用不同的标记字节（例如，TL 构造函数；参见 2.2.5）来区分不同的 “ 构造函数 “使用。最终，帐户状态本身被保存为TVM 永久存储的单元集合。

2.4 分片链之间的通信

TON 的一个重要组成部分是区块链之间的消息传递系统。包括相同工作链的下的分片链和不同工作链下的分片链。

2.4.1 消息，帐户和交易：系统的一种鸟瞰视角下的。

消息从一个帐户发送到另一个帐户。 一个交易包含：一个接收一条消息的帐户、这个账户根据某些规则更改其状态、以及向其他帐户生成多个（可能是一个或零个）新消息。 每条消息仅会被生成并被接收（传递）一次。

这意味着消息在系统中起着至关重要的作用，与帐户（智能合约）相当。 从无限分片范式（参见 2.1.2）的角度来看，每个帐户都在其独立的 “帐户链” 中，并且它影响其他帐户状态的唯一方法是发送消息。

2.4.2 作为流程或参与者的帐户；Actor 模型。

有人可能会将帐户（和智能合约）视为 “processes” 或 “actors”，它们能够处理传入的消息，更改其内部状态并生成一些出站消息。 这与所谓的 Actor 模型密切相关，在 Erlang 等语言中使用（然而，Erlang 中的 actor 通常称为 “processes”）。 由于处理入站消息的结果也允许现有参与者创建新的actors（即，智能合约），因此与 Actor 模型的对应基本上是完整的。

2.4.3 消息接收者。

任何消息都有其接收者，其特征是目标工作链标识符 w（默认情况下假定与发起的分片链一致）和收件方帐户 account_id。 *account_id* 的确切格式（即字节数字）取决于 w； 但是，分片始终由其首位（最重要的）64 字节确定。

2.4.4 消息发送者。

在大多数情况下，消息具有发送者，再次由（w'，account_id'）对表示。它位于消息接收者和消息值之后。 有时，发送者不重要或者是在区块链系统之外（即，并非一个智能合约），在这种情况下，该字段不存在。
请注意，Actor 模型不要求消息具有隐式发送者。 相反，消息可以包含对应该发送请求答案的 Actor 的引用； 通常它与发送者一致。 但是，在加密货币（拜占庭）环境中的消息中具有不可伪造发送者签名字段是有用的。

2.4.5 消息值。

消息的另一个重要特征是其附加值，它在由源（source）和目标工作链支持的一个或多个代币中。 消息的值紧接着消息接收者之后显示； 它本质上是（currency_id，value）对的列表。 请注意，“简单（simple）” 帐户之间的 “简单” 值转移只是空（无操作）消息，并附加了一些值。 另一方面，稍微复杂的消息体可能包含简单的文本或二进制注释（例如，关于支付的目的）。

2.4.6 外部消息，或 “来源不明的消息（messages from nowhere）”。

一些“来源不明的消息”会进入系统——也就是说，它们不是由位于区块链中的账户（智能合约或非智能合约）生成的。一个典型的例子是当用户想要将一些资金从她控制的账户转移到其他账户时，这种情况就会发生。在这种情况下，用户将一条“来源不明的消息”发送到她自己的帐户，请求它生成带有指定值的接收帐户的消息。如果此消息已正确签名，则她的帐户会收到该消息并生成所需的出站消息。

实际上，人们可能会将一个 “简单” 帐户视为具有预定义代码的智能合约的特例。这个智能合约只收到一种消息。这样的入站消息必须包含由于传递（处理）入站消息而生成的出站消息列表以及签名。智能合约检查签名，如果正确，则生成所需的消息。

当然，“来源不明的消息”与正常信息之间存在差异，因为“来源不明的消息”不能承担附加值，因此它们不能自行支付它们的 “Gas”（即它们的processing）。相反，在被包含在一个新的分片区块之前，它们在有一个小额gas限制的条件下执行；如果执行失败（签名不正确），则外部消息被认为是不正确的并被丢弃。如果在执行有小额gas限制的进程时没有失败，则消息可以被包含在新的分片区块中进行完全处理，并从从接收者的帐户中支取（处理任务所需要的）Gas。“来源不明的消息”还可以定义一些除了用于重新分配给验证人的gas费用之外的交易费用，它们是从接收方的账户中被抵扣。

从这个意义上说，外部消息是来自其他区块链系统（例如，比特币和以太坊）的交易候选人。

2.4.7 日志（Log）消息，或 “去向不定的消息（messages to nowhere）”。

类似地，有时可以生成特殊消息并将其导到特定的分片链，而不是将其传递给其接收者。为了易于任何对接收到的分片更新有疑问的人可观察，这些消息被记录下来。这些记录的消息可以在用户的控制台中输出，或者在离线服务器上触发某些脚本的执行。 从这个意义上说，它们代表了 “区块链超级计算机” 的外部 “输出”，就像 外部消息代表 “区块链超级计算机” 的外部 “输入” 一样。

2.4.8 与链下服务和外部区块链互动。

这些外部输入和输出消息可用于与链下服务和其他（外部）区块链（如比特币或以太坊）进行交互。 有人可能会在 TON 区块链内部创建代币或加密货币，并与比特币、以太坊或以太坊区

块链中定义的任何 ERC-20 代币挂钩，并使用由脚本生成和处理驻留在某些第三方链下服务器上的外部消息和 “去向不定的消息”，以实现 TON区块链与这些外部区块链之间的交互。

2.4.9 消息正文。

消息正文只是一个字节序列，其含义仅由接收工作链和（或）智能合约的确定。 对于使用 TON VM 的区块链，这可以是通过 Send()操作自动生成的任何 TVM cell 的串行化。 简单地通过用所引用的 cell 递归地替换 TON VM cell 中的所有引用来获得这种序列化。 最终，出现一串原始字节，通常由 4 字节 “消息类型” 或 “消息构造函数” 预先设置，用于选择接收智能合约。
另一种选择是使用 TL 序列化对象（参见 2.2.5）作为消息体。 这对于不同工作链之间的通信尤其有用，也许某个工作链不使用 TON VM。

2.4.10 Gas 限制和其他工作链 / VM 特定参数。

有时，消息需要携带有关gas限制、gas价格、交易费用以及依赖于接收工作链的类似值的信息，并且仅与接收工作链相关，但对于原始工作链而言并非必要。 这些参数包含在消息体中或之前，有时（取决于工作链）具有特殊的 4 字节前缀，表明它们的存在（可以通过 TL 方案定义；参见 2.2.5）。

2.4.11 创建消息：智能合约和交易。

这两个新消息的来源。 大多数消息是在智能合约执行期间（通过 TON VM 中的 Send()操作）创建的，此时调用某个智能合约来处理传入消息。消息也可能来自外部，作为 “外部消息” 或 “去向不定的消息”（参见 2.4.6）[14]

2.4.12 传递信息。

当消息到达包含其目标帐户的分片链时，它将 “传递（delivered）” 到其目标帐户[15]。 接下来会发生什么取决于工作链； 从外部的角度来看，重要的是这样的消息永远不能从这个分片链进一步转发。

对于基础工作链的分片链，交付包括将附带数值（减去任何 gas 支付）添加到接收帐户的余额，如果收款账户是智能合约则根据消息的规则来调用。实际上，智能合约只有一个用于处理所有传入消息的入口点，它必须通过查看它们的前几个字节来区分不同类型的消息（例如，包含 TL 构造函数的前四个字节；参见 2.2.5 ）。

2.4.13 消息的传递是一种交易。

因为消息的传递改变了帐户或智能合约的状态，所以它是接收分片链中的特殊“交易”。忽略细节的话，基本上所有 TON 区块链交易都是向接收账户（智能合约）发送一个入站消息。

2.4.14 同一智能合约实例之间的消息。

回想一下，智能合约可能是本地的（即，像任何普通账户一样驻留在一个分片链中）或全局的（即，在所有分片中具有实例，或者至少在所有分片中具有某个已知深度 d；参见 2.3.18）。

[14] 只有基本工作链及其分片链才能真正实现上述要求。 其他工作链可能提供其他创建消息的方法。
[15] 作为一个退化的案例，这个分片链可能与原始的分片链重合 - 例如，如果我们在一个尚未拆分的工作链内工作。

如果需要，全局智能合约的实例可以交换特殊消息以在彼此之间传递信息和价值。 在这种情况下，（不可伪造的）发件人 account_id 变得很重要（参见 2.4.4）。

2.4.15 发送给智能合约任何实例的消息；通配符地址。

有时，消息（例如，客户端请求）需要被传递到全局智能合约的所有实例，通常是最接近的（例如，如果存在与发送者相同的分片链中的一个，则它是明显的候选者）。 一种方法是使用 “通配符接收者地址wildcard recipient address”，允许目标 account_id 的前 d 位采用任意值。 实际上，通常会将这些 d 位设置为与发送方的 *account_id* 中相同的值。

2.4.16 输入队列不存在。

区块链（通常是分片链；有时是主链）接收的所有消息——或者基本上由驻留在某个 分片链内的 “帐户链account-chain”——立即传递（即，由接收帐户处理）。 因此，没有 “输入队列input queue”。 相反，如果由于区块总大小和 Gas 使用的限制，并非所有发往特定分片链的消息都可以被处理，一些消息只会在原始分片链的输出队列中堆积。

2.4.17 输出队列。

从无限分片范例（参见 2.1.2）的角度来看，每个账户链（即每个账户）都有自己的输出队列，包括它已生成但尚未发送给接收者的所有消息。 当然，账户链只有虚拟存在； 它们被分组为不同的分片链，而分片链有一个输出 “队列queue”，它由属于分片链的所有帐户的输出队列的并集组成。
此分片链输出 “队列（queue）” 仅对其成员消息强加部分顺序。 也就是说，必须在后续块中生成的任何消息之前提供在前一个区块中生成的消息，并且必须按照它们生成的顺序传送由同一帐户生成并具有相同目的地的任何消息。

2.4.18 可靠，快速的链间消息传递。

对于像 TON 这样的可扩展多区块链项目来说，能够在不同的分片链之间转发和传递消息（参见 2.1.3）至关重要。即使系统中有数百万消息在传递，它们也可以被可靠且快速地传送（即，消息不应丢失或传送不止一次）。 TON 区块链通过结合使用两个 “消息路由（message routing）” 机制实现了这一目标。

2.4.19 超立方体路由（Hypercube routing ）：确保消息被传递的 “慢速路径 slow path”。

TON 区块链使用 “超立方体路由（Hypercube routing）” 作为一种缓慢但安全可靠的方式，将消息从一个分片链传递到另一个分片链，如有必要，可使用多个中间分片链进行传输。否则，任何给定的分片链的验证人都需要追踪所有其他分片链的状态（输出队列），这将需要大量的算力和网络带宽，因为分片链的总量增长，反而限制了系统的可扩展性。因此，无法直接从任何分片向其他分片传递消息。相反，每个分片仅 “连接（connected）” 到不同于其(*w*,*s*)分片标识符的一个十六进制数字的分片（参见 2.1.8）。这样，所有的分片链都构成了一个 “超立方体（hypercube）” 图形，并且消息沿着这个超立方体的边缘传播。

如果将消息发送到与当前分片不同的分片，则当前分片标识符的一个十六进制数字（决定性选择）将被目标分片的相应数字替换，并且所得到的标识符将通过消息发给最近的目标。[16]

超立方体路由的主要优点是区块有效性条件意味着创建分片链区块的验证人必须收集并处理来自 “相邻（neighboring）” 分片链的输出队列的消息，以免丢失它们的抵押。 通过这种方式，可以预期任何消息迟早会到达其最终目的地； 消息也不能在传输过程中丢失或接受两次。

请注意，超立方体路由引入了一些额外的延迟和费用，因为必须通过几个中间的分片链转发消息。 然而，这些中间分片链的数量增长非常缓慢，如对数链 N 的总数的对数 log N（更确切的说 [log16 N]-1）。例如，如果 N≈250，则最多只有一个中间跳； 对于 N≈4000 个 分片链，最多两个。 通过四个中间跃点，我们可以支持多达一百万个分片链。 我们认为这对于系统的基本无限可扩展性来说是一个非常小的代价。

2.4.20 即时超立方体路由：消息的 “快速路径（fast path）”。

TON 区块链的一个新特点是它引入了一条 “快速路径”，用于将消息从一个分片链转发到任何其他分片链，允许在大多数情况下完全绕过 2.4.19 的 “慢（slow）” 超立方体路由，并将消息传递到最后一个目的地分片链的下一个区块。

这个想法如下：在 “慢slow” 超立方体路由期间，消息沿着超立方体的边缘（在网络中）传播，但是在每个中间顶点处被延迟（大约五秒）以在继续其航行之前被提交到相应的分片链中。

为了避免不必要的延迟，可以使用沿超立方体边缘的合适的 Merkle 证明来中继消息，而无需等待将其提交到中间的分片链中。实际上，网络消息应该从原始分片的 “任务组（task group）”（参见 2.6.8）的验证人转发到目的地 “任务组” 的指定分片区块生成器（参见 2.6.9）。这可以直接完成，而不需要沿着超立方体的边缘。当带有 Merkle 证明的消息到达目的地分片链的验证人（更确切地说，校对人[collators]；参见 2.6.5）时，它们可以立即将其提交到新的区块中，而无需等待消息完成其沿着 “慢速路径（slow path）”。然后沿着超立方体边缘发送回传送确认以及合适的 Merkle 证据，并且可以通过提交特殊交易来用于沿着 “慢速路径” 停止消息的行进。

请注意，这种 “即时交付（instant delivery）” 机制并不能取代 2.4.19 中描述的 “慢速”但防止故障的机制。因为验证人不会因丢失或仅仅决定不将 “快速路径” 消息提交到其区块链的新区块而受到惩罚， “慢速路径”仍被需要[17]。

因此，两种消息转发方法并行运行，只有在将 “快速” 机制的成功证明提交到中间分片链时才会中止 “慢速” 机制。[18]

2.4.21 从相邻分片链的输出队列收集输入消息。

当提出用于分片链的新区块时，邻近的一些输出消息（如 2.4.19 的路由分支所示）分片链作为 “输入（input）” 消息包含在新块中并立即传送（即处理）。 处理这些临近的输出消息的顺序，会存在一些规则。 基本上来说，必须在任何 “较新（newer）” 消息之前，先传递 “较旧（

[16]这不一定是用于计算超立方体路由的下一跳的算法的最终版本。 特别地，十六进制数字可以由r比特组代替，其中r是可配置参数，不一定等于4。

[17] 但是，验证人有一些动机尽快这样做，因为他们将能够收集与慢速路径上尚未消耗的消息相关的所有转发费用。

[18] 事实上，人们可能暂时或永久地完全禁用“即时交付”机制，系统将继续工作，尽管速度较慢。

older）”消息（来自指向较旧主链区块的分片链的区块）；对于来自相同相邻分片链的消息，必须遵守 2.4.17 中描述的输出队列的部分顺序。

2.4.22 从输出队列中删除消息。

一旦观察到输出队列消息已经被相邻的分片链传递，它就会被特殊交易从输出队列中显式删除。

2.4.23 防止双重传递消息。

为了防止从相邻分片链的输出队列中获取的消息的传输两次，每个分片链（更确切地说，其中的每个帐户链）将最近传递的消息（或仅仅是它们的哈希）的集合保存为其状态的一部分。当观察到传递的消息由其始发的相邻分片链从输出队列中删除时（参见 2.4.22），它也会从最近传递的消息的集合中删除。

2.4.24 转发用于其他分片链的消息。

超立方体路由（参见 2.4.19）里有时出站消息不会传递到包含预期接收者的分片链，而是传递到位于到目的地的超立方体路径上的相邻分片链。这种情况下，“传递（delivery）”包括将入站消息移动到出站队列。这在区块中具体显示为包含消息本身的特殊转发事件。从本质上讲，这看起来好像是分片链中的某个人收到了消息，并且结果生成了一条相同的消息。

2.4.25 为转发和保留消息支付。

转发交易实际上花费了一些 gas（取决于转发的消息的大小），因此将从代表该分片链的验证人转发的消息的数值中扣除 gas费用。这种转发支付通常远小于当消息最终传递给其接收者时所支付的 gas 支付，即使该消息由于超立方体路由而被多次转发。此外，只要消息保存在某个分片链的输出队列中，它就是分片链全局状态的一部分，因此特殊交易也可以收集长时间保存全局数据的支付。

2.4.26 来自主链的消息。

消息可以直接从任何分片链发送到主链，反之亦然。但是，在主链中发送消息和处理消息的 gas 价格非常高，因此只有在真正需要时才会使用此功能——例如，验证人可以存入他们的抵押中。在一些情况下，可以定义发送到主链的消息的最小抵押（附加值），仅当消息被接收方视为 “有效（valid）” 时才返回。

消息无法通过主链自动路由。workchain*id = 1* 的消息（1 是表示主链的特殊 *workchain*id）无法传递给主链。

原则上，可以在主链内创建消息转发智能合约，但使用它的价格会很高。

2.4.27 同一个分片链中的帐户之间的消息。

在某些情况下，消息由属于某个分片链的帐户生成，发往同一个分片链中的另一个帐户。例如，这发生在一个尚未拆分为多个分片链的新工作链中，因为负载可控。

此类消息可能会累积在分片链的输出队列中，然后作为后续块中的传入消息进行处理（为此目的，任何分片都被视为临近分片）。但是，在大多数情况下，可以在原始区块链自身之内传递这些消息。

为了实现这一点，将对一个分片链区块中包括的所有交易强加上一个偏序，并且根据这个偏序来处理这些交易（每个交易包括向某个账户传递消息）。特别是，一项交易可以处理关于这一偏序的一些输出消息。

在这种情况下，消息正文不会被复制两次。相反，始发和处理交易会参考消息的副本。

2.5 分片链的全局状态。“Bag of Cells” 思想

现在我们准备描述TON区块链的全局状态，或者基础工作链的分片链。我们从 “高级（high-level）” 或 “逻辑（logical）” 描述开始，其中包括全局状态是代数类型 *ShardchainState* 的值。

2.5.1分片链状态作为帐户链状态的集合。

根据“无限分片范式”（参见 2.1.2），任何分片链只是一个（临时）虚拟 “帐户链” 集合，每个只包含一个帐户。这意味着，本质上，全局分片链状态必须是一个hashmap。

ShardchainState := (Account AccountState) (23)

如果我们正在讨论分片的状态*(w,s)*（参见 2.1.8），那么所有出现在这个hashmap索引中的 account_id 必须以前缀 s 开头。

实际上，我们可能希望将 AccountState 分成几个部分（例如，将帐户输出消息队列分开以简化相邻分片链的验证），并在 ShardchainState 内部有几个hashmap（Account ➔ FountStateParti）。我们还可以向 ShardchainState 添加少量 “全局” 或 “整数” 参数（例如，属于该分片的所有帐户的总余额，或所有输出队列中的消息总数）。从 “逻辑”（“高级”）的角度来看，（23）是分片链全局状态看起来的良好的第一步。AccountState 和 ShardchainState 形式描述的字段可以借助 TL 方案（参见 2.2.5）完成。

2.5.2 拆分和合并分片链状态。

请注意，分片链状态（23）的无限分片范例描述显示了在分割或合并分片时应如何处理此状态。实际上，这些状态转换结果是使用hashmap的操作，非常简单。

2.5.3 账户链状态。

（虚拟）帐户链状态只是一个帐户的状态，由 AccountState 字段来描述。 通常它具有 2.3.20 中列出的全部或部分字段，具体取决于所使用的具体构造函数。

2.5.4 全局工作链状态。

与（23）类似，我们可以通过相同的公式定义全局工作链状态，但允许使用 account_id 获取任何值，而不仅仅是一个分片的值。 类似于 2.5.1 中的注释也适用于这种情况：我们可能希望将该hashmap拆分为几个hashmap，我们可能希望添加一些 “整数（integral）” 参数，例如总余额。

本质上，全局工作链状态必须由 和分片链状态同类型的ShardchainState 给出，因为如果此工作链的所有现有的分片链合并为一个，我们将会获得分片链的状态。

2.5.5 从 Low-level （低级）来看：“Bag of Cells”。

帐户链或分片链状态的 “低级（low-level）” 描述也是对上面给出的 “高级（high-level）” 描述的补充。这个描述非常重要，因为它非常普遍，为通过网络去表示、存储、序列化和传输 TON 区块链所使用的几乎所有数据，提供了通用基础（区块、分片链状态、智能合约存储、Merkle 证明等）。同时，这种普遍的 “低级” 描述一旦被理解和实施，就可以使我们只关注 “高级” 来考虑。

回想一下，TVM 通过 TVM cells树或简称cells（参见 2.3.14 和 2.2.5）表示任意代数类型的值（例如，包括（23）的 ShardchainState）。

任何这样的cell由两个描述符字节组成，定义了某些标志和值 0≤b≤128，原始字节数，0≤c≤4，即对其他cell的引用数量。然后是 b 原始字节和 c cell引用。[19]

Cell引用的确切格式取决于实现以及cell是位于 RAM中，还是磁盘、网络数据包、区块中等等。一个有用的抽象模型在于想象所有cells都保存在内容可寻址的内存中，cells的地址等于其（sha256）哈希。回想一下，一个cell的（Merkle）哈希是通过用它（递归计算的）哈希替换对其子cells的引用，并对生成的字节串进行哈希化来精确计算。

以这种方式，如果我们使用cells的哈希来引用cells（例如，其他cells的内部描述），则系统稍微简化，并且一个cell的哈希开始与表示它的字节串的哈希重合。

现在我们看到 TVM 表示的任何对象，包括全局分片链状态，可以表示为 “bag of cells”——例如，一组cells及对其中一个的 “根（root）” 引用（例如，通过哈希）。请注意，从此描述中删除了重复的cells（“bag of cells” 是一组cells，而不是多重集合的细胞[a multiset of cells]），因此抽象树表示可能实际上成为有向无环图（DAG）表示。

有人甚至可能将这种状态保存在 B-或 B + -tree 的磁盘上，包含所有相关的cells（可能带有一些额外的数据，如子树高度或参考计数器），由cell哈希索引。然而，这种想法的天真实现将导致一个智能合约的状态分散在磁盘文件的远端部分，我们宁愿避免它。[20]

现在我们将详细解释 TON 区块链所使用的几乎所有对象如何表示为一个“bag of cells”，从而证明这种方法的普遍性。

2.5.6 分片链的区块作为 “Bag of Cells”。

分片链的区块本身也可以用代数类型描述，并存储为一个 “bag of cells”。然后，可以简单地通过以任意顺序连接表示一个 “bag of cells”中每个cell的字节串来获得块的朴素二进制表示。例如，通过在区块头提供所有cell的偏移列表，并且尽可能将该列表中具有 32 位索引的其他

[19] 可以证明，如果同样经常需要存储在cells树中的所有数据的Merkle样张，则应该使用具有b +ch≈2（h + r）的cell来最小化平均Merkle校样大小，其中h = 32是 散列大小（以字节为单位），r≈4是cell引用的“字节大小”。 换句话说，一个cell应该包含两个引用和一些原始字节，或者一个引用和大约36个原始字节，或者根本不包含72个原始字节的引用。

[20] 一种更好的实现是将智能合约的状态维持序列化字符串（如果它很小），或者保存在单独的B树中（如果它很大）；那么代表区块链状态的顶级结构将是一个B树，其叶子允许包含对其他B树的引用。

cells的哈希引用替换为该表示，可以改进和优化该表示。然而，人们应该想象一个块本质上是一个 “bag of cells”，所有其他技术细节只是次要的优化和实现问题。

2.5.7 更新 “Bag of Cells” 的对象。

想象一下，我们有一个旧版本的某个对象表示为 “bag of cells”，我们想要代表同一个对象的新版本，据说与前一个对象没有太大差别。人们可能只是将新状态表示为具有其自身根的另一个 “bag of cells”，并从中移除旧版本中出现的所有cell。剩下的 “bag of cells” 本质上是对象的更新。拥有此对象的旧版本和更新的每个人都可以计算新版本，只需将两组cells联合起来，然后删除旧的根（如果引用计数器变为零，则减少其引用计数器并取消分配cell） 。

2.5.8 更新账户状态。

可以使用 2.5.7 中描述的思想来表示对帐户状态, 或分片链的全局状态或任何hashmap的更新。 这意味着当我们收到一个新的分片链区块（这是一个 “bag of cells”）时，我们不仅仅是单独解释这个 “bag of cells”，而是首先将它与代表分片链前一状态的 “bag of cells” 结合起来。 在这个意义上，每个块可以 “包含（contain）” 区块链的整个状态。

2.5.9 区块的更新

回想一下，一个区块本身就是一个 “bag of cells”，因此，如果有必要编辑一个区块，可以类似地将 “区块更新（block update）” 定义为 “bag of cells”。在 “bag of cells”的情况下进行解释， 则是该区块的先前版本。这大致是 2.1.17 中讨论的 “垂直块（vertical blocks）” 背后的想法。

2.5.10 Merkle 证明作为 “Bag of Cells”。

请注意，（广义）Merkle 证明——例如，一个断言 x [i] = y 从已知值 Hash（x）= h（参见 2.3.10 和 2.3.15）开始——也可以表示为 “bag of cells”。 也就是说，只需要提供一个cells的子集，该子集对应于从 x：Hashmap（n，X）的根到其所需分支的路径，其索引为 i:2^x和值 y:X。对不处于该路径上的这些cells的子集的引用，将在该证明中保持“未解析”，由cell哈希表示。还可以提供同步的Merkle证明，如，x [i] = y和x [i'] = y'，通过包括在“bag of cells”中的cells，它们位于从x的根到对应于索引i和i'的叶子的两条路径的并集上。

2.5.11 Merkle 证明作为全节点的响应。

实质上，具有分片链（或帐户链）状态的完整副本的完整节点可以在轻节点（例如，运行 TON 区块链客户端的轻节点）请求时提供 Merkle 证明，从而仅使用此 Merkle 证明中提供的 cells，启用接收器在没有外部帮助的情况下执行一些简单的查询。轻节点可以将序列化格式的查询发送到整个节点，并通过 Merkle 证明或 Merkle 证明接收正确答案，因为请求者应该只能使用 Merkle 证明中包含的cells来计算答案。这个 Merkle 证明只包含一个 “bag of cells”，只包含那些属于分片链状态的cells，这些cells在执行轻节点查询时已被完整节点访问。这种方法尤其可用于执行智能合约的 “获取查询（get queries）”（参见 4.3.12）。

2.5.12 用 Merkle 有效证明增强更新和状态更新。

回想一下（参见 2.5.7），我们可以从旧值（old value）x来描述对象状态的变化：从X到一个新值（new value）x'：X通过一次“更新（update）”，它只是“bag of cells”，包含代表新值的子树中的那些cell x'，但不在表示旧值 x 的子树中，因为假定接收器具有旧值 x 及其所有cells的副本。

但是，如果接收器没有 x 的完整副本，但只知道它的（Merkle）哈希 h = Hash（x），它将无法检查更新的有效性（即，所有 “dangling”cell更新中的引用确实引用 x）树中的细胞。人们希望得到 “可验证的（verifiable）” 更新，并通过 Merkle 证明旧状态中所有被引用的细胞的存在。然后任何只知道 h = Hash（x）的人都能够检查更新的有效性并自己计算新的 h'= Hash（x'）。

因为我们的 Merkle 证明本身就是 “bag of cells”（参见 2.5.10），所以可以将这样的增强更新构建为 “bag of cells”，其中包含 x 的旧根，其中的一些后代以及来自 x 的根到它们，以及 x' 的新根和它不属于 x 的所有后代。

2.5.13分片链区块中的账户状态更新。

应如 2.5.12 中所讨论的那样，扩充分片链区块中的帐户状态更新。 否则就有人可能会提交一个包含无效状态更新的块，例如旧状态中缺少某个cell； 证明这种阻滞无效将是有问题的（质疑者将如何证明一个cell不属于先前的状态？）。

现在，如果块中包含的所有状态更新都得到了增强，则很容易检查它们的有效性，并且它们的无效性也很容易显示为违反（广义）Merkle 哈希的递归定义属性。

2.5.14 “Bag of Cells” 思想

先前的考虑表明，我们需要在 TON 区块链或网络中存储或传输的所有内容都可以表示为 “bag of cells”。这是 TON 区块链设计理念的重要组成部分。一旦解释了 “Bag of Cells” 方法并定义了 “bag of cells” 的一些 “low-level” 序列化，就可以在高层次简单抽象定义出（依赖）代数数据类型的所有内容（区块格式，分片链和帐户状态等）。“万物皆a bag of cells” 统一的理念，大大简化了看似无关的服务的实施（比如 5.1.9 举例涉及支付通道）。

2.5.15 TON 区块链的区块“头（header）”。

通常，区块链中的一个区块以小头开始，包含前一个区块的哈希，它自身的创建时间，区块链中包含的所有交易的树的 Merkle 哈希，等等。然后将区块哈希定义为该小头的哈希。因为区块头最终取决于区块中包含的所有数据，所以不能在不改变其哈希的情况下改变区块。

在 TON区块链的区块使用的 “bag of cells” 方法中，没有指定的区块头。相反，区块哈希被定义为区块的根cell的（Merkle）哈希。因此，区块链的顶部（根）cells可能被认为是该区块的小 “头（header）”。

但是，根cell可能不包含通常从这种区块头中预期的所有数据。实质上，需要区块头包含*区块数据类型(Block datatype)*中定义的一些字段。通常，这些字段将包含在几个cells中，包括根目录。这些是共同构成所讨论的字段值的 “Merkle 证明” 的cells。有人可能会在任何其他cells之前一开始就坚持要求一个区块链包含这些 “header cells”。然后只需要下载区块序列化的前几个字节，以获得所有“header cells”，并知道所有预期的字段。

2.6 创建和验证新的区块。

TON 区块链最终由分片链和主链区块组成。为了保障系统平稳运行，这些区块必须通过网络来创建、验证并其传播给所有相关方。

2.6.1 验证人（Validator）。

新区块由特殊的指定节点（称为验证人）创建和验证。基本上，任何希望成为验证人的节点都可以成为一个验证人，只要它能够将足够大量的质押资产（在 TON 代币中，即 Grams；参见附录 A）存入主链。 验证人因其出色的工作而获得了一些 “奖励”，也就是说，所有来自交易（消息）、存储和 gas的费用都被投入到新生成的区块和一些新铸造的币中，这反映了整个社区对保证 TON Blockchain 平稳运作的的验证人的“感激（gratitude）”该收入按比例分配给所有参与的验证人。

然而，作为验证人是一项高度责任。如果验证人签署了无效区块，则会迎来通过丢失部分或全部质押资产的处罚，并暂时或永久地从验证人集合中排除。如果验证人不参与创建区块，则不会收到与该区块相关的奖励份额。如果验证人长期放弃创建新区块，它可能会丢失部分质押资产，并被暂停或永久排除在验证人集合之外。

所有这一切都意味着验证人不会 “什么都不做” 就赚到钱。实际上，它必须追踪所有或一些分片链的状态（每个验证人负责验证和创建分片链某个子集中的新区块），执行这些分片链中智能合约请求的所有计算，接收有关其他的更新分片链等。该活动需要相当大的存储空间、算力和网络带宽。

2.6.2 验证人（Validator）代替矿工（Miner）。

回想一下，TON区块链使用的是共识算法是 POS，而不是POW这种被当前版本的以太坊以及大多数其他加密货币采用的工作量证明方法。这意味着人们不能通过提供一些证明工作（计算许多其他无用的哈希）来 “挖（mine）” 一个新的区块，并因此获得一些新的币。相反，人们必须成为验证人，并花费计算资源来存储和处理 TON 区块链的请求和数据。简而言之，某人如果想要开采新的代币，则必须成为验证人。这样一来，验证人就是新的矿工。

然而，除了成为验证人外，还有其他一些方法来赚币。

2.6.3 提名人（Nominator）和 “mining pools（矿池）”

要成为验证人，需要购买并安装多个高性能服务器并需要维持良好稳定网络连接。这并不像目前开发比特币所需的 ASIC 设备那么昂贵。然而，绝对不能在家用电脑上挖掘新的 TON代币，更别提智能手机了。

在比特币，以太坊和其他工作量证明加密货币挖掘社区中，有一个挖掘池的概念，挖掘池其中许多节点个体不具由挖掘新区块的计算能力，但他们团结在一起努力并在之后分享奖励。

这对应在权益证明世界中的相应概念则提名人。从本质上讲，这是一个节点借钱来帮助验证人增加其抵押物；验证人然后将其奖励的相应份额（或之前商定的一部分——比如 50%）分配给提名人。

通过这种方式，提名人也可以参与 “挖矿（mining）” 并获得与其愿意为此目的存入的金额成比例的一些奖励。它仅获得验证人奖励的相应份额的一小部分，因为它仅提供 “资本（capital）”，但不需要购买计算能力，存储和网络带宽。

但是，如果验证人因无效行为而失去其抵押物，则提名人也会失去其抵押物。从这个意义上说，提名人分担风险。它必须明智地选择其指定的验证人，否则可能会赔钱。从这个意义上说，提名人做出加权决定，并用他们的资金对某些验证人进行 “投票（vote）”。

另一方面，这个提名或借出系统使人们能够成为验证人，而无需先向 Grams（TON 币）投入大量资金。换句话说，它可以阻止那些使用大量的 Grams 垄断验证人的人。

2.6.4 渔夫（Fisherman）：通过指出别人的错误来获得金钱。

获得一些奖励而不是验证人的另一种方法是成为一名渔夫。基本上，任何节点都可以通过在主链中存入少量存款而成为渔夫。然后，它可以使用特殊的主链交易来发布先前由验证人签名和发布的一些（通常是分片链）区块的（Merkle）无效证明。如果其他验证人同意该无效证明，则违规验证人将被处罚（通过丢失部分质押资产），并且渔夫获得一些奖励（从违规验证人处没收的一小部分币）。之后，必须按照 2.1.17 中的描述更正无效（分片链）块。纠正无效的主链区块可能涉及在先前提交的主链区块之上创建 “垂直” 块（参见 2.1.17）；没有必要创建主链的分支。

通常，渔夫需要成为至少一些分片链的完整节点，并通过运行至少一些智能合约的代码来花费一些计算资源。虽然渔夫不需要具有作为验证人的计算能力，但我们认为成为渔夫的自然候选人是准备处理新区块的潜在验证人，但尚未被选为验证人（例如，由于未能存放足够大的抵押）。

2.6.5 校对人（Collator）：通过给验证人提供出新区块的建议来获得金钱。

另一种获得一些奖励而不是验证人的方法是成为一名校对人。这是一个节点，它准备并向验证人建议新的分片链候选区块，补充（校对）从该分片链的状态和其他（通常是邻近的）分片链获取的数据，以及合适的 Merkle 证明。（这是必要的。例如，当某些消息需要从相邻的分片链转发时。）然后验证人可以轻松地检查建议的候选区块的有效性，而无需下载此或其他分片链的完整状态。
因为验证人需要提交新的（校对后的）候选区块以获得一些（“挖矿”）奖励，所以将一部分奖励支付给愿意提供合适的候选区块的校对人是有意义的。通过这种方式，验证人可以通过将其外包给校对人来摆脱观察相邻分片链状态的必要性。

但是，我们希望在系统的初始部署阶段不会有单独的指定校对人，因为所有验证人都可以自己充当校对人。

2.6.6 校对人或验证人：从帮助用户整合交易中获取金钱。

用户可以向一些校对人或验证人打开小额支付通道，并支付少量币以换取在分片链中包含他们的交易的权利。

2.6.7 选举全局验证人人。

每个月选出一组 “全局（global）” 验证人（主链每出 2^{19} 个块选举一次）。这个设置普遍会被提前一个月确定并被广泛宣告。

为了成为验证人，节点必须将一些TON币（Grams）转移到主链中，然后将它们作为预备的质押资产s发送至一个特殊的智能合约。与质押资产一起发送的另一个参数是 $I \geq 1$，与可能的

最小值比起来，这是该节点愿意接受的最大验证负载。*l* 还有一个全局上限（另一个可配置参数）参数 L，当 L=10 时，我们把 *l* 称为 L。

然后通过这个智能合约选择全局验证人，简单地通过最大建议质押选择最多 T 个候选人并公布它们的身份。最初，验证人的总数是 T=100；我们预计随着负载的增加它可能增长到1000。T是一个可配置参数（参见 2.1.21）。

每个验证人的实际质押资产计算如下：如果最高的T建议的抵押物是 $s_1 \geq s_2 \geq \cdots \geq s_T$，第*i*个验证人的实际赌注设置为$s'_i := \min(s_i, l_i \cdot s_T)$。通过这种方式，$s'_i / s'_T \leq l_i$，因此第*i*个验证人获得的负载不超过最弱验证人的li≤L倍（因为负载最终与抵押物成比例）。

然后当选的验证人可以撤回质押资产中未使用的部分，即$s_i - s'_i$。未能成功获选的验证人候选人可以撤回它们提出的所有建议质押资产。

每个验证人都发布其*公共签名密钥*，不一定等于质押资产来源的账户的公钥。[21]

验证人的质押资产被冻结，直到它们当选的期限结束，或者还有一个月的额外时间，以防出现新的纠纷（即，找到由这些验证人其中之一签名的无效块）。在此之后，将返还质押资产，以及验证人的铸币份额及在此期间处理的交易费用。

2.6.8 选举验证人“任务组（task groups）”。

全局验证人（其中每个验证人被认为拥有的资产远远多于质押资产——否则验证人可能倾向于伪造几个身份并将自己的质押资产分发给每一个假身份）仅用于验证新的主链区块。分片链区块仅通过特别选择的验证人子集进行验证，这些验证人子集取自 2.6.7 中所述选择的全局验证人集。
为每个分片定义的这些验证人“子集”或“任务组”每小时轮换一次（实际上，每 2^{10} 个主链区块轮换一次），并且它们会提前一小时知道，这样每个验证人都知道验证需要哪些分片，并且可以为此做好准备（例如，通过下载丢失的分片链数据）。

用于为每个分片（w，s）选择验证人任务组的算法是确定性伪随机。它使用验证人嵌入的伪随机数到每个主链区块（由使用阈值签名的共识生成）来创建随机种子，然后计算为每个验证人计算 哈希，例如（code(w).code(s).validator*id.rand*seed）。然后验证人按该哈希的值进行排序，并选择前几个验证人，以便至少具有验证人总质押资产的 20 / T，并且至少包含 5 个验证人。

这种选择可以通过特殊的智能合约来完成。在这种情况下，选择算法很容易升级，而不需要 2.1.21 中提到的投票机制的硬分叉。到目前为止提到的所有其他“常量（constants）”（例如 2^{19}, 2^{10}, T, 20, 和 5）也是可配置参数。

2.6.9 在每个任务组上轮换优先级顺序。

根据先前主链区块和（分片链）区块序列号的哈希， 对于分片任务组的成员而言具有一定的“优先级（priority）”顺序。如上所述，该顺序通过生成和排序某些哈希来确定。

[21] 在每个验证人选举的过程中，生成和使用密钥对是有意义的。

当需要生成新的分片链区块时，选择创建该区块的分片任务组验证人通常是关于该轮换 “优先级” 顺序中的第一个。如果它无法创建区块，则第二个或第三个验证人可以执行该操作。从本质上讲，它们都可以推荐它们的候选区块，但具有最高优先级的验证人建议的候选区块应该作为拜占庭容错（BFT，Byzantine Fault Tolerant）共识协议的结果而获胜。

2.6.10 分片链候选区块的传播。

由于分片链任务组成员资格是提前一小时知道的，因此使用 TON网络的一般机制（参见 3.3），其成员可以利用该时间构建专用的 “分片验证人多播覆盖网络（shard validators multicast overlay network）”。当需要生成新的分片链区块时——通常在最近的主链区块传播后一两秒钟——每个人都知道谁具有最高优先级来生成下一个区块（参见 2.6.9）。该验证人将自行或者在校对人的帮助下创建一个新的校对过的候选区块，（参见 2.6.5）。验证人必须检查（验证）该候选区块（特别是如果它已由某个校对人准备）并使用其（验证人）私钥对其进行签名。然后使用预先安排的多播覆盖网络将候选区块传播到任务组的其余部分（任务组创建其自己的专用覆盖网络，如 3.3 中所述，然后使用 3.3.15 中描述的流式多播协议的版本来传播候选区块）。

真正的 BFT 方式是使用一个拜占庭组播协议（Byzantine multicast protocol），例如 Honey Badger BFT 中使用的协议：用（N，2N / 3）——擦除码来编码候选区块，直接发送 1 / N 结果数据给组里的每个成员，并期望它们将其部分数据直接组播到组的所有其他成员。

然而，更快更直接的方法（参见 3.3.15）是将候选区块分成一系列有符号的一千字节块（“块(chunks)”），用 Reed-Solomon 或者一个喷泉码（例如 RaptorQ 代码）来增加它们的序列，并开始向 “组播网格（multicast mesh）”（即覆盖网络）中的邻近节点发送区块，期望它们进一步传播这些区块。一旦验证人获得足够的块以从它们重建候选区块，它就签署确认收据并通过其邻近节点将其传播到整个组。然后它的邻近节点停止向它发送新的区块，但是可以继续发送这些区块的（原始）签名，相信该节点可以通过自己应用 Reed-Solomon 或 喷泉码（具有所有必要的数据）来生成后续区块，并将它们与签名结合起来传播给尚未准备好的相邻区块。如果在删除所有 “坏” 节点后 “组播网格（multicast mesh）”（覆盖网络）仍保持连接（回想一下，以拜占庭方式允许多达三分之一的节点是坏的，即以任意恶意方式行事），该算法将尽可能快地传播候选区块。

不仅指定的高优先级区块创建者可以将其候选区块组播到整个组。优先级第二和第三的验证人可以立即或在未能从最高优先级验证人接收候选区块之后开始组播它们的候选区块。但是，通常只有具有最大优先级的候选区块将由所有（实际上，至少为任务组的三分之二）验证人签名并作为新的分片链区块提交。

2.6.11 验证候选区块。

一旦验证人接收到候选区块并且检查了其原始验证人的签名，则接收验证人通过执行其中的所有交易并检查它们的结果是否与所声明的一致来检查该候选区块的有效性。从其他区块链导入的所有消息必须由校对数据中的合适 Merkle 证明支持，否则候选区块被视为无效（并且，如果将此证明提交给主链，则验证人已经签署了此候选区块则可能会受到惩罚）。另一方面，如果发现候选区块有效，则接收验证人对其进行签名，并通过 “组播网状网络 （multicast mesh network）” 或直接网络消息将其签名传播到该组中的其他验证人。

我们想强调一个验证人不需要访问这个或相邻的分片链的状态，以便检查（校对的）候选区块的有效性。[22]这允许验证非常快速地进行（没有磁盘访问），并且减轻验证人的计算和存储负担（特别是如果它们愿意接受外部校对人的服务来创建候选区块者）。

2.6.12 选举下一个候选区块。

一旦候选区块（通过质押）收集任务组中验证人至少三分之二的有效性签名，它就有资格被提交为下一个分片链区块。运行 BFT 协议以对所选择的候选区块（可能存在多于一个）进行一致，所有 “好” 验证人优选该轮具有最高优先级的候选区块。作为运行该协议的结果，通过至少三个验证人（通过质押）的签名来增强该区块。这些签名不仅证明了所讨论的块的有效性，而且证明了它是由 BFT 协议选出的。之后，区块（没有校对的数据）与这些签名组合，以确定的方式序列化，并通过网络传播给所有相关方。

2.6.13 验证人必须保留它们签名过的区块。

在它们加入任务组及其后至少一个小时（或者更确切地说是 2^{10}个块）期间，验证人应该保留它们已签署和提交的块。未能向其他验证人提供签名区块可能会受到惩罚。

2.6.14 将新分片链区块的块头和签名传播到所有验证人。

验证人使用类似于为每个任务组创建的组播网状网络，将新生成的分片链区块的区块头和签名传播到全局验证人集。

2.6.15 主链生成新的区块。

在生成所有（或几乎所有）新的分片链区块之后，可以生成新的主链区块。该过程与分片链区块（参见 2.6.12）基本相同，不同之处在于所有验证人（或至少三分之二）都必须参与此过程。因为新的分片链区块的区块头和签名被传播至所有验证人，所以每个分片链中最新区块的哈希可以且必须包含在新的主链区块中。一旦这些哈希值被提交到主链区块中，外部观察者和其他分片链可能会认为新的分片链区块已提交且不可变（参见 2.1.13）。

2.6.16 验证人必须同步主链的状态。

主链和分片链之间一个值得注意的区别是，所有验证人都需要追踪主链状态，而不依赖于校对数据。这很重要，因为验证人任务组的知识来自主链状态。

2.6.17 分片链的区块是并行生成和传播的。

通常，每个验证人都是几个分片链任务组的成员；它们的数量（因此验证人的负载）大约与验证人的抵押总比成比例。这意味着验证人并行运行新的分片链区块生成协议的几个实例。

2.6.18 区块保留攻击的缓解。

因为验证人的总集合在仅看到其标题和签名之后将新分片链区块的哈希值插入主链中，所以生成该区块的验证人很可能会合谋并试图避免完整地发布新区块。这将导致相邻分片链的验证人无法创建新区块，因为一旦将其哈希值提交到主链中，它们必须至少知道新区块的输出消息队列。

[22] A可能的例外是相邻分片链的输出队列的状态，需要保证2.4.21中描述的消息排序要求，因为在这种情况下Merkle样张的大小可能变得过高。

为了缓解这种情况，新区块必须收集来自其他验证人的签名（例如，相邻分片链的任务组加起来的三分之二），证明这些验证人确实具有该区块的副本并且愿意将它们发送给其他其他验证人。只有在这些签名出现后，新区块的哈希才能包含在主链中。

2.6.19 主链区块比分片链区块生成更晚。

主链区块大约每五秒生成一次，分片链区块也是如此。 然而，虽然所有分片链中的新区块生成基本上同时（通常由新的主链区块的释放触发），但是故意延迟生成新的主链区块是为了允许包含主链中新生成的分片链区块新的哈希值。

2.6.20 慢的验证人可能收到较低的奖励。

如果验证人动作“慢”，则可能无法验证新的候选区块，并且可以在没有其参与的情况下收集提交新块所需的签名的三分之二。 在这种情况下，它将获得与此块相关的奖励的较低份额。 这为验证人提供了优化其硬件、软件和网络连接的激励，以便尽可能快地处理用户交易。

但是，如果验证人在提交之前未能对块进行签名，则其签名可能包含在下一个区块之一中，然后包含在部分奖励中（指数级递减，具体取决于自生成以来已生成的块数，例如，如果验证人延迟了k区块，那么就是0.9^k）则仍将给予该验证人。

2.6.21 验证人签名的 “深度（Depth）”。

通常，当验证人对区块进行签名时，签名仅证明区块的相对有效性：如果该区块和其他分片链中所有的先前区块都有效，则该区块有效。 验证人不能因为将先前区块中提交的无效数据视为理所当然而受到惩罚。

但是，区块的验人者签名具有称为 “深度” 的整数参数。 如果它不为零，则意味着验证人也断言指定数量的先前区块的（相对）有效性。 这是 “慢” 或 “暂时离线（temporarily offline）”的验证人捕获并签署一些未经签名提交的区块的方法。 他们仍然会获得一些奖励（参见 2.6.20）。

2.6.22 验证人负责签名的分片链区块的相对有效性；绝对有效性如下。

我们想再次强调，在分片链区块 B 上的验证人签名仅证明该区块的相对有效性（或者如果签名具有 “Depth”d，则可能也是 d 先前块的相对有效性，参见 2.6.21；但这并不会影响以下的讨论。换句话说，验证人断言通过应用 2.2.6 中描述的块评估函数 ev_block 从先前状态 s 获得分片链的下一状态 ：

$$s' = ev\_block(B)(s) \quad (24)$$

以这种方式，如果原始状态 s 证明为 “错误（incorrect）”（例如，由于先前区块中有一个无效），则不能惩罚签名块区 B 的验证人。渔夫（参见 2.6.4）只有在发现相对无效的区块时才应该投诉。 PoS 系统作为一个整体努力使每个区块相对有效，而不是递归（或绝对）有效。但是请注意，如果区块链中的所有区块都相对有效，那么所有区块和整个区块链都是绝对有效的；使用区块链长度的数学归纳可以很容易地显示这个陈述。通过这种方式，对于区块相对有效性的断言验证共同证明了整个区块链的绝对有效性。

注意，通过对区块 B 进行签名，验证人在给定原始状态 s 的情况下断言该区块是有效的（即，（24）的结果不是指示不能计算下一状态的值⊥）。以这种方式，验证人必须对在（24）的评估期间访问的原始状态细胞执行最小的正式检查。

例如，假设预期包含从提交到区块中交易访问帐户的原始余额的cell原来具有零个原始字节，而不是预期的 8 或 16。 然后原始余额根本无法从cell取回，并在尝试处理区块时发生 “未处理的异常（unhandled exception）”。在这种情况下，验证人不应该在遭受到惩罚时签署这样的区块。

2.6.23 签名主链的区块。

主链区块的情况有所不同：通过签署主链区块，验证人不仅断言其相对有效性，还断言所有先前区块的相对有效性，直到该验证人承担其责任时的第一个区块（但不再向后）。

2.6.24 验证人的总量。

在目前为止所描述的系统中，要选择的验证人总数的上限 T（参见 2.6.7）不能超过，比如几百或一千，因为所有验证人都应该参与 BFT 共识协议用于创建每个新的主链区块，并且不清楚这些协议是否可以扩展到数千个参与者。更重要的是，主链区块必须（通过质押）收集所有验证人中至少三分之二的签名，并且这些签名必须包含在新区块中（否则，系统中的所有其他节点都没有理由在自己没有验证新区块时信任它。如果超过，例如，每个主链区块中必须包含1000个验证人签名，这将意味着每个主链区块中的更多数据，由所有完整节点存储并通过网络传播，并且花费更多的处理能力来检查这些签名（在 PoS 系统中，完整节点不需要自己验证区块，但是它们需要检查验证人的签名）。

虽然限制 T 到1000个验证人似乎对 TON 区块链的部署的第一阶段来说已经足够，但是当分片链总数变得大到几百个验证人不足以处理时，必须为未来的增长做出规定。他们都是。为此，我们引入了一个额外的可配置参数 T'≤T（最初等于 T），并且只有顶部 T' 选举的验证人（通过质押）才能创建和签署新的主链区块。

2.6.25 去中心化的系统。

人们可能怀疑 TON区块链证明系统依赖于 T≈1000 验证人来创建所有分片链和主链区块，与传统的工作量证明区块链相比，必然会变得 “过于中心化”。比如比特币或以太坊，每个人（原则上）都可以挖掘一个新区块，而没有明确的矿工总数上限。

然而，POW 如比特币和以太坊，目前需要大量的算力[高 “哈希率（hash rates）”]来挖掘新区块，并且概率很低。因此，出块往往集中某几个人手中，他们投入大量资金建立研发中心，这些研发中心充满了针对挖矿优化的定制设计硬件；算力掌握在几个大型采矿池的手中，这些采矿池集中并协调那些无法自己提供足够 “算力” 的人而接入网络。

因此，截至 2017 年，超过 75%的新的以太坊或比特币区块由不到 10 名矿工生产。事实上，两个最大的 以太坊 矿池共同产生了超过一半的新区块！显然，这样的系统比依赖于 T≈1000 个节点来生成新区块的系统更加中心化。

人们可能还会注意到成为 TON区块链验证人所需的投资——即购买硬件（例如，几个高性能服务器）和质押资产（如果需要，可以通过一系列提名人轻松收集；参照 2.6.3）——比成为一个成功的比特币或以太坊个人矿工所需的付出低得多。实际上，2.6.7 的参数 L 将迫使提名

人不加入最大的“矿池”（即，已经积累了最大质押资产的验证人），而是寻找目前接受来自提名人资金的较小验证人，或者甚至创建新的验证人，因为这将允许验证人更高比例的 s'i / si——并且还可以使用提名人的质押资产，因此从挖矿中获得更大的奖励。通过这种方式，TON 的 POS 实际上鼓励分散（创建和使用更多验证人）并惩罚中心化。

2.6.26 区块的相对可靠性。

区块的（相对）可靠性取决于已对该区块签名的所有验证人的总质押资产。换句话说，如果这个区块被证明无效，这那某些参与者就会失去部分质押资产。如果关注转移价值低于区块可靠性的交易，可以认为它们足够安全。从这个意义上讲，相对可靠性是外部观察者在特定区块中可以拥有的信任度量。

请注意，我们说的是区块的相对可靠性，因为它保证区块是有效的，前提是区块和所有其它所指的分片链区块都有效（参见 2.6.22）。

区块的相对可靠性在提交后可以增长——例如，当添加了迟来的验证人签名时（参见 2.6.21）。另一方面，如果这些验证人中的一个由于其与其他区块有关的不当行为而失去部分或全部质押资产，则区块的相对可靠性会降低。

2.6.27 “强化（Strengthening）”区块链

为验证人提供激励很重要，它可以尽可能地提高区块的相对可靠性。 实现此目的的一种方法是向验证人分配一小笔奖励，以便将签名添加到其他分片链的区块中。 即使是“将来可能（would-be）”的有效者，他们已经存入的质押资产不足以通过质押进入最高的 T 验证人并被纳入全局验证人（参见 2.6.7），也可能参加这项活动（如果他们同意保留他们的质押资产而不是在失去选举后退出它。 这些潜在的验证人可能同时成为渔夫（参见 2.6.4）：如果他们不得不检查某些区块的有效性，他们也可以选择报告无效区块并收集相关的奖励。

2.6.28 区块的递归可靠性。

还可以将区块的递归可靠性定义为其相对可靠性的最小值以及它所引用的所有区块的递归可靠性（例如主链区块、先前的分片区链和相邻的分片链的一些区块）。换句话说，如果该区块被证明是无效的，或者因为它本身无效或者因为它所依赖的区块之一是无效的，那么至少这个数量的钱将被某人丢失。如果真的不确定是否信任区块中的特定交易，则应该计算该区块的递归可靠性，而不仅仅是相对区块的递归可靠性。

在计算递归可靠性时走得太远是没有意义的，因为如果我们看得太远，我们将看到由验证人签名的块，其质押资产已经解冻并撤回。在任何情况下，我们都不允许验证人自动重新考虑那些旧的区块（即如果使用了当前的可配置参数值，则在两个月前创建），并创建从它们开始的分叉或使用辅助设备纠正它们“垂直区块链（vertical blockchains）”（参见 2.1.17），即使它们结果无效。我们假设两个月的时间段为检测和报告任何无效区块提供了充足的机会，因此如果在此期间没有对区块进行质询，则根本不可能受到质疑。

2.6.29 轻节点 POS 的结果。

TON区块链使用的权益证明方法的一个重要结果是，TON 区块链的轻节点（运行轻节点的客户端）不需要下载所有分片链甚至主链区块的“头（headers）”，以便能够自行检查由完整节点提供给它的 Merkle 证明的有效性作为其查询的答案。

实际上，因为最新的分片链区块哈希包含在主链区块中，所以完整节点可以很轻松地提供 Merkle 证明，即给定的分片链区块从主链区块的已知哈希开始有效。接下来，工作中的节点只需要知道主链的第一个区块（其中第一组验证人被公布），其中（或至少其中的哈希）可以内置到客户端软件中，并且仅大约每个月发生一个主链区块，其中宣布新选出的验证人集，因为该块将由前一组验证人签名。从那开始，它可以获得几个最新的主链区块，或者至少它们的区块头和验证人签名，并使用它们作为检查由完整节点提供的 Merkle 证明的基础。

2.7 拆分和合并分片链

TON区块链最具特色、最独特的功能之一是一个分片能在负载变高时自动将分为两个，并在负载下降时合二为一（参见 2.1.10）。由于其独特性和对整个项目扩容的重要性，我们须对此进行详细讨论。

2.7.1 分片配置

想一下，在任何给定的时刻，每条工作链会被分成一条或几条分片链（w，s）（参见 2.1.8）。这些分片链可用一个根节点为（w，∅）的二叉树的许多叶子来表示，每个非叶子节点（w，s）都有子节点（w，s.0）和（w，s.1）。通过这种方式，属于工作链 w 的每个帐户都被具体分配到一个分片，并且知道当前分片链配置的人都可以确定包含帐户 account_id 的分片（*w，s）：它是唯一一个用二进制字符串 s 作为account_id前缀的分片*。

分片配置——即此分片二叉树，或给定 w 的所有活跃节点（w，s）的集合（对应分片 二叉树的叶子）——是主链状态的一部分，并且可用于所有追踪主链的人。[23]

2.7.2 最新的分片配置和状态

想一下，最新的分片链区块的哈希值包含在每个主链区块中。这些哈希值按分片二叉树（实际上是树的集合，每个工作链一个）排列。这样每个主链区块都包含最新的分片配置。

2.7.3 宣布并执行分片配置更改

可以通过两种方式更改分片配置：一个分片（w,s）可以分为两个（w,s.0）和（w,s.1），或两个 “兄弟” 分片（w,s.0）和（w,s.1）可以合并为一个分片（w,s）。

这些拆分 / 合并操作会预先在几个区块上公布（例如，2^6；这是可配置参数），首先在相应的分片链区块的 “区块头” 中，然后在引用这些分片链区块的主链区块中。所有相关方都需要这种预先公布，以准备计划是式更改（例如，如 3.3 中所述，搭建覆盖多播网络传播新创建的分片链的新区块）。之后更改得以进行，（在拆分的情况下）首先进入分片链区块的（区块头）（在合并的情况下，两个分片链的区块都应该进行更改），然后传播到主链区块。通过这种方式，主链区块不仅定义了创建之前的最新分片配置，还定义了下一个即将分片配置。

[23] 实际上，分片配置完全由最后一个主链区块决定；这样就简化了分片配置的流程。

2.7.4 新分片链的验证人任务组

想一下，每个分片（即每个分片链）通常配有一个验证人子集（验证人任务组），该子集专门用于创建和验证相应分片链中新的区块（参见 2.6.8）。这些任务组隔一段时间（大约一小时）选出，并且提前一段时间（大约一小时）知晓，并且在此期间是不可更改。[24]

但是，由于拆分 / 合并操作，实际的分片配置可能会在此期间发生更改。必须将任务组分配给新创建的分片。可通过如下操作：

请注意，任何活跃分片（w，s）都将是某些唯一确定的原始分片（w，s'）的衍生，即s' 是 s 的前缀，或者将是原始分片（w，s'）的子树的根，其中 s 将是每个 s' 的前缀。在第一种情况下，我们只需将原始分片（w，s'）的任务组重新用作新分片（w，s）的任务组。在后一种情况下，新分片（w，s）的任务组将是所有原始分片（w，s'）的任务组的并集，这些原始分片都是分片树中（w，s）的衍生。

通过这种方式，每个活跃分片（w，s）都有一个明确定义的验证人子集（任务组）。拆分分片时，两个子代继承了原始分片整个任务组。合并两个分片时，它们的任务组也会合并。

追踪主链状态的任何人都可以为每个活跃分片计算验证人任务组。

2.7.5 原任务组工作期间限制拆分 / 合并的操作

最终将考虑新的分片配置，并且将自动为每个分片分配新的专用验证人子集（任务组）。在此之前，必须对拆分 / 合并操作施加一定的限制；否则，如果原始分片快速分成 2^k 个新分片，则原始任务组可能最终会同时验证一个很大k值的 2^k 个分片链。

这是通过对从原始分片配置（用于选择当前负责的验证人任务组的配置）中删除活动分片配置的距离施加限制来实现的。例如，如果 s' 是 s 的前身（即s'是二进制字符串s的前缀），则可要求分片树中从活动分片（w，s）到原始分片（w，s'）的距离不得超过 3。如果 s' 是 s 的后代（即 s 是 s' 的前缀，则距离不得超过 2。否则，不允许拆分或合并操作。

粗略地说，在给定验证人任务组的工作期间对分片（例如，三个）或合并（例如，两个）的次数施加限制。除此之外，在通过合并或拆分创建了一个分片之后，一段时间（一定数量的块）内不能重新配置。

2.7.6 确定拆分操作的必要性

分片链的拆分操作通过某些形式条件触发（例如，如果连续 64 个区块，则分片链区块至少满 90%）。这些条件由分片链任务组监视。如果满足这些条件，首先在新的分片链区块的区块头中包含“拆分准备”标志（并且传播到引用该分片链区块的主链区块）。然后，在几个区块之后，“拆分完成”标志被包含在分片链区块的区块头中（并传播到下一个主链区块）。

[24] 除非有些节点由于签署无效区块而被暂时或永久禁止，被自动剔除在项目组外。

2.7.7 执行拆分操作

在分片链（w，s）的区块 B 中包含“拆分完成”标志之后，该分片链中不会有后续区块 B'。 相反，将分别创建分片链（w，s.0）和（w，s.1）的两个区块 B_0' and B_1'，两者都将区块 B 引用为它们的前一个区块（并且都将通过块头中的flag标记已拆分分片）。下一个主链区块将包含新分片链的区块 B_0' 和 B_1' 的哈希值；不允许包含分片链（w，s）新区块 B' 的哈希值，因为“拆分完成”事件已经被提交到先前的主链区块中。

请注意，两个新的分片链将由与旧验证人相同的验证人任务组验证，因此它们将自动获得其状态的副本。从无限分片范式（Infinite Sharding Paradigm）的角度来看，状态拆分操作本身非常简单（参见 2.5.2）。

2.7.8 确定合并操作的必要性。

分片链合并操作的必要性可通过某些形式的条件检测到（例如，对于 64 个连续区块，两个兄弟分片链区块的大小之和不超过最大区块大小的 60%）。这些形式条件还应考虑这些区块所消耗的总 gas，并将其与当前区块 gas限制进行比较，否则区块可能会很小，因为有一些计算密集型交易会阻止更多的交易。

这些条件由兄弟分片（w，s.0）和（w，s.1）的验证人任务组监视。请注意，兄弟分片必须是超立方体路由的相连分片（参见 2.4.19），因此来自任一分片的任务组的验证人将在某种程度上监视兄弟分片。

当这些条件得到满足时，任何一个验证人子组都可以通过发送特殊消息向另一个验证人建议它们进行合并。然后，它们组合成一个临时的 “合并任务组”，具有组合的成员能够运行 BFT 共识算法，并在必要时传播区块更新和区块候选。

如果它们就合并的必要性和准备情况达成共识，则 “合并准备” 标志将被提交到每个分片链的某些区块的区块头中，同时还有兄弟任务组至少三分之二的验证人的签名（并传播到下一个主链区块，以便每个人都可以为即将进行的重新配置做好准备）。但是，它们继续为某些预定义数量的区块创建单独的分片链区块。

2.7.9 执行合并操作

之后，当来自两个原始任务组的验证人准备好成为合并的分片链的验证人时（这可能涉及从兄弟分片链转移桩体和状态合并操作），将提交“合并完成”标志至它们分片链区块的区块头中（此事件传播到下一个主链区块），并停止在单独的分片链中创建新区块（一旦出现合并完成标志，则禁止在单独的分片链中创建区块）。反之，则创建一个合并的分片链区块（由两个原始任务组的并集），在其“区块头”中引用它的两个 “前区块”。这反映在下一个主链区块中，它将包含合并分片链区块新创建的的哈希值。之后，合并的任务组继续在合并的分片链中创建区块。

2.8 区块链项目的分类

我们将通过比较 TON区块链与现有的区块链项目来结束对 TON区块链的讨论。然而在此之前，我们必须引入足够普遍的区块链项目分类，基于此分类的特定区块链项目比较将在2.9 里展开。

2.8.1 区块链项目的分类

第一步，我们先提出区块链（即区块链项目）的一些分类标准。这必须忽略所考虑项目的一些具体和独特的特征，所以任何分类都可能不完整，并且是表面的。但是我们认为这是必要的，至少提供了区块链领域项目粗略鸟瞰的第一步。

我们考虑的标准清单如下：

- 单链与多链架构（参见 2.8.2）
- 共识算法：POS 权益证明与 POW 工作量证明（参见 2.8.3）
- 对于权益证明系统，具体的区块生成、验证和所使用的共识算法（两个算法包括 DPOS 与 BFT；参见 2.8.4）
- 支持 “任意”（图灵完备）智能合约（参见 2.8.6）

多链系统有额外的分类标准（参见 2.8.7）：

- 所包含区块链的类型和规则：同构、异构（参见 2.8.8）、混合（参见 2.8.9）。 联盟链（参见 2.8.10）。
- 缺少或存在内部或外部的主链（参见 2.8.11）
- 系统支持分片（参见 2.8.12）。 静态或动态分片（参见 2.8.13）。
- 所包含区块链之间的相互作用：松散耦合和紧密耦合系统（参见 2.8.14）

2.8.2 单链和多链项目

第一个分类标准是系统中区块链的数量。最老也是最简单的项目只包含一条链（简称 “单链项目”）；更复杂的项目使用（或者更确切地说，计划使用）多个区块链（“多链项目”）。

单链项目通常更简单，测试更好；它们经受住了时间的考验。它们的主要缺点是性能低，或者交易吞吐量最低，每秒 10 笔（比特币）到 140 笔[25]（以太坊）不等。一些专门的系统（如比特股）每秒能够处理成千上万的交易，代价是要求区块链状态适合内存，并将处理限制在预定义的交易中，然后由高度优化的代码执行，这些代码用 C ++ 这样的语言编写（没有 VM）。

[25]同15类似，当前情况。现已有更新计划，将以太坊的交易吞吐量提高至几倍以上。

多链项目保证了大家都想要的可扩展性，可以提供更大的总状态和更高的tps，代价是使项目更加复杂，并且其实现起来更具挑战性。因此，目前几乎没有太多的多链项目在运行，但大多数提出的项目都是多链项目。我们相信未来属于多链项目。

2.8.3 创建和验证区块：工作量证明与权益证明

另一个重要的区别是算法和协议，用于创建和传播新区块，检查有效性，如果需要分叉则选择一种分叉。

两种最常见的范例是工作量证明（PoW）和权益证明（PoS）。工作量证明方法通常允许任何节点创建（“挖”）新区块（并获得与挖矿相关的一些奖励），如果它很幸运地在其它竞争者前解决反之就无用的计算问题（通常涉及大量的哈希值计算）。在分叉的情况下（例如，如果两个节点在前一个区块后发布两个都可能有效但不同的区块），则最长的分叉获胜。通过这种方式，保证区块链的不可更改性的是基于生成区块链所花费的工作量（算力资源）：任何想要创建区块链分叉的人都需要重新做这项工作来创建替代方案已提交区块的版本。为此，它需要控制创建新区块所花费的总算力的 50%以上，不然另外一条分叉成为最长链的成功率指数型下降。

权益证明方法基于一些特殊节点（验证人）做出的大量质押（stake，加密货币中的概念）来声明它们已经检查（验证）了一些区块并且发现它们是正确的。验证人签名区块，并为此获得一些小奖励；但是，如果一个验证人被抓到签了一个不正确的区块，并且有证据表明，那么其部分或全部质押资产将被没收。由此，验证人对区块链有效性的总 质押资产量确保了区块链的有效性和不可更改性。

利用权益证明方法更加自然，它激励验证人（取代 PoW 矿工）执行有用的计算（需要检查或创建新的区块，尤其是通过执行区块列出的所有交易进行检查或创建）而不是计算可能无用的哈希。通过这种方式，验证人将购买更适合于处理用户交易的硬件，以便领取与这些交易相关的奖励，从整个系统的角度来看，这似乎是非常有用的投资。

然而，权益证明制度在实施方面更具挑战性，因为需要许多罕见但可能的条件。例如，一些恶意验证人可能合谋破坏系统以获取一些利润（如，改变它们自己的加密货币余额）。这导致了一些不同寻常的游戏理论问题。

简而言之，权益证明更符合自然规律，更有前途，特别是对于多链项目而言（因为如果有许多区块链，工作量证明需要大量的计算资源），但必须更仔细计划和实施。大多数目前正在运行的区块链项目，尤其是最老的项目（比如比特币和最初的以太坊），都使用了工作量证明。

2.8.4 POS 的演变：DPOS vs BFT

虽然工作量证明算法彼此非常相似，并且主要区别在于必须为挖掘新区块而计算的哈希函数，但是权益证明算法有更多可能性。很值得对权益证明进行细分。

这里必须要回答下关于权益证明算法的问题：

- 谁可以生成（“挖”）新区块——任何全节点，或只是生成（相对）小验证人子集的成员？（大多数 PoS 系统需要新生成的区块，并由一名指定验证人签名。）

- 验证人是否通过其签名保证区块的有效性，或者是否所有全节点都可以自行验证所有区块？（可扩展的 PoS 系统必须依赖验证人签名，而不是要求所有节点验证所有区块链的全部区块。）

- 是否有下一个区块链区块的（提前知晓的）指定出块人，以便其他人无法生产该区块？

- 新创建的区块最初仅由一个验证人（其出块人）签署的，还是必须从一开始就收集大多数验证人签名？

虽然根据这些问题的答案，似乎有 2^4 种可能的 PoS 算法类别，但实际区别归结为两种主要的 PoS 方法。 实际上，用于可扩展多链系统的大多数现代 PoS 算法以相同的方式回答前两个问题：只有验证人才能生成新区块，并且它们保证区块有效性，而不需要所有全节点检查所有区块的有效性。

至于最后两个问题，它们的答案非常相关，基本上只留下两个基本选项：

- 委托权益证明（DPOS）：每个区块都有一个众所周知的指定出块人；除此之外没有人可以出块；新区块最初只由其出块的验证人签名。

- 拜占庭容错（BFT）PoS 算法：有一个已知的验证人子集，其中任何一个都可以提出一个新区块；几个建议候选人中的下一个实际区块（在公布到其他节点之前必须由大多数验证人验证和签名）的选择是通过拜占庭容错共识协议来实现的。

2.8.5 DPOS 和 BFT POS 的比较

BFT 方法的优点在于，新产生的区块从一开始就具有大多数验证人的签名，证明其有效性。另一个优点是，如果大多数验证人正确执行 BFT 共识协议，则根本不会出现任何分叉。另一方面，BFT 共识算法往往相当复杂，需要更多时间让验证人子集达成共识。因此不能经常生成区块。这就是为什么我们期望 TON 区块链（从这个分类的角度来看是一个 BFT 项目）每5秒只产生一次区块。在实践中，如果验证人遍布全球，这个间隔可能会减少到 2-3 秒（但我们不承诺这一点），但不会减少更多。

DPOS 算法的优点非常简单直接。它可以经常生成新的区块——比如每两秒1个区块，或者甚至每秒1个区块[26]，因为它依赖于事先已知的指定区块生产者。

但是，DPOS 要求所有节点——或至少所有验证人——验证收到的所有区块，因为生成和签署新区块的验证人不仅确认该区块的相对有效性，而且还确认它所引用的前一区块的有效性，以及之后回到链上的所有区块（可能直到当前验证人子集的工作期开始）。当前验证人子集上存在预定顺序，因此对于每个区块，都有指定的出块人（即期望生成该块的验证人）；这些指定的出块人以循环方式进行轮流。通过这种方式，区块首先仅由其出块的验证人签名；然后，挖下一个区块挖时，它的出块人选择引用这个区块而非其之前一个区块（否则它的区块将位于较短的链中，可能导致输掉 “最长分叉” 竞争），下一个区块的签名本质上也是前一区块的附加签名。通过这种方式，新区块逐渐收集更多验证人的签名——例如，在生成接下来20个区

[26] 有的人甚至提出DPOS半秒的区块产生时间，即使验证人遍布各大洲也不太实际。

块所需的时间内收集 20 个签名。一个全节点要么需要等待这20个签名，要么自己验证区块，从一个充分确认区块（比如回退20个区块）开始，这可能不是那么容易。

DPOS 算法明显的缺点是，只有在挖了 20 多个区块之后，新区块（以及提交到新区块的交易）才能达到与BFT相同的信任级别（2.6.28 中讨论的“递归可靠性”）。 与之相对的BFT 算法，可立即提供此级别的信任（例如，20 个签名）。另一个缺点是 DPOS 使用“最长分叉胜利”方法切换到其他分叉；如果至少一些出块人未能在我们感兴趣的区块之后生成后续的区块（或者由于网络分区或复杂的攻击，我们未能观察到这些区块），这很可能导致分叉。

我们认为 BFT 方法虽然比 DPOS 实现更复杂，区块之间需要更长的时间间隔，但更适合于“紧密耦合”（参见 2.8.14）多链系统，因为其他区块链在新区块中看到已提交的交易之后可立即行动（例如，为交易生成一条消息），而不用等待 20 次有效性确认（即，接下来的 20 个区块），或等待接下来的6个区块以确保没有分叉出现并自行验证新区块（在可扩展的多链系统中验证其他区块链的区块可能无法实现）。因此，它们可以实现可扩展性，同时保持高可靠性和可用性（参见 2.8.12）。

另一方面，DPOS 可能是 “松散耦合” 多链系统的理想选择，其中不需要区块链之间的快速交互——例如，如果每个区块链（“工作链”）代表一个单独的分布式交换，区块链间的交互仅限于将代币从一个工作链转移到另一个工作链中（或者更确切地说，以一个接近 1:1 的速率将一个山寨币交易到另一个工作链中）。这是比特股项目实际在做的，它非常成功地使用了 DPOS。

总而言之，虽然 DPOS 可以生成新的区块并且更快地包含交易（区块之间的间隔更小），但这些交易达到的信任级别（其他区块链和链下应用所“提交”和“不可更改”的级别） 速度比 BFT 系统慢得多——比如说，30 秒[27]而不是 5 秒。更快的交易包含并不意味着更快的交易提交。如果需要快速的区块链间交互，这可能会成为一个巨大的问题。在这种情况下，必须放弃 DPOS，选择 BFT PoS。

2.8.6 支持交易里的图灵完备代码，即允许执行任意智能合约

区块链项目通常会在其区块中收集一些交易，这会以一种被认为有用的方式改变区块链状态（例如，将一定量加密货币从一个账户转移到另一个账户）。一些区块链项目可能只允许某些特定的预定义分片的事务（例如，在签名正确的情况下，从一个帐户到另一帐户的价值转移）。其他人可能会在交易中支持某种有限形式的脚本。最后，一些区块链支持在交易中执行任意复杂的代码，使系统（至少在原理上）能够支持任意应用程序，只要系统的性能允许。这通常与 “图灵完备的虚拟机和脚本语言”（意味着任何可以用其他计算语言编写的程序可以重写以在区块链内执行）和 “智能合约”（区块链中的程序）有关。

当然，对任意智能合约的支持使系统真正具有灵活性。另一方面，这种灵活性需要付出代价：这些智能合约的代码必须在某个虚拟机上执行，并且当有人想要创建或验证区块时，必须每次为每个交易执行此操作。与预定义且不可更改的简单交易类型相比，这会降低系统的性能，而简单的交易类型可以通过使用诸如 C ++（而不是某些虚拟机）之类的语言实现它们的处理来优化。

[27] 例如，至今为止最优秀的一个DPOS项目，EOS承诺45秒的确认和区块链间交互延迟的时间（参照[5]，“交易确认”和“链间交互延迟”）。

最终，对图灵完备智能合约的支持似乎在任何通用区块链项目中都是可取的；否则，区块链项目的设计者必须事先决定他们的区块链将用于哪些应用程序。事实上，正是比特币不支持智能合约才有了以太坊等新的项目。

在（异构；参见 2.8.8）多链系统中，一些区块链（即工作链）中支持图灵完备智能合约，另一些支持预定义高度优化过的交易，这样可保持 “两全其美”。

2.8.7 多链系统的分类

到目前为止，该分类对于单链和多链系统都是有效的。然而，多链系统承认了更多的分类标准，反映了系统中不同区块链之间的关系。我们现在来讨论这些标准。

2.8.8 区块链类型：同构与异构系统

在多链系统中，所有区块链基本上是相同类型并且具有相同的规则（即，使用相同格式的交易，用于执行智能合约代码的相同虚拟机，共享相同的密码保存等等），这种相似性被明确使用，但每个区块链中的数据不同。在这种情况下，我们将这一系统称为同构的。否则，不同的区块链（这种情况通常称为工作链）可以有不同的“规则”。那么，我们就将这一系统称为异构的。

2.8.9 同构-异构混合系统

有时我们有一个混合系统，其中的区块链有几组类型或规则，但是存在许多具有相同规则的区块链，并且这是公认的事实。然后它是一个同构-异构混合系统。据我们所知，TON 区块是这类系统仅有的一个例子。

2.8.10 若干工作链的异构系统具有相同的规则，联盟链

一些情况下，具有相同规则的若干区块链（工作链）可以存在于异构系统中，但是它们之间的交互与具有不同规则的区块链之间的相互作用相同（即，它们的相似性未被明确使用）。即使它们似乎使用“同样的”加密货币，它们实际上使用不同的“山寨币”（加密货币的不同化身）。有时人们甚至可以使用某些机制将这些山寨币按近1：1 的比例兑换。然而，在我们看来这并不能使系统同构化；它仍然是异构的。我们将这种具有相同规则的异构工作链集合称为联盟链。

虽然允许用相同规则创建多个工作链的异构系统（即联盟链）看起来似乎是构建可扩展系统的廉价方式，但这种方法也有许多缺点。从本质上讲，如果有人在许多具有相同规则的工作链中运行大型项目，这（实际上）并非大型项目，而是该项目的许多小实例。这就像一个聊天应用程序（或游戏），在任一聊天（或游戏）房间中最多只允许 50 个成员，但通过创建新房间来“扩容”，以在必要时容纳更多用户。由此，很多用户可以参与聊天或游戏，但我们可以说这样的系统真正可扩展吗？

2.8.11 内部或外部存在主链

有时，多链项目有一个独特的 “主链”（有时称为 “控制区块链”），例如可用于存储系统的整体配置（所有活跃区块链的集合，或者更确切地说是工作链 ）、当前的验证人集（POS系统）

等等。有时其他区块链与主链“绑定”，例如通过将最新区块的哈希值存入其中（TON区块链也是如此）。

在某些情况下，主链是外部的，这意味着它不是该项目的一部分，而是一些其它预先存在的区块链，最初与新项目的使用完全无关，并且对其不可知。例如，我们可以尝试使用以太坊区块链作为外部项目的主链，并为此（比如挑出和惩罚验证人）将特殊智能合约发布至以太坊区块链。

2.8.12 分片支持

一些区块链项目（或系统）本身支持分片，这意味着几个（必然是同构的；参见 2.8.8）区块链被认为是单个（从高级别的角度来看）虚拟区块链的分片。例如，可以创建具有相同规则的 256 个分片链，并根据其 account_id 的第一个字节将帐户的状态保持在一个选定的分片中。

分片是扩展区块链系统的一种自然方法，因为如果它被正确实现，系统中的用户和智能合约根本不需要知道分片链的存在。实际上，当负载变得过高时，人们通常希望为现有的单链项目（例如以太坊）增加分片。

另一种扩展方法是使用 2.8.10 中描述的异构工作链的“联盟链”，允许每个用户将她的帐户保存在她选择的一个或多个工作链中，并在必要时将资金从她一条工作链的帐户转移到另一个工作链中，基本上执行 1：1 的山寨币交换操作。这种方法的缺点已在 2.8.10 中讨论过。

然而，分片要快速可靠地实现并不容易，因为它意味着不同分片链之间的大量消息。例如，如果账户在 N 个分片之间均匀分配，并且交易只是从一个账户到另一个账户的简单资金转账，那么在单链中只会执行所有交易的一小部分（1 / N）；几乎所有（1——1 / N）交易都涉及两个区块链，需要区块链间通信。如果我们希望这些交易快速进行，我们需要用一个快速系统在分片链之间传输消息。换句话说，区块链项目需要在 2.8.14 中描述的意义上 “紧密耦合”。

2.8.13 动态和静态分片

分片可能是动态的（如果在必要时自动创建其他分片）或静态的（如有预定义数量的分片，只能通过硬分叉来修改）。大多数分片提案都是静态的。TON区块链使用动态分片（参见 2.7）。

2.8.14 区块链之间的相互作用：松散耦合系统和紧密耦合系统

可以根据组成区块链之间支持的交互级别对多链项目进行分类。

最低水平的支持是不同区块链之间没有任何交互作用。我们在这里不考虑这种情况，因为我们宁愿说这些区块链不是一个区块链系统的一部分，而只是相同区块链协议的单独实例。

下一级支持是对区块链之间的消息传递没有任何具体支持，原则上使交互成为可能，但很难用。我们称这种系统为 “松散耦合”；我们必须发送消息并在区块链之间传递价值，好像它们是属于完全独立的区块链项目的区块链一样（例如，比特币和以太坊；想象两方想要将比特币区块链中的一些BTC转为以太坊区块链中的ETH）。换句话说，我们必须在源区块链的区块中包含出站消息（或其生成交易）。然后，她（或其他一方）必须等待足够的确认（例如给定数量

的后续区块）以将原始交易视为“已提交”和“不可变更”，以便执行外部操作。只有这样，才能提交将消息中继到目标区块链的交易（可能连同原始交易的引用和Merkle存在证明）。

如果在传输消息之前没有等待足够长的时间，或者由于其他原因发生了分叉，则两个区块链的连接状态变得不一致：消息被传递到第一个区块链（最终选择的分叉）中从未生成过的第二个区块链中。

有时添加对消息传递的部分支持，方式是通过标准化消息格式以及所有工作链区块中输入和输出消息队列的定位（这在异构系统中尤其有用）。虽然这在一定程度上促进了消息传递，但它在概念上与先前的情况没有太大的不同，因此这种系统仍然 “松散耦合”。

相比之下，“紧密耦合” 系统包括在所有区块链之间提供快速消息传递的特殊机制。所期望的行为是能够在原始区块链的区块中生成消息之后立即将消息传递到另一个工作链。另一方面，“紧密耦合”系统也可望在分叉的情况下保持整体一致性。虽然这两个要求乍一看似乎是矛盾的，但我们认为 TON 区块链使用的机制（将分片链区块哈希包含在主链区块中；使用“垂直”区块链来修复无效区块，参见 2.1.17 ；超立方路由，参见 2.4.19；即时超立方体路由，参见 2.4.20）能使其成为“紧密耦合”系统，也许是目前唯一一个例子。

当然，构建“松散耦合”系统要简单得多；然而，快速有效的分片（参见 2.8.12）要求系统 “紧密耦合”。

2.8.15 简化分类，几代区块链项目

到目前为止，我们提出的分类将所有区块链项目拆分为许多类。但是我们使用的分类标准在实践中恰好相关。这使我们能够对区块链项目分类提出简化的“按代际分类”的方法，通过一些例子粗略反映现实。尚未实施和部署的项目以斜体显示；一代区块链项目中最重要的特征用粗体显示。

- 第一代：单链，**PoW**，不支持智能合约。示例：比特币（2009）和许多其他无趣的模仿者（莱特币、门罗币……）。
- 第二代：单链，PoW，**支持智能合约**。示例：以太坊（2013 年；2015 年部署），原始形式。
- 第三代：单链，**PoS**，支持智能合约。示例：*未来的以太坊*（2018 年或更晚）。
- 其他第三（3'）代：**多链**，PoS，不支持智能合约，松散耦合。示例：比特股 （2013-2014；使用 DPOS）。
- 第四代：**多链，PoS，智能合约支持，松散耦合。** 示例：*EOS*（2017年；使用 DPOS），*波卡*（2016年；使用 BFT）。
- 第五代：多链，PoS 与 BFT，智能合约支持，**紧密耦合，分片。** 示例：*TON*（2017年）。

虽然并非所有区块链项目都能归属于这些类别中的一个，但大多数都属于这些类别。

2.8.16 改变区块链项目 “基因（genome）” 的复杂性

上述分类定义了区块链项目的 “genome（基因）”。这个基因组非常“僵硬（rigid）”：一旦项目部署并且被很多人使用，就几乎不可能改变它。一种选择是需要一系列硬分叉（这需要社区多数人的批准），即使这样，为了保持以后兼容，改变也需要非常保守（例如，改变虚拟机的语义可能打破现有的智能合约）。另一种选择是创建具有不同规则的新的 “侧链”，并以某种方式将它们绑定到原始项目的一个区块链（或多个区块链）。有人可能会使用现有单链项目的区块链作为外部主链，作为一个基本上是新的独立项目。[28]

我们的结论是项目的 “基因”一旦部署就很难改变。即使从 PoW 开始并计划在未来用 PoS 替换也是相当复杂的。[29]最初设计的项目没有分片而后再加入，这种支持几乎是不可能的。[30]实际上，将智能合约的支持添加到老项目（即比特币）中被认为是不可能的（或者说至少是大多数比特币社区不欢迎的），并最终导致创建一个新的区块链项目——以太坊。

2.8.17 TON 区块链的 “基因（genome）”

因此，如果想要构建可扩展的区块链系统，必须从一开始就仔细选择其基因。如果系统要支持将来在部署时未知的某些其他特定功能，那么它应该从一开始就支持“异构”工作链（具有可能不同的规则）。为了使系统真正可扩展，它必须从一开始就支持分片；只有当系统 “紧密耦合” 时（参见 2.8.14），分片才有意义，这反过来意味着存在主链、快速的区块链间消息系、BFT PoS 的使用等等。

项目	年份	G.	共识	Sm.	Ch.	R.	Sh.	Int.
Bitcoin	2009	1	PoW	否	1			
Ethereum	2003，2005	2	PoW	是	1			
NXT	2014	2+	PoS	是	1			
Tezos	2017,？	2+	PoS	是	1			
Casper	2015,(2017)	3	PoW/Pos	是	1			
BitShares	2013,2014	3’	DPos	否	m	ht.	否	L
EOS	2016,(2018)	4	DPos	是	m	ht.	否	L
Polkadot	2016,(2019)	4	PoS BFT	是	m	ht.	否	L
Cosmos	2017, ?	4	PoS BFT	是	m	ht.	否	L
TON	2017,(2018)	5	PoS BFT	是	m	mx	dyn.	T

表1：一些著名区块链链项目概要。这些列是：项目——项目名称；年份——公告年份和部署年份；g .-代际（见2.8.15）；共识——公式算法（见2.8.3和2.8.4）；Sm.–支持任意代码（智能合约；见2.8.6）；Ch.——单/多链系统（见2.8.2）；R .——异质/同质多链系统（见2.8.8）；SH.——支持分片（见2. 8。12）；Int.-区块链之间的交互，[L(oose)]——松散，或[T(ight)]——紧密（见2.8.14）。

[28] 例如，Plasma项目计划使用以太坊区块链作为其（外部）主链；它与以太坊并没有太多交互，并且所提出和实施的团队与以太坊项目毫无关系。

[29] 截至2017年，以太坊依旧在努力从PoW向PoW+PoS系统转变；我们希望它未来能真正实现PoS。

[30] 2015年，以太坊有提出过几个分片方案；在不破坏以太坊或者创建一个基本上独立的平行项目的情况下，如何实施并部署依旧未知。

当考虑到所有这些影响时，TON 区块链项目做出的大多数设计选择看起来都很自然，并且几乎是唯一可能的选择。

2.9 与其他区块链项目的比较

我们通过尝试在包含现有和建议的区块链项目的地图上找到它TON Blockchain的位置来结束我们对 TON Blockchain 及其最重要和独特特征的讨论。我们使用 2.8 中描述的分类标准，以统一的方式讨论不同的区块链项目，并构建这样的“区块链项目图”。 我们将此地图表示为表 1，然后分别简要讨论几个项目，指出它们可能不适合一般方案的特性。

2.9.1 比特币（Bitcoin）[12]；https://bitcoin.org/。

比特币（2009年）是第一个也是最著名的区块链项目。这是一个典型的第一代区块链项目：单链，使用PoW和“最长分叉胜利”的分叉选择算法，它没有图灵完备脚本语言（但是，支持没有循环的简单脚本）。比特币区块链没有帐户的概念；它使用 UTXO（Unspent Transaction Output）模型。

2.9.2 以太坊（Ethereum）[2] ；https://ethereum.org/。

以太坊（2015年）是第一个支持图灵完备智能合约的区块链。因此，它是典型的第二代项目，也是其中最受欢迎的项目。它使用了工作量证明，但有智能合约和账户。

2.9.3 未来币（NXT） ； https://nxtplatform.org/。

NXT（2014）是第一个基于 PoS 的区块链和货币。它仍然是单链，没有支持智能合约。

2.9.4 Tezos；https://www.tezos.com/。

Tezos（2018 年或更晚）是一个基于 PoS 的单链项目。我们在这里提到它，是因为它的独特功能：它的区块解释函数 ev_block（参见 2.2.6）不是固定的，而是由 OCaml 模块决定的，可以通过将新版本提交到区块链中（并对拟议的变更投票）来升级。通过这种方式，人们将能够通过首先部署 “vanilla” Tezos 区块链，然后逐步改变所需方向的区块解释功能来创建自定义单链项目，而无需任何硬分叉。

这个想法虽然很有趣，但有明显的缺点，它禁止在其他语言（如 C ++）中进行任何优化实现，因此基于 Tezos 的区块链注定会降低性能。我们认为通过发布所提出的区块解释函数 ev_trans 的正式规范可能已经获得了类似的结果，而不用修复特定的实现。

2.9.5 Casper [31]

Casper 是即将推出的以太坊 PoS 算法；2017年（或2018年）逐步部署，倘若成功，将会把以太坊改为单链 PoS 或混合 PoW + PoS 系统，并提供智能合约支持，将以太坊转变为第三代项目。

[31] https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost/

2.9.6 比特股（BitShares）[8]；https://bitshares.org。

BitShares（2014）是基于分布式区块链的交换的平台。 它是一个没有智能合约的异构多块链 DPoS 系统；假设区块链状态适合内存，它只允许一小组预定义的专用交易分片实现其高性能，这些分片可以在 C++ 中有效实现。它也是第一个使用权益委托证明（DPoS）的区块链项目，至少在某些特殊目的下证明了它的可行性。

2.9.7 EOS[5]；https://eos.io。

EOS（2018 或更高版本）是一种提议的异构多链 DPoS 系统，支持智能合约和消息传递的最小支持（在 2.8.14 中描述的意义上仍为松散耦合）。这是以前成功创建 BitShares 和 Steemit 项目的同一团队的尝试，展示了 DPoS 共识算法的优点。可扩展性将通过为需要扩容的项目创建专用工作链（例如，分布式交换可能使用支持一组特殊优化专门交易的工作链，类似于 BitShares 所做的那样），并通过创建具有相同规则的多个工作链（在 2.8.10 中描述的类似的联盟链）来实现。这种可扩展性方法的缺点和局限性已在 2.8.5、2.8.12 和 2.8.14讨论，请参见这三部分，它们更详细地讨论了关于DPoS、分片、工作链之间的交互及其对区块链系统可扩展性的影响。

同时，即使无法“在一个区块链中创建 Facebook”（参见 2.9.13），（例如）EOS 或其他公链，我们认为，对于一些高度专业、弱互动的分布式应用而言，EOS可能成为最方便的平台，类似于 BitShares（去中心化交易）和 SteemIt（分布式博客平台）。

2.9.8 波卡（Polkadot） https://polkadot.io/。

Polkadot（2019 年或更晚）是最精心设计和最详细的多链PoS项目之一；它的开发由以太坊联合创始人之一领导。该项目与我们 TON区块链的路线图最接近。[事实上，我们的“渔夫（fishermen）” 和 “提名人（nominators）” 的术语应归功于Polkadot。]

Polkadot 是一个异构的松耦合多链PoS项目，使用拜占庭容错（BFT）共识生成新区块和主链（可能是外部的——如以太坊区块链）。它还使用超立方体路由，有点像（2.4.19 中所述）TON 的慢速版本。

它的独特之处在于，它不仅可以创建链，还可以创建私有链。这些私有链也可以与其他公链、Polkadot或其他链交互。

因此，Polkadot 可能成为大规模私有链的平台，例如，银行财团可以将其用于快速资金转移，或者大型公司可能使用私有链技术的任何其他用途。

但是，Polkadot 不支持分片，也没有紧密耦合。这与 EOS 类似，有些妨碍它的可扩展性。（Polkadot 可能更好一点，因为 Polkadot 使用 BFT PoS，而不是 DPoS。）

2.9.9 Universa https://universa.io。

我们在这里提到这个不寻常的区块链项目，原因只有一个，是因为它是迄今为止唯一一个通过明确引用类似于我们的无限分片范式（Infinite Sharding Paradigm）的项目（参见 2.1.2）。

它的另一个特点是它绕过了与拜占庭容错相关的所有复杂性，承诺只有项目信任和许可的合作伙伴才会被认可为验证人，因此他们永远不会提交无效的区块。这是一个有趣的决定；然而，它使区块链项目有意中心化，而区块链项目通常要避免中心化（为什么有人会在充满信任的中心化环境中对区块链有所需求？）。

2.9.10 Plasma https://plasma.io。

Plasma（2019？）是另一个以太坊联合创始人（领导）的非传统区块链项目。它旨在减轻以太坊的某些局限性，而不引入分片。本质上，它是一个独立于以太坊的项目，引入了一个（异构的）工作链的层次结构，与顶层的以太坊区块链（用作外部主链）绑定在一起。资金可以从层次结构中顶层的任何区块链转移（从作为根的以太坊区块链开始），以及要做一些描述性工作。然后在子工作链中完成必要的计算（可能需要在树下进一步转发原始作业的部分），将结果传递出去，并获取奖励。实现一致性和验证这些工作链的问题被（支付通道引发的）机制所规避，允许用户单方面将资金从行为不当的工作链中提取到其上属工作链（尽管速度缓慢），并将其资金和工作重新分配给另一个工作链。

通过这种方式，Plasma 可能成为与以太坊区块链绑定的分布式计算平台，类似于“共同计算处理器”。但是，这似乎不是实现真正的通用可伸缩性的一种方法。

2.9.11 专用区块链项目

还有一些专用区块链项目，例如 Filecoin（该系统激励用户提供磁盘空间来存储愿意为其付费的其他用户的文件）、Golem（一种基于区块链的算力租借平台，用于像3D渲染这样的专门应用）或 SONM（另一个类似的算力借贷项目）。这些项目在区块链组织层面上没有引入任何概念上新的东西；相反，它们是特定的区块链应用程序，可以通过运行于通用区块链的智能合约来实现，前提是它可以提供所需的性能。因此，此类项目可能会使用现有或计划中的区块链项目之一作为其基础，例如 EOS、Polkadot 或 TON。如果一个项目需要 “真正的” 可扩展性（基于分片），那么最好使用 TON；如果通过定义一个自己的工作链系列来满足在“联盟”环境中工作的内容，为其目的明确优化，它可能会选择 EOS 或 Polkadot。

2.9.12 TON区块链

TON（Telegram Open Network）区块链（计划 2018 年）是本文档中描述的项目。它只在成为第一个第五代区块链项目——即 BFT PoS多链项目，混合同构 / 异构，支持（可分片）自定义工作链，具有本地分片支持，并且紧密耦合（特别是可以保留所有分片链一致状态同时，快速在分片间转发消息）。因此，它将是一个真正可扩展的通用区块链项目，本质上可以实现在区块链中实现的任何应用程序。当通过 TON 项目的其他组件进行扩充时，其可能性将进一步扩大。

2.9.13 是否可能 “让 Facebook 上链”？

有时，人们声称，可以在区块链上建立 Facebook 规模的分布式应用程序社交网络。通常，一些区块链项目会被引用作为此类应用程序的“主机（host）”。

我们不能说这在技术上不可能。当然，我们需要一个具有真正分片（即 TON）的紧密耦合的区块链项目，以便这样一个大型应用程序不会运转太慢（例如，将一个分片链中用户的消息和

更新传递给另一个分片链中的朋友，延迟应当在合理范围内）。但是我们认为不需要如此，这也永远不会达成，因为价格过高。

让我们把“Facebook 上链”作为思想实验；任何其他类似规模的项目也可以作为一个例子。一旦将 Facebook 上传到区块链中，目前由 Facebook 服务器完成的所有操作将被序列化为某些区块链中的交易（例如，TON 的分片链），并且将由这些区块链的所有验证人执行。如果我们希望每个区块收集至少（立即或最终，如在 DPOS 系统中） 20 个验证人签名，则必须执行每个操作，至少 20 次。类似地，Facebook 的服务器在其磁盘上保存的所有数据将保存在相应分片链的所有验证人的磁盘上（即至少有 20 个副本）。

因为验证人本质上与Facebook 目前使用的那些服务器相同（或者可能是服务器的集群，但这不会影响这个参数的有效性），因为，我们看到在区块链中运行 Facebook 相关的总硬件费用比以传统方式实施的费用至少高 20 倍。

事实上，费用仍然会高得多，因为区块链的虚拟机比运行优化的编译代码的“裸 CPU”慢，并且其存储未针对特定于 Facebook 的问题进行优化。人们可以通过制作适合 Facebook 的一些特殊交易的特定工作链来部分缓解这个问题；这是 BitShares 和 EOS 实现高性能的方法，也可用于 TON 区块链。然而，一般的区块链设计本身仍会产生一些额外的限制，例如，必须将所有操记录为区块中的交易，在 Merkle 树中组织这些交易，计算和检查它们的 Merkle 哈希值，以进行传播等等。

因此，一个保守的估计是，为了验证托管Facebook规模的社交网络的区块链项目，人们需要与 Facebook 现在使用的性能相同的服务器的100倍。有些人将不得不为这些服务器付费，要么是拥有分布式应用程序的公司（想象在每个 Facebook 页面上看到 700 个广告而不是 7 个），要么是其用户。无论哪种方式，这似乎在经济上都不可行。

我们认为，并不是所有内容都应该上链。例如，没有必要将用户照片保留在区块链中；在区块链中记录这些照片的哈希值并将照片保存在分布式的链下存储（例如 Filecoin 或 TON 存储）中将是一个更好的主意。这就是为什么 TON 不仅仅是一个区块链项目，而是如第 1 章和第 4 章所述以 TON 区块链为中心的几个要素（TON P2P 网络，TON 存储，TON 服务）的集合。

3 TON网络

任何区块链项目都不仅需要规范区块格式和区块验证标准，还需要一个用于传播新区块、发送和收集交易候选人等的网络协议。换句话说，每个区块链项目都必须建立专门的P2P网络。因为区块链项目通常需要去中心化，所以这个网络必须是点对点的，我们不能依赖中心化的服务器群并使用传统的客户端——服务器架构，例如，传统的在线银行应用程序。即使轻客户端（如加密货币轻钱包智能手机应用）必须以类似客户端——服务器的方式连接到全节点，假定用来连接全节点的协议足够标准化，那么如果先前连接的全节点发生故障，轻客户端还可以再切换另一个全节点。

虽然可以很容易地满足单链区块链（如比特币或以太坊）的网络需求（需要构建一个 “随机” 的点对点覆盖网络，并通过一个gossip协议传播所有新区块和交易候选人），TON区块链等多链项目的要求更高（例如，可以订阅只有一些而不需要去不分片链的更新）。因此，TON区块链和TON项目的网络部分部分将在这里进行讨论。

另一方面，一旦所需要用来支持TON区块链的更复杂网络协议到位，结果是它们很容易用于不一定与TON区块链的当前直接需求相关的目的，因此它们会在TON生态系统里提供更多的可能性和灵活性。

3.1 抽象数据报表网络层（简称 ADNL）

构建TON网络协议的基础是（TON）抽象（数据表）网络层。它使所有节点能够假设某些由256位 “抽象网络地址” 表示的“网络标识”，并且可以使用这些256位网络地址进行通信（给各方发送数据报表作为第一步）以识别发送方和接收方。人们不必担心IPv4或IPv6地址、UDP端口号等；它们会被抽象网络层隐藏。

3.1.1 抽象网络地址。

一个*抽象网络地址*或*抽象地址*，或简称*地址*，是256位整数，本质上等于256位ECC公钥。公钥可以任意生成，从而根据节点的喜好，创建任意多的不同网络身份。然而，必须知道相应的私钥，才能接收（和解密）用于这种地址的消息。
实际上，地址本身并不是公钥；相反，它是序列化的TL对象（参见 2.2.5）的 256 位哈希（Hash = sha256），可以根据其构造函数（前四个字节）描述几种类型的公钥和地址。在最简单的情况下，这个序列化的TL对象只包含一个4字节的魔术数字和一个256位的椭圆曲线加密（ECC）公钥；在这种情况下，地址将等于这个36字节结构的哈希值。然而，可以使用2048位RSA密钥或任何其他公钥加密方案来取代。
当节点知晓另一个节点的抽象地址时，它还必须接收其 “原象”（即序列化的TL对象，其哈希值等于那个抽象地址），否则它将无法加密并将数据报表发送到该地址。

3.1.2 较低级别的网络。UDP的实现。

从几乎所有TON网络组件的角度来看，唯一存在的是能够（不可靠）将数据报表从一个抽象地址发送到另一个抽象地址的网络（the Abstract Datagram Networking Layer；ADNL）。原则上，抽象数据报网络层（ADNL）可以在不同的现有网络技术上实现。但是，我们将在IPv4 / IPv6网络（例如 Internet 或 Intranet）中通过UDP实现它，如果UDP不可用，则使用可选的TCP回退。

3.1.3 基于UDP最简单的ADNL的例子。

最简单地例子是，从发送者的抽象地址向任何其他抽象地址（具有已知的原像）发送数据报表可以用以下方式实现。
假设发送者以某种方式知道拥有目的地抽象地址的接收方的IP地址和UDP端口，也知道接收方和发送方都使用的抽象地址都来源于256位ECC公钥。
在这种情况下，发送者只是增加了将要通过ECC签名（配合它的私钥完成）和它的源地址（如果接收者尚不知道源地址的原像，则是发送原像） 发送的数据报表。结果用接收者的公钥加密，嵌入UDP数据报并发送到接收者的已知IP和端口。由于UDP数据报表的前256位包含接收者的抽象地址，因此接收者可以识别应使用哪个私钥来解密数据报的其余部分。只有在那之后发送者的身份才能被披露。

3.1.4 发件人的明文地址是一种安全性较低的方式。

有时，使用安全性较低的方式就足够了，即接收者和发送者的地址以明文形式保存在 UDP 数据报中的时候； 发送方的私钥和接收方的公钥组合在一起，使用ECDH（椭圆曲线 Diffie-Hellman）生成一个 256 位的共享密钥，随后使用该共享密钥，还有一起被使用的是未加密部分中包含的随机256位随机数，以导出用于加密的AES密钥。例如，可以通过在加密之前将原始明文数据的哈希值连接到明文来提供完整性。
这种方法的优点是，如果预计有多份数据报表在两个地址之间切换，则只能计算一次共享密钥，然后进行高速缓存； 然后，加密或解密下一个数据报表将不再需要较慢的椭圆曲线运算。

3.1.5 通道和通道标识符。

在最简单的情况下，携带嵌入式TON ADNL数据报表的UDP数据报表的前256位将等于接收者的地址。然而，总得来说它们构成了*通道标识符*。有不同类型的通道。其中一些是点对点的；创建的双方希望在未来交换大量数据，并生成一个共享秘钥。实现方式包括通过交换几个如3.1.3或3.1.4所述的加密数据包，或通过运行经典或椭圆曲线Diffie-Hellman（如果需要额外的安全性），或者只通过一方生成生成随机共享密钥并将其发送给另一方。
在此之后，一个通道识别码由共享秘密与一些附加数据（如发送者和接收者的地址）组合而成，例如通过哈希算法，并且该标识符被用作UDP数据报表的前256位，这些数据报表借助该共享私钥加密。

3.1.6 通道作为一种隧道标识符。

通常，“通道” 或 “通道标识符” 仅选择处理接收器已知的入站UDP数据报表的方式。如果通道是接收者的抽象地址，那么处理方法就如3.1.3或3.1.4所述；如果通道是 3.1.5 中讨论的已建立的点对点通道，则处理过程在于借助共享私钥对数据报表进行解密，如*loc. cit.*等中解释的那样。
特别是，当直接接收者简单地将所接收的消息转发给其他人——实际接收者或另一个代理时，通道标识符实际上可以选择 “隧道”（tunnel）。一些加密或解密步骤（让人联想到 “onion routing”[6] 甚至 “garlic routing”[32]）可能会在此过程中完成，而另一个通道标识符可能会用于重新加密的转发数据包（例如，点对点通道可以用来将数据包转发给路径上的下一个接收者）。
通过这种方式，可以在TON抽象数据报网络层的层面上添加一些对 “tunneling” 和 “proxying” 的支持——与*TOR*或*I²P*项目提供的类似——而不会影响到所有高层TON抽象数据报表网络协

[32] https://geti2p.net/en/docs/how/garlic-routing

议的功能，所有高层TON抽象数据报表网络协议对此添加而言都是不可知的。这是利用了*TON Proxy*服务（参见 4.1.11）。

3.1.7 零通道和bootstrap问题。

通常，TON ADNL节点将具有一些 “相邻节点列表（neighbor table）”，其包含关于其他已知节点的信息，例如它们的抽象地址及其原像（即公钥）及其IP地址和UDP端口。然后它将通过使用从这些已知节点获取的信息作为特殊查询的答案逐渐扩展此表，并且有时会有修剪的记录。
但当TON ADNL节点刚启动时，可能发生此节点不知道其他节点的情况，只知道节点的IP地址和UDP端口，而不知道其他节点的抽象地址。这种情况会在如下时候发生：例如，如果轻节点无法访问任何先前缓存的节点以及任何硬编码到软件中的节点，并且必须要求用户输入节点的IP地址或DNS域，然后通过DNS解决 。
在这种情况下，节点将数据包发送到相关节点的特殊 “零通道”。这不需要知道接收者的公钥（但消息仍应包含发件人的身份和签名），因此消息无需加密即可传输。它通常应该仅用于获取接收器的身份（可能是专门为此创建的一次性身份），然后以更安全的方式开始通信。
一旦知道至少一个节点，就很容易通过更多条目填充 “相邻节点列表（neighbor table）” 和 “路由表（routing table）”，从发送到已知节点的特殊查询答案中知道它们。
并非所有节点都需要处理发送到零通道的数据报表，但那些用作bootstrap轻客户端的节点应该支持这个功能。

3.1.8 ADNL上类似TCP的流协议。

ADNL是基于256位抽象地址的不可靠（小规模）数据报表协议，可用作更复杂网络协议的基础。例如可以使用ADNL作为IP的抽象替代物来构建类似TCP的流协议。但是TON项目的大多数组件都不需要这样的流协议。

3.1.9 RLDP或ADNL上的可靠大数据报表协议。

使用基于ADNL的可靠的任意大小的数据报表协议（称为 RLDP）代替类似 TCP 的协议。例如，可以使用这种可靠的数据报表协议将RPC查询发送到远程主机并通过它们接收回复（参见 4.1.5）。

3.2 TON DHT：类似Kademlia的分布式哈希表

TON分布式哈希表（DHT）在TON项目的网络部分起着至关重要的作用，用于定位网络中的其他节点。例如，一个想要将交易提交到分片链的客户端可能想要找到该分片链的验证人或校验者，或者至少某个可能将客户端的交易中继到校验者的节点。这可以通过在TON DHT中查找特殊键来完成。TON DHT的另一个重要应用是它可以用来快速填充新节点的相邻节点列表（参见 3.1.7），只需查找随机密钥或新节点的地址即可。如果一个节点对其入站数据报表使用代理和隧道，则它在TON DHT中发布隧道标识符及其入口点（例如，IP地址和UDP端口）；紧接着希望将数据报发送到该节点的所有节点将首先从DHT获得联系信息。
TON DHT是*类似Kademlia的分布式哈希表*系列的成员[10]。

3.2.1 TON DHT的密钥。

TON DHT的密钥是简单的256位整数。在大多数情况下，它们被计算为TL序列化对象的SHA256值（参见2.2.5），称为*密钥的原像*或*密钥的描述*。在某些情况下，TON网络节点的

抽象地址（参见3.1.1）也可以用作TON DHT的密钥，因为它们也是256位的，它们也是TL序列化对象的哈希值。例如，如果节点不害怕公布其IP地址，任何知道其抽象地址的人都可以找到它，方法是简单地将该地址作为DHT中的密钥来查找地址。

3.2.2 DHT的值。

分配给这些256位密钥的*值*基本上是有限长度的任意字节字符串。这种字节字符串的解释由相应密钥的原像决定；通常通过查找密钥的节点和存储密钥的节点一起才能知道密钥的值。

3.2.3 DHT的节点。半永久性网络身份。

TON DHT的密钥值映射保留在DHT的节点上——基本上是TON网络的所有成员。为此，除了3.1.1中描述的任何数量的短暂和永久抽象地址之外，TON网络的任何节点（可能除了一些非常轻的节点）至少具有一个 “半永久地址”，也将其标识为TON DHT的成员。这个*半永久地址*或*DHT地址*不应该经常更改，否则其他节点将无法找到他们正在寻找的密钥。如果节点不想显示其 “真实” 标识，则它会生成一个单独的抽象地址，仅用于参与DHT。但是，此抽象地址必须是公共的，因为它将与节点的IP地址和端口相关联。

3.2.4 Kademlia距离。

现在我们有256位密钥和256位（半永久）节点地址。我们在 256 位序列集上引入所谓的*XOR距离*或*Kademlia距离*d_K，由下式给出

$$d_K(x, y) := (x \oplus y) \quad \text{解释为一个未签名的256位整数} \qquad (25)$$

这里x⊕y表示相同长度的两位序列的按位异或（bitwise eXclusive OR）（或 XOR）。
Kademlia距离在所有256位序列的集合2^{256}上引入度量。尤其是，如果当且仅当$x = y$, $d_K(x,y) = d_K(y, x)$, *以及*$d_K(x, z) \le d_K(x, y) + d_K(y, z)$时，得出结论$d_K(x,y) = 0$。另一个重要特性是*在距x的任何给定距离处只有一个点*：$d_K(x, y) = d_K(x, y')$，暗含了$y = y'$。

3.2.5 类似Kademlia的DHTs和TON DHT。

如果期望将Kademlia上的最近节点*s*的密钥值*K*保持为*K*（即，从其地址到K的带有最小Kademlia距离最小的*s*节点。），那么我们说带有256位密钥和256位节点地址的分布式哈希表（DHT）是*类似Kademlia的DHT*。
这里是一个小参数，比如当s = 7时提高DHT的可靠性（如果我们只将密钥保存在一个节点上——最接近*K*的那个节点，当那个节点离线时，那个密钥的值就会丢失）。
根据这个定义，TON DHT是类似Kademlia的DHT。它是通过3.1中描述的ADNL协议实现的。

3.2.6 Kademli路由表。

参与Kademlia-like DHT的每个节点通常都维护一个Kademlia路由表。在TON DHT的情况下，它由n = 256个桶（buckets）组成，编号从0到n-1。第*i*-th个桶将包含关于一些已知节点的信息（“最佳” 节点的一个固定数量*t*，可能还有一些额外的候选人），这些已知节点位于距离节点地址a的Kademli 距离2^i到$2^{i+1}-1$之间[33]。这个信息包括它们的（半永久）地址、IP 地址和UDP端口，以及一些可用性信息，如最后一次ping的时间和延迟。

[33] 如果存储桶中有足够多的节点，则可以进一步细分为例如八个子桶，这取决于Kademlia距离的前四位。这会加快DHT查找速度。

当Kademlia节点了解其他Kademlia节点作为查询结果时，它将这个查询结果包含在其路由表的合适桶中，首先作为候选人。然后，如果该桶中的一些 “最佳” 节点失败（例如，长时间不响应ping查询），则可以用一些候选人替换它们。通过这种方式，Kademlia路由表保持填充状态。
来自Kademlia路由表的新节点也包含在3.1.7中描述的ADNL相邻节点列表中。如果经常使用来自Kademlia路由表的桶中的 “最佳” 节点，那么从3.1.5中中描述的意义上来说，可以建立一个通道以便于加密数据报表。
TON DHT的一个特殊功能是它尝试选择往返延迟最小的节点作为Kademlia路由表的桶的 “最佳” 节点。

3.2.7 （Kademlia网络查询。）Kademlia节点通常支持以下网络查询：

- PING——检查节点可用性。
- STORE (*key,value*)——要求节点将*value*保持为密钥*key*的值。对于TON DHT，STORE查询稍微复杂一些（参见 3.2.9）。
- FIND_NODE (*key,l*)——要求节点将Kademlia最近的已知节点*l*（从其Kademlia路由表）返回到密钥。
- Find_VALUE (*key, l*)——与上面相同，但如果节点知道与密钥*key*对应的值，则只返回到该值。

当任何节点想要查找密钥*K*的值时，它首先创建*s*'节点的集合*S*（对于*s*'的一些小值，比如*s*'= 5），*S*是已知节点中就Kademlia距离而言最接近*K*的（例如它们来自Kademlia路由表）。然后向它们中的每一个发送Find_*Value*查询，并且在它们的答案中提到的节点也包含在*S*中。如果之前没有这样做，那么来自*S*的*s*'节点（最接近于K）也被发送 Find_Value查询，并且该过程一直持续到找到值或集合S停止增长为止。这是就Kademlia距离而言，距离*K*最近的节点的一种 “集束搜索（beam search）”。
如果要设置某个密钥*K*的值，使用*Find_Node*查询而不是*Find*_Value查询，则对 *s*'≥*s* 运行相同的过程，用来查找距离K最近的节点。之后，STORE查询发送给所有这些节点。
在实现类似Kademlia的DHT时有一些不太重要的细节（例如，任何节点都应该查找最近的节点*s*，比如每小时一次，并通过STORE查询重新发布所有存储的密钥）。我们暂时会忽略它们。

3.2.8 启动Kademlia节点。

当Kademlia节点联网时，它首先通过查找自己的地址来填充其Kademlia路由表。在此过程中，它标识出距离自身最近的节点*s*。它可以从这些节点中下载所有已知的(key,value)对，以填充它的部分DHT。

3.2.9 在TON DHT中存储值。

在TON DHT中存储值与一般像Kademlia的DHT略有不同。当有人希望存储一个值时，她必须不仅要向STORE查询提供密钥*K*本身，还要提供其*原像*——例如TL序列化的字符串（开头带有几个预定义的TL构造函数的其中一个），原像包含对字符串的 “ 描述 “。稍后由节点保存此密钥描述以及密钥和值。
密钥描述了所存储对象的 “类型字段”，其 “所有者” 以及未来更新时的 “更新规则”。所有者通常通过密钥描述中包含的公钥来标识。如果包含公钥，通常只接受由相应私钥签名的更新。存储对象的 “类型字段” 通常只是一个字节字符串。然而，在某些情况下，它可能更复杂——例如，输入隧道描述（参见 3.1.6）或节点地址的集合。

“更新规则” 也可以不同。在某些情况下，假使新值由所有者签名（签名必须保留为值的一部分，之后由其他任何节点在获取密钥值之后检查），更新规则只允许用新值替换旧值。在其他情况下，旧值会以某种方式影响新值。例如，它可以包含序列号，只有在新序列号较大时才会覆盖旧值（以防止重新操作的攻击）。

3.2.10 TON DHT中的分布式 “流追踪器” 和 “网络兴趣小组”。

另一个有趣的情况是，当值包含节点列表时——可能带有其IP地址和端口，或仅带有其抽象地址——并且 “更新规则” 在于将请求者包括在此列表中，假使她可以确认她的身份。
该机制可以用于创建分布式 “流追踪器”，在“流追踪器”上，对某个 “流”（文件）感兴趣的所有节点可以找到对相同的流感兴趣或已经具有副本的其他节点。
TON存储（参见 4.1.8）使用该技术来查找具有所需文件副本的节点（例如，分片链或旧块的状态的快照）。但是，更重要的用途是创建 “覆盖组播子网” 和 “网络兴趣小组”（参见 3.3）。这个想法是只有一些节点对特定分片链的更新感兴趣。如果分片链的数量变得非常大，甚至找到对同一分片感兴趣的一个节点可能会变得复杂。这种 “分布式流追踪器” 提供了一种查找这些节点的便捷方法。另一种选择是从验证人请求它们，但这不是一种可扩展的方法，验证人可能选择不响应来应对任意未知节点的查询。

3.2.11 Fall-back keys

到目前为止所描述的大多数 “密钥类型”，用它们的TL描述来说具有额外的32位整数字段，通常等于零。然而，如果无法从TON DHT中检索或更新通过哈希描述获得的密钥，则增加该字段中的值，并进行新的尝试。以这种方式，通过在受到攻击的密钥附近创建许多抽象地址并控制相应的DHT节点，我们不能 “捕获” 和 “审查” 密钥（即执行密钥保留的攻击）。

3.2.12 定位服务。

某些服务位于TON网络中，并通过如3.1中描述的TON ADNL（基于其构建的高级协议）提供，这些服务可能希望在某处发布其抽象地址，以便其客户知道在哪里找到它们。
然而，在TON区块链中发布服务的抽象地址可能不是最好的方法，因为抽象地址可能需要经常更改，也因为为了可靠性或负载平衡的目的而提供多个地址是有意义的。
一种替代方法是将公钥发布到TON区块链中，并使用一个特殊的DHT密钥，表明公钥作为TL描述字符串中的 “所有者”（参见 2.2.5），用来发布 服务的抽象地址的最新列表。这是利用Ton服务的方法之一。

3.2.13 定位TON区块链账户的所有者。

在大多数情况下，TON区块链帐户的所有者不希望与抽象网络地址相关联，尤其是 IP 地址，因为这可能会侵犯他们的隐私。但是，在某些情况下，TON区块链帐户的所有者可能希望发布可以联系她的一个或多个抽象地址。
典型的情况是TON支付 “闪电网络” 中的节点（参见 5.2），也就是即时加密货币转账平台。公共TON支付节点可能不仅希望与其他对等节点建立支付通道，而且还要发布抽象网络地址，这个地址可用来在后续沿着已建立的支付通道转账支付时联系它。
一种选择是在创建支付通道的智能合约中包含抽象网络地址。更灵活的选择是在智能合约中包含公钥，然后按照 3.2.12 中的说明使用DHT。
最自然的方法是使用控制TON区块链中帐户的相同私钥，用来签署和发布TON DHT中与该帐户关联的抽象地址的更新。这几乎与3.2.12中描述的方式相同；但是，使用的DHT密钥需要一

个特殊的密钥描述，这个私钥描述只包含*account_id*本身，等同于 “account description” 的SHA256值，该值包含帐户的公钥。包含在此DHT密钥值中的签名也将包含帐户描述。
通过这种方式，用于定位TON区块链账户的一些所有者的抽象网络地址这一机制得以建立。

3.2.14 找到抽象地址的位置。

请注意，TON DHT虽然通过TON ADNL实现，但它本身也被TON ADNL用于多种用途。
其中最重要的是从256位抽象地址开始定位节点或其联系人数据。这是必要的，因为即使没有提供其他信息，TON ADNL也应该能够将数据报表发送到任意的256位抽象地址。
为此，在DHT中将256位抽象地址简单地作为密钥被查找。具有该地址的节点（即，使用该地址作为公共半永久DHT地址）被搜索到，在这种情况下，可以知晓其IP地址和端口；或者，可以检索输入隧道描述作为所讨论的密钥的值，由正确的私钥签名，在这种情况下，该隧道描述将用于将ADNL数据报表发送到预期的接收者。
请注意，为了使抽象地址 “public”（可从网络中的任何节点访问），其所有者必须将其用作半永久性DHT地址，或者发布（在DHT中，关键值等于所考虑的抽象地址）带有另一个公共抽象地址（例如，半永久地址）的隧道描述输入，这个公共抽象地址作为隧道的入口点。另一种选择是简单地发布其IP地址和UDP端口。

3.3 覆盖网络（Overlay Networks） 和组播消息（ Multicasting Messages）

在像TON区块链这样的多链系统中，甚至完整节点通常也只对获取某些分片链的更新（即新区块）感兴趣。为此，必须在TON网络内部建立一个特殊的覆盖（子）网络，位于3.1 中讨论的ADNL协议之上，每个分片链都有一个。
因此，构建对任意覆盖子网的需求在增加，构建任意覆盖子网对任何愿意参与的节点都开放。基于ADNL的特殊gossip协议将在这些覆盖网络中运行。特别是，这些gossip协议可用于在这种子网内传播（广播）任意数据。

3.3.1 覆盖网络。

*覆盖（子）网络*简单地是在一些较大网络内实现的（虚拟）网络。 通常，只有较大网络的一些节点参与覆盖子网，并且这些节点之间只有一些 “links”，不管是实体的还是虚拟的，才是覆盖子网的一部分。
通过这种方式，如果包含网络被表示为图形（在数据报表网络的情况下可以使用完整的图形，例如 ADNL，这样任何节点都可以轻松地与任何其他节点通信），覆盖子网络是这个图形的*子图*。
在大多数情况下，覆盖网络使用基于较大网络的网络协议构建的一些协议来实现。它可以使用与较大网络相同的地址，或使用自定义地址。

3.3.2 TON中的覆盖网络。

TON中的覆盖网络建立在3.1中讨论的ADNL协议之上；它们也使用256位ADNL抽象地址作为覆盖网络中的地址。每个节点通常选择它的一个抽象地址加倍作为在覆盖网络中的地址。
与ADNL相比，TON覆盖网络通常不支持将数据报表发送到任意其他节点。相反，在一些节点之间建立一些 “半永久链接”（相对于所考虑的覆盖网络称为 “相邻节点”），并且消息通常沿着这些链接（即从一个节点到其中一个相邻节点）被转发。以这种方式，TON覆盖网络是ADNL网络的（完整）图形内的（非完整）子图。

可以使用专用的对等ADNL通道来实现与TON覆盖网络中的相邻节点的链接（参见 3.1.5）。覆盖网络的每个节点维护一份相邻节点的列表（与覆盖网络有关），包含它们的抽象地址（用于在覆盖网络中识别它们）和一些链接数据（例如，用于与它们通信的ADNL通道）。

3.3.3 私人和公共覆盖网络。

一些覆盖网络是*公共的*，这意味着任何节点都可以随意加入它们。其他是*私有的*，意味着只允许某些节点被接纳（例如，那些可以证明其身份为验证人的节点）。一些私有覆盖网络甚至可以为 “general public” 所知。这种覆盖网络的信息仅对某些可信节点可用；例如，它可以使用公钥加密，并且只有具有相应私钥副本的节点才能解密此信息。

3.3.4 中心化控制的覆盖网络。

一些覆盖网络是*中心化控制*的，由一个或多个节点控制，或由一些广为人知的公钥所有者控制。其他的是*去中心化的*，这意味着没有特定的节点负责它们。

3.3.5 加入覆盖网络。

当一个节点想要加入一个覆盖网络时，它首先必须知道它的256位*网络标识符*，通常等于覆盖网络*描述*的SHA256——这是一个序列化对象的TL语言（参见2.2.5），它可能包含例如，覆盖网络的中心权威（即其公钥，可能是抽象地址[34]）；具有覆盖网络名称的字符串；如果这是与分片相关的覆盖网络，也可以是TON 区块链分片标识符，等等。
有时可以从网络标识符开始恢复覆盖网络描述，只需在TON DHT中查找即可。在其他情况下（例如，对于私有覆盖网络），我们必须获得网络描述以及网络标识符。

3.3.6 找到覆盖网络的一个成员的位置。

在节点获知其想要加入的覆盖网络的网络标识符和网络描述之后，它必须至少找到属于该网络的一个节点。
对于不想加入覆盖网络但只想与之通信的节点，也需要这样做；例如，可能存在专用于收集和传播特定分片链的交易候选人的覆盖网络，并且客户端可能想要连接到该网络的任何节点以建议建议。
用于定位覆盖网络的成员的方法在该网络的描述中定义。有时（特别是对于专用网络），我们必须已经知道能够加入的成员节点。在其他情况下，一些节点的抽象地址包含在网络描述中。更灵活的方法是在网络描述中仅指示负责网络的中心机构，然后通过由该中央机构签名的某些DHT密钥的值来获得抽象地址。
最后，真正去中心化的公共覆盖网络可以使用3.2.10中描述的 “分布式流追踪器” 机制，也可以在TON DHT的帮助下实现。

3.3.7 找到覆盖网络的更多成员。创建链接。

一旦找到了覆盖网络的一个节点，可以向该节点发送一个特殊的查询，要求提供其他成员的列表，例如，被查询节点的相邻节点，或者随机选择其中一个节点。
这使得加入成员能够通过选择一些新知晓的网络节点并建立到它们的链接（例如3.3.2中概述的专用ADNL点对点通道）来填充关于覆盖网络的 “邻接（adjacency）” 或 “相邻节点列表（neighbor list）”。之后，将向所有相邻节点发送特殊消息，表明新成员已准备好在覆盖网络中工作。相邻节点包含它们在相邻节点列表中的新成员的链接。

[34] 或者，如3.2.12中所述的那样，抽象地址可以存储在DHT中。

3.3.8 维护相邻节点列表。

覆盖网络节点必须时不时更新它的相邻节点列表。一些相邻节点，或者至少是它们的链接（通道）可能会停止响应；在这种情况下，必须将这些链接标记为 “暂停（suspended）”，必须做一些重新连接到此类相邻节点的尝试，并且如果这些尝试失败，则必须销毁链接。
另一方面，每个节点有时会从随机选择的相邻节点请求它的相邻节点列表（或做部分随机选择），方法是通过向它添加一些新发现的节点，并删除一些旧节点，这个过程可以是随机的，也可以取决于它们的响应时间和数据报丢失统计信息。

3.3.9 覆盖网络是一个随机子图。

通过这种方式，覆盖网络是ADNL网络里的随机子图。如果每个顶点的度数至少为3（例如，如果每个节点连接到至少三个相邻节点），则该随机图会以几乎为1的概率这种大家都知道的方式*连接*。更准确地说，带有n个顶点随机图的断链的概率是极小的，如果$n \geq 20$，则可以完全忽略这个概率（当然，当不同分片间的节点没有机会互相了解时，这不适用于全局网络分区的情况）。在另一方面，如果 n 小于 20，就足以要求每个顶点有例如至少十个相邻节点。

3.3.10 TON覆盖网络经过优化，可降低延迟。

TON覆盖网络优化了由下述之前的方法生成的 “随机” 网络图。每个节点都尝试保留至少三个带有最小往返时间的相邻节点，并很少改变这个 “快相邻节点” 的列表。同时，它还至少有三个完全随机选择的 “慢相邻节点”，因此覆盖网络图总是包含一个随机子图。这是保持连接性并防止将覆盖网络分成几个未连接的区域子网所必需的。至少三个带有中间往返时间的“中间相邻节点”也会被选择和保留，它们受限于一定的常数（实际上是快相邻节点和慢相邻节点的往返时间的一种函数）。
通过这种方式，覆盖网络的图形仍然保持足够的随机性用来连接，但是对降低延迟性和提高吞吐量做了优化。

3.3.11 覆盖网络中的gossip协议。

覆盖网络习惯于运行所谓的一种*gossip协议*，同时让每个节点仅与其相邻节点交互，gossip协议得以实现一些全局目标。例如，有一些gossip协议用来构建一个大致清单，这个清单包括（不是很大的）的覆盖网络的所有成员，或者用来计算一个（任意大的）覆盖网络成员的大概数，在每个节点上只使用有限的内存量（细节参见 [15, 4.4.3] or [1]）。

3.3.12 覆盖网络作为广播域。

在覆盖网络中运行的最重要的gossip协议是*广播协议*（broadcast protocol），旨在将广播消息传播给所有其他节点，这些广播消息由网络的任何节点或者可能是指定的发送方的一个节点生成。
实际上有几种广播协议，针对不同的用例进行了优化。最简单的一种是接收新广播消息并将它们中继到所有相邻节点，这些相邻节点还没有独立发送该消息副本过。

3.3.13 更复杂的广播协议。

某些应用可能需要更复杂的广播协议。例如对于发送量级比较大的广播消息，向相邻节点发送不是新接收的消息本身，而是发送其哈希（或新消息的哈希集合）是合乎情理的。在知道了先

前看不见的消息的哈希之后，相邻节点可以自己请求消息，例如使用3.1.9中讨论的可靠的大数据报表协议（RLDP）进行传输。这样，新消息将仅从一个相邻节点中下载。

3.3.14 检查覆盖网络的连接性。

如果存在一个已知节点（例如覆盖网络的 “所有者” 或 “创造者”）必须在这个覆盖网络中，那么这个覆盖网络的连接性可以被检查。然后，所讨论的节点时不时地广播包含当前时间、序列号及其签名的短消息。任何其他节点可以确定它仍然连接到覆盖网络，如果它不久前已经收到这样的广播。该协议可以扩展到几个众所周知的节点的用例；例如它们都将发送此类广播，并且所有其他节点都想要从超过一半的众所周知的节点接收广播。
在用于传播特定分片链的新区块（或仅新区块头）的覆盖网络这一例子中，节点用来检查连接线的一个好方法是追踪截至目前接收的最新区块。因为区块通常每五秒生成一次，如果说超过30秒都没有收到新区块，则该节点可能已经与覆盖网络断开连接。

3.3.15 流媒体广播协议。

最后，还有一个用于TON覆盖网络的*流媒体广播协议*，例如用于在某些分片链（“shardchain任务组”）的验证人中宣传候选区块，当然验证人也会为此目的创建一个私有覆盖网络。可以使用相同的协议将新的分片链区块传播到该分片链的所有完整节点。
该协议已在2.6.10中概述：新的（大）广播消息被分成例如*N*个一千字节的块；通过诸如Reed-Solomon或喷泉码（例如，RaptorQ代码 [9] [14]）之类的纠删码将这些块的序列增加到$M \geq N$的块，并且这些*M*块被流式传输到所有相邻节点，按区块编号升序排列。参与的节点收集这些块直到它们可以恢复原始的大消息（为此一个节点必须成功接收至少*N*个区块），然后指示它们的相邻节点停止以流的方式发送新块，因为现在这些节点可以自己生成后续的块，并拥有原始消息的副本。这些节点继续以流的方式生成后续块并将它们发送到它们的相邻节点，除非相邻节点反过来表明不再需要发送。
以这种方式，节点在进一步传播之前不需要完整地下载大消息。这可以最大限度地减少广播延迟，尤其是与3.3.10。中描述的优化结合使用时。

3.3.16 基于现有的覆盖网络构建新的覆盖网络。

有时人们不想从头开始构建覆盖网络。相反已知一个或多个先前存在的覆盖网络，并且预计新覆盖网络的成员资格与这些覆盖网络的组合成员资格显著重叠。
当一个TON分片链被分成两个，或者两个兄弟分片链被合并为一个时，就会出现一个重要的例子（参见2.7）。在第一种情况下，必须为每个新的分支链构建用于将新块传播到完整节点的覆盖网络；然而，可以预期每一个新的覆盖网络都包含在原始分片链的区块传播网络中（并且包括其大约一半的成员）。在第二种情况下，用于传播合并分片链的新块的覆盖网络，将大概包含两个覆盖网络成员的并集，这两个覆盖网络与两个并联的兄弟链有关。
在这种情况下，新覆盖网络的描述可以包含对相关现有覆盖网络列表的显式或隐式引用。希望加入新覆盖网络的节点可以检查它是否已经是其中一个现有网络的成员，并且在这些网络中查询它们的相邻节点是否也对新网络感兴趣。在肯定答复的情况下，可以为此类相邻节点建立新的点对点通道，并且这些通道可以包含在相邻节点列表中用于新覆盖网络。
这种机制并不完全取代3.3.6和3.3.7中描述的一般机制；相反，它们都是并行运行的，用于填充相邻节点列表。这是为了防止无意中将新的覆盖网络分成几个未连接的子网。

3.3.17 带有覆盖网络的覆盖网络。

另一个有趣的案例是*TON支付*（*TON Payments*）的实施（即用于即时链下价值转移的 “闪电网络”；参见 5.2）。在这种情况下，首先构建包含 “闪电网络” 的所有传输节点的覆盖网络。

然而，其中一些节点已在区块链中建立了支付通道；除了是3.3.6、 3.3.7和3.3.8中描述的一般覆盖网络算法选择的任何 “随机” 相邻节点之外，它们必须始终是该覆盖网络中的相邻节点。这些相邻节点（带有已建立的支付通道）的 “永久链接” 用于运行特定的闪电网络协议，从而有效地在包含的（最常连接的）覆盖网络内创建覆盖子网（如果出现问题，不一定会连接）。

4 TON 服务和应用

我们已经详细讨论了 TON 区块链和 TON Network 技术。 现在我们来解释一些将它们组合在一起以创建各种服务和应用程序的方法，并讨论 TON 项目本身从一开始或稍后将会提供给大家的一些服务。

4.1 TON 服务实施策略

我们首先讨论如何在 TON 生态系统内实现不同的区块链和网络相关的应用程序和服务。 首先按顺序进行简单分类：

4.1.1 应用和服务。

我们将交替使用 “application （申请）” 和 “service （服务）”。 但它们存在一种微妙且有些模糊的区别：应用程序通常直接向人类用户提供某些服务，而服务通常被其他应用程序和服务利用。 例如，TON Storage 是一项服务，因为它旨在为其他应用程序和服务保留文件，即使人类用户也可以直接使用它。 一个假设的 “区块链中的 Facebook”（参见 2.9.13）或 Telegram messenger，如果通过 TON Network 提供（即实施为 “ton-service”；参见 4.1.6），则更像是一个应用程序。即使某些 “机器人(bots)” 可能在没有人为干预的情况下自动访问它。

4.1.2 应用的位置：链上，链下或混合。

为 TON 生态系统设计的服务或应用程序需要保留其数据并将数据处理到某处。 这就导致了以下的应用程序（和服务）的分类：

- **链上应用程序**（参见 4.1.4）：所有数据和处理都在 TON Blockchain中。
- **链下应用程序**（参见 4.1.5）：所有数据和处理都在 TON 区域链之外，在通过 TON Network 提供的服务器上。
- **混合应用程序**（参见 4.1.7）：部分（但不是全部）数据和处理都在 TON Blockchain中；其余的是通过 TON Network 提供的链下服务器。

4.1.3 中心化：中心化的和去中心化的或分布式应用程序。

另一个分类标准是应用程序（或服务）是依赖于中心化服务器集群，还是真正的 “分布式的（参见 4.1.9）。 所有链上应用程序都自动去中心化和分发。 链下和混合应用可能表现出不同程度的中心化。现在让我们更详细地考虑上述可能性。

4.1.4 纯粹的 “链上” 应用程序：驻留在区块链中的分布式应用程序或 “DApps”。

4.1.2 中提到的一种可能的方法是在 TON Blockchain中完全部署 “分布式应用程序”（通常简称为 “DApp”），作为智能合约或智能合约的集合。所有数据将作为这些智能合约的永久状态的

一部分保留，并且与项目的所有交互将通过发送到这些智能合约或从这些智能合约接收的（TON Blockchain）消息来完成。
我们已经在 2.9.13 中讨论过这种方法有其缺点和局限性，但也有其优点：这样的分布式应用程序不需要运行服务器或存储其数据（它在 “区块链” 中运行——即在验证人的硬件上运行），并享有区块链极高（拜占庭）的可靠性和可访问性。这种分布式应用程序的开发人员不需要购买或租用任何硬件；他需要做的就是开发一些软件（即智能合约的代码）。在那之后，她将有效地从验证人那里租用计算能力，并以 Grams 的形式支付它，无论是她自己支付还是让用户买单。

4.1.5 纯粹的网络服务：“ton-sites” 和 “ton-services”。

另一个极端的选择是在某些服务器上部署服务，并通过 3.1 中描述的 ADNL 协议将其提供给用户，也可以使用一些更高级别的协议，如 3.1.9 中讨论的 RLDP，可用于以任何自定义格式向五福发送 RPC 查询并获取这些查询的答案。 这种情况中，该服务将完全处于链下，并将驻留在 TON Network 中，几乎不使用 TON Blockchain。
TON Blockchain可能仅用于定位抽象地址或服务的地址，如 3.2.12 中所述，可能借助于诸如 TON DNS（参见 4.3.1）之类的服务来促进域的翻译——例如翻译人类可读的字符串到抽象地址。
在某种程度上，ADNL 网络（即 TON Network ）类似于隐形互联网项目（I^{2}P），这种（几乎）纯粹的网络服务类似于所谓的 “eep-services”（即具有一个I^{2}P -地址作为其入口点，并通过I^{2}P网络提供给客户端）。我们会说，驻留在 TON Network 中的这种纯粹的网络服务是 “ton-services”。
“eep-service” 可以实现 HTTP 作为其客户端——服务器协议；在 TON Network 环境中，“ton-services” 可能只是使用 RLDP（参见 3.1.9）数据报来传输 HTTP 查询和响应。如果它使用 TON DNS 允许通过人类可读的域名查找其抽象地址，那么对网站的类比就变得几乎完美了。甚至可以编写一个专门的浏览器，或者在用户机器上本地运行的特殊代理（“ton-proxy”），从用户使用的普通 Web 浏览器接受任意 HTTP 查询（一旦是本地 IP 地址和 TCP 端口）将代理输入到浏览器的配置中，并通过 TON Network 将这些查询转发到服务的抽象地址。然后，用户将具有类似于万维网（WWW）的浏览体验。
在I^{2}P生态系统中，这种 “eep-services” 被称为 “eep-sites”。人们也可以在 TON 生态系统中轻松创建 “ton-sites”。这在某种程度上得益于 TON DNS 等服务的存在，TON DNS 利用 TON Blockchain和 TON DHT 将（TON）域名转换为抽象地址。

4.1.6 Telegram Messenger 作为 ton-service；MTProto 覆盖 RLDP。

顺便一提，Telegram Messenger [35]用于客户端——服务器交互的 MTProto[36]协议可以很容易地嵌入到 3.1.9 中讨论的 RLDP 协议中，从而有效地将Telegram转换为 ton-service。 由于 TON Proxy 技术可以透明地为 ton-site 或 ton-service 的最终用户开启，实施的程度低于 RLDP 和 ADNL 协议（参见 3.1.6），这将使得Telegram能够有效地不可阻止。当然，其他消息和社交网络服务也可能从这项技术中受益。

4.1.7 混合服务：部分链下，部分链上。

有些服务可能采用混合方法：做大部分链下处理，但也有一些链上部分（例如向用户登记他们的义务，反之亦然）。通过这种方式，部分状态仍将保留在 TON Blockchain中（即不可变的公共帐本），并且服务或其用户的任何不当行为都可能受到智能合约的惩罚。

[35] https://telegram.org/
[36] https://core.telegram.org/mtproto

4.1.8 示例：保持文件在链下；TON Storage。

TON Storage 提供了这种服务的一个例子。在最简单的形式中，它允许用户通过保持链上只存储要存储的文件的哈希值来链接存储文件，并且可能是一个其中一些其他方同意在预先商定的费用和时间内保留有问题的文件地智能合约。实际上，文件可以被细分为一些小尺寸（例如，1 千字节）的块，通过诸如 Reed-Solomon 或喷泉码的擦除代码来增强，可以为增强的块序列构造 Merkle tree hash。 这个 Merkle tree hash可以在智能合约中发布，而不是与文件的通常hash一起发布。这有点让人联想到文件存储在 torrent 中的方式。
一种更简单的存储文件形式是完全链下的：本质上是为新文件创建一个 “torrent”，并使用 TON DHT 作为此 torrent 的 “分布式 torrent 追踪器”（参见 3.2.10）。对于流文件，这实际上可能非常有效。但没有获得任何可用性的保证。例如，一个假设的 “区块链 Facebook”（参见 2.9.13），可能会选择将其用户的个人资料照片完全脱离这种 “torrent”，这将可能会令普通（不是特别受欢迎的）用户失去其照片，或至少有可能无法长时间呈现这些照片。 TON Storage 技术主要是链下的，但使用链上智能合约来强制存储文件的可用性，可能更适合这项任务。

4.1.9 分散的混合服务，或 “雾服务”。

到目前为止，我们已经讨论了中心化混合服务和应用。虽然他们的链上组件以去中心化和分布的方式处理，但他们的链下组件依赖于服务提供商的以通常的中心化方式控制的一些服务器。算力可能来自某一家大公司提供的云计算服务，而非某些专用服务器。但是，这不会导致服务的链下组件的去中心化。
实现服务的链下组件的区中心方法在于创建一个市场，任何拥有所需硬件并愿意租用其计算能力或磁盘空间的人都会向需要它们的人提供服务。
例如，可能存在一个注册表（也可能称为 “market” 或 “exchange”），其中所有对保持其他用户的文件感兴趣的节点都会发布其联系信息，以及它们的可用存储容量，可用性策略和价格。需要这些服务的人可能会来查找，如果对方同意，则在区块链中创建智能合约并上传文件以进行离线存储。通过这种方式，像 TON Storage 这样的服务变得真正分散，因为它不需要依赖任何集中的服务器集群来存储文件。

4.1.10 “雾计算（fog computing）” 平台作为分散的混合服务。

当想要执行某些特定计算（例如 3D 渲染或训练神经网络）时，这种分散的混合应用的另一个例子就出现了，并且通常需要特定且昂贵的硬件。 那些拥有这种设备的人可能通过类似的 “exchange” 提供他们的服务，而需要这些服务的人会租用相关设备，双方的义务通过智能合约登记。 这类似于 “雾计算” 平台，如 Golem（https://golem.network/ ）或 SONM（ https://sonm.io/ ），的承诺交付。

4.1.11 示例：TON Proxy 是雾服务。

TON 代理提供了雾服务的另一个示例，其中希望提供其服务（有或没有补偿）作为 ADNL 网络流量的隧道的节点可能会注册，而需要它们的节点可能会根据价格、延迟、和带宽选择其中一个节点。之后就可以使用由 TON Payments 提供的支付通道来处理那些代理服务的小额支付，例如每收集 128 KiB 就付一次款。

4.1.12 示例：TON Payments 是一项雾服务。

TON 支付平台（参见 5）也是这种去中心化的混合应用的一个例子。

4.2 连接用户和服务提供商

我们在 4.1.9 中已经看到，“fog services”（即混合分散服务）通常需要供给双方都在的市场，交易所或注册管理机构。
这些市场被分为链上、链下、混合服务、中心化、去中心化等。

4.2.1 示例：连接到 TON Payments。

例如，如果想要使用 TON Payments（参见 5），第一步是找到 “lightning network” 的现有的一些传输节点（参见 5.2），并与它们建立支付通道。借助于 “encompassing” 覆盖网络可以找到一些节点，该覆盖网络应该包含所有传输闪电网络节点（参见 3.3.17）。 但目前尚不清楚这些节点是否愿意创建新的支付通道。 因此，需要注册表来让准备创建新链接的节点发布它们的联系信息（例如它们的抽象地址）。

4.2.2 示例：将文件上传到 TON Storage。

类似地，如果想要将文件上传到 TON Storage，则必须找到一些愿意签署智能合约的节点，以绑定它们来保留该文件的副本（或者任何低于特定大小限制的文件）。 因此需要提供用于存储文件的服务节点的注册表。

4.2.3 链上，混合和链下注册管理机构。

这样的服务提供商注册表可以完全在链上实现，借助智能合约将注册表保留在其永久存储中。然而这将极其缓慢且昂贵。混合方法则更为有效，其中相对较小且很少更改的链上注册表仅用于指出某些节点（通过其抽象地址或其公钥，可用于定位实际的抽象地址，如 3.2.12），提供链下（中心化）注册服务。
最后，分散的纯粹的链下方法可能包括公共覆盖网络（参见 3.3），那些愿意提供服务的人，或那些想要购买服务的人，只需通过他们的私钥广播他们的邀约。如果要提供的服务非常简单，甚至可能不需要广播要约：覆盖网络本身的近似成员资格可以用作愿意提供特定服务的人的“注册表”。如果已知的节点尚未准备好满足其需求，需要此服务的客户端可能会定位（参见 3.3.7）并查询此覆盖网络的某些节点，然后查询其邻近节点。

4.2.4 在一个侧链上的注册或交易。

实施分散式混合注册管理机构的另一种方法是创建一个独立的专业区块链（“侧链”），由其自己的一套验证人维护，他们在链上智能合约中发布其身份并提供网络访问对这个垂直区块链的所有感兴趣的各方，通过专用的覆盖网络收集交易候选人和广播区块更新（参见 3.3）。 然后，此侧链的任何完整节点都可以维护自己的共享注册表副本（基本上等于此侧链的全局状态），并处理与此注册表相关的任意查询。

4.2.5 工作链中的注册表或交易。

另一种选择是在 TON Blockchain 内创建专用工作链，专门用于创建注册管理机、市场和交易所。 与使用基本工作链中的智能合约相比，这可能更有效，更便宜（参见 2.1.11）。 但是，这仍然比在侧链中维护注册管理机构更昂贵（参见 4.2.4）。

4.3 访问 TON Services

我们在 4.1 中讨论了可能用于创建驻留在 TON 生态系统中的新服务和应用程序的不同方法。现在我们讨论如何访问这些服务，以及 TON 将提供的一些 “helper services”，包括 TON DNS 和 TON 存储。

4.3.1 TON DNS：主要是链上分层域名服务。

TON DNS 是一种预定义的服务，它使用一组智能合约来保存从人类可读域名到 ADNL 网络节点（T6 区块链账户和智能合约）的（256 位）地址的映射。
虽然任何人原则上可以使用 TON Blockchain来实现这样的服务，但是当应用程序或服务想要将人类可读标识符转换为地址时，默认情况下使用这种具有众所周知的接口的预定义服务是有用的。

4.3.2 TON DNS 用例。

例如，希望将一些加密货币转移给另一个用户或商家的用户，可能更愿意记住该用户或商家的帐户的 TON DNS 域名，而不是复制黏贴他们的 256 位帐户标识符到轻钱包客户端的接收者字段。
类似地，TON DNS 可用于定位智能合约的帐户标识符或 ton-services 和 ton-sites 的入口点（参见 4.1.5），从而启用专用客户端（“ton-browser”）或通常的互联网浏览器 结合专门的 TON Proxy 扩展或独立应用程序，为用户提供类似 WWW 的浏览体验。

4.3.3 TON DNS 智能合约。

TON DNS 通过特殊（DNS）智能合约树实现。 每个 DNS 智能合约都负责注册某些固定域的子域。 将保留 TON DNS 系统的第一级域的 “根”DNS 智能合约位于主链中。 其帐户标识符必须硬编码到希望直接访问 TON DNS 数据库的所有软件中。
任何 DNS 智能合约都包含一个哈希图，将可变长度的以 null 结尾的 UTF-8 字符串映射到它们的 “值” 中。 此哈希图实现为二进制 Patricia 树，类似于 2.3.7 中描述的，但支持可变长度位串作为键。

4.3.4 DNS 哈希图或 TON DNS 记录的值。

至于这些值，它们是由 TL 方案描述的 “TON DNS 记录”（参见 2.2.5）。它们由一个 “magic number” 组成，选择一个支持的选项，然后选择一个帐户标识符，或一个智能合约标识符，或一个抽象网络地址（参见 3.1），或一个用于定位抽象地址的公钥服务（参见 3.2.12），或覆盖网络的描述等等。一个重要的案例是另一个 DNS 智能合约：在这种情况下，该智能合约用于解析其域的子域。通过这种方式，可以为不同的域创建单独的注册表，由这些域的所有者控制。
这些记录还可能包含到期时间，缓存时间（通常非常大，因为更新区块链中的值通常是昂贵的），并且在大多数情况下是对相关子域所有者的引用。所有者有权更改此记录（特别是所有者字段，从而将域名转移给其他人的控制权），并延长它。

4.3.5 注册现有域的新子域。

为了注册现有域的新子域，人们只需向智能合约发送消息，智能合约是该域的注册商，包含要注册的子域（即密钥），其中一个是预定义的值之一——由域名所有者确定的罚款格式、所有者身份、到期日期和一定数量的加密货币。
子域名以 “先到先得” 的方式注册。

4.3.6 从 DNS 智能合约中检索数据。

原则上，如果永久存储内的哈希值的结构和位置是智能合约，那么包含 DNS 智能合约的主链或分片链的任何完整节点都可以查找该智能合约的数据库中的任何子域。
但这种方法仅适用于某些 DNS 智能合约。 如果使用非标准 DNS 智能合约它将会失败。
相反，使用基于通用智能合约接口和获取方法的方法（参见 4.3.11）。 任何 DNS 智能合约都必须定义带有 “get method” 的 “known signature”，该方法被调用以查找密钥。 由于这种方法对其他智能合约也有意义，特别是那些提供链上和混合服务的合同，我们将在 4.3.11 中详细解释。

4.3.7 翻译 TON DNS 域。

一旦任何全节点（单独或代表某个轻客户端）可以查找任何 DNS 智能合约的数据库中的条目，就可以从熟悉的的固定根DNS智能合约（帐户）标识符开始，任意地翻译任意 TON DNS 域名。
例如，如果想要翻译 A.B.C，就可以在根域数据库中查找key .C，.B.C 和 A.B.C。 如果找不到第一个，但第二个找到了，并且其值是对另一个 DNS 智能合约的引用，则在该智能合约的数据库中查找 A 并检索最终值。

4.3.8 为轻节点翻译 TON DNS 域名。

通过这种方式，主链的全节点以及域名查找过程中涉及的所有分片链可以在没有外部帮助的情况下将任何域名转换为其当前值。 轻节点可以请求完整节点代表它执行此操作并返回该值以及 Merkle 证明（参见 2.5.11）。 这种 Merkle 证明将使得工作节点能够验证答案是否正确，因此与通常的 DNS 协议相比，这种 TON DNS 响应不能被恶意拦截器 “spoofed”。
因为没有节点可以预期是关于所有 shard 链的全节点，所以实际的 TON DNS 域转换将涉及这两种策略的组合。

4.3.9 专用的 “TON DNS servers”。

简单的 “TON DNS servers”可以被提供，其将接收 RPC“DNS” 查询（例如，通过 3.1 中描述的 ADNL 或 RLDP 协议），请求服务器一个给定域名，通过转发一些来处理这些查询必要时子查询到其他（全）节点，并返回原始查询的答案，如果需要，可以通过 Merkle 证明进行扩充。
这样的 “DNS 服务器” 可以使用 4.2 中描述的方法之一向任何其他节点，尤其是轻客户端提供其服务（免费或收费）。 请注意，如果这些服务器被认为是 TON DNS servers 的一部分，它将有效地将其从分布式链上服务转换为分布式混合服务（即 “fog service”）。
以上是对 TON DNS 服务的简要概述，TON DNS 服务是 TON Blockchain和 TON Network 实体的人类可读域名的可扩展链式注册表。

4.3.10 访问智能合约中保存的数据。

我们已经看到，有时需要访问存储在智能合约中的数据而不改变其状态。
如果知道智能合约实施的细节，就可以从智能合约的永久性存储中提取所有需要的信息，这些信息可供智能合约所在的分片链的所有全节点使用。但是这是一种非常不优雅的方式，很大程度上取决于智能合约的执行。

4.3.11 智能合约的 “Get methods”。

更好的方法是在智能合约中定义一些 get method，即某些分片的入站消息在交付时不会影响智能合约的状态，但会生成一个或多个包含 “result” 的输出消息。以这种方式，可以在仅知道智能合约实现具有已知签名的 get method（即，要发送的入站消息的已知格式和作为结果要接收的出站消息）时从智能合约获得数据。
符合面向对象的编程（OOP）的方式更加优雅。但是到目前为止它有一个明显的缺陷：执行时必须实际将交易提交到区块链中（将获取消息发送到智能合约），等待它由验证人提交和处理，从新的区块中提取答案，并且支付 gas（即用于在验证人的硬件上执行 “Get methods”）。这是一种资源浪费：“Get methods”不会改变智能合约的状态，因此不需要在区块链中执行。

4.3.12 智能合约“Get methods”的尝试性执行。

我们已经谈论过（参见 2.4.6）任何全节点都可以暂时执行任何智能合约的任何方法（即向智能合约发送任何消息），从智能合约的给定状态开始，而不实际提交相应的交易。全节点可以简单地将所考虑的智能合约的代码加载到 TON VM 中，从分片链的全局状态（分片链的所有全节点都知悉）初始化其永久存储，并执行智能合约代码。入站消息作为其输入参数。创建的输出消息将产生此计算的结果。
这样，任何全节点都可以评估任意智能合约的任意get methods，前提是它们的签名（即入站和出站消息的格式）是已知的。节点可以追踪在该评估期间访问的分片链状态的细胞，并创建所执行的计算的有效性的 Merkle 证明，以获得可能已经要求整个节点这样做的工作节点（参见 2.5.11）。

4.3.13 TL-schemes中的智能合约接口。

回想一下，智能合约实现的方法（即它接受的输入消息）本质上是一些 TL 序列化的对象，可以通过 TL-schemes来描述（参见 2.2.5）。 得到的输出消息也可以用相同的TL-schemes描述。 通过这种方式，智能合约提供给其他账户和智能合约的界面可以通过TL-schemes形式化。
特别是，智能合约支持的 Get methods（的一部分）可以通过这种形式化的智能合约接口来描述。

4.3.14 智能合约的公共接口。

请注意，形式化的智能合约接口，无论是 TL-schemes（表示为 TL 源文件；参见 2.2.5）还是序列化形式都可以发布——例如，在特殊领域中 智能合约帐户描述，存储在区块链中，或单独存储，如果此接口将被多次引用。 在后一种情况下，支持的公共接口的哈希可以合并到智能合约描述中而不是接口描述本身。
这种公共接口的一个例子是 DNS 智能合约，它应该实现至少一种用于查找子域的标准 Get methods（参见 4.3.6）。 注册新子域的标准方法也可以包含在 DNS 智能合约的标准公共接口中。

4.3.15 智能合约的用户界面。

智能合约的公共界面的存在也有其他优点。例如，钱包客户端应用程序可以在根据用户的请求检查智能合约时下载这样的界面，并显示智能合约支持的公共方法列表（即可用动作的列表），可能具有一些在正式版本中提供的人类可读的评论。

在用户选择这些方法之一之后，可以根据 TL-scheme自动生成表单，其中将提示用户所选方法所需的所有字段以及所需的加密货币量（例如 Grams）附在此请求上。提交此表单将创建一个新的区块链交易，其中包含刚刚撰写的消息，该消息来自用户的区块链账户。
通过这种方式，只要这些智能合约已经发布了他们的界面，用户就可以通过填写和提交某些表格，以用户友好的方式与钱包客户端应用程序中的任意智能合约进行交互。

4.3.16 ton-service 的用户界面。

事实证明 “ton-services”（即驻留在 TON Network 中的服务以及通过 ADNL 和 RLDP 协议接受 3 的查询；参见 4.1.5）也可能从 TL-scheme描述的公共接口中获益（ 参见 2.2.5）。
客户端应用程序（例如轻型钱包或 “ton-browser”）可能会提示用户选择其中一种方法，并使用界面定义的参数填写表单，类似于刚刚在 4.3.15 中讨论的内容。唯一的区别是生成的 TL-serialized 消息不作为区块链中的交易提交；相反它作为 RPC 查询被发送到所讨论的 “ton-service” 的抽象地址，并且根据形式接口（即 TL-scheme）解析和显示对该查询的响应。

4.3.17 通过 TON DNS 定位用户界面。

包含 ton 服务的抽象地址或智能合约帐户标识符的 TON DNS 记录还可以包含描述该实体的公共（用户）接口的可选字段，或者几个支持的接口。 然后客户端应用程序（无论是 wallet, ton-browser 还是 ton-proxy）将能够以统一的方式下载界面并与相关实体（无论是智能合约还是一个 ton-service）进行交互。

4.3.18 模糊链上和链下服务之间的区别。

通过这种方式，最终用户模糊了链上、链下和混合服务（参见 4.1.2）之间的区别：她只需将所需服务的域名输入到她的ton-browser的地址行或钱包中，其余部分将由客户端应用程序无缝处理。

4.3.19 轻钱包和 TON entity explorer 实体浏览器可以内置到 Telegram Messenger 客户端中。

在这一点上出现了一个有趣的机会。 实现上述功能的轻型钱包和 TON 实体浏览器可以嵌入到 Telegram Messenger 智能手机客户端应用程序中，从而将该技术带给超过 2 亿人。用户可以通过在消息中包含 TON URIs（参见 4.3.22）来向 TON 实体和资源发送超链接； 如果选择了这样的超链接，将由接收方的电报客户端应用程序在内部打开，并且将开始与所选实体的交互。

4.3.20 “ton-sites” 作为支持 HTTP 接口的 ton-services。

ton-site 只是一个支持 HTTP 接口的服务，可能还有其他一些接口。
可以在相应的 TON DNS 记录中宣布该支持。

4.3.21 超链接（Hyperlinks）。

请注意，ton-sites 返回的 HTML 页面可能包含 ton-hyperlinks-即通过特制 URIs 方案引用其他 ton-sites，智能合约和帐户参见 4.3.22）——包含摘要网络地址、帐户标识符或人类可读的 TON DNS 域名。 然后，当用户选择它时，“ton-browser” 可能会跟随这样的超链接，检测要使用的界面，并显示 4.3.15 和 4.3.16 中概述的用户界面表格。

4.3.22 超链接 URL 可以指定一些参数。

超链接 URL 不仅可以包含（TON）DNS 域或所讨论的服务的抽象地址，还可以包含要调用的方法的名称以及其部分或全部参数。可能的 URI 方案可能如下所示：

ton://<domain>/<method>?<field1>=<value1>&<field2>=. . .

当用户在 ton-browser 中选择这样的链接时，该动作是立即形成（特别是如果它是智能合约的获取方法，匿名调用），或显示部分填写的表格，由用户明确确认和提交（这可能是付款表格所必需的）。

4.3.23 POST 动作。

一个 ton-sites 可以嵌入到 HTML 页面中，它返回一些看似常见的 POST 表单，POST 操作通过合适的（TON）URL 引用 ton-sites，ton-services 或智能合约。 在这种情况下，一旦用户填写并提交该自定义表单，就会立即或在明确确认后采取相应的操作。

4.3.24 TON WWW。

以上内容将帮助创建整个网络的交叉引用实体，驻留在 TON Network中，最终用户可通过吨浏览器访问该网络，从而为用户提供类似 WWW 的浏览体验。 对于最终用户，这最终将使区块链应用程序与他们已经习惯的网站基本相似。

4.3.25 TON WWW 的优点。

这种 “链上和链下” 服务的 “TON WWW” 与传统服务相比具有一些优势。 例如，支付本质上被集成在系统中。 用户的身份可以始终呈现给服务（通过在交易和生成的 RPC 请求上自动生成的签名）或随意隐藏。 服务无需检查和重新检查用户凭据；这些凭证可以一次性发布在区块链中。用户网络匿名可以通过 TON 代理轻松保存，并且所有服务都将是有效且不可阻止的。小额支付也很容易，因为 ton-browser 可以与 TON 支付系统集成。

5 TON Payments

我们将在本文TON 项目的最后一部分简要讨论TON Payments，这是（小额）支付通道和“闪电网络”价值转移的平台。它将实现“即时”支付，无需将所有交易提交到区块链中、支付相关的交易费用（例如消耗的 gas）并等待五秒钟、直到确认包含有关交易的区块。

这种即时支付的总体开销很小，可以将它们用于小额支付。例如，TON file-storing 服务可能会为每下载 128 KiB向用户收费，或者付费 TON 代理可能需要为转发的每128 KiB提供一些小额微支付。

虽然 TON Payments 可能会晚于 TON 项目核心部分的发布，但一开始就需要考虑这些因素。例如，用于执行 TON区块链智能合约代码的 TON 虚拟机（TON VM；参见 2.1.20）必须支持默克尔校验的一些特殊操作。如果最初的设计中不存在此类支持，在之后的阶段添加则可

能会出现问题（参见 2.8.16）。但是，我们将看到 TON虚拟机天然支持“智能”支付通道（参见 5.1.9）。

5.1 支付通道

我们首先讨论点对点支付通道，以及如何在 TON 区块链中实施这些通道。

5.1.1 支付通道的思路

假设 A 和 B 两方都知道他们将来需要相互支付很多款项。他们不用将每笔付款提交到区块链中交易，而是创建一个共享的“资金池”（或者也可能是一个只有两个帐户的小型私人银行），并为其提供一些资金：A 贡献 a 币，B 贡献 b 币。这是通过在区块链中创建一个特殊的智能合约并将资金发送给智能合约来实现的。

在创建“资金池”之前，双方同意某个协议。他们将追踪池的状态——即共享池中的余额。最初，状态是（a，b），意思是a币实际上属于 A，b 币属于 B。然后，如果 A 想要向 B 支付d币，他们可以简单地同意新状态是（a '，b'）=（a - d，b + d）。之后，如果 B 想要向 A 支付d'币，那么状态将成为（a''，b''）=（a'+ d'，b' - d'），依此类推。

所有这些池内余额的更新都完全在链下完成。当双方决定从池中取出应付资金时，他们会根据池的最终状态进行操作。这是通过向智能合约发送特殊消息来实现的，其中包含商定的最终状态（a *，b *）以及 A 和 B 的签名。然后智能合约向 A发送a *币，向B发送b *币，并扣除自己的部分。

此智能合约、以及 A 和 B 用于更新池子状态的网络协议是 A 和 B 之间的简单支付通道。根据 4.1.2 中描述的分类，它是一种混合服务：其部分状态存在于区块链（智能合约）中，但其大多数状态更新都是链下（通过网络协议）执行的。如果一切顺利，双方将能够按照自己的意愿执行尽可能多的付款（唯一的限制是通道的容量不会超支——即，他们在付款通道中的余额都大于0），只要将两个交易提交到区块链中：一个用于打开（创建）支付通道（智能合约），另一个用于关闭（销毁）通道。

5.1.2 无需信任的支付通道

前面的例子有点不切实际，因为它假设双方都愿意合作，绝不会用欺骗的手段获得一些好处。想象一下，假如A 在a'<a的情况下，会选择不签署最终出余额（a'，b'）。这将使 B 陷入困境。

为防止这种情况，人们通常会尝试开发无需信任的支付通道协议，这些协议不要求各方相互信任，并规定惩罚任何试图作弊的人。

这通常借助签名来实现。支付通道智能合约知道 A 和 B 的公钥，并且如有必要，可以检查他们的签名。支付通道协议要求各方签署中间状态并将签名发送给彼此。然后，如果其中一方欺骗——例如假装支付通道的某些状态从未存在——可以通过在该状态上显示其签名来证明其不当行为。支付通道智能合约充当“链上仲裁者”，能够处理双方关于彼此的投诉，并通过没收所有资金并将其授予另一方来惩罚作恶方。

5.1.3 简单、双向同步的无需信任支付通道

考虑以下更实际的例子：支付通道的状态由三元组（δ_i, i, o_i ）描述，其中 i 是状态的序列号（原始数值是零，随后出现一个状态，数值增加1），δ_i 是通道不平衡（意味着 A 和 B 分别拥有 a+δ_i 和 b − δ_i 币），并且 o_i 是允许产生下一状态（A 或 B）的一方。 在取得进一步进展之前，每个状态必须由 A 和 B 签署。

现在，如果 A 想要将 d 币转移到支付通道内的 B，并且当前状态是 $S_i = (\delta_i, i, o_i)$ ，其中 o_i = A ，则它只是创建一个新状态 S_i+1 = (δ_i − d, i + 1, o_i+1) ，签名，并将其连同其签名一起发送给 B。然后 B 通过签名并将其签名副本发送给 A 来确认。之后，双方都拥有含新签名的新状态的副本，并且可能会有新的转账。

如果 A 想要在具有 oi = B的S_i状态中将币转到 B，则它首先要求 B 提交S_{i+1}的后续状态，具有相同不平衡$\delta_{i+1} = \delta_i$，但是o_{i+1} = A 。之后，A 将能够进行转账。

当双方同意关闭支付通道时，他们都将他们的特殊最终签名放在他们认为是最终的状态 S_k 上，并通过发送最终状态和最终签名来调用支付通道智能合约的最终或双方最终确定的方法。

如果另一方不同意提供其最终签名，或者仅仅是停止响应，则可以单方面关闭该通道。为此，想这么做的一方将调用单方面的最终确定方法，向智能合约发送其最终状态的版本、最终签名以及具有另一方签名的最新状态。在此之后，智能合约不会立即对收到的最终状态采取行动。相反，它会等待一段时间（例如，一天）以使另一方呈现其最终状态的版本。当另一方提交其版本并且结果与已提交的版本兼容时，“真实”的最终状态由智能合约计算并相应地分配资金。如果另一方未能将其最终状态的版本呈现给智能合约，则根据所呈现的最终状态的唯一副本重新分配资金。

如果双方中有一方作弊——例如，签署两个不同的作为最终状态，或者签署两个不同的下一状态 S_{i+1} 和 S'_{i+1}，或者签署无效的新状态 S_{i+1} （例如，不平衡状态δ_{i+1} <-a 或> b）——则另一方可以提交这种不当行为的证明到智能合约的第三种方法。作恶的一方将立即受到惩罚，完全失去其在支付通道中的份额。

这种简单的支付通道协议是公平的，即无论是否有另一方的合作，任何一方都可以随时获得应付款，并且如果它试图作弊，可能会失去提交到支付通道的所有资金。

5.1.4 同步支付通道，一个带有两个验证人的简单的虚拟区块链

以上简单、同步支付信道的示例可以按以下操作重新制作。想象一下，状态序列 $S_0, S_1, ..., S_n$ 实际上是非常简单的区块链的区块序列。该区块链的每个区块基本上仅包含区块链的当前状态，并且可能是对前一个区块的引用（即，其哈希值）。A 和 B 双方都充当此区块链的验证人，因此每个区块必须收集其两个签名。区块链的状态 S_i 定义为下一个区块的指定生产者 o_i，因此 A 和 B 之间没有生成下一个区块的竞争。出块节点 A 可创建区块，将资金从 A 转移到 B（即，减少不平衡：$\delta_{i+1} \le \delta_i$) ，并且 B 只能将资金从 B 转移到 A（即，增加δ）。

如果两个验证人就区块链的最终区块（和最终状态）达成一致，则通过收集双方的特殊“最终”签名来最终确定，并将其与最终区块一起提交给通道智能合约进行处理和相应地重新分配资金。

如果验证人签署了无效区块，或者创建了一个分叉，或者签署了两个不同的最终区块，则可以通过向智能合约提供其不当行为的证据来惩罚它，该合同充当两个验证人的“链上仲裁者”；然后，违方将失去保留在支付通道中的所有资金，这类似于失去其 stake 的验证人。

5.1.5 异步支付通道，具有两个工作链的虚拟区块链

5.1.3 中讨论的同步支付通道具有一定缺陷：在另一方没有确认之前的交易之前，不能开始下一笔交易（支付通道内的汇款）。这可以通过用两个交互虚拟工作链（或者说是分片链）的系统替换 5.1.4 中讨论的单链虚拟区块链来解决。

这些工作链中的第一条仅包含 A 的事务，区块只能由 A 生成；它的状态是 $S_i = (i, \varphi_i, j, \psi_j)$, 其中 i 是区块序列号（即到目前为止由 A 执行的交易或汇款的数量），φ_i 是目前为止从 A 转移到B 的总量，j 是 A 知道的 B 区块链中最近有效区块的序列号，ψ_j 是其 j 交易中从 B 转移到 A 的金额。放在第 j 个块上的 B 的签名也应该是这个状态的一部分。也可以包括该工作链的前一个块和另一个工作链的第 j 个块的哈希。如果 $i > 0$, $\psi_j \geq 0$ 且 $-a \leq \psi_j - \varphi_i \leq b$，$S_i$的有效性条件包括 $\varphi_i \geq 0$, $\varphi_i \geq \varphi_{i-1}$。

类似地，第二个工作链仅包含 B 的交易，区块仅由 B 生成；其状态为 $T_j = (j, \psi_j, i, \varphi_i)$，具有相似的有效性条件。

现在，如果 A 想要将一些钱转移到 B，它只需在其工作链中创建一个新块，签名并发送给 B，无需等待确认。

支付通道由 A 签署（其版本）区块链的最终状态（具有特殊的“最终签名”），B 签署其区块链的最终状态，并将这两个最终状态呈现给支付通道智能合约的最终确定方法。单方最终确定也是可能的，但在这种情况下，智能合约必须等待另一方提交其最终状态的版本，至少需要一定宽限期。

5.1.6 单向支付通道

如果只有 A 需要向 B 付款（例如，B 是服务提供商，A 是其客户），则可以创建单边支付通道。从本质上讲，它只是 5.1.5 中描述的没有第二条工作链的第一条工作链。相反，可以说 5.1.5 中描述的异步支付通道由两个单向支付通道组成，或由同一智能合约管理的“半通道”组成。

5.1.7 更复杂的支付通道，授权

我们将在后面的 5.2.4 中看到，“闪电网络”（参见 5.2），通过几个支付通道的链实现即时汇款，需要所涉及的支付通道更加复杂。

特别是，我们希望能够提交“授权”或“有条件转账”：A 同意向 B 发送 c 币，但 B 只有在满足某个条件时才会获得资金（例如，B 可以用一个已知的 v 值表示Hash(u) = v）。否则，A 可以在一段时间后收回资金。

这样的授权可以通过简单的智能合约在链上轻松实现。我们希望授权和其他类型的有条件转账可以在链下支付通道中进行，因为它们可以大大简化“闪电网络”中存在的支付通道链上的资金转移（参见 5.2.4）。

5.1.4 和 5.1.5 中概述的“支付通道，简单的区块链”方案在这里变得很方便。现在我们考虑一个更复杂的虚拟区块链，其状态包含一系列未实现的“授权”，以及锁定在此类授权中的资金数量。这个区块链——或异步情况下的两个工作链——必须通过它们的哈希明确引用之前的区块。不过，整个机制仍然相同。

5.1.8 复杂的支付通道智能合约面临的挑战

请注意，虽然复杂的支付通道的最终状态仍然很小，而且最终确定很简单（如果双方已就其应付金额达成一致，并且双方已签署协议，就无需做其他事），但单边确定方法和惩罚欺诈行为的方法需要更加复杂。实际上，他们必须能够接受默克尔校验不正当行为，并检查支付通道链上更复杂的交易是否在正确处理。

换句话说，支付通道智能合约必须能够使用默克尔校验，检查其“哈希有效性”，并且必须包含支付通道（虚拟）链的 ev*trans 和 ev*block 功能的实现（参见 2.2.6）。

5.1.9 TON VM 支持 “智能” 支付通道

用于运行 TON区块链智能合约代码的 TON VM 可以应对执行“智能”或复杂支付通道所需的智能合约（参见 5.1.8）。

在这一点上，“everything is a bag of cells” 范例（参见 2.5.14）变得非常方便。由于所有块（包括临时支付通道区块链的块）都表示为a bag of cells（并由一些代数数据分片描述），并且同样适用于消息和默克尔校验，因此可以轻松地将默克尔校验嵌入到发送到支付通道智能合约的入站消息。默克尔校验的“哈希条件”将会自动进行检查，并且当智能合约访问所呈现的“默克尔校验”时，它的使用方式就好像它是相应的代数数据分片的值——尽管不完整，树的一些子树被包含省略子树的默克尔哈希的特殊节点替换。然后，智能合约将使用该值，该值可能代表支付通道（虚拟）区块链的块及其状态，并将评估该区块的区块链的 ev_block 功能（参见 2.2.6）和以前的状态。然后，计算结束，并且可以将最终状态与区块中断言的状态进行比较，或者在尝试访问缺席子树时抛出“缺席节点”异常，表明默克尔校验无效。

通过这种方式，使用 TON 区块链智能合约实现智能支付通道区块链的验证码变得非常简单。有人可能会说，TON 虚拟机具有内置支持，可以检查其他简单区块链的有效性。唯一的限制因素是要合并到智能合约（即进入交易）的入站消息中默克尔校验的大小。

5.1.10 智能支付通道内的简单支付通道

我们想讨论在现有支付通道内创建简单（同步或异步）支付通道的可能性。

虽然这看起来有点令人费解，但理解和实施并不比 5.1.7 中讨论的“授权”困难得多。基本上，如果出现一些哈希问题的解决方案，并不是承诺向另一方支付 c 币，A 承诺根据某些其他（虚拟）支付通道区块链的最终结算向 B 支付最多 c 币。一般来说，这个其他支付通道区块链甚至不需要在 A 和 B 之间；它可能涉及其他方，比如 C 和 D 将分别提交 c 和 d 币至其简单的支付通道。（这种可能性在后面的 5.2.5 中使用。）

如果包含的支付通道是不对称的，则需要将两个授权提交到两个工作链中：如果“内部”简单支付通道的最终结算产生负的最终不平衡δ且 0≤-δ≤c，A 将承诺向 B 支付-δ个币；如果δ为正，则 B 必须承诺向 A 支付δ。另一方面，如果包含支付通道是对称的，这可以通过由A将单个含

参数（c，d）的“简单支付通道创建”交易一起提交到单个支付通道区块链中来完成（这将冻结属于A的 c 币），然后由 B 进行特殊的“确认交易”（这将冻结 B 的d币）。

我们希望内部支付通道非常简单（例如，5.1.3 中讨论的简单同步支付通道），以使要提交的默克尔校验的大小最小化。在上，外部支付通道必须是5.1.7 意义上的“智能”。

5.2 支付通道网络或 “闪电网络”

现在我们讨论 TON Payments 的“闪电网络”，它可以实现任意两个参与节点间的即时转账。

5.2.1 支付通道的限制

支付通道对于期望在他们之间进行大量转账的各方非常有用。但是如果只需向特定收款方转账一次或两次，那么与她创建支付通道将是不切实际的。除此之外，这意味着冻结支付通道中的大量资金，并且无论如何都需要至少两个区块链交易。

5.2.2 支付通道网络，或 “闪电网络”

支付通道网络通过支付通道链实现转账，克服了支付通道的缺陷。如果 A 希望将资金转移到 E，她不需要与 E 建立支付通道。通过几个中间节点（例如，四个支付通道，从 A 到 B，从 B 到 C，从 C 到 D，从 D 到 E）连接 A 和 E 的支付通道就足够了。.

5.2.3 支付通道网络概述

回想一下，支付通道网络，也称为 “闪电网络”，由一组参与节点组成，其中一些节点已在它们之间建立了长期支付通道。我们将看到这些支付通道必须是 5.1.7 意义上的“智能”。 当参与节点 A 想要将钱转移到任何其他参与节点 E 时，她试图找到在支付通道网络内连接 A 到 E 的路径。当找到这样的路径时，她沿着这条路径进行 “区块链资金转账”。

5.2.4 区块链资金转账

假设存在从 A 到 B，从 B 到 C，从 C 到 D，从 D 到 E 的支付通道链。此外，假设 A 想要将 x 个代币转移到 E。

一种简单的方法是沿着现有的支付通道将 x 个代币转移到 B，并要求他将钱进一步转发给 C。但是，为什么 B 不自己拿钱呢？因此，必须采用更复杂的方法，无需要求所有相关方相互信任。

这可以通过以下方式实现。A 生成一个大的随机数 u 并计算其哈希 v = Hash（u）。然后她创建一个授权，如果有一个带有哈希 v 的数字 u（参见 5.1.7），在她的支付通道中有 B，则向 B 支付 x个代币。这个授权包含 v，但不是 u，u仍然保密。

之后，B 在其支付通道中创建了与 C 类似的授权。他并不害怕给出这样的授权，因为他知道 A 给他的类似授权的存在。如果 C 曾提出哈希问题的解决方案来收取 B 承诺的 x 个代币，那么 B 将立即向 A 提出该方案，向 A 收取 x 个代币。

然后创建 C 到 D 和 D 到 E 的类似授权。当授权全部到位时，A 通过将解决方案 u 传达给所有相关方——或者只是向 E 传达来触发转账。

本文档省略了一些细节。例如，这些授权必须具有不同的到期时间，并且授权的金额可能在链上略有不同（B 可能只承诺 x——ε个代币给 C，其中ε是预先商定的小额转账费用）。我们暂时忽略这些细节，因为它们对于理解支付通道如何运作以及如何在 TON 中实施这些细节并不太重要。

5.2.5 支付通道链中的虚拟支付通道

现在假设 A 和 E 期望相互间有很多的支付往来。他们可能会在他们之间的区块链中创建一个新的支付通道，但这仍然非常昂贵，因为有些资金会被锁定在这个支付通道中。另一种选择是对每笔付款使用 5.2.4 中描述的区块链资金转账。但是，这将涉及大量网络活动以及所涉及的所有支付通道的虚拟区块链中的大量交易。

另一种方法是在支付通道网络中连接 A 到 E 的链内创建虚拟支付通道。为此，A 和 E 为他们的付款创建（虚拟）区块链，就像他们要在区块链中创建支付通道一样。然而，他们不是在区块链中创建支付通道智能合约，而是要求所有中间支付通道——连接 A 到 B、B 到 C等等通道——在中间创建简单的支付通道，绑定到 A 和E创建的虚拟区块链（参见 5.1.10）。换句话说，现在根据 A 和 E 之间的最终结算转移资金的授权存在于每个中间支付通道内。

如果虚拟支付通道是单向的，那么这种授权可以很容易实现，因为最终的不平衡δ将是负数，因此可以在中间支付通道内以与 5.2.4 中描述的相同的顺序创建简单的支付通道。它们的到期时间也可以以相同的方式设置。

如果虚拟支付通道是双向的，情况会稍微复杂一些。在这种情况下，应该根据最终结算将转移δ代币的授权分为两个半授权，如 5.1.10 所述：向前方向转移$\delta^- = \max(0, -\delta)$ 代币，以及在向后方向上传递$\delta^+ = \max(0, \delta)$ 。这些半授权可以在中间支付通道中独立创建，一个从 A 到 E方向的半授权链，另一个相反方向的链。

5.2.6 寻找闪电网络中的路径

到目前为止，有一点仍然未被讨论：A 和 E 将如何在支付网络中找到连接它们的路径？如果支付网络不是太大，则可以使用类似 OSPF 的协议：支付网络的所有节点创建覆盖网络（参见 3.3.17），然后每个节点传播所有可用连接（即，参与支付通道）通过gossip协议向其邻近节点提供信息。最终，所有节点都将拥有参与支付网络的所有支付通道的完整列表，并且能够自己找到最短路径——例如，通过应用 Dijkstra 修改的算法版本来考虑所涉及的支付通道的“容量”（即，可以沿通道转移的最大金额）。一旦找到候选路径，就可以通过包含完整路径的特殊 ADNL 数据报进行探测，并要求每个中间节点确认所讨论的支付通道的存在，并根据该路径进一步转发该数据报。之后，可以构建链，以及用于链转移或者在支付通道链中（参见 5.2.4）创建虚拟支付通道的协议可以运行。

5.2.7 优化

可以在此处进行一些优化。例如，只有闪电网络的传输节点需要参与 5.2.6 中讨论的类似 OSPF 的协议。希望通过闪电网络连接的两个“叶子”节点将彼此通信他们所连接的传输节点的

列表（即，他们已经建立了参与支付网络的支付通道）。然后，如上面 5.2.6 中所述，可以检查从一个列表到另一个列表中的传输节点的路径。

5.2.8 结论

我们已经概述了TON项目的区块链和网络技术如何足够完成创建TON Payments（一个用于链下即时转账和小额支付的平台）的任务。该平台对于TON生态系统中的服务非常有用，使他们可以在有需要时轻松收取小额支付。

结论

我们提出了一种可扩展的多区参见链架构，能够支持大规模流行的加密货币和去中心化应用程序，带有用户友好的界面。

为了实现必要的可扩展性，我们提出了*TON区块链*，一个“紧密耦合”的多区块链系统（参2.8.14），采用自下而上的分片方法（参见2.8.12和2.1.2）。为了进一步提高潜在绩效，我们引入了二维区块链（2-blockchain）机制来替换无效区块（参见2.1.17）和即时超立方体路由（Instant Hypercube Routing），以实现分片之间更快速的交流（参见2.4.20）。将TON区块链与现有和计划中的区块链项目（参见2.8和2.9）进行简要比较，突出了TON这种方法的好处，这是一个追求每秒处理数百万次交易的系统。

第3章中描述的*TON网络*，涵盖了计划的多区块链基础设施的网络需求。该网络组件还可以与区块链结合使用，以创建广泛的应用和服务，单独使用区块链是不可能的（参见2.9.13）。第4章讨论的这些服务包括*TON DNS*，这是一种将人类可读对象标识符转换为其地址的服务；*TON Storage*，一个用于存储任意文件的分布式平台；*TON Proxy*，一种将网络访问匿名化和访问TON驱动服务的服务；*TON Payments*（参见第5章），一个跨TON生态系统、用于即时脱链资金转移的平台，应用程序可以使用这个生态系统进行小额支付。

TON基础设施可以使用专门的轻客户端钱包和“ton浏览器”的桌面和智能手机app，为终端用户提供类似浏览器的体验（参见4.3.24），这使得在TON平台上进行的加密货币支付、与智能合约和其他服务的交互对大众用户可访问。这样的轻客户端可以集成到Telegram Messenger客户端（参见4.3.19），从而最终为数亿用户带来了大量基于区块链的应用程序。

参考文献

[1] K. Birman, Reliable Distributed Systems: Technologies, Web Services and Applications, Springer, 2005.
[2] V. Buterin, Ethereum: A next-generation smart contract and de- centralized application platform, https://github.com/ethereum/wiki/wiki/White-Paper, 2013.
[3] M. Ben-Or, B. Kelmer, T. Rabin, Asynchronous secure computa- tions with optimal resilience, in Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing, p. 183–192. ACM, 1994.
[4] M. Castro, B. Liskov, et al., Practical byzantine fault tolerance, Proceedings of the Third Symposium on Operating Systems Design and Implementation (1999), p. 173–186, available at http://pmg.csail.mit.edu/papers/osdi99.pdf.
[5] EOS.IO, EOS.IO technical white paper, https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md, 2017.
[6] D. Goldschlag, M. Reed, P. Syverson, Onion Routing for Anony- mous and Private Internet Connections, Communications of the ACM, 42, num. 2 (1999), http://www.onion-router.net/Publications/CACM-1999.pdf.
[7] L. Lamport, R. Shostak, M. Pease, The byzantine generals problem, ACM Transactions on Programming Languages and Systems, 4/3 (1982), p. 382–401.
[8] S. Larimer, The history of BitShares, https://docs.bitshares.org/bitshares/history.html, 2013.
[9] M. Luby, A. Shokrollahi, et al., RaptorQ forward error correction scheme for object delivery, IETF RFC 6330, https://tools.ietf.org/html/rfc6330, 2011.
[10] P. Maymounkov, D. Mazières, Kademlia: A peer-to-peer infor- mation system based on the XOR metric, in IPTPS ’01 revised pa- pers from the First International Workshop on Peer-to-Peer Systems, p. 53–65, available at http://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf, 2002.
[11] A. Miller, Yu Xia, et al., The honey badger of BFT protocols, Cryptology e-print archive 2016/99, https://eprint.iacr.org/2016/ 199.pdf, 2016.
[12] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, https://bitcoin.org/bitcoin.pdf, 2008.
[13] S. Peyton Jones, Implementing lazy functional languages on stock hardware: the Spineless Tagless G-machine, Journal of Functional Pro- gramming 2 (2), p. 127–202, 1992.
[14] A. Shokrollahi, M. Luby, Raptor Codes, IEEE Transactions on Information Theory 6, no. 3–4 (2006), p. 212–322.
[15] M. van Steen, A. Tanenbaum, Distributed Systems, 3rd ed., 2017.
[16] The Univalent Foundations Program, Homotopy Type Theory: Univalent Foundations of Mathematics, Institute for Advanced Study, 2013, available at https://homotopytypetheory.org/book.

[17] G. Wood, Polkadot: vision for a heterogeneous multi-chain frame- work, draft 1, https://github.com/w3f/polkadot-white-paper/raw/ master/PolkaDotPaper.pdf, 2016.

申明

转载或商用请保留作者与译者的姓名。
新版白皮书未来将更新在 https://github.com/opeakt/TONBUS，请留意更新。