# 易幣：
# 一種能為去中心化存儲網路創造數據冗餘與流通共享的公共區塊鏈平台

November 6, 2022
〈 draft v0.1 〉
Slinker HJ. Jiang
slinker@yicrypto.org

**摘要**
「易幣」是一個高度去中心化的公共區塊鏈平台，具備安全性與動態擴展性的共識機制能為點對點分散式存儲網路提供一個低通貨膨脹的激勵層，能有效創造數據冗餘並促進資訊的共享與流通。受到《易經》中「緯卦」與「爻變」操作的啟發，參與區塊鏈網路的輕節點無需維持全系統的最新狀態也能夠提出工作量證明來獲得記帳權；更輕量的存儲節點甚至只要分享閒置存儲空間及應答頻寬即有機會獲得獎勵；資源豐富且算力集中的全節點則可以透過驗證交易請求與執行自動合約獲得合理的手續費。「易幣」有望解決去中心化、安全性與可擴展性的不可能三角難題，初期能提供一個更便宜且穩定的雲端存儲替代方案、進而可以串聯龐大的 IoT 聯網裝置，最終成為 Web3.0 世代重要的基礎建設之一。

## 一. 前言 Introduction

### 1.1 Web 3.0

以 Web 3.0 為名的新一代全球資訊網涉及了許多可能的發展方向與特徵，其中對「去中心化」這一概念的實踐，正逐步體現在與區塊鏈有關的應用發展上。如果說全球資訊網就是結合數據（Data）、計算（Computing）及互動（Interaction）的產物，那麼星際文件系統 IPFS（InterPlanetary File System）/ Filecoin 就是針對「數據」的分散式存儲、以太坊的智能合約就是針對「計算」的去中心化執行、比特幣的公開式帳本就是針對交易的「互動」歷程，提供一個實驗性質的解決方案。從這個角度講，區塊鏈技術或許比諸如「語義網」（Semantic Web）或 AI 人工智慧網路，更貼近現今全球資訊網的發展軌跡。

回顧 Web 1.0 「唯讀」到 Web 2.0「 互動讀寫」的發展歷史，技術運用與概念普及是一個漸進式的過程，需要長時間、多方面的應用者參與；而 Web 3.0 對數據「擁有」的重視，正是一個可以在兼容現有網路環境下逐步發展的方向。如今我們已經可以實現某些 Web 3.0 概念的應用，例如為永久保存意見，不選擇在中心化論壇上進行論述，而是將短文記錄在比特幣對應某議題的匯款帳戶中；或是 2017 年加泰隆尼亞為避免西班牙中央政府干預，而在 IPFS 上架設獨立公投投票地點的查詢網頁。

Web 3.0 與其說是重視對數據的「擁有」（ownship），不如說是重視其「開放」、「意願」與「非同質」化（open, willing, nonfungible - ship），希望避免中心化巨頭對數據的片面性掌控並保有對數據的自主權。這就催生了巨大的去中心化數據存儲需求，而儲存及讀取數據的網路可以說是全球資訊網的基石，因此去中心化的浪潮勢必會對數據存儲的基礎建設造成典範轉移，並且帶動更巨大的商業和應用前景。

### 1.2 IPFS 與 BitTorrent

去中心化存儲主要透過點對點（P2P）網路上的分散式系統（distributed system）進行數據保存。目前較為重要的去中心化存儲相關的傳輸協議是 IPFS 與 BitTorrent，前者提供一個跨節點的文件系統、期待可以取代傳統的 HTTP 網路協議；後者以同時上傳、下載檔案資訊分塊的方式加速大型檔案的分享。

IPFS 與 BitTorrent 的共同特色之一，是參與存儲的節點可自行決定要保存感興趣的「資訊片段」多久，這相當於電腦中快取記憶體（cache）的概念，因此也有永久丟失數據的風險。為此， IPFS 導入了「固定服務」（pinning service）提供付費存儲的商業方案以及加密貨幣 Filecoin 的激勵機制，希望超越 BitTorrent 僅靠自願性「共享資源」的概念所無法達成的 " 永久存儲 " 目標。另一個共同特色是被儲存的「資訊片段」與存儲節點間沒有對應關係，所以 BitTorrent 需要追蹤伺服器（Tracker）、 IPFS 需要檢索節點，來記錄與維護保存「資訊片段」的存儲節點；只是追蹤伺服器破壞了 BitTorrent 去中心化的架構，成為容易被攻擊的脆弱環節；而 IPFS / Filecoin 則因為檢索節點與存儲節點分離，且缺乏對檢索節點的激勵機制，從而降低了數據下載的效率。

本文所提出的「易幣」（Yi Crypto, YiC）是在 P2P 對等網路中使用區塊鏈技術的一種原生加密貨幣（Cryptocurrency）。「易幣」是基於「未花費交易輸出」（Unspent Transaction Output, UTXO）模型，對數據傳輸、儲存、備援及貨幣交易、合約執行等貢獻行為的一種獎勵機制。「易幣」改善了上述二種存儲協議的缺點，還具有參與門檻低、激勵對象多、單位交易數量高等其它加密貨幣少見的優點，但在討論這些特性之前，有必要先對「易幣」與《易經》（Yi Jing）之間的關係進行說明。

### 1.3 易經、基因與計算機

#### 1.3.1 緯卦 Riffle Hexagram

《易經》是中華民族古老的哲學智慧，闡述了由三爻（Trigram）組成的八個「經卦」（又稱伏羲卦）與二個經卦重疊而成的「六爻卦」（Hexagram，即六十四個「成卦」） 的變化與意涵。如果比較基因 DNA 或 RNA 中，以三個鹼基為一組、所組成的六十四個遺傳密碼子（codon）── 其中，每個鹼基恰好對應著「成卦」的天（上、五爻）位、地（四、三爻）位、人（二、初爻）位；而二爻的陰陽變化正好能對應四種鹼基化學分子（A、T、C、G）── 可以發現與《易經》之間存在著驚人的巧合。如果將 DNA 雙鏈拆成左右兩鏈，左鏈由「成卦」的上、四、二的陰位爻、右鏈由「成卦」的五、三、初的陽位爻，重組可以得到二個「經卦」；反之，將二個「經卦」的爻位交錯也能組成一個「六爻卦」。這種依 " 奇偶爻位 " 取得卦象的方法，本文稱之為「緯卦」（ Riffle Hexagram），如：

天火同人（䷌）之「緯卦」為天澤履（䷉）。

水火既濟（䷾）之「緯卦」為澤雷隨（䷐）。

在「易幣」中這種對「資訊片段」依奇偶位置交錯拆解及組合的運用，被用來避免「索引鍵配對攻擊」（Key-Match Attack），詳細說明請參考〈 2.3.2 小節〉所述。

#### 1.3.2 爻變 XOR Hexagram

《易經》加強了十七世紀的微積分發明者──萊布尼茲（Gottfried W. Leibniz）對二進位制運算可用性的信念，雖然這在當時是被認為是『看不出二進位有何用處』的學問，但其應用之妙卻在現代電腦的邏輯與位元運算中顯露無遺。其中，老陰變陽、老陽變陰的「動爻」，相當於「或閘」（OR gate）的應用；如果以陰陽來表示每個爻位是否變化，這就相當於用一個卦象來表示「互斥或閘」（XOR gate），或者稱之為「爻變」（XOR Hexagram）。任意一個「成卦」皆可以找到一個特定的六爻進行「爻變」運算，並轉變成任意的六十四卦之一（或稱為「變卦」）。也就是說，可以用二個卦象的 " XOR 操作 " 來代表另一個卦象：

巽為風（䷸）之「離變」（䷝）後是兌為澤（䷹）。

水山蹇（䷦）之「升變」（䷭）後是坎為水（䷜）。

那麼，《易經》六十四卦最少可以用幾個「爻變」卦象表示呢？

本文提出一個猜想：

> ***隨著爻數增加，組成卦象的總數量會以　2　的倍數增長；而所需的最少不重覆「爻變」數量會等於「費波那契數列」（Fibonacci　sequence）或費氏數列　+1（當不等於 2 的次方階時）。***

舉例來說：用乾、震、坎、艮四個陽卦可以「爻變」成任意八卦之一，其中四種卦象可以重新用 2 bits 長的索引替代，若不計索引與值的存儲空間，則表達所需的資訊量為 4 bits；六十四卦則最少可用十四種「爻變」卦象組成，由 4bits 長的索引替代時，所需資訊量為 8 bits。

雖然上述二例，採用類似「字典壓縮演算法」（Dictionary based Data Compression Algorithms）的方式來記錄數據，反而會增加　33%　以上的容量；但是應用於「分散式雜湊表」（distributed hash table, DHT）作為存儲的「鍵—值」索引時，整個系統的壓縮率就成了 2 的冪次方與費氏數列的比值。例如，用「爻變」保存一個長度為 32 bits 所存儲的所有數據可能，最少只需要 3,524,578 個「鍵—值對」。這與直接存儲所有集合來比較，能可觀地減少約 99.18% 的空間；更遑論在以 256 KiB 長度進行存儲的 IPFS 或 BitTorrent 網路上，所能達到的 " 最高 " 資料壓縮比了。

### 1.4 易經與易幣

研究《易經》時有「變易」、「簡易」、「不易」的三個原則，這同樣也可以作為「易幣」的特色描述：其中，區塊鏈技術保證了數據的「不易性」（Immutability）、　「爻變」的「變易性」（Variability）提供了冗餘的數據備援與資料壓縮、「分片聚合」則透過「趨同選舉」的機制大幅提高動態擴展的「簡易性」（Simplicity）── 或許有人認為加密貨幣（Crypto）從來都不簡易吧？那麼就用合約與貨幣的「交易性」（Tradability）來充當「易幣」的第三個 " 易 " 特色吧。

## 二. 共識機制 Consensus Mechanism

### 2.1 尋址類型 Addressing

#### 2.1.1 資訊片段地址 Piece Address

與 BitTorrent 的位元「洪流」（torrent）相同，「資訊片段」（pieces）是一個長度固定為 256 KiB 、以位元記錄資訊的一種數據型式，如果記錄資訊少於固定長度時，未使用空間需先填塞（padding）隨機數值。數據內容經過　SHA3-256　後，以「混合大小寫校驗摘要地址編碼」（Mixed-case checksum address encoding，參考 EIP55）方式處理過之哈希值為其「索引鍵」（Hash Key）、分散儲存於對等式（peer-to-peer，P2P）的存儲網路中。

##### 2.1.1.1 塊鏈片段 Chuck-chain Pieces

「易幣」使用一種特定格式，作為打包區塊鏈（Block）、收據（Receipt）、種子文件（Seed）及自動合約（Smart Contract）等內容之用：

| 欄位 | 形態 | 描述 |
|---|---|---|
| Length | UInt32 | 剩餘長度。 |
| Nonce | Int32 | 負值為隨機數，正值保留給擴充長度用。 |
| Next | Char[40] | 以16進制表示下個索引鍵，當長度少於262137時，無此欄。 |
| Chunk | Bytes[...] | 長度介於262096~262136 的位元塊，不足時需填塞隨機數值。 |

其中「次片段」Next 欄位記錄著下一個「塊鏈片段」的索引鍵或「地址」（address），能用於從存儲網路中讀取後續分段的位元塊（chuck）資訊並依序組成一個完整的文件。使用這種格式打包的數據會形成「位元塊鏈」（chuck-chain），其中第一個片段稱為「首片段」（first piece），最後一個稱為「末片段」。「末片段」沒有 Next 欄位，所以可用空間會增加 40 Bytes。「位元塊鏈」雖然可以組成接近 4 GiB 的數據，但大型文件建議採用「種子」形式發布，可以加速數據的下載時間。

### 2.1.2 帳戶地址 Account Address

作為交易輸出入目標的「帳戶地址」是將 BLS（Bohen-Lynn-Shacham）公鑰或聚合公鑰加上固定前綴詞、經由 SHA3-256 及 EIP55 處理後的哈希值，可以理解成與公鑰長度接近（有前綴詞，但不用填塞至 256 KiB）的一種「資訊片段」。不過，在「易幣」的工作協議中，儲存「帳戶地址」的內容（也就是公鑰）並不像儲存「資訊片段」一樣有存儲獎勵，意即公鑰本身並不參與「工作量證明」的過程（請參考〈2.3.3 難度可調式工作量證明〉一節），其它節點也不會提供「帳戶地址」與其對應公鑰的查詢服務。

而且在 UTXO 模型中，一個「交易輸入」（TX Input）如果未曾於鏈上以公鑰及簽章進行提領，其它人是無法從其來源的「交易輸出」（TX Output）地址來判斷，這是一個用於交易的「帳戶地址」或是一個未曾發布過內容的「資訊片段地址」，這可以避免 " 商標蟑螂 " （Trademark Troll）攻擊。「商標蟑螂攻擊」算是一種「礦工可提取價值」（Miner Extractable Value, MEV）攻擊，具體說明請參考〈 2.2.2.2 小節〉所述。

### 2.1.3 節點地址 Node Address

在「易幣」的 P2P 存儲網路中，為降低 S/Kademlia 所述攻擊的影響，每一個網路節點採用一個由公鑰產出的「帳戶地址」作為節點身分（Node ID），並使用簽章後的訊息與其它節點進行通訊應答。每個節點至少會以一種身分執行各自的工作協議，例如：以「指向性流言協議」（Directional Gossip Protocol）傳播「交易請求」（Transaction Request）、「趨同選舉」（Convergent Election）的投票內容、存儲或傳送「邏輯距離」（ logic distance）最接近與具有高「賦價」（valuing）的「資訊片段」等工作。

主要的節點類型及其執行工作項目如下：

| | | |
|---|---|---|
| 路由節點<br>(Routing Node) | 啟動節點<br>(Boot Node) | 提供初始連線方式、加速區塊鏈發布效率。 |
| 輕節點<br>(Light Node) | 錢包節點<br>(Wallet Node) | 保存個別帳戶相關訊息、提出交易請求 （Transaction Request）。 |
| | 存儲節點<br>(Storage Node) | 提出存儲證明、傳播資料片段。 |
| | 記帳節點<br>(Miner Node) | 提出工作量證明、聚合分片並打包原始交易（Transaction）及衍生交易（Derivative）。 |
| 全節點<br>(Full Node) | 驗證節點<br>(Validator Node) | 發起交易驗證投票（Transaction Vote）。 |
| | 執行節點<br>(Executor Node) | 發起合約執行投票（Contract Vote）。 |

## 2.2 工作協議 Protocols

### 2.2.1 指向性流言協議 Directional Gossip Protocol

「指向性流言協議」允許在網路壅塞時，放棄對遠離目標地址的節點進行的訊息傳播，或者說，降低該流言轉發的「持續回合」（rounds）或「持續時間」（duration）。流言訊息的重要性依序為「Generate 訊息」、「TX 訊息」、「Store 訊息」、「Vote 訊息」，所以「交易請求」（TX 訊息）會盡可能地在「交易驗證節點」間傳播，而「投票傳播」（Vote 訊息）則可能只有接近「投票議題」（收據地址）的「合約執行節點」才能接收到完整的投票結果。除此之外，「Find_Value 訊息」、「Find_Proof 訊息」雖然不會自動在網路間進行流言傳播，但其檢索應用也是一種由操作者主動朝向 " 目標方向 " 發起訊息傳播的過程。這些「指向性流言協議」所使用的訊息格式，將在以下小節進行說明：

#### 2.2.1.1 Find_Value 訊息

在 Kademlia 網路協議中，K 桶（K-buckets）內儲存的網路節點是用來查詢「鍵—值」（Key-Value）的路由表。在「易幣」中「檢索節點」直接記錄「邏輯距離」（logic distance）相近的「資訊片段」內容，而非記錄保存該「資訊片段」的「存儲節點」訊息 —— 也就是說，「易幣」的「檢索節點」同時也是「存儲節點」。在對「Find_Value 訊息」的應答中，如果「檢索節點」儲存著對應 " 鍵 "（即資訊片段地址）的內容時會一併返回該值；反之，則與 Kademlia 的「Find_Node 訊息」一樣，只會回應更接近地址的節點列表 。

此外，「Find_Value 」 也採用了 BitTorrent 的「子片段」（sub-pieces）概念，引入「速率」及「索引」兩個參數：「速率」最大值為 256 KBps，預設為 2 KBps （值為 1，即 16 Kbps），此時「索引」範圍為 0 - 127 之間。若節點存有「資訊片段」的完整內容則會直接回傳對應的「子片段」數據；節點不論是否回傳「子片段」數據，皆需回應最接近目標地址的「存儲節點」列表。

#### 2.2.1.2 Find_Proof 訊息

「易幣」新增的「 Find_Proof 訊息」與上一小節的「Find_Value 訊息」 相似，但回傳的是存儲內容的證明，也就是指定「索引」之「子片段」的「校驗摘要」，例如：當參數「速率」為 255 而「索引」為 0 時，回應的「存儲證明」應該等同於該「資訊片段」的地址。

#### 2.2.1.3 Generate、Store、TX 、Vote 訊息

另外，「易幣」擴充「 Store 訊息」的存儲請求，優先對最接近目標地址的路由節點進行傳播，意即存儲請求的傳播具有「指向性」（directivity）。這種「指向性流言協議」還適用於「區塊產出」（Generate 訊息）、「交易請求」（TX 訊息）及「趨同選舉」時使用的「投票傳播」（Vote 訊息），其指向之目標地址依其情境有所不同：在「Generate 訊息」及「Store 訊息」時為「區塊鏈片段」等「資訊片段地址」、在「TX 訊息」時為「交易編號」（Transaction ID, TXID）、在「Vote 訊息」時則為「收據」（Receipt）的「首片段」地址。

### 2.2.2 交易請求協議 Transaction Request Protocol

「易幣」使用與比特幣相同的「未花費交易輸出」（UTXO）記帳模型，未分配額度即為交易手續費；但不採用其腳本（Bitcoin Script），而是直接定義以下數種交易請求的指令行為：

#### 2.2.2.1 匯款 / 賦價 / 版權宣告 / 委托

指令：OP_SEND {address}

*將指定款項「匯款」到「帳戶」地址或「委托」執行「自動合約」；對「資訊片段」地址「賦價」或「宣告版權」。*

對於一個「資訊片段」地址的首次「匯款」或「賦價」（valuing），可以被視為對該「資訊片段」的一種「創作共用」（Creative Commons）的「版權宣告」（copyright）── 意即首次「匯款」或「賦價」的來源帳戶被視為是這個 256 KiB 位元流的 " 創作者 " 。特別需要說明的是，在「易幣」的「爻變」操作下，每個「資訊片段」都只是可以互為冗餘的位元組合罷了，所以「版權宣告」的「標的」（subject）是位元流組合，而不是其記錄的數位內容。

「易幣」地址需要等到後續對該地址內的款項提供公鑰及簽章進行提取時、或是公布「資訊片段」地址的內容後，才能決定這是一個「帳戶地址」或是一個「資訊片段地址」，這不但不會影響「OP_SEND　指令」的操作，還可以避免下一小節所述的「商標蟑螂攻擊」。明確地說，對一個「資訊片段」的「匯款」（OP_SEND）等同於對其「賦價」（OP_VALUE），初次「匯款」或「賦價」的操作也具有「版權宣告」的意涵存在。

如果「OP_SEND 」指令的目標是一個已經啟動「轉址」（redirection）或「權利轉讓」（assigned）的地址，「交易驗證」時會以「衍生交易」（Derivative）方式提取款項到指定的新地址 （相關說明請參考〈 2.2.2.3 小節〉及〈3.3.1 交換律 小節〉所述）；如果目標是一個「充值」後的「自動合約地址」，則表示指定的「委托」款項將用於合約的執行。

#### 2.2.2.2 賦價 / 版權宣告 / 銷毀

指令：OP_VALUE {address}

*對「資訊片段」地址「賦價」或「宣告版權」；向「帳戶」地址「銷毀」款項。*

「匯款」（OP_SEND　指令）到「資訊片段地址」也是一種「賦價」的操作，但當「資訊片段」── 特別指，已被「權利轉讓」（assigned）的「種子」文件或被「見證」（witness）過的「自動合約」的「首片段」──的「匯款」對象已被設定指向新的地址時，需要透過此「OP_VALUE 指令」明確指定款項是用於「賦價」的用途。

「賦價」目標地址是一個「資訊片段」，也就是說並不存在提取時必要的私鑰，那麼匯入的款項跟「銷毀」（Burning）一樣，將永遠無法再被任何人提取使用。「賦價」的主要目的是作為一種存儲的 " 銷毀性 " 成本及 " 重要性 " 的參考指標，「存儲節點」可藉此判斷在有限空間的情況下應如何取捨；同時也與「工作量證明」的激勵機制有關。當參與 " 記帳獎勵 " 的「資訊片段」含有多個「賦價」的 UTXO（可能是 OP_SEND 或 OP_VALUE 指令）時，會被整合輸出（TX Output）為單一「賦價」的款項（ 餘額再重新「 OP_VALUE」至本片段地址），並記錄到「鑄造交易」（ Mint Transaction）中（ 正確的說，會與其它參與證明的「資訊片段」均分「賦價」的款項）。

一個「資訊片段」內容的公布（使用 2.2.1.3 所述的 「Store 訊息」）應該在「版權宣告」（ 即初次進行 OP_SEND 或 OP_VALUE 指令）後；如此，含有惡意的 " 商標蟑螂 " 將無法識別要攻擊的目標地址是否為有價值的「資訊片段」（例如一個「種子」文件的「首片段」）或是一個帳戶地址，而不敢冒然發起 MEV 攻擊（如果對「帳戶地址」搶先進行 OP_SEND，等同於匯款、搶先 OP_VALUE 則等於「銷毀」了款項）。若不幸攻擊成功，被攻擊方仍可改變「資訊片段」的 Nonce 值後，再對新的片段地址重新進行「版權宣告」。

#### 2.2.2.3 轉址 / 權利轉讓

指令：OP_ASSIGN {account | seed address} {account | smart contract address} {price}

*對「帳戶」地址進行「轉址」設定；對「種子」文件進行「權利轉讓」設定。*

*交易金額限定為 0 個「易幣」（Yi coin, YIC）。*

使用公鑰及簽章提取一個「帳戶」下「OP_SEND」的 UTXO 款項，再對該「帳戶」執行「OP_ASSIGN」指令，可以設定未來對這個「帳戶」操作的所有「OP_SEND 指令」會「轉址」到參數所指定的「帳戶」或「合約」地址上。如果「授權金」參數設定是 0 個 YIC，則「轉址」在 " 出塊 " 後立即生效；如果「授權金」大於 0 個 YIC 時，指定「帳戶」或「合約」後續對發起「轉址」的 " 原帳戶 " 要能累積大於等於「授權金」的「OP_SEND」款項，「轉址」才會生效。「轉址」生效後，在 " 原帳戶 " 地址下「OP_SEND」的 UTXO 仍可被提取使用，但後續的「匯款」交易將以「衍生交易」方式「轉址」到最終的「帳戶」或「合約」地址上。

對於「版權宣告」後的「種子」文件，則要使用 " 創作者 " 的公鑰及簽章提取一個 UTXO 款項來作為「權利轉讓」（assignment）的「交易輸入」（TX Input），同時「種子」文件如果由多個「塊鏈片段」組成，每個「次片段」皆需「版權宣告」，否則交易請求將被拒絕。無論「帳戶地址」或「種子」文件，在設定生效前都可以發起多次「OP_ASSIGN」指令：如果對同一個轉址或權利轉讓的新地址進行多個「授權金」設定，只要滿足最低值就能生效；如果有多個「OP_SEND」在同一「鏈高」下滿足了設定，只有「序列順序」（sequence order，請參考〈3.5.2 小節〉的說明）最早的「OP_ASSIGN」設定會生效。

#### 2.2.2.4 充值 / 見證 / 承保

指令：OP_CHARGE {smart contract addres}

*對「自動合約」的 " 手續費池 " 進行「充值」或表達審閱後「見證」合約的意願。*

「自動合約」在編譯前也是由一個或多個「塊鏈片段」所組成的文件，同樣也建議透過「OP_SEND 指令」來進行「版權宣告」與「賦價」，並在提出「存儲請求」（Store 訊息）並確定合約內容已被儲存後，才開始進行「充值」的操作。如同「OP_ASSIGN」的指令一樣，驗證「OP_CHARGE」指令時，會檢查組成片段是否已先進行「版權宣告」，否則交易請求將被拒絕。「自動合約」在「充值」後，「OP_SEND」指令就不再具有「賦價」的功能，而是「委托」合約的「執行節點」可以依合約內容使用該款項。因此，如果想要長期保存合約文件的話，應該明確地使用 「OP_VALUE」 指令以提高「賦價」。

由具公信力第三方進行的「OP_CHARGE」指令，可以視其同意成為該合約的 " 見證人 "（witness）或 " 承保人 "（insurer）。透過專業見證或保險機構對「自動合約」的審閱、認證或承保，可以避免合約詐騙或能獲得因漏洞而導致的損失賠償。

「OP_CHARGE」指令在 " 出塊 " 後會立即生效，並開始啟動合約的編譯或執行。編譯或執行合約的費用（相當於以太坊的 gas fee）是由「充值」款項組成的 " 手續費池 " 所支付，編譯會依合約的字符總長度、執行時會依指令碼的呼叫總次數，從「充值」中扣除相關費用。跟「OP_VALUE」指令相似，合約執行後含有多個「充值」的 UTXO 會被整合成交易輸出（TX Output）至單一的款項（餘額再重新「OP_CHARGE」至本合約地址），並記錄到「收據」內的「衍生交易」中。

當「充值」因 " 返回 "（合約如果有定義，當執行結束時可以「OP_SEND」餘額至指定地址）或扣除為零時，合約會停止執行。合約如果在中途停止，其原子性（Atomicity）保證只有執行費用會被提取，其它狀態應維持不變；任何希望合約能順利執行的 " 諦約者 " 應該在下一區塊產生前，先完成足夠金額的「充值」。

### 2.3 工作量證明 Proof of Work, PoW

「易幣」使用的共識機制是一種基於能提升 P2P 存儲的「冗餘度」（redundancy）與 「流通度」（circulation）的工作量證明（Proof of Work, PoW）。與常見使用工作量證明的加密

貨幣不同，「輕節點」（Light Node）即可進行 " 挖礦 " （即提出工作量證明以獲得記帳權），而「交易驗證」及「合約執行」則需要由「全節點」（Full Node）來執行。

「輕節點」通常只保留與自己相關有關的交易內容，而且不一定經常保持在線；當「輕節點」加入P2P「流言協議網路」（Gossip Protocol Network）後，可以選擇成為「錢包節點」（Wallet Node）、「存儲節點」（Storage Node）或「記帳節點」（Mining Node）。其中，「存儲節點」至少會儲存與「節點地址」距離相近或具有高「賦價」的「資訊片段」、依請求回應「存儲證明」（Find_Proof 訊息）或提供「下載」服務（Find_Value 訊息）；而「記帳節點」則必須更進一步驗證最新「區塊鏈」內容、從投票網路中聚合「交易驗證投票」（Transaction Vote）與「合約執行投票」（Contract Vote）等資訊 、打包、檢索合格的「存儲節點」並開始準備提出工作量證明。

### 2.3.1 冗餘度與流通度 Redundancy & Circulation

#### 2.3.1.1 冗餘證明 Proof of Redundancy, PoR

存儲系統的「冗餘度」（redundancy）不僅能依靠在多個節點上儲存片段複本來提升（這適用於 BitTorrent 協議），還能透過「爻變」（XOR Hexagram）來創造 " 冗餘 " ，這也是「易幣」的命名由來。「爻變」簡單地來說，就是一個任意的「資訊片段」可以由另外二個「資訊片段」用「互斥或」（XOR）產生；如果這二個「資訊片段」早已存在系統中（可透過「Find_Proof 訊息」確認）則可以提出一個「冗餘證明」（Proof of Redundancy, PoR），表示這三個「資訊片段」互為 " 冗餘片段 " 。

「爻變」也能用來加密檔案或壓縮一個巨型的 DHT 存儲系統：用一個高「賦價」的「資訊片段」作為 " 密鑰 " ，將組成檔案的「資訊片段」經 " XOR 操作 " 後得到的新片段進行「賦價」、記錄「種子」文件並發布「資訊片段」到「易幣」的存儲網路中。更極端的使用情境是將欲儲存的「資訊片段」先以遍歷所有「賦價」片段的方式進行 " XOR 操作 " 直到有另一個「賦價」的片段被找到，若找不到才對其進行「賦價」。這 " 有機會 " （因為就算有足夠時間能對所有「賦價」片段進行比對，也未必能找到）能降低整體文件的存儲成本，或是壓縮整個系統的存儲資訊量。

一個分散式的存儲系統中如果存在大量的「冗餘證明」，則「資訊片段」將不再僅為某文件的 " 分割片段 " ，而是一個能重覆使用的 " 字詞 " （參考 1.3.2 爻變 小節所述的「字典壓縮演算法」與 " 壓縮率猜想 " ），這能同時增加文件下載的隱密性（confidentiality）及存儲系統的穩健性（robustness）。

#### 2.3.1.2 流通證明 Proof of Circulation, PoC

除了上述以高「賦價」片段的 " XOR 操作 " 來提高存儲冗餘外，流通率高的「資訊片段」（例如熱門影音片段），在毋需要進行「賦價」的情況下也能以多個複本、快速地在節點中傳播，亦適合用於當作「冗餘證明」中的 " 冗餘片段 " 之一，但要如何用一個簡易的方式證明一個「資訊片段」具有高流通率並不容易。

「易幣」為此提出一種工作協議：要求「存儲節點」進行臨時性的存儲，或者說是成為臨時性的「串流伺服器」，這可以透過在「Find_Value 訊息」中，額外加入檢索片段所屬的「種子地址欄位」來達成。有了「種子」地址，「存儲節點」能夠預先下載後續的「資訊片段」以加速影音檔案的串流速度；同時這也讓「記帳節點」能夠統計出「種子」文件及其所含「資訊片段」被請求的頻率。「記帳節點」會先驗證「種子」文件的「權利轉讓」是否已經生效（即有設定地址可作為後續獎勵的對象），再將 " 高下載請求率 " 的二個相鄰「資訊片段」當成一個 512 KiB 的「隨選串流」（Stream On Demand, SoD）並試圖找到二個已知「賦價」片段能與其一互為 " 冗餘片段 " ；是則即可提出一個「流通證明」，證明「種子」文件的 " 創作者 " 至少貢獻了一個 " 高流通 " 的「資訊片段」足夠作為「賦價」片段的 " 冗餘備份 " 。

透過對任意二個已知的「賦價片段」 XOR 後的 " 索引鍵 " 進行「種子」內文的搜尋比對，是可以更高效地找出對應的「資訊片段」地址，而不需要真正擁有流通的「資訊片段」內容，這個作法稱為「索引鍵配對攻擊」（Key-Match Attack）。「流通證明」中需要提供三個「資訊片段」地址，其中一個「賦價」片段作為 XOR 參數，另外二個「資訊片段」（至少一個為已「賦價」片段）則在 " XOR 操作 " 後會對應「隨選串流」的內容，而前述攻擊僅能反向搜尋到「隨選串流」中「資訊片段」的地址而非其內容，在缺乏內容的情況下攻擊者難以透過「雜湊碰撞」來取得證明所需的第三個「資訊片段」地址。

### 2.3.2 頭獎證明 Proof of Jackpot, PoJ

上節提到「記帳節點」是利用二個相鄰「資訊片段」的請求率來判斷該存儲文件是否值得獲得 " 高流通 " 的獎勵，也就是說「記帳節點」應該實際暫存完整的「隨選串流」內容，而非利用「索引鍵配對攻擊」來提出證明。避免攻擊的方法是不要直接使用「種子」內記錄的「隨選串流」，而是採用一個計算或操作後的數據作為證明的對象。例如將「隨選串流」依 " 奇偶位置 " 抽出成為的二個新片段作為 " 冗餘片段 " 之一，這樣在沒有得到「隨選串流」的完整數據前，是無法僅靠組成「隨選串流」的二個「資訊片段」地址就能嘗試提出「流通證明」的。

這種 " 奇偶交錯 " 的方式啟發自一種稱為「緯卦」（Riffle Hexagram）的卦象：其演卦的方法是將「上卦」的爻置於陰位、「下卦」的爻置於陽位，即將兩個「經卦」（八卦）的天、地、人三爻，共組成「緯卦」（六十四卦）的天、地、人三爻（具體作法請見〈1.3.1 緯卦〉小節）。這種方法能夠改善上二節「冗餘證明」與「流通證明」的缺點，本文稱為「冗餘流通證明」（Proof of Circulation and Redundancy, PoCaR）或直接稱為「頭獎證明」（Proof of Jackpot, PoJ）。

為了加速其它節點對「頭獎證明」的驗證速度，「種子」文件中除了記錄「資訊片段」地址的 " Pieces 列表 " 欄位外，還引進了「隨選串流」所計算出「校驗摘要」的 " Checksums 列表 " 欄位。第一個「校驗摘要」對應著第一、二個「資訊片段」地址所組成的「隨選串流」；第二個「校驗摘要」則對應著第二、三個「資訊片段」地址，依此類推。當驗證「頭獎證明」時，取證明中的三個「資訊片段」用以「爻變」並以 " 奇偶爻位 " 重組成「隨選串流」後，即能與「種子」文件中記錄的「校驗摘要」進行驗證了。

一個推薦的「頭獎證明」過程，具體說明如下：

1. 選擇一個下載請求率高的「隨選串流」，將 " 奇位 " 數據抽出成為第一個臨時片段、" 偶位 " 數據抽出成為第二個臨時片段；
2. 將二個臨時片段與選定的「賦價」片段進行 XOR 後，得到二個新片段；
3. 比對二個新片段中應該至少有一個為「區塊鏈」或「收據片段」，且未曾與「隨選串流」一同提出過「頭獎證明」，否則選擇新的「賦價」片段重複第 2 步驟；
4. 比對前一步驟符合的「區塊鏈」或「收據片段」的「鏈高」，應該高於第 2 步驟中「賦價」片段上鏈時的「鏈高」，否則選擇新的「賦價」片段重複第 2 步驟；
5. 將前一步驟符合的「區塊鏈」或「收據片段」，對第 1 步驟的另一臨時片段進行 XOR 後，得到 " 新生 " 片段（得到一個 " 已賦價 " 片段的機率極低）；
6. 提出「頭獎證明」：找出了一個「區塊鏈」或「收據片段」可做為 " XOR 參數 " ，能將至少一個 " 已賦價 " 與 " 新生 " 片段「爻變」後，依 " 奇偶爻位 " 重組成「種子」文件中「校驗摘要」所對應的「隨選串流」。

依上述步驟，最多只需額外產生一個 " 新生 " 片段，便得以提出一個能提高 " 存儲穩健性 " 的「冗餘流通證明」。只是，為了避免利用產生虛假「資訊片段」的方式詐領獎勵，「頭獎證明」限制了作為 " XOR 參數 " 的片段種類與 " 賦價 " 片段上鏈時序的條件，這使得滿足條件

更為嚴苛。事實上，「頭獎證明」的出現頻率很低，尤其是在早期「記帳節點」與「賦價」片段數量皆很少的情況下，自然是無法滿足於固定時間產出區塊鏈的需求（意即無法進行難度調整）；同時該證明亦無法被區塊鏈本身保護（因為證明的內容與「記帳節點」本身的資訊無關）。因此，「易幣」不將其作為主要的工作量證明，而把它當作是一種如同中了 " 頭獎 " 的偶發事件，並以「叔塊」(Ommer Block）來保障位於 " 非最長鏈 " 的「頭獎證明」。

### 2.3.3 難度可調式工作量證明 Difficulty Adjustable Proof of Work, DAPoW

為了避免攻擊者用產生 " 虛假匹配 " 的「資訊片段」與「種子」的方式攻擊「頭獎證明」，因此規定了 " XOR 參數 " 片段的種類並加上「鏈高」限制，這使得原本難度就高的「頭獎證明」難上加難；而且即使提出「頭獎證明」，仍然有必要提出由區塊鏈保護的「難度可調式工作量證明」（ Difficulty Adjustable Proof of Work, DAPoW，或直接稱為「工作量證明」）以確保「記帳節點」的工作獎勵。

「工作量證明」與「頭獎證明」的流程幾乎相同，只是取消了第 3 步驟與第 4 步驟對「區塊鏈」、「收據片段」與「鏈高」的限制，僅要求三個 " 冗餘片段 " 中有一個 " 已賦價 " 片段即可。取而代之，要求作為 " XOR 參數 " 片段與打包的新區塊鏈「首片段」地址的距離小於等於指定的 " 區塊鏈距離難度 "（ difficulty of block）。這樣設計的目的除了導入一個 " 可控難度 "（ Adjustable Difficulty）外，還希望 " XOR 參數 " 片段（建議放置 " 新生 " 片段）與新區塊鏈「首片段」的 " 存儲地址 " 接近；如此一來，其它驗證節點可從網路的同一 " 方向 " 取得新「區塊鏈」片段及 " XOR 參數 " 片段進行內容驗證。

證明需要包含一個「種子」文件地址、" 隨選串流 " 的「校驗摘要」及三個「資訊片段」地址（至少有一個為「賦價」片段）。「記帳節點」需要挑選儲存上述四個「資訊片段」（含「種子」首片段）的 " 合格 "「存儲節點」進行獎勵；" 合格 " 是指「存儲節點」與 " 獎勵地址 " 的距離小於等於指定的 " 存儲距離難度 "（difficulty of storage）。早期由於節點稀疏，區塊鏈設定的 " 存儲距離難度 " 會較小（也就是接受較大的 " 存儲距離 "）；隨著「賦價」片段數量的增加，「存儲節點」會更傾向保存地址更為接近的「資訊片段」，因此 " 存儲距離難度 " 也可以適當地逐漸提高。

" 獎勵地址 " 除了設定為「資訊片段」地址外，為避免 " 自私節點 " 不傳播或不提供遠離自身的「資訊片段」所造成的「流言中斷攻擊」（Gossip Interruption Attack），足夠接近「資訊片段」與「記帳節點」邏輯距離 " 中點 "（mid-position）位置上的「存儲節點」也得以挑選出來進行獎勵；雖然 " 中點存儲節點 " 不是必要的獎勵對象（也可能找不到），但由於「記帳節點」本身可獲得的獎勵與其對「存儲節點」的 " 獎勵總和 " 正相關，因而執行有助於增加備用存儲來源的 " 中點獎勵 " 將可獲得更多的激勵。

## 2.4 存儲與檢索 Storage and Retrieval

提出「工作量證明」並挑選合格「存儲節點」的過程其實就是對 DHT 存儲網路進行多次「存儲」（Store 訊息）、「檢索」（Find_Proof 訊息）與「下載」（Find_Value 訊息）的請求過程。提出「工作量證明」時，「記帳節點」本地端應該已暫存若干的「賦價」片段（包含「區塊鏈」、「收據」片段）、候選的「種子」文件及組成其內容的「隨選串流」，並建議依以下步驟進行：

1. 依序挑選「工作量證明」中「種子」首片段與三個「資訊片段」當作 " 目標片段 " 地址以搜尋其合格的「存儲節點」；
2. 向 " 目標片段 " 與本地之邏輯距離 " 中點 " 所在的 K-bucket 內節點請求 " 中點 " 的「存儲證明」（即 Find_Proof 訊息，使用隨機的「索引」參數）；
3. 被詢節點應該無法提出「存儲證明」（因為 " 中點地址 " 剛好是「資訊片段」的機率很低），但能回應 " 更接近 " 的節點列表；

4. 重複向 " 更接近 " 節點請求 " 中點 " 的「存儲證明」，直到取得 " 合格 " 節點列表（即距離小於等於 " 存儲距離難度 " 的節點列表）；
5. 同時向多個 " 合格 " 節點請求 " 目標片段 " 的「存儲證明」（即 Find_Proof 訊息，使用預選的「索引」參數）以取得「子片段」（sub-piece）的「校驗摘要」。記錄「索引」、「校驗摘要」、" 邏輯距離 " 及 " 回應時間 " ；
6. 向 " 目標片段 " 所在 K-bucket 內節點請求「存儲證明」（應與步驟 1 並行），直到取得合格節點列表；
7. 同時向多個 " 合格 " 節點發出 " 目標片段 " 的「下載」請求（即 Find_Value 訊息，使用與步驟 5 相同的預選「索引」參數）。合格節點如回應「子片段」數據，則記錄「索引」、數據之「校驗摘要」、" 邏輯距離 " 及 " 回應時間 " ；
8. 比對步驟 5 與步驟 7 記錄之「索引」與「校驗摘要」的一致性（或與本地數據確認），挑選 " 邏輯距離 " 最短 " 、回應時間 " 最快的「存儲節點」進行獎勵。

提出「工作量證明」的三個「資訊片段」中，通常會包含一個 " 新生 " 片段（意即除了本地端外，其它節點不知道其片段內容），需要先發起「存儲請求」才能從回應的「存儲節點」中挑選出合格的獎勵對象，此時建議的步驟如下：

1. 向 " 中點 " 所在 K-bucket 內節點發起 " 新生 " 片段的「存儲請求」（Store 訊息）；
2. 等待接收「Store 訊息」的節點提出「Find_Value」請求，回傳 " 請求節點 " 對應「索引」的「子片段」數據並維護 " 請求節點 " 列表、記錄各節點請求的「索引」資訊；
3. 等待 " 新生 " 片段傳播一段時間或當 " 請求節點 " 列表中開始出現接近 " 新生 " 片段的 " 合格 " 節點後，執行下一步驟；
4. 向 " 新生 " 片段所在 K-bucket 內節點請求 " 新生 " 片段的「存儲證明」（Find_Proof 訊息，使用隨機「索引」參數），直到取得合格「存儲節點」的列表；
5. 同時向多個 " 合格 " 節點發出 " 新生 " 片段的「下載」請求（即 Find_Value 訊息，使用與 " 請求節點 " 列表記錄不同的預選「索引」參數）以取得「子片段」數據。記錄「索引」、數據之「校驗摘要」、" 邏輯距離 " 及 " 回應時間 " ；
6. 同時向多個 " 中點 " 的 " 合格 " 節點發出 " 新生 " 片段的「存儲證明」請求（即 Find_Proof 訊息，使用步驟 5 的預選「索引」參數），直到取得「子片段」的「校驗摘要」。記錄「索引」、「校驗摘要」、" 邏輯距離 " 及 " 回應時間 " ；
7. 比對步驟 5 與步驟 6 記錄之「索引」與「校驗摘要」的一致性（或與本地數據確認），挑選 " 邏輯距離 " 最短、" 回應時間 " 最快的「存儲節點」進行獎勵。

### 2.5 鑄造交易 Mint Transaction

依上節所述，提出「工作量證明」的過程中使用的「Find_Proof 訊息」、「Find_Value 訊息」及「Store 訊息」也正好是分散式存儲中最重要的三種工作協議。「存儲節點」因而無法區分協議請求是用於 " 無償共享 " 或是 " 獎勵對象 " 之用，這使得「易幣」能夠成為「去中心化存儲系統」的有效激勵機制。

「記帳節點」如果挑選最接近 " 同盟節點 " 的「賦價」片段當作固定的 " XOR 參數 " 時，那麼在同一秒內產生出來的「區塊鏈」地址皆會相同（因為沒有類似 Nonce 的可調欄位，除非持續加入新的交易請求或合約執行，否則打包的數據不會變更）；當證明失敗、需要變更「種子」文件的「隨選串流」對象或 " XOR 參數 " 時，也需要重新從網路中檢索出八個 " 合格 " 的「存儲節點」並且還要向「全節點」詢問「種子」文件及「存儲節點」的 " 最終轉址 "，才能將獎勵寫入「鑄造交易」（Mint Transaction）區中。可以說，「易幣」不是 " 算力集中 " 而是 " 網路 IO 集中 " 的一種挖礦；或者 " 挖礦 " 一詞可能並不適合用來描述「易幣」，因為節點間的互動效率才是決定 " 出塊 " 速度最主要的因素，而非重複嘗試數學難題。

「工作量證明」與〈2.3.2 小節〉提到的「頭獎證明」、「叔塊」（Ommer Block）的獎勵是被記錄在「鑄造交易」區內。獎勵的「序列順序」依序為「叔塊」>「頭獎證明」>「工作量證明」獎勵，內容則包含以下多個「鑄造交易」：

- 對前一「區塊鏈」及「叔塊」的「首片段」地址，給予 1.0 YIC 的 「賦價」。
- 對本次「區塊鏈」的「次片段」地址，給予 1.0 YIC 的「賦價」。
- 將三個「資訊片段」與「種子」的「首片段」與「賦價」操作有關的所有 UTXO 作為 " 輸入總和 " ，再平分「賦價」到這四個地址。
- 各分片的交易、合約「收據片段」，皆給予 1.0 YIC 的「賦價」。
- 「種子」文件的 " 最終轉址 " 獲得 0.1*(1+ " 輸入總和 " ) 個 YIC 的「匯款」。
- 合格「存儲節點」的 " 最終轉址 " 獲得 0.2 YIC , if distance<=10 或 0.1*(1+1/lg(distance) YIC, if distance>10 的「匯款」。
- 上項獎勵最多有三組，分別對應著與「叔塊」、「頭獎證明」及「工作量證明」；每組最多可以獎勵八個合格「存儲節點」。
- 「頭獎證明」的「記帳節點」額外獲得 10.0 YIC，「叔塊」的原「記帳節點」獲得 8.0 YIC 的「匯款」。
- 每個分片「驗證節點」獲得 1.0 YIC +交易手續費 的「匯款」。
- 每個分片「執行節點」獲得 1.0 YIC +執行手續費 的「匯款」。
- 「記帳節點」獲得上述最多廿四個合格「存儲節點」的 " 獎勵總和 " 的「匯款」。
- 「記帳節點」獲得分片中 " 最低 " 交易手續費, if maximum distance<=10 或 " 最低 " 交易手續費*1/lg(maximum distance), if distance>10 的「匯款」。
- 「記帳節點」獲得分片中 " 最低 " 執行手續費, if maximum distance<=10 或 " 最低 " 執行手續費*1/lg(maximum distance), if distance>10 的「匯款」。
- " 公益地址 " 獲得以上總和 2 % ， " 維運地址 " 獲得以上總和 5 % 的「匯款」。

### 2.6 通貨膨脹 Inflation

前一節有關「鑄造交易」中，「種子」及其 " 互為冗餘 " 的三個「資訊片段」是以平均「賦價」方式進行 " 存儲權重 " 的再配置。這就是〈 2.2.2.2 小節〉所提到「賦價」的 " 銷毀性 "（Burning）成本：即「賦價」用的「易幣」無法再次被提取使用，但可以通過「鑄造交易」的方式產生流動。事實上 " 賦價平均 " 是一種 " 稀釋手段 " ，「工作量證明」中的 " 新生 " 片段做為 " 已賦價 " 片段的 " 冗餘存在 " ，稀釋了原片段的 " 存儲權重 " ，使得「存儲節點」轉而尋求儲存有更高「賦價」的片段，因而鼓勵對文件存儲有久遠需求者再次投入「賦價」，藉此降低鑄造所產生的「通貨膨脹」。因此「易幣」無需隨著時間限縮 " 出礦額度 " 、能夠維持一定強度的獎勵以各種身分參與的輕、全節點，藉以打造更有活力的生態圈。

## 三. 動態擴展 Dynamic Expansion

### 3.1 分片機制 Sharding

當有大量「交易驗證」（Transaction Validation）及「合約執行」（Contract Execution）的請求需要處理時，單一節點可能無法在 " 出塊周期 " 內完成所有的工作；為此「易幣」採用一種稱為「分片」（Sharding）的擴展方案，可以將處理工作分配至最多 65,536 個平行程序上。這雖然無法縮短區塊鏈的 " 出塊 " 時間（受限於網路回應的整體效率，即使是調整 " 區塊鏈距離難度 " ，縮短範圍仍有其限度），但可以大幅提高「每秒事務處理量」（Transactions Per Second, TPS）以應付 Web3.0 的全球性交易需求。

「交易驗證」的分片規則較為單純，只要取「驗證節點」地址最後 8 個十六進制字符所對應的 UInt32 數值，再依 " 驗證分片數量 "（shards of validator）欄位求得餘數，餘數相同的「驗證節點」被分配在同一分片中，負責具有相同餘數之 " 輸入帳戶 "（Input Account）所

組成交易的驗證。也就是說，「易幣」要求構成一個交易請求的所有 " 輸入帳戶 " 需要具有相同的餘數，這可以簡單地避免分片間的「雙花攻擊」（Double Spending）。實務上，使用多個不同 " 輸入帳戶 " 的交易請求通常是因為單一來源的輸入金額不足，這可以透過使用固定 " 找零帳戶 " 來降低未來匯聚的難度，或是以「自動合約」方式執行涉及到多個 " 輸入帳戶 " 的交易請求。

「合約執行」的分片規則較為複雜，它要求依前一個「區塊鏈」的「首片段」地址的最後 5 個位元代表的正整數位置（範圍 0 - 31），從「執行節點」地址的對應字符位置開始取出 4 個十六進制字符所對應的 UInt16 數值（範圍 0 - 65535），再依 " 執行分片數量 "（shards of executor）欄位求得餘數，餘數相同的執行者被分配在同一分片中，負責具有相同餘數之「合約地址」的執行處理。因此「自動合約」每一次執行皆可能被重新分配到不同的分片、由不同的執行者處理，這可以平衡分片節點群之算力不均的情況，也可以避免惡意的 " 女巫攻擊 "（Sybil Attack）。

### 3.2 趨同選舉 Convergent Election

「易幣」的分片採用的是一種稱為「趨同選舉」（Convergent Election）的共識機制，與生物學的「趨同演化」（Convergent Evolution）相似：即處理相同交易或合約集合最終會產生出相同的結果集合。透過對交易或合約處理後之 " 結果集合 " 的多輪投票，最終會匯聚出 " 最高 " 的投票數，也就代表了在該分片中的最大共識。

#### 3.2.1 Vote 訊息

「投票節點」包含「驗證節點」及「執行節點」兩種身分，能從當前區塊裡的 " 驗證分片數量 " 或 " 執行分片數量 " 欄位，分辨出應處理的交易或合約是位於哪個分片中，並在處理完每一筆交易或合約後，就會在「記帳節點」及同一分片的網路節點間進行「投票傳播」（Vote 訊息）。「投票傳播」請求與「資訊片段」的「存儲請求」（Store 訊息）一樣，採用具有指向性的流言協議：「存儲請求」時，訊息會優先向已知最接近目標的地址傳播；而在「投票傳播」時，則是優先向最接近 " 投票議題 "（即「收據片段」）的地址傳播。

投票封包的格式如下:

| 項目 | 欄位 | 形態 | 描述 |
|---|---|---|---|
| Subject 議題 | Receipt | Char[40] | 包含原始交易（Transaction）或衍生交易（Derivative）收據的首分段地址。 |
| | Block | Char[40] | 當前區塊鏈的首分段地址。 |
| | Height | UInt64 | 當前區塊鏈的高度。 |
| | Type | Char[2] | TV：交易驗證投票（Transaction Vote）或<br>CV：合約執行投票（Contract Vote）。 |
| | Shard | UInt16 | 所屬分片索引值（餘數）。 |
| | Round | UInt32 | 投票回合。 |
| | Fee | UInt64 | 手續費，以爻（Yao）表示； $10^{15}$ yao = 1 YIC。 |

| 項目 | 欄位 | 形態 | 描述 |
| --- | --- | --- | --- |
| Voters 投票者列表 | Voter | Char[40] | 投票者地址。 |
| | Queue | UInt32 | 分片中交易或合約池內未處理的數量。用於計算執行率，以控制下一區塊的分片數量。 |
| | Distance | UInt8 | 投票者與收據地址邏輯距離（logic distance）的前置零位數量，範圍為 0 -160。 |
| | Signature | Char[..] | 投票簽章。 |

在每輪投票中，該輪的 " 投票議題 " （subject）即是 Receipt 欄位中的「收據片段」地址。正常情況下，同一議題從 Receipt 欄位到 Fee 欄位間的資訊集合應該一致，否則表示處理的結果出現了分叉。 " 投票者列表 " （Voters 欄位）則至少包含一個「投票節點」的資訊，使用列表的原因是因為流言傳播的特性：愈靠近「收據片段」的接收者應該會接收到愈多對該 " 投票議題 " 的投票，網路節點會將接收到而仍未傳送出的投票者資訊匯聚成 " 投票者列表 " 後再朝 " 議題方向 " 傳遞。從某個意義上來說，靠近「收據片段」地址的接收節點可以視為分片中的 " 協調者 " （Coordinator），只是這些 " 協調者 " 僅負責某一投票回合中，對某一議題選項的計票而已；同時 " 協調者 " 也可以不用與所有的投票者建立連線，這樣能避免網路壅塞造成的阻斷情況。

接收節點透過暫存 " 投票議題 " 及 " 投票者列表 " 的資訊，可以判斷個別投票者在每一回合中，是否僅投出一票，據以評估投票者的可信度；也可以建立某一投票者的 " 投票歷程 " 或是統計出同一分片中具較高手續費的「收據片段」、其 " 回合數 " 與 " 得票總數 " 等資訊，以作為 " 議題推薦 " 的來源，以提供「記帳節點」在「分片聚合」（Braid up）過程中變換 " 目標議題 " 的參考。

### 3.2.2 收據片段 Receipt Piece

「收據片段」也是一種「資訊片段」，與「區塊鏈片段」的結構一樣，用位元塊（chuck）進行數據儲存；其 " 檔頭 " 包含「排序雜湊樹根」（Sorted Merkle Root）及「布隆過濾器」（Bloom Filter），其後則保存了「原始交易」（Transaction）及「衍生交易」（Derivative）的交易訊息。「原始交易」是由「錢包節點」所提出的交易請求；「衍生交易」則是分片驗證「原始交易」或執行「自動合約」後衍生的交易結果。例如：對一個具有 " 轉址地址 " 的「帳戶地址」進行「OP_SEND」時，會額外增加一個輸入為 " 原始地址 " 、輸出為 " 最終轉址 " 的「衍生交易」。「衍生交易」是一種共識操作：即「衍生交易」中不需要提出公鑰及簽章即能提取 UTXO ，這也是「自動合約」執行後記錄其狀態改變的方式。

「原始交易」及「衍生交易」會依其「交易編號」（Transaction ID, TXID）先進行排序，再組成「二元雜湊樹根」(Merkle Root），排序確保了多個交易即使先後執行順序不同，仍會產生出相同的數據。「布隆過濾器」是另一個在「收據片段」中即使因執行順序不同，也能保證相同結果的數據，它是被用來作為快速判斷與 " 查詢帳戶 " 相關的交易是否被包含在此「收據片段」中的重要工具。最後，則是儲存排序後的「原始交易」或「衍生交易」內容的列表。

「收據片段」的內容在每輪投票後，也就是每次新增「原始交易」或「衍生交易」時，需要重新排序交易列表、產生新的「二元雜湊樹根」，並在「布隆過濾器」中增添新的向量賦值。當「記帳節點」想要查詢與某個「帳戶地址」有關的所有交易輸出入時，它可以從保存在「區塊鏈」 " 檔頭 "（即首片段的第一個子片段）內的「鑄造交易」及「布隆過濾器」中得知目標地址是否有交易被記錄在此「區塊鏈」中。只有在「布隆過濾器」計算出現 " 陽性 " 後，才需

再下載完整的「區塊鏈片段」，從中取得各分片「收據片段」的 " 檔頭 " ，再依序從 " 檔頭 " 保存的「布隆過濾器」中計算出是否需要下載完整的「收據片段」以進行內部檢索。如此一來就能大幅縮短「輕節點」（例如「錢包節點」）重建帳戶狀態所需時間並降低對系統的負擔。

### 3.3 分片聚合 Braid up

「分片聚合」的共識機制要求「記帳節點」在每一段 " 聚合時序 " （也就是一次對「工作量證明」的嘗試）、從每一分片網路中，挑選出一個含有「最高」手續費的 " 投票議題 " （即「收據片段」）。含 " 較高 " 手續費的「收據片段」未必屬於 " 較高 " 的投票回合，因為「投票節點」接收到「交易請求」集合的時序可能不同；而且「記帳節點」也有自行決定優先處理 " 較高 " 手續費之交易或合約的權利，所以這裡的 " 最高 " 只是一個短暫且動態的局部解。具體的 " 投票議題 " 挑選步驟建議如下：

1. 從監聽的「交易驗證」或「合約執行」的「Vote 訊息」中，同時挑選出多個 " 較高 " 手續費的「收據片段」作為 " 目標議題 " ；
2. 向已知參與該分片且最靠近 " 目標議題 " 的節點（「投票」或「記帳」節點皆可）傳送「 Find_Vote 訊息」，即詢問其 " 得票總數 " 、 " 投票歷程 " 或 " 議題推薦 " ；
3. 被詢問節點回覆是否對 " 目標議題 " 投下一票、其統計的 " 得票總數 " 、已知最接近 " 目標議題 " 的 " 投票節點列表 " 及這些「投票節點」各別的 " 投票歷程 " ，還有最接近的 " 記帳節點列表 " 及已知手續費最高的幾個 " 議題推薦 " ；
4. 由於 " 指向性傳播 " 的關係，回覆的 " 得票總數 " 有機會高於先前步驟所取得的數值，因為回應節點更接近目標，應能更快收集到投票傳播。若 " 得票總數 " 沒有增加，表示可能該議題可能不具共識，此時應從 " 議題推薦 " 或 " 投票歷程 " 中，挑選另一 " 目標議題 " ，並重複第 2 步驟；
5. 若 " 得票總數 " 合理增加，則重複第 2 步驟向更接近目標的節點詢問，直到找到 " 非陌生 " 且 " 有投票 " 的「投票節點」。當其距離小於等於指定的「分片距離難度」（ difficulty of shard）時，即為 " 合格 " 的「驗證節點」或「執行節點」；
6. 透過查詢「鑄造交易」中 " 獎勵對象 " 的 " 最終轉址 " 或 " 冗餘片段 " 與「賦價」有關的 UTXO 資訊，能確認 " 合格 " 節點是否具有「全節點」身分。注意：查詢「頭獎證明」（PoJ）所需訊息時，應避免洩露完整內容；
7. 請求「 Find_Value 訊息」，以取得完整的「收據片段」。若 " 合格 " 節點無法回應，則應挑選 " 更高 " 手續費的 " 目標議題 " 重複第 2 步驟；若無新的 " 目標議題 " ，則重複第 6 步驟，持續挑選其它的 " 合格 " 節點進行查詢；
8. 從「收據片段」取出「排序雜湊樹根」（Sorted Merkle Root）及「布隆過濾器」（ Bloom Filter），並與其它分片內數據依 " 分片索引值 " （餘數）進行排序後，打包聚合成「區塊鏈片段」；
9. 比對「區塊鏈」的「首片段」與「工作量證明」中 " XOR 參數 " 的距離是否 " 合格 " （需小於等於指定的 " 區塊鏈距離難度 " ），是則發布新「區塊鏈」（即廣播 Generate 訊息）；否則重新進行「分片聚合」。

#### 3.3.1 交換律 Commutative Property

與比特幣 UTXO 模型不同，「易幣」擴展了「OP_SEND 指令」 的操作，使其匯款對象會因該地址過往的交易狀態而改變。「驗證節點」在驗證「交易請求」時，需先確認匯款的 " 目標地址 " 是否已經達到「OP_ASSIGN 指令」的生效條件，如果「 OP_ASSIGN 」生效，生效的「轉址」會取代原本的 " 目標地址 " ；此時「驗證節點」還需依序對後續「轉址」進行同樣檢查，直到找到該次匯款的 " 最終轉址 " 為止。如果「交易驗證」時「 OP_ASSIGN 」生效於「 OP_SEND 」操作前，則會產生一個「衍生交易」，用來記錄「轉址」交易的資訊；如果「 OP_SEND 」操作在「 OP_ASSIGN 」生效前，則不用產生「衍生交易」。

由於上述特性，「分片聚合」如果不能保證其操作順序，則會喪失「交易驗證」的 " 可交換性 "（Commutative Property）。此外，「交易驗證」的分片規則是依 " 輸入 " 來源的「帳戶地址」進行分配，與 " 輸出 " 的 " 目標地址 " 無關，不同分片的「驗證節點」可能同時對某個目標「帳戶地址」進行「 OP_ASSIGN 」及「 OP_SEND 」 操作，且彼此互不知曉、也無法產生交互作用。同理，「合約執行」與「交易驗證」也分屬不同分片，當前「交易驗證」後的狀態改變也無法對「合約執行」產生的「衍生交易」產生同步的影響，反之亦然。為了維持「易幣」操作的「交換律」，並讓「趨同選舉」能夠產生出相同結果的收據集合，所以 " 當前區塊 " 是以 " 前一區塊 " 記錄下的 UTXO 作為交易的初始狀態。

例如：與「 OP_ASSIGN 」生效時同一 " 鏈高 " 下的「 OP_SEND 」 操作，仍會「匯款」至原本指定的 " 目標地址 " 。一個 " 期限競標 " 的合約在執行時，僅會從早於先前 " 鏈高 " 的交易中挑出「 OP_SEND 」操作 " 最高價 " 的得標者，而 " 當前區塊 " 才加入競標的「匯款」則會留待下一次 " 出塊 " 時才會進行處理。

### 3.3.2 忽略執行 Ignore Execution

由於「交易驗證」或「合約執行」（算力集中）的速度會遠快於 " 投票統計 "（網路 IO 集中），「分片聚合」時通常無法包含所有的「交易驗證」或「合約執行」請求，所以 " 交易池 " 中可能會留下一部分的「交易請求」，而某些合約則可能會被忽略而不無法加入記帳。另外，某些「交易驗證」或「合約執行」的集合（收據片段），可能無法滿足 " 分片距離難度 "（difficulty of shard），則該集合亦不會被納入當前的 " 聚合時序 "（一次對工作量證明的嘗試）的聚合中。幸好，「交易驗證」請求是動態且持續地被加入 " 交易池 " 中進行驗證，集合不斷變化的「收據片段」也能保證不會有特定交易被持續排除；同樣地，「自動合約」的集合在每次分片時會依規則重新分配；同時，手續費扣除等的一些微少變化也足以導致合約產生的「收據片段」會不斷改變。

此外，在投票過程中有一個「佇列」（Queue）的欄位，它跟「投票回合」（Round）欄位的加總，等於該分片中 " 交易池 " 或 " 合約池 " 的總數。合約分片在聚合時理想的「佇列」值應該為 0，這表示所有合約皆被執行完畢。若所有分片的「佇列」值都太大，則應該調整 " 執行分片數量 "（shards of executor），若只有少數分片的「佇列」值太大，則應該降低合約的 " 分片距離難度 "。還有一個更簡單、可以避免交易或合約被忽略的方法，就是提高其被執行的 " 優先權 "，也就是對合約進行「充值」（ OP_CHARGE 指令）或是增加交易的手續費用。

雖然滿足 " 分片距離難度 " 這個設定條件，在某些情況下會造成上述問題，但是這能鼓勵該「投票合回」仍 " 不合格 " 的「投票節點」積極進行新一輪的投票。新一輪投票會抬高聚合後能獲得的總手續費，使得前一輪 " 合格 " 的「投票節點」不得不放棄既有優勢，隨之也展開新一輪的投票；" 分片距離難度 " 也使得「記帳節點」無法隨意挑選偏愛（由同盟或分身擔任）的「投票節點」，而「全節點」與「輕節點」擔任「記帳節點」的優勢差異不大，所以能吸引大量「輕節點」參與投票傳播網路，從而降低少數惡意「投票節點」就能造成破壞的可能性。

## 3.4 分片攻擊 Shard Attack

「易幣」的共識機制要求「記帳節點」在每一分片中應挑選具 " 多數 " 投票且含有 " 最高 " 手續費的「交易驗證」或「合約執行」的集合。該集合是以「收據片段」進行記錄，與該「收據片段」地址 " 最近 " 距離的投票者，則為該分片中首選的獎勵對象。雖然共識機制中只要求「記帳節點」 " 足夠 " 接近即可，但「記帳節點」的獎勵是用全部分片中 " 最低 " 的那筆手續費及 " 最遠 " 距離共同計算得出（ 請參考〈2.5 鑄造交易〉一節），所以為獲得最大化的獎勵，「記帳節點」應會盡量挑選 " 最近 " 「收據片段」地址的「投票節點」。

設計讓「收據片段」當然的「存儲節點」（邏輯距離最接近）獲得獎勵的額外理由，是因為「收據片段」是由個別「投票節點」獨立驗證交易或執行合約集合後的產出；不實際進行相同

集合的操作，＂懶惰節點＂就無法得知確切內容，也就無法提出身為「投票節點」的證明。只是這樣一來，「記帳節點」也可能無法收集「收據片段」內容來進行「分片聚合」，因為「投票節點」會擔心＂懶惰節點＂透過請求證明來取得「收據片段」，並瓜分誠實「投票節點」被「記帳節點」挑選的機會。因此一個要求＂足夠＂靠近、滿足「分片距離難度」的條件限制，可以提供一個誘因讓合格「投票節點」在無法確定獎勵前，仍願意公開「收據片段」的內容。

### 3.4.1 盲目與從眾投票 Lemmings & Bandwagon Vote

一個不進行「交易驗證」或「合約執行」，只是＂從眾投票＂〈Bandwagon Vote〉的「全節點」，稱為＂懶惰節點＂。建立＂懶惰節點＂的成本較低，能用數量搶奪其它誠實「投票節點」被挑選的機會，如果誠實參與「交易驗證」或「合約執行」的「全節點」比例降低會嚴重影響分片的安全性。還好，光從「收據片段」的地址並無法推導出確切內容，更無法判斷「投票議題」的正確性，這能降低＂盲目跟投＂〈Lemmings Vote〉的發生率，或至少讓＂懶惰節點＂比＂誠實節點＂更晚才開始投票，因為＂懶惰節點＂需要時間確認該議題有複數得票，以避免投票到更惡意的＂虛偽議題＂上。投票時間的延後會降低被挑選的機率，再加上有「分片距離難度」的限制，將使得＂盲目跟投＂的風險過大而且效率不彰。

此外，「記帳節點」在連線到＂懶惰節點＂前，就應該已經得到其部分的＂投票歷程＂，因為前一連線節點在提供最接近＂目標議題＂的＂投票節點列表＂給「記帳節點」時，會一併提供其個別的＂投票歷程＂。每個「投票議題」只會出現在特定的「投票回合」，這使得＂投票歷程＂也只能形成特定的組成順序，因此無法每輪隨意投票給較接近＂同盟節點＂的議題。只要跟從其它節點取得的＂投票歷程＂比對＂懶惰節點＂前一輪的投票目標，即能發現＂盲目投票＂與＂從眾投票＂所造成的矛盾順序組合。

### 3.4.2 學人精 Imitator

進階的＂懶惰節點＂也可能會鎖定某一「投票節點」，模仿其完整的＂投票歷程＂，稱為＂學人精攻擊＂（Imitator Attack）。＂學人精＂只需要在足夠靠近該輪投票的「投票議題」時，再向模仿的「投票節點」詢問以取得「收據片段」內容即可。還好，因為＂指向性傳播＂的特性，取得完整＂投票歷程＂的網路成本很高；如果＂學人精＂恰好遠離該次「投票議題」，可能就不會或是要延後很久才會從網路上收到模仿的「投票節點」在該輪的投票傳播。由於「易幣」不提供直接詢問＂投票歷程＂的工作協議，＂學人精＂只能用該輪可能的「投票議題」作為「 Find_Vote 訊息」的參數，來詢問並試探被模仿的「投票節點」是否投下一票；或是重新選擇上輪投票相同的其它「投票節點」作為新的模仿對象，才能複製出合理且完整的＂投票歷程＂。

＂學人精＂為了加速模仿＂投票歷程＂，也許可以偽裝大量的「記帳節點」收集＂投票歷程＂，但是無法大量偽裝成「投票節點」，因為「記帳節點」在挑選合格的「投票節點」前，會先驗證其「全節點」的身分是否正確，也就是查詢某些「全節點」必須維持的全狀態數據。即使大量偽裝「記帳節點」並配合＂同盟＂的「全節點」進行協同模仿攻擊，複製＂投票歷程＂與模仿「全節點」工作所造成的時間差（受限於網路速度），仍然會降低其被挑選到的機率，還不如誠實執行「投票節點」的相應工作（受限於計算速度）來得有效率。

### 3.4.3 多重人格 Multiple Personality

惡意節點可能採用一種更有效率的協同攻擊，讓＂同盟＂的「投票節點」及「記帳節點」使用彼此的＂投票歷程＂，這相當於＂礦池＂（Mining Pool）的概念。此時＂同盟＂的「投票節點」會避免與外部節點直接連線，也不會向外部節點進行投票傳播，不讓外部節點有機會驗證其＂投票歷程＂。＂同盟＂對外的「記帳節點」會隨著外部節點所詢問的「 Find_Vote 訊

息」中之「投票議題」的欄位，選擇讓最接近的 " 同盟節點 " 套用對應的 " 投票歷程 " ，將外部節點引導並連線到最接近「投票議題」的 " 同盟節點 " 上。

" 同盟節點 " 如同有 " 多重人格 " （Multiple Personality）一般，能依詢問的「投票議題」套用對應的 " 投票歷程 " ，也能回應完整的「收據片段」，同時還能呼應 " 同盟 " 的「記帳節點」所提供的 " 偽證 " 。不過，不跟外部節點直接連線，不可避免地會延長其回應時間，導致外部「記帳節點」在等待過程中，可能直接選用回應時間較短的其它合格「投票節點」，或重新挑選 " 更高 " 手續費的其它「投票議題」為新的目標。此外，" 同盟 " 的「投票節點」只會在 " 同盟泡泡 " 內傳播，也就是說外部的「記帳節點」在進行投票傳播過程中，不會傳播 " 同盟節點 " 任何的 " 投票歷程 " 。「記帳節點」為了方便當區塊鏈出現分叉時能快速 " 回滾 " ，會暫存最近幾個 " 鏈高 " 的 " 投票歷程 " ，不存在過往暫存歷程中 " 陌生 " 的「投票節點」將不被視為 " 合格 " 的「投票節點」，這可避免這種形式的協同攻擊。

### 3.4.4 自私同盟 Selfish Alliance

採用 P2P 技術的存儲系統通常會利用某些「阻塞算法」（例如 BitTorrent 的 Tit-for-Tat 演算法）以避免 " 自私節點 " 對整體效能的衝擊。在「易幣」中，一個節點若不願意提供其它更接近的節點列表，其 " 連線排名 " 會逐漸降低，最終喪失參與網路的機會。只是可供連線的「存儲節點」數量很多，" 自私節點 " 可以不斷變換相鄰節點而在 " 存儲網路 " 中存續較長的時間，因此「易幣」導入了對 " 存儲距離中點 " 的獎勵機制；而在 " 投票網路 " 中可供連線的「投票節點」相對數量較少並且排斥 " 陌生 " 的「投票節點」，所以才不需要額外增加這類 " 選配 " （optional）的獎勵對象。

如果考慮已 " 合格 " 之「投票節點」的自私行為：一個「投票節點」為了增加自身被獎勵的機率，可能不願意提供其它更接近 " 目標議題 " 的 " 投票節點列表 " （當然也可能真的只是沒有其它更接近的「投票節點」）。不論無法提供更接近的 " 投票節點列表 " 的原因為何，提供更接近的 " 記帳節點列表 " ，亦能讓「記帳節點」可以從不同路徑趨近 " 目標議題 " ，再從中挑選出較佳的 " 合格 " 的「投票節點」。由於「記帳節點」是由不需分片的全部「輕節點」擔任，與經由分片後的「全節點」所擔任的「投票節點」相比，在數量上多出了數個 " 量級 " （order of magnitude），因此不提供更接近的 " 記帳節點列表 " 將有更高的 " 阻塞權重 " 。" 自私節點 " 也可能僅願提供 " 同盟 " 的節點列表，不過這在大型網路中會降低與系統的 " 連接度 " （connectivity），將更難被其它「記帳節點」挑選（可參考上一小節的內容）。總而言之，系統中誠實「記帳節點」的數量如果遠大於參與「交易驗證」與「合約執行」的「投票節點」，那麼個別「投票節點」的自私行為，將難對系統整體聚合的效率產生太大影響。

### 3.4.5 少數權利 Minority Rights

雖然「易幣」的共識機制希望獲得 " 最多得票數 " 的議題獲得挑選，藉以降低挑選到 " 惡意投票議題 " 的可能性，但這是基於 " 區塊鏈產出 " 時間大於或接近於 " 分片完全聚合 " 的時間（也就是假設在新區塊鏈 " 出塊 " 時，多數交易皆被驗證、全部合約皆被執行完畢）的情況下，此時無論先前經歷哪些 " 投票歷程 " ，最終多數投票會收斂到少量個「投票議題」上：例如，收斂到幾個剛剛才加入的「交易請求」或是最後幾個有較少「充值」的「自動合約」上。從宏觀的角度來看，少量「投票議題」在巨量複雜路徑的傳播過程中，獲得 " 最多贊同票數 " 的 " 投票議題 " 能接觸到最多的「記帳節點」，也就有最高的機率會被挑出聚合，這幾乎可以排除 " 虛偽議題 " 被挑選到的可能性了。

在最初的幾輪投票中，雖然 " 交易池 " 中只有上次遺留的少量請求可供驗證，但新增「交易請求」仍會不斷湧入；而可供執行的「自動合約」一開始很多，卻會隨著 " 投票回合 " 逐漸減少。無論是「交易驗證」或是「合約執行」，每個「投票節點」所挑選出的集合很可能不盡相同，舉一個極端的例子來說：當第一輪合約投票時，每個「投票節點」可能會投最接近自身的「投票議題」，而非擁有最高「充值」的合約一票。當然，要在「執行合約」前就先判斷產出

的「收據片段」是否接近自身是很困難的，光是手續費扣除的數據變化就足以使雜湊後「收據片段」的地址完全不同；不過「投票節點」仍然可以先同步執行多個合約，再考慮以何種順序組合來進行投票最為有利。

如上所述，投票初期各個議題的 " 得票數 " 可能十分平均，而統計所有議題 " 得票數 " 的時間成本極大，因為個別節點能接收到的投票傳播並不全面。如果每個「記帳節點」以 " 遍歷 " 最靠近「投票議題」節點的方式進行詢問 " 最完整 " 的 " 得票數 " ，很快就會造成整個網路的壅塞了。因此配合「指向性傳播協議」的特性，「易幣」建議「記帳節點」同時對多個 " 起始議題 " 進行朝向該 " 起始議題 " 方向的詢問：在每一個 " 聚合時序 " （準備提出工作量證明前），比較出 " 較高手續費 " 且有 " 較高得票數 " 的議題；並在下一個 " 聚合時序 " 時，再從前次 " 投票歷程 " 的基礎上，尋找下一輪 " 較高手續費 " 及 " 較高得票數 " 的 " 目標議題 " 。

這在動態的投票過程中，無可避免的只能得到一個短暫而局部的最佳解。或許可以認為這是一種尊重少數投票者的 " 民主風範 " （democratic manner），或是當作「記帳節點」用降低自身獎勵方式來換取挑選獎勵對象的一種特權。其它進行「區塊鏈」驗證的節點只要確認「收據片段」不是 " 虛偽議題 " 、合格的「投票節點」確實存在，即應給予尊重。同樣地，當系統中「記帳節點」由單純「輕節點」擔任的數量遠多於身兼「全節點」身分所擔任的數量時，特意挑選 " 同盟 " 或自身「投票節點」的行為對系統的衝擊也會降低。

### 3.5 自動合約 Smart Contract

「易幣」定義了一種可以在區塊鏈上運行一連串程序代碼的特殊協議，稱為「自動合約」。跟以太坊不同，「易幣」中「自動合約」並不是經由「交易請求」觸發，而是主動性、周期性地自動執行；「自動合約」的 " 執行周期 " 與 " 出塊 " 周期相同，合約「執行節點」在驗證 " 出塊 " 的正確性後，即會開始執行所屬分片 " 合約池 " 內尚有「充值」餘額的「自動合約」。

#### 3.5.1 合約編繹 Contract Compile

「全節點」可選擇擔任「合約執行節點」的身分並負責維護 " 合約池 " 的狀態。當一個全新區塊產生後，在聚合「交易驗證」（儲存在對應分片的「收據片段」上）的「原始交易」或「衍生交易」中，與「OP_CHARGE 指令」有關的、新建立的「合約片段」會被加入 " 合約池 " 中等待執行。合約第一個執行動作是從存儲網路 " 讀取 " 合約內容加以 " 編繹 " ，並產生一個「衍生交易」：「衍生交易」中會將所有「OP_CHARGE 指令」的 UTXO 進行 " 輸入匯聚 " 後扣除手續費，再以單一「OP_CHARGE」的方式 " 輸出 " 。「OP_CHARGE」的 " 輸出 " 包含額外的「EXTDATA」參數，被用來記錄 " 讀取失敗 " 、 " 編繹失敗 " 、 " 編繹成功 " 、 " 執行失敗 " 、 " 執行成功 " 、 " 合約註銷 " 等訊息。

啟動「自動合約」文件開始進行 " 編繹 " 的「 OP_CHARGE 指令」與啟動「種子」文件「權利轉讓」的「 OP_ASSIGN 指令」相似，交易的「驗證節點」只需要確認該地址對應的內容是「位元塊鏈」（chuck-chain）的格式且所有「次片段」亦皆已「賦價」即可，不用真正對「種子」文件進行解讀或對「自動合約」文件進行 " 編譯 " 。換言之，「種子」文件的正確性是由後續產出區塊的「記帳節點」檢驗；而合約的程式碼是否可 " 編譯執行 " 是由後續的「合約執行節點」檢驗，皆與當前執行「OP_CHARGE 指令」的「交易驗證節點」無關。

每次 " 執行周期 " 一到，合約指定的程序會被執行並扣除對應的「充值」以作為「執行節點」的手續費。所以即使是 " 讀取失敗 " 的合約也會持續不斷地扣除「充值」，此時 " 起草人 " （版權宣告者）應對「合約片段」增加「賦價」並再次發起「Store 訊息」，直到合約能被 " 讀取 " 並開始進行 " 編繹 " ；同樣地，" 編繹失敗 " 也會持續重試並扣除「充值」，但 " 起草人 " 可以在後續的「OP_CHARGE 指令」中增加「AVOID 參數」進行 " 合約註銷 " ，並取回剩餘的「充值」；如果 " 編繹 " 成功，「合約執行節點」也不會馬上執行合約內容，只會在「衍生交易」中產生一個 " 編繹成功 " 的 " 輸出 " （即會在 UTXO 中新增包含 OP_CHARGE EXTDATA compiled 的字串）後完成程序。合約執行會等到下一個 " 執行周期 " 才進行。

### 3.5.2 合約執行 Contract Execute

合約 " 編繹 " 後會在 UTXO 中留下 " 編繹成功 " 的 " 輸出 " ，它被用來提供「交易驗證節點」區分「OP_SEND 指令」的款項應該被當作「賦價」（即「OP_VALUE 指令」，可用來提高「合約片段」本身存儲的 " 重要性 " ）或是「委托」款項給「自動合約」進行運用。因此，合約 " 委托人 " 應該在合約 " 編繹成功 " 後，才進行「OP_SEND」的操作，否則其款項會被變作「賦價」而被 " 銷毀 " （如〈 2.2.2.2 小節〉所述）。「OP_SEND 指令」可以配合「EXTDATA 參數」記錄簡短的 " 附加訊息 " 。「EXTDATA 參數」可以用來標記「匯款」的來源或目的，例如：一位匯款人需要付出的付款金額是來自於其擁有的多個不同帳戶，而這些「帳戶地址」在交易時可能分屬於不同分片、無法在同一個「交易請求」中被驗證，此時 " 附加訊息 " 可以協助收款人總和多個付款來源。「EXTDATA 參數」也能被合約所定義的程序所讀取，因此上述存款或付款的需求也能透過一個類似銀行功能的「自動合約」來達成。

「自動合約」還能讀取「ORACLE 參數」用以載入 " 外部數據 " ，參數通常會指向一個最近產生、記錄重要數據的「資訊片段」地址。「ORACLE 參數」主要跟「OP_CHARGE 指令」一起使用，它被限定只有由合約的 " 起草人 " （即版權宣告者，首位對「合約片段」進行「匯款」或「賦價」的「帳戶地址」）或 " 首位見證人 " （即首位對「合約片段」進行「充值」的「帳戶地址」）發出才有效。「自動合約」每次執行後，至少會產生一個「衍生交易」，其中包含「OP_CHARGE 指令」用以匯總剩餘的「充值」（如〈 2.2.2.4 小節〉所述）；如果合約有需要記錄大量的 " 衍生數據 " 作為憑證，也能生成並發布「資訊片段」，並將「片段地址」寫入「ORACLE 參數」中，以供下次執行時載入。

「自動合約」主要處理的是對該地址進行「OP_CHARGE 指令」及「OP_SEND 指令」的 UTXO，其 " 序列順序 " （ sequence order ）是依所屬「區塊鏈」的 " 鏈高 " ；同一區塊內則依「鑄造交易」、「交易分片序列」、「合約分片序列」；同一分片的則依「交易編號」（Transaction ID，可能為「原始交易」或「衍生交易」）進行排序。對於某些交易順序敏感的事務，例如：限量發行或是設有出價上限的競價活動，格外有用。「自動合約」也可以實時調用其它合約內的函式或程序、讀取該合約的 " 初始狀態 " ；但無法以 " 被調用合約 " 身分產生任何的「衍生交易」，自然也無法對該合約產生任何即時影響。此外，「自動合約」產生的「衍生交易」也可能包含「 OP_VALUE」 、「OP_ASSIGN」、「OP_CHARGE」等指令，但因為合約本身無法提供公鑰及簽章，所以如果用合約執行「版權宣告」，未來也只能由合約來執行「權利轉讓」。

## 四. 討論與建議

### 4.1 生態系 Ecosystem

#### 4.1.1 物聯網 IoT

在「易幣」的「輕節點」所擔任的身分中，工作負載最輕的是「錢包節點」，其次是「存儲節點」。一般來說，「錢包節點」被認為是一種 " 軟體 " 或 " 應用 " 節點，而「存儲節點」則更偏向是一種 " 硬體 " 節點。「存儲節點」至少會使用一個「帳戶地址」作為網路的「通用唯一辨識碼」（UUID）、具備數據傳輸的能力及一定的儲存空間，這恰好也是「物聯網」設備的最低要求。「存儲節點」僅需回應與自身「帳戶地址」及存儲「資訊片段」相關的協議請求，或是負責維護相鄰的連線節點列表，即有一定機會獲得獎勵「易幣」。「物聯網」設備本身若需運算協作或有遠端存儲大量數據的需求，可以「易幣」進行資源交換、擴大單機的適用場域。可以預想，未來資訊通訊設備、尤其是企業或家用網路分享 / 路由器，應該內建「存儲節點」功能以善用其閒置頻寬及存儲空間。

#### 4.1.2 存儲代理服務

雖然有存儲需求的「易幣」參與者，可以直接對組成完整文件的個別「資訊片段」進行「賦價」並將其廣播出去，但即使給予「資訊片段」較高的「賦價」也無法保證能被存儲網路 " 永久 " 保存，因為「資訊片段」的「賦價」可能因參與「工作量證明」而被 " 稀釋 " 、或是因為整個系統的存儲成本上升或 " 資源短缺 " （「賦價」片段大量增加或「存儲節點」大量減少時），連帶提高了 " 保證存儲 " 的「賦價」平均值 ── 考慮當 " 偏僻 " 地址的「資訊片段」位於 " 熱門 " 存儲地址附近時，" 保證存儲 " 的門檻會變低；反之，當許多 " 高賦價 " 的「資訊片段」集中在缺乏「存儲節點」的地址附近時，便需要再增加「賦價」以提高鄰近「存儲節點」的儲存意願了。當然，每個被 " 稀釋 " 的「資訊片段」皆有其 " 冗餘片段 " 可供還原，只是這些 " 冗餘片段 " 同樣也可能有「賦價」不足的問題，需要額外的管理及維護成本。

如果想要儲存的檔案尺寸大於一個「資訊片段」的容量（ 256 KiB）時，可以選擇以「位元塊鏈」形式記錄；如果遠遠大於 256 KiB 則建議以「種子」形式保存，這時需要額外再對「種子片段」進行「賦價」；如果檔案尺寸遠小於 256 KiB 時，又會希望整合多個小型檔案到一個「資訊片段」中以降低「賦價」成本。以上種種情境皆需要持續監控存儲市場並即時針對個別片段進行管理，對於一般使用者而言太過複雜而且缺乏效率，使用「存儲代理服務」是較為簡單的方式。

### 4.1.3 匯流、串流與應用服務

影音創作者以「種子」形式發布的作品有機會獲得「工作量證明」的獎勵，其機率與影音的 " 熱門 " 程度正相關，因為「記帳節點」如果挑選 " 冷門 " 又無「賦價」的「資訊片段」提出證明會有導致 " 分叉 " 的風險。將「種子」提交到多個「影音匯流平台」是增加「資訊片段」被緩存的一個簡易策略 ──「匯流平台」提供的檢索及推薦工具可以有效提高曝光度，還有獲得廣告收益的機會。

「匯流平台」為了提供更好的觀看體驗，通常會建立龐大的「串流伺服器」群組，也會開發出對應的應用終端。伺服器或應用終端的底層，如移動裝置的作業系統（Operating System, OS）所緩存的「資訊片段」在「易幣」中是有價值的，建議連網的作業系統應該內建「存儲節點」的功能以獲取可能收益；同時伺服器與應用終端也能充當 " 礦池 " ，因為它們流通「資訊片段」、統計「種子」文件的熱門程度，並能利用 " 閒置算力 " 協助提出「工作量證明」。以下簡述一個加密傳輸串流內容並協助提出「頭獎證明」的步驟：

1. 應用端提交 OS 底層所暫存的「區塊鏈片段」清單給串流端；
2. 串流端將 512 KiB 緩存依奇偶位拆解成兩個 256 KiB 位元流、以隨機挑選的「區塊鏈片段」進行「爻變」後傳送給應用端；
3. 應用端計算兩個「爻變」後位元流的地址作為「校驗摘要」並回傳串流端；
4. 串流端驗證摘要、檢索兩個「校驗摘要」是否符合「頭獎證明」，是則提供給「匯流平台」的「記帳節點」提出證明；
5. 串流端告知應用端兩個「校驗摘要」是否正確、回傳採用的「爻變」地址及真實 512 KiB 緩存的「校驗摘要」；
6. 應用端「爻變」暫存的兩個 256 KiB 位元流，依奇偶位重組成 512 KiB 緩存；
7. 應用端計算、比對 512 KiB 緩存的「校驗摘要」是否正確，正確則將 512KiB 緩存加入播放序列，再重複第 2 步驟請求後續片段。

## 4.2 已知問題

### 4.2.1 鏈下存儲 Off- Chain Storage

嚴格來看，「易幣」的 " 鏈上存儲 " （on-chain storage）能力十分有限：「錢包節點」只需儲存「區塊鏈片段」的 " 檔頭 " 用來查詢與「帳戶地址」相關的資訊；「存儲節點」或「記帳節點」這種「輕節點」可能只儲存高「賦價」或接近自身地址的「區塊鏈片段」；只有「全節點」（「交易驗證」或「合約執行」節點）才有提供完整的「區塊鏈片段」以供新加入節點

查詢的義務而已。除了「區塊鏈片段」外，完整的交易歷史與合約狀態是被共同記錄在「收據片段」中，而「收據片段」是用「賦價」方式激勵「存儲節點」進行保存，對「存儲節點」而言，「收據片段」只是另外一種有 " 較高 " 「賦價」的「資訊片段」（因為「鑄造交易」會給予初始 1.0 YIC 的「賦價」）而已。

沒有被共識機制強制要求的存儲數據只能被稱為 " 鏈下數據 "（off-chain data）。從這個標準來看：「工作量證明」中相關的「種子」文件、「資訊片段」及「合約片段」或合約產生的 " 外部數據 "，雖然它們共同記錄完整「區塊鏈」訊息的一部分，但也是一種 " 鏈下數據 "，也會隨著參與「工作量證明」被 " 稀釋 " 而逐漸降低「賦價」（也可以認為是隨著時間而降低其重要性），是有永久丟失數據的風險存在的。

### 4.2.2 軟分叉 Soft Fork

值得注意的是，如果有一個「工作量證明」包含的「種子」文件或是「自動合約」因其「賦價」太低、喪失存儲價值，而導致新加入的「全節點」無法進行 " 全系統狀態 " 的重建、新生「區塊鏈」的驗證或合約的 " 讀取 "、" 編繹 " 乃至 " 執行 " 等工作，最終可能會導致 " 軟分叉 "。建議獲得 " 記帳權 " 的「記帳節點」應同時發起對「工作量證明」中相關「種子」文件與「資訊片段」的再「存儲請求」（Store 訊息），以緩存系統的快取（cache）狀態；而獲得「合約執行」獎勵的「全節點」也被建議保存 " 低賦價 " 的「合約片段」，以提供新加入的「全節點」重建系統狀態。或許隨著時間推移，造成 " 軟分叉 " 的「自動合約」最終會被「執行節點」汰除（無法在「趨同選舉」中取得共識），這也算是一種 " 退場機制 "；若要完全避免退場，則應該還是要由 " 諦約者 " 對「自動合約」進行「賦價」補充，才符合「市場供需法則」吧。

另一個可能造成 " 軟分叉 " 的情況，是基於網路基礎層或是「存儲節點」的「可靠性」（reliability）問題：對一個誠實的「記帳節點」來說，即使多方驗證「存儲證明」並挑選頻寬穩定的「存儲節點」進行獎勵，負責 " 區塊鏈驗證 " 的節點（「全節點」或其它「記帳節點」）複驗時仍有可能因為「存儲節點」的臨時斷網、主機當機或「分散式阻斷服務」（distributed denial-of-service，DDoS）而造成該次 " 出塊 " 被拒絕承認。這個情況或許可以透過其 " 直接連線節點 " 進行 " 名譽背書 " 來緩解：即 " 直接連線節點 " 維護有已知該「存儲節點」所保存的「資訊片段」列表所對應的「布隆過濾器」（Bloom Filter），並在「Find_Value 訊息」或「Find_Proof 訊息」查詢時一併提供證明，表示更接近的節點列表中確實有保有目標的「資訊片段」。只是過濾器要有效過濾一個「存儲節點」所保存的大量「資訊片段」，可能需要很長的資料結構，這也可能導致「Find_Value 訊息」或「Find_Proof 訊息」的回應效率下降；而且當儲存的集合有變更時（通常是刪除元素），更新過濾器的工作負擔也可能太高。

### 4.2.3 不可能三角 Blockchain Trilemma

多數加密貨幣所談論的「每秒事務處理量」（TPS），是指每秒能處理多少數量的交易數據，如果 TPS 太低則容易造成手續費高、確認時間長、甚至造成某些交易可能會被永遠忽略；更糟糕的是這個 " 每秒 " 並是不保證每秒皆有交易事務能被處理，充其量只是一個計算上的平均值而已，如果想要正確描述事務的處理效能就必須一併考慮 " 出塊周期 " 才行。比特幣的平均 " 出塊周期 " 被設定為 10 分鐘、以太坊大約 15 秒左右，而其它不是採用 PoW 共識的加密貨幣則號稱能以更短的時間產出區塊鏈。

事實上，" 效能 " 對於去中心化的系統而言往往是被妥協的一方，要想達到如 VisaNet 網路接近 56,000 TPS 的交易吞吐量，雖然可用 " 垂直性 " 的 Layer 2 方案導入 " 更中心化 "（centralized）的中介處理程序，但這違背了無須依賴第三方介入的初衷；想要只透過 " 橫向 " 的「分片擴展」達到真正意義上的高 TPS 是不可行的，主要原因在於去中心化網路的訊息傳遞與整合需要較長的反應時間，這同樣也是限制「易幣」交易效能的瓶頸所在，或許未來有著大頻寬、低延遲特性的 5G 網路的全面普及後，才可能從根本上解決這個問題。

從標榜高性能的 Solana（每秒 6 萬筆交易）在面對「高運算交易」時發生過多次 " 斷鏈 " 的悲劇，可以省思區塊鏈的「不可能三角」（Blockchain Trilemma）應該以何者為重；或者換個角度來看，" 效能 " 帶來的體驗如果是首要的追求目標，那麼比特幣可能永遠不會被發明出來。「易幣」雖然具有「分片聚合」技術所帶來的強大動態擴展性，但也為 " 高效運用 " 作出妥協：在「趨同選舉」過程中 " 高得票率 " 議題（即收據片段）內已被驗證的交易請求也可以作為一個臨時性的交易憑證（主要用於小額交易等 " 秒級應用 " 之中），不需要真正等到多個 " 出塊周期 " 成為不易回滾的最長主鏈後才算確認。

### 4.3 結論 Conclusion

現代通貨（currency）通常是由中央銀行控制其發行量，並由國家或國家聯盟透過政策、外交乃至於戰爭方式，控制通貨的需求狀態以維持對其的信任度。可以說，" 信用 " 是通用貨幣的本質，而比特幣十多年的實驗已經證實了運用區塊鏈技術的加密貨幣，在安全性上已達到其所宣稱的：能夠基於密碼學原理而非信任，使得有交易意願的雙方不需要透過第三方即能完成交易。當然，對貨幣的信賴不僅只是建立在交易安全性上，貨幣價值的穩定性更是重點，這才是為何大眾仍然只敢長期持有由國家或國家聯盟等級的貨幣當局所擔保的法定貨幣的原因。

在採用工作量證明的比特幣中，爭取 " 記帳權 " 所付出的資源就相當於法幣貨幣當局進行儲備、調控、甚至經濟戰爭的成本；或者說，如果比特幣沒有消耗國家級別的電力，那麼是否也無法支持國家體量的貨幣擔保呢？從這個角度來看，合併後採用 " 權益證明 " 的以太坊就失去了穩定貨幣價值的依據，只能由仍掌握流通量的 " 寡頭 " 或 " 聯盟 " 所組成的類貨幣當局，進行 " 中央集權式 " 的貨幣政策調控；或是透過以太坊白皮書所宣稱的「去中心化自治組織」（Decentralized Autonomous Organization, DAO）來進行匯率治理。雖然後者仍不失其去中心化的意旨，但如同以國力擔保貨幣價值是有必要的，採用「工作量證明」的共識機制也有其必要性，是其貨幣價值的一種體現，不應該隨便捨棄或污名化。重點是，如同國家財政政策投入的資源應該被用於促進經濟的正向發展，「工作量證明」的能量消耗也應該要能創造出額外的價值，這正是「易幣」：一種能為去中心化存儲網路創造數據冗餘與流通共享的公共區塊鏈平台與眾不同之處。