Telegram 開放網路

Dr. Nikolai Durov

November 1, 2025

Abstract

本文旨在提供 Telegram 開放網路（TON）及相關區塊鏈、點對點、分散式儲存和服務託管技術的首次描述。為了將本文件的篇幅控制在合理範圍內，我們主要關注 TON 平台的獨特和決定性功能，這些功能對於實現其既定目標至關重要。

簡介

Telegram 開放網路（*TON*）是一個快速、安全且可擴展的區塊鏈和網路專案，如有必要，能夠每秒處理數百萬筆交易，同時對使用者和服務提供者都友善。我們的目標是使其能夠託管目前提出和構思的所有合理應用程式。人們可以將 TON 想像成一個巨大的分散式超級電腦，或者更確切地說，是一個巨大的「超級伺服器」，旨在託管並提供各種服務。

本文並非旨在成為所有實作細節的終極參考。某些細節可能會在開發和測試階段發生變化。

Contents

1 TON 組件簡述

Telegram 開放網路（*TON*）是以下組件的組合：

- 一個靈活的多區塊鏈平台（*TON* 區塊鏈；參見第 **2** 章），能夠每秒處理數百萬筆交易，具有圖靈完備的智慧合約、可升級的正式區塊鏈規範、多加密貨幣價值轉移、支援微支付通道和鏈下支付網路。*TON* 區塊鏈呈現了一些新穎且獨特的功能，例如「自我修復」垂直區塊鏈機制（參見 **2.1.17**）和即時超立方體路由（參見 **2.4.20**），使其能夠同時具有快速、可靠、可擴展和自洽的特性。

- 一個點對點網路（*TON P2P* 網路，或簡稱 *TON* 網路；參見第 **3** 章），用於存取 TON 區塊鏈、傳送交易候選，以及接收有關客戶端感興趣的區塊鏈部分的更新（例如，與客戶端帳戶和智慧合約相關的部分），但也能夠支援任意的分散式服務，無論是否與區塊鏈相關。

- 一種分散式檔案儲存技術（*TON* 儲存；參見 **4.1.8**），可透過 *TON* 網路存取，由 TON 區塊鏈用於儲存區塊和狀態資料（快照）的存檔副本，但也可用於為使用者或平台上執行的其他服務儲存任意檔案，具有類似 torrent 的存取技術。

- 一個網路代理/匿名化層（*TON* 代理；參見 **4.1.11** 和 **3.1.6**），類似於 I^2P（隱形網際網路專案），在必要時用於隱藏 *TON* 網路節點的身分和 IP 位址（例如，從擁有大量加密貨幣的帳戶提交交易的節點，或希望隱藏其確切 IP 位址和地理位置以防範 DDoS 攻擊的高風險區塊鏈驗證者節點）。

- 一個類似 Kademlia 的分散式雜湊表（*TON DHT*；參見 **3.2**），用作 *TON* 儲存的「torrent 追蹤器」（參見 **3.2.10**）、*TON* 代理的「輸入通道定位器」（參見 **3.2.14**）以及 *TON* 服務的服務定位器（參見 **3.2.12**）。

- 一個用於任意服務的平台（*TON* 服務；參見第 **4** 章），駐留於 *TON* 網路和 *TON* 代理中並可透過它們存取，具有正式化的介面（參見 **4.3.14**），使瀏覽器或智慧型手機應用程式能夠進行互動。這些正式介面和持久服務入口點可以發佈在 TON 區塊鏈中（參見 **4.3.17**）；在任何給定時刻提供服務的實際節點可以透過 *TON DHT* 查找，從 TON 區塊鏈中發佈的資訊開始（參見 **3.2.12**）。服務可以在 TON 區塊鏈中建立智慧合約，以向其客戶提供某些保證（參見 **4.1.7**）。

- *TON DNS*（參見 **4.3.1**），一種為帳戶、智慧合約、服務和網路節點分配人類可讀名稱的服務。

- *TON* 支付（參見第 **5** 章），一個用於微支付、微支付通道和微支付通道網路的平台。它可用於快速的鏈下價值轉移，以及為 *TON* 服務提供的服務付費。

- TON 將允許輕鬆整合第三方訊息傳遞和社交網路應用程式，從而使區塊鏈技術和分散式服務最終對普通使用者可用且易於存取（參見 **4.3.24**），而不僅僅是少數早期加密貨幣採用者。我們將在我們的另一個專案 Telegram Messenger（參見 **4.3.19**）中提供此類整合的範例。

雖然 TON 區塊鏈是 TON 專案的核心，而其他組件可能被認為是為區塊鏈扮演支援角色，但它們本身具有有用且有趣的功能。結合起來，它們使平台能夠託管比僅使用 TON 區塊鏈更多樣化的應用程式（參見 **2.9.13** 和 **4.1**）。

2 TON 區塊鏈

我們從描述 Telegram 開放網路（TON）區塊鏈開始，這是該專案的核心組件。我們這裡的方法是「自上而下」的：我們首先對整體進行概述，然後提供每個組件的更多細節。

為了簡單起見，我們在這裡談論該 TON 區塊鏈，儘管原則上該區塊鏈協議的幾個實例可能獨立執行（例如，由於硬分叉的結果）。我們只考慮其中之一。

2.1 TON 區塊鏈作為 2-區塊鏈的集合

TON 區塊鏈實際上是一個區塊鏈的集合（甚至是一個區塊鏈的區塊鏈，或 *2*-區塊鏈的集合——這一點將在稍後的 **2.1.17** 中澄清），因為沒有單一的區塊鏈專案能夠實現我們每秒處理數百萬筆交易的目標，而不是目前標準的每秒數十筆交易。

2.1.1. 區塊鏈類型列表. 此集合中的區塊鏈包括：

- 唯一的主區塊鏈，或簡稱主鏈，包含有關協議和其參數當前值的一般資訊、驗證者及其權益的集合、目前活躍的工作鏈及其「分片」的集合，以及最重要的，所有工作鏈和分片鏈最新區塊的雜湊集合。

- 數個（最多 2^{32} 個）工作區塊鏈，或簡稱工作鏈，實際上是「工作馬」，包含價值轉移和智慧合約交易。不同的工作鏈可能有不同的「規則」，意味著不同的帳戶地址格式、不同的交易格式、不同的智慧合約虛擬機（VM）、不同的基礎加密貨幣等等。然而，它們都必須滿足某些基本的互通性標準，以使不同工作鏈之間的互動成為可能且相對簡單。在這方面，TON 區塊鏈是異質的（參見 **2.8.8**），類似於 EOS（參見 **2.9.7**）和 PolkaDot（參見 **2.9.8**）專案。

- 每個工作鏈又細分為最多 2^{60} 個分片區塊鏈，或簡稱分片鏈，與工作鏈本身具有相同的規則和區塊格式，但僅負責帳戶的子集，取決於帳戶地址的前幾個（最高有效）位元。換句話說，一種分片形式內建於系統中（參見 **2.8.12**）。由於所有這些分片鏈共享共同的區塊格式和規則，TON 區塊鏈在這方面是同質的（參見 **2.8.8**），類似於 Ethereum 擴展提案中討論的內容。[1]

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

- 分片鏈（和主鏈）中的每個區塊實際上不僅僅是一個區塊，而是一個小區塊鏈。通常，這個「區塊區塊鏈」或「垂直區塊鏈」恰好由一個區塊組成，那麼我們可能認為這只是分片鏈的相應區塊（在這種情況下也稱為「水平區塊鏈」）。然而，如果有必要修復不正確的分片鏈區塊，則會將新區塊提交到「垂直區塊鏈」中，包含無效「水平區塊鏈」區塊的替代品，或「區塊差異」，僅包含需要更改的該區塊先前版本的那些部分的描述。這是 TON 特有的機制，用於替換檢測到的無效區塊而不對所有相關的分片鏈進行真正的分叉；它將在 **2.1.17** 中更詳細地解釋。現在，我們只是注意到每個分片鏈（和主鏈）不是傳統的區塊鏈，而是一個區塊鏈的區塊鏈，或 *2D*-區塊鏈，或簡稱 *2*-區塊鏈。

2.1.2. 無限分片範式. 幾乎所有的區塊鏈分片提案都是「自上而下」的：人們首先想像一個單一的區塊鏈，然後討論如何將其拆分為幾個互動的分片鏈以提高效能並實現可擴展性。

TON 的分片方法是「自下而上」的，解釋如下。

想像一下，分片已經被推向極端，因此每個分片鏈中恰好保留一個帳戶或智慧合約。然後我們有大量的「帳戶鏈」，每個都只描述一個帳戶的狀態和狀態轉換，並相互傳送帶有價值的訊息以轉移價值和資訊。

當然，擁有數億條區塊鏈是不切實際的，因為每條區塊鏈中的更新（即新區塊）通常很少出現。為了更有效地實作它們，我們將這些「帳戶鏈」分組為「分片鏈」，因此分片鏈的每個區塊本質上是分配給該分片的帳戶鏈區塊的集合。因此，「帳戶鏈」在「分片鏈」內僅具有純粹的虛擬或邏輯存在。

我們稱這種觀點為無限分片範式。它解釋了 TON 區塊鏈的許多設計決策。

2.1.3. 訊息。即時超立方體路由. 無限分片範式指示我們將每個帳戶（或智慧合約）視為彷彿它本身就在自己的分片鏈中。那麼一個帳戶可能影響另一個帳戶狀態的唯一方式就是向其傳送一個訊息（這是所謂 Actor 模型的特殊實例，帳戶作為 Actor；參見 **2.4.2**）。因此，帳戶之間（以及分片鏈之間，因為來源和目的帳戶通常位於不同的分片鏈中）的訊息系統對於像 TON 區塊鏈這樣的可擴展系統至關重要。事實上，TON 區塊鏈的一個新穎功能，稱為即時超立方體路由（參見 **2.4.20**），使其能夠將在一個分片鏈的區塊中建立的訊息傳遞並處理到目的分片鏈的下一個區塊中，無論系統中的分片鏈總數如何。

2.1.4. 主鏈、工作鏈和分片鏈的數量. TON 區塊鏈恰好包含一個主鏈。然而，系統可能容納最多 2^{32} 個工作鏈，每個又細分為最多 2^{60} 個分片鏈。

2.1.5. 工作鏈可以是虛擬區塊鏈，而非真正的區塊鏈. 由於工作鏈通常細分為分片鏈，工作鏈的存在是「虛擬的」，意味著它不是下面 **2.2.1** 中提供的一般定義意義上的真正區塊鏈，而只是分片鏈的集合。當只有一個分片鏈對應於一個工作鏈時，這個唯一的分片鏈可能與工作鏈識別，在這種情況下，工作鏈至少在一段時間內成為「真正的」區塊鏈，從而獲得與傳統單區塊鏈設計的表面相似性。然而，無限分片範式（參見 **2.1.2**）告訴我們，這種相似性確實是表面的：潛在的大量「帳戶鏈」可以暫時分組為一個區塊鏈，這只是一個巧合。

2.1.6. 工作鏈的識別. 每個工作鏈由其編號或工作鏈識別符($workchain\_id : uint_{32}$）識別，這只是一個無符號 32 位元整數。工作鏈是由主鏈中的特殊交易建立的，定義（先前未使用的）工作鏈識別符和工作鏈的正式描述，至少足以用於該工作鏈與其他工作鏈的互動以及對該工作鏈區塊的表面驗證。

2.1.7. 新工作鏈的建立和啟動. 新工作鏈的建立可以由社群中的任何成員發起，準備支付發佈新工作鏈正式規範所需的（高昂的）主鏈交易費用。然而，為了使新工作鏈變得活躍，需要驗證者的三分之二共識，因為他們需要升級其軟體以處理新工作鏈的區塊，並透過特殊的主鏈交易發出信號表示他們已準備好與新工作鏈合作。對新工作鏈啟動感興趣的一方可能透過智慧合約分配的一些獎勵，為驗證者提供一些激勵以支援新工作鏈。

2.1.8. 分片鏈的識別. 每個分片鏈由一對 $(w, s) = (workchain\_id, shard\_prefix)$ 識別，其中 $workchain\_id : uint_{32}$ 識別對應的工作鏈，而 $shard\_prefix : \mathbf{2}^{0\ldots60}$ 是長度最多為 60 的位元字串，定義該分片鏈負責的帳戶子集。也就是說，所有以 $shard\_prefix$ 開頭（即，將 $shard\_prefix$ 作為最高有效位元）的 $account\_id$ 的帳戶將被分配到該分片鏈。

2.1.9. 帳戶鏈的識別. 回想一下，帳戶鏈只有虛擬存在（參見 **2.1.2**）。然而，它們有一個自然的識別符——即 $(workchain\_id, account\_id)$——因為任何帳戶鏈都包含有關恰好一個帳戶（簡單帳戶或智慧合約——這裡的區別並不重要）的狀態和更新的資訊。

2.1.10. 分片鏈的動態拆分和合併；參見 2.7. 一個不太複雜的系統可能使用靜態分片——例如，透過使用 $account\_id$ 的前八位元來選擇 256 個預定義分片之一。

TON 區塊鏈的一個重要功能是它實作了動態分片，意味著分片的數量不是固定的。相反，如果滿足某些正式條件（本質上，如果原始分片上的交易負載在很長一段時間內足夠高），分片 (w, s) 可以自動細分為分

片 $(w, s.0)$ 和 $(w, s.1)$。相反，如果負載在一段時間內保持太低，則分片 $(w, s.0)$ 和 $(w, s.1)$ 可以自動合併回分片 (w, s)。

最初，僅為工作鏈 w 建立一個分片 $(w, \emptyset)$。稍後，如果並且當這變得必要時（參見 **2.7.6** 和 **2.7.8**），它會被細分為更多分片。

2.1.11. 基本工作鏈或工作鏈零. 雖然最多可以定義 2^{32} 個具有其特定規則和交易的工作鏈，但我們最初只定義了一個，$workchain\_id = 0$。這個工作鏈，稱為工作鏈零或基本工作鏈，是用於處理 *TON* 智慧合約和轉移 *TON* 幣（也稱為 *Gram*；參見附錄 A）的工作鏈。大多數應用程式可能只需要工作鏈零。基本工作鏈的分片鏈將被稱為基本分片鏈。

2.1.12. 區塊生成間隔. 我們預期每個分片鏈和主鏈大約每五秒生成一個新區塊。這將導致相當小的交易確認時間。所有分片鏈的新區塊大約同時生成；主鏈的新區塊大約在一秒後生成，因為它必須包含所有分片鏈最新區塊的雜湊。

2.1.13. 使用主鏈使工作鏈和分片鏈緊密耦合. 一旦分片鏈區塊的雜湊被納入主鏈的區塊中，該分片鏈區塊及其所有祖先都被視為「規範的」，意味著它們可以從所有分片鏈的後續區塊中被引用為固定且不可變的東西。事實上，每個新的分片鏈區塊都包含最新主鏈區塊的雜湊，而從該主鏈區塊引用的所有分片鏈區塊都被新區塊視為不可變的。

本質上，這意味著在分片鏈區塊中提交的交易或訊息可以安全地在其他分片鏈的下一個區塊中使用，而無需等待（例如）二十個確認（即在同一區塊鏈的原始區塊之後生成的二十個區塊），然後才能轉發訊息或根據先前的交易採取其他行動，這在大多數提議的「鬆散耦合」系統（參見 **2.8.14**）中很常見，例如 EOS。這種在提交後僅五秒鐘就能在其他分片鏈中使用交易和訊息的能力，是我們相信我們的「緊密耦合」系統（同類首創）將能夠提供前所未有的效能（參見 **2.8.12** 和 **2.8.14**）的原因之一。

2.1.14. 主鏈區塊雜湊作為全域狀態. 根據 **2.1.13**，最後一個主鏈區塊的雜湊從外部觀察者的角度完全決定了系統的整體狀態。不需要單獨監控所有分片鏈的狀態。

2.1.15. 驗證者生成新區塊；參見 2.6. TON 區塊鏈使用權益證明（PoS）方法在分片鏈和主鏈中生成新區塊。這意味著有一組（比如說，最多幾百個）驗證者——特殊節點，它們透過特殊的主鏈交易存入權益（大量 TON 幣），以便有資格進行新區塊生成和驗證。

然後以確定性偽隨機方式將驗證者的較小子集分配給每個分片 (w, s)，大約每 1024 個區塊更換一次。這個驗證者子集建議下一個分片鏈區塊是什麼，並透過將客戶端提出的合適交易收集到新的有效區塊候選中來達成

共識。對於每個區塊，驗證者上有一個偽隨機選擇的順序，以確定誰的區塊候選在每次輪次中具有最高優先順序被提交。

驗證者和其他節點檢查提議的區塊候選的有效性；如果驗證者簽署了無效的區塊候選，它可能會因失去部分或全部權益，或被暫時從驗證者集合中停權一段時間而自動受到懲罰。之後，驗證者應該就下一個區塊的選擇達成共識，本質上透過 BFT（拜占庭容錯；參見 **2.8.4**）共識協議的有效變體，類似於 PBFT [4] 或 Honey Badger BFT [11]。如果達成共識，則建立新區塊，驗證者之間分配所包含交易的交易費用，加上一些新建立（「鑄造的」）幣。

每個驗證者可以被選舉參與多個驗證者子集；在這種情況下，預期它會並行執行所有驗證和共識演算法。

在生成所有新的分片鏈區塊或超時後，生成新的主鏈區塊，包括所有分片鏈最新區塊的雜湊。這是透過所有驗證者的 BFT 共識完成的。[2]

有關 TON PoS 方法及其經濟模型的更多細節在 **2.6** 節中提供。

2.1.16. 主鏈的分叉. 我們的緊密耦合方法產生的一個複雜性是，切換到主鏈中的不同分叉幾乎必然需要切換到至少某些分片鏈中的另一個分叉。另一方面，只要主鏈中沒有分叉，分片鏈中的分叉甚至是不可能的，因為分片鏈替代分叉中的區塊不能透過將其雜湊納入主鏈區塊而成為「規範的」。

一般規則是如果主鏈區塊 B' 是 B 的前驅，B' 包含 (w,s)-分片鏈區塊 $B'_{w,s}$ 的雜湊 $\mathit{HASH}(B'_{w,s})$，而 B 包含雜湊 $\mathit{HASH}(B_{w,s})$，則 $B'_{w,s}$ **必須**是 $B_{w,s}$ 的前驅；否則，主鏈區塊 B 無效。

我們預期主鏈分叉會很罕見，幾乎不存在，因為在 TON 區塊鏈採用的 BFT 範式中，它們只能在大多數驗證者行為不正確的情況下發生（參見 **2.6.1** 和 **2.6.15**），這將意味著違規者的重大權益損失。因此，不應該預期分片鏈中會出現真正的分叉。相反，如果檢測到無效的分片鏈區塊，它將透過 2-區塊鏈的「垂直區塊鏈」機制進行糾正（參見 **2.1.17**），該機制可以在不分叉「水平區塊鏈」（即分片鏈）的情況下實現此目標。同樣的機制也可用於修復主鏈區塊中的非致命錯誤。

2.1.17. 糾正無效的分片鏈區塊. 通常，只有有效的分片鏈區塊才會被提交，因為分配給分片鏈的驗證者在提交新區塊之前必須達成三分之二的拜占庭共識。然而，系統必須允許檢測先前提交的無效區塊並對其進行糾正。

當然，一旦發現無效的分片鏈區塊——無論是由驗證者（不一定分配給此分片鏈）還是由「漁夫」（系統中任何進行了一定存款以能夠對區塊有效性提出質疑的節點；參見 **2.6.4**）發現——無效性宣告及其證明將被提交到

[2]實際上，三分之二的權益就足以達成共識，但會努力收集盡可能多的簽名。

主鏈中，簽署無效區塊的驗證者將因失去部分權益和/或被暫時從驗證者集合中停權而受到懲罰（後一措施對於攻擊者竊取原本良性驗證者的私有簽名金鑰的情況很重要）。

然而，這還不夠，因為由於先前提交的無效分片鏈區塊，系統（TON 區塊鏈）的整體狀態變得無效。這個無效區塊必須被更新的有效版本替換。

大多數系統會透過「回滾」到此分片鏈中無效區塊之前的最後一個區塊，以及每個其他分片鏈中不受從無效區塊傳播的訊息影響的最後一個區塊，並從這些區塊建立新的分叉來實現這一點。這種方法的缺點是大量原本正確並已提交的交易會突然被回滾，並且不清楚它們以後是否會被包含。

TON 區塊鏈透過使每個分片鏈和主鏈（「水平區塊鏈」）的每個「區塊」本身成為一個小區塊鏈（「垂直區塊鏈」），包含此「區塊」的不同版本或它們的「差異」來解決這個問題。通常，垂直區塊鏈恰好由一個區塊組成，分片鏈看起來像傳統的區塊鏈。然而，一旦區塊的無效性被確認並提交到主鏈區塊中，無效區塊的「垂直區塊鏈」就允許在垂直方向上透過新區塊增長，替換或編輯無效區塊。新區塊由有關分片鏈的當前驗證者子集生成。

新「垂直」區塊有效的規則非常嚴格。特別是，如果無效區塊中包含的虛擬「帳戶鏈區塊」（參見 **2.1.2**）本身是有效的，則新垂直區塊必須保持不變。

一旦新的「垂直」區塊被提交到無效區塊之上，其雜湊就會在新的主鏈區塊中發佈（或者更確切地說，在新的「垂直」區塊中發佈，該區塊位於最初發佈無效分片鏈區塊雜湊的原始主鏈區塊之上），並且更改會進一步傳播到引用該區塊先前版本的任何分片鏈區塊（例如，那些從不正確區塊接收訊息的區塊）。這透過為先前引用「不正確」區塊的所有區塊在垂直區塊鏈中提交新的「垂直」區塊來修復；新的垂直區塊將引用最新（已糾正的）版本。同樣，嚴格的規則禁止更改實際上沒有受到影響（即，接收與先前版本中相同的訊息）的帳戶鏈。透過這種方式，修復不正確的區塊會產生「漣漪」，最終傳播到所有受影響分片鏈的最新區塊；這些變化也反映在新的「垂直」主鏈區塊中。

一旦「歷史改寫」漣漪到達最新區塊，新的分片鏈區塊就只以一個版本生成，成為最新區塊版本的後繼者。這意味著它們從一開始就會包含對正確（最新）垂直區塊的引用。

主鏈狀態隱含地定義了一個映射，將每個「垂直」區塊鏈的第一個區塊的雜湊轉換為其最新版本的雜湊。這使客戶端能夠透過其第一個（通常是唯一的）區塊的雜湊來識別和定位任何垂直區塊鏈。

2.1.18. TON 幣和多幣種工作鏈. TON 區塊鏈支援最多 2^{32} 種不同的「加密貨幣」、「幣」或「代幣」，由 32 位元 $currency\_id$ 區分。新的加密貨幣可以透過主鏈中的特殊交易新增。每個工作鏈都有一個基本加密貨幣，並

且可以有幾種附加的加密貨幣。

有一種特殊的加密貨幣，其 $currency\_id = 0$，即 *TON* 幣，也稱為 *Gram*（參見附錄 A）。它是工作鏈零的基本加密貨幣。它也用於交易費用和驗證者權益。

原則上，其他工作鏈可能以其他代幣收取交易費用。在這種情況下，應提供一些將這些交易費用自動轉換為 Gram 的智慧合約。

2.1.19. 訊息傳遞和價值轉移. 屬於相同或不同工作鏈的分片鏈可以相互發送訊息。雖然允許的訊息的確切形式取決於接收工作鏈和接收帳戶（智慧合約），但有一些通用欄位使跨工作鏈訊息傳遞成為可能。特別是，每個訊息都可以附帶一些價值，以一定數量的 Gram（TON 幣）和/或其他註冊的加密貨幣的形式，只要它們被接收工作鏈宣告為可接受的加密貨幣。

這種訊息傳遞的最簡單形式是從一個（通常不是智慧合約）帳戶到另一個帳戶的價值轉移。

2.1.20. TON 虛擬機. *TON* 虛擬機，也縮寫為 *TON VM* 或 *TVM*，是用於在主鏈和基本工作鏈中執行智慧合約程式碼的虛擬機。其他工作鏈可以與 TVM 一起或代替 TVM 使用其他虛擬機。

這裡我們列出了它的一些功能。它們在 **2.3.12**、**2.3.14** 等處進一步討論。

- TVM 將所有資料表示為（*TVM*）單元的集合（參見 **2.3.14**）。每個單元包含最多 128 個資料位元組和最多 4 個對其他單元的引用。作為「一切皆為單元集合」理念（參見 **2.5.14**）的結果，這使 TVM 能夠在必要時處理與 TON 區塊鏈相關的所有資料，包括區塊和區塊鏈全域狀態。
- TVM 可以處理任意代數資料類型的值（參見 **2.3.12**），表示為 TVM 單元的樹或有向無環圖。然而，它對代數資料類型的存在是不可知的；它只是處理單元。
- TVM 內建支援雜湊映射（參見 **2.3.7**）。
- TVM 是一個堆疊機器。其堆疊保存 64 位元整數或單元引用。
- 支援 64 位元、128 位元和 256 位元運算。所有 n 位元運算操作都有三種形式：用於無符號整數、用於有符號整數和用於模 2^n 的整數（在後一種情況下沒有自動溢位檢查）。
- TVM 具有從 n 位元到 m 位元的無符號和有符號整數轉換，對於所有 $0 \leq m, n \leq 256$，具有溢位檢查。

- 所有算術運算預設執行溢位檢查，極大地簡化了智慧合約的開發。
- TVM 具有「乘後移位」和「移位後除」算術運算，中間值以更大的整數類型計算；這簡化了定點運算的實作。
- TVM 提供對位元字串和位元組字串的支援。
- 支援某些預定義曲線的 256 位元橢圓曲線密碼學（ECC），包括 Curve25519。
- 也支援某些橢圓曲線上的 Weil 配對，對於 zk-SNARK 的快速實作很有用。
- 支援流行的雜湊函數，包括 SHA256。
- TVM 可以處理 Merkle 證明（參見 **5.1.9**）。
- TVM 提供對「大型」或「全域」智慧合約的支援。此類智慧合約必須了解分片（參見 **2.3.18** 和 **2.3.16**）。通常（本地）智慧合約可以不關心分片。
- TVM 支援閉包。
- 「無脊無標籤 *G*-機器」[13] 可以輕鬆在 TVM 內部實作。

除了「TVM 組合語言」之外，還可以為 TVM 設計幾種高階語言。所有這些語言都將具有靜態類型並支援代數資料類型。我們設想以下可能性：

- 類似 Java 的命令式語言，每個智慧合約類似於一個單獨的類別。
- 惰性函數語言（想想 Haskell）。
- 嚴格求值函數語言（想想 ML）。

2.1.21. 可設定參數. TON 區塊鏈的一個重要功能是其許多參數是可設定的。這意味著它們是主鏈狀態的一部分，可以透過主鏈中的某些特殊提案/投票/結果交易進行更改，而無需任何硬分叉。更改此類參數將需要收集三分之二的驗證者投票以及超過一半願意參與投票過程的所有其他參與者的投票以支持該提案。

2.2 區塊鏈概論

2.2.1. 一般區塊鏈定義. 一般來說，任何（真正的）區塊鏈都是區塊的序列，每個區塊 B 包含對前一個區塊的引用 $\text{BLK-PREV}(B)$（通常透過將前一個區塊的雜湊包含在當前區塊的標頭中），以及交易的列表。每個交易描述全域區塊鏈狀態的某種轉換；區塊中列出的交易依序應用，以從舊狀態計算新狀態，舊狀態是評估前一個區塊後的結果狀態。

2.2.2. 與 TON 區塊鏈的相關性. 回想一下，*TON* 區塊鏈不是真正的區塊鏈，而是 2-區塊鏈的集合（即區塊鏈的區塊鏈；參見 **2.1.1**），因此上述內容不能直接應用於它。然而，我們從這些關於真正區塊鏈的概論開始，以將它們用作我們更複雜構造的構建塊。

2.2.3. 區塊鏈實例和區塊鏈類型. 人們經常使用區塊鏈一詞來表示一般的區塊鏈類型及其特定的區塊鏈實例，定義為滿足某些條件的區塊序列。例如，**2.2.1** 指的是區塊鏈實例。

透過這種方式，區塊鏈類型通常是區塊列表類型 $Block^*$（即有限序列）的「子類型」，由滿足某些相容性和有效性條件的區塊序列組成：

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

定義 *Blockchain* 的更好方法是說 *Blockchain* 是一個依賴對類型，由對 $(\mathbb{B}, v)$ 組成，第一個分量 $\mathbb{B} : Block^*$ 的類型為 $Block^*$（即區塊列表），第二個分量 $v : isValidBc(\mathbb{B})$ 是 $\mathbb{B}$ 有效性的證明或見證。透過這種方式，

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

我們在這裡使用從 [16] 借來的類型依賴和的符號。

2.2.4. 依賴類型理論、Coq 和 TL. 請注意，我們在這裡使用（Martin-Löf）依賴類型理論，類似於 Coq[3] 證明助手中使用的理論。依賴類型理論的簡化版本也用於 *TL*（類型語言），[4] 它將用於 TON 區塊鏈的正式規範，以描述所有資料結構的序列化以及區塊、交易等的佈局。

事實上，依賴類型理論提供了證明是什麼的有用形式化，當需要為某個區塊提供無效性證明時，這種形式化證明（或其序列化）可能會派上用場。

2.2.5. TL，或類型語言. 由於 TL（類型語言）將用於 TON 區塊、交易和網路資料報的正式規範，因此值得簡要討論。

[3] `https://coq.inria.fr`

[4] `https://core.telegram.org/mtproto/TL`

TL 是一種適合描述依賴代數類型的語言，允許具有數值（自然數）和類型參數。每個類型透過幾個建構子來描述。每個建構子都有一個（人類可讀的）識別符和一個名稱，即位元字串（預設為 32 位元整數）。除此之外，建構子的定義包含欄位列表及其類型。

建構子和類型定義的集合稱為 *TL-方案*。它通常保存在一個或多個帶有後綴 `.tl` 的檔案中。

TL-方案的一個重要功能是它們確定了一種明確的方式來序列化和反序列化所定義的代數類型的值（或物件）。也就是說，當需要將值序列化為位元組流時，首先序列化用於該值的建構子的名稱。然後是每個欄位的遞迴計算的序列化。

TL 先前版本的描述，適合將任意物件序列化為 32 位元整數序列，可在 `https://core.telegram.org/mtproto/TL` 獲得。正在為描述 TON 專案使用的物件序列化而開發稱為 *TL-B* 的新版本 TL。這個新版本可以將物件序列化為位元組流甚至位元流（而不僅僅是 32 位元整數），並支援序列化到 TVM 單元樹中（參見 **2.3.14**）。TL-B 的描述將成為 TON 區塊鏈正式規範的一部分。

2.2.6. 區塊和交易作為狀態轉換運算子. 通常，任何區塊鏈(類型)*Blockchain* 都有一個相關的全域狀態（類型）*State* 和一個交易（類型）*Transaction*。區塊鏈的語義在很大程度上由交易應用函數決定：

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

這裡 $X^{?}$ 表示 $\textsc{Maybe}\, X$，即將 Maybe 單子應用於類型 X 的結果。這類似於我們使用 X^{*} 表示 $\textsc{List}\, X$。本質上，類型 $X^{?}$ 的值要麼是類型 X 的值，要麼是表示缺少實際值的特殊值 $\perp$（想想空指標）。在我們的情況下，我們使用 $State^{?}$ 而不是 $State$ 作為結果類型，因為如果從某些原始狀態呼叫，交易可能無效（想想嘗試從帳戶提取比實際存在更多的錢）。

我們可能更喜歡 ev_trans' 的柯里化版本：

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

因為區塊本質上是交易列表，所以區塊評估函數

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

可以從 ev_trans 衍生。它接受一個區塊 $B : Block$ 和前一個區塊鏈狀態 $s : State$（可能包括前一個區塊的雜湊）並計算下一個區塊鏈狀態 $s' = ev\_block(B)(s) : State$，它要麼是真實狀態，要麼是表示無法計算下一個狀態的特殊值 $\perp$（即，如果從給定的起始狀態評估，則區塊無效——例如，區塊包含試圖從空帳戶借記的交易。）

2.2.7. 區塊序列號. 區塊鏈中的每個區塊 B 都可以透過其序列號 $\text{BLK-SEQNO}(B)$ 引用，從第一個區塊的零開始，每次傳遞到下一個區塊時遞增一。更正式地說，

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

請注意，在存在分叉的情況下，序列號不能唯一識別區塊。

2.2.8. 區塊雜湊. 引用區塊 B 的另一種方法是透過其雜湊 $\text{BLK-HASH}(B)$，這實際上是區塊 B 的標頭的雜湊（但是，區塊的標頭通常包含依賴於區塊 B 所有內容的雜湊）。假設所使用的雜湊函數沒有碰撞（或至少它們非常不可能），則區塊由其雜湊唯一識別。

2.2.9. 雜湊假設. 在區塊鏈演算法的形式分析中，我們假設所使用的 k 位元雜湊函數 $\text{HASH} : \mathit{Bytes}^* \to \mathbf{2}^k$ 沒有碰撞：

$$\text{HASH}(s) = \text{HASH}(s') \Rightarrow s = s' \quad \text{對於任何 } s、s' \in \mathit{Bytes}^* \tag{7}$$

這裡 $\mathit{Bytes} = \{0 \dots 255\} = \mathbf{2}^8$ 是位元組類型，或所有位元組值的集合，Bytes^* 是任意（有限）位元組列表的類型或集合；而 $\mathbf{2} = \{0, 1\}$ 是位元類型，$\mathbf{2}^k$ 是所有 k 位元序列（即 k 位元數字）的集合（或實際上是類型）。

當然，(7) 在數學上是不可能的，因為從無限集合到有限集合的映射不能是單射的。更嚴格的假設是

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

然而，這對於證明來說不太方便。如果 (8) 在證明中最多使用 N 次，且 $2^{-k}N < \epsilon$ 對於某個小的 ϵ（例如，$\epsilon = 10^{-18}$），我們可以像 (7) 為真一樣進行推理，只要我們接受失敗機率 ϵ（即，最終結論將以至少 $1 - \epsilon$ 的機率為真）。

最後備註：為了使 (8) 的機率陳述真正嚴格，必須在所有位元組序列的集合 Bytes^* 上引入機率分佈。一種方法是假設相同長度 l 的所有位元組序列等可能，並將觀察到長度為 l 的序列的機率設定為 $p^l - p^{l+1}$，其中某個 $p \to 1-$。然後 (8) 應該被理解為條件機率 $P\big(\text{HASH}(s) = \text{HASH}(s') | s \neq s'\big)$ 當 p 從下方趨近於一時的極限。

2.2.10. TON 區塊鏈使用的雜湊. 我們目前為 TON 區塊鏈使用 256 位元 SHA256 雜湊。如果它被證明比預期的弱，它可以在未來被另一個雜湊函數替換。雜湊函數的選擇是協議的可設定參數，因此可以在沒有硬分叉的情況下更改，如 **2.1.21** 中所解釋的。

2.3 區塊鏈狀態、帳戶和雜湊映射

我們在上面注意到，任何區塊鏈都定義了某個全域狀態，每個區塊和每個交易都定義了該全域狀態的轉換。在這裡，我們描述 TON 區塊鏈使用的全域狀態。

2.3.1. 帳戶 ID. TON 區塊鏈——或至少其主鏈和工作鏈零——使用的基本帳戶 ID 是 256 位元整數，假定為特定橢圓曲線的 256 位元橢圓曲線密碼學（ECC）的公鑰。透過這種方式，

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

這裡 $Account$ 是帳戶類型，而 $account\_id : Account$ 是類型 $Account$ 的特定變數。

其他工作鏈可以使用其他帳戶 ID 格式，256 位元或其他。例如，可以使用 Bitcoin 風格的帳戶 ID，等於 ECC 公鑰的 SHA256。

然而，帳戶 ID 的位元長度 l 必須在工作鏈建立期間（在主鏈中）固定，並且必須至少為 64，因為 $account\_id$ 的前 64 位元用於分片和訊息路由。

2.3.2. 主要組件：雜湊映射. TON 區塊鏈狀態的主要組件是雜湊映射。在某些情況下，我們考慮（部分定義的）「映射」$h : \mathbf{2}^n \dashrightarrow \mathbf{2}^m$。更一般地，我們可能對複合類型 X 的雜湊映射 $h : \mathbf{2}^n \dashrightarrow X$ 感興趣。然而，來源（或索引）類型幾乎總是 $\mathbf{2}^n$。

有時，我們有一個「預設值」$empty : X$，雜湊映射 $h : \mathbf{2}^n \to X$ 由其「預設值」$i \mapsto empty$ 「初始化」。

2.3.3. 範例：TON 帳戶餘額. 一個重要的範例由 TON 帳戶餘額給出。它是一個雜湊映射

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

將 $Account = \mathbf{2}^{256}$ 映射到類型為 $uint_{128} = \mathbf{2}^{128}$ 的 Gram（TON 幣）餘額。此雜湊映射的預設值為零，意味著最初（在處理第一個區塊之前）所有帳戶的餘額為零。

2.3.4. 範例：智慧合約持久儲存. 另一個範例由智慧合約持久儲存給出，可以（非常粗略地）表示為雜湊映射

$$storage : \mathbf{2}^{256} \dashrightarrow \mathbf{2}^{256} \tag{11}$$

此雜湊映射的預設值也為零，意味著未初始化的持久儲存單元假定為零。

2.3.5. 範例：所有智慧合約的持久儲存. 由於我們有多個智慧合約，由 $account\_id$ 區分，每個都有其單獨的持久儲存，我們實際上必須有一個雜湊映射

$$Storage : Account \dashrightarrow (\mathbf{2}^{256} \dashrightarrow \mathbf{2}^{256}) \tag{12}$$

將智慧合約的 $account\_id$ 映射到其持久儲存。

2.3.6. 雜湊映射類型. 雜湊映射不僅僅是抽象的（部分定義的）函數 $\mathbf{2}^n \dashrightarrow X$；它有一個特定的表示。因此，我們假設我們有一個特殊的雜湊映射類型

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

對應於編碼（部分）映射 $\mathbf{2}^n \dashrightarrow X$ 的資料結構。我們也可以寫

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

或

$$Hashmap : nat \to Type \to Type \tag{15}$$

我們總是可以將 $h : Hashmap(n, X)$ 轉換為映射 $hget(h) : \mathbf{2}^n \to X^?$。從現在開始，我們通常寫 $h[i]$ 而不是 $hget(h)(i)$：

$$h[i] :\equiv hget(h)(i) : X^? \quad \text{對於任何 } i : \mathbf{2}^n \text{、} h : Hashmap(n, X) \tag{16}$$

2.3.7. 雜湊映射類型定義為 Patricia 樹. 邏輯上，可以將 $Hashmap(n, X)$ 定義為深度為 n 的（不完整的）二元樹，邊標籤為 0 和 1，葉子中的值類型為 X。描述相同結構的另一種方式是作為長度等於 n 的二進位字串的（按位元）字典樹。

在實踐中，我們更喜歡使用此字典樹的緊湊表示，透過將每個只有一個子節點的頂點與其父節點壓縮。所得的表示稱為 *Patricia* 樹或二進位基數樹。現在每個中間頂點恰好有兩個子節點，由兩個非空的二進位字串標記，左子節點以零開頭，右子節點以一開頭。

換句話說，Patricia 樹中有兩種類型的（非根）節點：

- $\textsc{Leaf}(x)$，包含類型 X 的值 x。
- $\textsc{Node}(l, s_l, r, s_r)$，其中 l 是（對）左子節點或子樹的引用，s_l 是標記連接此頂點與其左子節點的邊的位元字串（總是以 0 開頭），r 是右子樹，s_r 是標記到右子節點的邊的位元字串（總是以 1 開頭）。

第三種類型的節點，僅在 Patricia 樹的根處使用一次，也是必需的：

- $\textsc{Root}(n, s_0, t)$，其中 n 是 $\mathit{Hashmap}(n, X)$ 的索引位元字串的公共長度，s_0 是所有索引位元字串的公共前綴，t 是對 $\textsc{Leaf}$ 或 $\textsc{Node}$ 的引用。

如果我們想允許 Patricia 樹為空，則將使用第四種類型的（根）節點：

- $\textsc{EmptyRoot}(n)$，其中 n 是所有索引位元字串的公共長度。

我們透過以下方式定義 Patricia 樹的高度：

$$\textsc{height}(\textsc{Leaf}(x)) = 0 \tag{17}$$

$$\textsc{height}\big(\textsc{Node}(l, s_l, r, s_r)\big) = \textsc{height}(l) + \textsc{len}(s_l) = \textsc{height}(r) + \textsc{len}(s_r) \tag{18}$$

$$\textsc{height}\big(\textsc{Root}(n, s_0, t)\big) = \textsc{len}(s_0) + \textsc{height}(t) = n \tag{19}$$

最後兩個公式中的最後兩個表示式必須相等。我們使用高度為 n 的 Patricia 樹來表示類型 $\mathit{Hashmap}(n, X)$ 的值。

如果樹中有 N 個葉子（即，我們的雜湊映射包含 N 個值），則恰好有 $N - 1$ 個中間頂點。插入新值總是涉及透過在中間插入新頂點來拆分現有邊，並將新葉子新增為該新頂點的另一個子節點。從雜湊映射中刪除值會執行相反的操作：刪除葉子及其父節點，父節點的父節點和其另一個子節點直接連接。

2.3.8. Merkle-Patricia 樹. 在處理區塊鏈時，我們希望能夠透過將 Patricia 樹（即雜湊映射）及其子樹歸約為單一雜湊值來比較它們。實現這一點的經典方法由 Merkle 樹給出。本質上，我們想要描述一種在二進位字串定義的雜湊函數 $\textsc{Hash}$ 的幫助下，對類型 $\mathit{Hashmap}(n, X)$ 的物件 h 進行雜湊的方法，前提是我們知道如何計算物件 $x : X$ 的雜湊 $\textsc{Hash}(x)$（例如，透過將雜湊函數 $\textsc{Hash}$ 應用於物件 x 的二進位序列化）。

可以如下遞迴地定義 $\textsc{Hash}(h)$：

$$\textsc{Hash}\big(\textsc{Leaf}(x)\big) := \textsc{Hash}(x) \tag{20}$$

$$\textsc{Hash}\big(\textsc{Node}(l, s_l, r, s_r)\big) := \textsc{Hash}\big(\textsc{Hash}(l).\,\textsc{Hash}(r).\,\textsc{code}(s_l).\,\textsc{code}(s_r)\big) \tag{21}$$

$$\textsc{Hash}\big(\textsc{Root}(n, s_0, t)\big) := \textsc{Hash}\big(\textsc{code}(n).\,\textsc{code}(s_0).\,\textsc{Hash}(t)\big) \tag{22}$$

這裡 $s.t$ 表示（位元）字串 s 和 t 的連接，$\textsc{code}(s)$ 是所有位元字串 s 的前綴碼。例如，可以將 0 編碼為 10，將 1 編碼為 11，將字串結尾編碼為 0。

[5]

我們稍後將看到（參見 **2.3.12** 和 **2.3.14**），這是任意（依賴）代數類型值的遞迴定義雜湊的（稍微調整的）版本。

2.3.9. 重新計算 Merkle 樹雜湊. 這種遞迴定義 $\text{HASH}(h)$ 的方法，稱為 *Merkle* 樹雜湊，其優點是，如果明確儲存 $\text{HASH}(h')$ 與每個節點 h'（產生稱為 *Merkle* 樹的結構，或在我們的情況下，*Merkle–Patricia* 樹），則在向雜湊映射新增、從雜湊映射刪除或更改元素時，只需要重新計算最多 n 個雜湊。

透過這種方式，如果透過合適的 Merkle 樹雜湊表示全域區塊鏈狀態，則在每次交易後重新計算此狀態雜湊很容易。

2.3.10. Merkle 證明. 在所選雜湊函數 HASH 的「單射性」假設 (7) 下，可以構造一個證明，對於 $\text{HASH}(h)$ 的給定值 z，$h : \mathit{Hashmap}(n, X)$，對於某些 $i : \mathbf{2}^n$ 和 $x : X$，有 $\mathit{hget}(h)(i) = x$。這樣的證明將由 Merkle–Patricia 樹中從對應於 i 的葉子到根的路徑組成，並附加此路徑上出現的所有節點的所有兄弟節點的雜湊。

透過這種方式，輕節點[6]只知道某個雜湊映射 h 的 $\text{HASH}(h)$ 值（例如，智慧合約持久儲存或全域區塊鏈狀態）可能從完整節點[7]請求不僅是值 $x = h[i] = \mathit{hget}(h)(i)$，還請求這樣的值以及從已知值 $\text{HASH}(h)$ 開始的 Merkle 證明。然後，在假設 (7) 下，輕節點可以自己檢查 x 確實是 $h[i]$ 的正確值。

在某些情況下，客戶端可能想要獲得值 $y = \text{HASH}(x) = \text{HASH}(h[i])$——例如，如果 x 本身非常大（例如，雜湊映射本身）。然後可以提供 (i, y) 的 Merkle 證明。如果 x 也是雜湊映射，則可以從完整節點獲得從 $y = \text{HASH}(x)$ 開始的第二個 Merkle 證明，以提供值 $x[j] = h[i][j]$ 或只是其雜湊。

2.3.11. Merkle 證明對 TON 等多鏈系統的重要性. 請注意，節點通常不能成為 TON 環境中存在的所有分片鏈的完整節點。它通常只是某些分片鏈的完整節點——例如，包含其自己帳戶、它感興趣的智慧合約的分片鏈，或者該節點已被分配為驗證者的分片鏈。對於其他分片鏈，它必須是輕節

[5]可以證明，對於具有隨機或連續索引的 Patricia 樹的大約一半邊標籤，此編碼是最佳的。剩餘的邊標籤可能很長（即，幾乎 256 位元長）。因此，邊標籤的近似最佳編碼是對「短」位元字串使用帶前綴 0 的上述碼，並對「長」位元字串編碼 1，然後是包含位元字串 s 的長度 $l = |s|$ 的九個位元，然後是 s 的 l 個位元（其中 $l \geq 10$）。

[6]輕節點是不追蹤分片鏈完整狀態的節點；相反，它保留最少的資訊，例如最近幾個區塊的雜湊，並在需要檢查完整狀態的某些部分時依賴從完整節點獲得的資訊。

[7]完整節點是追蹤有關分片鏈的完整最新狀態的節點。

點——否則儲存、計算和網路頻寬需求將是令人望而卻步的。這意味著這樣的節點不能直接檢查關於其他分片鏈狀態的斷言；它必須依賴從這些分片鏈的完整節點獲得的 Merkle 證明，這與自己檢查一樣安全，除非 (7) 失敗（即，發現雜湊碰撞）。

2.3.12. TON VM 的特殊性. TON VM 或 TVM（Telegram 虛擬機），用於在主鏈和工作鏈零中執行智慧合約，與受 EVM（Ethereum 虛擬機）啟發的慣用設計有很大不同：它不僅處理 256 位元整數，而且實際上處理（幾乎）任意的「記錄」、「結構」或「和-積類型」，使其更適合執行以高階（特別是函數式）語言編寫的程式碼。本質上，TVM 使用標記的資料類型，與 Prolog 或 Erlang 實作中使用的資料類型沒有什麼不同。

首先可以想像 TVM 智慧合約的狀態不僅是雜湊映射 $\mathbf{2}^{256} \to \mathbf{2}^{256}$ 或 $\mathit{Hashmap}(256, \mathbf{2}^{256})$，而是（作為第一步）$\mathit{Hashmap}(256, X)$，其中 X 是具有多個建構子的類型，使其能夠儲存除 256 位元整數之外的其他資料結構，特別是其他雜湊映射 $\mathit{Hashmap}(256, X)$。這意味著 TVM（持久或臨時）儲存的單元——或 TVM 智慧合約程式碼中的變數或陣列的元素——可能不僅包含整數，還包含整個新的雜湊映射。當然，這意味著單元不僅包含 256 位元，還包含（例如）8 位元標籤，描述這 256 位元應該如何解譯。

事實上，值不需要精確地為 256 位元。TVM 使用的值格式由原始位元組序列和對其他結構的引用組成，以任意順序混合，在合適的位置插入一些描述符位元組以能夠區分指標和原始資料（例如，字串或整數）；參見 **2.3.14**。

此原始值格式可用於實作任意和-積代數類型。在這種情況下，值將首先包含一個原始位元組，描述正在使用的「建構子」（從高階語言的角度），然後是其他「欄位」或「建構子參數」，由原始位元組和對其他結構的引用組成，取決於所選的建構子（參見 **2.2.5**）。然而，TVM 不知道建構子及其參數之間的對應關係；位元組和引用的混合由某些描述符位元組明確描述。[8]

Merkle 樹雜湊擴展到任意這樣的結構：要計算這樣的結構的雜湊，所有引用都遞迴地替換為所引用物件的雜湊，然後計算所得位元組字串（包括描述符位元組）的雜湊。

透過這種方式，**2.3.8** 中描述的雜湊映射的 Merkle 樹雜湊只是應用於具有兩個建構子的類型 $\mathit{Hashmap}(n, X)$ 的任意（依賴）代數資料類型雜湊的特例。[9]

[8]任何 TVM 單元中存在的這兩個描述符位元組僅描述引用的總數和原始位元組的總數；引用保持在一起，在所有原始位元組之前或之後。

[9]實際上，Leaf 和 Node 是輔助類型 $\mathit{HashmapAux}(n, X)$ 的建構子。類型 $\mathit{Hashmap}(n, X)$ 具有建構子 Root 和 EmptyRoot，其中 Root 包含類型

2.3.13. TON 智慧合約的持久儲存. TON 智慧合約的持久儲存本質上由其「全域變數」組成，在呼叫智慧合約之間保留。因此，它只是一個「積」、「元組」或「記錄」類型，由正確類型的欄位組成，每個對應一個全域變數。如果全域變數太多，則由於對 TON 單元大小的全域限制，它們無法容納在一個 TON 單元中。在這種情況下，它們被分成幾個記錄並組織成樹，本質上成為「積的積」或「積的積的積」類型，而不僅僅是積類型。

2.3.14. TVM 單元. 最終，TON VM 將所有資料保存在（*TVM*）單元的集合中。每個單元首先包含兩個描述符位元組，指示此單元中存在多少個原始資料位元組（最多 128 個）以及存在多少個對其他單元的引用（最多四個）。然後是這些原始資料位元組和引用。每個單元恰好被引用一次，因此我們可以在每個單元中包含對其「父單元」（引用此單元的唯一單元）的引用。然而，此引用不需要是明確的。

透過這種方式，TON 智慧合約的持久資料儲存單元被組織成樹，[10]在智慧合約描述中保留對此樹根的引用。如果必要，從葉子開始遞迴計算整個持久儲存的 Merkle 樹雜湊，然後只需將單元中的所有引用替換為被引用單元的遞迴計算的雜湊，然後計算由此獲得的位元組字串的雜湊。

2.3.15. 任意代數類型值的廣義 Merkle 證明. 由於 TON VM 透過由（TVM）單元組成的樹表示任意代數類型的值，並且每個單元都有明確定義的（遞迴計算的）Merkle 雜湊，實際上取決於以此單元為根的整個子樹，我們可以為任意代數類型的值（部分）提供「廣義 Merkle 證明」，旨在證明具有已知 Merkle 雜湊的樹的某個子樹採用特定值或具有特定雜湊的值。這概括了 **2.3.10** 的方法，其中僅考慮了 $x[i] = y$ 的 Merkle 證明。

2.3.16. TON VM 資料結構中對分片的支援. 我們剛剛概述了 TON VM 如何在不過於複雜的情況下，在高階智慧合約語言中支援任意（依賴）代數資料類型。然而，大型（或全域）智慧合約的分片需要在 TON VM 層級上的特殊支援。為此，系統中新增了雜湊映射類型的特殊版本，相當於「映射」$Account \dashrightarrow X$。此「映射」可能看起來等同於 $Hashmap(m, X)$，其中 $Account = \mathbf{2}^m$。然而，當分片拆分為兩個時，或兩個分片合併時，此類雜湊映射會自動拆分為兩個，或合併回來，以便僅保留屬於相應分片的那些鍵。

2.3.17. 持久儲存的支付. TON 區塊鏈的一個值得注意的功能是向智慧合約收取儲存其持久資料（即擴大區塊鏈的總狀態）的費用。它的工作原理如下：

$HashmapAux(n, X)$ 的值。

[10]邏輯上； **2.5.5** 中描述的「單元集合」表示識別所有重複單元，在序列化時將此樹轉換為有向無環圖（dag）。

每個區塊宣告兩個費率，以區塊鏈的主要貨幣（通常是 Gram）計價：在持久儲存中保留一個單元的價格，以及在持久儲存的某個單元中保留一個原始位元組的價格。每個帳戶使用的單元和位元組總數的統計資料儲存為其狀態的一部分，因此透過將這些數字乘以區塊標頭中宣告的兩個費率，我們可以計算從帳戶餘額中扣除的用於在前一個區塊和當前區塊之間保留其資料的支付。

然而，並非在每個區塊中對每個帳戶和智慧合約徵收持久儲存使用的費用；相反，最後徵收此費用的區塊的序列號儲存在帳戶資料中，當對帳戶執行任何操作時（例如，價值轉移或智慧合約接收和處理訊息），在執行任何進一步操作之前，從帳戶餘額中扣除自上次此類支付以來所有區塊的儲存使用支付。如果此後帳戶的餘額變為負數，則帳戶被銷毀。

工作鏈可能宣告每個帳戶的一些原始資料位元組數量為「免費」（即不參與持久儲存支付），以使僅保留其一種或兩種加密貨幣餘額的「簡單」帳戶免於這些持續支付。

請注意，如果沒有人向帳戶傳送任何訊息，則不會收取其持久儲存支付，它可以無限期存在。然而，任何人都可以傳送（例如）一個空訊息來銷毀這樣的帳戶。可以向此類訊息的傳送者提供一個小激勵，從要銷毀的帳戶的原始餘額的一部分中收取。然而，我們預期驗證者會免費銷毀此類無力償還的帳戶，只是為了減少全域區塊鏈狀態大小並避免在沒有補償的情況下保留大量資料。

為保留持久資料而收取的支付在分片鏈或主鏈的驗證者之間分配（在後一種情況下與其權益成比例）。

2.3.18. 本地和全域智慧合約；智慧合約實例. 智慧合約通常僅駐留在一個分片中，根據智慧合約的 $account\_id$ 選擇，類似於「普通」帳戶。這通常對大多數應用程式來說已經足夠。然而，一些「高負載」智慧合約可能希望在某個工作鏈的每個分片鏈中都有一個「實例」。為了實現這一點，它們必須將其建立交易傳播到所有分片鏈中，例如，透過將此交易提交到工作鏈 w 的「根」分片鏈 $(w, \emptyset)$[11]並支付大額佣金。[12]

此操作有效地在每個分片中建立智慧合約的實例，具有單獨的餘額。最初，建立交易中轉移的餘額只是透過給分片 (w, s) 中的實例 $2^{-|s|}$ 部分的總餘額來分配。當分片拆分為兩個子分片時，所有全域智慧合約實例的餘額都會減半；當兩個分片合併時，餘額會相加。

在某些情況下，拆分/合併全域智慧合約的實例可能涉及（延遲）執行這些智慧合約的特殊方法。預設情況下，餘額如上所述拆分和合併，並且一些特殊的「帳戶索引」雜湊映射也會自動拆分和合併（參見 **2.3.16**）。

[11]更昂貴的替代方案是在主鏈中發佈此類「全域」智慧合約。

[12]這是一種針對所有分片的「廣播」功能，因此必須相當昂貴。

2.3.19. 限制智慧合約的拆分. 全域智慧合約可以在建立時限制其拆分深度 d，以使持久儲存費用更可預測。這意味著，如果具有 $|s| \geq d$ 的分片鏈 (w, s) 拆分為兩個，則兩個新分片鏈中只有一個繼承智慧合約的實例。此分片鏈是確定性選擇的：每個全域智慧合約都有一些「$account\_id$」，本質上是其建立交易的雜湊，其實例具有相同的 $account\_id$，前 $\leq d$ 位元替換為需要落入正確分片的合適值。此 $account\_id$ 選擇在拆分後哪個分片將繼承智慧合約實例。

2.3.20. 帳戶/智慧合約狀態. 我們可以總結以上所有內容以得出結論，帳戶或智慧合約狀態由以下組成：

- 區塊鏈主要貨幣的餘額
- 區塊鏈其他貨幣的餘額
- 智慧合約程式碼（或其雜湊）
- 智慧合約持久資料（或其 Merkle 雜湊）
- 關於使用的持久儲存單元和原始位元組數量的統計資料
- 上次（實際上是主鏈區塊編號）收取智慧合約持久儲存支付的時間
- 從此帳戶轉移貨幣和傳送訊息所需的公鑰(可選；預設等於 $account\_id$ 本身)。在某些情況下，更複雜的簽名檢查程式碼可能位於此處，類似於 Bitcoin 交易輸出所做的；那麼 $account\_id$ 將等於此程式碼的雜湊。

我們還需要在帳戶狀態或某個其他帳戶索引的雜湊映射中的某處保留以下資料：

- 帳戶的輸出訊息佇列（參見 **2.4.17**）
- 最近傳遞的訊息的（雜湊）集合（參見 **2.4.23**）

並非所有這些對於每個帳戶都是真正需要的；例如，智慧合約程式碼僅對智慧合約需要，而對「簡單」帳戶則不需要。此外，雖然任何帳戶必須在主要貨幣（例如，基本工作鏈的主鏈和分片鏈的 Gram）中具有非零餘額，但它在其他貨幣中的餘額可能為零。為了避免保留未使用的資料，定義了一個和-積類型（取決於工作鏈）（在工作鏈建立期間），它使用不同的標籤位元組（例如，TL 建構子；參見 **2.2.5**）來區分使用的不同「建構子」。最終，帳戶狀態本身作為 TVM 持久儲存的單元集合保留。

2.4 分片鏈之間的訊息

TON 區塊鏈的一個重要組件是區塊鏈之間的訊息系統。這些區塊鏈可以是同一工作鏈的分片鏈，也可以是不同工作鏈的分片鏈。

2.4.1. 訊息、帳戶和交易：系統的鳥瞰圖. 訊息從一個帳戶傳送到另一個帳戶。每個交易由一個帳戶接收一個訊息、根據某些規則更改其狀態以及生成幾個（可能是一個或零個）新訊息到其他帳戶組成。每個訊息恰好被生成和接收（傳遞）一次。

這意味著訊息在系統中扮演基本角色，與帳戶（智慧合約）的角色相當。從無限分片範式的角度來看（參見 **2.1.2**），每個帳戶都駐留在其單獨的「帳戶鏈」中，它能夠影響其他帳戶狀態的唯一方式是透過傳送訊息。

2.4.2. 帳戶作為進程或 actor；Actor 模型. 人們可以將帳戶（和智慧合約）視為「進程」或「actor」，它們能夠處理傳入的訊息、更改其內部狀態並因此生成一些出站訊息。這與所謂的 *Actor* 模型密切相關，該模型用於 Erlang 等語言（但是，Erlang 中的 actor 通常稱為「進程」）。由於現有 actor 也允許作為處理入站訊息的結果建立新的 actor（即智慧合約），因此與 Actor 模型的對應關係本質上是完整的。

2.4.3. 訊息接收者. 任何訊息都有其接收者，由目標工作鏈識別符 w（預設假定與原始分片鏈相同）和接收者帳戶 $account\_id$ 表徵。$account\_id$ 的確切格式（即位元數）取決於 w；然而，分片始終由其前（最高有效）64 位元確定。

2.4.4. 訊息傳送者. 在大多數情況下，訊息有一個傳送者，再次由 $(w', account\_id')$ 對表徵。如果存在，它位於訊息接收者和訊息值之後。有時，傳送者不重要或它是區塊鏈外部的某人（即不是智慧合約），在這種情況下，此欄位不存在。

請注意，Actor 模型不要求訊息具有隱式傳送者。相反，訊息可能包含對應答請求應傳送到的 Actor 的引用；通常它與傳送者一致。然而，在加密貨幣（拜占庭）環境中，在訊息中有一個明確的不可偽造的傳送者欄位是有用的。

2.4.5. 訊息值. 訊息的另一個重要特徵是其附加的值，以來源和目標工作鏈都支援的一種或幾種加密貨幣表示。訊息的值在訊息接收者之後的最開始處指示；它本質上是 $(currency\_id, value)$ 對的列表。

請注意，「簡單」帳戶之間的「簡單」價值轉移只是附加了一些值的空（無操作）訊息。另一方面，稍微複雜一點的訊息正文可能包含簡單的文字或二進位註解（例如，關於支付的目的）。

2.4.6. 外部訊息，或「來自無處的訊息」. 一些訊息「來自無處」進入系統——也就是說，它們不是由駐留在區塊鏈中的帳戶（智慧合約或非智慧合約）生成的。最典型的例子是當使用者想要從她控制的帳戶轉移一些資金到其他帳戶時。在這種情況下，使用者向她自己的帳戶傳送一個「來自無處的訊息」，請求它生成一個訊息到接收帳戶，攜帶指定的值。如果此訊息被正確簽名，她的帳戶接收它並生成所需的出站訊息。

事實上，人們可以將「簡單」帳戶視為具有預定義程式碼的智慧合約的特例。此智慧合約僅接收一種類型的訊息。這樣的入站訊息必須包含作為傳遞（處理）入站訊息的結果要生成的出站訊息列表，以及簽名。智慧合約檢查簽名，如果正確，則生成所需的訊息。

當然，「來自無處的訊息」和正常訊息之間存在差異，因為「來自無處的訊息」不能攜帶值，因此它們本身無法支付其「gas」（即其處理）。相反，它們在甚至被建議包含在新的分片鏈區塊中之前，使用小的 gas 限制進行試探性執行；如果執行失敗（簽名不正確），則「來自無處的訊息」被視為不正確並被丟棄。如果執行在小 gas 限制內沒有失敗，則訊息可能被包含在新的分片鏈區塊中並被完全處理，從接收者的帳戶中收取消耗的 gas（處理能力）的支付。「來自無處的訊息」還可以定義一些交易費用，這些費用從接收者的帳戶中扣除，除了 gas 支付之外，用於重新分配給驗證者。

在這個意義上，「來自無處的訊息」或「外部訊息」扮演其他區塊鏈系統（例如，Bitcoin 和 Ethereum）中使用的交易候選的角色。

2.4.7. 日誌訊息，或「通向無處的訊息」. 類似地，有時可以生成特殊訊息並路由到特定分片鏈，不是為了傳遞給其接收者，而是為了被記錄，以便接收有關該分片的更新的任何人都能輕鬆觀察到。這些記錄的訊息可能在使用者的控制台中輸出，或觸發在鏈下伺服器上執行某些腳本。在這個意義上，它們代表「區塊鏈超級電腦」的外部「輸出」，正如「來自無處的訊息」代表「區塊鏈超級電腦」的外部「輸入」一樣。

2.4.8. 與鏈下服務和外部區塊鏈的互動. 這些外部輸入和輸出訊息可用於與鏈下服務和其他（外部）區塊鏈（例如 Bitcoin 或 Ethereum）互動。人們可以在 TON 區塊鏈內建立與 Bitcoin、Ether 或 Ethereum 區塊鏈中定義的任何 ERC-20 代幣掛鉤的代幣或加密貨幣，並使用由駐留在某些第三方鏈下伺服器上的腳本生成和處理的「來自無處的訊息」和「通向無處的訊息」，來實作 TON 區塊鏈與這些外部區塊鏈之間的必要互動。

2.4.9. 訊息正文. 訊息正文只是位元組序列，其含義僅由接收工作鏈和/或智慧合約確定。對於使用 TON VM 的區塊鏈，這可能是任何 TVM 單元的序列化，透過 `Send()` 操作自動生成。這種序列化只需遞迴地將 TON VM 單元中的所有引用替換為所引用的單元即可獲得。最終，出現一個原

始位元組字串，通常在前面加上 4 位元組的「訊息類型」或「訊息建構子」，用於選擇接收智慧合約的正確方法。

另一個選項是使用 TL 序列化物件（參見 **2.2.5**）作為訊息正文。這對於不同工作鏈之間的通訊可能特別有用，其中一個或兩個工作鏈不一定使用 TON VM。

2.4.10. Gas 限制和其他工作鏈/VM 特定參數. 有時訊息需要攜帶有關 gas 限制、gas 價格、交易費用和類似值的資訊，這些資訊取決於接收工作鏈並且僅與接收工作鏈相關，但不一定與原始工作鏈相關。這些參數包含在訊息正文中或之前，有時（取決於工作鏈）帶有指示其存在的特殊 4 位元組前綴（可以由 TL-方案定義；參見 **2.2.5**）。

2.4.11. 建立訊息：智慧合約和交易. 有兩個新訊息的來源。大多數訊息是在智慧合約執行期間建立的（透過 TON VM 中的 `Send()` 操作），當呼叫某個智慧合約來處理傳入訊息時。或者，訊息可能作為「外部訊息」或「來自無處的訊息」從外部進入（參見 **2.4.6**）。[13]

2.4.12. 傳遞訊息. 當訊息到達包含其目的帳戶的分片鏈時，[14]它被「傳遞」到其目的帳戶。接下來發生的事情取決於工作鏈；從外部角度來看，重要的是這樣的訊息永遠不能從此分片鏈進一步轉發。

對於基本工作鏈的分片鏈，傳遞包括將訊息值（減去任何 gas 支付）新增到接收帳戶的餘額中，並且如果接收帳戶是智慧合約，則可能在之後呼叫接收智慧合約的訊息依賴方法。事實上，智慧合約只有一個入口點來處理所有傳入訊息，它必須透過查看訊息的前幾個位元組（例如，包含 TL 建構子的前四個位元組；參見 **2.2.5**）來區分不同類型的訊息。

2.4.13. 訊息的傳遞是一個交易. 由於訊息的傳遞會更改帳戶或智慧合約的狀態，因此它是接收分片鏈中的特殊交易，並明確註冊為交易。本質上，所有 TON 區塊鏈交易都包括將一個入站訊息傳遞到其接收帳戶（智慧合約），忽略一些次要的技術細節。

2.4.14. 同一智慧合約實例之間的訊息. 回想一下，智慧合約可能是本地的(即，像任何普通帳戶一樣駐留在一個分片鏈中)或全域的(即，在所有分片中都有實例，或至少在深度 d 以下的所有分片中都有實例；參見 **2.3.18**)。如果需要，全域智慧合約的實例可以交換特殊訊息以在彼此之間轉移資訊

[13]以上內容只需對基本工作鏈及其分片鏈字面上為真；其他工作鏈可能提供建立訊息的其他方式。

[14]作為退化情況，此分片鏈可能與原始分片鏈一致——例如，如果我們在尚未拆分的工作鏈內工作。

和價值。在這種情況下，（不可偽造的）傳送者 $account\_id$ 變得重要（參見 **2.4.4**）。

2.4.15. 向智慧合約的任何實例傳送訊息；萬用字元地址. 有時訊息（例如，客戶端請求）需要傳遞到全域智慧合約的任何實例，通常是最近的實例（如果有一個與傳送者駐留在同一分片鏈中，則它是明顯的候選者）。一種方法是使用「萬用字元接收者地址」，目的 $account\_id$ 的前 d 位元允許採用任意值。在實踐中，通常將這 d 位元設定為與傳送者的 $account\_id$ 中相同的值。

2.4.16. 輸入佇列不存在. 區塊鏈（通常是分片鏈；有時是主鏈）接收的所有訊息——或者本質上，駐留在某個分片鏈內的「帳戶鏈」接收的所有訊息——都會立即傳遞（即由接收帳戶處理）。因此，不存在「輸入佇列」。相反，如果由於對區塊總大小和 gas 使用的限制而無法處理所有目的為特定分片鏈的訊息，則一些訊息只是留在原始分片鏈的輸出佇列中累積。

2.4.17. 輸出佇列. 從無限分片範式的角度來看（參見 **2.1.2**），每個帳戶鏈（即每個帳戶）都有自己的輸出佇列，包含它已生成但尚未傳遞給其接收者的所有訊息。當然，帳戶鏈只有虛擬存在；它們被分組為分片鏈，分片鏈有一個輸出「佇列」，由屬於分片鏈的所有帳戶的輸出佇列的聯集組成。

此分片鏈輸出「佇列」僅對其成員訊息施加部分順序。也就是說，在前一個區塊中生成的訊息必須在後續區塊中生成的任何訊息之前傳遞，並且由同一帳戶生成且具有相同目的地的任何訊息必須按其生成順序傳遞。

2.4.18. 可靠且快速的鏈間訊息傳遞. 對於像 TON 這樣的可擴展多區塊鏈專案來說，能夠在不同分片鏈之間轉發和傳遞訊息（參見 **2.1.3**）至關重要，即使系統中有數百萬個分片鏈。訊息應該可靠地傳遞（即，訊息不應丟失或傳遞超過一次）且快速。TON 區塊鏈透過使用兩種「訊息路由」機制的組合來實現這一目標。

2.4.19. 超立方體路由：保證傳遞訊息的「慢路徑」. TON 區塊鏈使用「超立方體路由」作為一種緩慢但安全可靠的方式，將訊息從一個分片鏈傳遞到另一個分片鏈，如有必要，使用幾個中間分片鏈進行中轉。否則，任何給定分片鏈的驗證者將需要追蹤所有其他分片鏈的狀態（輸出佇列），隨著分片鏈總數的增長，這將需要令人望而卻步的計算能力和網路頻寬，從而限制系統的可擴展性。因此，不可能直接從任何分片將訊息傳遞到每個其他分片。相反，每個分片僅「連接」到其 (w, s) 分片識別符（參見 **2.1.8**）中恰好一個十六進位數字不同的分片。透過這種方式，所有分片鏈構成一個「超立方體」圖，訊息沿著此超立方體的邊傳播。

如果訊息被傳送到與當前分片不同的分片，則當前分片識別符的十六進位數字之一（確定性選擇）被目標分片的相應數字替換，所得識別符用作轉發訊息的近似目標。[15]

超立方體路由的主要優點是，區塊有效性條件意味著建立分片鏈區塊的驗證者必須從「鄰近」分片鏈的輸出佇列收集和處理訊息，否則將失去其權益。透過這種方式，任何訊息都可以預期遲早到達其最終目的地；訊息不會在中轉中丟失或被傳遞兩次。

請注意，超立方體路由會引入一些額外的延遲和費用，因為需要透過幾個中間分片鏈轉發訊息。然而，這些中間分片鏈的數量增長非常緩慢，為分片鏈總數 N 的對數 $\log N$（更準確地說，$\lceil \log_{16} N \rceil - 1$）。例如，如果 $N \approx 250$，最多有一個中間跳；對於 $N \approx 4000$ 個分片鏈，最多兩個。有四個中間跳，我們可以支援多達一百萬個分片鏈。我們認為這是為系統本質上無限的可擴展性付出的非常小的代價。事實上，甚至不需要付出這個代價：

2.4.20. 即時超立方體路由：訊息的「快路徑」. TON 區塊鏈的一個新穎功能是它引入了一個「快路徑」，用於將訊息從一個分片鏈轉發到任何其他分片鏈，在大多數情況下允許完全繞過 **2.4.19** 的「慢」超立方體路由，並將訊息傳遞到最終目的分片鏈的下一個區塊中。

想法如下。在「慢」超立方體路由期間，訊息沿著超立方體的邊在（網路中）傳播，但在每個中間頂點被延遲（大約五秒），以便在繼續其旅程之前被提交到相應的分片鏈中。

為了避免不必要的延遲，可以改為沿著超立方體的邊與適當的 Merkle 證明一起轉發訊息，而無需等待將其提交到中間分片鏈。事實上，網路訊息應該從原始分片的「任務組」（參見 **2.6.8**）的驗證者轉發到目的分片的「任務組」的指定區塊生產者（參見 **2.6.9**）；這可能直接完成，而不沿著超立方體的邊前進。當帶有 Merkle 證明的訊息到達目的分片鏈的驗證者（更準確地說，是整理者；參見 **2.6.5**）時，他們可以立即將其提交到新區塊中，而無需等待訊息沿著「慢路徑」完成其旅程。然後，帶有適當 Merkle 證明的傳遞確認沿著超立方體邊傳送回來，它可用於透過提交特殊交易來停止訊息沿著「慢路徑」的傳播。

請注意，這種「即時傳遞」機制並不取代 **2.4.19** 中描述的「慢」但防故障的機制。「慢路徑」仍然需要，因為驗證者不能因丟失或簡單地決定不將「快路徑」訊息提交到其區塊鏈的新區塊中而受到懲罰。[16]

[15]這不一定是用於計算超立方體路由下一跳的演算法的最終版本。特別是，十六進位數字可能被 r 位元組替換，其中 r 是可設定參數，不一定等於四。

[16]然而，驗證者有一些激勵盡快這樣做，因為他們將能夠收取與訊息相關的所有尚未在慢路徑上消耗的轉發費用。

因此，兩種訊息轉發方法並行執行，並且只有在「快」機制成功的證明被提交到中間分片鏈時，「慢」機制才會中止。[17]

2.4.21. 從鄰近分片鏈的輸出佇列收集輸入訊息. 當為分片鏈提出新區塊時，鄰近（在 **2.4.19** 的路由超立方體意義上）分片鏈的一些輸出訊息作為「輸入」訊息包含在新區塊中並立即傳遞（即處理）。關於這些鄰居的輸出訊息必須以何種順序處理，有一些規則。本質上，「較舊」的訊息（來自引用較舊主鏈區塊的分片鏈區塊）必須在任何「較新」的訊息之前傳遞；對於來自同一鄰近分片鏈的訊息，必須遵守 **2.4.17** 中描述的輸出佇列的部分順序。

2.4.22. 從輸出佇列刪除訊息. 一旦輸出佇列訊息被觀察到已被鄰近分片鏈傳遞，它就會透過特殊交易明確地從輸出佇列中刪除。

2.4.23. 防止訊息的雙重傳遞. 為了防止從鄰近分片鏈的輸出佇列取出的訊息被雙重傳遞，每個分片鏈（更準確地說，其內部的每個帳戶鏈）將最近傳遞的訊息（或只是它們的雜湊）的集合保留為其狀態的一部分。當觀察到已傳遞的訊息被其原始鄰近分片鏈從輸出佇列中刪除（參見 **2.4.22**）時，它也從最近傳遞的訊息集合中刪除。

2.4.24. 轉發目的為其他分片鏈的訊息. 超立方體路由（參見 **2.4.19**）意味著有時出站訊息不是傳遞到包含預期接收者的分片鏈，而是傳遞到位於通往目的地的超立方體路徑上的鄰近分片鏈。在這種情況下，「傳遞」包括將入站訊息移動到出站佇列。這在區塊中明確反映為特殊的轉發交易，包含訊息本身。本質上，這看起來彷彿訊息已被分片鏈內的某人接收，並因此生成了一個相同的訊息。

2.4.25. 轉發和保留訊息的支付. 轉發交易實際上花費一些 gas（取決於正在轉發的訊息的大小），因此代表此分片鏈的驗證者從正在轉發的訊息的值中扣除 gas 支付。這種轉發支付通常遠小於訊息最終傳遞給其接收者時收取的 gas 支付，即使訊息由於超立方體路由而被轉發多次。此外，只要訊息保留在某個分片鏈的輸出佇列中，它就是分片鏈全域狀態的一部分，因此也可以透過特殊交易收取長期保留全域資料的支付。

2.4.26. 往返主鏈的訊息. 訊息可以直接從任何分片鏈傳送到主鏈，反之亦然。然而，向主鏈傳送訊息和在主鏈中處理訊息的 gas 價格相當高，因此只有在真正必要時才會使用此能力——例如，驗證者存入其權益。在某些

[17]事實上，人們可能暫時或永久完全禁用「即時傳遞」機制，系統將繼續工作，儘管速度較慢。

情況下，可以為傳送到主鏈的訊息定義最小存款（附加值），只有在接收方認為訊息「有效」時才會返回。

訊息不能自動透過主鏈路由。具有 $workchain\_id \neq -1$（-1 是指示主鏈的特殊 $workchain\_id$）的訊息不能傳遞到主鏈。

原則上，可以在主鏈內建立訊息轉發智慧合約，但使用它的價格將是令人望而卻步的。

2.4.27. 同一分片鏈中帳戶之間的訊息. 在某些情況下，訊息由屬於某個分片鏈的帳戶生成，目的地是同一分片鏈中的另一個帳戶。例如，這發生在尚未拆分為多個分片鏈的新工作鏈中，因為負載是可管理的。

此類訊息可能累積在分片鏈的輸出佇列中，然後在後續區塊中作為傳入訊息處理（為此目的，任何分片都被視為自己的鄰居）。然而，在大多數情況下，可以在原始區塊本身內傳遞這些訊息。

為了實現這一點，對分片鏈區塊中包含的所有交易施加部分順序，並且交易（每個由向某個帳戶傳遞訊息組成）在遵守此部分順序的情況下處理。特別是，允許交易處理相對於此部分順序的前一個交易的某個輸出訊息。

在這種情況下，訊息正文不會被複製兩次。相反，原始交易和處理交易引用訊息的共享副本。

2.5 全域分片鏈狀態。「單元集合」理念。

現在我們準備描述 TON 區塊鏈的全域狀態，或至少是基本工作鏈的分片鏈的全域狀態。

我們從「高階」或「邏輯」描述開始，它包括說全域狀態是代數類型 *ShardchainState* 的值。

2.5.1. 分片鏈狀態作為帳戶鏈狀態的集合. 根據無限分片範式(參見 **2.1.2**)，任何分片鏈只是虛擬「帳戶鏈」的（臨時）集合，每個恰好包含一個帳戶。這意味著，本質上，全域分片鏈狀態必須是雜湊映射

$$ShardchainState := (Account \dashrightarrow AccountState) \tag{23}$$

其中所有作為此雜湊映射索引出現的 $account\_id$ 必須以前綴 s 開頭，如果我們討論分片 (w, s) 的狀態（參見 **2.1.8**）。

在實踐中，我們可能想要將 *AccountState* 拆分為幾個部分（例如，將帳戶輸出訊息佇列分開保留以簡化鄰近分片鏈的檢查），並在 *ShardchainState* 內擁有幾個雜湊映射 $(Account \dashrightarrow AccountStatePart_i)$。我們還可能向 *ShardchainState* 新增少量「全域」或「積分」參數，（例如，屬於此分片的所有帳戶的總餘額，或所有輸出佇列中的訊息總數）。

然而，(23) 是分片鏈全域狀態的良好第一近似，至少從「邏輯」（「高階」）角度來看是這樣。代數類型 *AccountState* 和 *ShardchainState* 的正式描述可以藉助 TL-方案（參見 **2.2.5**）完成，將在別處提供。

2.5.2. 拆分和合併分片鏈狀態. 請注意，分片鏈狀態 (23) 的無限分片範式描述顯示了當分片拆分或合併時應如何處理此狀態。事實上，這些狀態轉換變得是雜湊映射的非常簡單的操作。

2.5.3. 帳戶鏈狀態. （虛擬）帳戶鏈狀態只是一個帳戶的狀態，由類型 *AccountState* 描述。通常它具有 **2.3.20** 中列出的所有或部分欄位，取決於使用的特定建構子。

2.5.4. 全域工作鏈狀態. 類似於 (23)，我們可以透過相同的公式定義全域工作鏈狀態，但 *account_id* 允許採用任何值，而不僅僅是屬於一個分片的值。類似於 **2.5.1** 中所做的備註在這種情況下也適用：我們可能想要將此雜湊映射拆分為幾個雜湊映射，我們可能想要新增一些「積分」參數，例如總餘額。

本質上，全域工作鏈狀態必須由與分片鏈狀態相同的類型 *ShardchainState* 給出，因為它是我們將獲得的分片鏈狀態，如果該工作鏈的所有現有分片鏈突然合併為一個。

2.5.5. 低階視角：「單元集合」. 帳戶鏈或分片鏈狀態也有一個「低階」描述，補充上面給出的「高階」描述。這個描述非常重要，因為事實證明它非常通用，為透過網路表示、儲存、序列化和傳輸幾乎所有 TON 區塊鏈使用的資料（區塊、分片鏈狀態、智慧合約儲存、Merkle 證明等）提供了通用基礎。與此同時，這種通用的「低階」描述，一旦被理解和實現，使我們能夠只專注於「高階」考慮。

回想一下，TVM 透過 *TVM* 單元或簡稱單元的樹來表示任意代數類型的值（例如，(23) 的 *ShardchainState*）（參見 **2.3.14** 和 **2.2.5**）。

任何這樣的單元都由兩個描述符位元組組成，定義某些旗標和值 $0 \le b \le 128$（原始位元組的數量）和 $0 \le c \le 4$（對其他單元的參照數量）。然後是 b 個原始位元組和 c 個單元參照。[18]

單元參照的確切格式取決於實作以及單元是位於 RAM、磁碟、網路封包、區塊中等等。一個有用的抽象模型是想像所有單元都保存在內容定址記憶體中，單元的地址等於其（SHA256）雜湊。回想一下，單元的（Merkle）

[18]可以證明，如果需要同樣經常地對儲存在單元樹中的所有資料進行 Merkle 證明，則應使用 $b + ch \approx 2(h + r)$ 的單元來最小化平均 Merkle 證明大小，其中 $h = 32$ 是雜湊大小（以位元組為單位），$r \approx 4$ 是單元參照的「位元組大小」。換句話說，一個單元應該包含兩個參照和幾個原始位元組，或者一個參照和大約 36 個原始位元組，或者根本沒有參照但有 72 個原始位元組。

雜湊正是透過將對其子單元的參照替換為它們的（遞迴計算的）雜湊並雜湊結果位元組字串來計算的。

透過這種方式，如果我們使用單元雜湊來參照單元（例如，在其他單元的描述中），系統會有所簡化，並且單元的雜湊開始與表示它的位元組字串的雜湊一致。

現在我們看到任何可由 TVM 表示的物件，包括全域分片鏈狀態，都可以表示為「單元集合」——即單元的集合以及對其中一個單元的「根」參照（例如，透過雜湊）。請注意，重複的單元從此描述中移除（「單元集合」是單元的集合，而不是單元的多重集合），因此抽象樹表示實際上可能變成有向無環圖（dag）表示。

甚至可以將此狀態保存在磁碟上的 B-樹或 B+-樹中，包含所有相關單元（可能還有一些附加資料，例如子樹高度或參照計數器），按單元雜湊索引。然而，這個想法的天真實作將導致一個智慧合約的狀態分散在磁碟檔案的遙遠部分，這是我們寧願避免的。[19]

現在我們將詳細解釋 TON 區塊鏈使用的幾乎所有物件如何表示為「單元集合」，從而證明這種方法的通用性。

2.5.6. 分片鏈區塊作為「單元集合」. 分片鏈區塊本身也可以由代數類型描述，並儲存為「單元集合」。然後，可以簡單地透過以任意順序連接表示「單元集合」中每個單元的位元組字串來獲得區塊的天真二進位表示。這種表示可以改進和最佳化，例如，透過在區塊開頭提供所有單元的偏移列表，並在可能的情況下用此列表中的 32 位元索引替換對其他單元的雜湊參照。然而，人們應該想像一個區塊本質上是一個「單元集合」，而所有其他技術細節只是次要的最佳化和實作問題。

2.5.7. 物件更新作為「單元集合」. 想像我們有一個物件的舊版本，表示為「單元集合」，並且我們想要表示同一物件的新版本，假設與前一個版本沒有太大不同。可以簡單地將新狀態表示為另一個具有自己根的「單元集合」，並從中移除舊版本中出現的所有單元。剩餘的「單元集合」本質上是物件的更新。擁有此物件舊版本和更新的每個人都可以計算新版本，只需聯合兩個單元集合，並移除舊根（減少其參照計數器，如果參照計數器變為零則解除配置該單元）。

2.5.8. 帳戶狀態的更新. 特別地，帳戶狀態的更新，或分片鏈的全域狀態的更新，或任何雜湊映射的更新，都可以使用 **2.5.7** 中描述的想法來表示。這意味著當我們接收到一個新的分片鏈區塊（它是一個「單元集合」）時，

[19] 更好的實作是將智慧合約的狀態保存為序列化字串（如果它很小），或保存在單獨的 B-樹中（如果它很大）；然後表示區塊鏈狀態的頂層結構將是一個 B-樹，其葉節點允許包含對其他 B-樹的參照。

我們不僅僅透過它本身來解釋這個「單元集合」，而是首先將它與表示分片鏈先前狀態的「單元集合」聯合。從這個意義上說，每個區塊都可能「包含」區塊鏈的整個狀態。

2.5.9. 區塊的更新. 回想一下，區塊本身就是一個「單元集合」，因此，如果需要編輯區塊，同樣可以將「區塊更新」定義為「單元集合」，在此區塊先前版本的「單元集合」存在下進行解釋。這大致上就是 **2.1.17** 中討論的「垂直區塊」背後的想法。

2.5.10. Merkle 證明作為「單元集合」. 請注意，（廣義的）Merkle 證明——例如，從 $\textsc{Hash}(x) = h$ 的已知值開始斷言 $x[i] = y$（參見 **2.3.10** 和 **2.3.15**）——也可以表示為「單元集合」。也就是說，只需提供對應於從 $x : \mathit{Hashmap}(n, X)$ 的根到其所需葉（索引為 $i : \mathbf{2}^n$，值為 $y : X$）的路徑的單元子集。對不在此路徑上的這些單元的子單元的參照將在此證明中保持「未解析」，由單元雜湊表示。還可以提供同時的 Merkle 證明，比如 $x[i] = y$ 和 $x[i'] = y'$，方法是在「單元集合」中包含從 x 的根到對應於索引 i 和 i' 的葉的兩條路徑的聯合上的單元。

2.5.11. Merkle 證明作為全節點的查詢回應. 本質上，擁有完整分片鏈（或帳戶鏈）狀態副本的全節點可以在輕節點（例如，執行 TON 區塊鏈用戶端輕量版本的網路節點）請求時提供 Merkle 證明，使接收者能夠在沒有外部幫助的情況下執行一些簡單的查詢，僅使用此 Merkle 證明中提供的單元。輕節點可以以序列化格式將其查詢傳送到全節點，並接收帶有 Merkle 證明的正確答案——或者只是 Merkle 證明，因為請求者應該能夠僅使用 Merkle 證明中包含的單元來計算答案。此 Merkle 證明將簡單地由「單元集合」組成，僅包含在執行輕節點查詢時全節點存取的屬於分片鏈狀態的那些單元。這種方法特別可用於執行智慧合約的「取得查詢」（參見 **4.3.12**）。

2.5.12. 增強更新，或帶有有效性 Merkle 證明的狀態更新. 回想一下（參見 **2.5.7**），我們可以透過「更新」來描述物件狀態從舊值 $x : X$ 到新值 $x' : X$ 的變化，該「更新」只是一個「單元集合」，包含那些位於表示新值 x' 的子樹中但不在表示舊值 x 的子樹中的單元，因為假設接收者擁有舊值 x 及其所有單元的副本。

然而，如果接收者沒有 x 的完整副本，而只知道其（Merkle）雜湊 $h = \textsc{Hash}(x)$，它將無法檢查更新的有效性（即，更新中所有「懸掛」的單元參照確實參照到 x 的樹中存在的單元）。人們希望擁有「可驗證」的更新，透過舊狀態中所有被參照單元存在性的 Merkle 證明來增強。然後，任何只知道 $h = \textsc{Hash}(x)$ 的人都能夠檢查更新的有效性，並自己計算新的 $h' = \textsc{Hash}(x')$。

因為我們的 Merkle 證明本身就是「單元集合」（參見 **2.5.10**），可以將這樣的增強更新構造為「單元集合」，包含 x 的舊根、它的一些後代以及從 x 的根到它們的路徑，以及 x' 的新根和所有不屬於 x 的後代。

2.5.13. 分片鏈區塊中的帳戶狀態更新. 特別地，分片鏈區塊中的帳戶狀態更新應如 **2.5.12** 中討論的那樣進行增強。否則，某人可能會提交包含無效狀態更新的區塊，參照到舊狀態中不存在的單元；證明這樣的區塊無效將是有問題的（挑戰者如何證明一個單元不是先前狀態的一部分？）。

現在，如果區塊中包含的所有狀態更新都被增強，它們的有效性很容易檢查，它們的無效性也很容易顯示為違反（廣義）Merkle 雜湊的遞迴定義屬性。

2.5.14. 「一切皆為單元集合」理念. 之前的考慮表明，我們需要儲存或傳輸的所有內容，無論是在 TON 區塊鏈中還是在網路中，都可以表示為「單元集合」。這是 TON 區塊鏈設計理念的重要組成部分。一旦解釋了「單元集合」方法並定義了一些「單元集合」的「低階」序列化，就可以簡單地在抽象（相依）代數資料類型的高階上定義一切（區塊格式、分片鏈和帳戶狀態等）。

「一切皆為單元集合」理念的統一效果大大簡化了看似無關的服務的實作；參見 **5.1.9** 以獲取涉及支付通道的範例。

2.5.15. TON 區塊鏈的區塊「標頭」. 通常，區塊鏈中的區塊以小標頭開始，包含前一個區塊的雜湊、其建立時間、區塊中包含的所有交易樹的 Merkle 雜湊等等。然後將區塊雜湊定義為此小區塊標頭的雜湊。因為區塊標頭最終取決於區塊中包含的所有資料，所以不能在不改變其雜湊的情況下更改區塊。

在 TON 區塊鏈的區塊使用的「單元集合」方法中，沒有指定的區塊標頭。相反，區塊雜湊被定義為區塊根單元的（Merkle）雜湊。因此，區塊的頂（根）單元可能被視為該區塊的小「標頭」。

然而，根單元可能不包含通常從這種標頭預期的所有資料。本質上，人們希望標頭包含 *Block* 資料類型中定義的某些欄位。通常，這些欄位將包含在幾個單元中，包括根單元。這些單元共同構成相關欄位值的「Merkle 證明」。可能會堅持要求區塊在其他任何單元之前，在最開始就包含這些「標頭單元」。然後，只需下載區塊序列化的前幾個位元組，就可以獲得所有「標頭單元」，並了解所有預期的欄位。

2.6 建立和驗證新區塊

TON 區塊鏈最終由分片鏈和主鏈區塊組成。必須建立、驗證這些區塊並透過網路傳播到所有相關方，系統才能順利正確地運作。

2.6.1. 驗證者. 新區塊由稱為驗證者的特殊指定節點建立和驗證。本質上，任何希望成為驗證者的節點都可以成為驗證者，前提是它可以將足夠大的質押（以 TON 幣，即 Grams；參見附錄 **A**）存入主鏈。驗證者因良好的工作而獲得一些「獎勵」，即來自所有交易（訊息）的交易費、儲存費和燃料費，這些交易被提交到新生成的區塊中，以及一些新鑄造的代幣，反映了整個社群對驗證者保持 TON 區塊鏈運作的「感謝」。這筆收入按質押比例分配給所有參與的驗證者。

然而，成為驗證者是一項高度責任。如果驗證者簽署了一個無效的區塊，它可能會受到懲罰，失去部分或全部質押，並被暫時或永久排除在驗證者集合之外。如果驗證者不參與建立區塊，它不會收到與該區塊相關的獎勵份額。如果驗證者長時間放棄建立新區塊，它可能會失去部分質押並被暫停或永久排除在驗證者集合之外。

所有這些都意味著驗證者不是「憑空」獲得金錢。事實上，它必須追蹤所有或部分分片鏈的狀態（每個驗證者負責驗證和建立某個分片鏈子集中的新區塊），執行這些分片鏈中智慧合約請求的所有計算，接收有關其他分片鏈的更新等等。這項活動需要相當大的磁碟空間、計算能力和網路頻寬。

2.6.2. 驗證者而非礦工. 回想一下，TON 區塊鏈使用權益證明方法，而不是比特幣、當前版本的以太坊和大多數其他加密貨幣採用的工作量證明方法。這意味著無法透過提供某些工作量證明（計算大量無用的雜湊）來「挖掘」新區塊並因此獲得一些新代幣。相反，必須成為驗證者並花費計算資源來儲存和處理 TON 區塊鏈請求和資料。簡而言之，必須成為驗證者才能挖掘新代幣。在這方面，驗證者是新的礦工。

然而，除了成為驗證者之外，還有一些其他方式可以賺取代幣。

2.6.3. 提名者和「礦池」. 要成為驗證者，通常需要購買和安裝幾台高效能伺服器，並為它們取得良好的網際網路連線。這不像目前挖掘比特幣所需的 ASIC 裝置那麼昂貴。然而，絕對不能在家用電腦上挖掘新的 TON 代幣，更不用說智慧型手機了。

在比特幣、以太坊和其他工作量證明加密貨幣挖礦社群中，有礦池的概念，其中許多節點的計算能力不足以單獨挖掘新區塊，它們結合努力並在之後分享獎勵。

權益證明世界中的對應概念是提名者。本質上，這是一個節點將其資金

借給驗證者以幫助增加其質押；然後驗證者將其獎勵的相應份額（或一些先前商定的分數——比如 50%）分配給提名者。

透過這種方式，提名者也可以參與「挖礦」並獲得與其願意為此目的存入的金額成正比的一些獎勵。它只收到驗證者獎勵的相應份額的一小部分，因為它只提供「資本」，但不需要購買計算能力、儲存和網路頻寬。

然而，如果驗證者因無效行為而失去其質押，提名者也會失去其質押份額。從這個意義上說，提名者分擔風險。它必須明智地選擇其提名的驗證者，否則可能會賠錢。從這個意義上說，提名者做出加權決定並用他們的資金為某些驗證者「投票」。

另一方面，這種提名或借貸系統使人們能夠在不先投資大量 Grams（TON 幣）的情況下成為驗證者。換句話說，它防止那些持有大量 Grams 的人壟斷驗證者的供應。

2.6.4. 漁夫：透過指出他人的錯誤來獲得金錢. 在不成為驗證者的情況下獲得獎勵的另一種方式是成為漁夫。本質上，任何節點都可以透過在主鏈中進行小額存款來成為漁夫。然後，它可以使用特殊的主鏈交易來發布先前由驗證者簽署和發布的某些（通常是分片鏈）區塊的（Merkle）無效性證明。如果其他驗證者同意此無效性證明，則違規的驗證者將受到懲罰（失去部分質押），漁夫將獲得一些獎勵（從違規驗證者那裡沒收的代幣的一小部分）。之後，必須按照 **2.1.17** 中概述的方式更正無效的（分片鏈）區塊。更正無效的主鏈區塊可能涉及在先前提交的主鏈區塊之上建立「垂直」區塊（參見 **2.1.17**）；無需建立主鏈的分叉。

通常，漁夫需要至少成為某些分片鏈的全節點，並透過執行至少某些智慧合約的程式碼來花費一些計算資源。雖然漁夫不需要擁有與驗證者一樣多的計算能力，但我們認為成為漁夫的自然候選者是準備處理新區塊但尚未當選為驗證者的準驗證者（例如，因為未能存入足夠大的質押）。

2.6.5. 整理者：透過向驗證者建議新區塊來獲得金錢. 在不成為驗證者的情況下獲得獎勵的另一種方式是成為整理者。這是一個節點，它準備並向驗證者建議新的分片鏈區塊候選，並用從該分片鏈和其他（通常是相鄰的）分片鏈的狀態中取得的資料以及適當的 Merkle 證明進行補充（整理）。（例如，當需要從相鄰分片鏈轉發某些訊息時，這是必要的。）然後，驗證者可以輕鬆檢查提議的區塊候選的有效性，而無需下載此或其他分片鏈的完整狀態。

因為驗證者需要提交新的（整理的）區塊候選以獲得一些（「挖礦」）獎勵，所以將獎勵的一部分支付給願意提供合適區塊候選的整理者是有意義的。透過這種方式，驗證者可以透過將監視相鄰分片鏈的狀態外包給整理者來解放自己，而無需親自監視。

然而，我們預計在系統的初始部署階段不會有單獨的指定整理者，因為所有驗證者都能夠為自己充當整理者。

2.6.6. 整理者或驗證者：透過包含使用者交易來獲得金錢. 使用者可以向某些整理者或驗證者開啟小額支付通道，並支付少量代幣以換取在分片鏈中包含他們的交易。

2.6.7. 全域驗證者集合選舉. 「全域」驗證者集合每月選舉一次（實際上，每 2^{19} 個主鏈區塊）。此集合提前一個月確定並普遍知曉。

為了成為驗證者，節點必須將一些 TON 幣（Grams）轉入主鏈，然後將它們傳送到特殊智慧合約作為其建議的質押 s。與質押一起傳送的另一個參數是 $l \geq 1$，該節點願意接受的相對於最小可能值的最大驗證負載。l 還有一個全域上限（另一個可配置參數）L，例如等於 10。

然後，全域驗證者集合由此智慧合約選舉，只需選擇最多 T 個具有最大建議質押的候選人並發布他們的身份。最初，驗證者總數為 $T = 100$；我們預計隨著負載的增加，它將增長到 1000。這是一個可配置參數（參見 **2.1.21**）。

每個驗證者的實際質押計算如下：如果前 T 個提議的質押為 $s_1 \geq s_2 \geq \cdots \geq s_T$，則第 i 個驗證者的實際質押設定為 $s'_i := \min(s_i, l_i \cdot s_T)$。透過這種方式，$s'_i/s'_T \leq l_i$，因此第 i 個驗證者獲得的負載不會超過最弱驗證者的 $l_i \leq L$ 倍（因為負載最終與質押成正比）。

然後，當選的驗證者可以撤回其質押的未使用部分 $s_i - s'_i$。不成功的驗證者候選人可以撤回其所有提議的質押。

每個驗證者發布其公開簽署金鑰，不一定等於質押來源帳戶的公鑰。[20]

驗證者的質押被凍結，直到他們被選舉的期間結束，以及再多一個月，以防出現新的爭議（即，發現這些驗證者之一簽署的無效區塊）。之後，質押被退回，連同驗證者在此期間處理的交易中鑄造的代幣和費用的份額。

2.6.8. 驗證者「任務組」的選舉. 整個全域驗證者集合（其中每個驗證者被認為存在的多重性等於其質押——否則驗證者可能會被誘使假設幾個身份並在它們之間分割其質押）僅用於驗證新的主鏈區塊。分片鏈區塊僅由驗證者的特別選定子集驗證，這些子集取自如 **2.6.7** 中所述選擇的全域驗證者集合。

這些驗證者「子集」或「任務組」，為每個分片定義，每小時輪換一次（實際上，每 2^{10} 個主鏈區塊），並且提前一小時就知道，以便每個驗證者知道它需要驗證哪些分片，並可以為此做好準備（例如，透過下載缺失的分片鏈資料）。

[20]為每次驗證者選舉生成和使用新的金鑰對是有意義的。

用於為每個分片 (w, s) 選擇驗證者任務組的演算法是確定性偽隨機的。它使用驗證者嵌入到每個主鏈區塊中的偽隨機數(透過使用閾值簽章的共識生成)來建立隨機種子，然後例如為每個驗證者計算 $\text{HASH}(\text{CODE}(w).\,\text{CODE}(s).validator\_id.rand\_$ 然後驗證者按此雜湊的值排序，並選擇前幾個，以便至少擁有總驗證者質押的 $20/T$ 並至少由 5 個驗證者組成。

此選擇可以由特殊智慧合約完成。在這種情況下，選擇演算法可以輕鬆升級，而無需透過 **2.1.21** 中提到的投票機制進行硬分叉。到目前為止提到的所有其他「常數」(例如 2^{19}、2^{10}、T、20 和 5) 也是可配置參數。

2.6.9. 每個任務組的輪換優先順序. 根據前一個主鏈區塊的雜湊和（分片鏈）區塊序列號，在分片任務組的成員上施加了一定的「優先」順序。此順序是透過生成和排序一些雜湊來確定的，如上所述。

當需要生成新的分片鏈區塊時，被選擇來建立此區塊的分片任務組驗證者通常是關於此輪換「優先」順序的第一個。如果它未能建立區塊，第二或第三個驗證者可以這樣做。本質上，它們都可以建議其區塊候選，但由具有最高優先級的驗證者建議的候選應該作為拜占庭容錯（BFT）共識協議的結果獲勝。

2.6.10. 分片鏈區塊候選的傳播. 因為分片鏈任務組成員資格提前一小時就知道，他們的成員可以使用那段時間來建立專用的「分片驗證者多播覆蓋網路」，使用 TON 網路的一般機制（參見 **3.3**）。當需要生成新的分片鏈區塊時——通常在最近的主鏈區塊傳播後一到兩秒——每個人都知道誰擁有生成下一個區塊的最高優先級（參見 **2.6.9**）。此驗證者將建立新的整理區塊候選，可以自己建立，也可以在整理者的幫助下建立（參見 **2.6.5**）。驗證者必須檢查（驗證）此區塊候選（特別是如果它是由某個整理者準備的），並用其（驗證者）私鑰簽署它。然後，區塊候選使用預先安排的多播覆蓋網路傳播到任務組的其餘部分（任務組建立其自己的私有覆蓋網路，如 **3.3** 中所述，然後使用 **3.3.15** 中描述的串流多播協議的版本來傳播區塊候選）。

執行此操作的真正 BFT 方式是使用拜占庭多播協議，例如 Honey Badger BFT [11] 中使用的協議：透過 $(N, 2N/3)$-擦除碼對區塊候選進行編碼，將結果資料的 $1/N$ 直接傳送到組的每個成員，並期望他們將其資料部分直接多播到組的所有其他成員。

然而，執行此操作的更快、更直接的方式（另參見 **3.3.15**）是將區塊候選分割為一系列簽署的 1 KB 區塊（「塊」），透過 Reed–Solomon 或噴泉碼（例如 RaptorQ 碼 [9] [14]）增強其序列，並開始將塊傳輸到「多播網格」中的鄰居（即覆蓋網路），期望他們進一步傳播這些塊。一旦驗證者獲得足夠的塊來從中重建區塊候選，它就會簽署確認收據並透過其鄰居將其傳播到整個組。然後，其鄰居停止向它傳送新塊，但可以繼續傳送這些塊的（原始）簽章，相信此節點可以透過自己應用 Reed–Solomon 或噴泉碼（擁有

所有必要的資料）來生成後續塊，將它們與簽章結合，並傳播到尚未準備好的鄰居。

如果「多播網格」（覆蓋網路）在移除所有「壞」節點後保持連線（回想一下，最多允許三分之一的節點以拜占庭方式變壞，即以任意惡意方式行事），則此演算法將盡快傳播區塊候選。

不僅指定的高優先級區塊建立者可以將其區塊候選多播到整個組。優先級第二和第三的驗證者可以開始多播他們的區塊候選，可以立即開始，也可以在未能從最高優先級驗證者接收到區塊候選後開始。然而，通常只有具有最大優先級的區塊候選將由所有（實際上，由任務組的至少三分之二）驗證者簽署並提交為新的分片鏈區塊。

2.6.11. 區塊候選的驗證. 一旦區塊候選被驗證者接收並檢查其發起驗證者的簽章，接收驗證者透過執行其中的所有交易並檢查其結果是否與聲稱的一致來檢查此區塊候選的有效性。從其他區塊鏈匯入的所有訊息必須由整理資料中的適當 Merkle 證明支援，否則區塊候選被視為無效（並且，如果將此證明提交到主鏈，已經簽署此區塊候選的驗證者可能會受到懲罰）。另一方面，如果發現區塊候選有效，接收驗證者會簽署它並將其簽章傳播到組中的其他驗證者，可以透過「網格多播網路」，也可以透過直接網路訊息。

我們想要強調的是，驗證者不需要存取此分片鏈或相鄰分片鏈的狀態即可檢查（整理的）區塊候選的有效性。[21] 這允許驗證非常快速地進行（無需磁碟存取），並減輕驗證者的計算和儲存負擔（特別是如果他們願意接受外部整理者的服務來建立區塊候選）。

2.6.12. 下一個區塊候選的選舉. 一旦區塊候選收集到任務組中至少三分之二（按質押）的驗證者的有效性簽章，它就有資格作為下一個分片鏈區塊提交。執行 BFT 協議以就選擇的區塊候選（可能提議了不止一個）達成共識，所有「好」驗證者更喜歡此輪具有最高優先級的區塊候選。由於執行此協議，區塊被至少三分之二的驗證者（按質押）的簽章增強。這些簽章不僅證明相關區塊的有效性，而且證明其被 BFT 協議選舉。之後，區塊（沒有整理資料）與這些簽章結合，以確定性方式序列化，並透過網路傳播到所有相關方。

2.6.13. 驗證者必須保留他們簽署的區塊. 在他們的任務組成員資格期間以及之後至少一個小時（或者說 2^{10} 個區塊），驗證者預計會保留他們簽署和提交的區塊。未能向其他驗證者提供已簽署的區塊可能會受到懲罰。

[21] 一個可能的例外是相鄰分片鏈的輸出佇列狀態，需要保證 **2.4.21** 中描述的訊息排序要求，因為在這種情況下 Merkle 證明的大小可能變得令人望而卻步。

2.6.14. 將新分片鏈區塊的標頭和簽章傳播到所有驗證者. 驗證者使用類似於為每個任務組建立的多播網格網路，將新生成的分片鏈區塊的標頭和簽章傳播到全域驗證者集合。

2.6.15. 新主鏈區塊的生成. 在生成所有（或幾乎所有）新分片鏈區塊之後，可以生成新的主鏈區塊。該過程本質上與分片鏈區塊相同（參見 **2.6.12**），不同之處在於所有驗證者（或至少三分之二的驗證者）必須參與此過程。因為新分片鏈區塊的標頭和簽章被傳播到所有驗證者，每個分片鏈中最新區塊的雜湊可以且必須包含在新的主鏈區塊中。一旦這些雜湊被提交到主鏈區塊中，外部觀察者和其他分片鏈可以認為新的分片鏈區塊已提交且不可變（參見 **2.1.13**）。

2.6.16. 驗證者必須保留主鏈的狀態. 主鏈和分片鏈之間的一個值得注意的區別是，所有驗證者預計會追蹤主鏈狀態，而不依賴整理資料。這很重要，因為驗證者任務組的知識是從主鏈狀態衍生的。

2.6.17. 分片鏈區塊並行生成和傳播. 通常，每個驗證者是幾個分片鏈任務組的成員；它們的數量（因此驗證者的負載）與驗證者的質押大致成正比。這意味著驗證者並行執行新分片鏈區塊生成協議的幾個實例。

2.6.18. 減輕區塊保留攻擊. 因為驗證者總集合在僅看到新分片鏈區塊的標頭和簽章後就將其雜湊插入主鏈，所以有一個很小的概率，生成此區塊的驗證者將串謀並試圖避免完整發布新區塊。這將導致相鄰分片鏈的驗證者無法建立新區塊，因為一旦新區塊的雜湊被提交到主鏈中，他們至少必須知道新區塊的輸出訊息佇列。

為了減輕這種情況，新區塊必須從其他一些驗證者（例如，相鄰分片鏈的任務組聯合的三分之二）收集簽章，證明這些驗證者確實擁有此區塊的副本，並願意在需要時將它們傳送給任何其他驗證者。只有在呈現這些簽章之後，新區塊的雜湊才能被包含在主鏈中。

2.6.19. 主鏈區塊的生成晚於分片鏈區塊. 主鏈區塊大約每五秒生成一次，分片鏈區塊也是如此。然而，雖然所有分片鏈中新區塊的生成基本上同時執行（通常由新主鏈區塊的發布觸發），但新主鏈區塊的生成被刻意延遲，以允許在主鏈中包含新生成的分片鏈區塊的雜湊。

2.6.20. 慢速驗證者可能會收到較低的獎勵. 如果驗證者「慢」，它可能無法驗證新的區塊候選，並且提交新區塊所需的三分之二的簽章可能在沒有其參與的情況下收集到。在這種情況下，它將收到與此區塊相關的較低獎勵份額。

這為驗證者提供了最佳化其硬體、軟體和網路連線的動力，以便盡可能快地處理使用者交易。

然而，如果驗證者在提交區塊之前未能簽署區塊，其簽章可能會包含在接下來的區塊之一中，然後仍會給予此驗證者一部分獎勵（根據自那時以來生成了多少區塊呈指數遞減——例如，如果驗證者遲了 k 個區塊，則為 0.9^k）。

2.6.21. 驗證者簽章的「深度」. 通常，當驗證者簽署區塊時，簽章僅證明區塊的相對有效性：此區塊是有效的，前提是此分片鏈和其他分片鏈中的所有先前區塊都是有效的。驗證者不能因為將提交到先前區塊中的無效資料視為理所當然而受到懲罰。

然而，區塊的驗證者簽章有一個稱為「深度」的整數參數。如果它不為零，則意味著驗證者也斷言指定數量的先前區塊的（相對）有效性。這是「慢」或「暫時離線」驗證者追趕並簽署一些沒有他們簽章就已提交的區塊的方式。然後，仍會給予他們區塊獎勵的一部分（參見 **2.6.20**）。

2.6.22. 驗證者對已簽署的分片鏈區塊的相對有效性負責；絕對有效性隨之而來. 我們想要再次強調，驗證者對分片鏈區塊 B 的簽章僅證明該區塊的相對有效性（或者也許還證明 d 個先前區塊的相對有效性，如果簽章具有「深度」d，參見 **2.6.21**；但這不會對以下討論產生太大影響）。換句話說，驗證者斷言分片鏈的下一個狀態 s' 是透過應用 **2.2.6** 中描述的區塊評估函數 ev_block 從先前狀態 s 獲得的：

$$s' = ev\_block(B)(s) \tag{24}$$

透過這種方式，如果原始狀態 s 被證明是「不正確的」（例如，由於先前區塊之一的無效性），簽署區塊 B 的驗證者不能受到懲罰。漁夫（參見 **2.6.4**）只有在發現相對無效的區塊時才應該投訴。整個 PoS 系統努力使每個區塊相對有效，而不是遞迴（或絕對）有效。但是，請注意，如果區塊鏈中的所有區塊都是相對有效的，那麼所有區塊和整個區塊鏈都是絕對有效的；這個陳述很容易使用對區塊鏈長度的數學歸納法來證明。透過這種方式，易於驗證的區塊相對有效性的斷言一起證明了整個區塊鏈的強得多的絕對有效性。

請注意，透過簽署區塊 B，驗證者斷言給定原始狀態 s，區塊是有效的（即，(24) 的結果不是指示無法計算下一個狀態的值 $\perp$）。透過這種方式，驗證者必須對在評估 (24) 期間存取的原始狀態的單元執行最小的形式檢查。

例如，假設預期包含從提交到區塊的交易中存取的帳戶的原始餘額的單元結果有零個原始位元組，而不是預期的 8 或 16 個。然後，根本無法從單元中檢索原始餘額，並且在嘗試處理區塊時發生「未處理的例外」。在這種情況下，驗證者不應簽署這樣的區塊，否則將受到懲罰。

2.6.23. 簽署主鏈區塊. 主鏈區塊的情況有所不同：透過簽署主鏈區塊，驗證者不僅斷言其相對有效性，而且還斷言從該驗證者承擔責任的第一個區塊開始的所有先前區塊的相對有效性（但不進一步追溯）。

2.6.24. 驗證者總數. 到目前為止所描述的系統中，要選舉的驗證者總數的上限 T（參見 **2.6.7**）不能超過，比如說，幾百或一千，因為所有驗證者預計參與 BFT 共識協議以建立每個新的主鏈區塊，並且不清楚這樣的協議是否可以擴展到數千名參與者。更重要的是，主鏈區塊必須收集所有驗證者（按質押）中至少三分之二的簽章，並且這些簽章必須包含在新區塊中（否則系統中的所有其他節點將沒有理由信任新區塊而不自己驗證它）。如果必須在每個主鏈區塊中包含超過，比如說，一千個驗證者簽章，這將意味著每個主鏈區塊中有更多資料，要由所有全節點儲存並透過網路傳播，並且花費更多處理能力來檢查這些簽章（在 PoS 系統中，全節點不需要自己驗證區塊，但他們需要檢查驗證者的簽章）。

雖然將 T 限制為一千個驗證者似乎對於 TON 區塊鏈部署的第一階段來說綽綽有餘，但必須為未來的成長做好準備，當分片鏈總數變得如此之大，以至於幾百個驗證者將不足以處理所有分片鏈。為此，我們引入一個額外的可配置參數 $T' \leq T$（最初等於 T），並且只有前 T' 個當選的驗證者（按質押）預計建立和簽署新的主鏈區塊。

2.6.25. 系統的去中心化. 人們可能會懷疑，像 TON 區塊鏈這樣的權益證明系統，依賴於 $T \approx 1000$ 個驗證者來建立所有分片鏈和主鏈區塊，必然會變得「過於中心化」，與傳統的工作量證明區塊鏈（如比特幣或以太坊）相反，在那裡每個人（原則上）都可能挖掘新區塊，對礦工總數沒有明確的上限。

然而，流行的工作量證明區塊鏈，例如比特幣和以太坊，目前需要大量的計算能力（高「雜湊率」）才能以不可忽略的成功概率挖掘新區塊。因此，新區塊的挖掘往往集中在幾個大型參與者手中，他們在裝滿專為挖礦最佳化的客製化硬體的資料中心投資了大量資金；以及幾個大型礦池手中，這些礦池集中並協調較大群體的人的努力，這些人無法自己提供足夠的「雜湊率」。

因此，截至 2017 年，超過 75% 的新以太坊或比特幣區塊是由不到十個礦工產生的。事實上，兩個最大的以太坊礦池一起產生了一半以上的所有新區塊！顯然，這樣的系統比依賴 $T \approx 1000$ 個節點來產生新區塊的系統更加中心化。

人們還可能注意到，成為 TON 區塊鏈驗證者所需的投資——即購買硬體（比如，幾台高效能伺服器）和質押（如果需要，可以輕鬆透過提名者池收集；參見 **2.6.3**）——遠低於成為成功的獨立比特幣或以太坊礦工所需的投資。事實上， **2.6.7** 的參數 L 將迫使提名者不要加入最大的「礦池」（即

積累了最大質押的驗證者），而是尋找目前接受提名者資金的較小驗證者，甚至建立新的驗證者，因為這將允許使用驗證者的——進而也是提名者的——質押的更高比例 s_i'/s_i，從而從挖礦中產生更大的獎勵。透過這種方式，TON 權益證明系統實際上鼓勵去中心化（建立和使用更多驗證者）並懲罰中心化。

2.6.26. 區塊的相對可靠性. 區塊的（相對）可靠性只是簽署此區塊的所有驗證者的總質押。換句話說，這是如果該區塊被證明無效，某些行為者將失去的金額。如果人們關注的交易轉移的價值低於區塊的可靠性，則可以認為它們足夠安全。從這個意義上說，相對可靠性是外部觀察者可以對特定區塊擁有的信任度量。

請注意，我們談論區塊的相對可靠性，因為它是一種保證，即區塊是有效的，前提是先前的區塊和所有其他被參照的分片鏈區塊都是有效的（參見 **2.6.22**）。

區塊的相對可靠性可以在提交後增加——例如，當新增遲到的驗證者簽章時（參見 **2.6.21**）。另一方面，如果這些驗證者之一因與其他一些區塊相關的不當行為而失去部分或全部質押，則區塊的相對可靠性可能會降低。

2.6.27. 「強化」區塊鏈. 重要的是為驗證者提供激勵，使其盡可能提高區塊的相對可靠性。一種方法是透過為驗證者向其他分片鏈的區塊新增簽章分配少量獎勵。甚至「準」驗證者，他們存入的質押不足以進入按質押計算的前 T 個驗證者並被包含在全域驗證者集合中（參見 **2.6.7**），可能會參與此活動（如果他們同意在選舉失敗後保持其質押凍結而不是撤回它）。這樣的準驗證者可能會兼任漁夫（參見 **2.6.4**）：如果他們無論如何都必須檢查某些區塊的有效性，他們不妨選擇報告無效區塊並收集相關獎勵。

2.6.28. 區塊的遞迴可靠性. 還可以將區塊的遞迴可靠性定義為其相對可靠性與其參照的所有區塊（即主鏈區塊、先前的分片鏈區塊以及某些相鄰分片鏈的區塊）的遞迴可靠性的最小值。換句話說，如果區塊被證明無效，無論是因為它本身無效還是因為它依賴的區塊之一無效，那麼至少會有這麼多金額被某人損失。如果真的不確定是否信任區塊中的特定交易，應該計算此區塊的遞迴可靠性，而不僅僅是相對可靠性。

在計算遞迴可靠性時追溯太遠是沒有意義的，因為如果我們往回看得太遠，我們將看到由質押已經解凍和撤回的驗證者簽署的區塊。無論如何，我們不允許驗證者自動重新考慮那麼舊的區塊（即，如果使用當前的可配置參數值，則是兩個多月前建立的區塊），並從它們開始建立分叉或在「垂直區塊鏈」的幫助下更正它們（參見 **2.1.17**），即使它們被證明是無效的。我們假設兩個月的期間為檢測和報告任何無效區塊提供了充足的機會，因此如果區塊在此期間未受到質疑，則不太可能受到質疑。

2.6.29. 權益證明對輕節點的影響. TON 區塊鏈使用的權益證明方法的一個重要後果是，TON 區塊鏈的輕節點（執行輕用戶端軟體）不需要下載所有分片鏈甚至主鏈區塊的「標頭」，即可自己檢查全節點作為其查詢答案提供的 Merkle 證明的有效性。

事實上，因為最近的分片鏈區塊雜湊包含在主鏈區塊中，所以全節點可以輕鬆地提供 Merkle 證明，證明給定的分片鏈區塊從主鏈區塊的已知雜湊開始是有效的。接下來，輕節點只需要知道主鏈的第一個區塊（其中宣布了第一組驗證者），該區塊（或至少其雜湊）可能內建在用戶端軟體中，以及之後大約每個月只有一個主鏈區塊，其中宣布了新選舉的驗證者集合，因為該區塊將由先前的驗證者集合簽署。從那時起，它可以獲得幾個最新的主鏈區塊，或者至少獲得它們的標頭和驗證者簽章，並將它們用作檢查全節點提供的 Merkle 證明的基礎。

2.7 分片鏈的拆分和合併

TON 區塊鏈最具特色和獨特的功能之一是它能夠在負載變得過高時自動將分片鏈一分為二，並在負載減少時將它們合併回來（參見 **2.1.10**）。由於其獨特性及其對整個專案可擴展性的重要性，我們必須詳細討論它。

2.7.1. 分片配置. 回想一下，在任何給定時刻，每個工作鏈 w 都被分成一個或幾個分片鏈 (w, s)（參見 **2.1.8**）。這些分片鏈可以由二元樹的葉表示，根為 $(w, \emptyset)$，每個非葉節點 (w, s) 有子節點 $(w, s.0)$ 和 $(w, s.1)$。透過這種方式，屬於工作鏈 w 的每個帳戶都被分配到恰好一個分片，並且知道當前分片鏈配置的每個人都可以確定包含帳戶 $account\_id$ 的分片 (w, s)：它是唯一的分片，其二進位字串 s 是 $account\_id$ 的前綴。

分片配置——即此分片二元樹，或給定 w 的所有活動 (w, s) 的集合（對應於分片二元樹的葉）——是主鏈狀態的一部分，並且對追蹤主鏈的每個人都可用。[22]

2.7.2. 最新的分片配置和狀態. 回想一下，最近的分片鏈區塊的雜湊包含在每個主鏈區塊中。這些雜湊組織在分片二元樹中（實際上是樹的集合，每個工作鏈一個）。透過這種方式，每個主鏈區塊都包含最新的分片配置。

2.7.3. 宣布和執行分片配置的變更. 分片配置可以透過兩種方式變更：一個分片 (w, s) 可以被拆分為兩個分片 $(w, s.0)$ 和 $(w, s.1)$，或者兩個「兄弟」分片 $(w, s.0)$ 和 $(w, s.1)$ 可以被合併為一個分片 (w, s)。

這些拆分/合併操作會提前幾個（例如 2^6；這是一個可配置參數）區塊宣布，首先在相應分片鏈區塊的「標頭」中，然後在參照這些分片鏈區塊的

[22]實際上，分片配置完全由最後一個主鏈區塊確定；這簡化了對分片配置的存取。

主鏈區塊中。所有相關方都需要這個提前宣布來為計劃的變更做準備（例如，建立覆蓋多播網路以分發新建立的分片鏈的新區塊，如 **3.3** 中所討論的）。然後，變更首先提交到分片鏈區塊的（標頭）（在拆分的情況下；對於合併，兩個分片鏈的區塊都應提交變更），然後傳播到主鏈區塊。透過這種方式，主鏈區塊不僅定義了其建立之前的最新分片配置，還定義了下一個立即的分片配置。

2.7.4. 新分片鏈的驗證者任務組. 回想一下，每個分片，即每個分片鏈，通常都被分配一個驗證者子集（驗證者任務組），專門用於在相應的分片鏈中建立和驗證新區塊（參見 **2.6.8**）。這些任務組被選舉一段時間（大約一小時），並提前一段時間（也大約一小時）就知道，並在此期間是不可變的。[23]

然而，由於拆分/合併操作，實際的分片配置可能在此期間發生變化。必須將任務組分配給新建立的分片。這是按照以下方式完成的：

請注意，任何活動分片 (w, s) 要麼是某個唯一確定的原始分片 (w, s') 的後代，這意味著 s' 是 s 的前綴，要麼它將是原始分片 (w, s') 子樹的根，其中 s 將是每個 s' 的前綴。在第一種情況下，我們只需將原始分片 (w, s') 的任務組用作新分片 (w, s) 的任務組。在後一種情況下，新分片 (w, s) 的任務組將是分片樹中 (w, s) 的所有後代原始分片 (w, s') 的任務組的聯合。

透過這種方式，每個活動分片 (w, s) 都被分配一個明確定義的驗證者子集（任務組）。當分片被拆分時，兩個子分片都從原始分片繼承整個任務組。當兩個分片合併時，它們的任務組也會合併。

任何追蹤主鏈狀態的人都可以為每個活動分片計算驗證者任務組。

2.7.5. 在原始任務組責任期間對拆分/合併操作的限制. 最終，新的分片配置將被考慮在內，並且新的專用驗證者子集（任務組）將自動分配給每個分片。在此之前，必須對拆分/合併操作施加一定的限制；否則，如果原始分片快速拆分為 2^k 個新分片，則原始任務組可能最終同時驗證 2^k 個分片鏈，對於較大的 k。

這是透過對活動分片配置與原始分片配置（用於選擇目前負責的驗證者任務組的配置）之間的距離施加限制來實現的。例如，如果 s' 是 s 的前導（即 s' 是二進位字串 s 的前綴），則可能要求從活動分片 (w, s) 到原始分片 (w, s') 的分片樹距離不得超過 3，並且如果 s' 是 s 的後繼（即 s 是 s' 的前綴），則不得超過 2。否則，不允許拆分或合併操作。

粗略地說，對於給定的驗證者任務組集合的責任期間，正在對分片可以拆分（例如三次）或合併（例如兩次）的次數施加限制。除此之外，在透過

[23]除非某些驗證者因簽署無效區塊而被暫時或永久禁止——然後他們會自動從所有任務組中排除。

合併或拆分建立分片後，它不能在一段時間（一些區塊數）內重新配置。

2.7.6. 確定拆分操作的必要性. 分片鏈的拆分操作由某些形式條件觸發（例如，如果連續 64 個區塊中分片鏈區塊至少填滿 90%）。這些條件由分片鏈任務組監視。如果滿足這些條件，首先在新分片鏈區塊的標頭中包含「拆分準備」旗標（並傳播到參照此分片鏈區塊的主鏈區塊）。然後，在幾個區塊之後，「拆分提交」旗標包含在分片鏈區塊的標頭中（並傳播到下一個主鏈區塊）。

2.7.7. 執行拆分操作. 在「拆分提交」旗標包含在分片鏈 (w, s) 的區塊 B 中之後，該分片鏈中不能有後續區塊 B'。相反，將建立分片鏈 $(w, s.0)$ 和 $(w, s.1)$ 的兩個區塊 B'_0 和 B'_1，兩者都參照區塊 B 作為其先前區塊（並且兩者都將在標頭中透過旗標指示分片剛剛被拆分）。下一個主鏈區塊將包含新分片鏈的區塊 B'_0 和 B'_1 的雜湊；不允許包含分片鏈 (w, s) 的新區塊 B' 的雜湊，因為「拆分提交」事件已經提交到先前的主鏈區塊中。

請注意，兩個新分片鏈將由與舊分片鏈相同的驗證者任務組驗證，因此它們將自動擁有其狀態的副本。從無限分片範式的角度來看，狀態拆分操作本身非常簡單（參見 **2.5.2**）。

2.7.8. 確定合併操作的必要性. 分片合併操作的必要性也由某些形式條件檢測（例如，如果連續 64 個區塊中兄弟分片鏈的兩個區塊的大小總和不超過最大區塊大小的 60%）。這些形式條件還應考慮這些區塊花費的總燃料，並將其與當前區塊燃料限制進行比較，否則區塊可能恰好很小，因為存在一些計算密集型交易阻止包含更多交易。

這些條件由兩個兄弟分片 $(w, s.0)$ 和 $(w, s.1)$ 的驗證者任務組監視。請注意，關於超立方體路由（參見 **2.4.19**），兄弟必然是鄰居，因此任何分片的任務組的驗證者無論如何都會在某種程度上監視兄弟分片。

當滿足這些條件時，驗證者子組中的任何一個都可以透過傳送特殊訊息向另一個建議他們合併。然後，它們結合成一個臨時的「合併任務組」，具有組合成員資格，能夠執行 BFT 共識演算法並在必要時傳播區塊更新和區塊候選。

如果他們就合併的必要性和準備達成共識，「合併準備」旗標被提交到每個分片鏈的某些區塊的標頭中，以及至少三分之二的兄弟任務組驗證者的簽章（並傳播到下一個主鏈區塊，以便每個人都可以為即將到來的重新配置做好準備）。然而，他們繼續為一些預定義數量的區塊建立單獨的分片鏈區塊。

2.7.9. 執行合併操作. 之後，當來自兩個原始任務組聯合的驗證者準備好成為合併分片鏈的驗證者時（這可能涉及從兄弟分片鏈進行狀態轉移和狀

態合併操作），他們在其分片鏈的區塊標頭中提交「合併提交」旗標（此事件傳播到下一個主鏈區塊），並停止在單獨的分片鏈中建立新區塊（一旦出現合併提交旗標，就禁止在單獨的分片鏈中建立區塊）。相反，建立合併的分片鏈區塊（由兩個原始任務組的聯合），在其「標頭」中參照其兩個「先前區塊」。這反映在下一個主鏈區塊中，該區塊將包含合併分片鏈的新建立區塊的雜湊。之後，合併的任務組繼續在合併的分片鏈中建立區塊。

2.8 區塊鏈專案的分類

我們將透過將 TON 區塊鏈與現有和提議的區塊鏈專案進行比較來結束我們對 TON 區塊鏈的簡短討論。然而，在此之前，我們必須引入一個足夠通用的區塊鏈專案分類。基於此分類的特定區塊鏈專案的比較推遲到 **2.9**。

2.8.1. 區塊鏈專案的分類. 作為第一步，我們為區塊鏈（即區塊鏈專案）建議一些分類標準。任何這樣的分類都有些不完整和膚淺，因為它必須忽略所考慮的專案的一些最具體和獨特的功能。然而，我們認為這是提供至少一個非常粗略和近似的區塊鏈專案領域地圖的必要第一步。

我們考慮的標準列表如下：

- 單一區塊鏈 vs. 多區塊鏈架構（參見 **2.8.2**）
- 共識演算法：權益證明 vs. 工作量證明（參見 **2.8.3**）
- 對於權益證明系統，使用的確切區塊生成、驗證和共識演算法（兩個主要選項是 DPOS vs. BFT；參見 **2.8.4**）
- 對「任意」（圖靈完備）智慧合約的支援（參見 **2.8.6**）

多區塊鏈系統有額外的分類標準（參見 **2.8.7**）：

- 成員區塊鏈的類型和規則：同質、異質（參見 **2.8.8**），混合（參見 **2.8.9**）。聯邦（參見 **2.8.10**）。
- 主鏈的缺席或存在，內部或外部（參見 **2.8.11**）
- 對分片的原生支援（參見 **2.8.12**）。靜態或動態分片（參見 **2.8.13**）。
- 成員區塊鏈之間的互動：鬆耦合和緊耦合系統（參見 **2.8.14**）

2.8.2. 單一區塊鏈 vs. 多區塊鏈專案. 第一個分類標準是系統中區塊鏈的數量。最古老和最簡單的專案由單一區塊鏈組成（簡稱「單鏈專案」）；更複雜的專案使用（或者說，計劃使用）多個區塊鏈（「多鏈專案」）。

單鏈專案通常更簡單且經過更好的測試；它們經受住了時間的考驗。它們的主要缺點是效能低，或者至少是交易吞吐量低，對於通用系統來說，交易吞吐量在十（比特幣）到不到一百[24]（以太坊）筆每秒的水平。一些專用系統（例如 Bitshares）能夠每秒處理數萬筆專用交易，代價是要求區塊鏈狀態適合記憶體，並將處理限制為預定義的特殊交易集，然後由用 C++ 等語言編寫的高度最佳化的程式碼執行（這裡沒有 VM）。

多鏈專案承諾每個人都渴望的可擴展性。它們可能支援更大的總狀態和每秒更多的交易，代價是使專案變得更加複雜，並使其實作更具挑戰性。因此，已經執行的多鏈專案很少，但大多數提議的專案都是多鏈的。我們相信未來屬於多鏈專案。

2.8.3. 建立和驗證區塊：工作量證明 vs. 權益證明. 另一個重要的區別是用於建立和傳播新區塊、檢查其有效性以及選擇幾個分叉之一（如果出現）的演算法和協議。

兩種最常見的範式是工作量證明（*PoW*）和權益證明（*PoS*）。工作量證明方法通常允許任何節點建立（「挖掘」）新區塊（並獲得與挖掘區塊相關的一些獎勵），如果它幸運地在其他競爭者設法做到這一點之前解決了一個無用的計算問題（通常涉及計算大量雜湊）。在分叉的情況下（例如，如果兩個節點發布兩個有效但不同的區塊來跟隨前一個區塊），最長的分叉獲勝。透過這種方式，區塊鏈不可變性的保證基於生成區塊鏈所花費的工作（計算資源）量：任何想要建立此區塊鏈分叉的人都需要重做此工作以建立已提交區塊的替代版本。為此，需要控制超過 50% 用於建立新區塊的總計算能力，否則替代分叉成為最長分叉的機會將呈指數級降低。

權益證明方法基於某些特殊節點（驗證者）用加密貨幣計價的大額質押，以斷言他們已經檢查（驗證）了某些區塊並發現它們是正確的。驗證者簽署區塊，並因此獲得一些小額獎勵；然而，如果驗證者被發現簽署了不正確的區塊，並提供了此證明，則其部分或全部質押將被沒收。透過這種方式，區塊鏈的有效性和不可變性保證由驗證者對區塊鏈有效性投入的質押總量給出。

權益證明方法更自然，因為它激勵驗證者（取代 PoW 礦工）執行有用的計算（檢查或建立新區塊所需的，特別是透過執行區塊中列出的所有交易），而不是計算無用的雜湊。透過這種方式，驗證者將購買更適合處理使用者交易的硬體，以獲得與這些交易相關的獎勵，從整個系統的角度來看，這似乎是一項非常有用的投資。

然而，權益證明系統的實作更具挑戰性，因為必須為許多罕見但可能的條件做準備。例如，一些惡意驗證者可能串謀破壞系統以獲取一些利潤

[24]目前更像是 15。然而，正在計劃一些升級，以使以太坊交易吞吐量增加幾倍。

(例如，透過更改他們自己的加密貨幣餘額)。這導致一些非平凡的博弈論問題。

簡而言之，權益證明更自然且更有前途，特別是對於多鏈專案（因為如果有許多區塊鏈，工作量證明將需要令人望而卻步的計算資源量)，但必須更仔細地思考和實作。大多數目前執行的區塊鏈專案，特別是最古老的專案（例如比特幣和至少原始的以太坊)，使用工作量證明。

2.8.4. 權益證明的變體。DPOS vs. BFT. 雖然工作量證明演算法彼此非常相似，主要在挖掘新區塊必須計算的雜湊函數方面有所不同，但權益證明演算法有更多可能性。它們值得有自己的子分類。

本質上，必須回答有關權益證明演算法的以下問題：

- 誰可以產生（「挖掘」）新區塊——任何全節點，還是只有（相對）小的驗證者子集的成員？(大多數 PoS 系統要求新區塊由幾個指定驗證者之一生成和簽署。)
- 驗證者是否透過其簽章保證區塊的有效性，還是期望所有全節點自己驗證所有區塊？(可擴展的 PoS 系統必須依賴驗證者簽章，而不是要求所有節點驗證所有區塊鏈的所有區塊。)
- 下一個區塊鏈區塊是否有一個提前知道的指定產生者，以便沒有其他人可以產生該區塊？
- 新建立的區塊最初是否僅由一個驗證者（其產生者）簽署，還是必須從一開始就收集大多數驗證者的簽章？

雖然根據對這些問題的回答，似乎有 2^4 個可能的 PoS 演算法類別，但實際上的區別歸結為兩種主要的 PoS 方法。事實上，大多數現代 PoS 演算法，設計用於可擴展的多鏈系統，以相同的方式回答前兩個問題：只有驗證者可以產生新區塊，他們保證區塊有效性，而不需要所有全節點自己檢查所有區塊的有效性。

至於最後兩個問題，它們的答案被證明是高度相關的，基本上只留下兩個基本選項：

- 委託權益證明（*DPOS*)：每個區塊都有一個普遍知道的指定產生者；沒有其他人可以產生該區塊；新區塊最初僅由其產生驗證者簽署。
- 拜占庭容錯（*BFT*) PoS 演算法：有一個已知的驗證者子集，其中任何一個都可以建議新區塊；在幾個建議的候選中選擇實際的下一個區塊，該區塊必須在釋放給其他節點之前由大多數驗證者驗證和簽署，這是透過拜占庭容錯共識協議的版本實現的。

2.8.5. DPOS 和 BFT PoS 的比較. BFT 方法的優點是新產生的區塊從一開始就具有大多數驗證者證明其有效性的簽章。另一個優點是，如果大多數驗證者正確執行 BFT 共識協議，根本不會出現分叉。另一方面，BFT 演算法往往相當複雜，需要更多時間讓驗證者子集達成共識。因此，區塊不能太頻繁地生成。這就是為什麼我們預計 TON 區塊鏈（從此分類的角度來看是 BFT 專案）每五秒才產生一個區塊。實際上，這個間隔可能會減少到 2–3 秒（儘管我們不承諾這一點），但如果驗證者分散在全球各地，則不會進一步減少。

DPOS 演算法的優點是相當簡單直接。它可以非常頻繁地生成新區塊——比如說，每兩秒一次，或者甚至每秒一次，[25] 因為它依賴於提前知道的指定區塊產生者。

然而，DPOS 要求所有節點——或至少所有驗證者——驗證接收到的所有區塊，因為產生和簽署新區塊的驗證者不僅確認此區塊的相對有效性，而且還確認它所參照的先前區塊的有效性，以及鏈中更早的所有區塊（可能直到當前驗證者子集的責任期開始）。當前驗證者子集上有一個預定的順序，因此對於每個區塊都有一個指定的產生者（即預期生成該區塊的驗證者）；這些指定的產生者以循環方式輪換。透過這種方式，區塊首先僅由其產生驗證者簽署；然後，當下一個區塊被挖掘時，其產生者選擇參照此區塊而不是其前身之一（否則其區塊將位於較短的鏈中，這可能會在未來失去「最長分叉」競爭），下一個區塊的簽章本質上也是對先前區塊的額外簽章。透過這種方式，新區塊逐漸收集更多驗證者的簽章——比如說，在生成接下來的二十個區塊所需的時間內收集二十個簽章。全節點要麼需要等待這二十個簽章，要麼自己驗證區塊，從充分確認的區塊（比如說，往回二十個區塊）開始，這可能不那麼容易。

DPOS 演算法的明顯缺點是，與 BFT 演算法相比，新區塊（以及提交到其中的交易）只有在挖掘了更多二十個區塊之後才能達到相同的信任水平（如 **2.6.28** 中討論的「遞迴可靠性」），而 BFT 演算法立即提供這種信任水平（比如說，二十個簽章）。另一個缺點是 DPOS 使用「最長分叉獲勝」方法來切換到其他分叉；如果至少有一些產生者在我們感興趣的區塊之後未能產生後續區塊（或者由於網路分區或複雜攻擊，我們未能觀察到這些區塊），這使得分叉相當可能。

我們認為，雖然 BFT 方法實作起來更複雜，並且需要比 DPOS 更長的區塊間隔時間，但它更適合「緊耦合」（參見 **2.8.14**）多鏈系統，因為其他區塊鏈可以在新區塊中看到已提交的交易（例如，生成針對它們的訊息）後幾乎立即開始行動，而無需等待二十次有效性確認（即接下來的二

[25] 有些人甚至聲稱 DPOS 區塊生成時間為半秒，如果驗證者分散在幾個大陸，這似乎不現實。

十個區塊），或等待接下來的六個區塊以確保不會出現分叉並自己驗證新區塊（在可擴展的多鏈系統中，驗證其他區塊鏈的區塊可能變得令人望而卻步）。因此，它們可以在保持高可靠性和可用性的同時實現可擴展性（參見 **2.8.12**）。

另一方面，DPOS 可能是「鬆耦合」多鏈系統的好選擇，其中不需要區塊鏈之間的快速互動——例如，如果每個區塊鏈（「工作鏈」）代表一個單獨的分散式交易所，並且區塊鏈間的互動僅限於罕見的代幣從一個工作鏈轉移到另一個工作鏈（或者說，以接近 1 : 1 的比率交易駐留在一個工作鏈中的一個山寨幣與另一個）。這實際上是在 BitShares 專案中所做的，該專案非常成功地使用了 DPOS。

總而言之，雖然 DPOS 可以更快地生成新區塊並將交易包含到其中（區塊之間的間隔更小），但這些交易達到在其他區塊鏈和鏈下應用中使用它們作為「已提交」和「不可變」所需的信任水平比 *BFT* 系統慢得多——比如說，在三十秒內[26] 而不是五秒。更快的交易包含並不意味著更快的交易提交。如果需要快速的區塊鏈間互動，這可能會成為一個巨大的問題。在這種情況下，必須放棄 DPOS 並選擇 BFT PoS。

2.8.6. 對交易中圖靈完備程式碼的支援，即本質上任意的智慧合約. 區塊鏈專案通常在其區塊中收集一些交易，這些交易以認為有用的方式改變區塊鏈狀態（例如，將一定數量的加密貨幣從一個帳戶轉移到另一個帳戶）。一些區塊鏈專案可能只允許某些特定的預定義類型的交易（例如從一個帳戶到另一個帳戶的價值轉移，前提是提供正確的簽章）。其他專案可能支援交易中某種有限形式的腳本。最後，一些區塊鏈支援在交易中執行任意複雜的程式碼，使系統（至少在原則上）能夠支援任意應用，前提是系統的效能允許。這通常與「圖靈完備虛擬機和腳本語言」（意味著可以用任何其他計算語言編寫的任何程式都可以重新編寫為在區塊鏈內執行）和「智慧合約」（駐留在區塊鏈中的程式）相關聯。

當然，對任意智慧合約的支援使系統真正靈活。另一方面，這種靈活性是有代價的：這些智慧合約的程式碼必須在某個虛擬機上執行，並且每次有人想要建立或驗證區塊時，必須對區塊中的每個交易都這樣做。與預定義和不可變的簡單交易類型集合的情況相比，這減慢了系統的效能，後者可以透過用 C++ 等語言實作其處理來最佳化（而不是某個虛擬機）。

最終，對圖靈完備智慧合約的支援似乎在任何通用區塊鏈專案中都是可取的；否則，區塊鏈專案的設計者必須提前決定他們的區塊鏈將用於哪些應用。事實上，比特幣區塊鏈缺乏對智慧合約的支援是必須建立一個新的區塊鏈專案以太坊的主要原因。

[26]例如，EOS 是迄今為止提出的最好的 DPOS 專案之一，承諾 45 秒的確認和區塊鏈間互動延遲（參見 [5]，「交易確認」和「鏈間通訊的延遲」部分）。

在（異質的；參見 **2.8.8**）多鏈系統中，可以透過在某些區塊鏈（即工作鏈）中支援圖靈完備的智慧合約，並在其他區塊鏈中支援一小組預定義的高度最佳化的交易，來「兩全其美」。

2.8.7. 多鏈系統的分類. 到目前為止，分類對單鏈和多鏈系統都有效。然而，多鏈系統允許更多分類標準，反映系統中不同區塊鏈之間的關係。我們現在討論這些標準。

2.8.8. 區塊鏈類型：同質和異質系統. 在多鏈系統中，所有區塊鏈可能本質上屬於同一類型並具有相同的規則（即使用相同的交易格式、用於執行智慧合約程式碼的相同虛擬機、共享相同的加密貨幣等），並且這種相似性被明確利用，但每個區塊鏈中的資料不同。在這種情況下，我們說系統是同質的。否則，不同的區塊鏈（在這種情況下通常稱為工作鏈）可以有不同的「規則」。然後我們說系統是異質的。

2.8.9. 混合異質-同質系統. 有時我們有一個混合系統，其中有幾組區塊鏈的類型或規則，但存在許多具有相同規則的區塊鏈，並且這一事實被明確利用。然後它是一個混合異質-同質系統。據我們所知，TON 區塊鏈是這種系統的唯一範例。

2.8.10. 具有幾個具有相同規則的工作鏈的異質系統，或聯邦. 在某些情況下，具有相同規則的幾個區塊鏈（工作鏈）可以存在於異質系統中，但它們之間的互動與具有不同規則的區塊鏈之間的互動相同（即，它們的相似性沒有被明確利用）。即使它們似乎使用「相同的」加密貨幣，它們實際上使用不同的「山寨幣」（加密貨幣的獨立化身）。有時甚至可以有某些機制以接近 $1:1$ 的比率轉換這些山寨幣。然而，在我們看來，這並不使系統成為同質的；它仍然是異質的。我們說，這種具有相同規則的工作鏈的異質集合是一個聯邦。

雖然建立一個允許使用相同規則建立幾個工作鏈（即聯邦）的異質系統似乎是建立可擴展系統的廉價方式，但這種方法也有很多缺點。本質上，如果有人在許多具有相同規則的工作鏈中託管一個大型專案，她不會獲得一個大型專案，而是獲得該專案的許多小實例。這就像擁有一個聊天應用（或遊戲），允許任何聊天（或遊戲）房間中最多有 50 個成員，但透過在必要時建立新房間以容納更多使用者來「擴展」。因此，許多使用者可以參與聊天或遊戲，但我們能說這樣的系統是真正可擴展的嗎？

2.8.11. 主鏈的存在，外部或內部. 有時，多鏈專案有一個區別的「主鏈」（有時稱為「控制區塊鏈」），例如用於儲存系統的整體配置（所有活動區塊鏈的集合，或者說工作鏈）、當前的驗證者集合（對於權益證明系統）

等等。有時其他區塊鏈「綁定」到主鏈，例如透過將其最新區塊的雜湊提交到其中（這也是 TON 區塊鏈所做的）。

在某些情況下，主鏈是外部的，這意味著它不是專案的一部分，而是某個其他預先存在的區塊鏈，最初與新專案的使用完全無關，並且不知道它。例如，可以嘗試使用以太坊區塊鏈作為外部專案的主鏈，並為此目的將特殊智慧合約發布到以太坊區塊鏈（例如，用於選舉和懲罰驗證者）。

2.8.12. 分片支援. 一些區塊鏈專案（或系統）對分片有原生支援，這意味著幾個（必然是同質的；參見 **2.8.8**）區塊鏈被認為是單個（從高階角度來看）虛擬區塊鏈的分片。例如，可以使用相同的規則建立 256 個分片區塊鏈（「分片鏈」），並根據其 $account\_id$ 的第一個位元組選擇的恰好一個分片中保留帳戶的狀態。

分片是擴展區塊鏈系統的自然方法，因為如果正確實作，系統中的使用者和智慧合約根本不需要意識到分片的存在。事實上，當負載變得過高時，通常希望向現有的單鏈專案（例如以太坊）新增分片。

擴展的另一種方法是使用如 **2.8.10** 中描述的異質工作鏈的「聯邦」，允許每個使用者在她選擇的一個或幾個工作鏈中保留她的帳戶，並在必要時將資金從她在一個工作鏈中的帳戶轉移到另一個工作鏈，本質上執行 $1:1$ 山寨幣交換操作。這種方法的缺點已經在 **2.8.10** 中討論過。

然而，以快速可靠的方式實作分片並不那麼容易，因為它意味著不同分片鏈之間有大量訊息。例如，如果帳戶在 N 個分片之間均勻分布，並且唯一的交易是從一個帳戶到另一個帳戶的簡單資金轉移，則只有一小部分（$1/N$）的所有交易將在單個區塊鏈內執行；幾乎所有（$1 - 1/N$）交易將涉及兩個區塊鏈，需要區塊鏈間通訊。如果我們希望這些交易快速，我們需要一個在分片鏈之間轉移訊息的快速系統。換句話說，區塊鏈專案需要在 **2.8.14** 中描述的意義上「緊耦合」。

2.8.13. 動態和靜態分片. 分片可能是動態的（如果在必要時自動建立額外的分片）或靜態的（當有預定義數量的分片時，這在最好的情況下只能透過硬分叉來更改）。大多數分片提議是靜態的；TON 區塊鏈使用動態分片（參見 **2.7**）。

2.8.14. 區塊鏈之間的互動：鬆耦合和緊耦合系統. 多區塊鏈專案可以根據組成區塊鏈之間支援的互動水平進行分類。

最低水平的支援是不同區塊鏈之間根本沒有任何互動。我們不在這裡考慮這種情況，因為我們寧願說這些區塊鏈不是一個區塊鏈系統的部分，而只是同一區塊鏈協議的單獨實例。

下一個支援水平是缺乏對區塊鏈之間訊息傳遞的任何特定支援，使互動原則上可能，但尷尬。我們稱這樣的系統為「鬆耦合」；在它們中，必須在

區塊鏈之間傳送訊息和轉移價值，就好像它們是屬於完全獨立的區塊鏈專案的區塊鏈一樣（例如，比特幣和以太坊；假設兩方想要將保存在比特幣區塊鏈中的一些比特幣交換為保存在以太坊區塊鏈中的以太幣）。換句話說，必須將出站訊息（或其生成交易）包含在源區塊鏈的區塊中。然後她（或其他某方）必須等待足夠的確認（例如，給定數量的後續區塊）以將發起交易視為「已提交」和「不可變」，以便能夠基於其存在執行外部操作。只有這樣，才能提交將訊息中繼到目標區塊鏈的交易（可能連同發起交易的參照和存在性 Merkle 證明）。

如果在轉移訊息之前等待的時間不夠長，或者由於某些其他原因仍然發生分叉，則兩個區塊鏈的聯合狀態被證明是不一致的：訊息被傳遞到第二個區塊鏈中，但從未在（最終選擇的分叉）第一個區塊鏈中生成。

有時透過標準化訊息格式和所有工作鏈區塊中輸入和輸出訊息佇列的位置來新增部分訊息支援（這在異質系統中特別有用）。雖然這在一定程度上促進了訊息傳遞，但從概念上講與前一種情況沒有太大不同，因此這樣的系統仍然是「鬆耦合」的。

相比之下，「緊耦合」系統包括特殊機制以提供所有區塊鏈之間的快速訊息傳遞。所需的行為是能夠在發起區塊鏈的區塊中生成訊息後立即將訊息傳遞到另一個工作鏈。另一方面，「緊耦合」系統也被期望在分叉的情況下保持整體一致性。雖然這兩個要求乍一看似乎是矛盾的，但我們相信 TON 區塊鏈使用的機制（將分片鏈區塊雜湊包含到主鏈區塊中；使用「垂直」區塊鏈來修正無效區塊，參見 **2.1.17**；超立方體路由，參見 **2.4.19**；即時超立方體路由，參見 **2.4.20**）使其能夠成為「緊耦合」系統，也許是迄今為止唯一的系統。

當然，建立「鬆耦合」系統要簡單得多；然而，快速高效的分片（參見 **2.8.12**）要求系統是「緊耦合」的。

2.8.15. 簡化分類。區塊鏈專案的世代. 到目前為止我們建議的分類將所有區塊鏈專案分成大量類別。然而，我們使用的分類標準在實踐中恰好是高度相關的。這使我們能夠建議一種簡化的「世代」方法來對區塊鏈專案進行分類，作為對現實的非常粗略的近似，並附有一些範例。尚未實作和部署的專案以斜體顯示；世代的最重要特徵以**粗體**顯示。

- 第一代：單鏈，**PoW**，不支援智慧合約。範例：比特幣（2009）和許多其他無趣的模仿者（萊特幣、門羅幣等）。
- 第二代：單鏈，PoW，**智慧合約支援**。範例：以太坊（2013；2015 年部署），至少在其原始形式中。
- 第三代：單鏈，**PoS**，智慧合約支援。範例：未來的以太坊（2018 年或更晚）。

- 替代第三代（3′）：**多鏈**，PoS，不支援智慧合約，鬆耦合。範例：Bitshares（2013–2014；使用 DPOS）。
- 第四代：**多鏈，PoS，智慧合約支援**，鬆耦合。範例：*EOS*（2017；使用 DPOS），*PolkaDot*（2016；使用 BFT）。
- 第五代：多鏈，使用 BFT 的 PoS，智慧合約支援，**緊耦合，帶分片**。範例：*TON*（2017）。

雖然並非所有區塊鏈專案都精確地屬於這些類別之一，但大多數專案都屬於。

2.8.16. 改變區塊鏈專案「基因組」的複雜性. 上述分類定義了區塊鏈專案的「基因組」。這個基因組相當「剛性」：一旦專案部署並被許多人使用，幾乎不可能改變它。需要一系列硬分叉（這需要獲得社群多數人的批准），即使如此，變更也需要非常保守，以保持向後相容性（例如，改變虛擬機的語義可能會破壞現有的智慧合約）。另一種方法是建立具有不同規則的新「側鏈」，並以某種方式將它們綁定到原始專案的區塊鏈（或多個區塊鏈）。可以使用現有單區塊鏈專案的區塊鏈作為本質上是新的和獨立的專案的外部主鏈。[27]

我們的結論是，一旦部署，專案的基因組就很難改變。即使從 PoW 開始並計劃將來用 PoS 替換它也是相當複雜的。[28] 向最初設計時沒有支援分片的專案新增分片似乎幾乎是不可能的。[29] 事實上，將智慧合約支援新增到最初設計時沒有支援此類功能的專案（即比特幣）中被認為是不可能的（或至少是比特幣社群大多數人不希望的），並最終導致建立了一個新的區塊鏈專案——以太坊。

2.8.17. TON 區塊鏈的基因組. 因此，如果想要建立一個可擴展的區塊鏈系統，必須從一開始就仔細選擇其基因組。如果系統旨在在未來支援一些在部署時未知的額外特定功能，則應從一開始就支援「異質」工作鏈（可能具有不同的規則）。為了使系統真正可擴展，它必須從一開始就支援分片；分片只有在系統是「緊耦合」的情況下才有意義（參見 **2.8.14**），因此這反過來意味著主鏈的存在、區塊鏈間訊息的快速系統、BFT PoS 的使用等等。

[27]例如，Plasma 專案計劃使用以太坊區塊鏈作為其（外部）主鏈；它在其他方面與以太坊的互動不多，並且它本可以由與以太坊專案無關的團隊建議和實作。

[28]截至 2017 年，以太坊仍在努力從 PoW 轉變為組合的 PoW+PoS 系統；我們希望它有一天會成為真正的 PoS 系統。

[29]以太坊的分片提議可以追溯到 2015 年；不清楚如何在不破壞以太坊或建立本質上獨立的平行專案的情況下實作和部署它們。

Project	Year	G.	Cons.	Sm.	Ch.	R.	Sh.	Int.
Bitcoin	2009	1	PoW	no	1			
Ethereum	2013, 2015	2	PoW	yes	1			
NXT	2014	2+	PoS	no	1			
Tezos	2017, ?	2+	PoS	yes	1			
Casper	2015, (2017)	3	PoW/PoS	yes	1			
BitShares	2013, 2014	3′	DPoS	no	m	ht.	no	L
EOS	2016, (2018)	4	DPoS	yes	m	ht.	no	L
PolkaDot	2016, (2019)	4	PoS BFT	yes	m	ht.	no	L
Cosmos	2017, ?	4	PoS BFT	yes	m	ht.	no	L
TON	2017, (2018)	5	PoS BFT	yes	m	mix	dyn.	T

Table 1: 一些著名區塊鏈專案的摘要。各列為：*Project* – 專案名稱；*Year* – 公布年份和部署年份；*G.* – 世代（參見 **2.8.15**）；*Cons.* – 共識演算法（參見 **2.8.3** 和 **2.8.4**）；*Sm.* – 任意程式碼支援（智慧合約；參見 **2.8.6**）；*Ch.* – 單/多區塊鏈系統（參見 **2.8.2**）；*R.* – 異質/同質多鏈系統（參見 **2.8.8**）；*Sh.* – 分片支援（參見 **2.8.12**）；*Int.* – 區塊鏈之間的互動，(L) 鬆耦合或 (T) 緊耦合（參見 **2.8.14**）。

當考慮到所有這些含義時，為 TON 區塊鏈專案做出的大多數設計選擇顯得自然，並且幾乎是唯一可能的。

2.9 與其他區塊鏈專案的比較

我們透過試圖在包含現有和提議的區塊鏈專案的地圖上為 TON 區塊鏈及其最重要和獨特的功能找到一個位置來結束我們對 TON 區塊鏈的簡短討論。我們使用 **2.8** 中描述的分類標準以統一的方式討論不同的區塊鏈專案，並構建這樣一個「區塊鏈專案地圖」。我們將此地圖表示為表 1，然後單獨簡要討論幾個專案，以指出可能不符合一般方案的特點。

2.9.1. Bitcoin [12]; `https://bitcoin.org/`. *Bitcoin*（2009）是第一個也是最著名的區塊鏈專案。它是一個典型的第一代區塊鏈專案：它是單鏈，使用工作量證明與「最長分叉獲勝」的分叉選擇演算法，並且沒有圖靈完備的腳本語言（然而，支援沒有迴圈的簡單腳本）。比特幣區塊鏈沒有帳戶的概念；它使用 UTXO（未花費交易輸出）模型。

2.9.2. Ethereum [2]; `https://ethereum.org/`. *Ethereum*（2015）是第一個支援圖靈完備智慧合約的區塊鏈。因此，它是一個典型的第二代專案，也是其中最受歡迎的。它在單一區塊鏈上使用工作量證明，但擁有智慧合約和帳戶。

2.9.3. NXT; `https://nxtplatform.org/`. *NXT*（2014）是第一個基於 PoS 的區塊鏈和貨幣。它仍然是單鏈，並且沒有智慧合約支援。

2.9.4. Tezos; `https://www.tezos.com/`. *Tezos*（2018 或更晚）是一個提議的基於 PoS 的單區塊鏈專案。我們在此提及它是因為它的獨特功能：其區塊解譯函數 *ev_block*（參見 **2.2.6**）不是固定的，而是由一個 OCaml 模組決定，該模組可以透過將新版本提交到區塊鏈（並收集一些對提議變更的投票）來升級。這樣，人們將能夠透過首先部署一個「普通」Tezos 區塊鏈，然後逐漸朝期望的方向改變區塊解譯函數來建立自訂單鏈專案，而不需要任何硬分叉。

這個想法雖然有趣，但有一個明顯的缺點，即它禁止在其他語言如 C++ 中進行任何最佳化實作，因此基於 Tezos 的區塊鏈注定會有較低的效能。我們認為，透過發布提議的區塊解譯函數 *ev_trans* 的正式規範，而不固定特定的實作，也許可以獲得類似的結果。

2.9.5. Casper.[30] *Casper* 是以太坊即將推出的 PoS 演算法；如果成功，它在 2017 年（或 2018 年）的逐步部署將使以太坊變成一個具有智慧合約支援的單鏈 PoS 或混合 PoW+PoS 系統，將以太坊轉變為第三代專案。

2.9.6. BitShares [8]; `https://bitshares.org`. *BitShares*（2014）是一個分散式區塊鏈交易平台。它是一個異質多區塊鏈 DPoS 系統，沒有智慧合約；它透過僅允許一小組預定義的特化交易類型來實現其高效能，這些交易類型可以在 C++ 中高效實作，假設區塊鏈狀態適合記憶體。它也是第一個使用委託權益證明（DPoS）的區塊鏈專案，至少在某些特化目的上展示了其可行性。

2.9.7. EOS [5]; `https://eos.io`. *EOS*（2018 或更晚）是一個提議的異質多區塊鏈 DPoS 系統，具有智慧合約支援，並對訊息傳遞有一些最小支援（仍然是 **2.8.14** 中描述的鬆耦合意義上的）。這是先前成功建立 BitShares 和 SteemIt 專案的同一團隊的嘗試，展示了 DPoS 共識演算法的優勢。可擴展性將透過為需要它的專案建立特化工作鏈（例如，分散式交易所可能使用支援一組特殊最佳化交易的工作鏈，類似於 BitShares 所做的）以及透過使用相同規則建立多個工作鏈（在 **2.8.10** 中描述的聯盟意義上）來實現。這種可擴展性方法的缺點和限制已在 *loc. cit.* 中討論。另請參閱 **2.8.5**、**2.8.12** 和 **2.8.14**，以更詳細地討論 DPoS、分片、工作鏈之間的互動及其對區塊鏈系統可擴展性的影響。

同時，即使人們無法「在區塊鏈內建立 Facebook」（參見 **2.9.13**），無論是 EOS 或其他方式，我們認為 EOS 可能成為一些高度特化、弱互動的分散式應用程式的便利平台，類似於 BitShares（去中心化交易所）和 SteemIt（去中心化部落格平台）。

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

2.9.8. PolkaDot [17]; `https://polkadot.io/`. *PolkaDot*（2019 或更晚）是最深思熟慮和最詳細的提議多鏈權益證明專案之一；其開發由以太坊聯合創始人之一領導。這個專案是我們地圖上最接近 TON 區塊鏈的專案之一。（事實上，我們對「漁夫」和「提名人」的術語感謝 PolkaDot 專案。）

PolkaDot 是一個異質鬆耦合多鏈權益證明專案，使用拜占庭容錯（BFT）共識來產生新區塊，並有一個主鏈（可能是外部的——例如，以太坊區塊鏈）。它也使用超立方體路由，有點像 **2.4.19** 中描述的 TON 的（慢速版本）。

它的獨特功能是能夠建立不僅是公開的，還有私有的區塊鏈。這些私有區塊鏈也能夠與其他公開區塊鏈互動，無論是 PolkaDot 或其他方式。

因此，PolkaDot 可能成為大型私有區塊鏈的平台，例如可能被銀行聯盟用於快速相互轉移資金，或用於大公司可能對私有區塊鏈技術的任何其他用途。

然而，PolkaDot 沒有分片支援，也不是緊耦合的。這在一定程度上阻礙了其可擴展性，類似於 EOS 的可擴展性。（也許稍好一些，因為 PolkaDot 使用 BFT PoS 而不是 DPoS。）

2.9.9. Universa; `https://universa.io`. 我們在此提及這個不尋常的區塊鏈專案的唯一原因是，它是目前唯一一個在經過時對類似於我們的無限分片範式（參見 **2.1.2**）的事物做出明確引用的專案。它的另一個特點是，它透過承諾只有該專案的受信任和授權的合作夥伴才能被接納為驗證者，從而繞過了與拜占庭容錯相關的所有複雜性，因此他們永遠不會提交無效區塊。這是一個有趣的決定；然而，它本質上使區塊鏈專案故意中心化，這是區塊鏈專案通常想要避免的（在受信任的中心化環境中工作，為什麼需要區塊鏈？）。

2.9.10. Plasma; `https://plasma.io`). *Plasma*（2019？）是以太坊另一位聯合創始人提出的一個非常規區塊鏈專案。它應該在不引入分片的情況下減輕以太坊的一些限制。本質上，它是一個與以太坊分離的專案，引入了一個（異質）工作鏈層次結構，在頂層綁定到以太坊區塊鏈（用作外部主鏈）。資金可以從層次結構中的任何區塊鏈向上轉移（從以太坊區塊鏈作為根開始），以及要完成的工作描述。然後在子工作鏈中完成必要的計算（可能需要將原始工作的部分進一步向下轉發到樹中），將結果向上傳遞，並收取獎勵。實現一致性和驗證這些工作鏈的問題透過一種（受支付通道啟發的）機制來規避，允許使用者單方面從行為不當的工作鏈將其資金提取到其父工作鏈（儘管緩慢），並將其資金和工作重新分配到另一個工作鏈。

這樣，Plasma 可能成為綁定到以太坊區塊鏈的分散式計算平台，類似於「數學協處理器」。然而，這似乎不是實現真正的通用可擴展性的方法。

2.9.11. 特化區塊鏈專案. 還有一些特化區塊鏈專案，例如 FileCoin（一個激勵使用者提供磁碟空間以儲存願意為此付費的其他使用者的檔案的系統）、Golem（一個基於區塊鏈的平台，用於租用和出借計算能力，用於特化應用程式，如 3D 渲染）或 SONM（另一個類似的計算能力出借專案）。此類專案在區塊鏈組織層面上沒有引入任何概念上的新東西；相反，它們是特定的區塊鏈應用程式，可以透過在通用區塊鏈中執行的智慧合約來實作，前提是它可以提供所需的效能。因此，此類專案可能會使用現有或計劃中的區塊鏈專案之一作為其基礎，例如 EOS、PolkaDot 或 TON。如果專案需要「真正的」可擴展性（基於分片），最好使用 TON；如果它滿足於在「聯盟」上下文中工作，透過定義自己的工作鏈家族，明確為其目的最佳化，它可能選擇 EOS 或 PolkaDot。

2.9.12. TON 區塊鏈. TON（Telegram Open Network）區塊鏈（計劃於 2018 年）是我們在本文件中描述的專案。它被設計為第一個第五代區塊鏈專案——也就是說，一個 BFT PoS 多鏈專案，混合同質/異質，支援（可分片的）自訂工作鏈，具有原生分片支援，並且緊耦合（特別是能夠在分片之間幾乎即時轉發訊息，同時保持所有分片鏈的一致狀態）。因此，它將是一個真正可擴展的通用區塊鏈專案，能夠容納本質上可以在區塊鏈中實作的任何應用程式。當由 TON 專案的其他元件增強時（參見 **1**），其可能性甚至會進一步擴展。

2.9.13. 是否可能「將 Facebook 上傳到區塊鏈」? 有時人們聲稱，可以將 Facebook 規模的社交網路實作為駐留在區塊鏈中的分散式應用程式。通常會引用一個最喜歡的區塊鏈專案作為此類應用程式的可能「主機」。

我們不能說這在技術上是不可能的。當然，人們需要一個具有真正分片的緊耦合區塊鏈專案（即 TON），以便這樣一個大型應用程式不會工作得太慢（例如，以合理的延遲將駐留在一個分片鏈中的使用者的訊息和更新傳遞給駐留在另一個分片鏈中的朋友）。然而，我們認為這是不需要的，也永遠不會完成，因為價格會是高得令人望而卻步的。

讓我們將「將 Facebook 上傳到區塊鏈」視為一個思想實驗；任何其他類似規模的專案也可以作為例子。一旦 Facebook 上傳到區塊鏈，目前由 Facebook 伺服器完成的所有操作都將被序列化為某些區塊鏈（例如 TON 的分片鏈）中的交易，並將由這些區塊鏈的所有驗證者執行。如果我們期望每個區塊至少收集二十個驗證者簽名（立即或最終，如 DPOS 系統），則每個操作必須至少執行二十次。類似地，Facebook 伺服器在其磁碟上保留的所有資料都將保留在相應分片鏈的所有驗證者的磁碟上（即至少二十份副本）。

因為驗證者本質上與目前 Facebook 使用的伺服器（或者可能是伺服器叢集，但這不影響此論證的有效性）相同，我們看到，在區塊鏈中執行

Facebook 相關的總硬體費用至少是以傳統方式實作時的二十倍。

事實上，費用會更高得多，因為區塊鏈的虛擬機比執行最佳化編譯程式碼的「裸 CPU」慢，而且其儲存並未針對 Facebook 特定問題進行最佳化。人們可能透過製作一個具有一些為 Facebook 調整的特殊交易的特定工作鏈來部分減輕這個問題；這是 BitShares 和 EOS 實現高效能的方法，在 TON 區塊鏈中也可用。然而，一般的區塊鏈設計本身仍會施加一些額外的限制，例如需要將所有操作註冊為區塊中的交易、將這些交易組織在默克爾樹中、計算和檢查它們的默克爾雜湊、進一步傳播此區塊等等。

因此，一個保守的估計是，為了驗證一個託管該規模社交網路的區塊鏈專案，需要 100 倍以上與 Facebook 目前使用的相同效能的伺服器。必須有人為這些伺服器付費，要麼是擁有分散式應用程式的公司（想像在每個 Facebook 頁面上看到 700 個廣告而不是 7 個），要麼是其使用者。無論哪種方式，這似乎在經濟上都不可行。

我們相信並非一切都應該上傳到區塊鏈。例如，沒有必要將使用者照片保存在區塊鏈中；在區塊鏈中註冊這些照片的雜湊，並將照片保存在分散式鏈下儲存（例如 FileCoin 或 TON Storage）中將是一個更好的主意。這就是為什麼 TON 不僅僅是一個區塊鏈專案，而是以 TON 區塊鏈為中心的幾個元件（TON P2P 網路、TON Storage、TON Services）的集合，如第 **1** 和 **4** 章所述。

3 TON 網路

任何區塊鏈專案不僅需要區塊格式和區塊鏈驗證規則的規範，還需要用於傳播新區塊、發送和收集交易候選等的網路協定。換句話說，每個區塊鏈專案都必須建立一個專門的對等網路。這個網路必須是對等的，因為區塊鏈專案通常期望是去中心化的，因此不能依賴於一組中心化的伺服器並使用傳統的客戶端-伺服器架構，例如，經典的線上銀行應用程式那樣。即使是輕客戶端（例如，輕量級加密貨幣錢包智慧型手機應用程式），它們必須以類似客戶端-伺服器的方式連接到完整節點，實際上也可以自由連接到另一個完整節點，如果它們先前的對等節點停機，前提是用於連接到完整節點的協定足夠標準化。

雖然單區塊鏈專案（如比特幣或以太坊）的網路需求可以很容易地滿足（基本上需要建構一個「隨機」對等覆蓋網路，並透過八卦協定傳播所有新區塊和交易候選），但多區塊鏈專案（如 TON 區塊鏈）的要求要高得多（例如，必須能夠訂閱僅某些分片鏈的更新，而不一定是所有分片鏈）。因此，TON 區塊鏈和整個 TON 專案的網路部分至少值得簡要討論。

另一方面，一旦支援 TON 區塊鏈所需的更複雜的網路協定到位，事實證明它們可以輕易地用於不一定與 TON 區塊鏈的直接需求相關的目的，從而為在 TON 生態系統中建立新服務提供了更多的可能性和靈活性。

3.1 抽象資料包網路層

建構 TON 網路協定的基石是（*TON*）抽象（資料包）網路層。它使所有節點能夠承擔某些「網路身份」，由 256 位元「抽象網路位址」表示，並僅使用這些 256 位元網路位址來識別發送者和接收者來通訊（作為第一步發送資料包給彼此）。特別是，不需要擔心 IPv4 或 IPv6 位址、UDP 埠號等；它們被抽象網路層隱藏了。

3.1.1. 抽象網路位址. 抽象網路位址，或抽象位址，或簡稱位址，是一個 256 位元整數，基本上等於一個 256 位元 ECC 公鑰。這個公鑰可以任意生成，從而建立節點喜歡的任意多個不同的網路身份。然而，為了接收（和解密）針對此類位址的訊息，必須知道相應的私鑰。

事實上，位址不是公鑰本身；相反，它是序列化的 TL 物件(參見 **2.2.5**)的 256 位元雜湊（Hash = sha256），該物件可以根據其建構子（前四個位元組）描述幾種類型的公鑰和位址。在最簡單的情況下，這個序列化的 TL 物件僅由一個 4 位元組魔術數字和一個 256 位元橢圓曲線密碼學（ECC）公鑰組成；在這種情況下，位址將等於這個 36 位元組結構的雜湊。然而，也可以使用 2048 位元 RSA 金鑰，或任何其他公鑰密碼學方案。

當節點學習另一個節點的抽象位址時，它還必須接收其「原像」（即序列化的 TL 物件，其雜湊等於該抽象位址），否則它將無法加密並發送資料包到該位址。

3.1.2. 較低層網路。UDP 實作. 從幾乎所有 TON 網路元件的角度來看，唯一存在的是一個能夠（不可靠地）從一個抽象位址向另一個抽象位址發送資料包的網路（抽象資料包網路層）。原則上，抽象資料包網路層（ADNL）可以在不同的現有網路技術上實作。然而，我們將在 IPv4/IPv6 網路（例如網際網路或內聯網）中透過 UDP 實作它，如果 UDP 不可用，則可選擇 TCP 回退。

3.1.3. ADNL over UDP 的最簡單情況. 從發送者的抽象位址向任何其他抽象位址（具有已知原像）發送資料包的最簡單情況可以如下實作。

假設發送者以某種方式知道擁有目標抽象位址的接收者的 IP 位址和 UDP 埠，並且接收者和發送者都使用從 256 位元 ECC 公鑰衍生的抽象位址。

在這種情況下，發送者只需透過其 ECC 簽名（用其私鑰完成）和其來源位址（或來源位址的原像，如果接收者尚不知道該原像）來增強要發送的資料包。結果用接收者的公鑰加密，嵌入到 UDP 資料包中並發送到接收者的已知 IP 和埠。因為 UDP 資料包的前 256 位元包含接收者的抽象位址，接收者可以識別應使用哪個私鑰來解密資料包的其餘部分。只有在那之後，發送者的身份才會被揭示。

3.1.4. 不太安全的方式，發送者的位址以明文形式. 有時，當接收者和發送者的位址以明文形式保留在 UDP 資料包中時，一種不太安全的方案就足夠了；發送者的私鑰和接收者的公鑰使用 ECDH（橢圓曲線 Diffie–Hellman）結合在一起以生成一個 256 位元共享秘密，該秘密隨後與未加密部分中也包含的隨機 256 位元隨機數一起用於衍生用於加密的 AES 金鑰。完整性可以透過將原始明文資料的雜湊連接到加密前的明文來提供。

這種方法的優點是，如果期望在兩個位址之間交換多個資料包，則可以僅計算一次共享秘密，然後快取；然後，加密或解密下一個資料包將不再需要較慢的橢圓曲線操作。

3.1.5. 通道和通道識別碼. 在最簡單的情況下，攜帶嵌入的 TON ADNL 資料包的 UDP 資料包的前 256 位元將等於接收者的位址。然而，一般來說，它們構成一個通道識別碼。有不同類型的通道。其中一些是點對點的；它們由希望將來交換大量資料的兩方建立，並透過交換幾個如 **3.1.3** 或 **3.1.4** 中所述加密的封包、執行經典或橢圓曲線 Diffie–Hellman（如果需要額外的安全性），或只是由一方生成隨機共享秘密並將其發送給另一方來生成共享秘密。

之後，從與一些額外資料（例如發送者和接收者的位址）結合的共享秘密衍生通道識別碼，例如透過雜湊，並將該識別碼用作攜帶利用該共享秘密加密的資料的 UDP 資料包的前 256 位元。

3.1.6. 通道作為隧道識別碼. 一般來說，「通道」或「通道識別碼」只是選擇處理接收者已知的入站 UDP 資料包的方式。如果通道是接收者的抽象位址，則如 **3.1.3** 或 **3.1.4** 中所述進行處理；如果通道是 **3.1.5** 中討論的已建立的點對點通道，則處理包括如 *loc. cit.* 中所解釋的那樣利用共享秘密解密資料包，依此類推。

特別是，通道識別碼實際上可以選擇一個「隧道」，當立即接收者只是將接收到的訊息轉發給其他人——實際接收者或另一個代理。可能沿途執行一些加密或解密步驟（讓人想起「洋蔥路由」[6] 甚至「大蒜路由」[31]），並且可能對重新加密的轉發封包使用另一個通道識別碼（例如，可以使用對等通道將封包轉發給路徑上的下一個接收者）。

這樣，可以在 TON 抽象資料包網路層的層級上新增對「隧道」和「代理」的一些支援——有點類似於 TOR 或 I^2P 專案提供的支援——而不影響所有更高層級 TON 網路協定的功能，這些協定對此類新增是不可知的。這個機會被 *TON* 代理服務利用（參見 **4.1.11**）。

3.1.7. 零通道和引導問題. 通常，TON ADNL 節點將有一個「鄰居表」，其中包含有關其他已知節點的資訊，例如它們的抽象位址及其原像（即公鑰）以及它們的 IP 位址和 UDP 埠。然後，它將透過使用從這些已知節點學到的資訊作為對特殊查詢的答案來逐漸擴展此表，並有時修剪過時的記錄。

然而，當 TON ADNL 節點剛啟動時，可能發生它不知道任何其他節點，只能學習節點的 IP 位址和 UDP 埠，但不知道其抽象位址的情況。例如，如果輕客戶端無法存取任何先前快取的節點和軟體中硬編碼的任何節點，並且必須要求使用者輸入要透過 DNS 解析的節點的 IP 位址或 DNS 網域，就會發生這種情況。

在這種情況下，節點將向相關節點的特殊「零通道」發送封包。這不需要知道接收者的公鑰（但訊息仍應包含發送者的身份和簽名），因此訊息在沒有加密的情況下傳輸。它通常應僅用於獲取接收者的身份（也許是為此目的專門建立的一次性身份），然後開始以更安全的方式通訊。

一旦至少知道一個節點，就很容易透過更多條目填充「鄰居表」和「路由表」，從發送到已知節點的特殊查詢的答案中學習它們。

並非所有節點都需要處理發送到零通道的資料包，但用於引導輕客戶端的節點應支援此功能。

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

3.1.8. ADNL 上的類 TCP 串流協定. ADNL 是一種基於 256 位元抽象位址的不可靠（小尺寸）資料包協定，可用作更複雜網路協定的基礎。例如，可以使用 ADNL 作為 IP 的抽象替代來建構類 TCP 串流協定。然而，TON 專案的大多數元件不需要這種串流協定。

3.1.9. RLDP，或 ADNL 上的可靠大型資料包協定. 在 ADNL 上建構的可靠任意大小資料包協定，稱為 RLDP，用於代替類 TCP 協定。這種可靠資料包協定可用於，例如，向遠端主機發送 RPC 查詢並從它們接收答案（參見 **4.1.5**）。

3.2 TON DHT：類 Kademlia 分散式雜湊表

TON 分散式雜湊表（*DHT*）在 TON 專案的網路部分中起著至關重要的作用，用於定位網路中的其他節點。例如，想要將交易提交到分片鏈的客戶端可能想要找到該分片鏈的驗證者或整理者，或至少找到可能將客戶端的交易轉發給整理者的某個節點。這可以透過在 TON DHT 中查找特殊金鑰來完成。TON DHT 的另一個重要應用是，它可用於快速填充新節點的鄰居表（參見 **3.1.7**），只需查找隨機金鑰或新節點的位址。如果節點對其入站資料包使用代理和隧道，它會在 TON DHT 中發布隧道識別碼及其入口點（例如，IP 位址和 UDP 埠）；然後所有希望向該節點發送資料包的節點將首先從 DHT 獲取此聯繫資訊。

TON DHT 是類 *Kademlia* 分散式雜湊表 [10] 家族的成員。

3.2.1. TON DHT 的金鑰. TON DHT 的金鑰只是 256 位元整數。在大多數情況下，它們被計算為 TL 序列化物件（參見 **2.2.5**）的 sha256，稱為金鑰的原像或金鑰描述。在某些情況下，TON 網路節點的抽象位址（參見 **3.1.1**）也可以用作 TON DHT 的金鑰，因為它們也是 256 位元的，而且它們也是 TL 序列化物件的雜湊。例如，如果節點不擔心發布其 IP 位址，則知道其抽象位址的任何人都可以透過簡單地在 DHT 中查找該位址作為金鑰來找到它。

3.2.2. DHT 的值. 分配給這些 256 位元金鑰的值本質上是有限長度的任意位元組字串。這些位元組字串的解釋由相應金鑰的原像決定；通常，查找金鑰的節點和儲存金鑰的節點都知道它。

3.2.3. DHT 的節點。半永久網路身份. TON DHT 的金鑰-值對應保存在 DHT 的節點上——基本上是 TON 網路的所有成員。為此，TON 網路的任何節點（也許除了一些非常輕量級的節點之外），除了 **3.1.1** 中描述的任意數量的臨時和永久抽象位址之外，至少有一個「半永久位址」，它將其識

別為 TON DHT 的成員。這個半永久或 *DHT* 位址不應該太頻繁地更改，否則其他節點將無法找到它們正在尋找的金鑰。如果節點不想揭示其「真實」身份，它會生成一個單獨的抽象位址，僅用於參與 DHT 的目的。然而，此抽象位址必須是公開的，因為它將與節點的 IP 位址和埠相關聯。

3.2.4. Kademlia 距離. 現在我們有了 256 位元金鑰和 256 位元（半永久）節點位址。我們在 256 位元序列集上引入所謂的 *XOR* 距離或 *Kademlia* 距離 d_K，由下式給出

$$d_K(x,y) := (x \oplus y) \quad \text{解釋為無號 256 位元整數} \tag{25}$$

這裡 $x \oplus y$ 表示相同長度的兩個位元序列的位元互斥 OR（XOR）。

Kademlia 距離在所有 256 位元序列的集合 $\mathbf{2}^{256}$ 上引入度量。特別是，我們有 $d_K(x,y) = 0$ 當且僅當 $x = y$，$d_K(x,y) = d_K(y,x)$，以及 $d_K(x,z) \le d_K(x,y) + d_K(y,z)$。另一個重要性質是在與 x 的任何給定距離處只有一個點：$d_K(x,y) = d_K(x,y')$ 意味著 $y = y'$。

3.2.5. 類 Kademlia DHT 和 TON DHT. 我們說一個具有 256 位元金鑰和 256 位元節點位址的分散式雜湊表（DHT）是一個類 *Kademlia DHT*，如果它期望將金鑰 K 的值保存在 s 個 Kademlia 最近的節點到 K（即，從它們的位址到 K 的 Kademlia 距離最小的 s 個節點）。

這裡 s 是一個小參數，比如 $s = 7$，需要提高 DHT 的可靠性（如果我們只在一個節點上保存金鑰，即最接近 K 的節點，那麼如果該唯一節點離線，該金鑰的值將會丟失）。

根據此定義，TON DHT 是一個類 Kademlia DHT。它是在 **3.1** 中描述的 ADNL 協定上實作的。

3.2.6. Kademlia 路由表. 參與類 Kademlia DHT 的任何節點通常維護一個 *Kademlia* 路由表。在 TON DHT 的情況下，它由 $n = 256$ 個桶組成，編號從 0 到 $n-1$。第 i 個桶將包含有關一些已知節點（固定數量 t 的「最佳」節點，也許還有一些額外的候選節點）的資訊，這些節點距離節點位址 a 的 Kademlia 距離從 2^i 到 $2^{i+1}-1$。[32]這些資訊包括它們的（半永久）位址、IP 位址和 UDP 埠，以及一些可用性資訊，例如上次 ping 的時間和延遲。

當 Kademlia 節點作為某些查詢的結果學習到任何其他 Kademlia 節點時，它首先將其作為候選節點包含到其路由表的適當桶中。然後，如果該桶中的某些「最佳」節點失敗（例如，長時間不回應 ping 查詢），它們可以被某些候選節點替換。這樣，Kademlia 路由表保持填充狀態。

[32]如果桶中有足夠多的節點，它可以進一步細分為，比如，八個子桶，這取決於 Kademlia 距離的前四個位元。這將加速 DHT 查找。

Kademlia 路由表中的新節點也包含在 **3.1.7** 中描述的 ADNL 鄰居表中。如果經常使用 Kademlia 路由表的桶中的「最佳」節點，則可以建立 **3.1.5** 中描述的意義上的通道以促進資料包的加密。

TON DHT 的一個特殊功能是，它試圖選擇具有最小往返延遲的節點作為 Kademlia 路由表桶的「最佳」節點。

3.2.7. (Kademlia 網路查詢。) Kademlia 節點通常支援以下網路查詢：

- Ping – 檢查節點可用性。
- Store$(key, value)$ – 要求節點將 $value$ 保存為金鑰 key 的值。對於 TON DHT，Store 查詢稍微複雜一些（參見 **3.2.9**）。
- Find_Node(key, l) – 要求節點返回 l 個從其 Kademlia 路由表中 Kademlia 最接近 key 的已知節點。
- Find_Value(key, l) – 與上述相同，但如果節點知道與金鑰 key 對應的值，它只返回該值。

當任何節點想要查找金鑰 K 的值時，它首先建立一個包含 s' 個節點的集合 S（對於某個小值 s'，比如 $s' = 5$），這些節點相對於所有已知節點中的 Kademlia 距離最接近 K（即，它們取自 Kademlia 路由表）。然後，向它們每個發送 Find_Value 查詢，並將它們答案中提到的節點包含在 S 中。然後，來自 S 的 s' 個最接近 K 的節點，如果之前尚未完成，也會發送 Find_Value 查詢，並且此過程繼續，直到找到值或集合 S 停止增長。這是一種關於 Kademlia 距離最接近 K 的節點的「束搜尋」。

如果要設定某個金鑰 K 的值，則對 $s' \geq s$ 執行相同的程序，使用 Find_Node 查詢而不是 Find_Value，以找到 s 個最接近 K 的節點。之後，向所有節點發送 Store 查詢。

類 Kademlia DHT 的實作中有一些不太重要的細節（例如，任何節點應該查找 s 個最接近自己的節點，比如每小時一次，並透過 Store 查詢將所有儲存的金鑰重新發布到它們）。我們暫時會忽略它們。

3.2.8. 引導 Kademlia 節點. 當 Kademlia 節點上線時，它首先透過查找自己的位址來填充其 Kademlia 路由表。在此過程中，它識別最接近自己的 s 個節點。它可以從它們那裡下載它們已知的所有 $(key, value)$ 對，以填充其 DHT 的部分。

3.2.9. 在 TON DHT 中儲存值. 在 TON DHT 中儲存值與一般的類 Kademlia DHT 略有不同。當某人希望儲存值時，她不僅必須向 Store 查詢提供金鑰 K 本身，還必須提供其原像——即，TL 序列化字串（在開頭

具有幾個預定義的 TL 建構子之一），包含金鑰的「描述」。此金鑰描述稍後與金鑰和值一起由節點保存。

金鑰描述描述正在儲存的物件的「類型」、其「擁有者」以及將來更新時的「更新規則」。擁有者通常由金鑰描述中包含的公鑰識別。如果包含它，通常只接受由相應私鑰簽署的更新。儲存物件的「類型」通常只是一個位元組字串。然而，在某些情況下，它可能更複雜——例如，輸入隧道描述（參見 **3.1.6**），或節點位址的集合。

「更新規則」也可以不同。在某些情況下，它們只是允許用新值替換舊值，前提是新值由擁有者簽署（簽名必須作為值的一部分保留，以便稍後在其他節點獲取此金鑰的值後由它們檢查）。在其他情況下，舊值以某種方式影響新值。例如，它可以包含序列號，並且僅當新序列號更大時才覆寫舊值（以防止重放攻擊）。

3.2.10. TON DHT 中的分散式「種子追蹤器」和「網路興趣群組」. 另一個有趣的情況是當值包含節點列表——也許帶有它們的 IP 位址和埠，或只是帶有它們的抽象位址——並且「更新規則」包括將請求者包含在此列表中，前提是她可以確認其身份。

這種機制可用於建立分散式「種子追蹤器」，其中對某個「種子」（即某個檔案）感興趣的所有節點都可以找到對同一種子感興趣或已經擁有副本的其他節點。

TON Storage（參見 **4.1.8**）使用此技術來找到擁有所需檔案副本的節點（例如，分片鏈狀態的快照或舊區塊）。然而，它更重要的用途是建立「覆蓋多播子網路」和「網路興趣群組」（參見 **3.3**）。想法是只有某些節點對特定分片鏈的更新感興趣。如果分片鏈的數量變得非常大，甚至找到一個對同一分片感興趣的節點也可能變得複雜。這個「分散式種子追蹤器」提供了一種方便的方法來找到其中一些節點。另一個選擇是從驗證者那裡請求它們，但這不是一種可擴展的方法，而且驗證者可能選擇不回應來自任意未知節點的此類查詢。

3.2.11. 回退金鑰. 到目前為止描述的大多數「金鑰類型」在其 TL 描述中都有一個額外的 32 位元整數欄位，通常等於零。然而，如果透過雜湊該描述獲得的金鑰無法從 TON DHT 檢索或在 TON DHT 中更新，則此欄位中的值會增加，並進行新的嘗試。這樣，人們無法透過建立大量位於受攻擊金鑰附近的抽象位址並控制相應的 DHT 節點來「捕獲」和「審查」金鑰（即，執行金鑰保留攻擊）。

3.2.12. 定位服務. 一些位於 TON 網路中並透過 **3.1** 中描述的 TON ADNL 上構建的（更高層級協定）可用的服務，可能希望在某處發布其抽象位址，以便其客戶端知道在哪裡找到它們。

然而，在 TON 區塊鏈中發布服務的抽象位址可能不是最好的方法，因為抽象位址可能需要經常更改，並且為了可靠性或負載平衡目的提供幾個位址可能是有意義的。

一個替代方案是將公鑰發布到 TON 區塊鏈中，並使用在 TL 描述字串（參見 **2.2.5**）中指示該公鑰為其「擁有者」的特殊 DHT 金鑰來發布服務抽象位址的最新列表。這是 TON Services 利用的方法之一。

3.2.13. 定位 TON 區塊鏈帳戶的擁有者. 在大多數情況下，TON 區塊鏈帳戶的擁有者不希望與抽象網路位址關聯，尤其是 IP 位址，因為這可能侵犯他們的隱私。然而，在某些情況下，TON 區塊鏈帳戶的擁有者可能希望發布一個或多個可以聯繫她的抽象位址。

一個典型的情況是 TON Payments「閃電網路」（參見 **5.2**）中的節點，即時加密貨幣轉帳平台。公開的 TON Payments 節點可能不僅希望與其他對等節點建立支付通道，而且還希望發布一個抽象網路位址，該位址可用於稍後聯繫它以沿著已建立的通道轉移支付。

一個選擇是在建立支付通道的智慧合約中包含抽象網路位址。更靈活的選擇是在智慧合約中包含公鑰，然後如 **3.2.12** 中所解釋的那樣使用 DHT。

最自然的方法是使用控制 TON 區塊鏈中帳戶的相同私鑰來簽署和發布有關與該帳戶相關聯的抽象位址的 TON DHT 中的更新。這幾乎以與 **3.2.12** 中描述的相同方式完成；然而，所使用的 DHT 金鑰將需要一個特殊的金鑰描述，僅包含 *account_id* 本身，等於包含帳戶公鑰的「帳戶描述」的 sha256。此 DHT 金鑰的值中包含的簽名也將包含帳戶描述。

這樣，就可以使用一種機制來定位 TON 區塊鏈帳戶某些擁有者的抽象網路位址。

3.2.14. 定位抽象位址. 請注意，TON DHT 雖然在 TON ADNL 上實作，但本身被 TON ADNL 用於多種目的。

其中最重要的是從其 256 位元抽象位址開始定位節點或其聯繫資料。這是必要的，因為 TON ADNL 應該能夠向任意 256 位元抽象位址發送資料包，即使沒有提供額外資訊。

為此，256 位元抽象位址只需作為金鑰在 DHT 中查找。要麼找到具有此位址的節點（即，使用此位址作為公開的半永久 DHT 位址），在這種情況下可以學習其 IP 位址和埠；或者，可以檢索輸入隧道描述作為相關金鑰的值，由正確的私鑰簽署，在這種情況下，此隧道描述將用於向預期接收者發送 ADNL 資料包。

請注意，為了使抽象位址「公開」（從網路中的任何節點可到達），其擁有者必須將其用作半永久 DHT 位址，或在 DHT 金鑰（等於所考慮的抽象位址）中發布輸入隧道描述，其中包含其另一個公開抽象位址（例如，半永

久位址）作為隧道的入口點。另一個選擇是簡單地發布其 IP 位址和 UDP 埠。

3.3 覆蓋網路和多播訊息

在像 TON 區塊鏈這樣的多區塊鏈系統中，即使是完整節點通常也只對獲取某些分片鏈的更新（即新區塊）感興趣。為此，必須在 TON 網路內部，在 **3.1** 中討論的 ADNL 協定之上，為每個分片鏈建立一個特殊的覆蓋（子）網路。

因此，需要建立任意的覆蓋子網路，對任何願意參與的節點開放。在 ADNL 之上構建的特殊八卦協定將在這些覆蓋網路中執行。特別是，這些八卦協定可用於在此類子網路內傳播（廣播）任意資料。

3.3.1. 覆蓋網路. 覆蓋（子）網路只是在某個更大網路內實作的（虛擬）網路。通常，只有更大網路的某些節點參與覆蓋子網路，並且這些節點之間的只有某些「連結」，物理的或虛擬的，是覆蓋子網路的一部分。

這樣，如果包含網路表示為圖（在資料包網路（如 ADNL）的情況下，可能是一個完整圖，其中任何節點都可以輕易地與任何其他節點通訊），則覆蓋子網路是此圖的子圖。

在大多數情況下，覆蓋網路使用在更大網路的網路協定之上構建的某些協定來實作。它可以使用與更大網路相同的位址，或使用自訂位址。

3.3.2. TON 中的覆蓋網路. TON 中的覆蓋網路建立在 **3.1** 中討論的 ADNL 協定之上；它們也使用 256 位元 ADNL 抽象位址作為覆蓋網路中的位址。每個節點通常選擇其抽象位址之一作為其在覆蓋網路中的位址。

與 ADNL 相反，TON 覆蓋網路通常不支援向任意其他節點發送資料包。相反，在某些節點（相對於所考慮的覆蓋網路稱為「鄰居」）之間建立一些「半永久連結」，並且訊息通常沿著這些連結轉發（即，從一個節點到其鄰居之一）。這樣，TON 覆蓋網路是 ADNL 網路的（完整）圖內的（通常不完整的）子圖。

TON 覆蓋網路中到鄰居的連結可以使用專用的對等 ADNL 通道來實作（參見 **3.1.5**）。

覆蓋網路的每個節點維護一個鄰居列表（相對於覆蓋網路），其中包含它們的抽象位址（它們用於在覆蓋網路中識別它們）和一些連結資料（例如，用於與它們通訊的 ADNL 通道）。

3.3.3. 私有和公開覆蓋網路. 一些覆蓋網路是公開的，意味著任何節點都可以隨意加入。其他是私有的，意味著只有某些節點可以被接納（例如，那些可以證明其身份為驗證者的節點）。一些私有覆蓋網路甚至可能對「一

般公眾」是未知的。有關此類覆蓋網路的資訊僅向某些受信任的節點提供；例如，它可以用公鑰加密，並且只有擁有相應私鑰副本的節點才能解密此資訊。

3.3.4. 中央控制的覆蓋網路. 一些覆蓋網路由一個或幾個節點中央控制，或由某個廣為人知的公鑰的擁有者控制。其他是去中心化的，意味著沒有特定節點負責它們。

3.3.5. 加入覆蓋網路. 當節點想要加入覆蓋網路時，它首先必須學習其 256 位元網路識別碼，通常等於覆蓋網路描述的 SHA256——一個 TL 序列化物件（參見 **2.2.5**），它可能包含，例如，覆蓋網路的中央權威（即其公鑰，也許還有其抽象位址，[33]）一個帶有覆蓋網路名稱的字串、如果這是與該分片相關的覆蓋網路的 TON 區塊鏈分片識別碼等等。

有時可以從網路識別碼開始恢復覆蓋網路描述，只需在 TON DHT 中查找它即可。在其他情況下（例如，對於私有覆蓋網路），必須與網路識別碼一起獲得網路描述。

3.3.6. 定位覆蓋網路的一個成員. 在節點學習了它想要加入的覆蓋網路的網路識別碼和網路描述之後，它必須定位至少一個屬於該網路的節點。

這對於不想加入覆蓋網路，而只想與其通訊的節點也是需要的；例如，可能有一個專門用於收集和傳播特定分片鏈的交易候選的覆蓋網路，並且客戶端可能想要連接到該網路的任何節點以建議交易。

用於定位覆蓋網路成員的方法在該網路的描述中定義。有時（尤其是對於私有網路），必須已經知道一個成員節點才能加入。在其他情況下，網路描述中包含某些節點的抽象位址。一個更靈活的方法是在網路描述中僅指示負責網路的中央權威，然後抽象位址將透過由該中央權威簽署的某些 DHT 金鑰的值可用。

最後，真正去中心化的公開覆蓋網路可以使用 **3.2.10** 中描述的「分散式種子追蹤器」機制，也在 TON DHT 的幫助下實作。

3.3.7. 定位覆蓋網路的更多成員。建立連結. 一旦找到覆蓋網路的一個節點，就可以向該節點發送特殊查詢，請求其他成員的列表，例如，被查詢節點的鄰居，或其隨機選擇。

這使加入成員能夠透過選擇一些新學習的網路節點並建立到它們的連結（即，如 **3.3.2** 中所述的專用 ADNL 點對點通道）來填充其相對於覆蓋網路的「鄰接」或「鄰居列表」。之後，向所有鄰居發送特殊訊息，指示新成員已準備好在覆蓋網路中工作。鄰居將它們到新成員的連結包含在其鄰居列表中。

[33]或者，抽象位址可能如 **3.2.12** 中所解釋的那樣儲存在 DHT 中。

3.3.8. 維護鄰居列表. 覆蓋網路節點必須不時更新其鄰居列表。一些鄰居，或至少到它們的連結（通道），可能停止回應；在這種情況下，這些連結必須標記為「暫停」，必須進行一些嘗試重新連接到此類鄰居，並且如果這些嘗試失敗，則必須銷毀連結。

另一方面，每個節點有時從隨機選擇的鄰居請求其鄰居列表（或其某些隨機選擇），並使用它來部分更新其自己的鄰居列表，透過向其新增一些新發現的節點，並刪除一些舊節點，要麼隨機地，要麼根據它們的回應時間和資料包丟失統計資料。

3.3.9. 覆蓋網路是一個隨機子圖. 這樣，覆蓋網路成為 ADNL 網路內的隨機子圖。如果每個頂點的度數至少為三（即，如果每個節點連接到至少三個鄰居），則已知該隨機圖幾乎以等於一的機率是連接的。更準確地說，具有 n 個頂點的隨機圖不連接的機率呈指數級小，如果，比如，$n \geq 20$，則可以完全忽略這種機率。(當然，這不適用於全域網路分區的情況，當分區不同側的節點沒有機會相互了解時。）另一方面，如果 n 小於 20，則只需要求每個頂點具有，比如，至少十個鄰居就足夠了。

3.3.10. TON 覆蓋網路針對較低延遲進行最佳化. TON 覆蓋網路如下最佳化前述方法生成的「隨機」網路圖。每個節點嘗試保留至少三個具有最小往返時間的鄰居，很少更改此「快速鄰居」列表。同時，它還有至少三個其他「慢速鄰居」，它們是完全隨機選擇的，以便覆蓋網路圖始終包含隨機子圖。這是維護連通性並防止覆蓋網路分裂成幾個未連接的區域子網路所必需的。還選擇並保留至少三個「中間鄰居」，它們具有中間往返時間，由某個常數（實際上是快速和慢速鄰居的往返時間的函數）限制。

這樣，覆蓋網路的圖仍然保持足夠的隨機性以保持連接，但針對較低的延遲和較高的吞吐量進行了最佳化。

3.3.11. 覆蓋網路中的八卦協定. 覆蓋網路通常用於執行所謂的八卦協定之一，這些協定在讓每個節點僅與其鄰居互動的同時實現某些全域目標。例如，有八卦協定可以構建（不太大的）覆蓋網路的所有成員的近似列表，或使用每個節點有限的記憶體量來計算（任意大的）覆蓋網路成員數量的估計（詳見 [15, 4.4.3] 或 [1]）。

3.3.12. 覆蓋網路作為廣播域. 在覆蓋網路中執行的最重要的八卦協定是廣播協定，旨在將網路的任何節點或可能由指定發送者節點之一生成的廣播訊息傳播到所有其他節點。

事實上，有幾個廣播協定，針對不同的使用情況進行最佳化。其中最簡單的一個接收新的廣播訊息並將它們轉發給所有尚未自己發送該訊息副本的鄰居。

3.3.13. 更複雜的廣播協定. 一些應用程式可能需要更複雜的廣播協定。例如，對於廣播相當大尺寸的訊息，向鄰居發送的不是新接收到的訊息本身，而是其雜湊（或新訊息的雜湊集合）是有意義的。鄰居在學習到以前未見過的訊息雜湊後可以請求訊息本身，比如使用 **3.1.9** 中討論的可靠大型資料包協定（RLDP）進行傳輸。這樣，新訊息將僅從一個鄰居下載。

3.3.14. 檢查覆蓋網路的連通性. 如果存在一個必須在此覆蓋網路中的已知節點（例如，覆蓋網路的「擁有者」或「建立者」），則可以檢查覆蓋網路的連通性。然後，相關節點只需不時廣播包含當前時間、序列號和其簽名的短訊息。如果不久前接收到此類廣播，則任何其他節點都可以確信它仍然連接到覆蓋網路。此協定可以擴展到幾個眾所周知的節點的情況；例如，它們都將發送此類廣播，而所有其他節點將期望從超過一半的眾所周知的節點接收廣播。

在用於傳播特定分片鏈的新區塊（或只是新區塊標頭）的覆蓋網路的情況下，節點檢查連通性的好方法是追蹤到目前為止接收到的最新區塊。因為通常每五秒鐘生成一個區塊，如果超過，比如，三十秒沒有接收到新區塊，則該節點可能已與覆蓋網路斷開連接。

3.3.15. 串流廣播協定. 最後，TON 覆蓋網路有一個串流廣播協定，例如，用於在某個分片鏈（「分片鏈任務組」）的驗證者之間傳播區塊候選，當然，他們為此目的建立私有覆蓋網路。相同的協定可用於將新的分片鏈區塊傳播到該分片鏈的所有完整節點。

此協定已在 **2.6.10** 中概述：新的（大型）廣播訊息被分割成，比如，N 個一千位元組的塊；這些塊的序列透過擦除碼（如 Reed–Solomon 或噴泉碼（例如，RaptorQ 碼 [9] [14]））增強到 $M \geq N$ 個塊，並且這些 M 個塊按塊編號升序串流到所有鄰居。參與節點收集這些塊，直到它們可以恢復原始大型訊息（為此必須成功接收至少 N 個塊），然後指示其鄰居停止發送串流的新塊，因為現在這些節點可以自己生成後續塊，因為它們擁有原始訊息的副本。此類節點繼續生成串流的後續塊並將它們發送給其鄰居，除非鄰居反過來指示這不再必要。

這樣，節點不需要在進一步傳播之前完整下載大型訊息。這最小化了廣播延遲，尤其是與 **3.3.10** 中描述的最佳化相結合時。

3.3.16. 基於現有覆蓋網路構建新的覆蓋網路. 有時不想從頭開始構建覆蓋網路。相反，一個或多個先前存在的覆蓋網路是已知的，並且新覆蓋網路的成員資格預期與這些覆蓋網路的組合成員資格顯著重疊。

當 TON 分片鏈分裂為兩個，或兩個兄弟分片鏈合併為一個時（參見 **2.7**），會出現一個重要的例子。在第一種情況下，用於將新區塊傳播到完整節點的覆蓋網路必須為每個新分片鏈構建；然而，這些新覆蓋網路中

的每一個都可以預期包含在原始分片鏈的區塊傳播網路中（並且包含大約一半的成員）。在第二種情況下，用於傳播合併分片鏈的新區塊的覆蓋網路將大致由與正在合併的兩個兄弟分片鏈相關的兩個覆蓋網路的成員的聯合組成。

在這種情況下，新覆蓋網路的描述可能包含對相關現有覆蓋網路列表的明確或隱含引用。希望加入新覆蓋網路的節點可以檢查它是否已經是這些現有網路之一的成員，並查詢其在這些網路中的鄰居是否也對新網路感興趣。在肯定回答的情況下，可以建立到此類鄰居的新點對點通道，並且可以將它們包含在新覆蓋網路的鄰居列表中。

此機制並不完全取代 **3.3.6** 和 **3.3.7** 中描述的一般機制；相反，兩者並行執行並用於填充鄰居列表。這是防止新覆蓋網路無意中分裂成幾個未連接的子網路所需的。

3.3.17. 覆蓋網路內的覆蓋網路. 在 *TON Payments*（用於即時鏈下價值轉移的「閃電網路」；參見 **5.2**）的實作中出現了另一個有趣的情況。在這種情況下，首先構建包含「閃電網路」的所有中繼節點的覆蓋網路。然而，這些節點中的一些在區塊鏈中建立了支付通道；除了 **3.3.6**、**3.3.7** 和 **3.3.8** 中描述的一般覆蓋網路演算法選擇的任何「隨機」鄰居之外，它們在此覆蓋網路中必須始終是鄰居。這些到具有已建立支付通道的鄰居的「永久連結」用於執行特定的閃電網路協定，從而有效地在包含的（幾乎總是連接的）覆蓋網路內建立覆蓋子網路（如果出現問題，則不一定連接）。

4 TON 服務和應用程式

我們已經詳細討論了 TON 區塊鏈和 TON 網路技術。現在我們解釋一些可以組合它們來建立各種服務和應用程式的方法，並討論 TON 專案本身將提供的一些服務，無論是從一開始還是稍後。

4.1 TON 服務實作策略

我們首先討論如何在 TON 生態系統內實作不同的區塊鏈和網路相關應用程式和服務。首先，需要一個簡單的分類：

4.1.1. 應用程式和服務. 我們將互換使用「應用程式」和「服務」這兩個詞。然而，存在一個微妙且有點模糊的區別：應用程式通常直接向人類使用者提供一些服務，而服務通常由其他應用程式和服務利用。例如，TON Storage 是一個服務，因為它被設計為代表其他應用程式和服務保存檔案，即使人類使用者也可能直接使用它。一個假設的「區塊鏈中的 Facebook」（參見 **2.9.13**）或 Telegram 訊息應用程式，如果透過 TON 網路提供（即，實作為「ton-service」；參見 **4.1.6**），將更像是一個應用程式，即使一些「機器人」可能在沒有人工干預的情況下自動存取它。

4.1.2. 應用程式的位置：鏈上、鏈下或混合. 為 TON 生態系統設計的服務或應用程式需要在某處保存其資料並處理該資料。這導致以下應用程式（和服務）的分類：

- 鏈上應用程式（參見 **4.1.4**）：所有資料和處理都在 TON 區塊鏈中。
- 鏈下應用程式（參見 **4.1.5**）：所有資料和處理都在 TON 區塊鏈之外，在透過 TON 網路可用的伺服器上。
- 混合應用程式（參見 **4.1.7**）：一些（但不是全部）資料和處理在 TON 區塊鏈中；其餘的在透過 TON 網路可用的鏈下伺服器上。

4.1.3. 中心化：中心化和去中心化，或分散式應用程式. 另一個分類標準是應用程式（或服務）是否依賴於中心化伺服器叢集，或真正是「分散式」的（參見 **4.1.9**）。所有鏈上應用程式都自動去中心化和分散式。鏈下和混合應用程式可能表現出不同程度的中心化。

現在讓我們更詳細地考慮上述可能性。

4.1.4. 純「鏈上」應用程式：駐留在區塊鏈中的分散式應用程式，或「dapp」. 在 **4.1.2** 中提到的可能方法之一是將「分散式應用程式」（通常縮寫為「dapp」）完全部署在 TON 區塊鏈中，作為一個智慧合約或智慧合約的集合。所有資料都將作為這些智慧合約的永久狀態的一部分保留，並且與專案的所有互動都將透過發送到這些智慧合約或從這些智慧合約接收的（TON 區塊鏈）訊息來完成。

我們已經在 **2.9.13** 中討論過這種方法有其缺點和限制。它也有其優點：這樣的分散式應用程式不需要伺服器來執行或儲存其資料（它在「區塊鏈中」執行——即在驗證者的硬體上），並且享有區塊鏈的極高（拜占庭）可靠性和可存取性。這種分散式應用程式的開發者不需要購買或租用任何硬體；她需要做的只是開發一些軟體（即智慧合約的程式碼）。之後，她將有效地從驗證者那裡租用計算能力，並以 Gram 支付，要麼自己支付，要麼將此負擔轉嫁給其使用者。

4.1.5. 純網路服務：「ton-site」和「ton-service」. 另一個極端選擇是在某些伺服器上部署服務，並透過 **3.1** 中描述的 ADNL 協定向使用者提供服務，也許還有一些更高層級的協定，例如 **3.1.9** 中討論的 RLDP，可用於以任何自訂格式向服務發送 RPC 查詢並獲取這些查詢的答案。這樣，服務將完全離鏈，並將駐留在 TON 網路中，幾乎不使用 TON 區塊鏈。

TON 區塊鏈可能僅用於定位服務的抽象位址或位址，如 **3.2.12** 中所述，也許在 TON DNS（參見 **4.3.1**）等服務的幫助下，以促進將類似域的人類可讀字串轉換為抽象位址。

在 ADNL 網路（即 TON 網路）與 Invisible Internet Project（I^2P）相似的程度上，這種（幾乎）純粹的網路服務類似於所謂的「eep-service」（即，具有 I^2P 位址作為其入口點的服務，並透過 I^2P 網路向客戶端提供）。我們將說駐留在 TON 網路中的這種純網路服務是「ton-service」。

「eep-service」可能將 HTTP 實作為其客戶端-伺服器協定；在 TON 網路的背景下，「ton-service」可能只是使用 RLDP（參見 **3.1.9**）資料包來傳輸 HTTP 查詢和對它們的回應。如果它使用 TON DNS 允許其抽象位址透過人類可讀的域名查找，則與網站的類比幾乎完美。人們甚至可以編寫專用瀏覽器，或在使用者的機器上本地執行的特殊代理（「ton-proxy」），接受使用者使用的普通網頁瀏覽器的任意 HTTP 查詢（一旦將本地 IP 位址和代理的 TCP 埠輸入到瀏覽器的配置中），並透過 TON 網路將這些查詢轉發到服務的抽象位址。然後，使用者將擁有類似於萬維網（WWW）的瀏覽體驗。

在 I^2P 生態系統中，這樣的「eep-service」被稱為「eep-site」。也可以在 TON 生態系統中輕鬆建立「ton-site」。TON DNS 等服務的存在在一定程度上促進了這一點，這些服務利用 TON 區塊鏈和 TON DHT 將（TON）

域名轉換為抽象位址。

4.1.6. Telegram Messenger 作為 ton-service；RLDP 上的 MTProto. 我們想順便提一下，Telegram Messenger[34] 用於客戶端-伺服器互動的 MTProto 協定，[35] 可以輕鬆嵌入到 **3.1.9** 中討論的 RLDP 協定中，從而有效地將 Telegram 轉變為 ton-service。因為 TON Proxy 技術可以對 ton-site 或 ton-service 的終端使用者透明地開啟，在 RLDP 和 ADNL 協定的較低層級上實作（參見 **3.1.6**），這將使 Telegram 有效地無法封鎖。當然，其他訊息和社交網路服務也可能從這項技術中受益。

4.1.7. 混合服務：部分鏈下，部分鏈上. 一些服務可能使用混合方法：大部分處理在鏈下進行，但也有一些鏈上部分（例如，向其使用者註冊其義務，反之亦然）。這樣，部分狀態仍然保留在 TON 區塊鏈中（即不可變的公開分類帳），並且服務或其使用者的任何不當行為都可以透過智慧合約受到懲罰。

4.1.8. 範例：在鏈下保存檔案；TON Storage. 這種服務的一個例子是 *TON Storage*。在其最簡單的形式中，它允許使用者在鏈下儲存檔案，僅在鏈上保留要儲存的檔案的雜湊，並且可能在智慧合約中，一些其他方同意在給定的時間段內以預先協商的費用保存相關檔案。事實上，檔案可以細分為一些小尺寸（例如，1 千位元組）的塊，透過擦除碼（如 Reed–Solomon 或噴泉碼）增強，可以為增強的塊序列構建默克爾樹雜湊，並且可以在智慧合約中發布此默克爾樹雜湊，而不是或與檔案的通常雜湊一起。這有點讓人想起種子中儲存檔案的方式。

儲存檔案的更簡單形式是完全鏈下：可以基本上為新檔案建立「種子」，並將 TON DHT 用作此種子的「分散式種子追蹤器」（參見 **3.2.10**）。這對於流行檔案實際上可能運作得很好。然而，人們沒有得到任何可用性保證。例如，一個假設的「區塊鏈 Facebook」（參見 **2.9.13**），它選擇在此類「種子」中完全鏈下保存其使用者的個人資料照片，可能會冒失去普通（不是特別流行）使用者的照片的風險，或至少冒在長時間內無法呈現這些照片的風險。TON Storage 技術主要是鏈下的，但使用鏈上智慧合約來強制儲存檔案的可用性，可能更適合這項任務。

4.1.9. 去中心化混合服務，或「霧服務」. 到目前為止，我們已經討論了中心化混合服務和應用程式。雖然它們的鏈上元件以去中心化和分散式的方式處理，位於區塊鏈中，但它們的鏈下元件依賴於服務提供者以通常的中心化方式控制的某些伺服器。可以從大公司之一提供的雲端計算服務租用

[34] `https://telegram.org/`

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

計算能力，而不是使用一些專用伺服器。然而，這不會導致服務的鏈下元件去中心化。

實作服務的鏈下元件的去中心化方法包括建立一個市場，任何擁有所需硬體並願意租用其計算能力或磁碟空間的人都可以向需要它們的人提供服務。

例如，可能存在一個註冊表（也可以稱為「市場」或「交易所」），所有對保留其他使用者的檔案感興趣的節點都在其中發布其聯繫資訊，以及其可用儲存容量、可用性政策和價格。那些需要這些服務的人可能在那裡查找它們，並且如果另一方同意，在區塊鏈中建立智慧合約並上傳檔案以進行鏈下儲存。這樣，像 *TON Storage* 這樣的服務變得真正去中心化，因為它不需要依賴任何中心化的伺服器叢集來儲存檔案。

4.1.10. 範例：「霧計算」平台作為去中心化混合服務. 當人們想要執行一些特定的計算（例如，3D 渲染或訓練神經網路），通常需要特定和昂貴的硬體時，會出現這種去中心化混合應用程式的另一個例子。然後，擁有此類設備的人可能透過類似的「交易所」提供其服務，而那些需要此類服務的人將租用它們，雙方的義務透過智慧合約註冊。這類似於「霧計算」平台（例如 Golem（`https://golem.network/`）或 SONM（`https://sonm.io/`））承諾提供的服務。

4.1.11. 範例：TON Proxy 是一種霧服務. *TON Proxy* 提供了霧服務的另一個例子，希望作為 ADNL 網路流量的隧道（無論有無補償）提供服務的節點可能會註冊，而那些需要它們的節點可能會根據提供的價格、延遲和頻寬選擇其中一個節點。之後，可能使用 *TON Payments* 提供的支付通道來處理這些代理服務的微支付，例如每傳輸 128 KiB 收取一次費用。

4.1.12. 範例：TON Payments 是一種霧服務. TON Payments 平台（參見 **5**）也是這種去中心化混合應用程式的一個例子。

4.2 連接使用者和服務提供者

我們在 **4.1.9** 中看到，「霧服務」（即混合去中心化服務）通常需要一些市場、交易所或註冊表，需要特定服務的人可以在那裡與提供這些服務的人見面。

這些市場可能被實作為鏈上、鏈下或混合服務本身，中心化或分散式。

4.2.1. 範例：連接到 TON Payments. 例如，如果想要使用 TON Payments（參見 **5**），第一步是找到「閃電網路」（參見 **5.2**）的至少一些現有中繼節點，並在它們願意的情況下與它們建立支付通道。可以在「包含」覆

蓋網路的幫助下找到一些節點，該覆蓋網路應該包含所有中繼閃電網路節點（參見 **3.3.17**）。然而，不清楚這些節點是否願意建立新的支付通道。因此，需要一個註冊表，準備建立新連結的節點可以在其中發布其聯繫資訊（例如，其抽象位址）。

4.2.2. 範例：將檔案上傳到 TON Storage. 類似地，如果想要將檔案上傳到 TON Storage，她必須找到一些願意簽署智慧合約以約束它們保存該檔案（或任何低於某個大小限制的檔案）副本的節點。因此，需要提供儲存檔案服務的節點註冊表。

4.2.3. 鏈上、混合和鏈下註冊表. 這種服務提供者註冊表可能完全在鏈上實作，在智慧合約的幫助下，該智慧合約將註冊表保留在其永久儲存中。然而，這將是相當慢和昂貴的。混合方法更有效率，其中相對較小且很少更改的鏈上註冊表僅用於指出一些節點（透過其抽象位址，或透過其公鑰，可用於如 **3.2.12** 中所述定位實際抽象位址），這些節點提供鏈下（中心化）註冊表服務。

最後，去中心化、純鏈下方法可能由公開覆蓋網路（參見 **3.3**）組成，那些願意提供服務的人，或那些希望購買某人服務的人，只需廣播其報價，由其私鑰簽署。如果要提供的服務非常簡單，甚至可能不需要廣播報價：覆蓋網路本身的近似成員資格可能被用作願意提供特定服務的人的「註冊表」。然後，需要此服務的客戶端可能定位（參見 **3.3.7**）並查詢此覆蓋網路的一些節點，然後查詢它們的鄰居，如果已知的節點尚未準備好滿足其需求。

4.2.4. 側鏈中的註冊表或交易所. 實作去中心化混合註冊表的另一種方法包括建立一個獨立的專門區塊鏈（「側鏈」），由其自己的一組自封驗證者維護，他們在鏈上智慧合約中發布其身份，並透過專用覆蓋網路（參見 **3.3**）向所有對此專門區塊鏈感興趣的各方提供網路存取，收集交易候選並廣播區塊更新。然後，此側鏈的任何完整節點都可以維護其自己的共享註冊表副本（基本上等於此側鏈的全域狀態），並處理與此註冊表相關的任意查詢。

4.2.5. 工作鏈中的註冊表或交易所. 另一個選擇是在 TON 區塊鏈內建立專用工作鏈，專門用於建立註冊表、市場和交易所。這可能比使用駐留在基本工作鏈中的智慧合約（參見 **2.1.11**）更有效率和更便宜。然而，這仍然比在側鏈中維護註冊表（參見 **4.2.4**）更昂貴。

4.3 存取 TON 服務

我們在 **4.1** 中討論了建立駐留在 TON 生態系統中的新服務和應用程式可能採用的不同方法。現在我們討論如何存取這些服務，以及 TON 將提供的一些「輔助服務」，包括 *TON DNS* 和 *TON Storage*。

4.3.1. TON DNS：主要是鏈上的階層式域名服務. *TON DNS* 是一個預定義的服務，它使用智慧合約集合來保持從人類可讀域名到 ADNL 網路節點和 TON 區塊鏈帳戶和智慧合約的（256 位元）位址的對應。

雖然原則上任何人都可能使用 TON 區塊鏈實作此類服務，但擁有一個具有眾所周知介面的預定義服務是有用的，每當應用程式或服務想要將人類可讀識別碼轉換為位址時，預設使用它。

4.3.2. TON DNS 使用情境. 例如，希望向另一個使用者或商家轉移一些加密貨幣的使用者可能更喜歡記住該使用者或商家的帳戶的 TON DNS 域名，而不是手邊保留其 256 位元帳戶識別碼並將它們複製貼上到其輕量級錢包客戶端的收件人欄位中。

類似地，TON DNS 可用於定位智慧合約的帳戶識別碼或 ton-service 和 ton-site 的入口點（參見 **4.1.5**），使專用客戶端（「ton-browser」），或結合專用 ton-proxy 擴展或獨立應用程式的普通網際網路瀏覽器，能夠為使用者提供類似 WWW 的瀏覽體驗。

4.3.3. TON DNS 智慧合約. TON DNS 透過特殊（DNS）智慧合約樹來實作。每個 DNS 智慧合約負責註冊某個固定域的子域。「根」DNS 智慧合約（TON DNS 系統的一級域將保存在其中）位於主鏈中。其帳戶識別碼必須硬編碼到所有希望直接存取 TON DNS 資料庫的軟體中。

任何 DNS 智慧合約都包含一個雜湊表，將可變長度的以空字元結尾的 UTF-8 字串對應到其「值」。此雜湊表實作為二進位 Patricia 樹，類似於 **2.3.7** 中描述的樹，但支援可變長度位元字串作為鍵。

4.3.4. DNS 雜湊表的值，或 TON DNS 記錄. 至於值，它們是由 TL-scheme（參見 **2.2.5**）描述的「TON DNS 記錄」。它們由「魔術數字」組成，選擇支援的選項之一，然後是帳戶識別碼、智慧合約識別碼、抽象網路位址（參見 **3.1**）、用於定位服務的抽象位址的公鑰（參見 **3.2.12**）、覆蓋網路的描述等。一個重要的情況是另一個 DNS 智慧合約：在這種情況下，該智慧合約用於解析其域的子域。這樣，可以為不同的域建立單獨的註冊表，由這些域的擁有者控制。

這些記錄還可能包含到期時間、快取時間（通常非常大，因為太頻繁地更新區塊鏈中的值很昂貴），並且在大多數情況下引用相關子域的擁有者。

擁有者有權更改此記錄（特別是擁有者欄位，從而將域轉移到其他人的控制之下），並延長它。

4.3.5. 註冊現有域的新子域. 為了註冊現有域的新子域，只需向智慧合約發送訊息，該智慧合約是該域的註冊機構，包含要註冊的子域（即鍵）、幾種預定義格式之一中的值、擁有者的身份、到期日期以及由域擁有者決定的一定數量的加密貨幣。

子域按「先來先服務」的原則註冊。

4.3.6. 從 DNS 智慧合約檢索資料. 原則上，包含 DNS 智慧合約的主鏈或分片鏈的任何完整節點都可能能夠在該智慧合約的資料庫中查找任何子域，如果已知智慧合約的永久儲存中雜湊表的結構和位置。

然而，這種方法只適用於某些 DNS 智慧合約。如果使用非標準 DNS 智慧合約，它將徹底失敗。

相反，使用基於一般智慧合約介面和 *get* 方法（參見 **4.3.11**）的方法。任何 DNS 智慧合約都必須定義一個具有「已知簽名」的「get 方法」，該方法被呼叫以查找鍵。由於這種方法對其他智慧合約也有意義，尤其是那些提供鏈上和混合服務的智慧合約，我們在 **4.3.11** 中詳細解釋它。

4.3.7. 轉換 TON DNS 域. 一旦任何完整節點，無論是自己行動還是代表某個輕客戶端，都可以在任何 DNS 智慧合約的資料庫中查找條目，就可以遞迴轉換任意 TON DNS 域名，從眾所周知且固定的根 DNS 智慧合約（帳戶）識別碼開始。

例如，如果想要轉換 `A.B.C`，則在根域資料庫中查找鍵 `.C`、`.B.C` 和 `A.B.C`。如果找不到第一個，但找到第二個，並且其值是對另一個 DNS 智慧合約的引用，則在該智慧合約的資料庫中查找 `A` 並檢索最終值。

4.3.8. 為輕節點轉換 TON DNS 域. 這樣，主鏈的完整節點——以及參與域查找過程的所有分片鏈的完整節點——可以在沒有外部幫助的情況下將任何域名轉換為其當前值。輕節點可能請求完整節點代表其執行此操作並返回值，以及默克爾證明（參見 **2.5.11**）。此默克爾證明將使輕節點能夠驗證答案是否正確，因此此類 TON DNS 回應不能被惡意攔截者「欺騙」，這與通常的 DNS 協定形成對比。

因為不能期望任何節點都是所有分片鏈的完整節點，實際的 TON DNS 域轉換將涉及這兩種策略的組合。

4.3.9. 專用「TON DNS 伺服器」. 可以提供簡單的「TON DNS 伺服器」，它將接收 RPC「DNS」查詢（例如，透過 **3.1** 中描述的 ADNL 或 RLDP 協定），請求伺服器轉換給定的域，如果需要，透過將一些子查詢轉發給

其他（完整）節點來處理這些查詢，並返回對原始查詢的答案，如果需要，則透過默克爾證明增強。

此類「DNS 伺服器」可能使用 **4.2** 中描述的方法之一向任何其他節點，尤其是輕客戶端，提供其服務（無論是否免費）。請注意，如果這些伺服器被視為 TON DNS 服務的一部分，它們將有效地將其從分散式鏈上服務轉變為分散式混合服務（即「霧服務」）。

這結束了我們對 TON DNS 服務的簡要概述，這是一個針對 TON 區塊鏈和 TON 網路實體的人類可讀域名的可擴展鏈上註冊表。

4.3.10. 存取保存在智慧合約中的資料. 我們已經看到，有時需要存取儲存在智慧合約中的資料而不改變其狀態。

如果知道智慧合約實作的細節，則可以從智慧合約的永久儲存中提取所有所需資訊，該儲存對智慧合約所在分片鏈的所有完整節點可用。然而，這是一種非常不優雅的做事方式，非常依賴於智慧合約實作。

4.3.11. 智慧合約的「get 方法」. 更好的方法是在智慧合約中定義一些 *get* 方法，即一些類型的入站訊息，它們在傳遞時不影響智慧合約的狀態，但生成一個或多個包含 get 方法「結果」的輸出訊息。這樣，可以從智慧合約獲取資料，只需知道它實作具有已知簽名的 get 方法（即要發送的入站訊息和作為結果接收的出站訊息的已知格式）。

這種方式更優雅，符合物件導向程式設計（OOP）。然而，到目前為止，它有一個明顯的缺陷：必須實際將交易提交到區塊鏈（向智慧合約發送 get 訊息），等待它被驗證者提交和處理，從新區塊中提取答案，並支付 gas 費用（即在驗證者的硬體上執行 get 方法）。這是資源的浪費：get 方法無論如何都不會改變智慧合約的狀態，因此它們不需要在區塊鏈中執行。

4.3.12. 智慧合約的 get 方法的試探性執行. 我們已經指出（參見 **2.4.6**），任何完整節點都可以從智慧合約的給定狀態開始試探性地執行任何智慧合約的任何方法（即向智慧合約傳遞任何訊息），而不實際提交相應的交易。完整節點可以簡單地將所考慮的智慧合約的程式碼載入到 TON VM 中，從分片鏈的全域狀態（分片鏈的所有完整節點都知道）初始化其永久儲存，並以入站訊息作為其輸入參數執行智慧合約程式碼。建立的輸出訊息將產生此計算的結果。

這樣，任何完整節點都可以評估任意智慧合約的任意 get 方法，前提是已知它們的簽名（即入站和出站訊息的格式）。節點可以追蹤此評估期間存取的分片鏈狀態的單元，並建立執行的計算有效性的默克爾證明，以供可能要求完整節點這樣做的輕節點使用（參見 **2.5.11**）。

4.3.13. TL-scheme 中的智慧合約介面. 回想一下，智慧合約實作的方法（即它接受的輸入訊息）本質上是一些 TL 序列化物件，可以由 TL-scheme

描述（參見 **2.2.5**）。結果輸出訊息也可以由相同的 TL-scheme 描述。這樣，智慧合約向其他帳戶和智慧合約提供的介面可以透過 TL-scheme 正式化。

特別是，智慧合約支援的 get 方法（子集）可以由這樣的正式化智慧合約介面描述。

4.3.14. 智慧合約的公開介面. 請注意，正式化的智慧合約介面，無論是 TL-scheme 的形式（表示為 TL 原始檔案；參見 **2.2.5**）還是序列化形式，[36] 都可以發布——例如，在儲存在區塊鏈中的智慧合約帳戶描述中的特殊欄位中，或者如果此介面將被多次引用則單獨發布。在後一種情況下，支援的公開介面的雜湊可能被合併到智慧合約描述中，而不是介面描述本身。

這種公開介面的一個例子是 DNS 智慧合約的介面，它應該至少實作一個用於查找子域的標準 get 方法（參見 **4.3.6**）。用於註冊新子域的標準方法也可以包含在 DNS 智慧合約的標準公開介面中。

4.3.15. 智慧合約的使用者介面. 智慧合約的公開介面的存在也有其他好處。例如，錢包客戶端應用程式可以在根據使用者的請求檢查智慧合約時下載這樣的介面，並顯示智慧合約支援的公開方法（即可用操作）列表，如果正式介面中提供了任何人類可讀的註解，也許還會顯示它們。在使用者選擇這些方法之一後，可以根據 TL-scheme 自動生成一個表單，使用者將被提示輸入所選方法所需的所有欄位以及要附加到此請求的所需加密貨幣（例如 Gram）數量。提交此表單將建立一個新的區塊鏈交易，其中包含剛剛組成的訊息，從使用者的區塊鏈帳戶發送。

這樣，使用者將能夠透過填寫和提交某些表單以使用者友好的方式從錢包客戶端應用程式與任意智慧合約互動，前提是這些智慧合約已發布其介面。

4.3.16. 「ton-service」的使用者介面. 事實證明，「ton-service」（即駐留在 TON 網路中並透過 **3** 的 ADNL 和 RLDP 協定接受查詢的服務；參見 **4.1.5**）也可能從擁有由 TL-scheme 描述的公開介面中受益（參見 **2.2.5**）。客戶端應用程式，例如輕量級錢包或「ton-browser」，可能提示使用者選擇其中一個方法並填寫具有介面定義的參數的表單，類似於剛才在 **4.3.15** 中討論的內容。唯一的區別是，結果 TL 序列化訊息不作為區塊鏈中的交易提交；相反，它作為 RPC 查詢發送到相關「ton-service」的抽象位址，並且根據正式介面（即 TL-scheme）解析和顯示對此查詢的回應。

[36] TL-scheme 本身可以進行 TL 序列化；參見 `https://core.telegram.org/mtproto/TL-tl`。

4.3.17. 透過 TON DNS 定位使用者介面. 包含 ton-service 的抽象位址或智慧合約帳戶識別碼的 TON DNS 記錄還可能包含描述該實體的公開（使用者）介面或幾個支援的介面的可選欄位。然後，客戶端應用程式（無論是錢包、ton-browser 還是 ton-proxy）將能夠下載介面並以統一的方式與相關實體（無論是智慧合約還是 ton-service）互動。

4.3.18. 模糊鏈上和鏈下服務之間的區別. 這樣，對於終端使用者來說，鏈上、鏈下和混合服務之間的區別（參見 **4.1.2**）變得模糊：她只需將所需服務的域名輸入到其 ton-browser 或錢包的位址列中，其餘部分由客戶端應用程式無縫處理。

4.3.19. 輕量級錢包和 TON 實體瀏覽器可以內建到 Telegram Messenger 客戶端中. 此時出現了一個有趣的機會。實作上述功能的輕量級錢包和 TON 實體瀏覽器可以嵌入到 Telegram Messenger 智慧型手機客戶端應用程式中，從而將技術帶給超過 2 億人。使用者將能夠透過在訊息中包含 TON URI（參見 **4.3.22**）來發送到 TON 實體和資源的超連結；如果選擇這樣的超連結，它們將由接收方的 Telegram 客戶端應用程式內部開啟，並且將開始與所選實體的互動。

4.3.20. 「ton-site」作為支援 HTTP 介面的 ton-service. *ton-site* 只是支援 HTTP 介面的 ton-service，也許還有一些其他介面。此支援可以在相應的 TON DNS 記錄中宣布。

4.3.21. 超連結. 請注意，ton-site 返回的 HTML 頁面可能包含 *ton* 超連結——即透過特別製作的 URI 方案（參見 **4.3.22**）對其他 ton-site、智慧合約和帳戶的引用——包含抽象網路位址、帳戶識別碼或人類可讀的 TON DNS 域。然後，當使用者選擇這樣的超連結時，「ton-browser」可能跟隨它，偵測要使用的介面，並如 **4.3.15** 和 **4.3.16** 中所述顯示使用者介面表單。

4.3.22. 超連結 URL 可以指定一些參數. 超連結 URL 可能不僅包含相關服務的（TON）DNS 域或抽象位址，還包含要呼叫的方法的名稱以及其某些或全部參數。可能的 URI 方案可能如下所示：

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

當使用者在 ton-browser 中選擇這樣的連結時，要麼立即執行操作（特別是如果它是匿名呼叫的智慧合約的 get 方法），要麼顯示部分填寫的表單，由使用者明確確認和提交（這可能是支付表單所必需的）。

4.3.23. POST 操作. ton-site 可以在其返回的 HTML 頁面中嵌入一些看起來通常的 POST 表單，其 POST 操作透過適當的（TON）URL 引用

ton-site、ton-service 或智慧合約。在這種情況下，一旦使用者填寫並提交該自訂表單，就會採取相應的操作，要麼立即執行，要麼在明確確認後執行。

4.3.24. TON WWW. 以上所有內容將導致建立一個駐留在 TON 網路中的互相引用實體的整個網路，終端使用者可以透過 ton-browser 存取它，為使用者提供類似 WWW 的瀏覽體驗。對於終端使用者來說，這最終將使區塊鏈應用程式在根本上類似於他們已經習慣的網站。

4.3.25. TON WWW 的優點. 這個由鏈上和鏈下服務組成的「TON WWW」相較於傳統對應系統有一些優點。例如，支付原生整合在系統中。使用者身分可以隨時向服務呈現（透過對產生的交易和 RPC 請求自動產生的簽章），或根據需要隱藏。服務不需要檢查和再檢查使用者憑證；這些憑證可以一次性發布在區塊鏈中。使用者網路匿名性可以透過 TON Proxy 輕鬆保留，並且所有服務實際上都將無法被封鎖。微支付也是可能且容易的，因為 ton-browser 可以與 TON Payments 系統整合。

5 TON Payments

我們在本文中簡要討論的 TON 專案的最後一個組件是 *TON Payments*，即（微）支付通道和「閃電網路」價值轉移的平台。它將實現「即時」支付，無需將所有交易提交到區塊鏈、支付相關交易費用（例如，用於消耗的 gas），以及等待五秒直到包含相關交易的區塊被確認。

這種即時支付的整體開銷如此之小，以至於可以將它們用於微支付。例如，TON 檔案儲存服務可能會對使用者下載的每 128 KiB 資料收費，或者付費 TON Proxy 可能會對轉送的每 128 KiB 流量要求一些微小的微支付。

雖然 *TON Payments* 可能會在 TON 專案的核心組件之後發布，但需要在一開始就做一些考慮。例如，用於執行 TON 區塊鏈智慧合約程式碼的 TON 虛擬機（TON VM；參見 **2.1.20**）必須支援一些與 Merkle 證明相關的特殊操作。如果原始設計中不存在這種支援，則在後期階段添加它可能會變得有問題（參見 **2.8.16**）。然而，我們將看到，TON VM 天生支援「智慧」支付通道（參見 **5.1.9**）。

5.1 支付通道

我們從點對點支付通道的討論開始，以及如何在 TON 區塊鏈中實作它們。

5.1.1. 支付通道的想法. 假設兩方 A 和 B 知道他們將來需要相互進行大量支付。他們不是將每筆支付作為交易提交到區塊鏈中，而是建立一個共享的「資金池」（或者也許是一個恰好有兩個帳戶的小型私人銀行），並向其中貢獻一些資金：A 貢獻 a 枚代幣，B 貢獻 b 枚代幣。這是透過在區塊鏈中建立一個特殊的智慧合約，並將資金發送到該合約來實現的。

在建立「資金池」之前，雙方同意某個協定。他們將追蹤資金池的狀態——即他們在共享池中的餘額。最初，狀態為 (a, b)，意思是 a 枚代幣實際上屬於 A，而 b 枚代幣屬於 B。然後，如果 A 想要向 B 支付 d 枚代幣，他們只需同意新狀態為 $(a', b') = (a - d, b + d)$。之後，如果說 B 想要向 A 支付 d' 枚代幣，狀態將變為 $(a'', b'') = (a' + d', b' - d')$，依此類推。

所有這些池內餘額的更新都完全在鏈下進行。當雙方決定從池中提取其應得資金時，他們根據池的最終狀態進行提取。這是透過向智慧合約發送一條特殊訊息來實現的，該訊息包含雙方同意的最終狀態 (a^*, b^*) 以及 A 和 B 的簽章。然後智慧合約將 a^* 枚代幣發送給 A，將 b^* 枚代幣發送給 B，並自毀。

這個智慧合約，以及 A 和 B 用於更新池狀態的網路協定，就是 A 和 B 之間的一個簡單支付通道。根據 **4.1.2**中描述的分類，它是一個混合服務：其部分狀態駐留在區塊鏈中（智慧合約），但其大部分狀態更新都在鏈下執

行（透過網路協定）。如果一切順利，雙方將能夠相互進行任意多次支付(唯一的限制是通道的「容量」不被超越——即他們在支付通道中的餘額都保持非負)，只需將兩筆交易提交到區塊鏈中：一筆用於開啟（建立）支付通道（智慧合約)，另一筆用於關閉（銷毀）它。

5.1.2. 無需信任的支付通道. 前面的例子有些不切實際，因為它假設雙方都願意合作，並且永遠不會為了獲得某些優勢而作弊。例如，想像一下，A 會選擇不簽署 $a' < a$ 的最終餘額 (a', b')。這將使 B 處於困難的境地。

為了防止這種情況，通常會嘗試開發無需信任的支付通道協定，這些協定不需要各方相互信任，並為懲罰任何試圖作弊的一方做出規定。

這通常透過簽章的幫助來實現。支付通道智慧合約知道 A 和 B 的公鑰，並且可以在需要時檢查它們的簽章。支付通道協定要求各方簽署中間狀態並將簽章發送給對方。然後，如果其中一方作弊——例如，假裝支付通道的某個狀態從未存在過——可以透過顯示其在該狀態上的簽章來證明其不當行為。支付通道智慧合約充當「鏈上仲裁者」，能夠處理雙方對彼此的投訴，並透過沒收其所有資金並將其獎勵給另一方來懲罰有罪方。

5.1.3. 簡單的雙向同步無需信任支付通道. 考慮以下更現實的例子：讓支付通道的狀態由三元組 (δ_i, i, o_i) 描述，其中 i 是狀態的序號（最初為零，然後在出現後續狀態時增加一)，δ_i 是通道不平衡（意思是 A 和 B 分別擁有 $a + \delta_i$ 和 $b - \delta_i$ 枚代幣)，o_i 是允許產生下一個狀態的一方（A 或 B)。在取得任何進一步進展之前，每個狀態都必須由 A 和 B 簽署。

現在，如果 A 想要在支付通道內向 B 轉移 d 枚代幣，並且當前狀態是 $S_i = (\delta_i, i, o_i)$ 且 $o_i = A$，那麼它只需建立一個新狀態 $S_{i+1} = (\delta_i - d, i + 1, o_{i+1})$，簽署它，並將其與簽章一起發送給 B。然後 B 透過簽署並將其簽章的副本發送給 A 來確認它。之後，雙方都有新狀態的副本，並且都有雙方的簽章，可以進行新的轉移。

如果 A 想要在狀態 S_i 中向 B 轉移代幣，其中 $o_i = B$，那麼它首先要求 B 提交一個後續狀態 S_{i+1}，其具有相同的不平衡 $\delta_{i+1} = \delta_i$，但 $o_{i+1} = A$。之後，A 將能夠進行轉移。

當雙方同意關閉支付通道時，他們都在他們認為是最終狀態的 S_k 上放置他們的特殊最終簽章，並透過向支付通道智慧合約發送最終狀態以及兩個最終簽章來呼叫其清理或雙方終結方法。

如果另一方不同意提供其最終簽章，或者如果它只是停止回應，則可以單方面關閉通道。為此，希望這樣做的一方將呼叫單方面終結方法，向智慧合約發送其版本的最終狀態、其最終簽章以及具有另一方簽章的最新狀態。之後，智慧合約不會立即根據收到的最終狀態採取行動。相反，它會等待一定時間（例如，一天）讓另一方呈現其最終狀態的版本。當另一方提交其版本並且它與已提交的版本相容時，智慧合約會計算「真正的」最

終狀態，並用它來相應地分配資金。如果另一方未能向智慧合約呈現其最終狀態的版本，則根據呈現的最終狀態的唯一副本重新分配資金。

如果兩方中的一方作弊——例如，透過將兩個不同的狀態簽署為最終狀態，或透過簽署兩個不同的下一個狀態 S_{i+1} 和 S'_{i+1}，或透過簽署無效的新狀態 S_{i+1}（例如，不平衡 $\delta_{i+1} < -a$ 或 $> b$）——那麼另一方可以向智慧合約的第三個方法提交此不當行為的證明。有罪方立即受到懲罰，完全失去其在支付通道中的份額。

這種簡單的支付通道協定是公平的，因為任何一方總是可以獲得其應得的份額，無論是否有另一方的合作，並且如果它試圖作弊，很可能會失去其提交到支付通道的所有資金。

5.1.4. 同步支付通道作為具有兩個驗證者的簡單虛擬區塊鏈. 上述簡單同步支付通道的例子可以重新描述如下。想像狀態序列 S_0、S_1、...、S_n 實際上是一個非常簡單的區塊鏈的區塊序列。這個區塊鏈的每個區塊本質上只包含區塊鏈的當前狀態，也許還有對前一個區塊的引用（即其雜湊）。雙方 A 和 B 都充當這個區塊鏈的驗證者，因此每個區塊都必須收集他們兩人的簽章。區塊鏈的狀態 S_i 定義了下一個區塊的指定生產者 o_i，因此 A 和 B 之間沒有生產下一個區塊的競爭。生產者 A 被允許建立從 A 向 B 轉移資金的區塊（即，減少不平衡：$\delta_{i+1} \leq \delta_i$），而 B 只能從 B 向 A 轉移資金（即，增加 δ）。

如果兩個驗證者同意區塊鏈的最終區塊（和最終狀態），則透過收集雙方的特殊「最終」簽章，並將它們與最終區塊一起提交給通道智慧合約進行處理並相應地重新分配資金來終結它。

如果驗證者簽署無效的區塊，或建立分叉，或簽署兩個不同的最終區塊，則可以透過向智慧合約呈現其不當行為的證明來懲罰它，智慧合約充當兩個驗證者的「鏈上仲裁者」；然後違規方將失去其保存在支付通道中的所有資金，這類似於驗證者失去其質押。

5.1.5. 非同步支付通道作為具有兩個工作鏈的虛擬區塊鏈. **5.1.3**中討論的同步支付通道有一個缺點：在前一個交易被另一方確認之前，無法開始下一個交易（支付通道內的資金轉移）。這可以透過將 **5.1.4**中討論的單個虛擬區塊鏈替換為兩個相互作用的虛擬工作鏈（或者更確切地說是分片鏈）的系統來修復。

這些工作鏈中的第一個僅包含 A 的交易，其區塊只能由 A 產生；其狀態為 $S_i = (i, \phi_i, j, \psi_j)$，其中 i 是區塊序號（即，A 到目前為止執行的交易或資金轉移的計數），ϕ_i 是到目前為止從 A 轉移到 B 的總金額，j 是 A 知道的 B 區塊鏈中最新有效區塊的序號，ψ_j 是在其 j 筆交易中從 B 轉移到 A 的資金金額。B 放在其第 j 個區塊上的簽章也應該是此狀態的一部分。此工作鏈的前一個區塊和另一個工作鏈的第 j 個區塊的雜湊也可以包括在

內。S_i 的有效性條件包括 $\phi_i \geq 0$，如果 $i > 0$ 則 $\phi_i \geq \phi_{i-1}$，$\psi_j \geq 0$，以及 $-a \leq \psi_j - \phi_i \leq b$。

類似地，第二個工作鏈僅包含 B 的交易，其區塊僅由 B 產生；其狀態為 $T_j = (j, \psi_j, i, \phi_i)$，具有類似的有效性條件。

現在，如果 A 想要向 B 轉移一些資金，它只需在其工作鏈中建立一個新區塊，簽署它，並發送給 B，而無需等待確認。

支付通道透過 A 簽署（其版本的）其區塊鏈的最終狀態（使用其特殊「最終簽章」），B 簽署其區塊鏈的最終狀態，並將這兩個最終狀態呈現給支付通道智慧合約的清理終結方法來終結。單方面終結也是可能的，但在這種情況下，智慧合約將不得不等待另一方呈現其最終狀態的版本，至少在某個寬限期內。

5.1.6. 單向支付通道. 如果只有 A 需要向 B 進行支付（例如，B 是服務提供者，A 是其客戶），則可以建立單邊支付通道。本質上，它只是 **5.1.5**中描述的第一個工作鏈，沒有第二個。相反地，可以說 **5.1.5**中描述的非同步支付通道由兩個單向支付通道或「半通道」組成，由同一個智慧合約管理。

5.1.7. 更複雜的支付通道。承諾. 我們將在 **5.2.4**中看到，「閃電網路」（參見 **5.2**）能夠透過幾個支付通道的鏈進行即時資金轉移，這需要所涉及的支付通道具有更高程度的複雜性。

特別是，我們希望能夠提交「承諾」或「條件性資金轉移」：A 同意向 B 發送 c 枚代幣，但只有在滿足某個條件時 B 才能獲得資金，例如，如果 B 可以呈現某個字串 u，其中 $\textsc{Hash}(u) = v$ 對於已知的 v 值。否則，A 可以在一定時間後取回資金。

這樣的承諾可以很容易地透過簡單的智慧合約在鏈上實作。然而，我們希望承諾和其他類型的條件性資金轉移在支付通道中能夠在鏈下進行，因為它們大大簡化了沿著「閃電網路」中存在的支付通道鏈進行的資金轉移（參見 **5.2.4**）。

在 **5.1.4**和 **5.1.5**中概述的「支付通道作為簡單區塊鏈」的圖景在這裡變得很方便。現在我們考慮一個更複雜的虛擬區塊鏈，其狀態包含一組這樣的未履行「承諾」，以及鎖定在這些承諾中的資金金額。這個區塊鏈——或者在非同步情況下的兩個工作鏈——將不得不透過它們的雜湊明確引用前一個區塊。然而，一般機制保持不變。

5.1.8. 複雜支付通道智慧合約的挑戰. 請注意，雖然複雜支付通道的最終狀態仍然很小，並且「清理」終結很簡單（如果雙方已經就其應得金額達成一致，並且都簽署了他們的協議，則無需再做其他事情），但單方面終結方

法和懲罰欺詐行為的方法需要更複雜。實際上，它們必須能夠接受不當行為的 Merkle 證明，並檢查支付通道區塊鏈的更複雜交易是否已正確處理。

換句話說，支付通道智慧合約必須能夠處理 Merkle 證明，檢查它們的「雜湊有效性」，並且必須包含支付通道（虛擬）區塊鏈的 *ev_trans* 和 *ev_block* 函數的實作（參見 **2.2.6**）。

5.1.9. TON VM 對「智慧」支付通道的支援. 用於執行 TON 區塊鏈智慧合約程式碼的 TON VM，能夠勝任執行「智慧」或複雜支付通道所需的智慧合約的挑戰（參見 **5.1.8**）。

此時，「一切皆為單元集合」範式（參見 **2.5.14**）變得極為方便。由於所有區塊（包括臨時支付通道區塊鏈的區塊）都表示為單元集合（並由一些代數資料型別描述），訊息和 Merkle 證明也是如此，因此 Merkle 證明可以輕鬆嵌入到發送到支付通道智慧合約的入站訊息中。Merkle 證明的「雜湊條件」將自動檢查，當智慧合約存取所呈現的「Merkle 證明」時，它將處理它就像它是相應代數資料型別的值一樣——儘管不完整，樹的一些子樹被包含省略子樹的 Merkle 雜湊的特殊節點替換。然後智慧合約將處理該值，例如，該值可能表示支付通道（虛擬）區塊鏈的區塊及其狀態，並將在該區塊和前一個狀態上評估該區塊鏈的 *ev_block* 函數（參見 **2.2.6**）。然後，要麼計算完成，最終狀態可以與區塊中斷言的狀態進行比較，要麼在嘗試存取缺失的子樹時拋出「缺失節點」例外，表示 Merkle 證明無效。

透過這種方式，使用 TON 區塊鏈智慧合約實作智慧支付通道區塊鏈的驗證程式碼變得非常簡單。可以說 *TON* 虛擬機內建支援檢查其他簡單區塊鏈的有效性。唯一的限制因素是要併入智慧合約的入站訊息（即交易）中的 Merkle 證明的大小。

5.1.10. 智慧支付通道內的簡單支付通道. 我們想討論在現有支付通道內建立簡單（同步或非同步）支付通道的可能性。

雖然這看起來有些複雜，但它並不比 **5.1.7**中討論的「承諾」更難理解和實作。本質上，A 不是承諾如果呈現某個雜湊問題的解決方案就向另一方支付 c 枚代幣，而是承諾根據某個其他（虛擬）支付通道區塊鏈的最終結算向 B 支付最多 c 枚代幣。一般來說，這個其他支付通道區塊鏈甚至不需要在 A 和 B 之間；它可能涉及其他一些方，比如 C 和 D，願意分別向他們的簡單支付通道提交 c 和 d 枚代幣。(這種可能性稍後在 **5.2.5**中被利用。)

如果包含的支付通道是非對稱的，則需要將兩個承諾提交到兩個工作鏈中：如果「內部」簡單支付通道的最終結算產生負的最終不平衡 δ（其中 $0 \le -\delta \le c$），則 A 將承諾向 B 支付 $-\delta$ 枚代幣；如果 δ 為正，則 B 將不得不承諾向 A 支付 δ。另一方面，如果包含的支付通道是對稱的，這可以透過由 A 將帶有參數 (c, d) 的單個「簡單支付通道建立」交易提交到單個

支付通道區塊鏈中（這將凍結屬於 A 的 c 枚代幣），然後透過 B 提交特殊的「確認交易」（這將凍結 B 的 d 枚代幣）來完成。

我們期望內部支付通道極其簡單（例如， **5.1.3**中討論的簡單同步支付通道），以最小化要提交的 Merkle 證明的大小。外部支付通道將必須在 **5.1.7**中描述的意義上是「智慧的」。

5.2 支付通道網路，或「閃電網路」

現在我們準備討論 TON Payments 的「閃電網路」，它能夠在任何兩個參與節點之間進行即時資金轉移。

5.2.1. 支付通道的限制. 支付通道對於期望彼此之間進行大量資金轉移的各方很有用。然而，如果只需要向特定收件人轉移一兩次資金，則與她建立支付通道將不切實際。除其他事項外，這將意味著在支付通道中凍結大量資金，並且無論如何都需要至少兩個區塊鏈交易。

5.2.2. 支付通道網路，或「閃電網路」. 支付通道網路透過啟用沿支付通道鏈的資金轉移來克服支付通道的限制。如果 A 想要向 E 轉移資金，她不需要與 E 建立支付通道。只要有一個透過幾個中間節點連接 A 與 E 的支付通道鏈就足夠了——比如，四個支付通道：從 A 到 B、從 B 到 C、從 C 到 D 以及從 D 到 E。

5.2.3. 支付通道網路概述. 回想一下，支付通道網路，也稱為「閃電網路」，由一組參與節點組成，其中一些節點已在它們之間建立了長期存在的支付通道。我們稍後將看到，這些支付通道必須在 **5.1.7**的意義上是「智慧的」。當參與節點 A 想要向任何其他參與節點 E 轉移資金時，她嘗試在支付通道網路內找到連接 A 到 E 的路徑。當找到這樣的路徑時，她沿著這條路徑執行「鏈資金轉移」。

5.2.4. 鏈資金轉移. 假設存在從 A 到 B、從 B 到 C、從 C 到 D 以及從 D 到 E 的支付通道鏈。進一步假設 A 想要向 E 轉移 x 枚代幣。

一種簡化的方法是沿著現有支付通道向 B 轉移 x 枚代幣，並要求他將資金進一步轉發給 C。然而，不清楚為什麼 B 不會簡單地為自己拿走這些錢。因此，必須採用更複雜的方法，不需要所有相關方相互信任。

這可以透過如下方式實現。A 產生一個大的隨機數 u 並計算其雜湊 $v = \textsc{Hash}(u)$。然後她在與 B 的支付通道內建立一個承諾，如果呈現一個雜湊為 v 的數字 u，就向 B 支付 x 枚代幣（參見 **5.1.7**）。這個承諾包含 v，但不包含仍保密的 u。

之後，B 在他們的支付通道中向 C 建立類似的承諾。他不怕給出這樣的承諾，因為他知道 A 給他的類似承諾的存在。如果 C 曾經呈現雜湊問

題的解決方案來收取 B 承諾的 x 枚代幣，那麼 B 將立即向 A 提交此解決方案以從 A 收取 x 枚代幣。

然後建立 C 對 D 和 D 對 E 的類似承諾。當所有承諾都到位時，A 透過將解決方案 u 傳達給所有相關方——或者只傳達給 E 來觸發轉移。

此描述中省略了一些次要細節。例如，這些承諾必須有不同的到期時間，承諾的金額可能沿鏈略有不同（B 可能只向 C 承諾 $x-\epsilon$ 枚代幣，其中 ϵ 是預先約定的小額轉送費用）。我們暫時忽略這些細節，因為它們對於理解支付通道的工作原理以及如何在 TON 中實作它們不太相關。

5.2.5. 支付通道鏈內的虛擬支付通道. 現在假設 A 和 E 期望相互進行大量支付。他們可能會在區塊鏈中在它們之間建立一個新的支付通道，但這仍然會相當昂貴，因為一些資金將被鎖定在此支付通道中。另一種選擇是對每筆支付使用 **5.2.4**中描述的鏈資金轉移。然而，這將涉及大量網路活動和所有涉及的支付通道的虛擬區塊鏈中的大量交易。

另一種選擇是在支付通道網路中連接 A 到 E 的鏈內建立一個虛擬支付通道。為此，A 和 E 為他們的支付建立一個（虛擬）區塊鏈，就好像他們將在區塊鏈中建立一個支付通道一樣。然而，他們不是在區塊鏈中建立支付通道智慧合約，而是要求所有中間支付通道——那些連接 A 到 B、B 到 C 等的支付通道——在它們內部建立簡單的支付通道，綁定到 A 和 E 建立的虛擬區塊鏈（參見 **5.1.10**）。換句話說，現在每個中間支付通道內都存在根據 A 和 E 之間的最終結算轉移資金的承諾。

如果虛擬支付通道是單向的，則可以很容易地實作這樣的承諾，因為最終不平衡 δ 將是非正的，因此可以按照 **5.2.4**中描述的相同順序在中間支付通道內建立簡單的支付通道。它們的到期時間也可以以相同的方式設置。

如果虛擬支付通道是雙向的，情況會稍微複雜一些。在這種情況下，應該將根據最終結算轉移 δ 枚代幣的承諾拆分為兩個半承諾，如 **5.1.10**中所解釋的：在正向轉移 $\delta^- = \max(0, -\delta)$ 枚代幣，在反向轉移 $\delta^+ = \max(0, \delta)$ 枚代幣。這些半承諾可以在中間支付通道中獨立建立，一條半承諾鏈從 A 到 E 的方向，另一條鏈在相反方向。

5.2.6. 在閃電網路中尋找路徑. 到目前為止，有一點仍未討論：A 和 E 將如何在支付網路中找到連接它們的路徑？如果支付網路不太大，可以使用類似 OSPF 的協定：支付網路的所有節點建立一個覆蓋網路（參見 **3.3.17**），然後每個節點透過八卦協定將所有可用的連結（即，參與的支付通道）資訊傳播給其鄰居。最終，所有節點都將擁有所有參與支付網路的支付通道的完整列表，並將能夠自己找到最短路徑——例如，透過應用經修改以考慮所涉及的支付通道的「容量」（即，可以沿它們轉移的最大金額）的 Dijkstra 演算法版本。一旦找到候選路徑，就可以透過包含完整路

徑的特殊 ADNL 資料報探測它，並要求每個中間節點確認相關支付通道的存在，並根據路徑進一步轉發此資料報。之後，可以構建一條鏈，並且可以執行鏈轉移的協定（參見 **5.2.4**），或者在支付通道鏈內建立虛擬支付通道的協定（參見 **5.2.5**）。

5.2.7. 最佳化. 這裡可以進行一些最佳化。例如，只有閃電網路的中轉節點需要參與 **5.2.6**中討論的類似 OSPF 的協定。希望透過閃電網路連接的兩個「葉子」節點將相互傳達它們連接到的中轉節點列表（即，它們已建立參與支付網路的支付通道的節點）。然後可以如上文 **5.2.6**中所述檢查連接一個列表中的中轉節點到另一個列表中的中轉節點的路徑。

5.2.8. 結論. 我們已經概述了 TON 專案的區塊鏈和網路技術如何足以完成建立 *TON Payments* 的任務，這是一個用於鏈下即時資金轉移和微支付的平台。這個平台對於駐留在 TON 生態系統中的服務可能非常有用，使它們能夠在需要時和需要的地方輕鬆收取微支付。

結論

我們提出了一個可擴展的多區塊鏈架構，能夠支援大規模流行的加密貨幣和具有使用者友善介面的去中心化應用程式。

為了實現必要的可擴展性，我們提出了 *TON* 區塊鏈，一個「緊密耦合」的多區塊鏈系統（參見 **2.8.14**），採用自下而上的分片方法（參見 **2.8.12**和 **2.1.2**）。為了進一步提高潛在效能，我們引入了用於替換無效區塊的 2-區塊鏈機制（參見 **2.1.17**）和用於分片之間更快通訊的即時超立方體路由（參見 **2.4.20**）。TON 區塊鏈與現有和提議的區塊鏈專案的簡要比較（參見 **2.8**和 **2.9**）突顯了這種方法對於尋求每秒處理數百萬筆交易的系統的好處。

在第 **3**章中描述的 *TON* 網路涵蓋了所提議的多區塊鏈基礎設施的網路需求。該網路組件也可以與區塊鏈結合使用，以建立廣泛的應用程式和服務，這些應用程式和服務僅使用區塊鏈是不可能的（參見 **2.9.13**）。這些服務在第 **4**章中討論，包括 *TON DNS*，一種將人類可讀的物件識別碼翻譯為其位址的服務；*TON Storage*，一個用於儲存任意檔案的分散式平台；*TON Proxy*，一個用於匿名化網路存取和存取 TON 驅動服務的服務；以及 *TON Payments*（參見第 **5**章），一個用於在 TON 生態系統中進行即時鏈下資金轉移的平台，應用程式可以將其用於微支付。

TON 基礎設施允許專門的輕量級客戶端錢包和「ton-browser」桌面和智慧型手機應用程式，為終端使用者提供類似瀏覽器的體驗（參見 **4.3.24**），使大眾使用者可以存取加密貨幣支付以及與 TON 平台上的智慧合約和其他服務的互動。這樣的輕量級客戶端可以整合到 Telegram Messenger 客戶端中（參見 **4.3.19**），從而最終將豐富的基於區塊鏈的應用程式帶給數億使用者。

References

[1] K. Birman, *Reliable Distributed Systems: Technologies, Web Services and Applications*, Springer, 2005.

[2] V. Buterin, *Ethereum: A next-generation smart contract and decentralized application platform*, `https://github.com/ethereum/wiki/wiki/White-Paper`, 2013.

[3] M. Ben-Or, B. Kelmer, T. Rabin, *Asynchronous secure computations 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 Anonymous 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 information system based on the XOR metric*, in *IPTPS '01 revised papers 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 Programming* **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 framework*, draft 1, https://github.com/w3f/polkadot-white-paper/raw/master/PolkaDotPaper.pdf, 2016.

A TON 代幣，或 Gram

TON 區塊鏈，特別是其主鏈和基礎工作鏈的主要加密貨幣是 *TON* 代幣，也稱為 *Gram*（GRM）。它用於進行成為驗證者所需的存款；交易費用、gas 支付（即智慧合約訊息處理費用）和持久儲存支付通常也以 Gram 收取。

A.1. 細分和術語. 一個 *Gram* 細分為十億（10^9）個較小的單位，稱為 *nanogram*、*ngram* 或簡稱 *nano*。所有轉移和帳戶餘額都表示為 nano 的非負整數倍數。其他單位包括：

- 一個 *nano*、*ngram* 或 *nanogram* 是最小單位，等於 10^{-9} Gram。
- 一個 *micro* 或 *microgram* 等於一千（10^3）個 nano。
- 一個 *milli* 是一百萬（10^6）個 nano，或 Gram 的千分之一（10^{-3}）。
- 一個 *Gram* 等於十億（10^9）個 nano。
- 一個 *kilogram* 或 *kGram* 等於一千（10^3）個 Gram。
- 一個 *megagram* 或 *MGram* 等於一百萬（10^6）個 Gram，或 10^{15} 個 nano。
- 最後，一個 *gigagram* 或 *GGram* 等於十億（10^9）個 Gram，或 10^{18} 個 nano。

不需要更大的單位，因為 Gram 的初始供應量將限制在五十億（$5 \cdot 10^9$）個 Gram（即 5 Gigagram）。

A.2. 用於表示 gas 價格的較小單位. 如果需要更小的單位，將使用等於 2^{-16} nanogram 的「speck」。例如，gas 價格可能以 speck 表示。然而，要支付的實際費用，計算為 gas 價格和消耗的 gas 量的乘積，將始終向下舍入到最接近的 2^{16} speck 的倍數，並表示為整數個 nano。

A.3. 原始供應量、挖礦獎勵和通貨膨脹. Gram 的總供應量最初限制為 5 Gigagram（即五十億個 Gram 或 $5 \cdot 10^{18}$ 個 nano）。

這個供應量將非常緩慢地增長，因為驗證者挖掘新的主鏈和分片鏈區塊的獎勵會累積。這些獎勵每年約為驗證者質押的 20%（確切數字可能在未來調整），前提是驗證者勤勉地履行其職責，簽署所有區塊，從不離線，從不簽署無效區塊。透過這種方式，驗證者將有足夠的利潤投資於處理不斷增長的使用者交易數量所需的更好更快的硬體。

我們預計在任何給定時刻，平均最多有 10%[37]的 Gram 總供應量將綁定在驗證者質押中。這將產生每年 2% 的通貨膨脹率，結果將在 35 年內使 Gram 的總供應量翻倍（達到十 Gigagram）。本質上，這種通貨膨脹代表社群所有成員向驗證者支付的費用，以保持系統的運作。

另一方面，如果驗證者被發現行為不當，其質押的一部分或全部將被作為懲罰而沒收，其中較大的一部分隨後將被「燒毀」，從而減少 Gram 的總供應量。這將導致通貨緊縮。罰款的較小部分可能會重新分配給提交有罪驗證者不當行為證明的驗證者或「漁夫」。

A.4. Gram 的原始價格. 要出售的第一個 Gram 的價格將大約等於 $0.1（美元）。隨後每個要出售的 Gram（由 TON 基金會控制的 TON 儲備）的價格將比前一個高十億分之一。這樣，投入流通的第 n 個 Gram 將以大約

$$p(n) \approx 0.1 \cdot (1 + 10^{-9})^n \quad \text{美元} \tag{26}$$

的價格出售，或大約等值的其他（加密）貨幣金額，例如 BTC 或 ETH（由於市場匯率快速變化）。

A.4.1. 指數定價的加密貨幣. 我們說 Gram 是一種指數定價的加密貨幣，意思是投入流通的第 n 個 Gram 的價格大約為 $p(n)$，由公式給出

$$p(n) = p_0 \cdot e^{\alpha n} \tag{27}$$

其中特定值為 $p_0 = 0.1$ 美元和 $\alpha = 10^{-9}$。

更準確地說，一旦 n 個代幣投入流通，新代幣的一小部分 dn 價值 $p(n)\,dn$ 美元。（這裡 n 不一定是整數。）

這種加密貨幣的其他重要參數包括 n，流通中代幣的總數，以及 $N \geq n$，可以存在的代幣總數。對於 Gram，$N = 5 \cdot 10^9$。

A.4.2. 前 n 個代幣的總價格. 要投入流通的指數定價加密貨幣（例如 Gram）的前 n 個代幣的總價格 $T(n) = \int_0^n p(n)\,dn \approx p(0) + p(1) + \cdots + p(n-1)$ 可以透過以下公式計算

$$T(n) = p_0 \cdot \alpha^{-1}(e^{\alpha n} - 1) \quad . \tag{28}$$

A.4.3. 接下來 Δn 個代幣的總價格. 在 n 個先前存在的代幣之後投入流通的 Δn 個代幣的總價格 $T(n + \Delta n) - T(n)$ 可以透過以下公式計算

$$T(n+\Delta n) - T(n) = p_0 \cdot \alpha^{-1}(e^{\alpha(n+\Delta n)} - e^{\alpha n}) = p(n) \cdot \alpha^{-1}(e^{\alpha\,\Delta n} - 1) \quad . \tag{29}$$

[37]驗證者質押的最大總額是區塊鏈的可設定參數，因此如果需要，該限制可以由協定強制執行。

A.4.4. 用總值 T 購買接下來的代幣. 假設已經有 n 個代幣投入流通，並且想要花費 T（美元）購買新代幣。新獲得的代幣數量 Δn 可以透過將 $T(n+\Delta n)-T(n)=T$ 代入 (29) 來計算，得到

$$\Delta n = \alpha^{-1}\log\left(1+\frac{T\cdot\alpha}{p(n)}\right) \quad . \tag{30}$$

當然，如果 $T \lll p(n)\alpha^{-1}$，則 $\Delta n \approx T/p(n)$。

A.4.5. Gram 的市場價格. 當然，如果一旦 n 個 Gram 投入流通，自由市場價格低於 $p(n) := 0.1\cdot(1+10^{-9})^n$，則沒有人會從 TON 儲備購買新的 Gram；他們會選擇在自由市場上購買 Gram，而不增加流通中 Gram 的總量。另一方面，Gram 的市場價格不能比 $p(n)$ 高得多，否則從 TON 儲備獲得新的 Gram 將是有意義的。這意味著 Gram 的市場價格不會受到突然飆升（和下跌）的影響；這很重要，因為質押（驗證者存款）至少凍結一個月，gas 價格也不能變化太快。因此，系統的整體經濟穩定性需要某種機制來防止 Gram 匯率變化過於劇烈，例如上述機制。

A.4.6. 回購 Gram. 如果當總共有 n 個 Gram 流通時（即，不保存在由 TON 儲備控制的特殊帳戶中），Gram 的市場價格低於 $0.5\cdot p(n)$，TON 儲備保留回購一些 Gram 並減少流通中 Gram 總量 n 的權利。這可能是防止 Gram 匯率突然下跌所必需的。

A.4.7. 以更高價格出售新的 Gram. TON 儲備將根據價格公式 (26)僅出售 Gram 總供應量的一半（即 $2.5\cdot 10^9$ 個 Gram）。它保留根本不出售任何剩餘 Gram 的權利，或者以高於 $p(n)$ 的價格出售它們，但絕不會以更低的價格出售（考慮到快速變化的匯率的不確定性）。這裡的理由是，一旦至少一半的 Gram 已經售出，Gram 市場的總價值將足夠高，外部力量操縱匯率將比 Gram 部署初期更困難。

A.5. 使用未分配的 Gram. TON 儲備將僅使用大部分「未分配」的 Gram（大約 $5\cdot 10^9 - n$ 個 Gram）——即駐留在 TON 儲備的特殊帳戶和一些其他明確連結到它的帳戶中的那些——作為驗證者質押（因為 TON 基金會本身在 TON 區塊鏈的第一個部署階段可能不得不提供大部分驗證者），以及在主鏈中投票支援或反對有關「可設定參數」變更和其他協定變更的提案，以 TON 基金會（即其創建者——開發團隊）確定的方式。這也意味著 TON 基金會將在 TON 區塊鏈的第一個部署階段擁有多數投票權，如果需要調整大量參數，或者需要進行硬分叉或軟分叉，這可能很有用。後來，當所有 Gram 的不到一半仍在 TON 基金會的控制之下時，系統將變得更加民主。希望到那時它將變得更加成熟，無需頻繁調整參數。

A.5.1. 一些未分配的 Gram 將給予開發者. 在 TON 區塊鏈部署期間，預定義的（相對較小的）「未分配」Gram 數量（例如，200 Megagram，等於總供應量的 4%）將轉移到由 TON 基金會控制的特殊帳戶，然後可以從此帳戶向 TON 開源軟體的開發者支付一些「獎勵」，最短兩年的歸屬期。

A.5.2. TON 基金會需要 Gram 用於營運目的. 回想一下，TON 基金會將收到透過從 TON 儲備出售 Gram 獲得的法幣和加密貨幣，並將它們用於 TON 專案的開發和部署。例如，原始驗證者集以及 TON Storage 和 TON Proxy 節點的初始集可能由 TON 基金會安裝。

雖然這對於專案的快速啟動是必要的，但最終目標是使專案盡可能去中心化。為此，TON 基金會可能需要鼓勵安裝第三方驗證者和 TON Storage 和 TON Proxy 節點——例如，透過向它們支付儲存 TON 區塊鏈的舊區塊或代理所選服務子集的網路流量的費用。這些支付將以 Gram 進行；因此，TON 基金會將需要大量 Gram 用於營運目的。

A.5.3. 從儲備中取出預先安排的金額. TON 基金會將在 Gram 初始銷售結束後將 TON 儲備的一小部分——比如所有代幣的 10%（即 500 Megagram）——轉移到其帳戶，用於 **A.5.2**中概述的自己的目的。這最好與如 **A.5.1**中提到的為 TON 開發者預留的資金轉移同時進行。

轉移給 TON 基金會和 TON 開發者後，Gram 的 TON 儲備價格 $p(n)$ 將立即上漲一定數量，提前已知。例如，如果將 10% 的所有代幣轉移給 TON 基金會，並將 4% 轉移給開發者以示鼓勵，則流通中代幣的總量 n 將立即增加 $\Delta n = 7 \cdot 10^8$，Gram 的價格乘以 $e^{\alpha \Delta n} = e^{0.7} \approx 2$（即翻倍）。

剩餘的「未分配」Gram 將按照上文 **A.5**中的解釋由 TON 儲備使用。如果 TON 基金會此後需要更多 Gram，它將簡單地在自由市場上或透過從 TON 儲備購買 Gram 將其在代幣銷售期間先前獲得的一些資金轉換為 Gram。為了防止過度集中化，TON 基金會將永遠不會努力在其帳戶上擁有超過 Gram 總量的 10%（即 500 Megagram）。

A.6. Gram 的批量銷售. 當很多人同時想要從 TON 儲備購買大量 Gram 時，不立即處理他們的訂單是有意義的，因為這將導致結果非常依賴於特定訂單的時間和處理順序。

相反，購買 Gram 的訂單可以在一些預定義的時間段（例如，一天或一個月）內收集，然後一次性一起處理。如果 k 個訂單到達，其中第 i 個訂單價值 T_i 美元，則總金額 $T = T_1 + T_2 + \cdots + T_k$ 用於根據 (30) 購買 Δn 個新代幣，第 i 個訂單的發送者分配到這些代幣中的 $\Delta n \cdot T_i/T$。透過這種方式，所有買家以相同的平均價格每 Gram $T/\Delta n$ 美元獲得他們的 Gram。

之後，開始新一輪收集購買新 Gram 的訂單。

當購買 Gram 訂單的總價值變得足夠低時，這個「批量銷售」系統可能會被根據公式 (30)從 TON 儲備立即銷售 Gram 的系統所取代。

「批量銷售」機制可能會在收集 TON 專案投資的初始階段廣泛使用。