國立中山大學資訊工程學系

碩士論文

Department of Computer Science and Engineering

National Sun Yat-sen University

Master Thesis

比特幣共識演算法模擬與分析

Simulation and Analysis of Bitcoin Consensus Algorithm

研究生：施雲柔

Yun-Rou Shih

指導教授：官大智 博士，蔣依吾 博士

Dr. D. J. Guan ,Dr. John Y. Chiang

中華民國 107 年 9 月

September 2018

國立中山大學研究生學位論文審定書

本校資訊工程學系碩士班

研究生施雲柔（學號：M053040025）所提論文

比特幣共識演算法模擬與分析
Simulation and Analysis of Bitcoin Consensus Algorithm

於中華民國 107 年 8 月 29 日經本委員會審查並舉行口試，符合碩士學位論文標準。

學位考試委員簽章：

召集人 陳嘉玫 陳嘉玫　　委　員 官大智 官大智

委　員 王智弘 王智弘　　委　員 _____

委　員 _____　　委　員 _____

指導教授(官大智) 官大智 (簽名)

學年度： 107

學期： 1

校院： 國立中山大學

系所： 資訊工程學系

論文名稱 (中)： 比特幣共識演算法模擬與分析

論文名稱 (英)： Simulation and Analysis of Bitcoin Consensus Algorithm

學位類別： 碩士

語文別： 中文

學號： M053040025

提要開放使用： 是

頁數： 30

研究生 (中) 姓： 施

研究生 (中) 名： 雲柔

研究生 (英) 姓： Shih

研究生 (英) 名： Yun-Rou

指導教授 (中) 姓名： 官大智 博士，蔣依吾 博士

指導教授 (英) 姓名： Dr. D. J. Guan, Dr. John Y. Chiang

關鍵字 (中)： 比特幣, 區塊鍊, 乙太坊, 共識演算法, 去中心化

關鍵字 (英)： Bitcoin, blockchain, Ethereum, consensus algorithm, decentralization

Acknowledgments

碩士生涯即將進入尾聲，非常感謝官大智教授這兩年來的悉心教導，老師除了教導我許多專業知識，研究上給予我們協助，也教導我們許多做人處事的道理，讓我受益良多。感謝額碩學長在研究上的幫忙與指點，感謝浩媗，在我有疑問時，和我一起討論研究，也感謝其他同學們的幫助及陪伴。最後，謝謝我的父母及家人，讓我衣食無缺，可以專注在研究上，也在我遇到困難時，給予我鼓勵。再次感謝所有人，致上萬分的謝意。

施雲柔

8/24/2018

摘要

比特幣是目前區塊鍊最早也十分熱門的應用之一，其不需通過可信任第三方機構，能更有效率地交易，因為不須透過第三方，所以節點之間會形成點對點分散式網路來傳送區塊，共識演算法則確認節點們能達成共識，確定大家的帳本都是一致的。比特幣的共識演算法是選擇區塊鍊中的最長鏈，每經過 6 個區塊的運算後，該區塊就會被固定下來。而我們提出另外一個乙太坊的共識方法，是使用樹結構來儲存區塊鍊，每個樹節點的分支最多為三個，若 degree 大於 11 該區塊就會被確認。我們模擬並做了兩種方法的比較，認為比特幣的共識演算法是比乙太坊的共識演算法更適合及穩定。

關鍵詞：比特幣, 區塊鍊, 乙太坊, 共識演算法, 去中心化

Abstract

Bitcoin is one of the earliest and most popular applications in the blockchain. It does not need a trusted third-party organizations to trade more efficiently. Bitcoin nodes form a peer-to-peer distributed network to transfer blocks. Moreover, the consensus algorithm confirms that all nodes can reach a consensus and make sure that everyone's ledger are consistent. Bitcoin's consensus algorithm is to select the longest chain. After every 6 blocks, the block will be fixed and confirmed. And we propose another consensus method of Ethereum, which uses the tree structure to store the blockchain. Each tree node has a maximum of three branches. If the degree is greater than or equal to 11, the block will be confirmed. We simulated and compared this two methods, and believed that Bitcoin's consensus algorithm is more suitable and stable than Ethereum's consensus algorithm.

Keyword: Bitcoin, blockchain, Ethereum, consensus algorithm, decentralization

Contents

Chapter 1

緒論

1.1 研究背景

現今的金融交易都需要透過一個可信任的第三方，像是銀行或政府機構來確保交易的完成。而中本聰於 2008 年所提出的比特幣 [1]，解決了交易都必須經過第三方機構來完成的問題，通過去中心化和點對點分散式網路讓參與交易的每個人都能看見一個公開的資料庫帳本，並進行維護，大家都能成為比特幣的一個節點，來進行交易或參與製造比特幣，而節點所形成的點對點分散式網路會將所有比特幣的產生和交易紀錄廣播給全部的節點，讓大家一起維護同一份帳本，使其無法輕易的被偽造。比特幣的核心技術稱為區塊鏈，是一種不依賴第三方、通過自身分散式節點進行網路數據的存儲、驗證、傳遞和交流的一種技術方案。每個區塊會包含許多筆的交易，藉由計算工作量證明（Proof-of-Work），來決定區塊是否能被接上區塊鏈上，區塊鏈會一直持續延長，而且新區塊一旦加入到區塊鏈中，在連續得到 6 個區塊確認之後，區塊就會被固定下來，也代表大多數人都認同這些區塊，所以區塊內的交易也就能得到確認了。

1.2 研究動機

近年來，區塊鏈成為熱門核心技術之一，去中心化和分散式的特性顛覆了傳統，更加便捷及自主。區塊鏈廣泛地運用在各領域，而比特幣是區塊鏈最著名的應用之一，因此，我們實作出比特幣網路並了解其共識演算法，我們提出和比特幣不同的共識演算法，採用乙太坊的樹結構，並計算子樹數量的方法，去做實驗，並將結果做分析比較。

1.3 論文架構

本論文的其他部分包含: 第二章相關背景知識，第三章為比特幣共識演算法模擬，並詳細說明的兩種共識演算法，第四章會分析所提出的共識演算法，最後一章為結論及未來展望。

Chapter 2

基礎知識

比特幣是 2008 年中本聰（Satoshi Nakamoto）提出的一種數位貨幣 [1]，和傳統貨幣不一樣是，比特幣運作方式不仰賴中央銀行、政府企業為第三方，而是根據點對點分散網路所達成的網路協定，去中心化、分散式的虛擬貨幣，因此確保了任何一個人、機構、或政府都無法操控比特幣的交易、產生或製造通貨膨脹。

2.1 比特幣的特點

1. 去中心化: 不需仰賴一個中央機構來進行交易，所以更有效率，也不會受到政府或機構的影響，完全由節點們控制。

2. 分散性: 比特幣沒有任何中央銀行的背書，也沒有任何人、任何公司或政府擁有控制比特幣的權利，依賴參與比特幣的每個人來做維護，因此也沒有傳統貨幣通貨膨脹的疑慮。

3. 帳本公開性質: 每筆交易都會公開在網絡上供人查閱檢視，這些記錄會儲存於區塊鍊中，每個人都可以要一份有比特幣完整收付紀錄的帳本存在電腦上。[2]

2.2 比特幣的運作方式

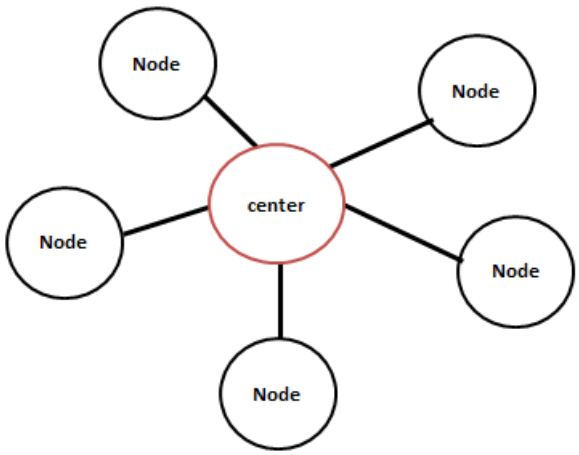
在此章節，我們將針對比特幣的運作方式來做深入的認識，了解比特幣如何建立 A Peer-to-Peer Electronic Cash System，在不經由第三方的情況下進行交易。

2.2.1 Transaction

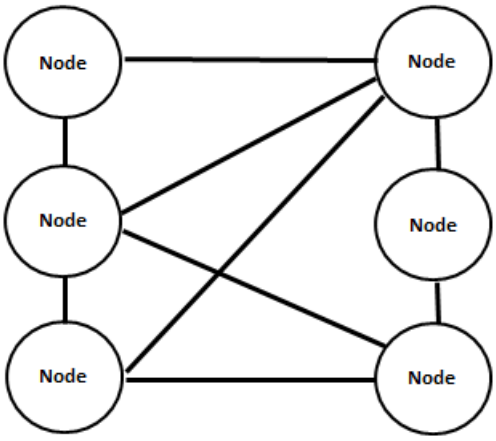
每個所有者將上一筆交易和自己的公鑰做 hash 之後，再做簽章，並將這些資料添加到硬幣的末端來將硬幣轉移到下一個。收款人可以驗證簽章以驗證所有權，[3] 每個區塊要放入哪些交易是由礦工決定，比特幣有區塊尺寸的限制，每個區塊只能放入有限的交易，部分的交易金額會做為報酬給挖礦的礦工，也就是區塊的創建者，礦工們通常會去計算交易費率 (fee rate)，費率高的交易會有比較大的機會被放入區塊中，藉此礦工能得到較高的報酬。另一方面，對使用者來說，他們能在確認時間和成本之間進行權衡，具有高時間要求的使用者可以支付高額的交易費用以便快速確認。[4]

2.2.2 network

比特幣網路是點對點網路，任何人都能加入或退出，若加入比特幣網路，就會成為節點，每個節點都會保留一個完整的分類帳副本，並且獨立地驗證從其他節點接收的信息。[5]



圖形 1. 有中心的中央網路系統



圖形 **2.** 比特幣所採用的點對點網路系統

運行網絡的步驟如下：

1. 新的交易被廣播到所有節點。
2. 每個節點將新的交易收集到區塊中。
3. 每個節點努力為其區塊找到困難的工作證明。
4. 當節點發現工作證明時，它將區塊廣播到所有節點。
5. 節點只有在其中的所有交易都有效且尚未用完時才接受該區塊。
6. 節點通過創建下一個區塊，來表示已接受上一個區塊，已接受的區塊的 hash 作為 previous hash。

節點會認為最長的區塊鍊是最正確的，並將繼續努力擴展它。如果兩個節點同時廣播不同版本的下一個新區塊，則一些節點可以首先接收一個或另一個。在這種情況下，他們接收收到的第一個，丟棄另一個。

新的交易廣播不一定需要到達所有節點。只要它們到達許多節點，他們將長時間進入一個區塊。區塊廣播也允許丟棄的消息。如果節點沒有接收到區塊，它將在接收到下一個區塊時意識到它丟失了一個區塊並且請求重送。

在比特幣網路中，節點會將新建的區塊廣播出去，讓網路中的其他節點們收到新的區塊並更新自己的分類帳副本。在節點傳輸的過程中，會有傳輸延遲的時間，造成區塊鍊分岔 (fork)，分類帳副本不一致。

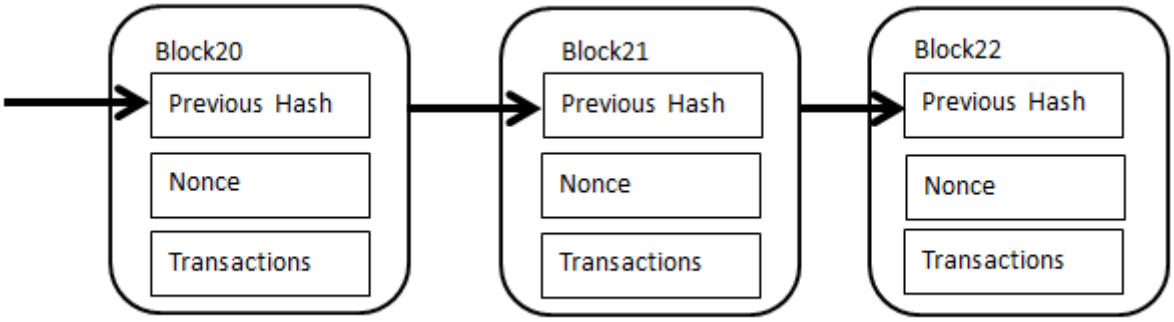
2.2.3 工作量證明

工作量證明是讓任一運算節點，花費時間和運算資源來計算出一組數學公式的結果，且要完成一次有效的工作量證明，需經過一連串地嘗試與失敗。不過，一旦這個數值被算出來後，其他參與節點也可用相關的數學公式，便能很容易去驗證這個值是否有效。[6]

比特幣區塊鏈採用 Hashcash（雜湊現金演算法）作為工作量證明，讓各節點經由 POW 計算來產生每一個有效的新區塊，再經由其他節點驗證並接受。進行 POW 計算的過程也被稱作挖礦。

由於每個區塊中 Block Header 會包含許多固定的值，其中只有 Nonce 值為一隨機值，因此每個節點進行 POW 計算時要算的就是，藉由不斷替換這個 Nonce 值，來讓這個區塊的 Block Header Hash 值，小於一個被設定好的難度目標值（Difficulty Target），至於為什麼要小於這個目標值，則是因為這個難度值意味著每個區塊在理論上應該要被產生完成。由於 POW 具有一定的難度，因此無法預期哪個運算節點可以最快算出新區塊，藉此來確保交易驗證的公正性。

難度值（Difficulty）是指節點要運算出低於困難度目標值的 Hash 值，平均需花多久時間，也就是平均要完成一次 POW 的時間。而比特幣區塊鏈目前設定為，大約每 10 分鐘會有節點成功算出新的區塊，不過這 10 分鐘只是基於理論值，實際每個新區塊產生的時間，有可能只需要 17 秒（第 407062 個區塊的實際產生時間），也有可能需要 20 分鐘以上。Difficulty 可動態調整，目前每產生 2016 個區塊會調整一次難度，以每 10 分鐘產生一區塊估算，大約是每兩周會調整一次 Difficulty。如果它們生成得太快，則難度增加。

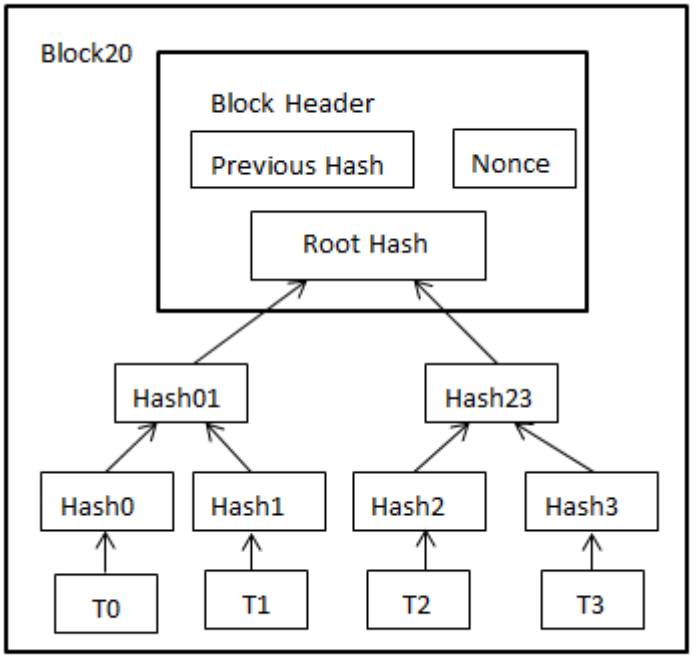


圖形 **3.** 將前一個區塊 *Hash* 後，儲存在下一個區塊中

2.2.4 回收磁盤空間

一旦硬幣中的最新交易被埋在足夠的區塊下，它之前的已用交易可以被丟棄以節省磁盤空間。為了方便這種情況而不破壞區塊的 hash，經由 Merkle Tree 將大量訊息縮短成一個 Hash 值。

在比特幣區塊鏈中，每筆交易產生後，都已經被 Hash 成一段代碼才廣播給各節點，不過這樣做還不夠，因為在各節點的區塊中，可能包含數百筆到數千筆的交易，因此，為節省儲存空間並減少資源耗費，比特幣區塊鏈的設計原理採用 Merkle Tree 機制，讓這些數百到數千筆的交易 Hash 值，經由兩兩一組形成一個新 Hash 值的方式，不斷重複進行，直到最後產生一組最終的 Hash 值，也就是 Merkle Tree Root，這個最終的 Hash 值便會被記錄到 Block Header 中，只有 32 Bytes 的大小。Merkle Tree 機制可大幅減少資料傳輸量與運算資源消耗，驗證時，只需驗證這個 Merkle Tree 的 Root 值即可。在 Merkle Tree 中對交易進行 hash 處理，只有根包含在塊的 hash 中。然後，可以通過將樹的分支截斷來壓縮舊塊。內部 hash 不需要儲存。



圖形 **4.** *Merkle Tree* 的 *Root* 值儲存在 *Header* 中

Chapter 3

比特幣共識演算法模擬

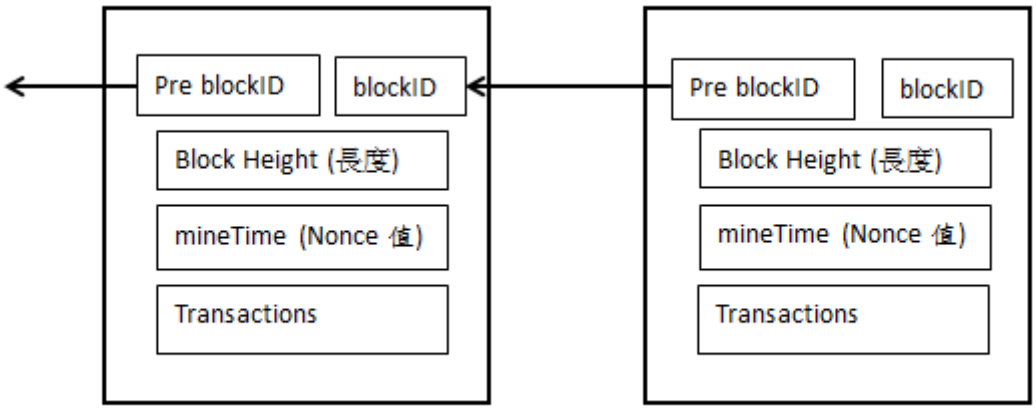
此章節會詳細說明我們如何建構出比特幣的環境，我們模擬區塊、節點以及事件等的發生，也詳細說明兩種共識演算法，比特幣共識演算法及乙太坊的共識演算法 [7]。

3.1 比特幣模擬環境

3.1.1 區塊

在 Bitcoin 中的每個區塊除了包含交易內容之外，還會記錄 nonce 值以及前一個區塊經過雜湊函數所產生的值 (previous hash)，藉由儲存 previous hash 的方式將區塊串聯起來，形成區塊鏈。在模擬中，每個區塊會有一個特定的 block ID，每個區塊除了自己的 ID 外，也會紀錄其所連接的上一個區塊的 ID(previous ID)，代替 previous hash 值，藉由儲存 previous block ID 的方式將區塊串聯起來，形成區塊鏈。

每個區塊的 Block Header 會包含許多資訊，其中所包含的 Nonce 值是一個藉由 hsah 隨機產生的數值，這樣的方式就是 Proof of Work(POW)，每個節點都會進行 POW 計算，算出這個 Nonce 值才能被接上區塊鍊，另一方面，這個 Nonce 值必需要小於一個被設定好的難度目標值（Difficulty Target），每過一段時間難度就會調整，為的就是不要讓區塊產生的過快或過慢。[8] 在模擬中，我們並沒有真的去計算 POW，因為計算出 nonce 值的時間是隨機的，所以我們也會隨機產生出計算出 nonce 值的時間，也就是此區塊產生的時間，並儲存為 mineTime 記錄在 block 中。

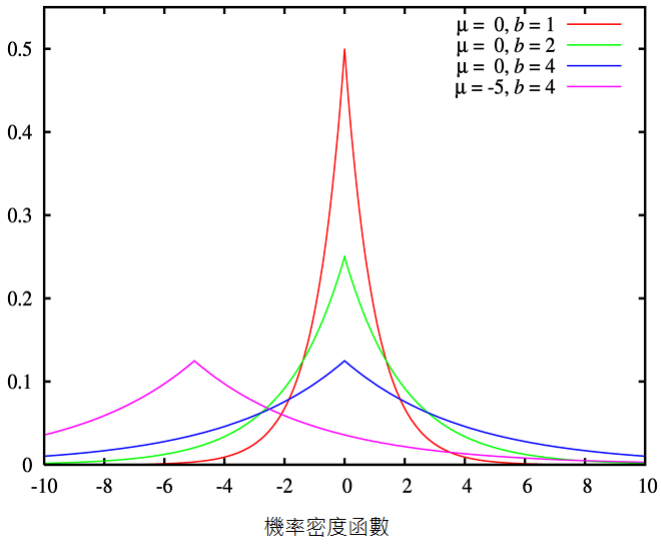


圖形 5.

3.1.2 傳輸延遲時間

比特幣是點對點網路，節點和節點間會有隨機產生的距離時間，這個時間也就是區塊傳送所需要的延遲時間，不同的節點間傳送所需要的時間也不相同，我們在設定傳輸延遲時間的時候，會以一個平均值加上誤差值，誤差值會採用拉普拉斯分布來產生。以下為拉普拉斯分布公式，μ 是位置參數，$b > 0$ 是尺度參數:

$$f(x|\mu,b) = \frac{1}{2b}\exp(-\frac{|x-\mu|}{b})$$



圖形 6. *Laplace distribution*

尺度參數的值越大，產生的值範圍也會越大，分布也會越廣。

3.1.3 節點的運作

1. 每個節點都會選擇最長的區塊鏈，繼續往下創建新的區塊。

2. 每個節點若成功地創建新區塊後會將其傳送給所有節點們。

3. 當節點收到新的區塊時，會將其接上區塊鍊。

4. 節點只有在其中的所有交易都有效且尚未用完時才接受該區塊。

5. 節點每次接收新區塊時會往前數 6 個區塊，會確認固定為 valid。

6. 所有節點確認 valid 的區塊都相同，才算是達成共識。

3.1.4 區塊產生時間/傳輸延遲時間關係

比特幣一個新區塊產生平均速率為 10 分鐘，網路延遲的平均時間為 25 秒的情況，因為每個節點都會計算 POW 創建新的區塊，所以在區塊產生時間很長，傳輸延遲時間很短的情況下，大家很快地就會接收到新創的區塊，將其接上區塊鍊並繼續算下一個區塊，但若是區塊產生時間和傳輸延遲時間很接近，代表會有些節點沒有辦法很快地接收到區塊，繼續算同一長度的區塊，區塊鍊就會有分岔的情況產生，所以當區塊產生平均速率與網路延遲的平均時間比值越小的話，區塊鍊分岔的情況就會越多。

3.2 共識演算法

3.2.1 比特幣共識演算法

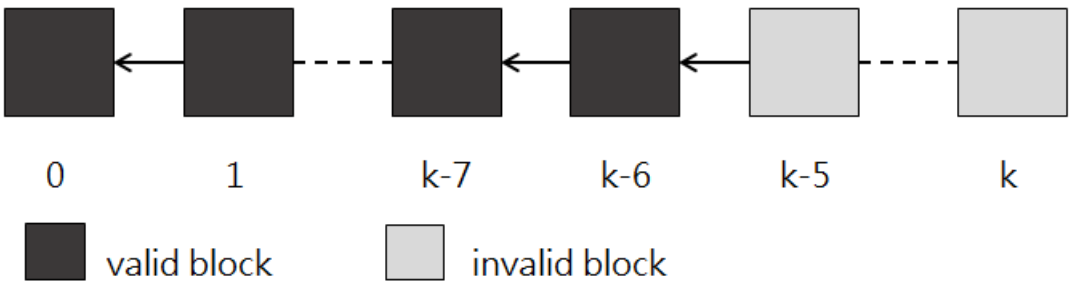
每一個節點都會分別儲存一份區塊鍊帳本，要讓所有節點同意相同的交易，讓大家達成共識，我們需要使用比特幣之共識演算法如下：

輸入: 一個代表區塊鍊的集合 $\mathbf{S} = \{b_1, b_2, \cdots, b_{n-1}\}$; 新產生的區塊 b_n
輸出: 一個代表已確認區塊鍊的集合 **T**

---

1. 加入新的區塊之後，更新整個區塊鏈 **S**
2. 根據每個區塊的 previous ID 找到所連接的區塊，找出區塊鏈 S 中長度最長鏈，長度為 k
3. 在最長鏈中，從最後的區塊往回數第六個區塊固定為 valid
4. Valid 的區塊儲存為 **T**

S 代表的是每個節點所儲存的區塊鍊帳本，根據上述的演算法，每一次算出新的區塊時，我們都會去判斷是否有區塊要被確認，被確認固定的區塊會被放入 **T** 中，代表所有節點都有相同的區塊，達成一致。[9]



圖形 **7.** 比特幣共識演算法

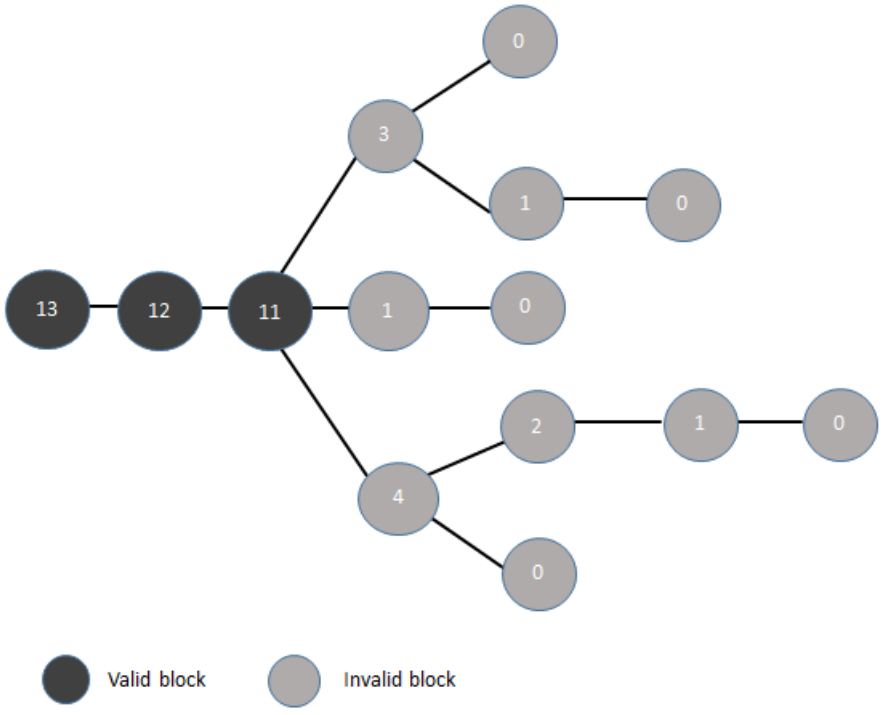
3.2.2 乙太坊共識演算法

乙太坊 [4] 將區塊鍊儲存為樹的架構，樹的每個節點最多只能有三個分支，每次接上一個新的區塊時，若是 degree 大於 11，該區塊就會被固定下來，樹結構之演算法 [10] 如下：

輸入: 一個代表區塊鍊的集合 $\mathbf{S} = \{b_1, b_2, \cdots, b_{n-1}\}$; 新產生的區塊 b_n
輸出: 一個代表已確認區塊鍊的集合 **T**

---

1. 加入新的區塊之後，更新整個區塊鏈 **S**
2. 從新加入的區塊往回將經過的區塊 tree counter 加 1，直到初始區塊 b_0
3. 在區塊鏈中，tree counter 大於等於 11 的區塊固定為 valid
4. Valid 的區塊儲存為 **T**



圖形 8. 乙太坊共識演算法

Chapter 4

共識方法分析與比較

在本章中，將會詳細說明兩個確認區塊的共識方法的結果與比較，第一個是比特幣所使用的共識方法，第二個是採用樹結構的乙太坊共識方法，在分支度 (degree) 數量大於等於 11 時，區塊會被確認。

我們的模擬中，節點數設為 200 個節點，假設每個節點的計算能力都相等，且每個節點都是礦工的情況下，測試區塊鍊長度和次數不定，紀錄達成共識的次數，然後分析並比較兩種方法。

- 節點數: 200 個，且計算能力皆相等
- 產生速率: 每個區塊的平均產生速率
- 延遲時間: 節點間的傳輸延遲時間
- 比值: 產生速率/延遲時間
- 共識機率: 共識的次數/總次數
- 丟棄率: 未被採用的區塊數/總區塊數

50	53	52	50	49	50	50	51	50	51
50	47	50	50	43	51	50	50	50	45
52	52	50	50	48	58	52	50	54	59
50	49	55	49	50	47	50	50	48	50
52	51	43	53	48	52	49	47	50	51
50	50	50	50	51	52	50	49	51	52
45	50	47	52	49	50	58	49	50	47
50	51	49	50	48	50	50	51	60	50
50	50	49	47	63	51	49	54	52	51
50	52	46	44	47	50	51	49	55	51
51	47	49	53	50	53	50	54	50	49
48	50	47	51	49	50	51	50	46	50
46	57	46	49	53	46	50	50	51	52
50	47	49	46	50	51	48	52	50	48
49	47	45	54	50	50	50	46	52	50
50	51	51	56	50	50	49	49	50	54
53	50	50	49	50	50	49	49	50	50
50	45	49	50	50	50	48	47	52	50
48	51	50	45	50	53	50	51	48	46
50	49	50	52	50	49	49	56	54	50
49	49	51	50	44	46	50	50	44	43
50	50	46	50	46	51	51	50	48	50
51	50	50	50	48	54	51	50	53	48
50	50	47	49	50	50	51	50	52	50
47	42	50	50	49	49	50	51	50	50
51	49	49	54	50	48	53	50	49	50
49	55	51	50	49	56	55	43	50	53
53	50	49	50	53	52	52	51	50	52
47	46	51	51	50	48	50	53	45	49
50	51	50	52	53	50	49	46	45	43
50	49	49	52	50	52	51	50	51	52
50	57	50	48	48	50	50	50	50	50
49	59	55	53	50	54	49	53	50	50
50	48	53	44	52	54	52	51	50	50
51	51	50	51	51	50	49	50	53	50
55	50	51	53	47	52	50	50	50	48
56	50	57	50	50	55	51	50	52	50
50	50	50	52	50	47	45	46	50	49
50	48	52	50	49	48	47	50	48	50
51	54	50	50	52	49	50	53	50	50
49	50	50	51	50	47	49	51	50	47

50	51	48	48	48	48	49	53	51	49
45	49	50	47	52	49	52	46	52	49
48	49	54	53	52	50	53	49	48	46
47	50	50	47	50	50	51	49	51	50
50	50	50	48	48	50	48	51	50	50
50	54	50	50	50	51	51	49	49	50
51	53	50	50	59	55	51	47	53	44
55	48	50	50	58	44	46	50	49	48
53	55	50	48	52	49	52	53	50	50
53	50	50	53	46	50	50	49	49	45
49	49	59	50	45	50	50	50	45	50
49	48	50	51	49	46	52	49	50	49
52	50	49	49	51	50	49	49	49	51
50	50	49	51	51	48	50	54	50	50
53	52	58	50	48	49	49	49	54	50
44	50	52	50	49	50	50	51	51	50
51	46	50	50	49	53	49	51	52	53
50	50	50	45	51	53	55	50	49	50
51	50	50	53	50	46	48	45	49	51
50	52	49	50	50	50	42	56	46	48
50	48	46	50	50	51	50	52	50	51
50	50	52	50	49	50	47	55	50	51
55	47	50	47	50	49	50	51	51	53
47	51	59	50	49	50	53	50	50	49
51	49	51	51	53	49	50	51	50	51
50	49	49	52	52	52	53	50	50	49
51	50	52	50	54	44	50	50	50	50
48	50	53	51	56	53	49	53	50	53
49	53	49	55	49	50	50	49	46	47
47	50	50	50	52	50	53	50	47	48
49	48	54	51	42	50	49	54	53	48
51	49	49	49	50	50	51	50	50	50
50	51	50	49	46	49	50	50	49	50
48	47	50	54	49	50	49	48	50	50
50	49	50	50	50	51	50	53	48	50
47	53	47	50	53	52	49	46	50	53
53	52	50	58	50	50	50	48	52	50
53	47	50	51	50	53	48	45	49	48
51	50	50	51	48	48	49	48	51	50
50	54	55	50	50	49	50	50	48	51

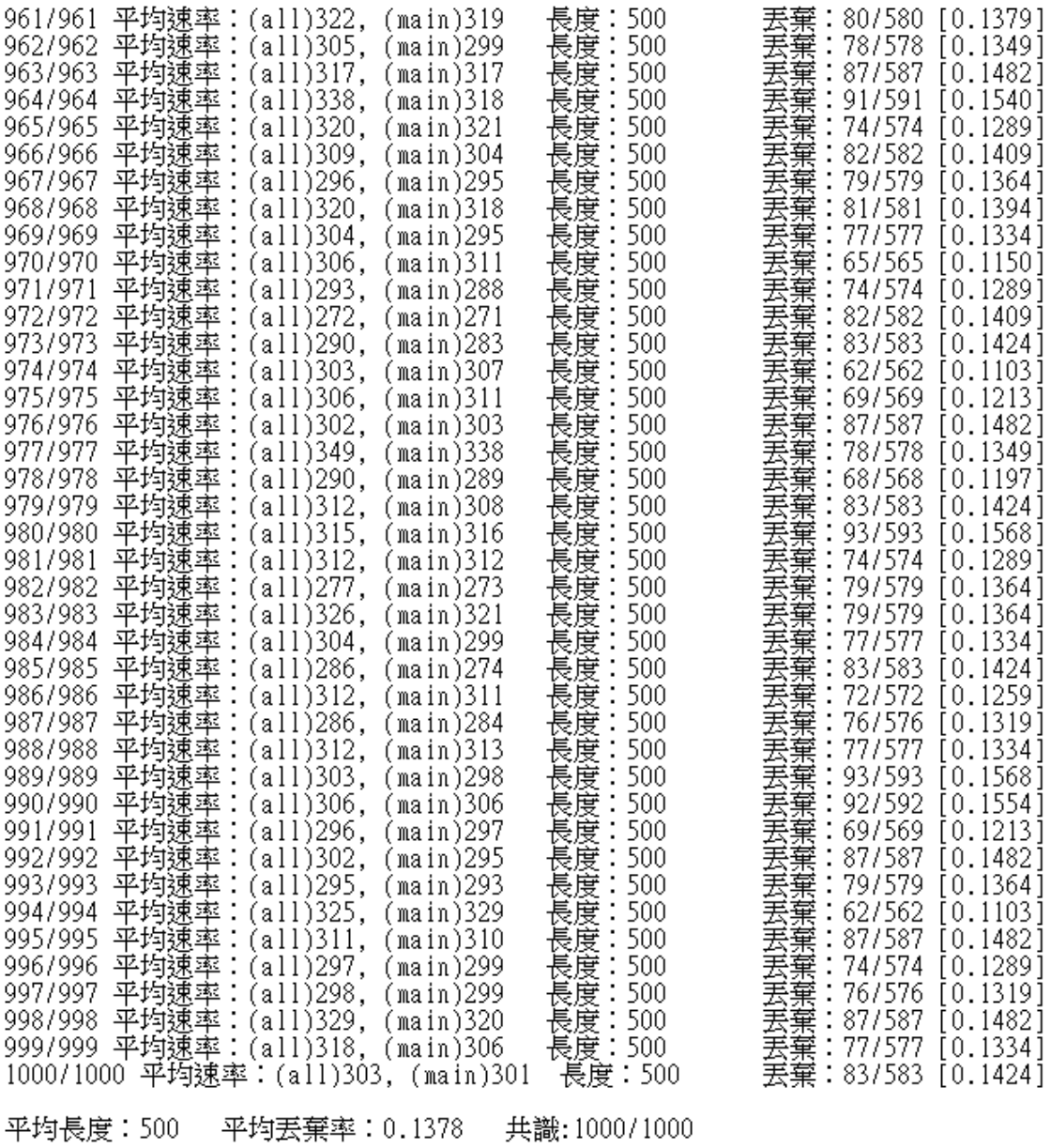
圖形 **9.** 產生節點間的傳輸延遲時間

此圖為我們若將節點數設為 *30* 個，產生 *900* 個節點距離，平均延遲時間平均值設為 *50* 加上誤差值，誤差值採用拉普拉斯分布 *L(0,2)*

我們的模擬中，節點數設為 200 個節點，所以會產生 40000 個節點距離，節點間的距離及方向不同，所需的傳輸時間也不同，也會根據情況調整拉普拉斯分布，產生出不同情況，節點間所需要的傳輸延遲時間。

共識次數	產生速率	區塊鍊長度	丟棄率
1/1	平均速率：(all)326, (main)314	長度：500	丟棄：84/584 [0.1438]
2/2	平均速率：(all)327, (main)321	長度：500	丟棄：85/585 [0.1453]
3/3	平均速率：(all)288, (main)291	長度：500	丟棄：70/570 [0.1228]
4/4	平均速率：(all)321, (main)311	長度：500	丟棄：84/584 [0.1438]
5/5	平均速率：(all)305, (main)302	長度：500	丟棄：73/573 [0.1274]
6/6	平均速率：(all)310, (main)311	長度：500	丟棄：85/585 [0.1453]
7/7	平均速率：(all)309, (main)304	長度：500	丟棄：99/599 [0.1653]
8/8	平均速率：(all)281, (main)274	長度：500	丟棄：88/588 [0.1497]
9/9	平均速率：(all)284, (main)280	長度：500	丟棄：80/580 [0.1379]
10/10	平均速率：(all)301, (main)301	長度：500	丟棄：79/579 [0.1364]
11/11	平均速率：(all)297, (main)289	長度：500	丟棄：85/585 [0.1453]
12/12	平均速率：(all)338, (main)326	長度：500	丟棄：84/584 [0.1438]
13/13	平均速率：(all)289, (main)292	長度：500	丟棄：92/592 [0.1554]
14/14	平均速率：(all)302, (main)300	長度：500	丟棄：95/595 [0.1597]
15/15	平均速率：(all)306, (main)299	長度：500	丟棄：92/592 [0.1554]
16/16	平均速率：(all)302, (main)305	長度：500	丟棄：88/588 [0.1497]
17/17	平均速率：(all)317, (main)309	長度：500	丟棄：78/578 [0.1349]
18/18	平均速率：(all)302, (main)298	長度：500	丟棄：99/599 [0.1653]
19/19	平均速率：(all)307, (main)305	長度：500	丟棄：87/587 [0.1482]
20/20	平均速率：(all)295, (main)295	長度：500	丟棄：76/576 [0.1319]
21/21	平均速率：(all)299, (main)304	長度：500	丟棄：72/572 [0.1259]
22/22	平均速率：(all)320, (main)316	長度：500	丟棄：66/566 [0.1166]
23/23	平均速率：(all)303, (main)294	長度：500	丟棄：84/584 [0.1438]
24/24	平均速率：(all)294, (main)287	長度：500	丟棄：89/589 [0.1511]
25/25	平均速率：(all)296, (main)295	長度：500	丟棄：77/577 [0.1334]
26/26	平均速率：(all)315, (main)306	長度：500	丟棄：75/575 [0.1304]
27/27	平均速率：(all)317, (main)309	長度：500	丟棄：80/580 [0.1379]
28/28	平均速率：(all)294, (main)285	長度：500	丟棄：66/566 [0.1166]
29/29	平均速率：(all)302, (main)292	長度：500	丟棄：71/571 [0.1243]
30/30	平均速率：(all)315, (main)309	長度：500	丟棄：88/588 [0.1497]
31/31	平均速率：(all)312, (main)312	長度：500	丟棄：71/571 [0.1243]
32/32	平均速率：(all)326, (main)317	長度：500	丟棄：85/585 [0.1453]
33/33	平均速率：(all)311, (main)307	長度：500	丟棄：90/590 [0.1525]
34/34	平均速率：(all)296, (main)288	長度：500	丟棄：70/570 [0.1228]
35/35	平均速率：(all)321, (main)314	長度：500	丟棄：103/603 [0.1708]
36/36	平均速率：(all)310, (main)307	長度：500	丟棄：74/574 [0.1289]
37/37	平均速率：(all)307, (main)298	長度：500	丟棄：76/576 [0.1319]
38/38	平均速率：(all)314, (main)310	長度：500	丟棄：84/584 [0.1438]
39/39	平均速率：(all)293, (main)294	長度：500	丟棄：104/604 [0.1722]
40/40	平均速率：(all)281, (main)290	長度：500	丟棄：81/581 [0.1394]
41/41	平均速率：(all)308, (main)307	長度：500	丟棄：73/573 [0.1274]
42/42	平均速率：(all)296, (main)291	長度：500	丟棄：80/580 [0.1379]
43/43	平均速率：(all)307, (main)308	長度：500	丟棄：90/590 [0.1525]
44/44	平均速率：(all)292, (main)281	長度：500	丟棄：78/578 [0.1349]
45/45	平均速率：(all)329, (main)327	長度：500	丟棄：67/567 [0.1182]
46/46	平均速率：(all)289, (main)279	長度：500	丟棄：84/584 [0.1438]
47/47	平均速率：(all)302, (main)291	長度：500	丟棄：74/574 [0.1289]
48/48	平均速率：(all)318, (main)310	長度：500	丟棄：66/566 [0.1166]
49/49	平均速率：(all)324, (main)316	長度：500	丟棄：83/583 [0.1424]
50/50	平均速率：(all)340, (main)330	長度：500	丟棄：78/578 [0.1349]
51/51	平均速率：(all)312, (main)304	長度：500	丟棄：79/579 [0.1364]
52/52	平均速率：(all)312, (main)301	長度：500	丟棄：84/584 [0.1438]
53/53	平均速率：(all)302, (main)294	長度：500	丟棄：80/580 [0.1379]
54/54	平均速率：(all)289, (main)277	長度：500	丟棄：71/571 [0.1243]
55/55	平均速率：(all)323, (main)324	長度：500	丟棄：77/577 [0.1334]
56/56	平均速率：(all)277, (main)276	長度：500	丟棄：89/589 [0.1511]
57/57	平均速率：(all)298, (main)301	長度：500	丟棄：87/587 [0.1482]
58/58	平均速率：(all)302, (main)292	長度：500	丟棄：83/583 [0.1424]
59/59	平均速率：(all)311, (main)314	長度：500	丟棄：82/582 [0.1409]

61/61 平均速率：(all)286, (main)285 長度：500 丟棄：83/583 [0.1424]
62/62 平均速率：(all)291, (main)286 長度：500 丟棄：67/567 [0.1182]
63/63 平均速率：(all)296, (main)292 長度：500 丟棄：76/576 [0.1319]
64/64 平均速率：(all)294, (main)289 長度：500 丟棄：86/586 [0.1468]
65/65 平均速率：(all)324, (main)313 長度：500 丟棄：81/581 [0.1394]
66/66 平均速率：(all)295, (main)293 長度：500 丟棄：81/581 [0.1394]
67/67 平均速率：(all)295, (main)296 長度：500 丟棄：78/578 [0.1349]
68/68 平均速率：(all)297, (main)288 長度：500 丟棄：86/586 [0.1468]
69/69 平均速率：(all)284, (main)278 長度：500 丟棄：69/569 [0.1213]
70/70 平均速率：(all)284, (main)285 長度：500 丟棄：69/569 [0.1213]
71/71 平均速率：(all)265, (main)266 長度：500 丟棄：87/587 [0.1482]
72/72 平均速率：(all)283, (main)279 長度：500 丟棄：86/586 [0.1468]
73/73 平均速率：(all)314, (main)304 長度：500 丟棄：71/571 [0.1243]
74/74 平均速率：(all)289, (main)286 長度：500 丟棄：85/585 [0.1453]
75/75 平均速率：(all)284, (main)282 長度：500 丟棄：78/578 [0.1349]
76/76 平均速率：(all)293, (main)296 長度：500 丟棄：68/568 [0.1197]
77/77 平均速率：(all)277, (main)280 長度：500 丟棄：68/568 [0.1197]
78/78 平均速率：(all)310, (main)310 長度：500 丟棄：92/592 [0.1554]
79/79 平均速率：(all)289, (main)287 長度：500 丟棄：91/591 [0.1540]
80/80 平均速率：(all)296, (main)285 長度：500 丟棄：77/577 [0.1334]
81/81 平均速率：(all)295, (main)290 長度：500 丟棄：76/576 [0.1319]
82/82 平均速率：(all)293, (main)291 長度：500 丟棄：79/579 [0.1364]
83/83 平均速率：(all)301, (main)292 長度：500 丟棄：80/580 [0.1379]
84/84 平均速率：(all)319, (main)317 長度：500 丟棄：78/578 [0.1349]
85/85 平均速率：(all)284, (main)284 長度：500 丟棄：82/582 [0.1409]
86/86 平均速率：(all)296, (main)285 長度：500 丟棄：87/587 [0.1482]
87/87 平均速率：(all)312, (main)314 長度：500 丟棄：81/581 [0.1394]
88/88 平均速率：(all)296, (main)286 長度：500 丟棄：94/594 [0.1582]
89/89 平均速率：(all)283, (main)278 長度：500 丟棄：84/584 [0.1438]
90/90 平均速率：(all)309, (main)308 長度：500 丟棄：72/572 [0.1259]
91/91 平均速率：(all)269, (main)265 長度：500 丟棄：96/596 [0.1611]
92/92 平均速率：(all)282, (main)284 長度：500 丟棄：95/595 [0.1597]
93/93 平均速率：(all)328, (main)328 長度：500 丟棄：78/578 [0.1349]
94/94 平均速率：(all)303, (main)298 長度：500 丟棄：84/584 [0.1438]
95/95 平均速率：(all)305, (main)305 長度：500 丟棄：75/575 [0.1304]
96/96 平均速率：(all)275, (main)277 長度：500 丟棄：91/591 [0.1540]
97/97 平均速率：(all)296, (main)291 長度：500 丟棄：85/585 [0.1453]
98/98 平均速率：(all)334, (main)331 長度：500 丟棄：84/584 [0.1438]
99/99 平均速率：(all)321, (main)318 長度：500 丟棄：90/590 [0.1525]
100/100 平均速率：(all)333, (main)307 長度：500 丟棄：81/581 [0.1394]
101/101 平均速率：(all)295, (main)295 長度：500 丟棄：72/572 [0.1259]
102/102 平均速率：(all)295, (main)295 長度：500 丟棄：80/580 [0.1379]
103/103 平均速率：(all)286, (main)289 長度：500 丟棄：76/576 [0.1319]
104/104 平均速率：(all)304, (main)303 長度：500 丟棄：78/578 [0.1349]
105/105 平均速率：(all)306, (main)304 長度：500 丟棄：67/567 [0.1182]
106/106 平均速率：(all)327, (main)315 長度：500 丟棄：85/585 [0.1453]
107/107 平均速率：(all)320, (main)323 長度：500 丟棄：77/577 [0.1334]
108/108 平均速率：(all)312, (main)310 長度：500 丟棄：91/591 [0.1540]
109/109 平均速率：(all)279, (main)279 長度：500 丟棄：72/572 [0.1259]
110/110 平均速率：(all)311, (main)300 長度：500 丟棄：82/582 [0.1409]
111/111 平均速率：(all)312, (main)300 長度：500 丟棄：82/582 [0.1409]
112/112 平均速率：(all)340, (main)329 長度：500 丟棄：92/592 [0.1554]
113/113 平均速率：(all)335, (main)325 長度：500 丟棄：74/574 [0.1289]
114/114 平均速率：(all)326, (main)320 長度：500 丟棄：63/563 [0.1119]
115/115 平均速率：(all)302, (main)292 長度：500 丟棄：71/571 [0.1243]
116/116 平均速率：(all)310, (main)306 長度：500 丟棄：88/588 [0.1497]
117/117 平均速率：(all)310, (main)305 長度：500 丟棄：81/581 [0.1394]
118/118 平均速率：(all)279, (main)282 長度：500 丟棄：79/579 [0.1364]
119/119 平均速率：(all)309, (main)302 長度：500 丟棄：69/569 [0.1213]
120/120 平均速率：(all)301, (main)292 長度：500 丟棄：86/586 [0.1468]



圖形 **10.** 數值由左至右分別為*:* 達成共識次數*/*總次數　產生速率*:* 所有區塊*,* 被 *valid* 的區塊　區塊鍊長度　丟棄區塊*/*總區塊 *[*丟棄率*]*

此圖為模擬比特幣共識演算法區塊產生速率*:300* 秒，延遲時間*:50+ L(0,2)*，產生區塊鍊長度為 *500*，測試 *1000* 次的部分數據結果。

從上圖實驗結果來看，平均丟棄率為 *1.378%*，測試 *1000* 次後，*1000* 次都達成了共識，所以共識機率為 *100%*。下面我們將主要紀錄丟棄率及共識機率去做分析與比較。

4.1 比特幣共識演算法

表 1 為比特幣共識演算法的模擬結果，我們將平均延遲時間固定為 50 加上誤差值，誤差值採用拉普拉斯分布 L(0,2)，網路延遲很平均情況下，測試不同產生速率，各種比值的共識機率，都能達到共識 100%。每次測試區塊鍊長度為 500，測試 1000 次，紀錄達成共識的次數。

產生速率 (s)	延遲時間 (s)	比值	共識機率	丟棄率
300	50+ L(0,2)	6	100%	1.401%
250	50+ L(0,2)	5	100%	1.677%
200	50+ L(0,2)	4	100%	1.963%
150	50+ L(0,2)	3	100%	2.496%
100	50+ L(0,2)	2	100%	3.294%
50	50+ L(0,2)	1	100%	4.952%

表格 **1.** 比特幣共識演算法 *(每次測試區塊鍊長度為 500，測試 1000 次)*

根據測試區塊鍊長度的不同，有可能會影響到共識機率，上表 1 為每次測試區塊鍊長度為 500，測試 1000 次的情況，下圖為我們將區塊鍊長度增加為 2000 個區塊，測試 500 次，也能達到共識 100%。

產生速率 (s)	延遲時間 (s)	比值	共識機率	丟棄率
300	50+ L(0,2)	6	100%	1.422%
250	50+ L(0,2)	5	100%	1.647%
200	50+ L(0,2)	4	100%	2.059%
150	50+ L(0,2)	3	100%	2.511%
100	50+ L(0,2)	2	100%	3.331%
50	50+ L(0,2)	1	100%	4.986%

表格 **2.** 比特幣共識演算法 *(*每次測試區塊鍊長度為 *2000*，測試 *500* 次*)*

表 2 顯示了不同比值，區塊鍊長度為 2000 的情況，因為比值越小越有可能產生不一致，所以我們選擇最小比值 1，產生速率 50 秒，延遲時間 50+ L(0,2) 的條件下，調整每次所產生的區塊鍊長度。從表 3 得到我們將區塊鍊長度增加，分別為 500、1000、2000、4000 和 6000 的條件下，比值 1，也沒有不共識的情況發生。

產生速率 (s)	延遲時間 (s)	比值	共識機率	區塊鍊長度
50	50+ L(0,2)	1	100%	500
50	50+ L(0,2)	1	100%	1000
50	50+ L(0,2)	1	100%	2000
50	50+ L(0,2)	1	100%	4000
50	50+ L(0,2)	1	100%	6000

表格 **3.** 比特幣共識演算法 *(*比值為 *1*，測試 *500* 次*)*

表 4 是針對比值 6、4、2 的情況，我們調整誤差值拉普拉斯分布，讓網路傳輸延遲時間的區間較廣也較不平均，能得到 99% ∼100% 的共識機率。

產生速率 (s)	延遲時間 (s)	延遲時間區間 (s)	比值	共識機率	丟棄率
300	50+ L(0,5)	0∼81	6	100%	1.408%
300	50+ L(0,10)	0∼110	6	100%	1.416%
300	50+ L(0,15)	0∼149	6	100%	1.411%
300	50+ L(0,20)	0∼232	6	100%	1.420%
200	50+ L(0,5)	0∼86	4	100%	1.981%
200	50+ L(0,10)	0∼114	4	100%	1.968%
200	50+ L(0,15)	0∼160	4	100%	1.988%
200	50+ L(0,20)	0∼228	4	100%	2.004%
100	50+ L(0,5)	0∼83	2	100%	3.302%
100	50+ L(0,10)	0∼133	2	100%	3.249%
100	50+ L(0,15)	0∼158	2	99.5%	3.228%
100	50+ L(0,20)	0∼221	2	99%	3.259%

表格 **4.** 比特幣共識演算法 *(*每次測試區塊鍊長度為 *500*，測試 *1000* 次*)*

由表 1 和表 2 可知，產生速率不同而延遲時間很平均的情況下，比特幣共識演算法能達到 100% 的共識，以及產生速率不同而延遲時間分布很不平均的情況下，也都能得到 99% ∼100% 的共識機率，比特幣共識演算法很穩定且合理。

4.2 乙太坊共識演算法

表格 3 為乙太坊共識演算法的模擬結果，我們將平均延遲時間固定為 50 加上誤差值，誤差值採用拉普拉斯分布 L(0,2)，網路延遲很平均情況下，測試不同產生速率，各種比值的共識機率，比值為 6 和 5 時，能達到共識 100%，比值為 4 時，共識機率會開始往下降，比值為 1 時，共識機率只剩下 29.5%。

產生速率 (s)	延遲時間 (s)	比值	共識機率	丟棄率
300	50+ L(0,2)	6	100%	1.383%
250	50+ L(0,2)	5	100%	1.674%
200	50+ L(0,2)	4	99%	1.946%
150	50+ L(0,2)	3	98.5%	2.436%
100	50+ L(0,2)	2	89.3%	3.230%
50	50+ L(0,2)	1	29.5%	4.722%

表格 **5.** 乙太坊共識演算法 *(*每次測試區塊鍊長度為 *500*，測試 *1000* 次*)*

表格 4 是針對比值 6、4、2 的情況，我們調整誤差值拉普拉斯分布，讓網路傳輸延遲時間的區間較廣也較不平均，從實驗結果來看輸延遲時間的區間差異越大，乙太坊共識演算法，也越容易產生不一致。

比值為 6 時，共識機率能維持在 90.5% 以上，比值為 4 和 3 時，共識機率會分別下降為 74.5% ∼95.1% 和 12.2% 63.8%

產生速率 (s)	延遲時間 (s)	延遲時間區間 (s)	比值	共識機率	丟棄率
300	50+ L(0,5)	0∼78	6	100%	1.386%
300	50+ L(0,10)	0∼113	6	95.5%	1.418%
300	50+ L(0,15)	0∼160	6	92.7%	1.431%
300	50+ L(0,20)	0∼228	6	90.5%	1.405%
200	50+ L(0,5)	0∼85	4	95.1%	2.022%
200	50+ L(0,10)	0∼108	4	86.5%	2.006%
200	50+ L(0,15)	0∼167	4	85.0%	2.010%
200	50+ L(0,20)	0∼231	4	74.5%	2.034%
100	50+ L(0,5)	0∼83	2	63.8%	3.218%
100	50+ L(0,10)	0∼121	2	23.5%	3.267%
100	50+ L(0,15)	0∼151	2	13.6%	3.319%
100	50+ L(0,20)	0∼225	2	12.2%	3.320%

表格 **6.** 乙太坊共識演算法 *(*每次測試區塊鍊長度為 *500*，測試 *1000* 次*)*

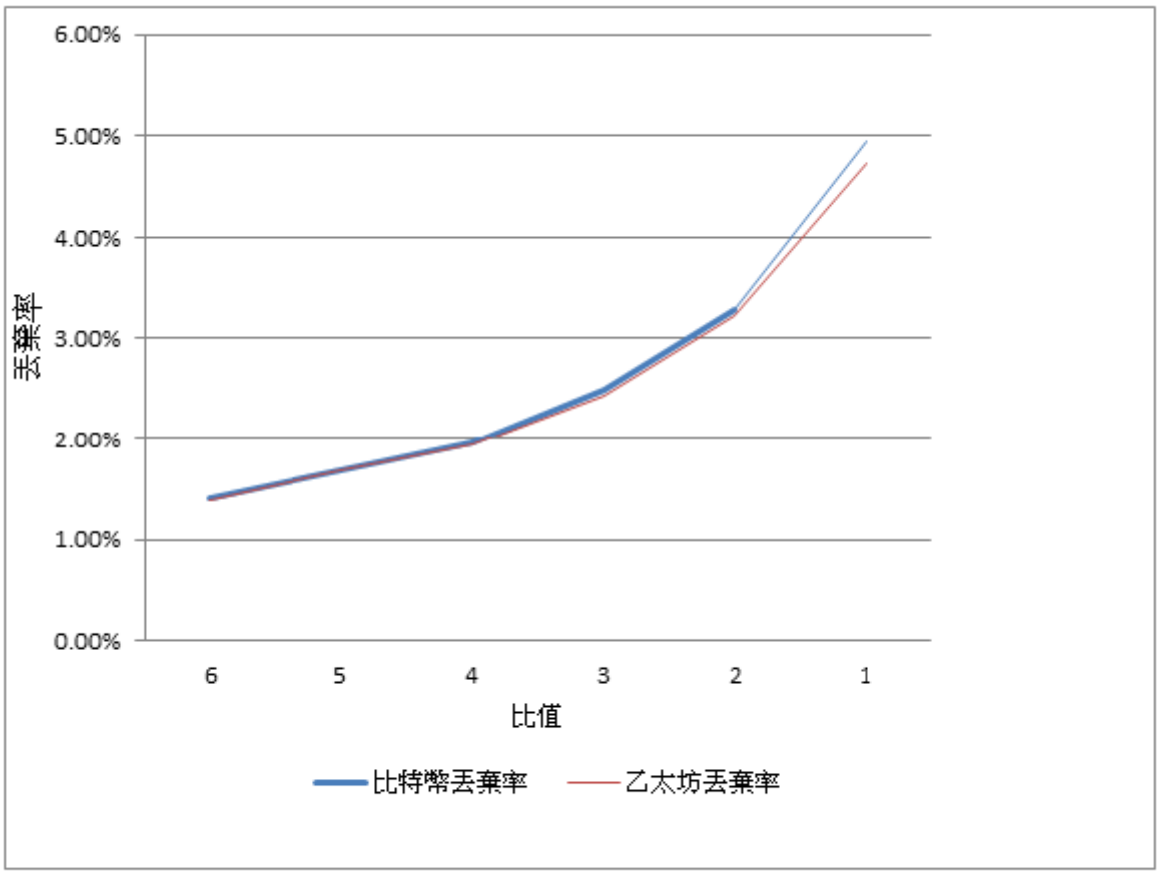
由表格 3 和表格 4 來看，產生速率不同而延遲時間很平均的情況下，乙太坊共識演算法的共識機率會隨著產生速率下降，以及產生速率不同而延遲時間分布很不平均的情況下，共識機率也會下降，乙太坊共識演算法較不穩定。

4.3 比較與分析

我們分別測試了比特幣和乙太坊共識演算法，在不同情況下的共識機率即丟棄率，此節，我們將比較兩種共識演算法的丟棄率及共識機率。表格 7 為比特幣和乙太坊共識演算法在延遲時間為 50+ L(0,2)，產生速率為 50~300，比值為 1~6 的條件下，比較兩種方法的丟棄率。

產生速率	延遲時間	比值	比特幣丟棄率	乙太坊丟棄率
300	50+ L(0,2)	6	1.401%	1.383%
250	50+ L(0,2)	5	1.677%	1.674%
200	50+ L(0,2)	4	1.963%	1.946%
150	50+ L(0,2)	3	2.496%	2.436%
100	50+ L(0,2)	2	3.294%	3.230%
50	50+ L(0,2)	1	4.952%	4.722%

表格 **7.** 比特幣和乙太坊共識演算法丟棄率比較表 *(*每次測試區塊鍊長度為 *500*，測試 *1000* 次*)*

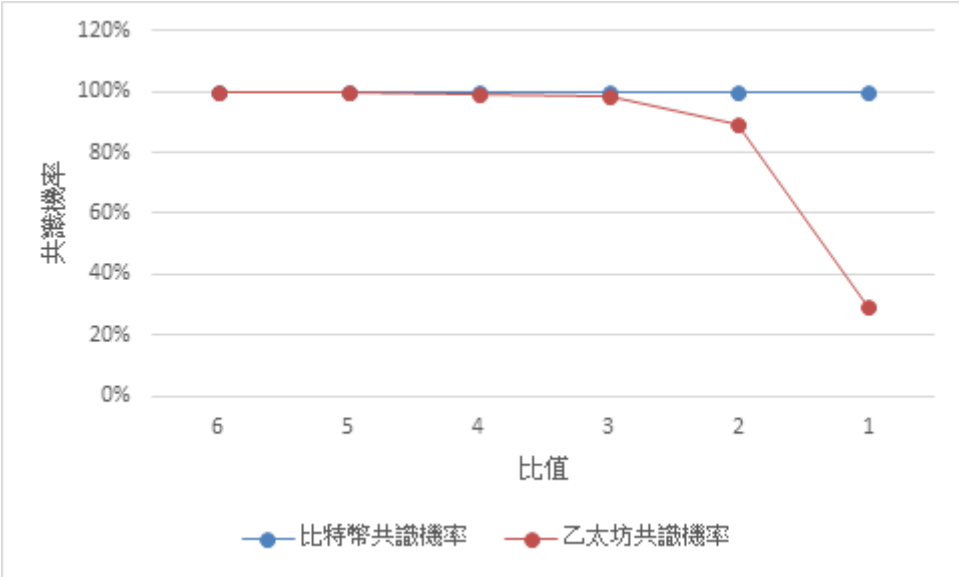


圖形 **11.** 丟棄率比較圖

由表格 7 及圖 11 可知，比特幣和乙太坊共識演算法的區塊丟棄率十分接近，但從表 7 來看，乙太坊的丟棄率數值還是略小於比特幣的。因為乙太坊共識演算法的樹結構只能允許最多 3 個分支，而比特幣共識演算法則沒有這樣的限制，可能會有多餘 3 個以上的分岔區塊產生，所以會有較多的區塊會被丟棄。

產生速率	延遲時間	比值	比特幣共識機率	乙太坊共識機率
300	50+ L(0,2)	6	100%	100%
250	50+ L(0,2)	5	100%	100%
200	50+ L(0,2)	4	100%	99%
150	50+ L(0,2)	3	100%	98.5%
100	50+ L(0,2)	2	100%	89.3 %
50	50+ L(0,2)	1	100%	29.5%

表格 **8.** 比特幣和乙太坊共識演算法共識機率比較表 *(*每次測試區塊鍊長度為 *500*，測試 *1000* 次*)*



圖形 **12.** 共識機率比較圖

表 8 及圖 12 可知，當區塊產生平均速率與網路延遲的比值下降時，比特幣的共識機率還是十分穩定，而乙太坊會有下降的趨勢。

從我們的實驗結果來看，乙太坊的區塊丟棄率較小於比特幣的區塊丟棄率，應該較容易達成共識，但是從圖 12 來看，比特幣共識演算法卻比乙太坊共識演算法所達成的共識還要高，也就是說比特幣共識演算法比乙太坊共識演算法較為穩定。

Chapter 5

結論與未來展望

在本論文中，我們提到每個節點都會挖礦，若有新的區塊產生，就會將新的區塊廣播出去，節點們收到新區塊會串聯起來形成區塊鏈。另外，我們也模擬了點對點分散式網路，讓每個節點間都有不同的傳輸延遲時間，所以節點收到新區塊的時間不一定相同，也有可能會產生分岔的情形。

我們運用比特幣和乙太坊的共識演算法來判斷區塊是否能被確認，所有節點是否達成一致。比特幣共識演算法採取選擇最長鏈，經過 6 個區塊產生後，區塊將會被固定下來。乙太坊共識演算法使用樹結構，樹的每個節點最多只能有三個分支，若是 degree 大於 11，該區塊就會被確認。

我們分析兩種方法的結果與比較，在產生速率不同而延遲時間很平均的情況下，以及產生速率不同而延遲時間分布很不平均的情況下，比特幣共識演算法能得到 99% 以上的共識機率，而乙太坊共識演算法就無法達到很穩定的共識機率。比較兩種共識方法的區塊丟棄率及共識機率後，比特幣共識演算法會比乙太坊共識演算法更穩定合理。

Bibliography

[1] S. Nakamoto, *Bitcoin : A Peer-to-Peer Electronic Cash System.* www.bitcoin.org, 2008.

[2] I.-C. Lin1 and T.-C. Liao, *A Survey of Blockchain Security Issues and Challenges.* International Journal of Network Security, Vol.19, 2017.

[3] B. R. Möser Malte, *Trends, Tips, Tolls: A Longitudinal Study of Bitcoin Transaction Fees.* 19th International Conference on Financial Cryptography and Data Security, 2015.

[4] G. Wood, *ETHEREUM: A secure decentralised generalised transaction ledger.* Yellow paper, 2015.

[5] C. Decker and R. Wattenhofert, *Information propagation in the Bitcoin network.* 13-th IEEE International Conference on Peer-to-Peer Computing, 2013.

[6] V. G. C. Natoli, *The Balance Attack Against Proof-Of-Work Blockchains: The R3 Testbed as an Example.* Tech. Rep., 2016.

[7] E. S. R. v. R. I. Eyal, A.E. Gencer, "Bitcoin-ng: A scalable blockchain protocol," in *13th USENIX Symposium on Networked Systems Design and Implementation*, p. 45–59, NSDI, 2016.

[8] V. G. C. Natoli, *The blockchain anomaly.* Proceedings of the 15th IEEE International Symposium on Network Computing and Applications, 2016.

[9] V. Gramoli, *From blockchain consensus back to Byzantine consensus.* Future Generation Computer Systems, 2017.

[10] A. Z. Y. Sompolinsky, "Secure high-rate transaction processing in bitcoin," in *Financial Cryptography and Data Security - 19th International Conference*, p. 507 – 527, FC 2015, 2015.