Telegram Open Network

Nikolai Durov
TL: Dr Awesome Doge

March 16, 2023

Abstract

此文旨在初步描述 Telegram Open Network (TON) 及其相關的區塊鏈、點對點、分散式儲存和服務托管技術。為了使本文篇幅縮小至合理範圍，我們主要著眼於 TON 平台的獨特和定義性功能，這些功能對於實現其所述目標至關重要。

簡介

Telegram Open Network (TON) 是一個快速、安全且可擴展的區塊鏈和網路計畫，如果必要，能夠處理數百萬筆交易，而且使用者和服務提供者都能夠輕鬆上手。我們的目標是讓它能夠托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，或者更準確地說是一個巨大的「超級伺服器」，旨在托管和提供各種服務。

本文不旨在成為關於所有實施細節的最終參考資料。在開發和測試階段，某些細節可能會發生變化。

Contents

1 TON 元件的簡要說明

Telegram Open Network (TON) 由以下組件組成：

- 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 Turing 完備的智能合約、可升級的正式區塊鏈規格、多加密貨幣價值轉移、支持微支付通道和離線支付網路等。「TON 區塊鏈」具有一些新的獨特功能，例如「自我修復」的垂直區塊鏈機制（參見**2.1.17**節）和即時超立方體路由（參見**2.4.20**節），使其能夠在同一時間快速、可靠、可擴展和自我一致。

- 一個點對點網路（「TON P2P 網路」或「TON 網路」；參見第 **3**章），用於訪問 TON 區塊鏈，發送交易候選項，並接收關於客戶端感興趣的區塊鏈部分（例如與客戶端帳戶和智能合約相關的部分）的更新，但也能夠支持任意分佈式服務，與區塊鏈有關或不相關。

- 一種分佈式文件存儲技術（「TON 存儲」；參見**4.1.8**節），通過「TON 網路」訪問，由 TON 區塊鏈用於存儲區塊和狀態數據（快照）的存檔副本，但也可用於存儲平台上運行的用戶或其他服務的任意文件，具有類似 Torrent 的訪問技術。

- 一個網路代理/匿名層（「TON 代理」；參見**4.1.11**和**3.1.6**節），類似於 I^2P（隱形網路計劃），用於必要時隱藏「TON 網路」節點的身份和 IP 地址（例如，使用大量加密貨幣的帳戶進行交易的節點，或希望隱藏其精確 IP 地址和地理位置作為對抗 DDoS 攻擊的高利益區塊鏈驗證者節點）。

- 一個類似 Kademlia 的分佈式哈希表（「TON DHT」；參見第 **3.2**章），用作「torrent 跟踪器」（參見**3.2.10**節）和「輸入隧道定位器」（參見**3.2.14**節）的 *TON* 存儲，以及 *TON* 服務的服務定位器（參見**3.2.12**節）。

- 一個任意服務平台（「TON 服務」；參見第 **4**章），駐留於並通過「TON 網路」和「TON 代理」提供，具有形式化的接口（參見**4.3.14**節），可實現類似於瀏覽器或智能手機應用程序的交互。這些形式化的接口和持久的服務入口可以在 TON 區塊鏈上發布（參見**4.3.17**節）；在任何給定時刻提供服務的實際節點可以從 TON 區塊鏈上發布的信息開始通過 *TON DHT* 查找（參見**3.2.12**節）。服務可以在 TON 區塊鏈上創建智能合約，向客戶提供一些保證（參見**4.1.7**節）。

- *TON DNS/*（參見**4.3.1**節），一個為帳戶、智能合約、服務和網路節點分配易於閱讀的名稱的服務。

- *TON* 支付（參見第 **5**章），一個微支付、微支付通道和微支付通道網路的平台。它可用於快速的離線價值轉移，以及支付由 *TON* 服務提供動力的服務。

- TON 將允許與第三方消息和社交網路應用程序輕鬆集成，從而使區塊鏈技術和分佈式服務最終可用並且對普通用戶可及（參見**4.3.24**節），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項目 Telegram Messenger 中提供這樣的集成示例（參見**4.3.19**節）。

雖然 TON 區塊鏈是 TON 項目的核心，其他元件可能被認為是為區塊鏈提供支援角色，但它們本身也具有有用和有趣的功能。結合起來，它們使得平台可以承載比僅使用 TON 區塊鏈更多樣化的應用程序（參見**2.9.13**和**4.1**節）。

2 TON 區塊鏈

我們從 Telegram Open Network（TON）區塊鏈的描述開始，這是該項目的核心組件。我們在這裡的方法是「自上而下」：我們首先給出一個整體的描述，然後對每個組件進行更詳細的說明。

為了簡化起見，我們在這裡談論「TON 區塊鏈」，即使在原則上，這個區塊鏈協議的多個實例可以獨立運行（例如，作為硬分叉的結果）。我們只考慮其中一個。

2.1 TON 區塊鏈作為 2-區塊鏈集合

TON 區塊鏈實際上是一個集合的區塊鏈（甚至是區塊鏈的區塊鏈，或 *2*-區塊鏈，這一點稍後將在**2.1.17**中進一步闡明），因為沒有單一的區塊鏈項目能夠實現我們的目標，即處理每秒數百萬筆交易，而不是現在的標準數十筆交易每秒。

2.1.1. List of blockchain types. The blockchains in this collection are:

- 唯一的主區塊鏈或主鏈，包含有關協議的一般信息和其當前參數值，驗證者及其賭注的集合，當前活動的工作鏈和它們的「分片」的集合，以及最重要的是，所有工作鏈和分片鏈最近的區塊的哈希集合。

- 數個（最多 2^{32} 個）工作區塊鏈或工作鏈，實際上是「工作馬匹」，包含價值轉移和智能合約交易。不同的工作鏈可能具有不同的「規則」，即不同的賬戶地址格式、交易格式、用於智能合約的不同虛擬機（VM）以及不同的基本加密貨幣等等。但是，它們都必須滿足某些基本的互操作性標準，以使不同工作鏈之間的交互作用可能且相對簡單。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**）項目。

- 每個工作鏈又被分為最多 2^{60} 個分片區塊鏈或分片鏈，具有與工作鏈本身相同的規則和區塊格式，但僅負責賬戶的一個子集，具體取決於賬戶地址的前幾位（最重要的幾位）。換句話說，一種分片形式被建立到系統中（參見**2.8.12**）。因為所有這些分片鏈共享一個常見的區塊格式和規則，所以在這方面，TON 區塊鏈是同質的（參見**2.8.8**），類似於 Ethereum 的一個擴展提案中討論的情況。[1]

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

- 每個分片鏈中的每個區塊（以及主鏈中的每個區塊）實際上不僅僅是一個區塊，而是一個小的區塊鏈。通常，這個「區塊鏈區塊」或「垂直區塊鏈」只包含一個區塊，那麼我們可能會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，則會提交一個新的區塊到「垂直區塊鏈」中，其中包含無效的「水平區塊鏈」區塊的替換或「區塊差異」，僅包含需要更改的前一版本中的某些部分的描述。這是一種 TON 特有的機制，用於替換檢測到的無效區塊，而不需要對涉及的所有分片鏈進行真正的分叉；它將在**2.1.17**中詳細解釋。現在，我們只是注意到，每個分片鏈（以及主鏈）不是傳統的區塊鏈，而是一個區塊鏈的區塊鏈，或 *2D*-區塊鏈，或只是一個 *2*-區塊鏈。

2.1.2. Infinite Sharding Paradigm. 幾乎所有的區塊鏈分片提案都是「自上而下」的：首先想象一個單一的區塊鏈，然後討論如何將它分割成多個相互交互的分片鏈以提高性能並實現可擴展性。

TON 的方法針對分片是「自下而上」的，其解釋如下。

假設分片被極端地使用，以便每個分片鏈中只保留一個帳戶或智能合約。然後，我們將擁有大量的「帳戶鏈」，每個帳戶鏈僅描述一個帳戶的狀態和狀態轉換，並相互發送攜帶價值的消息以傳輸價值和信息。

當然，擁有數億個區塊鏈是不切實際的，而且通常在它們中的更新（即新區塊）出現得相當少。為了更有效地實現它們，我們將這些「帳戶鏈」分組為「分片鏈」，以便每個分片鏈的每個區塊實質上是分配給此分片的帳戶鏈區塊的集合。因此，「帳戶鏈」只在「分片鏈」中具有純虛擬或邏輯存在。

We call this perspective the *Infinite Sharding Paradigm*. It explains many of the design decisions for the TON Blockchain.

2.1.3. Messages. Instant Hypercube Routing. 無限分片范式告訴我們，應將每個帳戶（或智能合約）視為獨立的分片鏈。因此，一個帳戶可能影響另一個帳戶的狀態的唯一方法是向其發送一個消息（這是所謂的 Actor 模型的一個特殊實例，其中帳戶作為 Actor；參見**2.4.2**）。因此，帳戶之間（以及分片鏈之間，因為源帳戶和目標帳戶通常位於不同的分片鏈中）的消息系統對於像 TON 區塊鏈這樣的可擴展系統至關重要。實際上，TON 區塊鏈的一個新功能，稱為瞬時超立方體路由（參見**2.4.20**），使其能夠在源分片鏈中創建的消息被傳遞到目標分片鏈的下一個區塊中進行處理，無論系統中分片鏈的總數為何。

2.1.4. Quantity of masterchains, workchains and shardchains. 一個 TON 區塊鏈恰好包含一個主鏈。然而，系統理論上可以容納多達 2^{32} 個工作鏈，每個工作鏈又可以細分為多達 2^{60} 個分片鏈。

2.1.5. Workchains can be virtual blockchains, not true blockchains. 由於工作鏈通常被細分為分片鏈，因此工作鏈的存在是“虛擬的”，這意味著它不是傳統定義中所說的真正區塊鏈（參見下文的**2.2.1**），而僅僅是一個分片鏈的集合。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以被視為該工作鏈，此時該工作鏈至少在一段時間內成為了一個“真正”的區塊鏈，從而在表面上類似於傳統的單一區塊鏈設計。然而，無限分片范式（參見**2.1.2**）告訴我們，這種相似性實際上是表面的：大量的“賬戶鏈”暫時可以分組成為一個區塊鏈純粹只是巧合。

2.1.6. Identification of workchains. 每個工作鏈都由其“編號”或“工作鏈標識符”（$workchain\_id : uint_{32}$）來標識，這只是一個無符號 32 位整數。工作鏈是通過主鏈中的特殊交易創建的，該交易定義了（之前未使用的）工作鏈標識符和工作鏈的正式描述，至少足以讓此工作鏈與其他工作鏈進行交互並對此工作鏈的區塊進行表面驗證。

2.1.7. Creation and activation of new workchains. 新工作鏈的創建可以由社區中的任何成員發起，只要準備支付（高昂的）主鏈交易費用以發布新工作鏈的正式規範。然而，為了使新工作鏈變得有效，需要兩個第三的驗證者共識，因為他們需要升級他們的軟件以處理新工作鏈的區塊，並通過特殊的主鏈交易表明他們準備好與新工作鏈一起工作。有興趣啟用新工作鏈的一方可能會通過某些由智能合約分配的獎勵來提供激勵，以鼓勵驗證者支持新工作鏈。

2.1.8. Identification of shardchains. 每個分片鏈都由一對 $(w, s) = (workchain\_id, shard\_prefix)$ 標識，其中 $workchain\_id : uint_{32}$ 標識相應的工作鏈，$shard\_prefix : \mathbf{2}^{0...60}$ 是一個長度最多為 60 的比特字符串，定義了該分片鏈負責管理的帳戶子集。具體而言，所有 $account\_id$ 以 $shard\_prefix$ 開頭（即具有 $shard\_prefix$ 作為最高有效位的帳戶）的帳戶都將被指定給此分片鏈。

2.1.9. Identification of account-chains. 回想一下，帳戶鏈只有虛擬存在（參見 **2.1.2**）。但是，它們有一個自然的標識符——即 $(workchain\_id, account\_id)$，因為任何帳戶鏈都包含有關正好一個帳戶（無論是簡單帳戶還是智能合約——區別在這裡並不重要）的狀態和更新信息。

2.1.10. Dynamic splitting and merging of shardchains; cf. 2.7. 一個不太複雜的系統可能使用靜態分片——例如，通過使用 $account\_id$ 的前 8 位中的高位來選擇 256 個預定義分片之一。

TON Blockchain 的一個重要特性是實現了動態分片，這意味著分片的數量並非固定的。相反，如果滿足某些形式條件（基本上是如果原始分片上

的交易負載足夠長時間高），則可以自動將分片 (w, s) 細分為分片 $(w, s.0)$ 和 $(w, s.1)$。反之，如果負載在某段時間內保持太低，則可以自動將分片 $(w, s.0)$ 和 $(w, s.1)$ 合併回分片 (w, s)。

最初，為工作鏈 w 創建了一個分片 $(w, \emptyset)$。如果需要的話（參見**2.7.6**和**2.7.8**），它會被細分為更多的分片。

2.1.11. Basic workchain or Workchain Zero. 雖然最多可以定義 2^{32} 個工作鏈及其特定規則和交易，但我們最初僅定義一個 $workchain\_id = 0$。這個工作鏈稱為工作鏈零或基本工作鏈，用於處理 *TON* 智能合約和轉移 *TON* 幣，也稱為 *Grams*（參見附錄 A）。大多數應用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. Block generation intervals. 我們預計每個分片鏈和主分片鏈都會大約每五秒生成一個新區塊。這將導致合理的交易確認時間。所有分片鏈的新區塊大約同時生成；主分片鏈的新區塊稍晚約一秒鐘生成，因為它必須包含所有分片鏈最新區塊的哈希值。

2.1.13. Using the masterchain to make workchains and shardchains tightly coupled. 一旦分片鏈區塊的哈希值被納入主分片鏈區塊中，該分片鏈區塊及其所有祖先都被視為「正確的」，這意味著它們可以被後續所有分片鏈的區塊作為固定且不可變的參考。實際上，每個新的分片鏈區塊包含最近的主分片鏈區塊的哈希值，而從該主分片鏈區塊引用的所有分片鏈區塊都被新區塊視為不可變。

基本上，這意味著提交到分片鏈區塊中的交易或消息可以安全地在其他分片鏈的下一個區塊中使用，而不需要等待，例如，在轉發消息或根據先前的交易採取其他操作之前等待二十次確認（即，在同一個區塊鏈中原始區塊之後生成的二十個區塊）。這在大多數提議的「鬆散耦合」系統中是常見的（例如 EOS），這能夠在提交後僅僅五秒就在其他分片鏈中使用交易和消息，這是我們相信我們的「緊密耦合」系統能夠提供前所未有的性能的原因之一（參見 **2.8.12**和 **2.8.14**）。

2.1.14. Masterchain block hash as a global state. 根據 **2.1.13**，從外部觀察者的角度來看，最後一個主分片鏈塊的哈希完全確定了系統的整體狀態。沒有必要單獨監控所有分片鏈的狀態。

2.1.15. Generation of new blocks by validators; cf. 2.6. TON Blockchain 使用 Proof-of-Stake（PoS）方法在分片鏈和主分片鏈中生成新區塊。這意味著存在一個包含最多幾百個驗證者的集合，這些驗證者是特殊節點，通過一個特殊的主分片鏈交易存入了賭注（大量的 TON 幣），以符合新區塊生成和驗證的資格要求。

然後，以一種確定性偽隨機的方式，每個分片 (w, s) 分配給一個較小的驗證者子集，每 1024 個區塊左右會進行更改。這個驗證者子集收集來自客戶端的合適提議交易，建議並達成共識，以確定下一個分片鏈區塊。對於每個區塊，都會有一個偽隨機選擇的順序，以確定哪個驗證者的區塊候選人在每次輪到時擁有最高優先級可提交。

驗證者和其他節點檢查提議的區塊候選人的有效性；如果驗證者簽署了一個無效的區塊候選人，可能會自動懲罰，失去部分或全部的股份，或被暫停一段時間的驗證者集合。之後，驗證者應就下一個區塊的選擇達成共識，基本上是通過一種高效的 BFT（拜占庭容錯）共識協議的變體，類似於 PBFT [4] 或 Honey Badger BFT [11]。如果達成共識，則創建新的區塊，驗證者之間分配其中包含的交易費用，以及一些新創造的（“鑄造的”）代幣。

每個驗證者可以被選為參與多個驗證者子集; 在這種情況下，它應該並行運行所有驗證和共識算法。

在生成所有新的分片鏈塊或超時後，將生成一個新的主鏈塊，其中包括所有分片鏈的最新塊的哈希。這是通過所有驗證者的 BFT 共識來完成的。[2]關於 TON PoS 方法及其經濟模型的更多細節，請參見第**2.6**節。

2.1.16. Forks of the masterchain. 由於我們採用緊密耦合的方法，因此在主分片鏈中切換到另一個分支幾乎必然需要在至少一些分片鏈中切換到另一個分支。另一方面，只要主分片鏈中沒有分支，分片鏈中也不可能有分支，因為替代分支中的任何區塊都無法通過將其哈希值納入主分片鏈區塊而成為“規範”的區塊。

一般規則是：如果主分片鏈區塊 B' 是區塊 B 的前身，B' 包含 (w, s)-分片鏈區塊 $B'w, s$ 的哈希值 $\mathit{HASH}(B'w, s)$，而 B 包含區塊 $B_{w,s}$ 的哈希值 $\mathit{HASH}(B_{w,s})$，則 $B'w, s$ 必須是 Bw, s 的前身；否則，主分片鏈區塊 B 是無效的。

我們預計主分片鏈的分支非常罕見，甚至可以說不存在，因為在 TON 區塊鏈採用的 BFT 范式中，只有在大多數驗證者的行為不正確的情況下（參見 **2.6.1** 和 **2.6.15**），才可能發生分支，這將導致犯罪者的巨大損失。因此，不應該期望分片鏈中出現真正的分支。相反，如果檢測到無效的分片鏈區塊，將通過 2D 區塊鏈的“垂直區塊鏈”機制（參見 **2.1.17**）進行更正，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正主分片鏈區塊中的非致命性錯誤。

2.1.17. Correcting invalid shardchain blocks. 通常情況下，只有有效的分片鏈塊會被提交，因為分配給分片鏈的驗證者必須在提交新塊之前達

[2]實際上，通過三分之二的權益即可實現共識，但是會盡可能地收集更多的簽名。

成三分之二的拜占庭共識。但系統必須允許檢測之前提交的無效塊及其修正。

當然，一旦找到無效的分片鏈塊，無論是由驗證者（不一定是分配給該分片鏈的驗證者）還是由“漁夫”（系統中的任何節點，可以通過一定的存款提出有關塊有效性的問題；參見 **2.6.4**）發現，無效性索賠及其證據將被提交到主分片鏈，簽署無效塊的驗證者將因此失去一部分賭注並且可能暫時被暫停作為驗證者（對於攻擊者竊取了本來是無惡意的驗證者的私有簽署密鑰的情況來說，後一項措施是很重要的）。

然而，這並不足夠，因為由於先前提交的無效分片鏈塊，系統（TON 區塊鏈）的整體狀態是無效的。這個無效塊必須被一個新的有效版本替換。

大多數系統將通過在這個分片鏈中的無效塊之前回滾到最後一個區塊，以及在每個其他分片鏈中從未受到從無效塊傳播的消息的最後幾個區塊，從這些區塊創建一個新的分支。這種方法的缺點是，大量本來正確且已提交的交易突然被回滾，而且不清楚它們是否會在以後被包含。

TON 區塊鏈通過使每個分片鏈和主分片鏈的「區塊」（「水平區塊鏈」）本身成為一個小區塊鏈（「垂直區塊鏈」）來解決這個問題，其中包含這個「區塊」的不同版本或其「差異」。通常，垂直區塊鏈只包含一個區塊，而分片鏈看起來像一個傳統的區塊鏈。然而，一旦一個區塊的無效性被確認並提交到主分片鏈區塊中，該區塊的「垂直區塊鏈」就允許在垂直方向上增加一個新區塊，以替換或編輯無效區塊。新區塊由相應分片鏈的當前驗證者子集生成。

為了使新的「垂直」區塊有效，其規則非常嚴格。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，那麼它必須在新的垂直區塊中保持不變。

一旦在無效區塊之上提交了新的「垂直」區塊，其哈希值就會在新的主鏈區塊（或者更準確地說，在原來發布無效分片鏈區塊哈希值的主鏈區塊之上的「垂直」區塊）中公布，並且將更改進一步傳播到任何參考先前版本此區塊的分片鏈區塊上（例如，那些從錯誤區塊接收到消息的區塊）。這通過為先前參考到「不正確」區塊的所有區塊中的垂直區塊鏈提交新的「垂直」區塊來修復。新的垂直區塊將參考最近（已更正的）版本。同樣，嚴格的規則禁止更改沒有受到實際影響的帳戶鏈（即，接收到的消息與先前版本相同）。通過這種方式，修復錯誤區塊會生成「漣漪」，最終向所有受影響分片鏈的最新區塊傳播；這些更改也會反映在新的「垂直」主鏈區塊中。

一旦“歷史重寫”漣漪到達最新的區塊，新的分片鏈區塊就只有一個版本，只是最新的區塊版本的後繼者。這意味著它們將從一開始就包含對正確（最新）的垂直區塊的引用。

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哈希轉換為其最新版本的哈希。這使得客戶端可以通過其第一個（通常

是唯一的）區塊的哈希識別和定位任何垂直區塊鏈。

2.1.18. TON coins and multi-currency workchains. TON 區塊鏈支持多達 2^{32} 種不同的“加密貨幣”、“硬幣”或“代幣”，通過 32 位的 *currency_id* 加以區分。新的加密貨幣可以通過主鏈中的特殊交易來添加。每個工作鏈都有一種基本加密貨幣，並且可以有幾種其他加密貨幣。

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

原則上，其他工作鏈可能會以其他代幣收取交易費。在這種情況下，應該提供一些智能合約，用於將這些交易費自動轉換為 Grams。

2.1.19. Messaging and value transfer. 屬於同一個或不同工作鏈的分片鏈可以彼此發送「消息」。雖然允許的消息形式取決於接收工作鏈和接收帳戶（智能合約），但存在一些共同字段，使得工作鏈之間的消息傳遞成為可能。特別地，每個消息可以附帶一些「值」，以一定量的 Grams（TON 幣）和/或其他已註冊的加密貨幣的形式提供，前提是接收工作鏈將其宣佈為可接受的加密貨幣。

這種消息傳遞的最簡單形式是從一個（通常不是智能合約）帳戶向另一個帳戶進行價值轉移。

2.1.20. TON Virtual Machine. 「TON 虛擬機」，也簡稱為「TON VM」或「TVM」，是用於在主鏈和基本工作鏈中執行智能合約代碼的虛擬機。其他工作鏈可以使用 TVM 以外的其他虛擬機或與 TVM 並行使用。

以下列出 TVM 的一些特點。在第2.3.12和2.3.14節中進一步討論。

- TVM 將所有數據表示為（*TVM*）單元格的集合（參見 **2.3.14**）。每個單元格包含多達 128 個數據字節和多達 4 個對其他單元格的引用。作為“一切皆為單元格的集合”哲學的結果（參見 **2.5.14**），這使 TVM 能夠處理與 TON 區塊鏈相關的所有數據，包括必要時的區塊和區塊鏈全局狀態。
- TVM 可以處理任意代數數據類型的值（參見 **2.3.12**），表示為 TVM 單元格的樹或有向無環圖。但是，它對於代數數據類型的存在是不可知的；它只處理單元格。
- TVM 具有對哈希映射的內置支持（參見 **2.3.7**）。
- TVM 是一個堆疊機。它的堆疊可以保存 64 位整數或單元格引用。

- 支持 64 位、128 位和 256 位算術運算。所有 n 位算術運算都有三種變體：無符號整數、有符號整數和模 2^n 的整數（在後一種情況下沒有自動溢出檢查）。
- TVM 具有從 n 位到 m 位的無符號和有符號整數轉換，其中 $0 \leq m, n \leq 256$，並進行了溢出檢查。
- 所有算術運算默認執行溢出檢查，大大簡化了智能合約的開發。
- TVM 具有“乘法後移位”和“移位後除法”算術運算，中間值在更大的整數類型中計算；這簡化了實現定點算術。
- TVM 提供位串和字節串的支持。
- 支持一些預定義曲線的 256 位橢圓曲線加密（ECC），包括 Curve25519。
- 支持某些橢圓曲線上的 Weil 配對，對於快速實現 zk-SNARKs 很有用。
- 支持流行的哈希函數，包括 SHA256。
- TVM 可以使用 Merkle 證明（參見**5.1.9**）。
- TVM 提供支持“大型”或“全局”智能合約。這種智能合約必須意識到分片（參見**2.3.18**和**2.3.16**）。通常（本地）智能合約可以忽略分片。
- TVM 支持閉包。
- 在 TVM 內部可以輕鬆實現「無脊椎無標籤的 G 機」[13]。

  除了「TVM 組合語言」之外，還可以為 TVM 設計幾種高級語言。所有這些語言都將具有靜態類型並支持代數數據類型。我們構想以下可能性：

- 一種類似 Java 的命令式語言，其中每個智能合約都類似於單獨的類。
- 一種惰性的函數式語言（類似於 Haskell）。
- 一種熱切的函數式語言（類似於 ML）。

2.1.21. Configurable parameters. TON 區塊鏈的一個重要特點是，它的許多參數是可配置的。這意味著它們是主鏈狀態的一部分，可以通過主鏈中某些特殊的提案/投票/結果交易進行更改，而無需進行硬分叉。更改此類參數將需要收集三分之二的驗證者投票以及所有其他希望參與投票過程的參與者的超過一半的投票，以支持該提議。

2.2 區塊鏈的一般性質

2.2.1. General blockchain definition. 通常情況下，任何（真正的）區塊鏈都是一個包含了一系列區塊的序列，每個區塊 B 都包含了對上一個區塊的引用 $\text{BLK-PREV}(B)$（通常是通過將上一個區塊的哈希值包含在當前區塊的標頭中），以及一個交易列表。每個交易描述了全局區塊鏈狀態的某些變換；在一個區塊中列出的交易被依次應用以計算新狀態，該新狀態始於上一個區塊的評估結果。

2.2.2. Relevance for the TON Blockchain. 請注意，*TON* 區塊鏈不是真正的區塊鏈，而是 2D 區塊鏈（即，區塊鏈的區塊鏈的集合；參見 **2.1.1**），因此上述內容不適用於它。然而，我們從真正的區塊鏈的這些一般性質開始，以將它們用作更複雜構造的基礎。

2.2.3. Blockchain instance and blockchain type. 人們通常使用區塊鏈一詞來表示一般的區塊鏈類型以及其特定的區塊鏈實例，這些實例是由一系列滿足某些條件的區塊組成的。例如，**2.2.1** 中引用的是區塊鏈實例。

In this way, a blockchain type is usually a "subtype" of the type $Block^*$ of lists (i.e., finite sequences) of blocks, consisting of those sequences of blocks that satisfy certain compatibility and validity conditions:

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

A better way to define *Blockchain* would be to say that *Blockchain* is a *dependent couple type*, consisting of couples $(\mathbb{B}, v)$, with first component $\mathbb{B} : Block^*$ being of type $Block^*$ (i.e., a list of blocks), and the second component $v : isValidBc(\mathbb{B})$ being a proof or a witness of the validity of $\mathbb{B}$. In this way,

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

We use here the notation for dependent sums of types borrowed from [16].

2.2.4. Dependent type theory, Coq and TL. 請注意，我們在這裡使用的是（Martin-L"of）依賴類型理論，類似於 Coq 證明助手中使用的那種類型理論。[3]依賴類型理論的簡化版本也用於 *TL*（類型語言）中。[4]它將用於 TON 區塊鏈的形式化規範，以描述所有數據結構的序列化和塊、交易等的佈局。

實際上，依賴類型理論為證明的含義提供了有用的形式化定義，而這樣的形式化證明（或它們的序列化）在需要為某些區塊提供無效證明時可能非常有用。

2.2.5. TL, or the Type Language. 由於 TL（Type Language）將用於 TON 區塊、交易和網絡數據包的正式規範，因此值得進行簡要討論。

TL 是一種適用於描述依賴代數類型的語言，這些類型允許具有數字（自然）和類型參數。每種類型都通過多個構造器來描述。每個構造器都具有（人可讀的）標識符和一個名稱，該名稱是一個位串（默認為 32 位整數）。除此之外，構造器的定義還包含一個字段列表以及它們的類型。

一系列構造函數和類型定義被稱為 *TL* 方案。通常它們被存儲在一個或多個以 `.tl` 為後綴的文件中。

TL 方案的一個重要特點是它們確定了一種將代數類型的值（或對象）序列化和反序列化的明確方式。也就是說，當需要將一個值序列化成一個字節流時，首先序列化用於該值的構造函數的名稱，然後按遞歸方式序列化每個字段的計算序列化。

一個之前版本的 TL 的描述，適用於將任意對象序列化為 32 位整數序列，可在`https://core.telegram.org/mtproto/TL`找到。為了描述 TON 項目中使用的對象的序列化，正在開發一個名為 *TL-B* 的新版本。這個新版本可以將對象序列化為字節和位元流（不僅僅是 32 位整數），並支持將對象序列化為 TVM 單元的樹（參見 **2.3.14**）。TL-B 的描述將成為 TON 區塊鏈的正式規範的一部分。

2.2.6. Blocks and transactions as state transformation operators. 通常情況下，任何區塊鏈（類型）*Blockchain* 都有一個相關聯的全局狀態（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確定：

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

這裡 $X^{?}$ 表示 $\textsc{Maybe}\, X$，即將 Maybe 代數結構應用到類型 X 上的結果。這與我們使用 X^{*} 表示 $\textsc{List}\, X$ 的方法類似。實際上，類型 $X^{?}$ 的值可以是

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

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

類型 X 的值，也可以是一個特殊值 $\perp$，表示實際值不存在（類比空指針）。在我們的情況下，我們使用 $State^?$ 作為結果類型，而不是 $State$，因為如果從某些原始狀態調用交易可能會無效（例如試圖從一個帳戶中提取比實際存在的錢更多的錢）。

我們可能更喜歡 ev_trans' 的柯里化版本：

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

由於區塊基本上是交易列表，因此區塊評估函數

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

可以從 ev_trans 導出。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哈希值），並計算下一個區塊鏈狀態 $s' = ev\_block(B)(s) : State$，它可以是一個真正的狀態，也可以是一個特殊值 $\perp$，表示無法計算下一個狀態（例如，如果從給定的起始狀態評估區塊是無效的，例如，區塊包含試圖扣除一個空帳戶的交易。）

每個區塊鏈中的區塊 B 可以通過其序列號$\text{BLK-SEQNO}(B)$ 進行引用，從第一個區塊開始從零開始，每當轉到下一個區塊時增加一。更正式地說，

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

請注意，在出現分叉的情況下，序列號不能唯一標識一個區塊。

2.2.7. Block hashes. 每個區塊鏈中的區塊 B 可以通過其序列號$\text{BLK-SEQNO}(B)$ 進行引用，從第一個區塊開始從零開始，每當轉到下一個區塊時增加一。更正式地說，

$$\text{BLK-SEQNO}(B) = \text{BLK-SEQNO}\big(\text{BLK-PREV}(B)\big) + 1 \tag{7}$$

請注意，在出現分叉的情況下，序列號不能唯一標識一個區塊。

2.2.8. Hash assumption. 在區塊鏈算法的形式化分析中，我們假設所使用的 k 位哈希函數 $\text{HASH} : Bytes^* \to \mathbf{2}^k$ 不會出現碰撞：

$$\text{HASH}(s) = \text{HASH}(s') \Rightarrow s = s' \quad \text{對於任何 } s,\ s' \in Bytes^* \text{ 成立} \tag{8}$$

這裡，$Bytes = 0 \ldots 255 = \mathbf{2}^8$ 表示字節類型或所有字節值的集合，而 $Bytes^*$ 表示任意（有限）字節列表的類型或集合。同時，$\mathbf{2} = 0, 1$ 表示位元類型，而 $\mathbf{2}^k$ 表示所有 k 位序列（即 k 位數字）的集合（或實際上是類型）。

當然，從數學上講，(8)是不可能的，因為從一個無限集合到一個有限集合的映射不能是單射的。一個更嚴謹的假設是

$$\forall s, s' : s \neq s', P\big(\textsc{Hash}(s) = \textsc{Hash}(s')\big) = 2^{-k} \tag{9}$$

然而，這對於證明來說並不方便。如果在證明中最多使用(9) N 次，且 $2^{-k}N < \epsilon$，對於某些小的 ϵ（例如，$\epsilon = 10^{-18}$），我們可以假設(8)成立，前提是我們接受一個失敗概率 ϵ（即，最終結論至少有 $1 - \epsilon$ 的概率是正確的）。

最後一句話：為了使式(9)的概率語句真正嚴格，必須在所有字節序列的集合 $Bytes^*$ 上引入一個概率分布。一種方法是假定所有長度為 l 的字節序列具有相同的等概率性，並將觀察到長度為 l 的序列的概率設置為 $p^l - p^{l+1}$，其中 $p \to 1-$。然後，當 p 從下方趨近於 1 時，應該將式(9)理解為條件概率 $P\big(\textsc{Hash}(s) = \textsc{Hash}(s') | s \neq s'\big)$ 的極限。

2.2.9. Hash used for the TON Blockchain. 目前，我們在 TON 區塊鏈中使用 256 位的 sha256 哈希。如果它比預期的要脆弱，則可以在未來將其替換為另一個哈希函數。哈希函數的選擇是協議的可配置參數，因此可以按照 **2.1.21**中的說明在不進行硬分叉的情況下進行更改。

2.3 Blockchain State, Accounts and Hashmaps

如前所述，任何區塊鏈都定義了某種全局狀態，每個區塊和每個交易都定義了這個全局狀態的轉換。在這裡，我們描述了 TON 區塊鏈使用的全局狀態。

2.3.1. Account IDs. TON 區塊鏈使用的基本帳戶 ID，至少包括其主鏈和 Workchain Zero 使用的帳戶 ID，是 256 位整數，假定是特定橢圓曲線的 256 位橢圓曲線加密（ECC）的公鑰。這樣，可以表示為：

$$account\_id : Account = uint_{256} = \mathbf{2}^{256} \tag{10}$$

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

其他工作鏈可以使用其他帳戶 ID 格式，256 位或其他。例如，可以使用等於 ECC 公鑰的 sha256 的比特幣式帳戶 ID。

然而，在工作鏈創建期間（在主鏈上），帳戶 ID 的位長 l 必須固定，並且必須至少為 64 位，因為 $account\_id$ 的前 64 位用於分片和消息路由。

2.3.2. Main component: *Hashmaps*. TON 區塊鏈狀態的主要組件是哈希映射。在某些情況下，我們會考慮到（部分定義的）“映射” $h : \mathbf{2}^n \dashrightarrow \mathbf{2}^m$。更一般地，我們可能會對復合類型 X 的哈希映射 $h : \mathbf{2}^n \dashrightarrow X$ 感興趣。然而，源（或索引）類型幾乎始終是 $\mathbf{2}^n$。

有時，我們會有一個“默認值” $empty : X$，並且哈希映射 $h : \mathbf{2}^n \to X$ 通過其“默認值” $i \mapsto empty$ 進行“初始化”。

2.3.3. Example: TON account balances. 一個重要的例子是 TON 帳戶餘額。它是一個哈希映射

$$balance : Account \to uint_{128} \tag{11}$$

將 $Account = \mathbf{2}^{256}$ 映射為類型為 $uint_{128} = \mathbf{2}^{128}$ 的 Gram（TON 幣）餘額。這個哈希映射的默認值為零，這意味著在初始狀態下（在處理第一個區塊之前），所有帳戶的餘額都為零。

2.3.4. Example: smart-contract persistent storage. 另一個例子是智能合約的持久化存儲，它可以（非常粗略地）表示為哈希映射

$$storage : \mathbf{2}^{256} \dashrightarrow \mathbf{2}^{256} \tag{12}$$

這個哈希映射也有一個默認值為零，這意味著持久化存儲的未初始化單元被假定為零。

2.3.5. Example: persistent storage of all smart contracts. 因為我們有超過一個智能合約，由 $account\_id$ 區分，每個智能合約都有自己的獨立持久化存儲，所以我們必須實際上有一個哈希映射

$$Storage : Account \dashrightarrow (\mathbf{2}^{256} \dashrightarrow \mathbf{2}^{256}) \tag{13}$$

將智能合約的 $account\_id$ 映射到它的持久化存儲。

2.3.6. Hashmap type. 這個哈希映射不僅僅是一個抽象（部分定義）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哈希映射類型

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

對應於編碼（部分）映射 $\mathbf{2}^n \dashrightarrow X$ 的數據結構。我們還可以寫成：

$$Hashmap(n : nat)(X : Type) : Type \tag{15}$$

或者

$$Hashmap : nat \to Type \to Type \tag{16}$$

我們可以總是將 $h : \mathit{Hashmap}(n, X)$ 轉換為一個映射 $\mathit{hget}(h) : \mathbf{2}^n \to X^?$。因此，我們通常寫 $h[i]$ 代替 $\mathit{hget}(h)(i)$：

$$h[i] :\equiv \mathit{hget}(h)(i) : X^? \quad \text{對於任何 } i : \mathbf{2}^n,\ h : \mathit{Hashmap}(n, X) \tag{17}$$

2.3.7. Definition of hashmap type as a Patricia tree. 4 / 4

在邏輯上，可以將 $\mathit{Hashmap}(n, X)$ 定義為深度為 n、具有邊緣標籤 0 和 1 的（不完整的）二叉樹，並且在葉子上具有類型 X 的值。另一種描述相同結構的方法是將其描述為長度為 n 的二進制字符串的（位）*trie*。

在實際應用中，我們更喜歡使用壓縮 Trie 的緊湊表示形式，即將只有一個子節點的節點與其父節點壓縮在一起。得到的表示形式稱為 *Patricia* 樹或二進制基數樹。現在，每個中間節點都有恰好兩個子節點，由兩個非空的二進制字符串標記，左子節點以零開頭，右子節點以一開頭。

換句話說，在 Patricia 樹中，有兩種（非根）節點類型：

- $\textsc{Leaf}(x)$，包含類型為 X 的值 x。
- $\textsc{Node}(l, s_l, r, s_r)$，其中 l 是左子樹或子樹的引用，s_l 是標記連接此節點與其左子節點的邊緣的位字符串（始終以 0 開始），r 是右子樹，而 s_r 是標記連接此節點與其右子節點的邊緣的位字符串（始終以 1 開始）。

第三種節點類型，僅在 Patricia 樹的根處使用：

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

如果我們希望允許 Patricia 樹為空，則將使用第四種（根）節點：

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

We define the height of a Patricia tree by

$$\textsc{height}\big(\textsc{Leaf}(x)\big) = 0 \tag{18}$$

$$\textsc{height}\big(\textsc{Node}(l, s_l, r, s_r)\big) = \textsc{height}(l) + \textsc{len}(s_l) = \textsc{height}(r) + \textsc{len}(s_r) \tag{19}$$

$$\textsc{height}\big(\textsc{Root}(n, s_0, t)\big) = \textsc{len}(s_0) + \textsc{height}(t) = n \tag{20}$$

最後兩個公式中的最後兩個表達式必須相等。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 個葉子節點（即我們的哈希映射包含 N 個值），那麼就會有 $N-1$ 個中間節點。插入新值總是涉及到通過在中間插入一個新節點來分裂現有的邊緣，並將一個新葉子節點添加為此新節點的另一個子節點。從哈希映射中刪除一個值則相反：一個葉子節點及其父節點被刪除，父節點的父節點及其另一個子節點直接相連。

2.3.8. Merkle-Patricia trees. 在處理區塊鏈時，我們希望能夠通過將 Patricia 樹（即哈希映射）及其子樹縮減為單個哈希值來進行比較。實現這一目標的傳統方法是 Merkle 樹。基本上，我們希望能夠通過一個哈希函數 Hash 對二進制字符串進行定義，來哈希類型為 $\mathit{Hashmap}(n, X)$ 的對象 h，前提是我們知道如何計算對象 $x : X$ 的哈希值 $\textsc{Hash}(x)$（例如，通過將對象 x 的二進制序列化應用哈希函數 Hash）。

我們可以遞歸地定義 $\textsc{Hash}(h)$，如下所示：

$$\textsc{Hash}\big(\textsc{Leaf}(x)\big) := \textsc{Hash}(x) \tag{21}$$

$$\textsc{Hash}\big(\textsc{Node}(l, s_l, r, s_r)\big) := \textsc{Hash}\big(\textsc{Hash}(l).\,\textsc{Hash}(r).\,\textsc{code}(s_l).\,\textsc{code}(s_r)\big) \tag{22}$$

$$\textsc{Hash}\big(\textsc{Root}(n, s_0, t)\big) := \textsc{Hash}\big(\textsc{code}(n).\,\textsc{code}(s_0).\,\textsc{Hash}(t)\big) \tag{23}$$

其中，$s.t$ 表示（位）字符串 s 和 t 的連接，$\textsc{code}(s)$ 是所有位字符串 s 的前綴碼。例如，可以通過用 10 編碼 0，用 11 編碼 1，並用 0 編碼字符串的末尾來對其進行編碼。
[5]

稍後我們會看到（參見**2.3.12**和**2.3.14**），這是一種遞歸定義的哈希值，用於任意（依賴）代數類型的值，稍加改進即可。

2.3.9. Recomputing Merkle tree hashes. 這種遞歸定義 $\textsc{Hash}(h)$ 的方法稱為 *Merkle* 樹哈希，其優點在於，如果將 $\textsc{Hash}(h')$ 與每個節點 h' 明確存儲在一起（得到一個稱為 *Merkle* 樹或在我們的情況下是 *Merkle-Patricia* 樹的結構），當向哈希映射添加、刪除或更改元素時，最多只需重新計算 n 個哈希值。

通過這種方式，如果用適當的 Merkle 樹哈希來表示全局區塊鏈狀態，則可以在每次交易後輕鬆地重新計算此狀態哈希值。

[5]可以證明，這種編碼對於帶有隨機或連續索引的 Patricia 樹的大約一半的邊標籤是最優的。其餘的邊標籤可能會很長（即，幾乎有 256 位）。因此，對於邊標籤來說，一種接近最優的編碼方法是使用上述編碼，並為“短”位字符串使用前綴 0 進行編碼，並為“長”位字符串（其中 $l \geq 10$）編碼 1，然後是包含長度 $l = |s|$ 的 9 位的 l 位 s 的位。

2.3.10. Merkle proofs. 在假設選擇的哈希函數 Hash 滿足式(8)的「單射性」的情況下，可以構造證明，對於給定的 $\text{Hash}(h)$ 值 $z, h : \mathit{Hashmap}(n, X)$，存在某些 $i : \mathbf{2}^n$ 和 $x : X$，滿足 $\mathit{hget}(h)(i) = x$。這樣的證明將由從對應於 i 的葉子到根的 Merkle-Patricia 樹上的所有節點的兄弟節點的哈希值構成的路徑增強而成。

可以這樣說，輕節點[6] 知道某些哈希映射 h（例如，智能合約持久存儲或全局區塊鏈狀態）的 $\text{Hash}(h)$ 的值，可能會從完整節點中請求不僅值 $x = h[i] = \mathit{hget}(h)(i)$，而是這樣的值連同從已知值 $\text{Hash}(h)$ 開始的 Merkle 證明。然後，在假設(8)成立的情況下，輕節點可以自行檢查 x 是否確實是 $h[i]$ 的正確值。

在某些情況下，客戶端可能希望獲得值 $y = \text{Hash}(x) = \text{Hash}(h[i])$，例如，如果 x 本身非常大（例如，一個哈希映射本身）。那麼可以提供 (i, y) 的 Merkle 證明。如果 x 也是一個哈希映射，那麼可以從完整節點獲取從 $y = \text{Hash}(x)$ 開始的第二個 Merkle 證明，以提供一個值 $x[j] = h[i][j]$ 或僅其哈希值。

2.3.11. Importance of Merkle proofs for a multi-chain system such as TON. 需要注意的是，節點通常不能成為 TON 環境中所有分片鏈的全節點。它通常只是某些分片鏈的全節點，例如包含它自己賬戶、感興趣的智能合約或分片鏈的驗證節點分配給它的分片鏈。對於其他分片鏈，它必須是輕節點，否則存儲、計算和網絡帶寬要求將是禁止性的。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；它必須依賴於從其他分片鏈的全節點獲取的 Merkle 證明，這與自己檢查一樣安全，除非(8)失效（即找到哈希碰撞）。

2.3.12. Peculiarities of TON VM. 用於在主鏈和 Workchain Zero 中運行智能合約的 TON VM 或 TVM（Telegram 虛擬機）與受 EVM（Ethereum 虛擬機）啟發的傳統設計明顯不同：它不僅使用 256 位整數，實際上還使用（幾乎）任意的“記錄”、“結構”或“總和乘積類型”，使其更適合執行用高級（特別是函數式）語言編寫的代碼。基本上，TVM 使用標記數據類型，類似於 Prolog 或 Erlang 實現中使用的數據類型。

首先，人們可能會想像 TVM 智能合約的狀態不僅是一個 hashmap $\mathbf{2}^{256} \to \mathbf{2}^{256}$ 或 $\mathit{Hashmap}(256, \mathbf{2}^{256})$，而是（作為第一步）$\mathit{Hashmap}(256, X)$，其中 X 是具有多個構造函數的類型，使其能夠存儲除了 256 位整數之外的其他數據結構，特別是其他 hashmap $\mathit{Hashmap}(256, X)$。這意味著 TVM（持久性或臨時性）存儲的單元，或者 TVM 智能合約代碼中的變量或數組

[6]一個輕節點是一個不跟踪分片鏈的完整狀態的節點；相反，它只保留最近幾個區塊的哈希等最小信息，當需要檢查完整狀態的某些部分時，會依賴從完整節點獲取的信息。

元素，可能包含的不僅是整數，還包含整個新的 hashmap。當然，這意味著一個單元不僅包含 256 位，還包含一個 8 位標記，描述這 256 位應該如何解釋。

事實上，值不需要精確為 256 位。TVM 使用的值格式包括原始字節序列和對其他結構的引用，混合在任意順序中，某些描述符字節插入適當位置以區分指針和原始數據（例如，字符串或整數）；參見**2.3.14**。

這種原始值格式可以用於實現任意和積代數類型。在這種情況下，值將首先包含一個原始字節，描述所使用的“構造函數”（從高級語言的角度來看），然後包含其他“字段”或“構造函數參數”，取決於所選擇的構造函數，包括原始字節和對其他結構的引用（參見**2.2.5**）。然而，TVM 不知道構造函數和它們的參數之間的對應關係；字節和引用的混合是由某些描述符字節明確描述的。[7]

Merkle 樹哈希被擴展到任意這樣的結構：為了計算這樣一個結構的哈希值，所有引用都被遞歸地替換為所引用對象的哈希值，然後計算結果字節串（包括描述符字節）的哈希值。

通過這種方式，哈希映射的 Merkle 樹哈希，如**2.3.8**中所描述的，只是應用於任意（相關的）代數數據類型的哈希的一種特殊情況，應用於具有兩個構造函數的類型 $Hashmap(n, X)$。【註：實際上，Leaf 和 Node 是輔助類型 $HashmapAux(n, X)$ 的構造函數。類型 $Hashmap(n, X)$ 具有構造函數 Root 和 EmptyRoot，其中 Root 包含類型 $HashmapAux(n, X)$ 的值。】

2.3.13. Persistent storage of TON smart contracts. TON 智能合約的持久存儲基本上包括其在智能合約調用之間保留的“全局變量”。因此，它只是一種“產品”、“元組”或“記錄”類型，由正確類型的字段組成，每個字段對應一個全局變量。如果全局變量太多，它們由於 TON 單元大小的全局限制而無法容納在一個 TON 單元中。在這種情況下，它們被分成多個記錄，並組織成一棵樹，基本上變成了“產品的產品”或“產品的產品的產品”類型，而不僅僅是一種產品類型。

2.3.14. TVM Cells. 最終，TON VM 將所有數據保存在一組（*TVM*）單元格中。每個單元格首先包含兩個描述符字節，指示此單元格中有多少原始數據字節（最多 128 字節）以及有多少對其他單元格的引用（最多四個）。然後是這些原始數據字節和引用。每個單元格恰好被引用一次，因此我們可以在每個單元格中包含對其“父”（僅引用此單元格的單元格）的引用。但是，此引用不需要是顯式的。

[7]任何 TVM 單元格中存在的這兩個描述符字節僅描述引用的總數和原始字節的總數；引用在所有原始字節之前或之後一起保留。

通過這種方式，TON 智能合約的持久數據存儲單元被組織成一棵樹，[8]，並在智能合約描述中保留對此樹根的引用。如有必要，將從葉子開始遞歸地計算整個持久存儲的默克爾樹哈希，然後簡單地將單元格中所有引用替換為所引用的單元格的遞歸計算哈希，然後計算所獲得的字節串的哈希。

2.3.15. Generalized Merkle proofs for values of arbitrary algebraic types. 由於 TON VM 通過由（TVM）單元格構成的樹來表示任意代數類型的值，而且每個單元格都有一個明確定義的（遞歸計算的）Merkle 哈希，實際上取決於以該單元格為根的整個子樹，因此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 哈希的樹的某個子樹取特定的值或具有特定的哈希值。這樣就推廣了 **2.3.10**中僅考慮 $x[i] = y$ 的 Merkle 證明的方法。

2.3.16. Support for sharding in TON VM data structures. 我們剛剛概述了 TON 虛擬機如何在不過於複雜的情況下支持高級智能合約語言中任意（依賴）代數數據類型。然而，大型（或全局）智能合約的分片需要 TON 虛擬機層面上的特殊支持。為此，系統中添加了一個特殊版本的哈希映射類型，相當於一個“映射”$Account \dashrightarrow X$。這個“映射”可能看起來與 $Hashmap(m, X)$ 等價，其中 $Account = \mathbf{2}^m$。然而，當一個分片被分成兩個，或者兩個分片被合併時，這些哈希映射會自動分成兩個，或者合併回來，以便只保留屬於相應分片的鍵。

2.3.17. Payment for persistent storage. TON Blockchain 的一個值得注意的特點是從智能合約收取的費用，用於存儲其持久數據（即擴大區塊鏈的總狀態）。其工作方式如下：

每個塊聲明兩個價格，以區塊鏈的主要貨幣（通常是 Gram）為名：在持久存儲中保持一個單元的價格，以及在持久存儲的某個單元中保持一個原始字節的價格。每個帳戶使用的單元和字節的總數的統計數據被存儲為其狀態的一部分，因此通過將這些數字乘以塊頭聲明的兩個價格，我們可以計算出從上一個塊到當前塊之間保持其數據所需從帳戶餘額扣除的費用。

然而，對於每個塊中的每個帳戶和智能合約都不會收取持久存儲使用費用；相反，上次收取此費用的塊的序列號存儲在帳戶數據中，當對帳戶進行任何操作時（例如，進行價值轉移或接收和處理由智能合約發送的消息），在執行任何進一步操作之前，自前一次收取費用以來所有塊的存儲使用費用都將從帳戶餘額中扣除。如果在此之後帳戶的餘額變成負數，則帳戶將被銷毀。

[8]在邏輯上；當序列化時，“單元袋”表示法（在 **2.5.5**中描述）識別所有重複的單元格，將此樹轉換為有向無環圖（DAG）。

一個工作鏈可以聲明每個帳戶的一定數量的原始數據字節是“免費的”（即不參與持久存儲支付），以便使“簡單”的帳戶免於這些不斷的支付，這些帳戶只保留一兩種加密貨幣的餘額。

需要注意的是，如果沒有人向帳戶發送任何消息，其持久存儲支付將不會被收取，並且它可以無限期存在。然而，任何人都可以向這樣的帳戶發送一個空消息以銷毀它。從要銷毀的帳戶的原始餘額的一部分收集的小獎勵可以給予發送此類消息的人。但是，我們預期驗證人將免費銷毀此類無力償還的帳戶，僅為了減少全局區塊鏈狀態的大小並避免在沒有補償的情況下保留大量數據。

收集用於保持持久數據的支付會在分片鏈或主鏈的驗證人之間分配（在後一種情況下與其份額成比例）。

2.3.18. Local and global smart contracts; smart-contract instances. 智能合約通常只存在於一個分片中，根據智能合約的 $account\_id$ 選擇，與“普通”帳戶類似。對於大多數應用程序，這通常足夠了。然而，一些“高負載”智能合約可能希望在某些工作鏈的每個分片鏈中都擁有一個“實例”。為了實現這一點，它們必須將創建交易傳播到所有分片鏈中，例如，通過將此交易提交到工作鏈 w 的“根”分片鏈 $(w, \emptyset)$[9]，並支付高額佣金。[10]

這個操作在每個分片中有效地創建了智能合約的實例，並具有單獨的餘額。最初，創建交易中轉移的餘額僅通過將分片 (w, s) 中的實例的總餘額的 $2^{-|s|}$ 部分分配給它來進行分配。當一個分片分裂成兩個子分片時，全局智能合約所有實例的餘額會減半；當兩個分片合併時，餘額會相加。

在某些情況下，分裂/合併全局智能合約的實例可能涉及對這些智能合約的特殊方法進行（延遲的）執行。默認情況下，餘額按照上述描述進行分裂和合併，同時某些特殊的“帳戶索引”哈希表也會自動進行分裂和合併（參見 **2.3.16**）。

2.3.19. Limiting splitting of smart contracts. 全局智能合約可以在創建時限制其分裂深度 d，以使持久存儲費用更可預測。這意味著，如果分片鏈 (w, s)，其中 $|s| \geq d$，分裂為兩個，只有一個新的分片鏈會繼承智能合約的實例。該分片鏈是由某種確定性方式選擇的：每個全局智能合約都有一些“$account\_id$”，它本質上是其創建交易的哈希值，而其實例具有相同的 $account\_id$，只是將前 $\leq d$ 位替換為需要落入正確分片的適當值。這個 $account\_id$ 選擇了分裂後哪個分片鏈會繼承智能合約實例。

2.3.20. Account/Smart-contract state. 綜上所述，帳戶或智能合約狀態包括以下內容：

[9]一種更昂貴的替代方法是在主鏈中發布這樣的“全局”智能合約。

[10]這是一種所有分片的“廣播”功能，因此必須相當昂貴。

- 以區塊鏈主要貨幣表示的餘額
- 區塊鏈中其他貨幣的餘額
- 智能合約代碼（或其哈希）
- 智能合約持久數據（或其 Merkle 哈希）
- 有關使用持久存儲單元和原始字節數的統計信息
- 最後一次（實際上是主鏈區塊號）收集智能合約持久存儲費用的時間
- 從此帳戶轉移貨幣和發送消息所需的公鑰（可選；默認情況下等於 *account_id* 本身）。在某些情況下，可能會在這裡放置更複雜的簽名檢查代碼，類似於比特幣交易輸出的處理方式；此時 *account_id* 將等於此代碼的哈希。

此外，我們還需要在帳戶狀態或某些其他帳戶索引哈希表中保存以下數據：

- 帳戶的輸出消息隊列（參見 **??**）
- 最近傳遞消息的集合（哈希值）（參見 **2.4.23**）

並非所有這些都對每個帳戶都是必需的；例如，僅對智能合約需要智能合約代碼，而對於“簡單”帳戶不需要。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）中擁有非零餘額，但它可能在其他貨幣中具有零餘額。為了避免保留未使用的數據，定義了一種總和乘積類型（取決於工作鏈），它使用不同的標記字節（例如，TL 構造函數器；參見 **2.2.5**）區分使用的不同“構造函數器”。最終，帳戶狀態本身被保留為 TVM 持久存儲的單元集合。

2.4 Messages Between Shardchains

TON 區塊鏈的一個重要組成部分是區塊鏈之間的消息系統。這些區塊鏈可能是同一個工作鏈的分片鏈，也可能是不同工作鏈的分片鏈。

2.4.1. Messages, accounts and transactions: a bird's eye view of the system. 消息是從一個帳戶發送到另一個帳戶。每個交易包括一個帳戶接收一條消息，根據某些規則更改其狀態，並生成若干個（也可能是一個或零個）新的消息發送到其他帳戶。每個消息恰好生成和接收（交付）一次。

這意味著消息在系統中扮演了一個基礎性的角色，與帳戶（智能合約）的角色相當。從無限分片范式的角度來看（參見 **2.1.2**），每個帳戶都位於

其單獨的「帳戶鏈」中，它能夠影響其他帳戶狀態的唯一方法是通過發送消息。

2.4.2. Accounts as processes or actors; Actor model. 我們可以將帳戶（和智能合約）視為「進程」或「演員」，它們能夠處理傳入的消息，改變其內部狀態並生成一些外發消息作為結果。這與所謂的「Actor 模型」密切相關，該模型在 Erlang 等語言中使用（但是，Erlang 中的演員通常被稱為「進程」）。由於現有演員（即智能合約）也允許通過處理入站消息而創建新演員，因此與 Actor 模型的對應基本上是完整的。

2.4.3. Message recipient. 任何消息都有其接收者，由目標工作鏈識別符 w（默認情況下假定與起始分片鏈的相同）和接收者帳戶 *account_id* 所表示。*account_id* 的格式（即位數）取決於 w; 但是，該分片始終由其前 64 位（最高有效位）確定。

2.4.4. Message sender. 在大多數情況下，消息具有一個發送者，再次由一對 $(w', account\_id')$ 表示。如果存在，它位於消息接收者和消息值之後。有時，發件人不重要，或者是區塊鏈外的某個人（即不是智能合約），在這種情況下，此字段不存在。

請注意，演員模型不要求消息具有隱含的發送者。相反，消息可能包含對應答請求應發送到的演員的引用; 通常與發件人相同。但是，在加密貨幣（拜占庭）環境中，在消息中具有明確的不可偽造的發送者字段是有用的。

2.4.5. Message value. 消息的另一個重要特徵是其附加的值，它由源工作鏈和目標工作鏈都支持的一個或多個加密貨幣組成。消息的價值在其接收者之後立即指示; 它本質上是一個 $(currency\_id, value)$ 對的列表。

請注意，「簡單」帳戶之間的「簡單」價值轉移只是附帶某些值的空（無操作）消息。另一方面，稍微複雜一些的消息體可能包含簡單的文本或二進制評論（例如，關於付款目的的評論）。

2.4.6. External messages, or "messages from nowhere". 有些消息「從無處」進入系統---也就是說，它們不是由駐留在區塊鏈中的帳戶（智能合約或非智能合約）生成的。最典型的例子是當用戶想要從自己控制的帳戶轉移一些資金到其他帳戶時。在這種情況下，用戶向自己的帳戶發送一個「來自無處的消息」，要求它生成一個消息到接收帳戶，攜帶指定的值。如果此消息被正確簽署，她的帳戶就會接收到它並生成所需的發送消息。

事實上，可以將「簡單帳戶」視為具有預定義代碼的智能合約的特殊情況。此智能合約僅接收一類消息。此類入站消息必須包含一個生成的外發消息列表，以及簽名。智能合約會驗證簽名，並在簽名正確時生成所需的消息。

當然，「無來源的消息」和正常消息之間存在區別，因為「無來源的消息」不能攜帶價值，因此無法為自己的「gas」（即處理費用）付款。相反，在新的 shardchain 區塊中包含這些消息的建議之前，它們會受到一個小的 gas 限制的暫時執行。如果執行失敗（即簽名不正確），則認為「無來源的消息」不正確並予以棄置。如果在小的 gas 限制內未失敗，則可以將該消息包含在新的 shardchain 區塊中並完全處理，使用的 gas 支付（處理能力）會從接收方的帳戶中扣除。此外，「無來源的消息」還可以定義一些交易費用，該費用會從接收方的帳戶中扣除，以供重新分配給驗證者。

在這個意義上，“來自無處的消息”或“外部消息”扮演了其他區塊鏈系統（例如比特幣和以太坊）中使用的交易候選人的角色。

2.4.7. Log messages, or “messages to nowhere”. 同樣地，有時可以生成特殊消息並路由到特定的分片鏈，不是要傳送給其接收者，而是要記錄下來以便任何接收有關該分片的更新的人輕鬆觀察。這些記錄的消息可以在用戶的控制台中輸出，或觸發在鏈外服務器上運行某些腳本。在這個意義上，它們代表“區塊鏈超級計算機”的外部“輸出”，就像“來自無處的消息”代表“區塊鏈超級計算機”的外部“輸入”一樣。

2.4.8. Interaction with off-chain services and external blockchains. 這些外部輸入和輸出消息可以用於與鏈外服務和其他（外部）區塊鏈（如比特幣或以太坊）交互。人們可以在 TON 區塊鏈中創建與比特幣、以太幣或在以太坊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“來自無處的消息”和“發送到無處的消息”，由某些第三方鏈外服務器上的腳本生成和處理，實現 TON 區塊鏈與這些外部區塊鏈之間所需的交互。

2.4.9. Message body. 「消息體」只是一個字節序列，其含義僅由接收的工作鏈和/或智能合約確定。對於使用 TON VM 的區塊鏈，這可以是通過“Send()”操作自動生成的任何 TVM 單元格的序列化。這樣的序列化只需通過遞歸地將 TVM 單元格中的所有引用替換為所引用的單元格即可獲得。最終出現一個原始字節串，通常在其前面加上一個 4 字節的“消息類型”或“消息構造函數”，用於選擇接收智能合約的正確方法。

另一個選項是使用 TL 序列化對象作為消息體（參見 **2.2.5**）。這對於不使用 TON VM 的不同工作鏈之間的通信可能特別有用。

2.4.10. Gas limit and other workchain/VM-specific parameters. 有時候，消息需要攜帶有關 gas 限制、gas 價格、交易費用和類似值的信息，這些值取決於接收工作鏈，只與接收工作鏈有關，但不一定與發起工作鏈有關。這些參數包含在或在消息體之前，有時（取決於工作鏈）帶有特殊的 4 字節前綴，指示它們的存在（可以由 TL 方案定義；參見 **2.2.5**）。

2.4.11. Creating messages: smart contracts and transactions. 有兩種新消息的來源。大多數消息是在智能合約執行期間創建的（通過 TVM 中的 `Send()` 操作），當某個智能合約被調用來處理傳入的消息時。或者，消息可以來自外部作為“外部消息”或“來自無處的消息”（參見 **2.4.6**）。[11]

2.4.12. Delivering messages. 當一個消息到達包含其目標帳戶的分片鏈時，[12]它將被“傳遞”給其目標帳戶。接下來發生什麼取決於工作鏈；從外部的角度來看，重要的是這樣的消息永遠不可能從這個分片鏈進一步轉發。

對於基本工作鏈的分片鏈，交付包括將消息值（減去任何 gas 費用）添加到接收帳戶的餘額中，並且如果接收帳戶是智能合約，還可能在之後調用與消息有關的方法。事實上，智能合約只有一個進入點來處理所有傳入的消息，並且必須通過查看它們的前幾個字節（例如，包含 TL 構造的前四個字節; 參見**2.2.5**）來區分不同類型的消息。

2.4.13. Delivery of a message is a transaction. 因為消息的交付會更改帳戶或智能合約的狀態，所以它是接收分片鏈中的一個特殊交易，並明確地註冊為這樣的交易。實質上，所有 TON Blockchain 交易都包括將一個傳入消息交付給其接收帳戶（智能合約），忽略了一些次要的技術細節。

2.4.14. Messages between instances of the same smart contract. 回想一下，智能合約可以是本地的（即，像任何普通帳戶一樣存在於一個分片鏈中）或者是全局的（即，在所有分片中都有實例，或者至少在所有深度為 d 的分片中都有實例；參見 **2.3.18**）。如果需要，全局智能合約的實例可以交換特殊消息來傳輸信息和價值。在這種情況下，（不可偽造的）發送者 *account_id* 變得重要（參見 **2.4.4**）。

2.4.15. Messages to any instance of a smart contract; wildcard addresses. 有時需要將消息（例如客戶端請求）傳遞給任何一個全局智能合約的實例，通常是最接近的實例（如果有一個實例存在於與發送者相同的分片鏈中，則是明顯的候選）。一種做法是使用“萬用接收者地址”，允許目標 *account_id* 的前 d 位取任意值。實際上，通常會將這些 d 位設置為與發送者的 *account_id* 相同的值。

2.4.16. Input queue is absent. 所有由區塊鏈（通常是分片鏈；有時是主鏈）接收到的消息，或者實質上由某個分片鏈內的「帳戶鏈」接收到的

[11]以上僅對基本工作鏈及其分片鏈需要是文字上的真實，其他工作鏈可能提供其他創建消息的方式。

[12]作為退化的情況，這個分片鏈可能與起始的分片鏈重合——例如，如果我們在一個尚未被劃分的工作鏈中工作。

消息，都會立即被交付（即由接收帳戶處理）。因此，實際上不存在「輸入隊列」。相反，如果由於區塊大小和 gas 使用的總限制，無法處理特定分片鏈的所有消息，則一些消息只能在起始分片鏈的輸出隊列中累積。

2.4.17. Output queues. 從無限分片範式（參見 **2.1.2**）的角度來看，每個帳戶鏈（即每個帳戶）都有自己的輸出隊列，由其生成但尚未交付給其接收者的所有消息組成。當然，帳戶鏈只有虛擬存在；它們被分組成為分片鏈，每個分片鏈都有一個輸出「隊列」，由屬於該分片鏈的所有帳戶的輸出隊列的聯合組成。

這個分片鏈輸出「隊列」僅對其成員消息施加部分排序。換句話說，在前一個區塊中生成的消息必須在後續區塊中生成的任何消息之前交付，而由同一帳戶生成且目的地相同的任何消息必須按照它們的生成順序交付。

2.4.18. Reliable and fast inter-chain messaging. 對於像 TON 這樣的可擴展的多區塊鏈項目來說，能夠在不同的分片鏈之間轉發和傳遞消息（參見 **2.1.3**）非常重要，即使系統中有數百萬個分片鏈也是如此。這些消息應該被可靠地（即，消息不應該丟失或傳送多次）且快速地傳送。TON Blockchain 通過使用兩種“消息路由”機制的組合來實現此目標。

2.4.19. Hypercube routing: “slow path” for messages with assured delivery. TON Blockchain 使用“超立方路由”作為從一個 Shardchain 到另一個 Shardchain 傳遞消息的一種緩慢但安全可靠的方式，必要時使用多個中間的 Shardchain 進行中轉。否則，任何特定 Shardchain 的驗證人都需要跟踪所有其他 Shardchain 的狀態（即其輸出隊列），隨著 Shardchain 總量的增加，這將需要過多的計算能力和網絡帶寬，從而限制系統的可擴展性。因此，不可能直接從任何 Shard 向每個其他 Shard 傳遞消息。相反，每個 Shard 僅與在其 (w, s) Shard 標識符的一個十六進制數字上有區別的 Shard 相“連接”（參見 **2.1.8**）。通過這種方式，所有 Shardchain 構成一個“超立方體”圖，消息沿著這個超立方體的邊傳播。

如果將消息發送到與當前 Shard 不同的 Shard，則當前 Shard 標識符的一個十六進制數字（由決定性地選擇）將被目標 Shard 的相應數字替換，並且使用生成的標識符作為近似目標將消息轉發到。（註：這不一定是計算超立方體路由的下一個跳躍的算法的最終版本。特別地，十六進制數字可以被 r 位組替換，其中 r 是可配置的參數，不一定等於 4。）

超立方體路由的主要優勢在於，區塊有效性條件意味著創建 Shardchain 區塊的驗證人必須收集並處理來自“相鄰”Shardchain 的輸出隊列中的消息，否則將失去其權益。以這種方式，任何消息最終都能到達其最終目的地；消息不會在傳輸中丟失或重複傳遞。

請注意，由於需要通過幾個中間的 Shardchain 轉發消息，因此超立方體路由引入了一些額外的延遲和開銷。然而，這些中間 Shardchain 的數量增長非常緩慢，僅與總 Shardchain 數 N 的對數 $\log N$（更精確地說，是 $\lceil \log_{16} N \rceil - 1$）有關。例如，如果 $N \approx 250$，最多只有一個中間跳躍；對於 $N \approx 4000$ 個 Shardchain，最多有兩個中間跳躍。通過四個中間跳躍，我們可以支持高達 100 萬個 Shardchain。我們認為這是為系統實現基本無限擴展性所付出的非常小的代價。實際上，甚至不需要支付這個代價：

2.4.20. Instant Hypercube Routing: "fast path" for messages. TON Blockchain 的一個新特性是引入了一個"快速路徑"用於轉發消息，使得在大多數情況下可以完全繞過**2.4.19**中的"慢速"超立方體路由，將消息傳遞到最終目的地 Shardchain 的下一個區塊。

具體的想法如下。在"慢速"超立方體路由期間，消息在超立方體的邊緣上（在網絡中）傳播，但在每個中間節點上（約 5 秒）被延遲，以便在繼續傳播之前進行提交到相應的 Shardchain 中。

為了避免不必要的延遲，可以在不等待將消息提交到中間 Shardchain 的情況下，沿著超立方體的邊緣中繼消息和適當的 Merkle 證明。事實上，應將網絡消息從原始 Shard 的"任務組"（參見 **2.6.8**）的驗證程序轉發到目標 Shard 的"任務組"的指定區塊生成器（參見 **2.6.9**）。這可能是直接完成，而不需沿著超立方體的邊緣進行。當具有 Merkle 證明的此消息到達目的地 Shardchain 的驗證程序（更準確地說，是確定器；參見 **2.6.5**）時，他們可以立即將其提交到新塊中，而不必等待消息完成其沿著"慢速路徑"的行程。然後，使用適當的 Merkle 證明發送確認交付，沿著超立方體的邊緣返回，並可用於通過提交特殊交易停止消息沿"慢速路徑"的行程。

請注意，此"即時傳遞"機制不會取代 **2.4.19**中描述的"慢速"但可靠的機制。仍然需要"慢速路徑"，因為驗證者不能因為丟失或者決定不將"快速路徑"消息提交到他們區塊鏈的新塊中而受到懲罰。[13]

因此，兩種消息轉發方法並行運行，"慢速"機制只有在"快速"機制的成功證明提交到中間分片鏈時才會中止。[14]

2.4.21. Collecting input messages from output queues of neighboring shardchains. 當為分片鏈提出新塊時，某些相鄰（在**2.4.19**中的路由超立方體的意義上）分片鏈的輸出消息將作為"輸入"消息包含在新塊中並立即傳遞（即進行處理）。這裡有某些規則，關於這些鄰居輸出消息必須按照哪個順序進行處理。基本上，必須在處理任何"較新"的消息之前，先

[13]但是，驗證者有一些激勵措施盡快這樣做，因為他們將能夠收集與消息相關的所有轉發費用，這些費用尚未沿著慢速路徑消耗完。

[14]事實上，人們可以暫時或永久地完全禁用"即時傳遞"機制，系統仍然可以運作，雖然速度會更慢。

傳遞“舊”的消息（來自引用較老的主鏈塊的分片鏈塊）；對於來自同一相鄰分片鏈的消息，必須遵守**2.4.17**中描述的輸出隊列的偏序關係。

2.4.22. Deleting messages from output queues. 一旦觀察到輸出隊列消息已經被相鄰分片鏈傳遞，就會通過特殊交易將其從輸出隊列中明確刪除。

2.4.23. Preventing double delivery of messages. 為了防止從相鄰分片鏈的輸出隊列中取出的消息被重複傳遞，每個分片鏈（更準確地說，其中的每個帳戶鏈）將最近傳遞的消息的集合（或僅其哈希值）作為其狀態的一部分保留。當觀察到已傳遞的消息被其源頭的相鄰分片鏈（參見**2.4.22**）從輸出隊列中刪除時，它也從最近傳遞的消息集合中刪除。

2.4.24. Forwarding messages intended for other shardchains. 超立方體路由（參見**2.4.19**）意味著有時會將外發消息傳遞給相鄰的分片鏈，而不是包含目標收件人的分片鏈，這些相鄰分片鏈位於通往目的地的超立方體路徑上。在這種情況下，“傳遞”意味著將入站消息移動到出站隊列。這在塊中明確地反映為一筆特殊的轉發交易，其中包含消息本身。基本上，這看起來像是某人在分片鏈內部收到了該消息，並生成了一條相同的消息。

2.4.25. Payment for forwarding and keeping a message. 實際上，轉發交易會花費一些燃料（取決於轉發的消息大小），因此為代表此分片鏈的驗證者扣除了轉發費用。即使由於超立方體路由，該消息已轉發多次，此轉發費用通常比最終將消息傳遞給其收件人時所收取的燃料費用要小得多。此外，只要消息保留在某個分片鏈的輸出隊列中，它就是該分片鏈的全局狀態的一部分，因此還可以通過特殊交易收取長期保留全局數據的費用。

2.4.26. Messages to and from the masterchain. 消息可以直接從任何分片鏈發送到主鏈，反之亦然。但是，發送消息到主鏈並在主鏈中處理消息的燃料價格相當高，因此只有在真正必要的情況下才會使用此功能，例如驗證者存入股份時。在某些情況下，可以為發送到主鏈的消息定義最小的存款（附加值），只有在接收方認為消息是“有效”的情況下才會退還。

消息無法通過主鏈自動路由。具有 $workchain\_id \neq -1$（其中 -1 是表示主鏈的特殊 $workchain\_id$）的消息無法傳送到主鏈。

原則上，可以在主鏈內部創建一個消息轉發智能合約，但使用它的代價是不可承受的。

2.4.27. Messages between accounts in the same shardchain. 在某些情況下，某個屬於某個分片鏈的帳戶生成了一條消息，該消息的目的地是該分片鏈中的另一個帳戶。例如，在尚未分裂為多個分片鏈的新工作鏈中，由於負載可管理，此類情況可能發生。

這些消息可能會累積在分片鏈的輸出隊列中，然後在後續區塊中作為入站消息進行處理（對於此目的，任何分片都被認為是自己的鄰居）。但是，在大多數情況下，可以在源區塊本身內傳遞這些消息。

為了實現這一點，所有包含在分片鏈區塊中的交易都受到部分順序的限制，交易（每個交易包括將消息傳遞給某個帳戶）是在尊重這個部分順序的情況下進行處理的。特別地，允許交易根據這個部分順序處理先前交易的某些輸出消息。

在這種情況下，消息主體不會被複製兩次。相反，發送和處理交易將引用消息的共享副本。

2.5 Global Shardchain State. “Bag of Cells” Philosophy.

現在我們可以描述 TON 區塊鏈的全局狀態，或者至少是基礎工作鏈的分片鏈的全局狀態。

我們從“高層次”或“邏輯”描述開始,即全局狀態是代數類型 *ShardchainState* 的值。

2.5.1. Shardchain state as a collection of account-chain states. 根據無限分片範式（參見**2.1.2**），任何分片鏈都只是包含恰好一個帳戶的虛擬“帳戶鏈”（暫時）集合。這意味著，基本上全局分片鏈狀態必須是一個哈希表

$$ShardchainState := (Account \dashrightarrow AccountState) \tag{24}$$

其中此哈希表中的所有 *account_id* 索引必須以 s 為前綴，如果我們正在討論分片 (w, s) 的狀態（參見**2.1.8**）。

因為有無限分片的設計（參見 **2.1.2**），任何 shardchain 只是包含一個虛擬「帳戶鏈」集合（暫時性的），每個鏈只包含一個帳戶。因此，全局 shardchain 狀態本質上必須是哈希表

$$ShardchainState := (Account \dashrightarrow AccountState) \tag{25}$$

其中，這個哈希表的所有索引 *account_id* 都必須以 s 為前綴，如果我們討論的是 (w, s) 的 shard 狀態（參見 **2.1.8**）。

在實踐中，我們可能希望將 *AccountState* 分為幾個部分（例如，將帳戶輸出消息隊列保持獨立，以簡化其被鄰近 shardchain 檢查的流程），並在 *ShardchainState* 內部使用幾個哈希表 $(Account \dashrightarrow AccountStatePart_i)$。我們還可以添加少量「全局」或「整體」參數到 *ShardchainState* 中（例如，所有帳戶總餘額或所有輸出隊列中的總消息數）。

然而，(25)至少從"邏輯"("高層次")角度來看是描述分片鏈全局狀態的良好近似值。代數類型 *AccountState* 和 *ShardchainState* 的形式描述可以使用 TL-scheme（參見 **2.2.5**）完成，將在其他地方提供。

2.5.2. Splitting and merging shardchain states. 注意到分片鏈狀態 (25) 的無限分片範式描述顯示了當分片被分裂或合併時應該如何處理此狀態。事實上，這些狀態轉換通過使用 hashmap 實現起來非常簡單。

2.5.3. Account-chain state. (虛擬的) 帳戶鏈狀態只是一個帳戶的狀態，由類型 *AccountState* 描述。通常情況下，它具有列在 **2.3.20** 中的所有或部分字段，具體取決於使用的特定構造函數。

2.5.4. Global workchain state. 注意，和 (25) 類似，我們可以使用相同的公式定義全域的 *workchain* 狀態，但 *account_id* 可以採取任何值，而不僅僅是屬於一個 shard 的值。和 **2.5.1** 中提到的類似，我們可能想要將此 hashmap 拆分為數個 hashmap，並且可能要添加一些「整體」參數，例如總餘額。

基本上，全域 workchain 狀態的型別必須與 shardchain 狀態相同，因為如果所有現有的 workchain 中的 shardchain 突然合併為一個 shardchain，那麼我們將得到 shardchain 狀態。

2.5.5. Low-level perspective: "bag of cells". 除了上述"高層次"描述之外，還有一個關於賬戶鏈或分片鏈狀態的"低層次"描述，這是相當重要的，因為它被證明是相當通用的，為表示、存儲、序列化和通過網絡傳輸 TON Blockchain 使用的幾乎所有數據（塊、分片鏈狀態、智能合約存儲、默克爾證明等）提供了一個共同的基礎。同時，一旦理解並實現了這樣一個通用的"低層次"描述，就可以將我們的注意力集中在"高層次"考慮上。

回想一下，TVM 通過 TVM 單元樹或簡稱為單元（請參閱 **2.3.14** 和 **2.2.5**）來表示任意代數類型（包括 (25)中的 *ShardchainState* 型）。

每個這樣的 cell 由兩個描述字節（descriptor bytes）構成，定義了某些標誌和值，其中 $0 \le b \le 128$ 表示原始字節數，$0 \le c \le 4$ 表示對其他 cell 的引用數量。接下來是 b 個原始字節和 c 個 cell 引用。[15]

單元格引用的確切格式取決於實現方式以及該單元格是否位於 RAM、磁盤、網絡封包或區塊中等。一個有用的抽象模型是想像所有單元格都保存在內容可尋址內存中，其中單元格的地址等於其（SHA256）哈希值。請

[15]可以證明，如果一個包含多個 cell 的樹上所有數據的 Merkle 證明同等頻繁地需要，則應使用 $b + ch \approx 2(h + r)$ 的 cell 來最小化平均 Merkle 證明大小，其中 $h = 32$ 是 hash 大小（以字節為單位），$r \approx 4$ 是 cell 引用的"字節大小"。換句話說，一個 cell 應該包含兩個引用和一些原始字節，或者一個引用和約 36 個原始字節，或者完全沒有引用並且有 72 個原始字節。

注意，單元格的（Merkle）哈希確切地是通過將其子單元格的引用替換為它們（遞歸計算的）哈希值並哈希結果字節串來計算的。

這樣，如果我們使用單元哈希來引用單元格（例如，在其他單元格的描述中），系統會簡化一些，單元格的哈希開始與表示它的字節串的哈希相一致。

現在我們看到，任何可以用 *TVM* 表示的對象，包括全局分片鏈狀態，都可以表示為一個“單元袋”——即，一組單元格以及其中一個單元格的“根”引用（例如，通過哈希）。請注意，此描述中會移除重複的單元格（“單元袋”是單元格的集合，而不是多重集合），因此抽象樹表示實際上可能變成有向無環圖（dag）表示。

甚至可以在一個 *B*-或 *B*+-tree 上將此狀態保留在磁盤上，該 *B*-或 *B*+-tree 包含所有相關單元格（可能帶有一些附加數據，例如子樹高度或引用計數），並以單元格哈希索引。然而，將此想法的天真實現會導致一個智能合約的狀態分散在磁盤文件的不同部分，這是我們不希望發生的。[16]

現在我們將詳細解釋 TON Blockchain 使用的幾乎所有對象如何表示為“單元包”，從而展示了這種方法的通用性。

2.5.6. Shardchain block as a “bag of cells”. 一個分片鏈塊本身也可以描述為一種代數類型，並存儲為一個“單元包”。然後，可以通過將表示“單元包”中每個單元的字節字符串串聯在一起，以任意順序獲取塊的天真二進制表示形式。例如，可以在塊的開始提供所有單元的偏移列表，並在可能的情況下將對其他單元的哈希引用替換為此列表中的 32 位索引，從而改進和優化此表示法。然而，應該想象一個塊基本上是一個“單元包”，而所有其他技術細節僅是次要的優化和實現問題。

2.5.7. Update to an object as a “bag of cells”. 想像一下，我們有一個表示為“單元包”的某個對象的舊版本，我們希望表示相同對象的新版本，並假設新版本不會與上一個版本差太多。人們可以簡單地將新狀態表示為另一個具有自己根的“單元包”，並從其中刪除所有出現在舊版本中的單元格。剩下的“單元包”本質上是對該對象的更新。每個擁有此對象的舊版本和更新的人都可以通過合併這兩個單元包，並刪除舊的根（減少其引用計數，如果引用計數為零則釋放該單元格）來計算新版本。

2.5.8. Updates to the state of an account. 特別是，可以使用 **2.5.7**中描述的想法來表示對帳戶狀態、分片鏈的全局狀態或任何哈希表的更新。這意味著當我們收到一個新的分片鏈塊（它是一個“單元包”）時，我們不

[16]更好的實現方式是，如果智能合約的狀態很小，則將其保留為序列化字符串，如果智能合約的狀態很大，則在單獨的 *B*-tree 中保留其狀態；然後，代表區塊鏈狀態的頂級結構將是一個 *B*-tree，其葉子節點可以包含對其他 *B*-tree 的引用。

僅僅是單獨解釋這個"單元包"，而是首先將其與表示分片鏈以前狀態的"單元包"結合起來來解釋它。在這個意義上，每個塊可能"包含"整個區塊鏈的狀態。

2.5.9. Updates to a block. 回想一下，區塊本身是一個"單元包"，因此，如果需要編輯區塊，可以類似地定義一個"區塊更新"為一個"單元包"，在存在作為此區塊以前版本的"單元包"存在的情況下進行解釋。這大致上是 **2.1.17**中討論的"垂直區塊"背後的思想。

2.5.10. Merkle proof as a "bag of cells". 需要注意的是，一個（廣義的）Merkle 證明，例如從已知的 $\textsc{Hash}(x) = h$ 開始斷言 $x[i] = y$（參見 **2.3.10**和 **2.3.15**），也可以表示為一個"單元包"。換句話說，只需要提供一個細胞子集，對應到一條從 $x : \mathit{Hashmap}(n, X)$ 的根到其所需的索引 $i : \mathbf{2}^n$ 和值 $y : X$ 的葉子的路徑。這個證明中不在這條路徑上的子代引用將被保留為單元格哈希值。還可以提供同時證明 $x[i] = y$ 和 $x[i'] = y'$ 的 Merkle 證明，方法是在"單元包"中包含從 x 的根到對應於索引 i 和 i' 的葉子的兩條路徑上的單元格。

2.5.11. Merkle proofs as query responses from full nodes. 基本上，具有分片鏈（或帳戶鏈）完整副本的完整節點可以在輕節點（例如運行 T0N 區塊鏈客戶端的輕版本的網絡節點）請求時提供 Merkle 證明，使接收者能夠執行一些簡單的查詢而無需外部幫助，僅使用此 Merkle 證明中提供的單元格。輕節點可以將其查詢以序列化格式發送到完整節點，並使用 Merkle 證明接收正確答案-或者只使用 Merkle 證明，因為請求者應該能夠僅使用包含在 Merkle 證明中的單元格計算答案。這個 Merkle 證明將簡單地由"單元格包"組成，僅包含完整節點在執行輕節點查詢時訪問過的分片鏈狀態中的那些單元格。特別是，這種方法可以用於執行智能合約的"獲取查詢"（參見 **4.3.12**）。

2.5.12. Augmented update, or state update with Merkle proof of validity. 回想一下（參見 **2.5.7**），我們可以通過"更新"來描述從舊值 $x : X$ 到新值 $x' : X$ 的對象狀態的更改，該更新只是一個"單元包"，其中包含位於表示新值 x' 的子樹中的那些單元格，但不包含位於表示舊值 x 的子樹中的任何單元格，因為假定接收者擁有舊值 x 及其所有單元格的副本。

然而，如果接收者沒有 x 的完整副本，而只知道它的（Merkle）哈希值 $h = \textsc{Hash}(x)$，它將無法檢查更新的有效性（即所有"懸空"的單元格引用是否都指向 x 的樹中存在的單元格）。人們希望擁有"可驗證"的更新，加上對舊狀態中所有引用單元格存在的 Merkle 證明。然後，任何只知道 $h = \textsc{Hash}(x)$ 的人都能夠檢查更新的有效性，並自行計算出新的 $h' = \textsc{Hash}(x')$。

因為我們的 Merkle 證明本身就是“單元格包”（參見 **2.5.10**），因此可以將這樣的增強更新表示為一個“單元格包”，其中包含 x 的舊根、一些其子孫以及從 x 的根到它們的路徑，以及 x' 的新根和所有不屬於 x 的子孫。

2.5.13. Account state updates in a shardchain block. 特別是，應按照 **2.5.12**中討論的方式增加在 shardchain 區塊中的帳戶狀態更新。否則，某些人可能會提交包含無效狀態更新的區塊，這些狀態更新引用了在舊狀態中不存在的單元格；證明此類區塊的無效性將是問題的（挑戰者如何證明一個單元格不是先前狀態的一部分?）。

如果一個塊中包含的所有狀態更新都是擴充的，那麼它們的有效性很容易檢查，它們的無效性也很容易表明為（廣義）Merkle 哈希的遞歸定義特性的違反。

2.5.14. “Everything is a bag of cells” philosophy. 前面的討論顯示，我們需要在 TON 區塊鏈或網絡中存儲或傳輸的所有內容都可以表示為“單元包”。這是 TON 區塊鏈設計哲學的重要部分。一旦解釋了“單元包”方法並定義了一些“低級”的“單元包”序列化，就可以在抽象（依賴）代數數據類型的高級水平上定義所有內容（塊格式、分片鏈和帳戶狀態等）。

“一切皆為單元包”的哲學具有統一效應，可以顯著簡化看似不相關服務的實現；參見**5.1.9**，其中提供了一個與支付通道有關的例子。

2.5.15. Block “headers” for TON blockchains. 通常，在區塊鏈中，一個區塊以一個小的標頭開始，其中包含前一個區塊的哈希值、創建時間、包含在該區塊中的所有交易的梅克爾哈希值等。然後，定義區塊哈希為該小區塊標頭的哈希值。由於區塊標頭最終取決於區塊中包含的所有數據，因此無法在不改變其哈希的情況下更改區塊。

在 TON 區塊鏈的區塊中使用的「單元包」方法中，沒有指定的區塊標頭。相反，區塊哈希定義為該區塊的根單元格（Merkle）哈希。因此，該區塊的頂級（根）單元格可以被視為該區塊的一個小的「標頭」。

然而，根單元格可能不包含通常期望從此類標頭中獲取的所有數據。本質上，我們希望標頭包含 *Block* 數據類型中定義的某些字段。通常，這些字段將包含在幾個單元格中，包括根單元格。這些單元格共同構成了相關字段值的「Merkle 證明」。人們可能會堅持在區塊的開始處包含這些「標頭單元格」，在任何其他單元格之前。然後，只需要下載區塊序列化的前幾個字節，就可以獲取所有的「標頭單元格」，並學習所有期望的字段。

2.6 Creating and Validating New Blocks

TON 區塊鏈最終由分片區塊和主區塊組成。為了使系統平穩、正確地運行，這些區塊必須經由網路創建、驗證和傳播到所有相關方。

2.6.1. Validators. 新區塊由特殊的指定節點（稱為驗證者）創建和驗證。基本上，任何希望成為驗證者的節點都可以成為驗證者，只要它可以將足夠大的股份（以 TON 硬幣即 Grams 計算，參見附錄 **A**）存入主區塊鏈中。驗證者通過創建新生成的區塊中所有交易（消息）的交易、存儲和 gas 費用以及一些新鑄的硬幣獲得一些「獎勵」，以表彰整個社區對驗證者維持 TON 區塊鏈運作的「感激之情」。這個收入按比例分配給所有參與的驗證者，與他們的股份成正比。

然而，成為驗證者是一項高責任的工作。如果驗證者簽署了無效的區塊，它可能會因此失去部分或全部股份，並被暫時或永久地排除在驗證者的集合之外。如果驗證者未參與創建區塊，它將不會收到與該區塊相關的獎勵份額。如果驗證者長時間不參與創建新區塊，它可能會失去部分股份，並被暫停或永久排除在驗證者的集合之外。

所有這些意味著驗證者不會白白地得到金錢。事實上，它必須跟踪所有或某些分片區塊鏈的狀態（每個驗證者負責驗證和創建某個分片區塊鏈的新區塊），執行這些分片區塊鏈中智能合約所請求的所有計算，接收其他分片區塊鏈的更新等等。這些活動需要相當大的磁盤空間、計算能力和網絡帶寬。

2.6.2. Validators instead of miners. 回想一下，TON 區塊鏈使用的是權益證明方法，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證明方法。這意味著不能通過提供一些工作證明（計算大量無用哈希）來「挖掘」新區塊，並因此獲得一些新的硬幣。相反，必須成為驗證者，花費自己的計算資源來存儲和處理 TON 區塊鏈的請求和數據。簡而言之，「必須成為驗證者才能挖掘新的硬幣。」在這方面，驗證者是新的挖礦者。

然而，除了成為驗證者外，還有其他賺取硬幣的方法。

2.6.3. Nominators and "mining pools". 成為驗證者，通常需要購買和安裝幾台高性能服務器，並為它們獲取良好的互聯網連接。這不像挖掘比特幣時需要的 ASIC 設備那樣昂貴。然而，一定不能在家用計算機上，更不能在智能手機上挖掘新的 TON 硬幣。

在比特幣、以太坊和其他工作證明加密貨幣挖掘社區中，存在著所謂的挖礦池概念，許多節點沒有足夠的計算能力來自行挖掘新的區塊，因此將他們的努力結合起來，並在之後分享獎勵。

在權益證明世界中，對應的概念是提名人。基本上，這是一個節點借出自己的資金來幫助驗證者增加其股份；然後，驗證者將相應份額的獎勵（或其先前同意的一部分，比如 50

通過這種方式，提名人也可以參與「挖礦」，並獲得與其願意為此目的存款的金額成比例的獎勵。它只接收驗證者獎勵相應份額的一部分，因為它只提供「資本」，而不需要購買計算能力、存儲和網絡帶寬。

但是，如果驗證者因為無效行為而失去了其賭注，提名人也會失去其賭注的份額。在這個意義上，提名人分擔風險。它必須明智地選擇其提名的驗證者，否則可能會損失金錢。在這個意義上，提名人通過其資金權重決策並「投票」給某些驗證者。

另一方面，這種提名或借貸系統使人們可以在不先投資大量 Grams（TON 硬幣）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. Fishermen: obtaining money by pointing out others' mistakes. 另一種在不成為驗證者的情況下獲得一些獎勵的方法是成為 *fisherman*。基本上，任何節點都可以通過在主鏈上進行一個小的存款來成為 fisherman。然後，它可以使用特殊的主鏈交易來發佈（Merkle）無效證明，這些證明證明某些（通常是分片鏈）塊之前已被驗證者簽署和發布。如果其他驗證者同意這個無效證明，違反規定的驗證者將被懲罰（失去部分賭注），而 fisherman 將獲得一些獎勵（從違反規定的驗證者沒收的硬幣的一部分）。之後，必須按照 **2.1.17**中所述的方式對無效的（分片鏈）塊進行更正。更正無效的主鏈塊可能涉及在先前提交的主鏈塊之上創建「垂直」塊（參見 **2.1.17**）；不需要創建主鏈的分叉。

通常，一個 fisherman 至少需要成為一些分片鏈的完整節點，並通過運行至少一些智能合約的代碼來花費一些計算資源。儘管 fisherman 不需要像驗證者一樣擁有那麼多計算能力，但我們認為一個自然的 fisherman 候選人是一個準備處理新塊但尚未當選為驗證者的人（例如，由於未能存入足夠的賭注）。

2.6.5. Collators: obtaining money by suggesting new blocks to validators. 另一種在不成為驗證者的情況下獲得一些報酬的方式是成為整合器。整合器是一個節點，它準備並建議驗證者使用新的分片鏈區塊候選人，該候選人補充了來自該分片鏈及其他（通常是相鄰的）分片鏈的狀態以及適當的 Merkle 證明的數據。（例如，當需要從相鄰的分片鏈轉發一些消息時，這是必要的。）然後，驗證者可以輕鬆檢查建議的區塊候選人的有效性，而不必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（整理後的）區塊候選方案才能獲得一些（「挖礦」）獎勵，因此向願意提供合適的區塊候選方案的整理者支付一部分獎勵

是有意義的。通過這種方式，驗證者可以通過外包來源於鄰近的分片鏈的狀態，使自己免於必須監視它。

然而，在系統的初始部署階段，我們預期不會有單獨指定的整理者，因為所有驗證者都將能夠充當自己的整理者。

2.6.6. Collators or validators: obtaining money for including user transactions. 用戶可以向某些整理者或驗證者開啟微支付通道，支付少量硬幣，以換取將其交易納入分片鏈。

2.6.7. Global validator set election. 「全球」驗證者集合每個月選舉一次（實際上是每 2^{19} 個主分片鏈區塊）。該集合是提前一個月確定的並且全球公知。

為了成為一個驗證者，節點必須將一些 TON 硬幣（Grams）轉移到主分片鏈，然後將它們作為建議的押金 s 發送到一個特殊的智能合約。與押金一起發送的另一個參數是 $l \geq 1$，表示此節點相對於最小可能的驗證負載所願意接受的最大值。還有一個全局上限（另一個可配置的參數）L，等於 10，限制了 l 的大小。

然後，全球驗證者集合是由此智能合約選出的，它只需選擇最大建議押金的 T 個候選者並公佈其身份。最初，驗證者的總數為 $T = 100$；隨著負載的增加，我們預計它將增長到 1000。這是一個可配置的參數（參見 **2.1.21**）。

每個驗證者的實際押金如下計算：如果前 T 個建議的押金為 $s_1 \geq s_2 \geq \cdots \geq s_T$，則第 i 個驗證者的實際押金設置為 $s'_i := \min(s_i, l_i \cdot s_T)$。通過這種方式，$s'_i/s'_T \leq l_i$，因此第 i 個驗證者不會獲得比最弱驗證者的負載多出 $l_i \leq L$ 倍的負載（因為負載最終與押金成比例）。

然後當選的驗證者可以撤回未使用部分的股份，即 $s_i - s'_i$。未獲選的驗證者候選人可以撤回他們提出的所有股份。

每個驗證者都公佈其公共簽名金鑰，不一定等於來自帳戶的公鑰。[17]

驗證者的賭注在他們當選的期限結束之前凍結，並在之後的一個月內，以防發生新的爭端（即，發現由其中一個驗證者簽署的無效區塊）。之後，賭注將退還，並連同驗證者在此期間鑄造的硬幣和處理交易的費用的份額一起退還。

2.6.8. Election of validator "task groups". 整個全球驗證者集合（其中每個驗證者的存在次數等於其股份的多重性，否則驗證者可能會誘惑假設多個身份並將其賭注分散在它們之間）僅用於驗證新的主區塊。僅從選自全局驗證者集合的特別選定的驗證者子集驗證分片區塊，這些子集是如**2.6.7**所述選擇的全局驗證者集合。

[17]為每次驗證者選舉生成並使用新的密鑰對是有意義的。

每個分片定義的這些驗證者「子集」或「任務組」每小時輪換一次（實際上是每 2^{10} 個主區塊），而且提前一小時就已經知道，以便每個驗證者都知道它將需要驗證哪些分片，並可以為此做好準備（例如通過下載缺失的分片數據）。

用於選擇每個分片 (w, s) 的驗證者任務組的算法是一個具有決定性偽隨機性質的算法。它使用驗證者嵌入到每個主鏈區塊（通過閾值簽名生成的區塊）中的偽隨機數來創建一個隨機種子，然後為每個驗證者計算例如 $\text{HASH}(\text{CODE}(w).\,\text{CODE}(s).\mathit{validator\_id}.\mathit{rand\_seed})$。然後，驗證者按此哈希值進行排序，選擇前幾個驗證者，以便至少佔總驗證者股份的 $20/T$，並且由至少 5 個驗證者組成。

這個選擇可以由一個特殊的智能合約完成。在這種情況下，選擇算法可以通過 **2.1.21**中提到的投票機制輕鬆進行升級，而無需進行硬分叉。到目前為止提到的所有其他「常數」（例如 2^{19}，2^{10}，T，20 和 5）也都是可配置的參數。

2.6.9. Rotating priority order on each task group. 對於一個分片任務組中的成員，存在一定的「優先級」順序，這取決於前一個主分片區塊和（分片）區塊序列號的哈希。這個順序是通過生成並排序一些哈希值來確定的，如上所述。

當需要生成一個新的分片區塊時，選擇創建此區塊的分片任務組驗證者通常是優先級順序中的第一個。如果它無法創建該區塊，則第二或第三個驗證者可以進行創建。基本上，所有驗證者都可以提出他們的區塊候選人，但是應該以具有最高優先級的驗證者提出的候選人為勝利者，作為拜占庭容錯（BFT）共識協議的結果。

2.6.10. Propagation of shardchain block candidates. 因為分片任務組成員是提前一小時已知的，所以他們可以利用這段時間使用 TON 網路的一般機制（參見 3.3 節）構建專用的「分片驗證者多播覆蓋網路」。當需要生成一個新的分片區塊時——通常是在最近的主分片區塊傳播後一秒或兩秒——每個人都知道誰有最高的優先權生成下一個區塊（參見 2.6.9 節）。此驗證者將創建一個新的整合區塊候選人，要麼自己創建，要麼與整合者（參見 2.6.5 節）一起創建。驗證者必須檢查（驗證）此區塊候選人（特別是如果它是由某些整合者準備的），並使用其（驗證者）私鑰對其進行簽名。然後，區塊候選人使用預先安排的多播覆蓋網路（任務組創建自己的私有覆蓋網路，如 3.3 節所述，然後使用 3.3.15 節中描述的流式多播協議的一個版本）傳播到任務組的其餘成員。

一種真正的 BFT 方式是使用拜占庭散播協議，例如在 Honey Badger BFT [11] 中使用的協議：通過一個 $(N, 2N/3)$-擦除碼編碼區塊候選者，直

接將生成的數據的 $1/N$ 發送給該組的每個成員，並期望他們直接將其數據的部分向該組的所有其他成員進行多播。

然而，一種更快速和直接的方法（參見**3.3.15**），是將區塊候選者分成一系列簽名的 1KB 塊（「chunk」），通過 Reed-Solomon 或噴泉碼（例如 RaptorQ 代碼 [9] [14]）擴充它們的序列，開始將 chunks 傳輸到「多播網格」（即覆蓋網絡）中的鄰居，期望它們進一步傳播這些 chunks。一旦驗證者獲得足夠的 chunks 來從中重建區塊候選者，它就會簽署確認收據並將其通過其鄰居傳播到整個群體中。然後，其鄰居停止向其發送新的 chunks，但可能繼續向其發送這些 chunks 的（原始）簽名，認為此節點可以通過自己應用 Reed-Solomon 或噴泉碼（擁有所有必要的數據）生成後續 chunks，將它們與簽名結合並向其尚未準備好的鄰居傳播。

如果在刪除所有「不良」節點（回想一下，在拜占庭方式下，最多有三分之一的節點可以是「壞的」，即表現出任意惡意行為）之後，「多播網格」（覆蓋網絡）仍保持連接，則此算法將盡快傳播區塊候選者。

不僅指定的高優先級區塊創建者可以將其區塊候選者多播到整個群體。第二和第三優先級的驗證者可以開始多播其區塊候選者，無論是立即還是在未能收到最高優先級驗證者的區塊候選者之後。然而，通常只有具有最大優先級的區塊候選者將被所有驗證者（實際上是至少三分之二的任務組）簽署並作為新的 shardchain 區塊提交。

2.6.11. Validation of block candidates. 一旦驗證者接收到區塊候選者並檢查了其起始驗證者的簽名，接收驗證者將通過執行其中所有交易並檢查其結果是否與聲稱的結果相符來檢查此區塊候選者的有效性。從其他區塊鏈導入的所有消息都必須在匯編數據中支持適當的 Merkle 證明，否則該區塊候選者將被視為無效（如果已經向主鏈提交了此事實的證明，已經簽署此區塊候選者的驗證者可能會受到懲罰）。另一方面，如果發現區塊候選者是有效的，接收驗證者將對其進行簽名並將其簽名傳播給群體中的其他驗證者，通過「網格多播網絡」或直接網絡消息。

我們要強調的是，驗證者不需要訪問此或相鄰分片區塊的狀態，就可以檢查（彙總的）區塊候選的有效性。[18]這使得驗證可以非常快速地進行（無需磁盤訪問），並減輕了驗證者的計算和存儲負擔（特別是如果他們願意接受外部匯總器為創建區塊候選提供服務）。

2.6.12. Election of the next block candidate. 一旦一個區塊候選收集到了任務組驗證者賭注三分之二的有效性簽名，它就有資格作為下一個分片區塊提交。運行 BFT 協議以達成有關所選擇的區塊候選（可能有多個提

[18]唯一可能的例外是相鄰分片區塊的輸出隊列狀態，需要保證 **2.4.21**中描述的消息順序要求，因為在這種情況下 Merkle 證明的大小可能會變得過大。

出）的共識。所有「好的」驗證者都優先選擇此輪優先級最高的區塊候選。通過運行這個協議，該區塊將被至少三分之二（按賭注計算）的驗證者的簽名加強。這些簽名不僅證明了該區塊的有效性，而且證明了它是通過 BFT 協議選舉的。之後，該區塊（不包括匯總數據）與這些簽名結合，以確定性的方式進行序列化，並傳播到所有相關方。

2.6.13. Validators must keep the blocks they have signed. 在他們在任務組中的成員資格期間，以及之後至少一個小時（或者說 2^{10} 個區塊），預計驗證者將保留他們已簽署和提交的區塊。未能向其他驗證者提供已簽署的區塊可能會受到懲罰。

2.6.14. Propagating the headers and signatures of new shardchain blocks to all validators. 驗證者使用類似於為每個任務組創建的多播網絡的多播網絡，將新生成的分片區塊的標題和簽名傳播到全球驗證者集合中。

2.6.15. Generation of new masterchain blocks. 在所有（或幾乎所有）新的分片區塊被生成後，可以生成新的主區塊。過程基本上與分片區塊相同（參見 **2.6.12**），唯一的區別是必須有所有驗證者（或至少三分之二的驗證者）參與此過程。由於新的分片區塊的標題和簽名被傳播到所有驗證者，因此必須將每個分片區塊中最新區塊的哈希包含在新的主區塊中。一旦這些哈希被提交到主區塊中，外部觀察者和其他分片區塊可以將新的分片區塊視為已提交和不可變的（參見 **2.1.13**）。

2.6.16. Validators must keep the state of masterchain. 主區塊和分片區塊之間一個值得注意的區別是，所有驗證者都需要跟踪主區塊的狀態，而不能依賴於匯總數據。這很重要，因為驗證者任務組的知識是從主區塊狀態中派生的。

2.6.17. Shardchain blocks are generated and propagated in parallel. 通常，每個驗證者都是幾個分片區塊任務組的成員；它們的數量（因此也是驗證者的負載）大約與驗證者的股份成比例。這意味著驗證者並行運行幾個新的分片區塊生成協議的實例。

2.6.18. Mitigation of block retention attacks. 因為驗證者的總集合只看到新的分片區塊的標題和簽名後即會將其哈希插入主區塊，所以發生生成新區塊的驗證者共謀並試圖避免完整地發布新區塊的概率很小。這將導致鄰近分片區塊的驗證者無法創建新區塊，因為一旦新區塊的哈希已被提交到主區塊中，它們必須至少了解新區塊的輸出消息隊列。

為了緩解這個問題，新區塊必須收集來自其他驗證者（例如，相鄰分片區塊任務組的聯合的三分之二）的簽名，證明這些驗證者確實擁有此區塊

的副本，如果需要，願意將它們發送給任何其他驗證者。只有在呈現這些簽名之後，新區塊的哈希才能包含在主區塊中。

2.6.19. Masterchain blocks are generated later than shardchain blocks. 主區塊和分片區塊的生成大約每五秒鐘進行一次。然而，盡管所有分片區塊中新區塊的生成基本上是在同一時間運行（通常是由於發布新主區塊觸發），但新主區塊的生成被刻意延遲，以便允許在主區塊中包含新生成的分片區塊的哈希。

2.6.20. Slow validators may receive lower rewards. 如果驗證者速度較慢，它可能無法驗證新的區塊候選者，並且可能會在其未參與的情況下收集到所需的三分之二簽名以提交新區塊。在這種情況下，它將獲得與此區塊相關的較低份額的獎勵。

這激勵驗證者優化其硬件、軟件和網絡連接，以盡可能快地處理用戶交易。

然而，如果驗證者未能在提交區塊之前簽署，其簽名可能會包含在下一個區塊中，然後仍會給予此驗證者一部分獎勵（根據生成了多少區塊呈指數下降，例如如果驗證者晚了 k 個區塊，還會給予 0.9^k 的獎勵）。

2.6.21. "Depth" of validator signatures. 通常，當驗證者簽署一個區塊時，該簽名只證明該區塊的相對有效性：只有在該分片區塊和其他分片區塊的所有先前區塊都是有效的情況下，該區塊才是有效的。如果之前的區塊中提交了無效數據，驗證者不能因此受到懲罰。

然而，驗證者對區塊的簽名具有一個整數參數，稱為"深度"。如果它不為零，這意味著驗證者還會聲明指定數量的先前區塊（相對）有效。這是"緩慢"或"暫時離線"的驗證者追趕進度並簽署已經提交但沒有他們簽名的區塊的一種方法。然後，區塊獎勵的一部分仍將分配給他們（參見 **2.6.20**）。

2.6.22. Validators are responsible for *relative* validity of signed shardchain blocks; absolute validity follows. 我們再次強調，驗證者對分片區塊 B 的簽名僅證明該區塊的相對有效性（或者如果簽名具有「深度」d，那麼也可能證明 d 個先前區塊的相對有效性，參見 **2.6.21**；但這對以下討論沒有太大影響）。換句話說，驗證者斷言分片區塊的下一個狀態 s' 是通過應用 **2.2.6**中描述的區塊評估函數 ev_block 從先前狀態 s 得到的：

$$s' = ev\_block(B)(s) \tag{26}$$

這樣，簽署區塊 B 的驗證者如果原始狀態 s 被證明是「不正確的」（例如，由於先前某個區塊無效），則不會受到懲罰。如果發現了一個相對無效

的區塊，漁夫（參見 **2.6.4**）應該投訴。PoS 系統作為一個整體努力使每個區塊都是相對有效的，而不是遞歸（或絕對）有效的。但是，注意，如果區塊鏈中的所有區塊都是相對有效的，那麼它們全部以及整個區塊鏈都是絕對有效的；這個語句可以通過對區塊鏈長度進行數學歸納來輕鬆證明。這樣，易於驗證的相對有效性斷言與區塊一起證明了整個區塊鏈的絕對有效性，這是更強大的斷言。

請注意，簽署區塊 B 的驗證者斷言，給定原始狀態 s，該區塊是有效的（即，(26)的結果不是值 $\perp$，這表示無法計算下一個狀態）。這樣，驗證者必須對在評估(26)期間訪問的原始狀態的單元進行最小的形式檢查。

例如，假設從提交到區塊的交易中訪問的帳戶的原始餘額所在的單元應該包含 8 或 16 個原始字節，但實際上是零。那麼，簡單地無法從該單元檢索原始餘額，並且在嘗試處理該區塊時會發生「未處理的異常」。在這種情況下，驗證者不應該簽署此類區塊，否則就會受到懲罰。

2.6.23. Signing masterchain blocks. 對於主鏈區塊，情況略有不同：簽署主鏈區塊的驗證者不僅斷言該區塊的相對有效性，還斷言所有先前的區塊都是相對有效的，直到這個驗證者承擔責任時的第一個區塊（但不包括更早的區塊）。

2.6.24. The total number of validators. 到目前為止，根據所描述的系統，選舉的驗證者總數的上限 T 不能超過幾百或一千，因為預期所有驗證者都會參與 BFT 共識協議以創建每個新的主鏈區塊，而這種協議能否擴展到數千個參與者尚不清楚。更重要的是，主鏈區塊必須收集至少所有驗證者的三分之二（按權益計）的簽名，並且這些簽名必須包含在新區塊中（否則，系統中的其他節點沒有理由信任新區塊，除非他們自己對其進行驗證）。如果每個主鏈區塊必須包含超過一千個驗證者簽名，這將意味著每個主鏈區塊中有更多的數據需要被所有全節點存儲並在網絡中傳播，以及需要花費更多的處理能力來檢查這些簽名（在 PoS 系統中，全節點不需要自己驗證區塊，而是需要檢查驗證者的簽名）。

儘管將 T 限制為一千個驗證者似乎足夠用於部署 TON 區塊鏈的第一階段，但必須為未來的增長做出準備，當分片鏈的總數變得如此之大，以至於數百個驗證者將不足以處理它們。為此，我們引入了一個額外的可配置參數 $T' \leq T$（最初等於 T），並且僅期望由持股數最高的前 T' 個當選驗證者創建並簽署新的主鏈區塊。

2.6.25. Decentralization of the system. 有人可能會懷疑，像 TON 區塊鏈這樣依賴於 $T \approx 1000$ 個驗證者創建所有分片鏈和主鏈區塊的 PoS 系統，與傳統的 PoW 區塊鏈（如比特幣或以太坊）相比，可能變得“過於集

中”，在這些區塊鏈中，每個人（原則上）都可以挖掘新的區塊，而不需要對總挖礦者數量進行明確的上限。

然而，像比特幣和以太坊這樣的 PoW 區塊鏈，目前需要大量的計算能力（高“哈希率”）才能挖掘出新的區塊，並且有可觀的成功機率。因此，新區塊的挖掘往往集中在少數幾個大型參與者手中，他們投資大量資金在充滿了針對挖礦進行優化的自定義硬件的數據中心中，或是集中協調更多無法自行提供足夠“哈希率”的人的努力的幾個大型挖礦池。

因此，截至 2017 年，超過 75%的新以太坊或比特幣區塊由不到十個挖礦者產生。事實上，最大的兩個以太坊挖礦池一起生產超過一半的新區塊！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集中化。

有一點值得注意的是，成為 TON 區塊鏈驗證者所需的投資，即購買硬件（例如數個高性能伺服器）和股份（如果需要，可以通過提名者池輕鬆收集；參見**2.6.3**節），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**節中的參數 L 將迫使提名者不加入最大的「挖礦池」（即積累了最大股份的驗證者），而是尋找目前接受提名者資金的較小驗證者，甚至創建新的驗證者，因為這將允許更高比例的驗證者的---並且也是提名者的---股份被使用，因此從挖礦中獲得更大的獎勵。通過這種方式，TON 權益證明系統實際上鼓勵分散化（創建和使用更多驗證者），並懲罰集中化。

2.6.26. Relative reliability of a block. 一個區塊的（相對）可靠性，就是簽署此區塊的所有驗證者的總股份。換句話說，如果此區塊被證明是無效的，某些角色將會損失的金額就是這個總股份。如果關注的是低於區塊可靠性的價值轉移交易，則可以認為它們足夠安全。從這個意義上講，相對可靠性是外部觀察者對特定區塊可以信任的度量。

請注意，我們談論的是一個區塊的相對可靠性，因為它是一個保證，只要前一個區塊和所有其他分片鏈的區塊參考的都是有效的，那麼這個區塊就是有效的（參見**2.6.22**節）。

一個區塊的相對可靠度在其確認後可能會提高 - 例如，當被延遲的驗證者的簽名被添加時（參見**2.6.21**）。另一方面，如果這些驗證者中的一個由於其與某些其他區塊相關的不良行為而失去部分或全部股份，則該區塊的相對可靠度可能會降低。

2.6.27. “Strengthening” the blockchain. 提供激勵以鼓勵驗證者盡可能增加區塊的相對可靠度是非常重要的。其中一種方法是為驗證者分配小獎勵，以便對其他分片區塊添加簽名。甚至那些“未來”的驗證者，由於押金不足以進入前 T 名驗證者，因此不能被包括在全球驗證者集合中（參見**2.6.7**），也可能參與此活動（如果他們同意保持押金凍結，而不是在失去選舉後撤回押金）。此類“未來”的驗證者還可以兼顧漁民的角色（參

見**2.6.4**）：如果他們必須檢查某些區塊的有效性，那麼他們可以選擇報告無效區塊並收集相關的獎勵。

2.6.28. Recursive reliability of a block. 還可以定義一個區塊的遞迴可靠度，其為其相對可靠度和其所參照的所有區塊的遞迴可靠度（即主鏈區塊、前一個分片區塊以及某些相鄰分片區塊的區塊）的最小值。換句話說，如果該區塊被證明是無效的，要麼是因為它本身無效，要麼是因為它依賴的某個區塊無效，那麼至少會有這個數量的錢會被某個人損失。如果真的不確定是否信任區塊中的某個特定交易，則應計算此區塊的遞迴可靠度，而不僅僅是相對可靠度。

在計算遞迴可靠度時，過度往回追朔是沒有意義的，因為如果我們往回追朔太遠，我們會看到由押金已經被解凍並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這麼舊的區塊（即使用可配置參數的當前值，如果創建時間超過兩個月），並從這些區塊開始創建分支或使用“垂直區塊鏈”來修正它們（參見**2.1.17**），即使它們最終被證明是無效的。我們假設兩個月的時間足以發現和報告任何無效的區塊，因此如果在這段時間內沒有對某個區塊提出異議，那麼它就不太可能被挑戰了。

2.6.29. Consequence of Proof-of-Stake for light nodes. TON 區塊鏈使用的 PoS 方法的一個重要結果是，TON 區塊鏈的輕節點（運行輕量級客戶端軟件）不需要下載所有分片鏈或甚至主鏈區塊的“標頭”，就可以通過自己檢查由全節點提供的默克爾證明的有效性，作為對其查詢的回答。

事實上，因為最新的分片鏈區塊哈希包含在主鏈區塊中，所以全節點可以輕鬆地提供一個默克爾證明，證明從已知的主鏈區塊哈希開始，給定的分片鏈區塊是有效的。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），它（或至少其哈希）可能已經內建到客戶端軟件中，以及其之後的每個月僅需了解一個主鏈區塊，其中宣布了新當選的驗證者組，因為該區塊將由上一組驗證者簽署。從那個時候開始，它可以獲取幾個最新的主鏈區塊，或至少它們的標頭和驗證者簽名，並將它們用作檢查全節點提供的默克爾證明的基礎。

2.7 Splitting and Merging Shardchains

TON 區塊鏈最具特色和獨特的功能之一是當負載過高時，自動將一個分片鏈分成兩部分，並在負載減輕時將它們合併（參見**2.1.10**）。由於這種功能的獨特性和對整個項目可擴展性的重要性，我們必須對其進行詳細討論。

2.7.1. Shard configuration. 回想一下，任何一個時刻，每個工作鏈 w 都被分成一個或多個分片鏈 (w, s)（參見**2.1.8**）。這些分片鏈可以用一棵二

叉樹的葉子來表示，其根為 $(w, \emptyset)$，每個非葉子節點 (w, s) 有兩個子節點 $(w, s.0)$ 和 $(w, s.1)$。通過這種方式，屬於工作鏈 w 的每個賬戶都被分配到正好一個分片鏈上，並且每個知道當前分片鏈配置的人都可以確定包含賬戶 $account\_id$ 的分片 (w, s)：它是唯一的一個二進制字符串 s 是 $account\_id$ 的前綴。

分片配置——即這個分片二叉樹或對於給定的 w 的所有活動 (w, s) 的集合（對應於分片二叉樹的葉子節點）——是主鏈狀態的一部分，並且對於跟踪主鏈的每個人都是可用的。[19]

2.7.2. Most recent shard configuration and state. 回想一下，最新的分片鏈區塊的哈希值包含在每個主鏈區塊中。這些哈希值按照分片二叉樹的方式組織起來（實際上，每個工作鏈都有一棵分片二叉樹）。通過這種方式，每個主鏈區塊都包含了最新的分片配置。

2.7.3. Announcing and performing changes in the shard configuration. 分片配置可以通過兩種方式進行更改：一個分片 (w, s) 可以被分裂成兩個分片 $(w, s.0)$ 和 $(w, s.1)$，或者兩個“兄弟”分片 $(w, s.0)$ 和 $(w, s.1)$ 可以合併成一個分片 (w, s)。

這些分裂/合併操作是提前幾個區塊（例如 2^6 個區塊；這是一個可配置的參數）宣布的，首先在相應分片鏈區塊的“標頭”中，然後在引用這些分片鏈區塊的主鏈區塊中宣布。這種提前的宣布對於所有相關方準備計劃中的更改（例如構建一個覆蓋多播網絡來分發新的新創分片鏈的區塊，如**3.3**所討論的）是必要的。然後進行更改，首先是（分裂的情況下是）分片鏈區塊的（標頭中的）更改（對於合併，兩個分片鏈的區塊都應該提交更改），然後傳播到主鏈區塊。通過這種方式，主鏈區塊不僅定義了它的創建之前最新的分片配置，而且還定義了下一個即將到來的分片配置。

2.7.4. Validator task groups for new shardchains. 回想一下，每個分片，即每個分片鏈，通常被指定一個專門創建和驗證相應分片鏈中新區塊的驗證者子集（驗證者任務組）（參見**2.6.8**）。這些任務組在一定時間內（大約一小時左右）進行選舉，並且在此期間的某個時間（也大約一小時左右）已知，並且在此期間內是不可變的。[20]

然而，由於分裂/合併操作，實際的分片配置可能在此期間內發生變化。必須將任務組分配給新創建的分片。分配方法如下：

請注意，任何活動分片 (w, s) 都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，或者它將是原始分片 (w, s') 的一個子樹的根，

[19] 實際上，分片配置完全由最後一個主鏈區塊決定；這簡化了獲取分片配置的過程。

[20] 除非一些驗證者因簽署無效區塊而被暫時或永久禁止，那麼它們將自動從所有任務組中排除。

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') 的任務組作為新分片 (w, s) 的任務組。在後一種情況下，新分片 (w, s) 的任務組將是所有原始分片 (w, s') 的任務組的聯合，這些分片是在分片樹中是 (w, s) 的後代。

通過這種方式，每個活動分片 (w, s) 都被分配了一個明確的驗證者子集（任務組）。當一個分片被分裂時，兩個子分片都從原始分片繼承整個任務組。當兩個分片被合併時，它們的任務組也會被合併。

任何跟踪主鏈狀態的人都可以計算每個活動分片的驗證者任務組。

2.7.5. Limit on split/merge operations during the period of responsibility of original task groups. 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，必須對分裂/合併操作施加一定的限制；否則，如果原始分片快速分裂成 2^k 個新分片，原始任務組可能會同時驗證 2^k 個分片鏈。

因此，必須將一些額外的條件納入考慮，以確保分裂和合併操作不會導致驗證者子集（任務組）超負荷。

這是通過對活動分片配置可以從原始分片配置（用於選擇當前任務組的驗證者）中移除的距離進行限制來實現的。例如，如果 s' 是 s 的前身（即 s' 是二進制字符串 s 的前綴），那麼可以要求從活動分片 (w, s) 到原始分片 (w, s') 的距離不得超過 3；如果 s' 是 s 的後繼（即 s 是二進制字符串 s' 的前綴），那麼該距離不得超過 2。否則，不允許進行分裂或合併操作。

粗略地說，這是在一定時間內對分片進行多次分裂（例如三次）或合併（例如兩次）次數進行限制。此外，在通過合併或分裂創建分片後，該分片在一段時間內（幾個塊）無法重新配置。

2.7.6. Determining the necessity of split operations. 對於一個分片鏈，分裂操作是由某些正式條件觸發的（例如，如果連續 64 個塊中分片鏈塊的容量至少為 90%）。這些條件由分片鏈任務組監視。如果條件滿足，首先在新的分片鏈塊標題中包含一個“分裂準備”標誌（並傳播到引用此分片鏈塊的主分片鏈塊）。然後，幾個塊之後，將“分裂提交”標誌包含在分片鏈塊的標題中（並傳播到下一個主分片鏈塊）。

2.7.7. Performing split operations. 在分片鏈 (w, s) 的區塊 B 中包含“分裂提交”標誌之後，就不會有後續的區塊 B'。相反，會創建兩個分片鏈 $(w, s.0)$ 和 $(w, s.1)$ 的區塊 B'_0 和 B'_1，分別引用區塊 B 作為它們的前一個區塊（它們的標頭都會指示該分片剛剛被分裂）。下一個主鏈區塊將包含新分片鏈 B'_0 和 B'_1 的哈希值；它不允許包含分片鏈 (w, s) 的新區塊 B' 的哈希值，因為“分裂提交”事件已經被提交到了前一個主鏈區塊。

請注意，這兩個新的分片鏈將由與舊的相同的驗證者任務組進行驗證，因此它們將自動具有其狀態的副本。從無限分片範式的角度來看，狀態分割操作本身相當簡單（參見**2.5.2**）。

2.7.8. Determining the necessity of merge operations. 分片合併操作的必要性也是通過某些形式條件（例如，如果在 64 個連續的區塊中，兩個兄弟分片鏈的區塊大小之和不超過最大區塊大小的 60%），來檢測的。這些形式條件還應該考慮到這些區塊花費的總 gas 量，並將其與當前的區塊 gas 限制進行比較，否則這些區塊可能會很小，因為存在一些計算密集型交易，防止更多交易被包含。

這些條件由兄弟分片鏈 $(w, s.0)$ 和 $(w, s.1)$ 的驗證者任務組監視。請注意，兄弟分片鏈必定是相對於超立方路由（參見**2.4.19**）的鄰居，因此任何一個分片鏈的驗證者任務組都會在某種程度上監視兄弟分片鏈。

當滿足這些條件時，任何一個驗證者子組都可以向另一個子組發送一個特殊消息，建議它們合併。然後，它們結合成一個臨時的“合併驗證者任務組”，具有結合成員，能夠運行 BFT 共識算法並在必要時傳播區塊更新和區塊候選。

如果它們達成共識，認為有必要且已準備好進行合併，就會在每個分片鏈的某些區塊標頭中提交“合併準備”標誌，以及兄弟任務組中至少三分之二的驗證者的簽名（並傳播到下一個主分片鏈區塊，以便每個人都能為即將到來的重新配置做好準備）。然而，它們會繼續為一些預定的區塊創建獨立的分片鏈區塊。

2.7.9. Performing merge operations. 之後，當原始任務組的聯合已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），它們會在其分片鏈區塊的標頭中提交“合併提交”標誌（此事件會傳播到下一個主分片鏈區塊），並停止在獨立的分片鏈中創建新區塊（一旦出現合併提交標誌，創建獨立分片鏈區塊就被禁止）。取而代之的是，由原始任務組的聯合創建了一個合併分片鏈區塊，它在其“標頭”中引用了它的“前一個區塊”。這反映在下一個主分片鏈區塊中，它將包含新創建的合併分片鏈區塊的哈希值。之後，合併任務組會繼續在合併分片鏈中創建區塊。

2.8 Classification of Blockchain Projects

在將 TON 區塊鏈與現有和提出的區塊鏈項目進行比較之前，我們將通過引入足夠通用的區塊鏈項目分類來結束我們對 TON 區塊鏈的簡要討論。基於這種分類的特定區塊鏈項目的比較被推遲到 **2.9**中。

2.8.1. Classification of blockchain projects. 作為第一步，我們提出了一些區塊鏈（即區塊鏈項目）的分類標準。任何此類分類都有些不完整和膚淺，因為它必須忽略一些正在考慮中的項目的最具體和獨特的特徵。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項目地圖的必要第一步。

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**)
- 共識算法：PoS（權益證明）vs. PoW（工作量證明）(參見 **2.8.3**)
- 對於 PoS 系統，使用的精確區塊生成、驗證和共識算法（主要有 DPOS vs. BFT 兩個選項；參見 **2.8.4**）
- 是否支持“任意”（圖靈完整）智能合約（參見 **2.8.6**）

多區塊鏈系統具有額外的分類標準（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**）。還有聯盟區塊鏈（參見 **2.8.10**）。
- 是否有內部或外部的主鏈（參見 **2.8.11**）
- 是否支持原生的分片（參見 **2.8.12**）。靜態或動態分片（參見 **2.8.13**）。
- 成員區塊鏈之間的交互作用：鬆散耦合和緊密耦合系統（參見 **2.8.14**）

2.8.2. Single-blockchain vs. multi-blockchain projects. 第一個分類標準是系統中區塊鏈的數量。最古老和最簡單的項目由一個單一區塊鏈（簡稱“單鏈項目”）組成；更複雜的項目使用（或者計劃使用）多個區塊鏈(“多鏈項目”)。

單鏈項目通常更簡單且經過更好的測試；它們已經經受住了時間的考驗。它們的主要缺點是低性能，或者至少在一般系統中，交易吞吐量只有十（比特幣）至不到一百[21]（以太坊）每秒。一些特殊系統（例如 Bitshares）能夠以每秒數萬個專門交易的速度進行處理，但代價是需要將區塊鏈狀態放入內存中，並將處理限制為一個預定義的特殊交易集，這些交易然後由用 C++ 等語言編寫的高度優化代碼執行（這裡沒有虛擬機）。

多鏈項目承諾了每個人都渴望的可擴展性。它們可以支持更大的總狀態和更多的交易量，但代價是使項目變得更加複雜，實現更具挑戰性。因此，已經運行的多鏈項目很少，但大多數提出的項目都是多鏈的。我們相信未來屬於多鏈項目。

[21]目前大約為 15。然而，正在計劃一些升級，使以太坊的交易吞吐量增加數倍。

2.8.3. Creating and validating blocks: Proof-of-Work vs. Proof-of-Stake. 另一個重要的區分是用於創建和傳播新區塊、檢查它們的有效性以及在它們出現多個分支時選擇其中一個的算法和協議。

最常見的兩種範式是“工作量證明（Proof-of-Work，PoW）”和“權益證明（Proof-of-Stake，PoS）”。工作量證明方法通常允許任何節點在其他競爭對手解決相同問題之前，通過解決一個毫無用途的計算問題（通常涉及大量散列計算）來創建（“挖掘”）新區塊（並獲得與挖掘區塊相關的一些獎勵）。在分支的情況下（例如，如果兩個節點發布了兩個相互獨立但有效的區塊來跟隨上一個區塊），最長的分支將獲勝。這樣，區塊鏈不可變性的保證是基於生成區塊鏈所花費的“工作”（計算資源）的量：任何希望創建該區塊鏈的分支的人都需要重新完成這些工作，以創建已提交區塊的替代版本。為此，需要控制創建新區塊所花費的總計算資源的 50%以上，否則替代分支將具有指數級低的成為最長分支的機會。

Proof-of-Stake 方法基於一些特殊節點（即驗證者）投入的大量加密貨幣（即 *stakes*），聲明它們已經檢查（即驗證）了一些區塊，並且發現它們是正確的。驗證者對區塊進行簽名，並因此獲得一些小的獎勵。但是，如果驗證者被發現簽署了不正確的區塊，並且有相應的證明，它的部分或全部賭注將被沒收。以這種方式，區塊鏈的有效性和不可變性的保證是由驗證者對區塊鏈有效性的總賭注給出的。

Proof-of-Stake 方法在某些方面更自然，因為它激勵驗證者（代替 PoW 礦工）執行有用的計算（特別是通過執行在區塊中列出的所有交易來檢查或創建新區塊），而不是計算毫無意義的哈希。這樣，驗證者會購買更適合處理用戶交易的硬件，以便接收與這些交易相關的獎勵，從整個系統的角度來看，這似乎是一個非常有用的投資。

然而，Proof-of-Stake 系統在實現上有些具有挑戰性，因為必須提供許多罕見但可能存在的情況。例如，一些惡意驗證者可能會合謀破壞系統以獲取一些利潤（例如，通過更改自己的加密貨幣餘額）。這導致了一些非常複雜的博弈理論問題。

總之，Proof-of-Stake 更自然、更有前途，特別是對於多鏈項目（因為如果存在多個區塊鏈，Proof-of-Work 將需要過高的計算資源），但必須更加仔細地思考和實施。目前運行的大多數區塊鏈項目，特別是最古老的項目（例如比特幣和至少最初的以太坊），使用 Proof-of-Work。

2.8.4. Variants of Proof-of-Stake. DPOS vs. BFT. Proof-of-Work 算法非常相似，主要區別在於挖掘新區塊需要計算的哈希函數，而 Proof-of-Stake 算法的可能性更多。它們值得有自己的子分類。

基本上，必須回答關於 Proof-of-Stake 算法的以下問題：

- 誰可以產生（“挖掘”）新區塊——任何完整節點，還是只有（相對）

小的一部分驗證者成員？（大多數 PoS 系統要求新區塊由幾個指定的驗證者之一生成和簽名。）

- 驗證者是否通過其簽名保證區塊的有效性，還是所有完整節點都預期自己驗證所有區塊？（可擴展的 PoS 系統必須依賴於驗證者的簽名，而不是要求所有節點驗證所有區塊的所有區塊鏈。）
- 是否有指定的下一個區塊鏈區塊的生產者，在事先已知，以便沒有其他人可以代替生產該區塊？
- 新創建的區塊最初是否由一個驗證者（其生產者）簽署，還是必須從一開始就收集大多數驗證者的簽名？

雖然根據這些問題的回答似乎有 2^4 種可能的 PoS 算法類型，但在實踐中，區分主要歸結為兩種 PoS 的方法。事實上，大多數現代 PoS 算法都設計用於可擴展的多鏈系統中，以相同的方式回答前兩個問題：只有驗證者可以生成新區塊，並且它們保證區塊的有效性，而不需要所有全節點自行檢查所有區塊的有效性。

至於最後兩個問題，它們的答案相互關聯，基本上只有兩種基本選擇：

- 委託型 *PoS (DPOS)*：每個區塊都有一個普遍已知的指定生成者；沒有其他人可以生成該區塊；新區塊最初僅由其生成驗證者簽署。
- 拜占庭容錯 *(BFT) PoS* 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的區塊；從多個建議的候選區塊中選擇實際的下一個區塊，在被發布到其他節點之前，必須由大多數驗證者進行驗證和簽署，這是通過某種形式的拜占庭容錯共識協議實現的。

2.8.5. Comparison of DPOS and BFT PoS. BFT 方法的優點是，新生成的區塊從一開始就擁有大多數驗證者的簽名，證明了其有效性。另一個優點是，如果大多數驗證者正確執行 BFT 共識協議，就不會出現分叉。另一方面，BFT 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，不能太頻繁地生成區塊。這就是為什麼我們期望 TON 區塊鏈（從這個分類的角度來看，它是一個 BFT 項目）每五秒產生一個區塊。在實踐中，如果驗證者分散在全球，這個時間間隔可能會減少到 2-3 秒（但我們不保證）。

DPOS 算法的優點是非常簡單直接。它可以很快地生成新區塊——比如說每兩秒一個，甚至每秒一個[22]，因為它依賴於事先已知的指定區塊生產者。

[22]一些人甚至聲稱 DPOS 區塊生成時間只有半秒，但如果驗證者分散在幾個大陸上，這似乎不現實。

然而，DPOS 要求所有節點——或者至少所有驗證者——驗證接收到的所有區塊，因為生成並簽署新區塊的驗證者不僅確認該區塊的相對有效性，還確認了其所引用的上一個區塊以及整個區塊鏈中更早的所有區塊的有效性（可能一直到當前驗證者子集負責期的開始）。當前驗證者子集上有預定的順序，因此對於每個區塊都有一個指定的生產者（即預期生成該區塊的驗證者）；這些指定的生產者以輪流的方式輪換。通過這種方式，一個區塊一開始只被其生產驗證者簽署；然後，當下一個區塊被挖掘出來，它的生產者選擇引用這個區塊而不是其中一個前驅（否則它的區塊將位於更短的鏈上，可能在將來失去“最長分叉”的競爭），下一個區塊的簽名實質上也是對前一個區塊的額外簽名。通過這種方式，新區塊逐漸收集更多驗證者的簽名——比如，在需要生成接下來的二十個區塊的時間內收集二十個簽名。一個全節點將需要等待這二十個簽名，或者從足夠確認的區塊（比如，從二十個區塊之前）開始自己驗證該區塊，這可能並不容易。

DPOS 算法的明顯缺點是，與 BFT 算法相比，新區塊（及其中提交的交易）只有在挖掘了二十個以上的區塊後才能達到同樣的信任水平（如**2.6.28**中所討論的“遞歸可靠性”）。另一個缺點是，DPOS 對於切換到其他分叉使用“最長分叉勝出”的方法；如果至少一些生產者無法在我們感興趣的區塊之後產生後續區塊（或者由於網絡分割或複雜攻擊而無法觀察到這些區塊），這使得分叉相當可能發生。

我們認為，BFT 方法雖然在實現上更複雜，並且需要比 DPOS 更長的區塊間隔時間，但對於“緊密耦合”（參見**2.8.14**）的多鏈系統更為適合，因為其他區塊鏈可以在看到新區塊中的已確認交易（例如，生成對它們有用的消息）後幾乎立即開始運作，而不必等待二十個有效性確認（即下一個二十個區塊），或者等待下一個六個區塊，以確保沒有分叉出現並自行驗證新區塊（在可擴展的多鏈系統中驗證其他區塊鏈的區塊可能變得不可行）。因此，它們可以實現可擴展性，同時保持高可靠性和可用性（參見**2.8.12**）。

另一方面，在“鬆散耦合”的多鏈系統中，DPOS 可能是一個不錯的選擇，其中不需要快速的區塊鏈之間互動，例如，如果每個區塊鏈（“工作鏈”）代表一個獨立的分散式交換所，並且區塊鏈之間的互動僅限於從一個工作鏈轉移代幣到另一個工作鏈（或者更確切地說，以接近 1:1 的匯率交易一個工作鏈中的替代幣以換取另一個工作鏈中的替代幣）。這實際上是 BitShares 項目所採用的方式，並且相當成功地使用了 DPOS。

總結一下，雖然 DPOS 可以更快速地生成新區塊和包含交易（區塊之間的間隔更小），但這些交易要達到在其他區塊鏈和離線應用程序中使用所需的“已確認”和“不可變”的信任級別，需要的時間比 BFT 系統要慢得多，比如說 30 秒[23] 而不是五秒。更快速的交易包含並不意味著更快速的

[23]例如，EOS 是迄今提出的最佳 DPOS 項目之一，承諾 45 秒確認和區塊鏈間互動延遲

交易確認。如果需要快速的區塊鏈間交互，這可能會成為一個巨大的問題。在這種情況下，必須放棄 DPOS，改為使用 BFT PoS。

2.8.6. Support for Turing-complete code in transactions, i.e., essentially arbitrary smart contracts. 區塊鏈項目通常在其區塊中收集一些交易，這些交易以被認為有用的方式改變區塊鏈狀態（例如，從一個帳戶轉移一定量的加密貨幣到另一個帳戶）。某些區塊鏈項目可能僅允許某些特定預定義類型的交易（例如，當提供正確的簽名時，從一個帳戶向另一個帳戶轉移價值）。其他項目可能支持某種有限形式的交易腳本。最後，一些區塊鏈支持在交易中執行任意複雜的代碼，使系統（至少原則上）能夠支持任意應用，前提是系統的性能足夠。這通常與“圖靈完備虛擬機和腳本語言”（意味著可以使用任何其他計算語言編寫的程序可以被重新編寫以在區塊鏈內執行），以及“智能合約”（這些是駐留在區塊鏈中的程序）有關。

當然，支持任意智能合約使系統真正具有靈活性。另一方面，這種靈活性是有代價的：這些智能合約的代碼必須在某些虛擬機上執行，並且每當有人想要創建或驗證一個區塊中的交易時，都必須這樣做。這將使系統的性能下降，相較於一組預定義且不可變的簡單交易類型，後者可以通過在諸如 C ++（而不是某些虛擬機）的語言中實現其處理以進行優化。

最終，對於任何通用區塊鏈項目來說，支持圖靈完備的智能合約似乎是理想的；否則，區塊鏈項目的設計者必須事先決定其區塊鏈將用於哪些應用。實際上，比特幣區塊鏈不支持智能合約，這是一個新的區塊鏈項目以太坊必須被創建的主要原因。

在一個（異質性的；參見**2.8.8**）多鏈系統中，可以在某些區塊鏈（即工作鏈）中支持圖靈完備的智能合約，在其他區塊鏈中支持一小組預定義的高度優化的交易，這樣可以得到“兩全其美”的效果。

2.8.7. Classification of multichain systems. 到目前為止，這些分類對單鏈和多鏈系統都是有效的。然而，多鏈系統還有幾個更多的分類標準，反映了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. Blockchain types: homogeneous and heterogeneous systems. 在多鏈系統中，所有區塊鏈可能基本上是相同類型並且具有相同的規則（即使用相同的交易格式、用於執行智能合約代碼的相同虛擬機、共享相同的加密貨幣等），並且這種相似性被明確地利用，但每個區塊鏈中的數據不同。在這種情況下，我們稱該系統為同質的。否則，不同的區塊鏈（在這種情況下通常稱為工作鏈）可能具有不同的“規則”。然後，我們說該系統是異質的。

（參見 [5] 中的“交易確認”和“區塊鏈間通信的延遲”部分）。

2.8.9. Mixed heterogeneous-homogeneous systems. 有時我們會有一個混合系統，其中存在幾個區塊鏈類型或規則集，但有許多具有相同規則的區塊鏈存在，並且明確利用了這一事實。然後這是一個混合的異質-同質系統。據我們所知，TON 區塊鏈是這種系統的唯一例子。

2.8.10. Heterogeneous systems with several workchains having the same rules, or *confederations*. 在某些情況下，在異質系統中可能存在具有相同規則的多個區塊鏈（工作鏈），但它們之間的交互與具有不同規則的區塊鏈之間的交互相同（即，它們的相似性不是明確利用）。即使它們似乎使用“相同”的加密貨幣，實際上它們使用不同的“替代幣”（加密貨幣的獨立化身）。有時甚至可以有某些機制以接近 $1:1$ 的比率轉換這些替代幣。但是，在我們看來，這並不使系統同質化；它仍然是異質的。我們稱這樣的具有相同規則的異質工作鏈集合為聯邦。

儘管創建具有相同規則的多個工作鏈（即聯邦）的異質系統可能是構建可擴展系統的一種廉價方式，但這種方法也有很多缺點。本質上，如果某人在許多具有相同規則的工作鏈中托管一個大型項目，她不會獲得一個大型項目，而是獲得了許多該項目的小實例。這就像擁有一個聊天應用程序（或遊戲），該應用程序允許在任何聊天（或遊戲）房間中最多有 50 個成員，但在需要時通過創建新的房間來容納更多用戶來“擴展”。因此，很多用戶可以參與聊天或遊戲，但我們能說這樣的系統真正具有可擴展性嗎？

2.8.11. Presence of a masterchain, external or internal. 有時，多鏈項目有一個顯著的“主鏈”（有時稱為“控制區塊鏈”），例如用於存儲系統的整體配置（所有活動區塊鏈的集合，或者更準確地說是工作鏈）、當前的驗證者集合（對於 PoS 系統）等等。有時其他區塊鏈與主鏈“綁定”，例如通過將它們的最新區塊的哈希提交到主鏈中（這也是 TON 區塊鏈所做的事情）。

在某些情況下，主鏈是“外部”的，這意味著它不是該項目的一部分，而是一個先前存在的區塊鏈，最初與新項目的使用完全無關且對其無關注。例如，可以嘗試使用以太坊區塊鏈作為外部項目的主鏈，並發布特殊的智能合約到以太坊區塊鏈中以此為目的（例如，選舉和懲罰驗證者）。

2.8.12. Sharding support. 一些區塊鏈項目（或系統）具有原生支持分片的功能，這意味著幾個（必須是同質的；參見**2.8.8**）區塊鏈被視為一個（從高層次的角度看）虛擬區塊鏈的分片。例如，可以創建 256 個具有相同規則的分片區塊鏈（“分片鏈”），並根據其 *account_id* 的第一個字節在一個精確的分片中保持帳戶的狀態。

分片是對區塊鏈系統進行擴展的自然方法，因為如果它被正確實現，系統中的用戶和智能合約根本不需要知道分片的存在。實際上，當負載變得

太高時，人們通常希望在現有的單鏈項目（如以太坊）中添加分片。

另一種擴展的方法是使用如**2.8.10**所述的異質工作鏈的“聯盟”，讓每個用戶可以在她選擇的一個或多個工作鏈中保持她的帳戶，在需要時從她在一個工作鏈中的帳戶轉移資金到另一個工作鏈，實際上執行 $1:1$ 的替代幣兌換操作。這種方法的缺點已經在**2.8.10**中討論過了。

然而，分片在快速且可靠的實現上並不容易，因為它意味著在不同分片區塊之間需要大量的消息傳遞。例如，如果帳戶在 N 個分片區塊之間均勻分布，而且唯一的交易只是從一個帳戶轉移資金到另一個帳戶，那麼只有一小部分（$1/N$）的所有交易會在單一的區塊鏈上進行；幾乎所有的交易（$1-1/N$）都涉及兩個區塊鏈，需要進行區塊鏈間通信。如果我們希望這些交易快速完成，我們需要一個快速的系統來在分片區塊之間傳輸消息。換句話說，在 **2.8.14**中描述的意義上，區塊鏈項目需要是“緊密耦合”的。

2.8.13. Dynamic and static sharding. 分片可能是動態的（如果需要時自動創建額外的分片）或靜態的（當預定的分片數量是固定的，最多只能通過硬分叉進行更改）。大多數分片提案都是靜態的；TON 區塊鏈使用動態分片（參見**2.7**）。

2.8.14. Interaction between blockchains: loosely-coupled and tightly-coupled systems. 多區塊鏈項目可以根據其支持的組成區塊鏈之間互動的程度進行分類。

最低級的支持程度是不支持任何不同區塊鏈之間的互動。我們在這裡不考慮這種情況，因為我們更傾向於說這些區塊鏈不是同一區塊鏈系統的一部分，而只是同一區塊鏈協議的分離實例。

下一個支持程度是缺乏任何針對區塊鏈間消息傳遞的具體支持，這使得區塊鏈之間的互動原則上是可能的，但是相對困難。我們將這樣的系統稱為“鬆散耦合”；在這樣的系統中，必須像處理完全獨立的區塊鏈項目（例如比特幣和以太坊）一樣，在區塊鏈之間發送消息和轉移價值。換句話說，必須將出站消息（或其生成交易）包括在源區塊鏈的區塊中。然後，她（或其他人）必須等待足夠的確認（例如，一定數量的後續區塊）才能將起始交易視為“已提交”和“不可變”，以便能夠基於其存在執行外部操作。只有在這之後，才能提交將消息轉發到目標區塊鏈的交易（可能包括引用和原始交易存在的 Merkle 證明）。

如果在轉移消息之前等待的時間不夠長，或者由於其他原因發生了分叉，那麼兩個區塊鏈的聯合狀態將變得不一致：一個消息被傳送到第二個區塊鏈中，而該消息從未在（第一個區塊鏈的最終選擇分支中）生成。

有時，通過對所有工作鏈的區塊中的消息格式和輸入輸出消息隊列位置進行標準化，可以添加對消息傳遞的部分支持（這在異質系統中尤其有用）。

雖然這在一定程度上有助於消息傳遞，但在概念上與前一種情況沒有太大的不同，因此這樣的系統仍然是“鬆散耦合”的。

相比之下，“緊密耦合”的系統包括特殊機制，以在所有區塊鏈之間提供快速的消息傳遞。期望的行為是，能夠在消息在原始區塊鏈的區塊中生成後立即將其傳送到另一個工作鏈中。另一方面，“緊密耦合”的系統也被期望在出現分叉的情況下維持整體一致性。雖然這兩個要求乍一看似乎矛盾，但我們認為 TON 區塊鏈使用的機制（將分片區塊哈希包括在主鏈區塊中；使用“垂直”區塊鏈修復無效區塊，參見**2.1.17**；超立方體路由，參見**2.4.19**；Instant Hypercube Routing，參見**2.4.20**）使其成為一個“緊密耦合”的系統，或許是迄今為止唯一的一個。

當然，構建一個“鬆散耦合”的系統要簡單得多；然而，快速高效的分片（參見**2.8.12**）要求系統是“緊密耦合”的。

2.8.15. Simplified classification. Generations of blockchain projects. 到目前為止，我們提出的分類將所有區塊鏈項目分為許多類。然而，我們使用的分類標準在實踐中具有相當大的相關性。這使我們能夠提出一種簡化的“世代”方法來分類區塊鏈項目，作為現實的非常粗略的近似，並提供一些示例。尚未實施和部署的項目以斜體顯示；一個世代的最重要特徵以粗體顯示。

- 第一代：單鏈、**PoW**、不支援智能合約。例如：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、支援智能合約。例如：以太坊（2013 年；2015 年部署），至少在其原始形式中。
- 第三代：單鏈、**PoS**、支援智能合約。例如：未來的以太坊（2018 年或之後）。
- 另一種第三代（3′）：多鏈、PoS、不支援智能合約、鬆散耦合。例如：比特股（2013-2014 年；使用 DPOS）。
- 第四代：多鏈，**PoS**，支援智能合約，鬆散耦合。例如：*EOS*（2017 年；使用 DPOS）、*PolkaDot*（2016 年；使用 BFT）。
- 第五代：多鏈、帶 BFT 的 PoS、支援智能合約、緊密耦合，具有分片功能。例如：*TON*（2017 年）。

儘管不是所有區塊鏈項目都能準確地歸入這些類別中的一個，但其中大多數都能歸入其中之一。

2.8.16. Complications of changing the "genome" of a blockchain project. 上述分類定義了區塊鏈項目的"基因組"。這個基因組是相當"嚴格"的：一旦項目部署並被許多人使用，就幾乎不可能改變它。需要一系列的硬分叉（這需要社區大多數人的批准），即使這樣做，改變也需要非常保守，以保持向後兼容性（例如，改變虛擬機的語義可能會破壞現有的智能合約）。另一個選擇是創建新的"側鏈"，具有不同的規則，並以某種方式將它們綁定到原始項目的區塊鏈（或區塊鏈）上。可以將現有單一區塊鏈項目的區塊鏈用作基本上是新的獨立項目的外部主鏈。[24]

我們的結論是，一旦部署後，項目的基因組很難改變。即使從 PoW 開始，並計劃在將來將其替換為 PoS 也相當複雜。[25]在原始設計中沒有支持分片的項目中添加分片似乎幾乎不可能。[26]事實上，將對智能合約的支持添加到原始設計中沒有此類功能的項目（即比特幣）被認為是不可能的（或至少是大多數比特幣社區所不希望的），最終導致創建一個新的區塊鏈項目，即以太坊。

2.8.17. Genome of the TON Blockchain. 2 / 2

因此，如果想要建立一個可擴展的區塊鏈系統，就必須從一開始就仔細選擇其基因。如果該系統旨在在未來支持一些在部署時尚不知道的特定附加功能，則應從一開始就支持"異構"的工作鏈（具有潛在不同的規則）。為了使系統真正可擴展，它必須從一開始就支持分片；分片只有在系統是"緊密耦合"的情況下才有意義（參見**2.8.14**），因此這又意味着存在主鏈，快速的區塊鏈間通信系統，使用 BFT PoS 等。

考慮到所有這些影響因素，TON 區塊鏈項目所做的大多數設計選擇似乎都是自然的，幾乎是唯一可能的選擇。

2.9 Comparison to Other Blockchain Projects

我們簡要討論了 TON 區塊鏈及其最重要且獨特的功能，現在試圖將其與現有和擬議的區塊鏈項目的地圖相結合。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項目，並構建這樣一個"區塊鏈項目地圖"。我們將此地圖表示為表 1，然後分別簡要討論一些項目，以指出它們的特異性可能不符合一般方案。

[24]例如，Plasma 項目計劃使用以太坊區塊鏈作為其（外部）主鏈；否則它不與以太坊互動，它可以由與以太坊項目無關的團隊提出和實現。

[25]截至 2017 年，以太坊仍在努力從 PoW 過渡到結合 PoW 和 PoS 的系統；我們希望它有一天能成為真正的 PoS 系統。

[26]有關以太坊的分片提案可追溯至 2015 年；目前尚不清楚它們如何在不破壞以太坊或創建基本上獨立的平行項目的情況下實施和部署。

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: 一些值得注意的區塊鏈項目總結。列分別為：項目 - 項目名稱; 年份 - 宣布年份和部署年份; *G.* - 世代 (參見**2.8.15**)；*Cons.* - 共識算法 (參見**2.8.3** 和 **2.8.4**)；*Sm.* - 對任意代碼（智能合約；參見**2.8.6**）的支持；*Ch.* - 單一/多個區塊鏈系統（參見**2.8.2**）；*R.* - 異構/同質多鏈系統（參見**2.8.8**）；*Sh.* - 分片支持（參見**2.8.12**）；*Int.* - 區塊鏈之間的交互作用，(L) 鬆散或 (T) 緊密（參見**2.8.14**）。

2.9.1. Bitcoin [12]; `https://bitcoin.org/`. 比特幣（2009 年）是第一個也是最著名的區塊鏈項目。它是一個典型的第一代區塊鏈項目：它是單鏈式的，它使用工作量證明（Proof-of-Work）的「最長分支獲勝」分支選擇算法，並且它沒有一個圖靈完備的腳本語言（然而，它支持沒有循環的簡單腳本）。比特幣區塊鏈沒有帳戶的概念；它使用 UTXO（未花費的交易輸出）模型。

2.9.2. Ethereum [2]; `https://ethereum.org/`.*Ethereum/*（2015）是第一個支援圖靈完備智能合約的區塊鏈。因此，它是一個典型的第二代項目，並且是其中最受歡迎的項目。它在單個區塊鏈上使用工作量證明，但具有智能合約和帳戶。

2.9.3. NXT; `https://nxtplatform.org/`.*NXT/*（2014）是第一個基於 PoS 的區塊鏈和貨幣。它仍然是單一鏈，並且沒有智能合約支援。

2.9.4. Tezos; `https://www.tezos.com/`. *Tezos/*（2018 年或以後）是一個建議採用 PoS 的單一區塊鏈項目。我們在此提到它是因為它具有獨特的功能：它的區塊解釋函數 *ev_block*（參見**2.2.6**）不是固定的，而是由 OCaml 模塊決定，可以通過提交新版本到區塊鏈（並收集一些提議變更的投票）來升級。通過這種方式，人們將能夠通過首先部署“vanilla” Tezos 區塊鏈，然後逐漸改變區塊解釋函數以期朝著期望的方向創建定制的單一鏈項目，而不需要進行硬分叉。

這個想法雖然很有趣，但明顯的缺點是它禁止在其他語言（如 C++）中

進行優化實現，因此基於 Tezos 的區塊鏈注定會具有較低的性能。我們認為，通過發布所提出的區塊解釋函數 *ev_trans* 的正式規範，而不是固定特定的實現，可能會得到類似的結果。

2.9.5. Casper.[27] *Casper/* 是 Ethereum 的一種即將推出的 PoS 算法；如果其逐步在 2017 年（或 2018 年）成功部署，將把 Ethereum 變成一個支持智能合約的單鏈 PoS 或混合 PoW+PoS 系統，從而將 Ethereum 轉變為一個第三代項目。

2.9.6. BitShares [8]; `https://bitshares.org`. *BitShares/*（2014）是一個基於區塊鏈的分散式交易所平台。它是一個異質的多區塊鏈 DPoS 系統，沒有智能合約；通過只允許一小組預定義的專門事務類型，它實現了高性能，這些事務類型可以在 C++ 中高效實現，假設區塊鏈狀態符合內存大小。它也是第一個使用委託股權證明（DPoS）的區塊鏈項目，至少展示了其對於某些特定目的的可行性。

2.9.7. EOS [5]; `https://eos.io`. *EOS/*（2018 年或以後）是一個提出的異質多區塊鏈 DPoS 系統，支持智能合約和一些最小的消息支持（仍然是鬆散耦合的，如 **2.8.14**所述）。這是由之前成功創建了 BitShares 和 SteemIt 項目的同一團隊嘗試，展示了 DPoS 共識算法的優勢。通過為需要的項目創建專門的工作鏈（例如，分散式交換可能使用支持一組優化事務的特殊工作鏈，類似於 BitShares 所做的）以及創建具有相同規則的多個工作鏈（聯盟/，如 **2.8.10**所述），將實現可擴展性。關於這種可擴展性方法的缺點和限制已在 *loc. cit.* 中討論。參見**2.8.5**、**2.8.12**和**2.8.14**，以了解 DPoS、分片、工作鏈之間的交互作用以及其對區塊鏈系統可擴展性的影響的更詳細討論。

同時，即使無法「在區塊鏈內創建一個 Facebook」（參見**2.9.13**），我們認為 EOS 可能會成為某些高度專門化且弱交互的分佈式應用程序的便捷平台，類似於 BitShares（去中心化交換）和 SteemIt（去中心化博客平台）。

2.9.8. PolkaDot [17]; `https://polkadot.io/`. *PolkaDot/*（2019 年或以後）是最仔細的多鏈 PoS 項目之一，其開發由以太坊聯合創始人之一領導。這個項目是我們地圖上最接近 TON Blockchain 的項目之一。（事實上，我們感謝 PolkaDot 項目為「漁夫」和「提名人」等術語提供了我們的術語。）

PolkaDot 是一個異質鬆散耦合的多鏈 PoS 項目，採用拜占庭容錯（BFT）共識生成新區塊和主鏈（可以是外部的，例如以太坊區塊鏈）。它還使用超立方路由，有些像 TON 在 **2.4.19**中描述的（較慢的版本）。

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

它的獨特之處在於它不僅能夠創建公共區塊鏈，還能夠創建私有區塊鏈。這些私有區塊鏈還能夠與其他公共區塊鏈交互，無論是 PolkaDot 還是其他區塊鏈。

因此，PolkaDot 可能成為大規模私有區塊鏈的平台，例如，銀行聯盟可以使用它快速地相互轉移資金，或者任何大型企業對私有區塊鏈技術可能有的其他用途。

然而，PolkaDot 沒有分片支持，也沒有緊密耦合。這在某種程度上阻礙了它的可擴展性，這與 EOS 的可擴展性相似。（也許稍微好一些，因為 PolkaDot 使用 BFT PoS 而不是 DPoS。）

2.9.9. Universa; `https://universa.io`. 我們在這裡提及這個不尋常的區塊鏈項目的唯一原因是因為它是迄今為止唯一一個在路過中明確提到類似於我們的無限分片模式（參見**2.1.2**）的項目。它的另一個特點是通過承諾只有項目的可信和授權合作夥伴才能作為驗證者被允許加入，因此他們永遠不會提交無效的區塊，從而繞過了所有與拜占庭容錯相關的復雜問題。這是一個有趣的決定；然而，它本質上使一個區塊鏈項目有意成為集中化的，而區塊鏈項目通常希望避免這種情況（為什麼在一個受信任的集中化環境中工作還需要區塊鏈呢?）。

2.9.10. Plasma; `https://plasma.io`). *Plasma/*（2019 年?）是來自以太坊的另一位聯合創始人的非傳統區塊鏈項目。它旨在減輕以太坊的一些限制，而不引入分片。本質上，它是一個獨立於以太坊的項目，引入了一個（異質）工作鏈的層次結構，綁定在以太坊區塊鏈上（作為外部主鏈）的頂層。從層次結構中的任何區塊鏈（以以太坊區塊鏈為根）轉移資金以及所需的工作描述，然後在子工作鏈中進行必要的計算（可能需要將原始工作的某些部分向下傳遞），將其結果向上傳遞，並收集獎勵。通過一個（支付通道啟發的）機制，允許用戶將其資金從行為不當的工作鏈單方面提取到其父工作鏈（雖然速度較慢），並重新分配其資金和工作到另一個工作鏈，以達到一致性和驗證這些工作鏈的問題被迴避。

通過這種方式，Plasma 可能成為綁定到以太坊區塊鏈上的分布式計算平台，類似於「數學協處理器」。然而，這似乎不是實現真正通用的可擴展性的方法。

2.9.11. Specialized blockchain projects. 還有一些特殊的區塊鏈項目，例如 FileCoin（一個系統，激勵用戶提供他們的磁盤空間來存儲其他願意付費的用戶的文件）、Golem（基於區塊鏈的平台，用於租賃和借貸特定應用程序（如 3D 渲染）的計算能力）或 SONM（另一個類似的計算能力租借項目）。這樣的項目在區塊鏈組織的層面上並沒有引入任何概念上的創新；相反，它們是特定的區塊鏈應用程序，可以由運行在通用區塊鏈中的智

能合約實現，前提是它能夠提供所需的性能。因此，這類項目很可能會使用現有或計劃中的區塊鏈項目作為其基礎，例如 EOS、PolkaDot 或 TON。如果一個項目需要「真正的」可擴展性（基於分片），最好使用 TON；如果它滿足於在「聯邦」的上下文中工作，通過定義一系列自己的工作鏈明確地為其目的進行優化，它可能會選擇 EOS 或 PolkaDot。

2.9.12. The TON Blockchain. TON（Telegram Open Network）區塊鏈（計劃於 2018 年推出）是我們在本文中描述的項目。它旨在成為第一個第五代區塊鏈項目，即一個 BFT PoS-多鏈項目，混合同質/異質，支持（可分片的）自定義工作鏈，具有本地分片支持，並緊密耦合（特別是能夠在保持所有分片鏈的一致狀態的同時幾乎瞬間轉發分片之間的消息）。因此，它將成為一個真正可擴展的通用區塊鏈項目，能夠容納基本上可以在區塊鏈中實現的任何應用程序。當與 TON 項目的其他組件（參見 **1**）一起增強時，它的可能性甚至更加廣泛。

2.9.13. Is it possible to "upload Facebook into a blockchain"? 有時人們聲稱，將來可以通過區塊鏈上的分佈式應用程序實現一個與 Facebook 相當規模的社交網絡。通常會引用一個喜愛的區塊鏈項目作為這樣一個應用程序的可能「主機」。

我們不能說這是技術上的不可能。當然，需要一個緊密耦合的區塊鏈項目，並且要有真正的分片支持（即 TON），以便這樣一個大型應用程序不會工作得太慢（例如，將來自一個分片鏈中的用戶的消息和更新傳遞到其他分片鏈中的朋友，需要合理的延遲）。然而，我們認為這是不必要的，而且永遠不會實現，因為代價是禁止性的。

讓我們將「將 Facebook 上傳到區塊鏈中」視為一個思想實驗；任何類似規模的其他項目也可以作為例子。一旦將 Facebook 上傳到區塊鏈中，當前由 Facebook 服務器執行的所有操作都將作為交易序列化到某些區塊鏈中（例如，TON 的分片鏈），並由這些區塊鏈的所有驗證者執行。每個操作將需要執行至少 20 次，如果我們希望每個區塊至少收集到 20 個驗證者的簽名（立即或在 DPOS 系統中的預期時間內）。同樣，Facebook 服務器上保存的所有數據都將保存在相應分片鏈的所有驗證者的磁盤上（即至少 20 份拷貝）。

由於驗證者本質上是與當前由 Facebook 使用的服務器相同的服務器（或者可能是服務器集群，但這不會影響這個論點的有效性），因此我們可以看出，將 Facebook 運行在區塊鏈上的總硬件成本至少比以常規方式實現高出 20 倍。

事實上，成本還會更高，因為區塊鏈的虛擬機比運行編譯優化代碼的「裸 CPU」要慢，而且其存儲並未針對 Facebook 特定問題進行優化。通過製作一個具有一些適用於 Facebook 的特殊交易的特定工作鏈，可以部分緩

解這個問題；這是 BitShares 和 EOS 實現高性能的方法，TON 區塊鏈中也有。然而，一般的區塊鏈設計本身仍然會施加一些額外的限制，例如需要在區塊中註冊所有操作作為交易、將這些交易組織成默克爾樹、計算和檢查它們的默克爾哈希、進一步傳播該區塊等。

因此，保守估計，為了驗證托管規模如此之大的社交網絡的區塊鏈項目，需要使用與 Facebook 現在使用的性能相同的服務器 100 倍。有人必須為這些服務器付費，無論是擁有分佈式應用程序的公司（想象一下在每個 Facebook 頁面上看到 700 個廣告而不是 7 個），還是它的用戶。無論哪種方式，這都似乎在經濟上不可行。

我們相信「不是所有東西都應該上傳到區塊鏈中」。例如，沒有必要在區塊鏈中保存用戶的照片；在區塊鏈中註冊這些照片的哈希值並將照片保存在分佈式的離線存儲（例如 FileCoin 或 TON Storage）中會是一個更好的想法。這就是為什麼 TON 不僅僅是一個區塊鏈項目，而是一個圍繞 TON 區塊鏈的幾個組件（TON P2P 網絡、TON 存儲、TON 服務）的集合，如**1**和**4**章所述。

3 TON Networking

任何區塊鏈項目都需要不僅規定區塊格式和區塊鏈驗證規則，還需要一個用於傳播新區塊、發送和收集交易候選的網絡協議。換句話說，每個區塊鏈項目都必須建立一個專門的點對點網絡。這個網絡必須是點對點的，因為區塊鏈項目通常被期望是去中心化的，因此不能依賴於一個集中式的服務器群和使用傳統的客戶端-服務器架構，例如，像傳統的在線銀行應用程序一樣。即使輕客戶端（例如，輕量級加密貨幣錢包智能手機應用程序）必須以客戶端-服務器的方式連接到全節點，如果用於連接全節點的協議標準化程度足夠高，它們實際上是可以自由地連接到另一個全節點的，如果它們之前的對等節點失效的話。

雖然單一區塊鏈項目（例如比特幣或以太坊）的網絡需求可以相當容易地滿足（基本上需要構建一個“隨機”的點對點覆蓋網絡，並通過一個流言協議傳播所有新的區塊和交易候選），但多區塊鏈項目（例如 TON 區塊鏈）要求更高（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 項目的網絡部分至少值得簡要討論。

另一方面，一旦為支持 TON 區塊鏈所需的更複雜的網絡協議建立起來，就會發現它們可以很容易地用於與 TON 區塊鏈的即時需求不一定相關的目的，從而為在 TON 生態系統中創建新服務提供更多的可能性和靈活性。

3.1 Abstract Datagram Network Layer

建立 TON 網絡協議的基石是（*TON*）抽象（數據報）網絡層。它使所有節點能夠假定某些「網絡身份」，這些身份由 256 位的「抽象網絡地址」表示，並使用這些 256 位網絡地址來識別發送方和接收方進行通信（首先發送數據報）。特別是，您不需要擔心 IPv4 或 IPv6 地址、UDP 端口號等；它們被抽象網絡層隱藏。

3.1.1. Abstract network addresses. 一個抽象網絡地址或抽象地址或簡稱地址，是一個 256 位整數，基本上等於 256 位 ECC 公鑰。這個公鑰可以隨意生成，因此可以創建出節點所需的各種不同的網絡身份。但是，為了接收（和解密）針對這樣的地址發送的消息，必須知道相應的私有密鑰。

實際上，地址不是公鑰本身；相反，它是一個序列化的 TL 對象（參見**2.2.5**），其 256 位哈希（Hash = sha256）可以描述幾種不同類型的公鑰和地址，具體取決於它的構造函數（前四個字節）。在最簡單的情況下，這個序列化的 TL 對象僅由一個 4 字節的魔數和一個 256 位的橢圓曲線加密（ECC）公鑰組成；在這種情況下，地址將等於此 36 字節結構的哈希值。但是，可以使用 2048 位 RSA 密鑰，或者任何其他公鑰加密方案。

當一個節點獲取了另一個節點的抽象地址時，它還必須接收它的「前像」（即，序列化的 TL 對象，其哈希等於該抽象地址），否則它將無法對該地址加密和發送數據報。

3.1.2. Lower-level networks. UDP implementation. 從幾乎所有 TON 網絡組件的角度來看，唯一存在的是一個網絡（抽象數據報網絡層），能夠（不可靠地）從一個抽象地址發送數據報到另一個地址。原則上，抽象數據報網絡層（ADNL）可以在不同的現有網絡技術上實現。但是，我們將在 IPv4 / IPv6 網絡（例如互聯網或企業內部網絡）上使用 UDP 來實現它，如果 UDP 不可用，則可以選擇使用 TCP 回退。

3.1.3. Simplest case of ADNL over UDP. 將從發送方的抽象地址發送數據報文到任何其他已知前像的抽象地址的最簡單情況可以實現如下。

假設發送方以某種方式知道擁有目標抽象地址的接收方的 IP 地址和 UDP 端口，並且接收方和發送方都使用從 256 位 ECC 公鑰派生的抽象地址。

在這種情況下，發送方只需使用其私鑰對其要發送的數據報文進行 ECC 簽名，然後附加其源地址（如果接收方不知道該前像，則附加源地址的前像）。然後使用接收方的公鑰加密結果，將其嵌入到 UDP 數據報文中，並發送到接收方已知的 IP 和端口。由於 UDP 數據報文的前 256 位包含接收方的抽象地址，接收方可以識別應使用哪個私鑰來解密數據報文的其餘部分。僅在此之後，發送方的身份才會被揭示。

3.1.4. Less secure way, with the sender's address in plaintext. 有時，當接收方和發送方的地址在 UDP 數據報文中以明文形式保存時，一個不太安全的方案就足夠了。使用 ECDH（橢圓曲線迪菲－赫爾曼）將發送方的私鑰和接收方的公鑰結合起來生成一個 256 位的共享密鑰。接下來，與隨機的 256 位 nonce 一起，使用該共享密鑰來推導用於加密的 AES 密鑰。例如，在加密之前將原始明文數據的哈希串連接到明文中，可以提供完整性保護。

這種方法的優點是，如果預計在兩個地址之間交換多個數據報文，則可以僅計算一次共享密鑰，然後將其緩存。因此，在加密或解密下一個數據報文時，不再需要較慢的橢圓曲線運算。

3.1.5. Channels and channel identifiers. 在最簡單的情況下，攜帶嵌入式 TON ADNL 數據報的 UDP 數據報的前 256 位將等於接收方的地址。但是，通常它們構成一個通道標識符。有不同類型的通道。其中一些是點對點的；它們由希望在未來交換大量數據的兩個方創建，通過交換幾個按照**3.1.3**或**3.1.4**所述加密的數據包，通過運行經典或橢圓曲線 Diffie-

Hellman（如果需要額外安全性），或者僅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，通道標識符是從共享秘密結合一些附加數據（例如發件人和收件人的地址）中派生出來的，例如通過哈希，並且該標識符用作攜帶使用該共享秘密加密的數據的 UDP 數據報的前 256 位。

3.1.6. Channel as a tunnel identifier. 一般來說，“通道”或“通道標識符”僅選擇已知接收方的入站 UDP 數據報處理方式。如果通道是接收方的抽象地址，則按照**3.1.3**或**3.1.4**中所述的方式進行處理；如果通道是在**3.1.5**中討論的已建立點對點通道，則處理過程包括利用共享密鑰解密數據報，如 *loc. cit.* 中所述，以此類推。

特別是，通道標識符實際上可以選擇一個“通道”，當直接收件人僅將接收到的消息轉發給其他人 - 實際收件人或另一個代理。可能會進行一些加密或解密步驟（類似於“洋蔥路由”[6] 或甚至“大蒜路由”[28]），並且另一個通道標識符可能會用於重新加密轉發的封包（例如，對等通道可以用於將封包轉發到路徑上的下一個接收者）。

這樣，可以在 TON 抽象數據報網絡層級別上添加對“隧道”和“代理”的一些支持，這與 TOR 或 I^2P 項目提供的支持有些類似，而不會影響所有更高級的 TON 網絡協議的功能，這些協議對於這樣的添加是不知情的。這個機會被 *TON* 代理服務所利用（參見**4.1.11**）。

3.1.7. Zero channel and the bootstrap problem. 一個 TON ADNL 節點通常會有一個“鄰居表”，其中包含其他已知節點的信息，例如它們的抽象地址、它們的前像（即公鑰）和它們的 IP 地址和 UDP 端口。然後，它將通過使用從這些已知節點獲得的信息作為對特殊查詢的答案來逐漸擴展此表，有時會刪除過時的記錄。

然而，當 TON ADNL 節點剛啟動時，可能會出現它不知道任何其他節點的情況，並且只能學習到一個節點的 IP 地址和 UDP 端口，但不知道它的抽象地址。例如，如果輕客戶端無法訪問任何先前緩存的節點和任何硬編碼到軟件中的節點，並且必須要求用戶輸入節點的 IP 地址或 DNS 域名，通過 DNS 解析。

I 在這種情況下，節點將向問題節點的特殊“零通道”發送封包。這不需要知道接收者的公鑰（但消息仍應包含發件人的身份和簽名），因此消息在未加密的情況下傳輸。它通常僅用於獲取接收者的身份（可能是專門為此目的創建的一次性身份），然後以更安全的方式開始通信。

一旦至少知道一個節點，就可以通過向已知節點發送特殊查詢的答案來填充“鄰居表”和“路由表”中的更多項目。

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

並非所有節點都需要處理發送到零通道的數據報，但用於啟動輕客戶端的節點應支持此功能。

3.1.8. TCP-like stream protocol over ADNL. ADNL 是一種不可靠（小型）數據報協議，基於 256 位抽象地址，可用作更複雜網絡協議的基礎。例如，可以構建一個類似 TCP 的流協議，使用 ADNL 作為 IP 的抽象替代品。但是，TON 項目的大多數組件不需要這樣的流協議。

3.1.9. RLDP, or Reliable Large Datagram Protocol over ADNL. 使用建立在 ADNL 之上的可靠任意大小數據報協議（RLDP）來代替類似 TCP 的協議。例如，可以使用此可靠數據報協議向遠程主機發送 RPC 查詢並從其接收答案（參見**4.1.5**）。

3.2 TON DHT: Kademlia-like Distributed Hash Table

TON 分散式哈希表（*DHT*）/在 TON 項目的網絡部分中發揮著關鍵作用，用於定位網絡中的其他節點。例如，想要將交易提交到分片鏈中的客戶端可能希望找到該分片鏈的驗證者或收集者，或者至少找到一個可以轉發客戶端交易到收集者的節點。這可以通過在 TON DHT 中查找特殊的鍵來實現。TON DHT 的另一個重要應用是可以通過查找隨機鍵或新節點的地址，快速填充新節點的鄰居表（參見**3.1.7**）。如果一個節點對其入站數據報使用代理和隧道，它將在 TON DHT 中發布隧道標識符和其入口點（例如，IP 地址和 UDP 端口）；然後所有希望向該節點發送數據報的節點都將首先從 DHT 中獲取此聯繫信息。

TON DHT 是 *Kademlia-like* 分散式雜湊表（*Kademlia-like distributed hash tables*）家族的成員之一 [10]。

3.2.1. Keys of the TON DHT. TON DHT 的金鑰/（*keys*）為 256 位元整數。在大多數情況下，它們是以 TL 序列化物件（參見 **2.2.5**）的 SHA256 值計算而得，稱為該金鑰的前像/（*preimage*）或金鑰說明（*key description*）。在某些情況下，TON 網路節點（參見 **3.1.1**）的抽象地址也可以用作 TON DHT 的金鑰，因為它們也是 256 位元的 TL 序列化物件的雜湊值。例如，如果一個節點不怕公開其 IP 地址，任何知道其抽象地址的人都可以透過在 DHT 中尋找該地址作為金鑰來找到它。

3.2.2. Values of the DHT. 這些 256 位元金鑰分配的值/（*values*）基本上是有限長度的任意位元組字串。這些位元組字串的解釋取決於相應金鑰的前像，通常由查找該金鑰的節點和儲存該金鑰的節點都知道。

3.2.3. Nodes of the DHT. Semi-permanent network identities. TON DHT 的金鑰-值映射儲存在 DHT 的節點/（*nodes*）上——實質上是 TON 網路的所有成員。為此，除了在 **3.1.1** 中描述的任意數量的短暫和永久抽象地址之外，TON 網路的任何節點（也許除了某些非常輕量的節點）都至少有一個「半永久地址」，用於識別其為 TON DHT 的成員。這個半永久/或 *DHT* 地址/不應該太經常更改，否則其他節點將無法找到它們正在查找的金鑰。如果一個節點不想透露其「真實」身份，它會生成一個單獨的抽象地址，僅用於參與 DHT。但是，這個抽象地址必須是公開的，因為它將與節點的 IP 地址和端口關聯。

3.2.4. Kademlia distance. 現在我們既有 256 位元金鑰，也有 256 位元（半永久）節點地址。我們引入所謂的 *XOR* 距離/或 *Kademlia* 距離 d_K 在 256 位元序列集合上，給定為

$$d_K(x,y) := (x \oplus y) \quad \text{解釋為一個 256 位元的無符號整數} \tag{27}$$

這裡 $x \oplus y$ 表示兩個相同長度的位元序列進行按位 eXclusive OR（XOR）運算。

Kademlia 距離在所有 256 位元序列組成的集合 $\mathbf{2}^{256}$ 上引入了一個度量。特別地，當且僅當 $x = y$ 時 $d_K(x,y) = 0$，$d_K(x,y) = d_K(y,x)$，並且 $d_K(x,z) \le d_K(x,y) + d_K(y,z)$。另一個重要的性質是，從 x 到任何給定距離的點只有一個：$d_K(x,y) = d_K(x,y')$ 意味著 $y = y'$。

3.2.5. Kademlia-like DHTs and the TON DHT. 如果一個分散式雜湊表（DHT）具有 256 位元金鑰和 256 位元節點地址，並且預期將金鑰 K 的值儲存在到與 K 的 Kademlia 距離最小的 s 個節點上（即，從它們的地址到 K 的 Kademlia 距離最小的 s 個節點），我們稱其為 *Kademlia-like DHT*/。

這裡 s 是一個小的參數，例如 $s = 7$，用於提高 DHT 的可靠性（如果我們只在一個節點上儲存金鑰，即距離 K 最近的節點，則如果該唯一節點離線，該金鑰的值將丟失）。

根據此定義，TON DHT 是一個 Kademlia-like DHT。它是在 **3.1** 中描述的 ADNL 協議上實現的。

3.2.6. Kademlia routing table. 參與 Kademlia-like DHT 的任何節點通常都維護一個 *Kademlia* 路由表（*Kademlia routing table*）。在 TON DHT 的情況下，它由 $n = 256$ 個桶組成，從 0 到 $n - 1$ 編號。第 i 個桶將包含有關某些已知節點（一個固定數量 t 的「最佳」節點，以及可能的一些額外候選節點）的資訊，這些節點與節點的地址 a 的 Kademlia 距離為 2^i 到

$2^{i+1} - 1$ 不等。[29] 這些資訊包括它們（半永久性的）地址、IP 地址和 UDP 埠，以及一些可用性資訊，例如上次 ping 的時間和延遲時間。

當一個 Kademlia 節點通過某些查詢學習到任何其他 Kademlia 節點時，它將其作為候選節點加入到其路由表的適當桶中。然後，如果該桶中的一些「最佳」節點失敗（例如，長時間不回應 ping 查詢），則它們可以被某些候選節點替換。通過這種方式，Kademlia 路由表保持著節點的數據。

來自 Kademlia 路由表的新節點也會被包含在 **3.1.7** 中描述的 ADNL 鄰居表中。如果 Kademlia 路由表的一個桶中的「最佳」節點經常使用，則可以建立一個通道（如 **3.1.5** 中所述）來促進數據報文的加密。

TON DHT 的一個特別功能是，它試圖選擇往返延遲最小的節點作為 Kademlia 路由表中“最佳”的節點，放入相應的桶中。

3.2.7. (Kademlia 網絡查詢) 一個 Kademlia 節點通常支持以下網絡查詢：

- Ping – 檢查節點是否可用。
- Store($key, value$) – 要求節點將 $value$ 作為 key 的值保存。對於 TON DHT，Store 查詢稍微複雜一些（參見 **3.2.9**）。
- Find_Node(key, l) – 要求節點返回到 key 的 l 個 Kademlia 最近已知節點（從其 Kademlia 路由表中）。
- Find_Value(key, l) – 與上面相同，但如果節點知道與 key 相對應的值，它只會返回該值。

當任何節點想要查找鍵 K 的值時，它首先創建一個包含 s' 個節點的集合 S（對於某個小的值 s'，例如 $s' = 5$），這些節點相對於所有已知節點之間的 Kademlia 距離最接近 K（即，它們來自 Kademlia 路由表）。然後向它們每個人發送一個 Find_Value 查詢，並將其答案中提到的節點包含在 S 中。然後對於 S 中最接近 K 的 s' 個節點，如果之前沒有進行過 Find_Value 查詢，則也會發送一個 Find_Value 查詢，這個過程會繼續進行，直到找到值或 S 停止增長。這是一種基於 Kademlia 距離的最接近 K 節點的“波束搜索”方法。

如果要設置某個鍵 K 的值，則對於 $s' \geq s$，使用 Find_Node 查詢運行相同的過程，以查找到 K 的 s 個最近節點。然後向它們所有發送 Store 查詢。

[29] 如果一個桶中有足夠多的節點，它可以進一步細分為，例如，八個子桶，取決於 Kademlia 距離的前四位。這將加速 DHT 查找。

在實現類似 Kademlia 的 DHT 中還有一些不太重要的細節（例如，任何節點應該每小時查找一次其自身的 s 個最近節點，並通過 Store 查詢將所有存儲的鍵重新發布給它們）。我們暫時忽略它們。

3.2.8. Booting a Kademlia node. 當 Kademlia 節點上線時，它首先通過查找自己的地址來填充其 Kademlia 路由表。在這個過程中，它識別出距離自己最近的 s 個節點。它可以從它們那裡下載它們所知道的所有 $(key, value)$ 對，以填充其 DHT 的部分。

3.2.9. Storing values in TON DHT. 在 TON DHT 中存儲值與一般的 Kademlia-like DHT 稍有不同。當有人希望存儲一個值時，她必須向 Store 查詢提供不僅是鍵 K 本身，還要提供它的逆像---即，一個 TL 序列化字符串（開頭有幾個預定義的 TL 構造器），其中包含鍵的“描述”。稍後，節點將保留此鍵描述，以及鍵和值。

鍵描述描述了正在存儲的對象的“類型”，其“所有者”以及未來更新的“更新規則”。所有者通常由鍵描述中包含的公鑰識別。如果它被包含，通常只接受由相應的私鑰簽名的更新。存儲對象的“類型”通常只是一個字節串。然而，在某些情況下，它可能更複雜---例如，一個輸入隧道描述（參見 **3.1.6**），或者一個節點地址的集合。

「更新規則」也可能不同。在某些情況下，它們只是允許使用者以簽名的方式替換舊值為新值（簽名必須與值一起儲存，以便其他節點在獲取此密鑰的值後進行後續檢查）。在其他情況下，舊值會以某種方式影響新值。例如，它可以包含序號，只有在新序號較大時才會覆蓋舊值（以防重放攻擊）。

3.2.10. Distributed “torrent trackers” and “network interest groups” in TON DHT. 另一個有趣的情況是，當值包含一個節點列表時，也許包含它們的 IP 地址和端口，或僅包含它們的抽象地址，而「更新規則」則是在確認請求者的身份後將其包含在此列表中。

這種機制可以用來創建一個分佈式的「種子追蹤器」，所有對某個「種子」（即某個文件）感興趣的節點可以找到其他對同一個種子感興趣或已經擁有一個副本的節點。

TON Storage/（參見 **4.1.8**）使用此技術來查找擁有所需文件副本（例如分片鏈狀態快照或舊區塊）的節點。然而，它更重要的用途是創建「覆蓋網絡組播子網絡」和「網絡感興趣組」（參見 **3.3**）。其想法是，只有一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，即使找到一個對同一分片鏈感興趣的節點也可能變得復雜。這種「分佈式種子追蹤器」提供了一種方便的方法來查找這些節點中的一部分。另一種選擇是向驗證者請求，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. Fall-back keys. 迄今為止，大部分已描述的「鍵類型」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，如果從該描述進行雜湊得到的鍵無法從 TON DHT 中檢索或更新，則會增加此欄位中的值，並嘗試進行新的嘗試。這樣，就無法通過創建許多位於攻擊鍵附近並控制相應的 DHT 節點的抽象地址來「捕獲」和「審查」鍵（即進行鍵保留攻擊）。

3.2.12. Locating services. 位於 TON 網絡並可通過建立在 TON ADNL 之上的（更高層次協議）訪問的某些服務，可能希望在某個地方發布其抽象地址，以便其客戶知道在哪裡找到它們。

然而，將服務的抽象地址發布在 TON 區塊鏈上可能不是最好的方法，因為抽象地址可能需要經常更改，而且提供多個地址以實現可靠性或負載平衡也是有道理的。

另一種方法是將公鑰發布到 TON 區塊鏈中，並在 TL 描述字符串（參見 **2.2.5**）中使用一個特殊的 DHT 鍵，指示該公鑰為其“所有者”，以發布服務抽象地址的最新列表。這是 TON 服務所採用的方法之一。

3.2.13. Locating owners of TON blockchain accounts. 在大多數情況下，TON 區塊鏈帳戶的所有者不希望與抽象的網絡地址，尤其是 IP 地址相關聯，因為這可能會侵犯他們的隱私。然而，在某些情況下，TON 區塊鏈帳戶的所有者可能希望發布一個或多個抽象地址，以便可以通過它們與他人聯繫。

一個典型的例子是 TON Payments「閃電網絡」（參見 **5.2**）中的節點，這是一個即時加密貨幣轉帳平台。一個公共的 TON Payments 節點不僅希望與其他同行建立支付通道，還希望發布一個抽象網絡地址，以便在以後的時間通過這些已建立的通道進行支付轉移時可以與它聯繫。

其中一種方法是在創建支付通道的智能合約中包含抽象網絡地址。更靈活的方法是在智能合約中包含公鑰，然後如 **3.2.12**所述使用 DHT。

最自然的方法是使用控制 TON 區塊鏈帳戶的私鑰來簽署並發布關於該帳戶所關聯的抽象地址的 TON DHT 更新。這個過程幾乎與 **3.2.12**中描述的方式相同。但是，使用的 DHT 鍵需要一個特殊的鍵描述，只包含 *account_id* 本身，等於“帳戶描述”的 SHA256，其中包含帳戶的公鑰。簽名也包含在此 DHT 鍵的值中，包含帳戶描述。

這樣，就提供了一個查找某些 TON 區塊鏈帳戶所有者的抽象網絡地址的機制。

3.2.14. Locating abstract addresses. 需要注意的是，TON DHT 雖然是建立在 TON ADNL 之上，但 TON ADNL 本身也使用 TON DHT 來進行幾個目的。

其中最重要的目的是，從 256 位元抽象地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 位元抽象地址發送數據包，即使沒有提供額外的資訊。

為此，只需在 DHT 中查找 256 位元抽象地址作為索引鍵。如果找到具有此地址（即使用此地址作為公開半持久性 DHT 地址）的節點，則可以了解其 IP 地址和端口；或者，可能會檢索到鍵值的輸入隧道描述，該描述是由正確的私密金鑰簽署的，這樣該隧道描述就會用於向目標收件人發送 ADNL 數據包。

需要注意的是，為了使一個抽象地址「公開」（可從網路中的任何節點訪問），其擁有者必須使用它作為半持久 DHT 地址，或在考慮該抽象地址的 DHT 索引鍵中發佈一個輸入隧道描述，該描述具有其另一個公開抽象地址（例如半持久地址）作為隧道的入口點。另一個選擇是簡單地公佈其 IP 地址和 UDP 端口。

3.3 覆蓋網路和多點傳送訊息

在像 TON 區塊鏈這樣的多區塊鏈系統中，即使是全節點通常也只對一些分片區塊鏈的更新（即新區塊）感興趣。為此，在 TON 網路中，需要在 ADNL 協議（參見 **3.1**）之上建立一個特殊的覆蓋（子）網路，每個分片區塊鏈都需要建立一個。

因此，需要建立任意的覆蓋子網路，開放給任何願意參與的節點。在這些覆蓋網路中運行建立在 ADNL 之上的特殊流言協議。特別是，可以使用這些流言協議在此類子網路中傳播（廣播）任意數據。

3.3.1. Overlay networks. 一個覆蓋（子）網路（*overlay (sub)network*）是一個簡單的（虛擬）網路，實現在某個更大的網路內。通常，只有一些更大網路的節點參與覆蓋子網路，而這些節點之間的一些「連結」，無論是實體的或虛擬的，才是覆蓋子網路的一部分。

這樣，如果將包含的網路表示為一個圖形（例如，在數據包網路（如 ADNL）的情況下，可能是一個完整的圖形，其中任何節點都可以輕易地與其他節點通訊），則覆蓋子網路是該圖形的一個子圖（*subgraph*）。

在大多數情況下，覆蓋網路是使用一些建立在更大網路的網路協議之上的協議來實現的。它可以使用與更大的網路相同的地址，或使用自定義的地址。

3.3.2. Overlay networks in TON.TON 中的覆疊網路是建立在在 **3.1**中討論的 ADNL 協議之上，同時在覆疊網路中使用 256 位元的 ADNL 抽象位址作為地址。通常每個節點會選擇一個抽象位址作為其在覆疊網路中的位址。

與 ADNL 不同，TON 覆疊網路通常不支援傳送資料包到任意的其他節點。相反地，一些「半永久鏈結」在一些節點之間建立（對於所考慮的覆疊網路而言，這些節點被稱為「鄰居」），並且訊息通常會沿著這些鏈結傳送（即從一個節點傳送到其鄰居之一）。這樣，TON 覆疊網路是 ADNL 網路中的一個（通常不完整的）子圖。

TON 覆疊網路中的鄰居連結可以使用專用的點對點 ADNL 通道來實現（參見 **3.1.5**）。

覆疊網路的每個節點都維護一個鄰居清單（相對於覆疊網路），其中包含其抽象位址（用於在覆疊網路中識別它們）和一些連結資料（例如，用於與它們通信的 ADNL 通道）。

3.3.3. Private and public overlay networks. 有些覆疊網路是公開的，這意味著任何節點都可以自由地加入它們。其他的則是私有的，這意味著只有特定的節點可以加入（例如，那些可以證明自己是驗證者的節點）。有些私有覆疊網路甚至對「一般公眾」都是未知的。有關這種覆疊網路的資訊僅提供給某些受信任的節點；例如，可以使用公鑰對其進行加密，只有具有相應私鑰副本的節點才能解密此資訊。

3.3.4. Centrally controlled overlay networks. 有些覆疊網路是由一個或多個節點或某個廣為人知的公鑰擁有者中央控制 的。其他的則是去中心化 的，這意味著沒有特定的節點負責它們。

3.3.5. Joining an overlay network. 當一個節點想要加入一個覆疊網路時，它首先必須學習它的 256 位元網路識別碼，通常等於覆疊網路描述的 SHA256 — 一個 TL-序列化對象（參見 **2.2.5**），其中可能包含覆疊網路的中央控制機構（即其公鑰和可能的抽象位址[30]），一個包含覆疊網路名稱的字串，如果這是與該分片相關的覆疊網路，還有一個 TON 區塊鏈分片識別碼等等。

有時，可以通過在 TON DHT 中查找獲取覆疊網路描述以恢復覆疊網路描述，從而得到網路識別碼。在其他情況下（例如，對於私有覆疊網路），必須在網路識別碼的同時獲取網路描述。

3.3.6. Locating one member of the overlay network. 當一個節點學習了想要加入的覆蓋網路的網路識別符和網路描述後，必須找到至少一個屬於該網路的節點。

對於不想加入覆蓋網路，但僅想與之通信的節點，也需要這樣做。例如，可能有一個專門用於收集和傳播特定分片鏈交易候選的覆蓋網路，客戶端可能希望連接到該網路的任何節點以建議交易。

[30]或者，抽象位址可能會像在 **3.2.12**中解釋的那樣存儲在 DHT 中。

定位覆蓋網路成員的方法在該網路的描述中定義。有時（特別是對於私有網路），必須已經知道成員節點才能加入。在其他情況下，某些節點的抽象地址包含在網路描述中。更靈活的方法是僅在網路描述中指示負責該網路的中央權威，然後通過由該中央權威簽署的某些 DHT 鍵的值來獲取抽象地址。

最後，真正去中心化的公共覆蓋網路可以使用在 **3.2.10** 中描述的「分佈式 Torrent 跟踪器」機制，也可以在 TON DHT 的幫助下實現。

3.3.7. Locating more members of the overlay network. Creating links. 一旦找到覆蓋網路的一個節點，可以向該節點發送一個特殊的查詢，請求其他成員的列表，例如，被查詢節點的鄰居或其中隨機選擇的節點列表。

這使加入的成員能夠在覆蓋網路中建立「鄰接」或「鄰居列表」，選擇一些新學習的網路節點並與它們建立連接（即，專用的 ADNL 點對點通道，如 **3.3.2** 中所述）。之後，向所有鄰居發送特殊消息，指示新成員已準備好在覆蓋網路中工作。鄰居將其與新成員的連結包括在其鄰居列表中。

3.3.8. Maintaining the neighbor list. 覆蓋網路節點必須不時更新其鄰居列表。有些鄰居，或至少是與其相關的連接（通道）可能停止響應；在這種情況下，這些連接必須被標記為「暫停」，並且必須嘗試重新連接到這些鄰居，如果這些嘗試失敗，則必須銷毀這些連接。

另一方面，每個節點有時會從隨機選擇的鄰居那裡請求其鄰居列表（或其中隨機選擇的節點列表），並使用它部分更新自己的鄰居列表，通過添加一些新發現的節點並刪除一些舊的節點，無論是隨機刪除還是取決於它們的響應時間和數據包丟失統計數據。

3.3.9. The overlay network is a random subgraph. 這樣，覆蓋網路就成為 ADNL 網路內的一個隨機子圖。如果每個頂點的度數至少為三（即每個節點至少連接三個鄰居），則已知此隨機圖的聯通性概率幾乎為一。更精確地說，具有 n 個頂點的隨機圖被斷開的概率是指數級小的，如果 $n \geq 20$，可以完全忽略此概率。（當然，在全局網路分割的情況下，位於分割不同側的節點沒有機會相互了解，這種情況除外。）另一方面，如果 n 小於 20，只需要要求每個頂點至少有十個鄰居即可。

3.3.10. TON overlay networks are optimized for lower latency. TON 覆蓋網路將由之前的方法所生成的隨機'' 網路圖進行優化，其方法如下：每個節點試圖保持至少三個具有最小往返時間的鄰居，並極少更改此列表中的快速鄰居''。同時，它還有至少三個其他完全隨機選擇的慢速鄰居''，這樣覆蓋網路圖中始終會包含一個隨機子圖。這是為了維護連通性並防止覆

蓋網路分裂成多個未連接的區域子網路。同時還會選擇並保留至少三個中間鄰居''，其往返時間介於某個定值範圍內（實際上是快速鄰居和慢速鄰居的往返時間函數）。

透過這種方式，覆蓋網路的圖仍然保持足夠的隨機性以維持連通性，同時又進行了優化，以實現更低的延遲和更高的吞吐量。

3.3.11. Gossip protocols in an overlay network. 覆蓋網路經常用於運行所謂的 *gossip protocols*，這些協議在讓每個節點僅與其鄰居進行交互的同時實現某些全局目標。例如，存在用於構建（不太大的）覆蓋網路所有成員的近似列表，或者使用每個節點僅有的有限內存估計（任意大的）覆蓋網路成員數量的 gossip 協議（有關詳細信息，請參閱 [15, 4.4.3] 或 [1]）。

3.3.12. Overlay network as a broadcast domain. 在覆蓋網路中運行的最重要的 gossip 協議是廣播協議，旨在將網絡中任何節點生成的廣播消息或者指定發送節點之一生成的消息傳播給所有其他節點。

實際上，有幾種優化不同用例的廣播協議。其中最簡單的協議接收新的廣播消息，並將其中繼到尚未自己發送該消息副本的所有鄰居。

3.3.13. More sophisticated broadcast protocols. 有些應用可能需要更複雜的廣播協議。例如，對於廣播大量數據的消息，將新接收到的消息本身發送給鄰居可能並不合理，而是應該發送其雜湊值（或多個新消息的雜湊值集合）。在鄰居學習到之前未見過的消息雜湊值後，可以通過可靠的大型數據報協議（RLDP）進行傳輸，例如在 **3.1.9** 中所討論的那樣。通過這種方式，新消息將只能從一個鄰居處下載。

3.3.14. Checking the connectivity of an overlay network. 如果在覆蓋網路中存在一個已知節點（例如，覆蓋網路的“擁有者”或“創建者”），則可以檢查該覆蓋網路的連通性。然後，相應節點定期廣播包含當前時間、序列號和簽名的短消息。任何其他節點都可以確定它是否仍然連接到覆蓋網路，只需在不久前收到此類廣播即可。此協議可以擴展到多個已知節點的情況。例如，它們都將發送這樣的廣播，所有其他節點都期望從超過一半的已知節點接收廣播。

在用於傳播特定 shardchain 的新區塊（或僅是新的區塊標頭）的覆蓋網路中，節點檢查連通性的一種好方法是跟踪到目前為止接收到的最新區塊。因為通常每五秒會生成一個新的區塊，如果超過，比如說，三十秒沒有收到新的區塊，則該節點可能已從覆蓋網路斷開連接。

3.3.15. Streaming broadcast protocol. 最後，TON 覆蓋網絡中有一個流傳播協議（*streaming broadcast protocol*），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者。當然，他們為此目的創建了

一個私有覆蓋網絡。同樣的協議也可以用於將新的分片鏈區塊傳播到該分片鏈的所有完整節點。

此協議已在 **2.6.10**中概述：新的（大型）廣播消息被分成，例如 N 個一千字節的塊；這些塊的序列通過編碼技術（例如 Reed-Solomon 或者噴泉編碼（如 RaptorQ code [9] [14]））擴展為 $M \geq N$ 個塊，然後以升序的塊號流式傳輸到所有鄰居。參與的節點收集這些塊，直到他們能夠恢復原始的大型消息（至少要成功接收 N 個塊），然後指示其鄰居停止發送流的新塊，因為這些節點現在可以自行生成後續的塊，因為它們已經擁有原始消息的副本。除非鄰居再次表示這不再需要，否則這些節點將繼續生成流的後續塊並將其發送給鄰居。

這樣，節點在將消息進一步傳播之前就不需要完全下載大型消息。這最小化了廣播延遲，特別是當與 **3.3.10**中描述的優化結合使用時。

3.3.16. Constructing new overlay networks based on existing ones. 有時候我們不想從頭開始構建覆蓋網絡。相反，我們已知一個或多個先前存在的覆蓋網絡，並且預期新覆蓋網絡的成員資格會與這些覆蓋網絡的成員資格有顯著的重疊。

一個重要的例子是當一個 TON 分片鏈被分成兩個部分，或者兩個兄弟分片鏈被合併成一個（參見 **2.7**）。在第一種情況下，用於將新區塊傳播到完整節點的覆蓋網絡必須為每個新分片鏈分別建立；然而，這些新覆蓋網絡中的每一個都可以預期包含在原始分片鏈的區塊傳播網絡中（並且包含大約一半的成員）。在第二種情況下，用於傳播合併的分片鏈的新區塊的覆蓋網絡將大致由與被合併的兩個兄弟分片鏈相關的兩個覆蓋網絡的成員的聯合組成。

在這種情況下，新覆蓋網絡的描述可能包含與一個或多個相關現有覆蓋網絡的列表的明確或隱含引用。想加入新覆蓋網絡的節點可以檢查它是否已經是這些現有網絡之一的成員，並詢問這些網絡中的鄰居是否也對新網絡感興趣。如果得到肯定的答案，就可以建立新的點對點通道到這些鄰居，並將它們包括在新覆蓋網絡的鄰居列表中。

這個機制並不完全取代 **3.3.6**和**3.3.7**中描述的一般機制；相反，兩者並行運行，用於填充鄰居列表。這是為了防止新的覆蓋網絡被意外分裂成多個未連接的子網絡。

3.3.17. Overlay networks within overlay networks. 在實現 *TON Payments*（用於即時離線價值轉移的“閃電網絡”；參見 **5.2**）時出現了另一個有趣的情況。在這種情況下，首先構建包含所有“閃電網絡”的中轉節點的覆蓋網絡。然而，其中一些節點在區塊鏈中已經建立了支付通道；除了任何由一般覆蓋網絡算法選擇的“隨機”鄰居之外，它們必須始終是這個覆蓋網絡中的鄰居。這些與建立的支付通道的鄰居的“永久鏈接”用於運

行特定的閃電網絡協議，從而在包圍（幾乎始終連接的）覆蓋網絡內部有效地創建一個覆蓋子網絡（如果情況不對，可能不會連接）。

4 TON Services and Applications

我們已經廣泛討論了 TON 區塊鏈和 TON 網絡技術。現在，我們將解釋一些將它們結合起來創建各種服務和應用程序的方式，並討論 TON 項目本身將提供的一些服務，無論是從一開始還是在以後的某個時間。

4.1 TON Service Implementation Strategies

我們首先討論不同的區塊鏈和網絡相關應用和服務如何在 TON 生態系統中實現。首先，需要進行簡單的分類：

4.1.1. Applications and services. 我們將「應用程序」和「服務」這兩個詞語互換使用。然而，它們之間有一個微妙而有點模糊的區別：一個應用程序通常直接向人類用戶提供一些服務，而一個服務通常被其他應用程序和服務所利用。例如，TON 存儲是一個服務，因為它被設計來代表其他應用程序和服務存儲文件，即使一個人類用戶也可以直接使用它。如果一個假想的「區塊鏈上的 Facebook」（參見 **2.9.13**）或 Telegram 聊天應用程序通過 TON 網絡提供（即實現為一個「ton-service」；參見 **4.1.6**），則更可能是一個應用程序，即使一些「機器人」也可以在沒有人類干預的情況下自動訪問它。

4.1.2. Location of the application: on-chain, off-chain or mixed. 設計為 TON 生態系統的服務或應用程序需要在某個地方存儲其數據並處理這些數據。這導致了以下應用程序（和服務）的分類：

- 鏈上應用程序（參見 **4.1.4**）：所有數據和處理都在 TON 區塊鏈上。
- 離線應用程序（參見 **4.1.5**）：所有數據和處理都在 TON 網絡上可用的服務器上，而不在 TON 區塊鏈上。
- 混合應用程序（參見 **4.1.7**）：部分數據和處理在 TON 區塊鏈上，其餘部分在 TON 網絡上可用的離線服務器上。

4.1.3. Centralization: centralized and decentralized, or distributed, applications. 另一個分類標準是應用程序（或服務）是否依賴於集中式服務器集群，或者真正地「分散式」（參見 **4.1.9**）。所有的鏈上應用程序自動是去中心化和分散式的。離線和混合應用程序可能會展示不同程度的集中化。

現在讓我們更詳細地考慮上述可能性。

4.1.4. Pure "on-chain" applications: distributed applications, or "dapps", residing in the blockchain. 其中一個可能的方法，如**4.1.2**所述，是將"分散式應用程序"（通常簡稱為"dapp"）完全部署在 TON 區塊鏈上，作為一個智能合約或一組智能合約。所有數據都將作為這些智能合約的永久狀態的一部分保存，並且所有與該項目的互動都將通過（TON 區塊鏈）消息發送到這些智能合約或從這些智能合約接收消息來完成。

我們在**2.9.13**中已經討論了這種方法的缺點和限制。它也有優點：這樣的分散式應用程序不需要運行的服務器或存儲其數據（它在區塊鏈上運行——即在驗證者的硬件上），並享受區塊鏈的極高（拜占庭式）可靠性和可訪問性。這樣的分散式應用程序的開發人員不需要購買或租用任何硬件；她只需要開發一些軟件（即智能合約的代碼）。之後，她將有效地從驗證者中租用計算能力，並以 Gram 為代價支付，可以由她自己支付，也可以由她的用戶承擔這個負擔。

4.1.5. Pure network services: "ton-sites" and "ton-services". 另一個極端的選擇是在一些服務器上部署服務並通過**3.1**中描述的 ADNL 協議提供給用戶，也許還可以使用一些更高層次的協議，比如**3.1.9**中討論的 RLDP，可以用來以任何自定義格式向服務發送 RPC 查詢並獲取這些查詢的答案。這樣，服務將完全不在區塊鏈上，幾乎不使用 TON 區塊鏈，而是駐留在 TON 網絡上。

TON 區塊鏈可能僅用於定位服務的抽象地址或地址，如**3.2.12**所述，可能需要使用像 TON DNS（參見**4.3.1**）這樣的服務來促進類似於人類可讀的域名字符串到抽象地址的轉換。

在 ADNL 網絡（即 TON 網絡）類似於隱形網絡項目（I^2P）的情況下，這樣的（幾乎）純網絡服務類比於所謂的"eep 服務"（即以 I^2P 地址作為入口的服務，並通過 I^2P 網絡提供給客戶端）。我們將這些駐留在 TON 網絡上的純網絡服務稱為"ton 服務"。

"eep 服務"可以實現 HTTP 作為其客戶端-服務器協議；在 TON 網絡上下文中，"ton 服務"可能只使用 RLDP（參見**3.1.9**）數據包將 HTTP 查詢和響應傳輸到它們。如果它使用 TON DNS 允許通過人類可讀的域名查找其抽象地址，則與 Web 站點的類比幾乎完美。甚至可以編寫一個專門的瀏覽器，或者在用戶端本地運行的特殊代理（"ton 代理"），接受普通 Web 瀏覽器用戶任意的 HTTP 查詢（一旦將代理的本地 IP 地址和 TCP 端口輸入到瀏覽器的配置中），並通過 TON 網絡轉發這些查詢到服務的抽象地址。然後用戶將獲得與 World Wide Web（WWW）類似的瀏覽體驗。

在 I^2P 生態系統中，這樣的"eep 服務"被稱為"eep 站點"。在 TON 生態系統中，也可以輕鬆地創建"ton 站點"。這在某種程度上得到了像 TON DNS 這樣的服務的幫助，它利用 TON 區塊鏈和 TON DHT 將（TON）域

名轉換為抽象地址。

4.1.6. Telegram Messenger as a ton-service; MTProto over RLDP. 我們順帶提一下，Telegram Messenger[31]用於客戶端-服務器交互的 MTProto 協議[32]可以輕鬆嵌入到**3.1.9**中討論的 RLDP 協議中，從而有效地將 Telegram 變成一個 ton 服務。由於 TON 代理技術可以在 ton 站點或 ton 服務的最終用戶端之間透明地切換，實現在 RLDP 和 ADNL 協議的更低層級上（參見**3.1.6**），這將使 Telegram 變得非常難以阻擋。當然，其他消息傳遞和社交網絡服務也可能從這項技術中受益。

4.1.7. Mixed services: partly off-chain, partly on-chain. 有些服務可能會使用混合方法：大部分的處理都在區塊鏈外進行，但也有一些在鏈上的部分（例如，為了向用戶註冊其義務及其反向義務）。這樣，部分狀態仍將保存在 TON 區塊鏈中（即，一個不可變的公共賬本），服務或其用戶的任何不當行為都可以通過智能合約受到懲罰。

4.1.8. Example: keeping files off-chain; TON Storage. 這樣一種服務的例子是 TON 存儲（*TON Storage*）。在其最簡單的形式中，它允許用戶在鏈外存儲文件，只在鏈上保存要存儲的文件的哈希值，以及可能有一個智能合約，在該合約中，一些其他方同意以預先協商的費用在一定時間內保留相關的文件。實際上，文件可以被分成一些小塊（例如 1 千字節），並增加一種消除碼，例如 Reed-Solomon 或噴泉碼，可以構造出增強的塊序列的 Merkle 樹哈希，並且這個 Merkle 樹哈希可能會發布在智能合約中，代替或與文件的通常哈希一起。這有些類似於在 Torrent 中存儲文件的方式。

存儲文件的一種更簡單的方式是完全離線的：基本上可以為新文件創建一個“種子”，並使用 TON DHT 作為此“種子”的“分佈式種子追蹤器”（參見**3.2.10**）。對於流行的文件，這實際上可能非常有效。但是，這種方式無法保證可用性。例如，假設存在一個“區塊鏈 Facebook”（參見**2.9.13**），該社交媒體選擇將用戶的個人資料照片完全存儲在這樣的“種子”中，可能會冒失去普通（不是特別受歡迎的）用戶的照片的風險，或者至少冒長時間無法呈現這些照片的風險。相比之下，TON Storage 技術主要是離線的，但使用基於鏈上智能合約來強制實現存儲文件的可用性，可能更適合這個任務。

4.1.9. Decentralized mixed services, or “fog services”. 迄今為止，我們已討論了集中式的混合服務和應用程序。儘管它們的鏈上組件以去中心化和分散式的方式處理，位於區塊鏈中，但它們的鏈下組件仍然依賴於一些

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

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

由服務提供商控制的伺服器，採用了通常的集中式模式。計算能力可能不是使用一些專門的伺服器，而是從大型公司提供的雲計算服務中租用。但是，這並不會導致服務的鏈下組件去中心化。

將服務的鏈下組件實現去中心化的方法之一是創建一個市場，任何擁有所需硬體且願意租用其計算能力或磁碟空間的人都可以向需要這些服務的人提供其服務。

例如，可能存在一個註冊表（也可以稱為“市場”或“交易所”），在這個註冊表中，所有對保留其他用戶文件感興趣的節點都會公佈其聯繫信息以及其可用的存儲容量、可用性策略和價格。需要這些服務的人可能會在那裡查找這些信息，如果對方同意，則在區塊鏈中創建智能合約並上傳文件以進行鏈下存儲。通過這種方式，像 *TON Storage/*這樣的服務真正實現了去中心化，因為它不需要依賴任何集中式的伺服器集群來存儲文件。

4.1.10. Example: “fog computing” platforms as decentralized mixed services. 另一個這樣的去中心化混合應用的例子是當人們想要執行某些特定的計算時（例如 3D 渲染或訓練神經網絡），通常需要特定且昂貴的硬體。這時，那些擁有這些設備的人可以通過類似的“交易所”提供他們的服務，而需要這些服務的人將租用這些服務，雙方的義務將通過智能合約進行登記。這與“雲霧計算”平台（例如 Golem (`https://golem.network/`) 或 SONM (`https://sonm.io/`)）承諾提供的類似。

4.1.11. Example: TON Proxy is a fog service. *TON Proxy* 這提供了另一個“雲霧服務”的例子，其中希望提供其服務（帶或不帶報酬）作為 ADNL 網絡流量隧道的節點可以註冊，需要這些服務的人可以根據所提供的價格、延遲和帶寬選擇其中一個節點。之後，可以使用 *TON Payments/*提供的付款通道處理這些代理的微支付服務，例如每 128 KiB 轉移收取一次付款。

4.1.12. Example: TON Payments is a fog service. *TON Payments/*平台（參見 **5**）也是這樣一個去中心化混合應用的例子。

4.2 連接用戶和服務提供商

我們在 **4.1.9**中看到，“雲霧服務”（即去中心化混合服務）通常需要一些市場、交易所或註冊表，以便需要特定服務的人能夠找到提供這些服務的人。

這些市場可能會作為鏈上、鏈下或混合服務本身實現，可以是集中式的或分散式的。

4.2.1. Example: connecting to TON Payments. 例如，如果想要使用 TON Payments（參見 **5**），第一步是找到至少一些現有的“閃電網絡”轉

發節點（參見 **5.2**），如果他們願意，與它們建立付款通道。可以使用“包容性”的覆蓋網絡來找到一些節點，該網絡應該包含所有的閃電網絡轉發節點（參見 **3.3.17**）。然而，不清楚這些節點是否願意創建新的付款通道。因此，需要一個註冊表，節點可以在其中發布其聯繫信息（例如抽象地址），以顯示它們已準備好創建新的通道。

4.2.2. Example: uploading a file into TON Storage. 同樣地，如果想要將文件上傳到 TON Storage 中，必須找到一些願意簽署智能合約以綁定自己保存該文件副本（或者說任何大小在某個限制以下的文件副本）的節點。因此，需要一個提供文件存儲服務的節點註冊表。

4.2.3. On-chain, mixed and off-chain registries. 這樣一個服務提供者的註冊表可以完全在鏈上實現，通過一個智能合約將註冊表保存在其永久存儲中。但是，這將非常緩慢且昂貴。更高效的方法是使用混合方法，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（通過它們的抽象地址或它們的公鑰，這些公鑰可以用於定位實際的抽象地址，如 **3.2.12**所述），這些節點提供鏈下（集中式）註冊服務。

最後，一種純粹的鏈下去中心化方法可能包括一個公共覆蓋網絡（參見 **3.3**），在這個網絡中，願意提供其服務的人或尋找某人服務的人只需用其私鑰簽署其提供或需求的服務並進行廣播即可。如果要提供的服務非常簡單，甚至不需要廣播提供的服務：覆蓋網絡本身的近似成員資格可以用作願意提供特定服務的“註冊表”。然後，需要此服務的客戶端可以定位（參見 **3.3.7**）並查詢此覆蓋網絡的某些節點，如果已知的節點未能滿足其需求，則查詢其鄰居節點。

4.2.4. Registry or exchange in a side-chain. 實現去中心化混合註冊表的另一種方法是創建一個獨立的專門區塊鏈（“側鏈”），由其自己宣稱的驗證者集合維護，他們在鏈上的智能合約中發布其身份並向所有感興趣的方提供對此專門區塊鏈的網絡訪問，通過專用的覆蓋網絡（參見 **3.3**）收集交易候選人並廣播區塊更新。然後，此側鏈的任何全節點都可以維護其自己的共享註冊表副本（本質上等於此側鏈的全局狀態），並處理與此註冊表相關的任意查詢。

4.2.5. Registry or exchange in a workchain. 另一種選擇是在 TON 區塊鏈內部創建一個專門的工作鏈，專門用於創建註冊表、市場和交易所。這可能比使用基本工作鏈中的智能合約更有效率且更少花費。但是，這仍然比在側鏈中維護註冊表更昂貴（參見 **4.2.4**）。

4.3 Accessing TON Services

在 **4.1**中，我們討論了在 TON 生態系統中創建新服務和應用程序的不同方法。現在，我們討論如何訪問這些服務以及 TON 提供的一些“輔助服務”，包括 *TON DNS* 和 *TON Storage*。

4.3.1. TON DNS: a mostly on-chain hierarchical domain name service. *TON DNS*/是一個預定義的服務，它使用一組智能合約來維護從人可讀的域名到 ADNL 網絡節點和 TON 區塊鏈帳戶和智能合約（256 位地址）的映射。

儘管原則上任何人都可以使用 TON 區塊鏈實現此類服務，但有一個具有良好已知接口的預定義服務，可以在應用程序或服務希望將人可讀識別符轉換為地址時默認使用，這是很有用的。

4.3.2. TON DNS use cases. 例如，一位用戶想要向另一位用戶或商家轉移加密貨幣，可能更喜歡記住收款人的 TON DNS 域名，而不是記住其 256 位的帳戶識別符，並將它們複製並粘貼到其輕量級錢包客戶端的收款人欄位中。

同樣地，TON DNS 可用於定位智能合約的帳戶識別符或 ton-services 和 ton-sites 的入口點（參見 **4.1.5**），從而使得一個專門的客戶端（“ton-browser”）或一個普通的互聯網瀏覽器結合專門的 ton-proxy 擴展或獨立應用程序，為用戶提供類似 WWW 的瀏覽體驗。

4.3.3. TON DNS smart contracts. TON DNS 通過特殊（DNS）智能合約樹來實現。每個 DNS 智能合約負責註冊某個固定域的子域。位於主鏈中的“根”DNS 智能合約將保留 TON DNS 系統的一級域名。其帳戶識別符必須編碼到所有希望直接訪問 TON DNS 數據庫的軟件中。

任何 DNS 智能合約都包含一個哈希映射，將可變長度的以空字符結尾的 UTF-8 字符串映射到它們的“值”。該哈希映射實現為一個二進制 Patricia 樹，類似於 **2.3.7**中描述的那種樹，但支持可變長度位串作為鍵。

4.3.4. Values of the DNS hashmap, or TON DNS records. 至於值，它們是由 TL 方案描述的“TON DNS 記錄”（參見 **2.2.5**）。它們由“魔數”組成，選擇其中一個支持的選項，然後是帳戶識別符、智能合約識別符、抽象網絡地址（參見 **3.1**）、用於定位服務抽象地址的公鑰（參見 **3.2.12**）、網絡覆蓋的描述等。另一個重要的情況是另一個 DNS 智能合約的情況：在這種情況下，該智能合約用於解析其域名的子域名。通過這種方式，可以為不同的域名創建單獨的註冊表，由這些域名的所有者控制。

這些記錄還可以包含過期時間、緩存時間（通常很長，因為在區塊鏈中太頻繁地更新值是昂貴的），以及在大多數情況下對所討論的子域名的所有

者的引用。所有者有權更改此記錄（特別是所有者字段），從而將該域名轉移到其他人的控制下，並加以延長。

4.3.5. Registering new subdomains of existing domains. 為了註冊現有域名的新子域名，只需向該域名的註冊機構發送一條消息，該消息包含要註冊的子域名（即鍵）、以幾種預定義的格式之一表示的值、所有者的身份、過期日期以及由該域名的所有者確定的一定量的加密貨幣。

子域名是按照“先到先得”的原則進行註冊的。

4.3.6. Retrieving data from a DNS smart contract. 原則上，只要包含 DNS 智能合約的主鏈或分片鏈的任何全節點都可能能夠查找該智能合約數據庫中的任何子域名，如果已知哈希映射在智能合約的持久存儲內部的結構和位置。

然而，這種方法僅適用於某些 DNS 智能合約。如果使用非標準的 DNS 智能合約，這種方法會失敗。

相反，採用了基於智能合約通用接口和“get 方法”（參見 **4.3.11**）的方法。任何 DNS 智能合約必須定義一個“已知簽名”的“get 方法”，用於查找鍵。由於這種方法對於其他智能合約也是有意義的，特別是那些提供鏈上和混合服務的智能合約，因此我們在 **4.3.11**中對其進行了詳細的解釋。

4.3.7. Translating a TON DNS domain. 一旦任何全節點可以代表某些輕客戶端或自己查找任何 DNS 智能合約的數據庫中的條目，任意 TON DNS 域名就可以從眾所周知且固定的根 DNS 智能合約（帳戶）識別符開始遞歸地轉換。

例如，如果要轉換 `A.B.C`，則需要在根域數據庫中查找鍵 `.C`、`.B.C` 和 `A.B.C`。如果第一個鍵沒有找到，但是第二個鍵存在，且其值是另一個 DNS 智能合約的引用，那麼就需要在該智能合約的數據庫中查找 `A` 並檢索最終值。

4.3.8. Translating TON DNS domains for light nodes. 這樣，主鏈的完整節點以及參與域名查找過程的所有分片鏈的完整節點都可以在不需要外部幫助的情況下將任何域名轉換為其當前值。輕節點可能會請求一個完整節點代表它進行此操作並返回值，以及一個 Merkle 證明（參見 **2.5.11**）。這個 Merkle 證明可以使輕節點驗證答案是否正確，因此此類 TON DNS 響應不能被惡意攔截器“欺騙”，這與通常的 DNS 協議不同。

由於沒有節點可以被期望對所有分片鏈都是完整節點，實際的 TON DNS 域名轉換將涉及這兩種策略的結合。

4.3.9. Dedicated “TON DNS servers”. 可以提供一個簡單的“TON DNS 服務器”，該服務器將通過 RPC“DNS”查詢（例如通過 **3.1**中描述

的 ADNL 或 RLDP 協議）收到請求，要求該服務器轉換給定的域名，必要時轉發一些子查詢到其他（完整）節點，並在需要時返回帶有 Merkle 證明的原始查詢答案。

這些“DNS 服務器”可以向任何其他節點和特別是輕客戶端提供它們的服務（免費或不免費），使用 **4.2**中描述的其中一種方法。請注意，如果將這些服務器視為 TON DNS 服務的一部分，它們將有效地將其從一個分散的鏈上服務轉變為一個分散的混合服務（即“fog 服務”）。

這樣我們的簡要概述就結束了，對 TON DNS 服務的介紹，這是 TON Blockchain 和 TON Network 實體的可擴展鏈上註冊表，用於人類可讀的域名。

4.3.10. Accessing data kept in smart contracts. 我們已經看到，有時需要訪問存儲在智能合約中的數據，而不更改其狀態。

如果知道智能合約實現的細節，可以從智能合約的持久存儲中提取所有所需信息，這些信息對該智能合約所在的分片鏈的所有完整節點都是可用的。但是，這是一種相當不優雅的做法，非常依賴智能合約的實現。

4.3.11. “Get methods” of smart contracts. 更好的方法是在智能合約中定義一些「獲取方法」。也就是一些入站消息的類型，當其被傳送時不會影響智能合約的狀態，但會生成一個或多個輸出消息，包含獲取方法的結果。這樣，只要知道智能合約實現了具有已知簽名的獲取方法（即已知要發送的入站消息和要接收的出站消息的格式），就可以從智能合約中獲取數據。

現在的作法比較優雅，符合物件導向程式設計的概念。但它仍有一個明顯的缺點：需要實際提交一個交易到區塊鏈上（將 get 消息發送到智能合約），等待它被驗證節點確認並處理，從新區塊中提取答案，以及支付 gas 費用（即在驗證者的硬件上執行 get 方法）。這樣會浪費資源，因為 get 方法不會改變智能合約的狀態，因此它們不需要在區塊鏈上執行。

4.3.12. Tentative execution of get methods of smart contracts. 我們已經提到（參見 **2.4.6**），任何全節點都可以在給定智能合約狀態的基礎上，暫時執行任何智能合約的任何方法（即向智能合約發送任何消息），而無需實際提交相應的交易。全節點可以將要考慮的智能合約代碼加載到 TON VM 中，從片段鏈的全局狀態（由片段鏈的所有全節點知道）初始化其持久性存儲，並將接收到的消息作為輸入參數執行智能合約代碼。產生的輸出消息將產生此計算的結果。

通過這種方式，任何全節點都可以評估任意智能合約的任意 get 方法，前提是它們的簽名（即入站和出站消息的格式）已知。該節點可以跟踪在

此評估過程中訪問的片段鏈狀態的單元格，並為輕節點（可能已要求全節點這樣做，參見 **2.5.11**）的利益而創建所執行計算的默克爾證明。

4.3.13. Smart-contract interfaces in TL-schemes. 回想一下，智能合約實現的方法（即其接受的輸入消息）本質上是一些 TL 序列化對象，可以通過 TL 方案進行描述（參見 **2.2.5**）。生成的輸出消息也可以用同樣的 TL 方案進行描述。通過這種方式，智能合約向其他帳戶和智能合約提供的接口可以通過 TL 方案進行形式化。

特別是，智能合約支持的（子集）get 方法可以通過這種形式化的智能合約接口進行描述。

4.3.14. Public interfaces of a smart contract. 請注意，形式化的智能合約接口可以以 TL 方案（表示為 TL 源代碼文件；參見 **2.2.5**）或序列化形式的方式發布。例如，可以在智能合約帳戶描述的特殊字段中存儲在區塊鏈上，或者單獨存儲，如果這個接口將被引用多次。在後一種情況下，支持的公共接口的哈希值可以納入智能合約描述中，而不是接口描述本身。

這樣一個公共接口的例子是 DNS 智能合約的接口，它應該實現至少一個標準的 get 方法來查找子域名（參見 **4.3.6**）。註冊新子域名的標準方法也可以包含在 DNS 智能合約的標準公共接口中。

4.3.15. User interface of a smart contract. 智能合約的公共接口的存在還有其他好處。例如，錢包客戶端應用程序可以在用戶的請求下檢查智能合約時下載這樣一個接口，並顯示智能合約支持的公共方法列表（即可用操作），如果在形式接口中提供了任何人可讀的注釋，可能會顯示這些注釋。在用戶選擇這些方法之一之後，可以根據 TL 方案自動生成一個表單，其中用戶將被提示輸入所選方法所需的所有字段以及所需的加密貨幣（例如 Grams）的金額。提交此表單將創建一個新的區塊鏈交易，其中包含剛剛組成的消息，從用戶的區塊鏈帳戶發送。

通過這種方式，如果這些智能合約發布了它們的接口，用戶將能夠通過填寫和提交某些表單以用戶友好的方式從錢包客戶端應用程序與任意智能合約進行交互。

4.3.16. User interface of a "ton-service". 事實證明，"ton-services"（即位於 TON Network 中並通過 **3**中的 ADNL 和 RLDP 協議接受查詢的服務；參見 **4.1.5**）也可以從具有由 TL 方案（參見 **2.2.5**）描述的公共接口中受益。客戶端應用程序（例如輕量級錢包或 "ton-browser"）可能提示用戶選擇其中一種方法，並使用由接口定義的參數填寫表單，與 **4.3.15**中所討論的類似。唯一的區別在於，產生的 TL 序列化消息不是作為區塊鏈中的交易提交的，而是作為 RPC 查詢發送到相應的 "ton-service" 的抽象地址，對此查詢的回應被解析並根據形式接口（即 TL 方案）顯示。

4.3.17. Locating user interfaces via TON DNS. TON DNS 記錄包含 ton-service 的抽象地址或智能合約帳戶標識符，還可以包含一個可選字段，描述該實體的公共（用戶）接口，或者多個支持的接口。然後，客戶端應用程序（無論是錢包、ton-browser 還是 ton-proxy）將能夠下載接口並以統一的方式與該實體（無論是智能合約還是 ton-service）進行交互。

4.3.18. Blurring the distinction between on-chain and off-chain services. 通過這種方式，對於最終用戶，on-chain、off-chain 和混合服務之間的區別（參見 **4.1.2**）被模糊化了：她只需將所需服務的域名輸入到她的 ton-browser 或錢包的地址欄中，其餘部分將由客戶端應用程序無縫處理。

4.3.19. A light wallet and TON entity explorer can be built into Telegram Messenger clients. 此時出現了一個有趣的機會。輕量級錢包和 TON 實體探索器可以實現上述功能，並嵌入到 Telegram Messenger 智能手機客戶端應用程序中，從而將技術帶給超過 2 億人。用戶將能夠通過在消息中包含 TON URI（參見 **4.3.22**）來向 TON 實體和資源發送超鏈接；如果選中此類超鏈接，將由接收方的 Telegram 客戶端應用程序在內部打開，並開始與所選實體進行交互。

4.3.20. "ton-sites" as ton-services supporting an HTTP interface. "ton-site" 只是支持 HTTP 接口（或者其他接口）的 ton-service。這種支持可以在相應的 TON DNS 記錄中宣布。

4.3.21. Hyperlinks. 需要注意的是，ton-site 返回的 HTML 頁面可能包含 "ton-hyperlinks"---也就是通過特別設計的 URI 方案（參見 **4.3.22**）引用其他 ton-sites、智能合約和帳戶的鏈接，包含抽象網絡地址、帳戶標識符或可讀的 TON DNS 域名。然後，當用戶選擇該超鏈接時，"ton-browser" 可能會跟隨該鏈接，檢測要使用的接口，並像**4.3.15**和 **4.3.16**中概述的那樣顯示用戶界面表單。

4.3.22. Hyperlink URLs may specify some parameters. 超鏈接 URL 可能不僅包含服務的（TON）DNS 域名或抽象地址，還可能包含要調用的方法名以及其一些或所有參數。其 URI 方案可能如下所示：

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

當用戶在 ton-browser 中選擇這樣的鏈接時，要麼立即執行操作（特別是如果它是匿名調用智能合約的 get 方法），要麼顯示一個部分填充的表單，需要用戶明確確認並提交（對於付款表單可能需要這樣做）。

4.3.23. POST actions. 一個 ton-site 可能會將一些通常看起來像是 POST 表單嵌入到其返回的 HTML 頁面中，POST 操作引用由適當的（TON）URL 引用的 ton-site、ton-service 或智能合約。在這種情況下，一旦用戶填寫並提交了自定義表單，將採取相應的操作，可以立即採取操作或在明確確認後採取操作。

4.3.24. TON WWW. 所有上述內容都將導致在 TON 網絡中創建一整個互相參照的實體網絡，用戶可以通過 ton 瀏覽器訪問這個網絡，從而獲得類似於 WWW 的瀏覽體驗。對於終端用戶而言，這最終將使區塊鏈應用與他們已經習慣的網站基本相似。

4.3.25. Advantages of TON WWW. 這個由鏈上和鏈下服務組成的“TON WWW”與其傳統對應物相比具有一些優勢。例如，支付在系統中內在地得到了整合。用戶身份始終可以向服務提供商呈現（通過自動生成的交易和 RPC 請求上的簽名），或者按需隱藏。服務不需要檢查和重新檢查用戶的憑據；這些憑據可以一次性地在區塊鏈上發布。用戶的網絡匿名性可以通過 TON 代理輕松保持，且所有服務都將有效地無法屏蔽。微支付也是可能且易於實現的，因為 ton 瀏覽器可以與 TON Payments 系統集成。

5 TON Payments

TON 項目中最後一個我們將在本文中簡要討論的組件是 TON Payments，這是一個用於（微）支付通道和“閃電網絡”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中，支付相關的交易費用（例如，用於消耗的 gas），並等待 5 秒，直到包含所述交易的區塊得到確認。

這種即時支付的整體開銷非常小，可以用於微支付。例如，TON 文件存儲服務可能會按每 128 KiB 下載的數據收取費用，或者付費的 TON 代理可能會要求每 128 KiB 中繼的流量收取一些微小的微支付。

儘管 *TON Payments/*可能比 TON 項目的核心組件發佈時間晚，但一些考慮需要在一開始就進行。例如，用於執行 TON 區塊鏈智能合約的代碼的 TON 虛擬機（TON VM；參見**2.1.20**）必須支持一些具有 Merkle 證明的特殊操作。如果在原始設計中沒有此支持，稍後添加可能會變得棘手（參見**2.8.16**）。然而，我們將看到，TON VM 在開箱即用時即帶有對「智能」付款通道的自然支持（參見**5.1.9**）。

5.1 Payment Channels

我們首先討論點對點付款通道及其在 TON 區塊鏈中的實現方式。

5.1.1. The idea of a payment channel. 假設兩個派對 A 和 B 知道他們未來將需要彼此進行許多付款。他們創建了一個共享的「資金池」（或者可能是具有兩個帳戶的小型私人銀行），並向其中捐贈了一些資金：A 貢獻了 a 個代幣，B 貢獻了 b 個代幣。這是通過在區塊鏈中創建一個特殊的智能合約並將資金發送到其中來實現的。

在創建「資金池」之前，雙方同意遵守某種協議。他們將跟踪池的狀態，即共享池中他們的餘額。最初，狀態是 (a, b)，表示實際上有 a 個代幣屬於 A，b 個代幣屬於 B。然後，如果 A 想向 B 支付 d 個代幣，他們可以簡單地同意新狀態是 $(a', b') = (a - d, b + d)$。然後，如果，比如，$B$ 想向 A 支付 d' 個代幣，狀態將變為 $(a'', b'') = (a' + d', b' - d')$，以此類推。

所有這些在池內的餘額更新都是完全在鏈下完成的。當兩個派對決定從池中提取他們的應有資金時，他們根據池的最終狀態進行提款。這是通過向智能合約發送一條特殊消息來實現的，該消息包含了已經協商過的最終狀態 (a^*, b^*) 以及 A 和 B 的簽名。然後，智能合約將 a^* 個代幣發送給 A，b^* 個代幣發送給 B，並自我銷毀。

這個智能合約與 A 和 B 用於更新池狀態的網路協議構成了一個簡單的A 和 B 之間的付款通道。根據**4.1.2**中描述的分類，它是一個混合服務：它的部分狀態存儲在區塊鏈中（智能合約），但它的大部分狀態更新是在鏈

下進行的（通過網路協議）。如果一切順利，這兩方將能夠彼此進行任意次數的付款（唯一的限制是付款通道的「容量」不會超過限制——即，它們在付款通道中的餘額仍然保持非負），僅提交兩個交易到區塊鏈中：一個用於打開（創建）付款通道（智能合約），另一個用於關閉（銷毀）它。

5.1.2. Trustless payment channels. 前面的例子有點不切實際，因為它假設兩個派對都願意合作，永遠不會作弊以獲得某些優勢。例如，假設 A 不願意簽署最終餘額 (a', b')，其中 $a' < a$，這將使 B 陷入困境。

為了防止這種情況，人們通常會試圖開發不需要派對互相信任的付款通道協議，並提供懲罰任何企圖作弊的派對的規定。

這通常是通過簽名來實現的。付款通道智能合約知道 A 和 B 的公鑰，如果需要，它可以檢查他們的簽名。付款通道協議要求各方簽署中間狀態並將簽名發送給對方。然後，如果其中一方作弊——例如，假裝付款通道的某個狀態從未存在過——它的不良行為可以通過顯示它在該狀態上的簽名來證明。付款通道智能合約作為一個「在鏈仲裁者」，能夠處理兩方關於對方的投訴，並通過沒收作弊方的所有資金並將其獎勵給對方來懲罰其罪行。

5.1.3. Simple bidirectional synchronous trustless payment channel. 考慮以下更現實的例子：假設付款通道的狀態可以由三元組 (δ_i, i, o_i) 描述，其中 i 是狀態的序列號（最初為零，然後在出現後續狀態時增加一），δ_i 是通道不平衡度（表示 A 和 B 分別擁有 $a + \delta_i$ 和 $b - \delta_i$ 個代幣），o_i 是允許生成下一個狀態的派對（A 或 B）。在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 想在付款通道內將 d 個代幣轉移給 B，且當前狀態為 $S_i = (\delta_i, i, o_i)$，其中 $o_i = A$，那麼它只需創建一個新狀態 $S_{i+1} = (\delta_i - d, i+1, o_{i+1})$，簽署它，並將其發送給 B 以及自己的簽名。然後，B 通過簽署並向 A 發送其簽名的副本來確認它。之後，雙方都有一個帶有他們雙方簽名的新狀態的副本，可以進行新的轉移。

如果 A 想在狀態 S_i 中向 B 轉移代幣，其中 $o_i = B$，那麼它首先會要求 B 提交一個具有相同不平衡度 $\delta_{i+1} = \delta_i$，但 $o_{i+1} = A$ 的後續狀態 S_{i+1}。之後，A 將能夠進行轉移。

當兩個派對同意關閉付款通道時，他們都會在他們認為是最終狀態 S_k 上放置他們的特殊最終簽名，並通過向付款通道智能合約發送最終狀態以及兩個最終簽名來調用付款通道智能合約的清理或雙方最終化方法。

如果另一方不同意提供其最終簽名，或者只是停止回應，可以單方面關閉通道。為此，希望這樣做的一方將調用單方最終化方法，向智能合約發送其版本的最終狀態、其最終簽名以及具有其他派對簽名的最新狀態。之後，智能合約不會立即對接收到的最終狀態進行操作。相反，它會等待一定時間（例如一天），以便其他派對提交其版本的最終狀態。當其他派對提

交其版本並且與已經提交的版本兼容時，智能合約會計算出「真實」的最終狀態，並用於相應地分配貨幣。如果其他派對未能向智能合約提交其版本的最終狀態，那麼貨幣將根據唯一的最終狀態副本重新分配。

如果其中一方作弊，例如將兩個不同的狀態簽署為最終狀態，或者簽署兩個不同的下一個狀態 S_{i+1} 和 $S'i+1$，或者簽署一個無效的新狀態 $Si+1$（例如，不平衡度 $\delta_{i+1} < -a$ 或 $> b$）---那麼另一方可以向智能合約的第三種方法提交此不當行為的證據。有罪方將立即被懲罰，完全失去其在付款通道中的份額。

這種簡單的付款通道協議在任何情況下都是公平的，任何一方都可以始終按時獲得其應得的款項，無論是否得到對方的合作，並且如果它試圖作弊，很可能會失去其承諾給付款通道的所有資金。

5.1.4. Synchronous payment channel as a simple virtual blockchain with two validators. 上述簡單同步付款通道的示例可以重新設計如下。想象一下，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序列。此區塊鏈的每個區塊基本上僅包含區塊鏈的當前狀態，以及可能是先前區塊的參考（即，其哈希）。雙方 A 和 B 均作為此區塊鏈的驗證者，因此每個區塊必須收集它們兩個人的簽名。區塊鏈的狀態 S_i 定義了下一個區塊的指定生成器 o_i，因此 A 和 B 之間沒有爭奪下一個區塊的競爭。生成器 A 可以創建轉移資金從 A 到 B 的區塊（即減少不平衡度：$\delta_{i+1} \leq \delta_i$），而 B 只能從 B 向 A 轉移資金（即增加 δ）。

如果兩個驗證者對區塊鏈的最終區塊（和最終狀態）達成共識，則通過收集兩個方的特殊“最終”簽名並將其提交到通道智能合約進行處理，以重新分配貨幣。

如果驗證者簽署了無效的區塊，或創建了分支，或簽署了兩個不同的最終區塊，則可以通過向智能合約展示其不良行為的證據來受到懲罰，該合約作為兩個驗證者的“在鏈仲裁者”。接著，違規方將失去其在支付通道中保留的所有貨幣，這類似於驗證者失去其份額。

5.1.5. Asynchronous payment channel as a virtual blockchain with two workchains. 在**5.1.3**中討論的同步支付通道存在某些缺點：在另一方確認前，無法開始下一筆交易（在支付通道內進行的貨幣轉移）。可以通過將**5.1.4**中討論的單個虛擬區塊鏈替換為兩個交互作用的虛擬工作鏈系統（或更準確地說是碎片鏈）來解決這個問題。

其中一個工作鏈只包含 A 的交易，只有 A 可以生成其區塊；其狀態為 $S_i = (i, \phi_i, j, \psi_j)$，其中 i 是區塊序列號（即到目前為止由 A 執行的交易或資金轉移次數），ϕ_i 是到目前為止從 A 轉移給 B 的總金額，j 是 B 的區塊鏈中 A 知道的最新有效區塊的序列號，ψ_j 是在 B 的 j 個交易中轉移給 A 的金額。B 在其第 j 個區塊上的簽名也應是此狀態的一部分。此工作鏈的

前一個區塊的哈希值和另一個工作鏈的第 j 個區塊的哈希值也可以包含在內。S_i 的有效條件包括 $\phi_i \geq 0$，如果 $i > 0$，則 $\phi_i \geq \phi_{i-1}$，$\psi_j \geq 0$，以及 $-a \leq \psi_j - \phi_i \leq b$。

同樣地，第二個工作鏈只包含 B 的交易，只有 B 可以生成其區塊；其狀態為 $T_j = (j, \psi_j, i, \phi_i)$，具有類似的有效條件。

現在，如果 A 想要向 B 轉移一些金錢，它只需在其工作鏈中創建一個新區塊，簽署它並發送給 B，而不必等待確認。

付款通道由 A 簽署其區塊鏈的（其版本的）最終狀態（使用其特殊的「最終簽名」）以及 B 簽署其區塊鏈的最終狀態來完成，並將這些最終狀態呈現給付款通道智能合約的「清潔最終化」方法。單方面最終化也是可能的，但在這種情況下，智能合約必須等待對方在某些寬限期內呈現其版本的最終狀態。

5.1.6. Unidirectional payment channels. 如果只有 A 需要向 B 進行付款（例如，B 是服務提供商，而 A 是其客戶），則可以創建一個單方面的付款通道。基本上，它只是**5.1.5**中描述的第一個工作鏈，沒有第二個工作鏈。相反，可以說**5.1.5**中描述的異步付款通道由同一個智能合約管理的兩個單向付款通道或「半通道」組成。

5.1.7. More sophisticated payment channels. Promises. 我們將在**5.2.4**中稍後看到，「閃電網路」（參見**5.2**）通過多個付款通道的鏈條實現即時資金轉移，需要付款通道具有更高的複雜度。

特別地，我們希望能夠承諾「條件式資金轉移」：A 同意向 B 轉移 c 個幣，但只有在某些條件得到滿足時 B 才會得到這筆錢，例如，如果 B 可以提供一個字串 u 使得 $\textsc{Hash}(u) = v$，其中 v 是已知的值。否則，在一段時間後 A 可以取回這筆錢。

這樣的承諾可以通過一個簡單的智能合約在鏈上輕鬆實現。然而，我們希望能夠在付款通道中實現承諾和其他類型的條件式資金轉移，因為它們可以大大簡化存在於「閃電網路」中的付款通道鏈中的資金轉移（參見**5.2.4**）。

在**5.1.4**和**5.1.5**中概述的「付款通道作為簡單區塊鏈」圖像在這裡變得很方便。現在，我們考慮一個更複雜的虛擬區塊鏈，其狀態包含一組未履行的「承諾」和鎖定在此類承諾中的資金量。這個區塊鏈（在異步情況下為兩個工作鏈）將不得不通過它們的哈希明確地引用先前的區塊。然而，一般機制保持不變。

5.1.8. Challenges for the sophisticated payment channel smart contracts. 需要注意的是，儘管複雜的付款通道的最終狀態仍然很小，而且「清潔」的最終化是簡單的（如果雙方已經就其應付的金額達成協議，並且

兩者已簽署了協議，剩下的就沒有其他事情可做了），但單方面的最終化方法和懲罰欺詐行為的方法需要更複雜。事實上，它們必須能夠接受有關不當行為的默克爾證明，並檢查付款通道區塊鏈的更複雜交易是否已經正確處理。

換句話說，付款通道智能合約必須能夠使用 Merkle 證明來檢查其「哈希有效性」，並且必須包含支付通道（虛擬）區塊鏈的 *ev_trans* 和 *ev_block* 函數的實現（參見**2.2.6**）。

5.1.9. TON VM support for "smart" payment channels. 用於運行 TON 區塊鏈智能合約的 TON 虛擬機可以應對執行「智能」或複雜的付款通道所需的智能合約的挑戰（參見**5.1.8**）。

此時「一切皆為單元袋」的範式（參見**2.5.14**）變得非常方便。由於所有區塊（包括短暫的支付通道區塊鏈的區塊）都被表示為單元袋（並由一些代數數據類型描述），而且對於消息和默克爾證明也是如此，因此默克爾證明可以很容易地嵌入到發送給支付通道智能合約的入站消息中。默克爾證明的「哈希條件」將自動進行檢查，當智能合約訪問所呈現的「Merkle 證明」時，它將像處理對應代數數據類型的值一樣處理它，儘管它是不完整的，其中一些子樹被替換為包含省略子樹的 Merkle 哈希的特殊節點。然後智能合約將處理該值，該值可能表示支付通道（虛擬）區塊鏈的一個區塊及其狀態，並將對此區塊和先前狀態的該區塊鏈的 *ev_block* 函數（參見**2.2.6**）進行評估。然後，如果計算完成，最終狀態可以與區塊中斷言的狀態進行比較，或者當嘗試訪問缺失子樹時拋出「缺失節點」異常，這表示該 Merkle 證明無效。

這種方法讓智能支付通道區塊鏈驗證代碼的實現變得相當簡單，並使用 TON 區塊鏈智能合約。我們可以說，*TON* 虛擬機器內建了檢查其他簡單區塊鏈的有效性的支援。唯一的限制因素是將 Merkle 證明合併到傳入智能合約（即交易）中的大小。

5.1.10. Simple payment channel within a smart payment channel. 我們想討論在現有支付通道內創建一個簡單（同步或異步）支付通道的可能性。

雖然這可能看起來有些繁複，但理解和實現它並不比**5.1.7**中討論的「承諾」更難。基本上，A 承諾根據某些其他（虛擬的）付款通道區塊鏈的最終結算向 B 支付多達 c 個硬幣，而不是承諾在提交某些哈希問題的解決方案時向對方支付 c 個硬幣。一般來說，這個其他付款通道區塊鏈甚至不需要在 A 和 B 之間；它可能涉及到其他方，比如 C 和 D，他們願意分別將 c 和 d 個硬幣投入到他們的簡單付款通道中。（這種可能性稍後在**5.2.5**中加以利用。）〈〈

如果包含的付款通道是非對稱的，則需要在兩個工作鏈中承諾兩個事項：如果「內部」簡單付款通道的最終結算產生負的最終不平衡 δ，且 $0 \leq -\delta \leq c$，A 將承諾向 B 支付 $-\delta$ 個硬幣；如果 δ 是正的，B 將不得不承諾向 A 支付 δ 個硬幣。另一方面，如果包含的付款通道是對稱的，這可以通過將參數 (c, d) 的「簡單付款通道創建」交易提交到單個的付款通道區塊鏈中進行，然後由 B 提交一個特殊的「確認交易」（凍結 B 的 d 個硬幣）來實現。

我們預計內部付款通道非常簡單（例如，**5.1.3**中討論的簡單同步付款通道），以最小化要提交的 Merkle 證明的大小。外部付款通道將必須在**5.1.7**中描述的意義上「智能」。

5.2 Payment Channel Network, or "Lightning Network"

現在，我們已準備討論 TON 支付的 "閃電網絡"，該網絡可在任何兩個參與節點之間實現即時的資金轉移。

5.2.1. Limitations of payment channels. 支付通道對於預期彼此之間有許多資金轉移的各方非常有用。然而，如果一方只需要向特定收款人轉帳一次或兩次，那麼與其建立支付通道將是不切實際的。除其他外，這將意味著在支付通道中凍結大量資金，而且仍需要至少兩筆區塊鏈交易。

5.2.2. Payment channel networks, or "lightning networks". 支付通道網絡通過啟用沿著支付通道的鏈的資金轉移，克服了支付通道的限制。如果 A 想要向 E 轉帳，她不需要與 E 建立支付通道。只需要擁有一條支付通道鏈路，通過幾個中間節點將 A 與 E 相連，比如說四個支付通道：從 A 到 B，從 B 到 C，從 C 到 D，再從 D 到 E。

5.2.3. Overview of payment channel networks. 回想一下，支付通道網絡，也稱為 "閃電網絡"，由一組參與節點組成，其中一些節點之間建立了長期的支付通道。稍後我們將看到，這些支付通道必須在 **5.1.7** 中是 "智能" 的。當參與節點 A 想要向任何其他參與節點 E 轉移資金時，她會嘗試在支付通道網絡中找到將 A 連接到 E 的路徑。當找到這樣的路徑時，她會沿著此路徑進行 "鏈式轉帳"。

5.2.4. Chain money transfers. 假設存在一條支付通道鏈路，從 A 到 B，從 B 到 C，從 C 到 D，最後從 D 到 E。進一步假設 A 想要向 E 轉移 x 個代幣。

一種簡單的方法是沿著現有的支付通道將 x 個代幣轉移到 B，並要求他將資金進一步轉移給 C。然而，並不明顯為什麼 B 不會簡單地拿走這筆

資金。因此，必須採用一種更複雜的方法，而不要求所有相關方之間彼此信任。

可以通過以下方式實現。A 生成一個大隨機數 u，並計算其雜湊 $v = \text{Hash}(u)$。然後，她在與 B 的支付通道內創建一個承諾，即如果出現一個具有雜湊 v 的數字 u（參見 **5.1.7**），則支付 x 個代幣給 B。此承諾包含 v，但不包含仍然保持機密的 u。

此後，B 在他們的支付通道中創建一個類似的承諾，向 C 承諾支付 x 個代幣。他不害怕給出這樣的承諾，因為他知道 A 已經給了他一個類似的承諾。如果 C 曾經提供了一個解決方案來收集 B 承諾的 x 個代幣，那麼 B 將立即向 A 提交此解決方案，以從 A 那裡收集 x 個代幣。

然後創建了 C 向 D 和 D 向 E 的類似承諾。當承諾都到位時，A 通過向所有相關方通信解決方案 u（或僅向 E）來觸發轉移。

此描述中省略了一些細節。例如，這些承諾必須具有不同的過期時間，並且沿著鏈路所承諾的金額可能會略有不同（B 可能只向 C 承諾支付 $x-\epsilon$ 個代幣，其中 ϵ 是一個小的預先協定的過境費）。我們暫時忽略這些細節，因為對於理解支付通道的工作原理以及如何在 TON 中實現它們並不太相關。

5.2.5. Virtual payment channels inside a chain of payment channels. 現在假設 A 和 E 預期彼此之間要進行許多支付。他們可能會在區塊鏈中之間建立一個新的支付通道，但這仍然相當昂貴，因為某些資金將被鎖定在這個支付通道中。另一個選擇是對每次支付都使用 **5.2.4** 中描述的鏈式轉帳。但是，這將涉及大量的網絡活動和涉及所有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絡中連接 A 和 E 的鏈路中創建一個虛擬支付通道。為此，A 和 E 創建了一個（虛擬）區塊鏈，用於他們的支付，就好像他們將在區塊鏈中創建一個支付通道。但是，他們不是在區塊鏈中創建支付通道智能合約，而是要求所有中間支付通道——即連接 A 到 B，B 到 C 等的支付通道——在其中創建簡單的支付通道，並與 A 和 E 創建的虛擬區塊鏈相關聯（參見 **5.1.10**）。換句話說，現在每個中間支付通道中都存在一個根據 A 和 E 的最終結算進行資金轉移的承諾。

如果虛擬支付通道是單向的，那麼可以相當容易地實現此類承諾，因為最終的不平衡量 δ 將是非正的，因此可以按照 **5.2.4** 中所述的相同順序在中間支付通道內創建簡單的支付通道。它們的過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，情況就稍微複雜了。在這種情況下，應將根據最終結算進行的資金轉移的承諾分成兩個半承諾，如 **5.1.10** 中所述：在正向方向上轉移 $\delta^- = \max(0, -\delta)$ 個代幣，並在反向方向上轉移

$\delta^+ = \max(0, \delta)$ 個代幣。這些半承諾可以在中間支付通道中獨立創建，一個半承諾鏈路從 A 到 E，另一個半承諾鏈路從相反方向創建。

5.2.6. Finding paths in the lightning network. 到目前為止，還有一個點未討論：A 和 E 如何在支付網絡中找到連接它們的路徑？如果支付網絡不是太大，可以使用類似 OSPF 的協議：支付網絡的所有節點創建一個疊加網絡（參見 **3.3.17**），然後每個節點通過一個八卦協議向其鄰居傳播所有可用的鏈路（即參與的支付通道）信息。最終，所有節點都將擁有所有參與支付網絡的支付通道的完整列表，並且將能夠自行找到最短路徑。例如，通過應用修改後考慮到所涉及的支付通道的"容量"（即可以轉移的最大金額）的 Dijkstra 算法的一個版本。找到候選路徑後，可以通過一個特殊的 ADNL 數據報文進行探測，其中包含完整的路徑，要求每個中間節點確認所涉及的支付通道的存在並根據路徑進一步轉發此數據報文。之後，可以構建一個鏈路，並運行一個鏈式轉移協議（參見 **5.2.4**），或在支付通道鏈路中創建一個虛擬支付通道的協議（參見 **5.2.5**）。

5.2.7. Optimizations. 這裡可以進行一些優化。例如，僅需要轉發網絡的中繼節點參與在 **5.2.6** 中討論的類似 OSPF 的協議中。希望通過閃電網絡連接的兩個"葉子"節點將彼此連接到的中繼節點列表（即與其建立了參與支付網絡的支付通道的節點）相互通信。然後，可以檢查連接一個列表中的中繼節點與另一個列表中的中繼節點的路徑，如 **5.2.6** 中概述的那樣。

5.2.8. Conclusion. 我們概述了 TON 項目的區塊鏈和網絡技術如何足以創建 *TON Payments*，一個用於離鏈即時金錢轉移和微支付的平台。這個平台對於生活在 TON 生態系統中的服務非常有用，可以讓它們輕鬆地在需要的時間和地點收集微支付。

Conclusion

我們提出了一種可擴展的多區塊鏈架構，能夠支持大量使用者的加密貨幣和分散式應用程式，並提供用戶友好的介面。

為了實現必要的可擴展性，我們提出了 *TON* 區塊鏈，這是一個「緊密耦合」的多區塊鏈系統（參見 **2.8.14**），採用從下而上的分片方法（參見 **2.8.12** 和 **2.1.2**）。為了進一步提高潛在的性能，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**）以及用於更快速地在分片之間進行通訊的 Instant Hypercube Routing（參見 **2.4.20**）。將 TON 區塊鏈與現有和提議的區塊鏈項目進行簡要比較（參見 **2.8** 和 **2.9**），突顯了此方法對於需要處理每秒數百萬筆交易的系統的優點。

TON 網絡（見第 **3** 章），涵蓋了所提出的多區塊鏈基礎設施的網絡需求。這個網絡組件也可以與區塊鏈結合使用，創建使用僅依靠區塊鏈無法實現的廣泛應用和服務（參見 **2.9.13**）。這些服務在第 **4** 章中進行了討論，包括 *TON DNS*，用於將可讀的對象標識符轉換為它們的地址；*TON Storage*，一個用於存儲任意文件的分散式平台；*TON Proxy*，一個用於匿名網絡訪問和訪問基於 TON 的服務的服務；以及 *TON Payments*（參見第 **5** 章），這是一個跨 TON 生態系統進行即時離線貨幣轉移的平台，應用程式可以用於微支付。

TON 基礎設施允許專門的輕量級客戶端錢包和「ton-browser」桌面和智能手機應用程序（參見 **4.3.24**），為最終用戶提供類似於瀏覽器的體驗，使加密貨幣支付和與 TON 平台上的智能合約和其他服務的交互對大眾用戶可用。這樣的輕量級客戶端可以集成到 Telegram Messenger 客戶端中（參見 **4.3.19**），最終將大量基於區塊鏈的應用程序帶給數以億計的用戶。

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 The TON Coin, or the Gram

TON 區塊鏈的主要加密貨幣，特別是其主鏈和基礎工作鏈，是 *TON Coin*，也被稱為 *Gram* (GRM)。它用於支付成為驗證者所需的存款；交易費用、gas 支付（即智能合約信息處理費用）和持久存儲費用通常也會以 Gram 的形式收取。

A.1. Subdivision and terminology. 一個 *Gram* 可以被細分為十億 (10^9) 個更小的單位，稱為 *nanogram*、*ngram* 或簡稱為 *nano*。所有轉帳和賬戶餘額都以非負整數倍的 *nano* 來表示。其他的單位包括：

- 一個 *nano*、*ngram* 或 *nanogram* 是最小的單位，等於 10^{-9} Grams。
- 一個 *micro* 或 *microgram* 等於一千 (10^3) 個 *nano*。
- 一個 *milli* 等於一百萬 (10^6) 個 *nano*，或者是一 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) Grams。
- 一個 *megagram*，或 *MGram*，等於一百萬 (10^6) Grams，或者 10^{15} 個 *nano*。
- 最後，一個 *gigagram*，或 *GGram*，等於十億 (10^9) Grams，或者 10^{18} 個 *nano*。

由於最初的 Grams 供應量將被限制在五十億 ($5 \cdot 10^9$) Grams (即 5 個 *Gigagram*)，因此不需要更大的單位。

A.2. Smaller units for expressing gas prices. 如果需要更小的單位，將使用等於 2^{-16} 個 *nanogram* 的「specks」。例如，gas 的價格可以用 specks 表示。然而，實際支付的費用，計算為 gas 價格與消耗的 gas 數量之乘積，將始終向下取整到最接近的 2^{16} 個 specks 的倍數，並表示為一個整數數量的 *nano*。

A.3. Original supply, mining rewards and inflation. Grams 的總供應量最初限制在 5 個 *Gigagram*（即五十億 Grams 或 $5 \cdot 10^{18}$ 個 *nano*）。

這個供應量將會非常緩慢地增加，因為給予驗證者挖掘新主鏈和分片鏈區塊的獎勵累積。這些獎勵將相當於每年驗證者所持股份的約 20%（具體

數字可能在未來進行調整），前提是驗證者勤勉履行職責、簽署所有區塊、不離線且不簽署無效區塊。這樣，驗證者就能獲得足夠的利潤，以投資於處理越來越多用戶交易所需的更好、更快的硬件。

我們預計，在任何給定時刻，綁定在驗證者股份中的 Grams 的總供應量平均最多為總供應量的 10% [33]。這將產生每年 2% 的通脹率，因此將在 35 年內將 Grams 的總供應量翻倍（達到十個 *Gigagram*）。實際上，這種通脹代表了社區所有成員為保持系統運行而支付給驗證者的費用。

另一方面，如果發現驗證者行為不當，將收回其部分或全部股份作為懲罰，並隨後「銷毀」其中的更大部分，減少 Grams 的總供應量。這將導致通縮。罰款的一小部分可能會重新分配給驗證者或提交有關有罪驗證者行為不當的「fisherman」。

A.4. Original price of Grams. 第一個出售的 Gram 的價格將約為 $0.1（美元）。每個接下來出售的 Gram （由 TON 基金會控制的 TON 儲備）的價格都將比前一個高十億分之一。通過這種方式，第 n 個被投放流通的 Gram 將以大約以下價格售出：

$$p(n) \approx 0.1 \cdot (1 + 10^{-9})^n \quad \text{USD}, \tag{28}$$

或者是一個大致相等的數量（由於市場匯率的快速變化）的其他（加密）貨幣，例如 BTC 或 ETH。

A.4.1. Exponentially priced cryptocurrencies. 我們稱 Gram 為一種指數定價加密貨幣，這意味著要投放流通的第 n 個 Gram 的價格大約是由以下公式給出的 $p(n)$：

$$p(n) = p_0 \cdot e^{\alpha n} \tag{29}$$

其中 $p_0 = 0.1$ 美元，$\alpha = 10^{-9}$。

更準確地說，當 n 個貨幣被投放流通時，新貨幣的一小部分 dn 的價值為 $p(n), dn$ 美元（其中 n 不一定是整數）。

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

A.4.2. Total price of first n coins. The total price $T(n) = \int_0^n p(n)\, dn \approx p(0) + p(1) + \cdots + p(n-1)$ of the first n coins of an exponentially priced cryptocurrency (e.g., the Gram)

$$T(n) = p_0 \cdot \alpha^{-1}(e^{\alpha n} - 1) \quad . \tag{30}$$

[33]驗證者股份的最大總額是區塊鏈的可配置參數，因此如果需要，協議可以強制執行此限制。

A.4.3. Total price of next Δn coins. 在已有 n 枚貨幣之後投放 Δn 枚貨幣的總價格 $T(n+\Delta n)-T(n)$ 可以通過以下公式計算：

$$T(n+\Delta n)-T(n)=p_0\cdot\alpha^{-1}(e^{\alpha(n+\Delta n)}-e^{\alpha n})=p(n)\cdot\alpha^{-1}(e^{\alpha\,\Delta n}-1) \quad . \tag{31}$$

A.4.4. Buying next coins with total value T. 假設已經投放了 n 枚貨幣，並且想花費 T（美元）購買新貨幣。新獲得的貨幣量 Δn 可以通過將 $T(n+\Delta n)-T(n)=T$ 代入 (31) 計算得到：

$$\Delta n=\alpha^{-1}\log\left(1+\frac{T\cdot\alpha}{p(n)}\right) \quad . \tag{32}$$

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

A.4.5. Market price of Grams. 若自由市場價格低於 $p(n):=0.1\cdot(1+10^{-9})^n$，當有 n Gram 進入流通後，沒有人會從 TON 儲備中購買新的 Gram，他們會選擇在自由市場上購買 Gram，而不會增加 Gram 的總量。另一方面，Gram 的市場價格也不太可能高於 $p(n)$，否則從 TON 儲備中獲得新的 Gram 就是有意義的。這意味著 Gram 的市場價格不會受到突然的暴漲（和暴跌）；這很重要，因為利益激勵（驗證者押金）至少凍結了一個月，而 gas 價格也不能太快地變化。因此，系統的整體經濟穩定性需要某種機制來防止 Gram 的匯率變化太劇烈，例如上述所述的機制。

A.4.6. Buying back the Grams. 如果 Gram 的市場價格低於 $0.5\cdot p(n)$，當 Gram 的總量為 n Gram（即不在 TON 儲備控制的特殊帳戶中）時，TON 儲備保留買回一些 Gram 並減少 Gram 的總量 n 的權利。這可能需要防止 Gram 匯率的突然下跌。

A.4.7. Selling new Grams at a higher price. TON 儲備只會根據公式 (28) 出售總供應量的一半（即 $2.5\cdot10^9$ Gram）的 Grams。它保留不出售剩餘的 Grams 的權利，或者以高於 $p(n)$ 的價格出售它們，但從不以較低的價格出售（考慮到快速變化的匯率的不確定性）。這裡的理念是，一旦至少一半的 Grams 已經出售，Gram 市場的總價值將足夠高，外部勢力將更難操縱匯率，這與 Gram 的部署之初相比更為困難。

A.5. Using unallocated Grams. TON 儲備將只使用大部分“未分配”的 Grams（大約 $5\cdot10^9-n$ Gram）——即駐留於 TON 儲備和一些明確與其相關聯的帳戶中的 Grams——作為利益激勵（因為在 TON Blockchain 的首次部署階段中，TON 基金會本身很可能需要提供大部分的利益激勵），以及在主鏈上投票支持或反對有關“可配置參數”和其他協議更改的提案，

以 TON 基金會（即其創建者——開發團隊）所確定的方式進行。這也意味著在 TON Blockchain 的首次部署階段中，TON 基金會將擁有大多數的投票權，如果最終需要調整許多參數，或者需要進行硬分叉或軟分叉，這將非常有用。之後，當不到一半的 Grams 仍由 TON 基金會控制時，系統將變得更加民主。希望到那時候，系統已經變得更加成熟，不需要太頻繁地調整參數。

A.5.1. Some unallocated Grams will be given to developers. 在 TON Blockchain 的部署過程中，一定量（相對較小）的“未分配”Grams（例如，200 兆 Gram，相當於總供應量的 4%）將轉移到由 TON 基金會控制的特殊帳戶中，然後可能從該帳戶向開源 TON 軟件的開發人員支付一些“獎勵”，最短購回期為兩年。

A.5.2. The TON Foundation needs Grams for operational purposes. 2 / 2

回想一下，TON 基金會將收到從 TON 儲備售出 Grams 所得的法定貨幣和加密貨幣，並用於 TON 項目的開發和部署。例如，TON 基金會可以安裝最初的利益激勵，以及一組初始的 TON 存儲和 TON 代理節點。

儘管這對項目的快速啟動是必要的，但最終目標是使項目盡可能地去中心化。為此，TON 基金會可能需要鼓勵第三方安裝利益激勵、TON 存儲和 TON 代理節點——例如，通過為存儲 TON Blockchain 的舊區塊或代理所選服務的網絡流量支付報酬。這些支付將以 Grams 形式進行；因此，TON 基金會需要大量的 Grams 來進行運營。

A.5.3. Taking a pre-arranged amount from the Reserve. 在 Gram 的初始銷售結束後，TON 基金會將轉移一小部分 TON 儲備的 Grams 到其帳戶中——例如，所有幣的 10%（即 500 兆 Gram）——以便按照 **A.5.2** 中概述的自己的目的使用。最好同時進行這一操作和轉移用於 TON 開發人員的資金，如 **A.5.1** 中所述。

在向 TON 基金會和 TON 開發人員的轉移後，TON 儲備的 Gram 價格 $p(n)$ 將立即增加一定量，預先已知。例如，如果所有幣的 10% 用於 TON 基金會的目的，4% 用於鼓勵開發人員，那麼流通中的幣的總量 n 將立即增加 $\Delta n = 7 \cdot 10^8$，Gram 的價格將增加 $e^{\alpha,\Delta n} = e^{0.7} \approx 2$ 的倍數（即翻倍）。

剩餘的“未分配”Grams 將按照 **A.5** 中所述由 TON 儲備使用。如果 TON 基金會之後需要更多的 Grams，它將簡單地將其之前在幣的銷售中獲得的一些資金轉換為 Grams，不論是在自由市場上還是從 TON 儲備購買 Grams。為了防止過度集中，TON 基金會永遠不會試圖擁有超過所有 Grams 總量的 10%（即 500 兆 Gram）的 Grams。

A.6. Bulk sales of Grams. 當許多人同時想從 TON 儲備購買大量 Grams 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特定訂單的時間和處理序列。

相反，購買 Grams 的訂單可以在某個預定義的時間段（例如一天或一個月）內收集，然後一次全部處理。如果有 k 筆訂單，第 i 筆訂單價值為 T_i 美元，那麼總金額 $T = T_1 + T_2 + \cdots + T_k$ 將用於根據 (32) 購買 Δn 個新幣，第 i 筆訂單的發送者被分配 $\Delta n \cdot T_i/T$ 的這些幣。這樣，所有買家都以同樣的平均價格 $T/\Delta n$ 美元每 Gram 獲得其 Grams。

之後，開始新一輪的收集購買新 Grams 的訂單。

當購買 Grams 的訂單總價值足夠低時，這種“大宗銷售”系統可以用根據公式 (32) 從 TON 儲備立即出售 Grams 的系統替換。

“大宗銷售”機制可能在收集 TON 項目的投資初始階段被廣泛使用。