國立政治大學資訊科學系
Department of Computer Science
National Chengchi University

碩士論文

Master's Thesis

# 改良 Kademlia 在高度網路擾動下的搜尋成功率

# Improving Search Success Rate of Kademlia Under High Churn

研 究 生：簡豪廷

指導教授：蔡子傑

中華民國一一零年一月

January 2021

DOI:10.6814/NCCU202100314

# 改良 Kademlia 在高度網路擾動下的搜尋成功率

# Improving Search Success Rate of Kademlia Under High Churn

研 究 生：簡豪廷　　Student：Hao-Ting Jian

指導教授：蔡子傑　　Advisor：Tzu-Chieh Tsai

國立政治大學

資訊科學系

碩士論文

A Thesis

submitted to Department of Computer Science

National Chengchi University

in partial fulfillment of the Requirements

for the degree of

Master

in

Computer Science

中華民國一一零年一月

January 2021

DOI:10.6814/NCCU202100314

# 致謝

能夠完成碩士學位對我而言意義重大，首先要感謝我的指導老師蔡子傑教授，能夠成為您的學生是件幸運的事情。蔡老師有耐心地引導我，指出研究上的盲點並適時給予建議，協助我完成研究順利取得碩士學位。

必須感謝實驗室的夥伴們。我的同儕-煙童與陳遠，兩位都是友善又可靠的朋友，與你們共處的時光相當快樂，因為有你們與我共同面對學業和研究上的挑戰，我才能順利走完這段旅程，取得學位之外，與你們相識是我在碩士期間的最大收穫。感謝辰瑋在美食沙漠中帶我找到一片綠洲，感謝宗霖在我寫程式至夜晚時仍在周遭陪伴。

最後，我要感謝我的父母，你們以行動支持我在學業上的任何決定。當我提出要攻讀資訊領域的碩士學位時，你們鼓勵我去取得學位。在我取得政大資科所的入學資格後，你們讓我能夠無後顧之憂的完成碩士學業。我能夠在今日取得學位，或是在未來取得更高的成就，都是因為你們一路支持我。



DOI:10.6814/NCCU202100314

# 摘要

Kademlia 是一種被廣泛使用的 P2P 分散式雜湊表技術，在高度網路擾動環境中仍存在資料搜尋失敗的問題。本文以模擬的方式對以 Kademlia 運作的 P2P 系統進行分析，整理出在 Kademlia 技術下資料搜尋失敗的原因，並提出改良方法以少量成本增長提升資料搜尋成功率。

高度網路擾動會造成部分節點的路由表失效，同時使部份節點無法有效認識周遭節點。前者會導致儲存資料後資料難以被搜尋，或者搜尋資料時無法取得資料；後者會使資料儲存者與搜尋者找到的節點不一致，導致搜尋失敗。本文提出以紀錄長時間上線的節點，在路由表失效時向其連線以處理路由表失效的問題；及以向遠距離的節點詢問資料處理節點無法有效認識周遭節點的問題。實驗結果證實實施兩者後能使得資料搜尋成功率提升。

相較於傳統上以超級節點支援的 P2P 網路，我們的方法能以較低的通訊成本達到相當的資料搜尋成功率。

關鍵字: Kademlia、點對點、網路擾動、分散式雜湊表

DOI:10.6814/NCCU202100314

# Abstract

Kademlia is a widely deployed P2P-Distributed Hash Table network, however there exists some search failure issue under high churn. This thesis analyzes the reasons for Kademlia search failure by simulating the Kademlia-based P2P system and proposed some methods to improve Kademlia search success rate with little extra cost.

High churn causes Kademlia Routing Table to be invalid, and in such case, nodes would have insufficient acknowledge of surrounding nodes. Invalid Routing Tables not only make data hard to be found but also directly lead to search failure. Insufficient acknowledge of surrounding nodes would make the traversing nodes and data nodes different and cause search failure. To this end, we suggest that nodes can connect with long-stay nodes to handle the invalid Routing Table invalid problem, and nodes can ask the far nodes for information to avoid the problem for insufficient acknowledge of surrounding nodes. Simulation experiments confirm that the search success rate of Kademlia increases when the above two enhancements are implemented.

Furthermore, we also compare the performance with traditional supernode supported P2P network. Simulation results also show that our method maintains the comparable search success rate but with less overhead.

Keywords: Kademlia、P2P、Churn、Distributed Hash Table



DOI:10.6814/NCCU202100314

# 目次



DOI:10.6814/NCCU202100314

# 圖次





DOI:10.6814/NCCU202100314

# 表次



DOI:10.6814/NCCU202100314

# 第一章 緒論

## 第一節　背景與動機

傳統網路服務依照主從式架構進行，由網路服務供應商的伺服器對用戶提供服務，這種網路架構需要網路服務供應商以高資本及人力資源來建立伺服器，以應付來至用戶的服務請求。有些網路服務供應商在建立伺服器時錯估了用戶數量及使用需求，在大量人群使用網路服務時伺服器無法負荷，導致服務中止，或是伺服器受到不肖人士的分散式阻斷服務攻擊時會被迫暫停服務。

有些大型公司在全球擁有無數個伺服器中心，能夠負擔極大量的用戶使用需求，並具備抵禦不肖份子惡意攻擊的能力，卻在提供服務時蒐集用戶的影音資料、言論與搜尋習慣等資料，進行資料分析後投入商業使用。蒐集用戶資料並進行分析，將用戶數據化後商業化的行為是否危害個人隱私受到廣泛討論。近年來個人的網路隱私意識抬頭，也有人質疑個人資訊是否應該在有效的合約之下被網路公司任意使用。

影音網站及社群網站在發展初期鼓勵用戶參與網路活動並發表個人言論，在人們習慣並且大量使用社群服務後，影音網站及社群網站卻以不明確的標準隱藏與服務平台意見相左的言論，同時將具有商業或政治目的文章及影音內容擺放到用戶面前。網路言論自由是否該被私人公司所控制，人們享受網路社群或影音服務時是否由私人公司決定我們的視聽內容都受到熱烈討論。

P2P 網路的分散式特性能夠避免網路服務有大量使用需求時或受到攻擊時整個系統失去服務能力的問題，個人資訊若以適當的加密方法保護並存在分散的網路節點中，也能避免個人資訊遭到網路公司不當使用的問題，分散式社群服務能夠避免言論被私人公司所限制，也不必由私人公司決定我們每天接收到的資訊。我們相信 P2P 網路若結合妥善的密碼學技術與應用領域後，將會有極大的發展可能，改變現有的網路應用模式。



DOI:10.6814/NCCU202100314

近年來區塊鏈技術流行，比特幣以及以太坊被廣泛認識並且應用，說明人們接受並需要此類型去中心化、安全可靠的 P2P 網路系統。區塊鏈的去中心化及不可竄改的特性有極高的應用價值，如金融領域、合約及證書紀錄或其他商業應用。但區塊鏈技術亦有其缺點：每份資料都將存在所有節點之中，需要耗費大量空間做儲存；需要高運算效能設備做運算，對運算能力的需求隨著整體區塊鏈擴大而增加；交易完成後需要數十秒的時間更新資訊。儲存空間需求過大導致一般民眾不願提供資源成為節點，而運算效能的需求只有少數計算中心能夠滿足，同時也有過度浪費電力資源等問題，交易完成後更新資料的時間更成為商業應用的阻礙。

若能去除主從式架構與區塊鏈技術的缺點，建立一套完善的 P2P 系統，將對於網路應用範疇有所助益。在理想的 P2P 架構中，能避免單一或是少數伺服器遭受攻擊時，造成網路服務中斷，或是多數人的個人隱私遭到公司或是非法集團的濫用，同時用戶的言論不會輕易受到限制，也能夠決定自己的視聽內容，同時此系統又能夠免除區塊鏈需要極大量儲存空間以及運算力的缺點。為了發展出完善的 P2P 系統，我以 Kademlia(一種透過分散式雜湊表技術形成的 P2P 網路技術，各個節點依照一連串數字所形成的 ID 進行辨識，並由所有節點形成虛擬網，節點能夠提供資料存取服務。Petar & David,2002) 為研究對象，進行 P2P 技術的研究。

Kademlia 是世界上最常被使用的 P2P 技術，被許多 P2P 網路或應用系統當作用戶間通訊往來的主要運作架構。先前的研究指出以 Kademlia 作為基礎運作架構的 P2P 系統存在資料搜尋失敗的問題，我也對 Kademlia 是否會發生搜尋失敗的情況進行測試。測試時，由於資料備份數量較少以及 P2P 網路用戶時常連線與離線的特性，Kademlia 仍會發生無法順利存取資料的錯誤。因此我以改良 Kademlia 的搜尋成功率為目標進行此研究，若順利改良此 P2P 技術，將會對後人的 P2P 技術或者相關應用有所助益。



DOI:10.6814/NCCU202100314

## 第二節　研究目的

先前的 Kademlia 相關研究對於由於網路不通順、硬體效能不足及自私節點等導致的資料搜尋失敗原因有較充足的研究，並且提出適當的解決方案，但是對於搜尋時無法找到持有資料的節點所導致的資料搜尋失敗情境尚未提出完善的解決方案。我的研究將針對搜尋時無法找到持有資料的節點所導致的資料搜尋失敗情境進行研究，分析造成此現象的原因並提出對應的解決方案，將著重於改變 Routing Table 的演算法，藉此增強 Kademlia 的資料搜尋成功率。實驗以用 Python 撰寫的 Kademlia 模擬器進行模擬，在不同的節點連線與離線環境中進行測試，最後以資料搜尋成功率及網路通訊成本做為衡量指標，比較不同改良方法的成效。





DOI:10.6814/NCCU202100314

# 第二章 文獻探討

## 第一節　Kademlia

Kademlia 是一種透過分散式雜湊表實現的結構式對等網路技術，由 Petar Maymounkov 與 David Mazieres (2002)所提出，為世界上最常使用的 P2P 技術。Kademlia 系統中的用戶都被視為節點，節點以各自的 Routing Table 紀錄網路中的節點資訊，所有節點共同形成覆蓋網路，系統能提供儲存資料及取得資料的基本服務。每個節點以一組不重複的 160-bit key 作為 ID，ID 代表節點的地址，同時是儲存資料與取得資料時的搜尋目標。資料會經由 Hash Function 轉換為 Key 值，以遞迴搜尋的方式尋找最靠近 Key 的 K 個節點，被存入所搜尋到最靠近 key 的 K 個節點之中。節點同時具備用戶與伺服器的身分，可享受儲存資料與搜尋資料的服務，同時必須負擔系統的維護及提供資料儲存空間的工作，節點之間以 UDP 傳遞資訊。節點以 XOR 運算進行距離計算，同一節點與不同節點的距離不相同，此定位方式保證在搜尋過程中每次訊息往來都會得到更有用的結果，將搜尋資料的時間限制在 $\log_2 N$ 步之內(N 為系統的總結點數量)。使用平行化的搜尋方式，能夠減少節點失敗(節點無法回覆)時所造成的延遲。將儲存的資料備份在多個異地節點中，避免遭到惡意攻擊時失去服務能力，亦可在節點離線時，由其他節點保持網路服務能力。藉由 Re-publish 機制，使資料長期保存在網路中。Kademlia 藉由獨特的定位及搜尋方式保證系統的搜尋效率及容錯性，以異地備份資料確保系統在受攻擊或結點失效時的服務能力。



DOI:10.6814/NCCU202100314

## 一、 距離計算

節點之間以 XOR 運算來計算距離。距離的相對遠近會影響 Routing Table 內所存的節點資訊、每次詢問後的回傳資訊、遞迴搜尋是否終止以及資料的存放目標，距離為 Kademlia 技術的核心觀念，影響整個系統的運作模式。

以計算節點 00101…與 10010…(ID 長度皆為 160bit)的距離為例，兩者的距離以二進位表示為 10111…，共同的前綴數量為 0，若以指數表示此距離大於 $2^{159}$ 小於 $2^{160}$。

每個節點會依照距離將遇見的節點做分群，依照共同的前綴數量或距離分群。對於 00101...而言，會將遇見的節點分為:

1… (0 個共同前綴)/距離為[$2^{159}$,$2^{160}$)

01…(1 個共同前綴)/距離為[$2^{158}$,$2^{159}$)

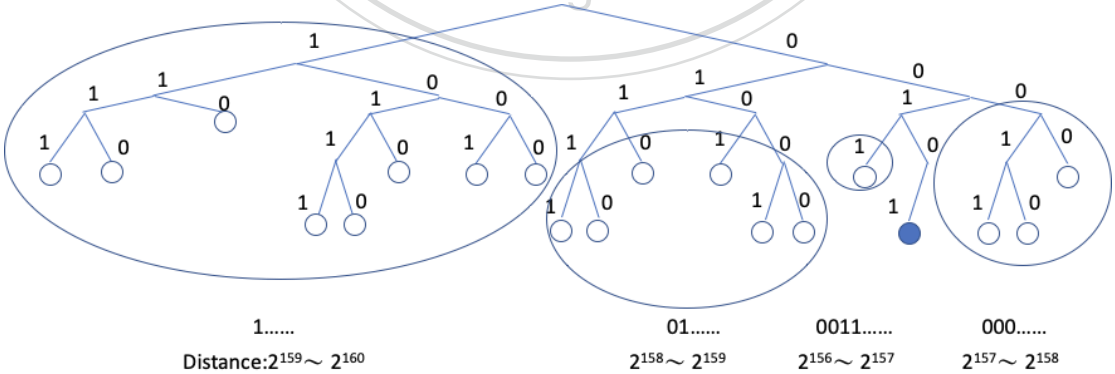
000… (2 個共同前綴)/距離為[$2^{157}$,$2^{158}$)

0010…(3 個共同前綴)/距離為[$2^{156}$,$2^{157}$)

……

依照前綴數量或距離將節點分類。

圖 2- 1 依照距離或共同前綴數量將結點分類




依照前綴數量或距離將節點後，會依照分類的結果將該節點的資訊存入 Routing Table 內對應的 K-bucket 中。



DOI:10.6814/NCCU202100314

## 二、 Routing Table 與 K-bucket

Routing Table 紀錄自己所接觸過的節點，Routing Table 會依照共同的前綴數量/距離作分群，每一群都被稱為 K-bucket，每個 K-bucket 最多存有 K 個節點的資訊。

節點將以<IP address, UDP port, Node ID>的形式紀錄。在隨後的表格中以 6bits，K=4 為例，在表格中僅顯示 Node ID 以節省空間。

| Table of node-000000 | | |
|---|---|---|
| **prefix** | **distance** | **nodes** |
| 0 | $2^5$～ $2^6$ | 100000,101001,111001,100110 |
| 1 | $2^4$～ $2^5$ | 01…, … |
| 2 | $2^3$～ $2^4$ | 001…, … |
| 3~6 | 0～ $2^3$ | 0001...,000000 |

表 2- 1 節點 0 的 Routing Table

Node-000000...以共同前綴數量/距離將認識的節點分類，將其資訊存入不同 K-bucket 中。

當節點產生時，Routing Table 中只有一組距離為 0～ $2^{160}$ 的 K-bucket，其中僅存有自己的節點資訊。

| Table of node-000000 | | |
|---|---|---|
| **prefix** | **distance** | **nodes** |
| 0 | 0～ $2^6$ | 000000 |

表 2- 2 當節點產生時的 Routing Table

與節點來往，並確認對方在線時，以 XOR 計算彼此間的共同前綴數量/距離，找出 Routing Table 中對應的 K-bucket，若對方的資訊已存在該 K-bucket 中，將其節點資訊移動至此 K-bucket 的尾端。



DOI:10.6814/NCCU202100314

若對方的資訊不存在 K-bucket 中，則分為以下三種情境:

1.K-bucket 未滿 K 個節點資訊，將新節點資訊加入 K-bucket 尾端。

| Table of node-000000 | | |
|---|---|---|
| **prefix** | **distance** | **nodes** |
| 0 | 0～ $2^6$ | 000000,101001,111001,000110 |

表 2- 3 將新節點資訊加入 K-bucket 尾端

2.K-bucket 已滿 K 個節點資訊，同時自己的節點資訊存在該 K-bucket 中。

將 K-bucket 依距離分裂為二，節點存入對應的 K-bucket 中。

例： 0～ $2^{160}$ 分裂為 $2^{159}$～ $2^{160}$ 與 0～ $2^{159}$。

| Table of node-000000 | | |
|---|---|---|
| **prefix** | **distance** | **nodes** |
| 0 | $2^5$～ $2^6$ | 101001,111001,100000 |
| 1 | 0～ $2^5$ | 000000,000110 |

表 2- 4 K-bucket 分裂為二

3. K-bucket 已滿，自己的節點資訊不存在該 K-bucket 中。

對 K-bucket 中的第一個節點送出 PING 指令。若該節點回覆，將其資訊轉移至該 K-bucket 尾端，放棄新節點資訊；未回覆，將該節點資訊移除，將新節點資訊加入 K-bucket 尾端。

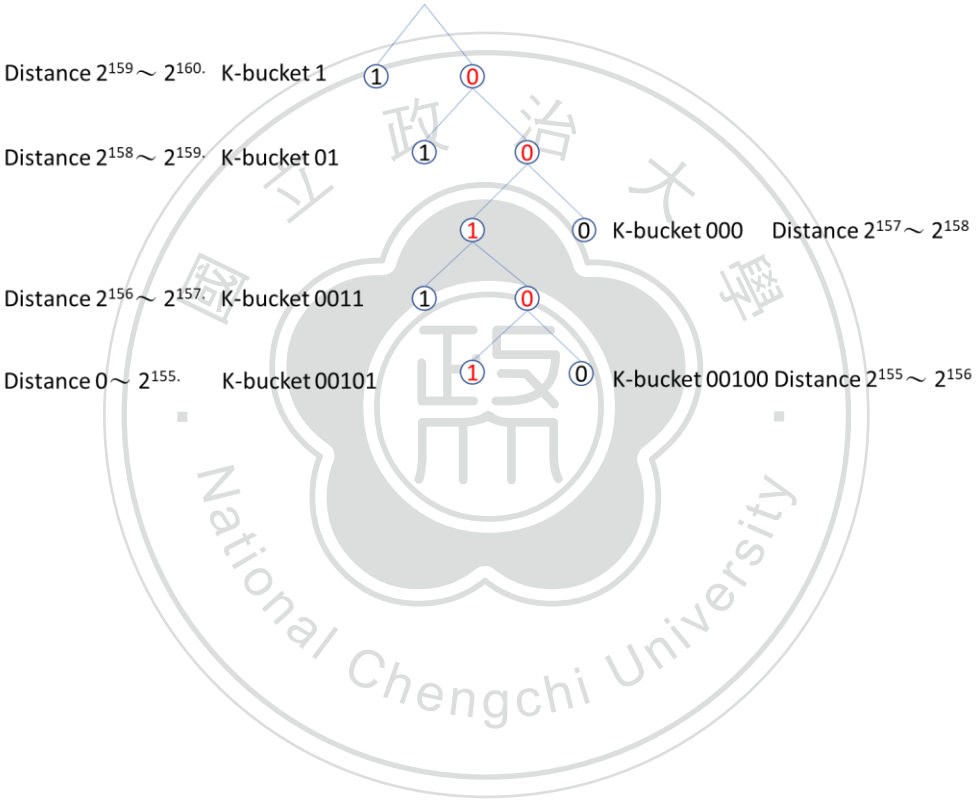
以二元樹來說明 Kademlia 的 Routing Table，在圖 2-2 中以節點 00101…為例，在二元樹的每一層中，與節點 00101…不同的分支都會被分類到同一個 K-bucket，1 開頭的節點會被分至同一個 K-bucket，其距離為[$2^{159}$,$2^{160}$)；01 開頭的節點會被分至同一個 K-bucket，其距離為[$2^{158}$,$2^{169}$)，當這些 K-bucket 內部已經滿



DOI:10.6814/NCCU202100314

K 個節點同時認識同一分類的新節點時，會將其中未回覆 PING 指令的節點剃除，將新節點加入 K-bucket 中。最底層包含節點 00101…的 K-bucket 內部已 K 個節點同時認識同一分類的新節點時，則會將此 K-bucket 一分為二，形成一個與 00101…不同分支的 K-bucket 與一個包含 00101…本身的 K-bucket。

圖 2- 2 以二元樹表示 Routing Table








DOI:10.6814/NCCU202100314

## 三、 RPCs 遠端控制指令

節點之間能夠傳遞四種指令，分別為 PING、STORE、FIND_NODE 以及 FIND_VALUE，以這四種指令完成 Kademlia 的所有功能。

PING

檢查節點是否在線。

STORE

將資料存入節點。收到指令的節點將資料存入客戶端。

FIND_NODE

向節點詢問對方的 Routing Table 中最靠近特定 ID 的 K 個節點資訊。收到指令的節點將回傳 Routing Table 中距離特定 ID 最近的 K 個節點資訊。

FIND_VALUE

進行與 FIND_NODE 相同的流程，此外會詢問節點中是否存有特定資料。收到指令的節點若持有資料便回傳資料，若無則回傳 Routing Table 中距離特定 ID 最近的 K 個節點資訊。

## 四、 Lookup

當一節點試圖找尋到距離某 ID 最近的 K 個節點時，其運作流程被稱為 Lookup。不論是將資料儲存至 Kademlia 系統中、在 Kademlia 系統尋找資料或單純尋找距離某 ID 最近的 K 個節點，都是藉由 Lookup 達成目的。



DOI:10.6814/NCCU202100314

Lookup 流程：

發起搜尋的節點(後稱搜尋者)會計算 Key 與自己 ID 間的同前綴數量/距離，將 Routing Table 中的所有節點資訊依照與 Key 同前綴數量多到少/距離近到遠整理為一列目標節點序列。從目標節點序列中找出最前端的 α 個節點，將 FIND_NODE 指令傳送給該 α 個節點，收到 FIND_NODE 指令的節點會在 Routing Table 中找出最靠近 Key 的 K 個節點資訊，並將最靠近 Key 的 K 個節點資訊回傳給搜尋者。搜尋者在收到回傳的節點資訊後，將這些節點依照與 Key 同前綴數量多到少/距離近到遠整理至目標節點序列中，從目標節點序列的最前端開始找出 α 個尚未訪問過的節點，將 FIND_NODE 指令傳送給該 α 個節點，一直重複整理目標節點序列、傳送 FIND_NODE 等流程，直到滿足終止條件。搜尋節點或儲存資料時，終止條件為 K 個最靠近 Key 的節點皆收到 FIND_NODE 指令且回覆搜尋者。

圖 2- 3 Lookup 示意圖


Lookup 流程示範:

假設 ID 長度為 6bits，範圍自 000000~111111，K=4，α=2。由節點 63 將 Data 存到系統中。以 Hash Function 處理資料，由 Hash(Data)得到 Key 值 000000，由節點 63 將 Data 存入最靠近節點 0 的 K 個節點中。

| Table of node-111111 | | |
|---|---|---|
| **prefix** | **distance** | **nodes** |
| 0 | $2^5$～ $2^6$ | 000000,001001,011001,000110 |
| 1 | $2^4$～ $2^5$ | 10… |
| 2 | $2^3$～ $2^4$ | 110… |
| 3~6 | 0～ $2^3$ | 111111, 1110… |

表 2- 5 節點 63 的 Routing Table



DOI:10.6814/NCCU202100314

節點 63 從 Routing Table 中依照與 Key 值 000000 的距離近到遠/共同前綴數量多到少整理出目標節點序列，目標節點序列:0,6,9,22, …。節點 63 向 α 個最靠近 Key 的節點傳送出 FIND_NODE 指令，節點 63 傳送出 FIND_NODE 指令給節點 0 與節點 6。

節點 0 不在線。

| Table of node-000110 | | |
|---|---|---|
| **prefix** | **distance** | **nodes** |
| 0 | $2^5$～ $2^6$ | 1… |
| 1 | $2^4$～ $2^5$ | 01… |
| 2 | $2^3$～ $2^4$ | 001010,001101,001110,001001 |
| 3~6 | 0～ $2^3$ | 000110,000010,000001 |

表 2- 6 節點 6 的 Routing Table

節點 6 計算出與 Key 的距離/共同前綴數量，藉此找出對應的的 K-bucket，從 K-bucket 中取出最靠近 Key 的 K 個節點。節點 6 回傳節點 1,2,6,9 的資訊給節點 63。

節點 63 收到資訊後，將目標節點序列重新整理後為:1,2,6,9,22…。節點 63 從目標節點序列中找出尚未詢問過的 α 個節點，找到節點 1,2。節點 63 傳送出 FIND_NODE 指令給節點 1 與節點 2。

節點 1 計算出與 Key 的距離/共同前綴數量，藉此找出對應的的 K-bucket，從 K-bucket 中取出最靠近 Key 的 K 個節點。節點 1 回傳節點 1,2,5,9 的資訊給節點 63。



DOI:10.6814/NCCU202100314

| Table of node-000001 | | |
|---|---|---|
| **prefix** | **distance** | **nodes** |
| 0 | $2^5$～ $2^6$ | 1… |
| 1 | $2^4$～ $2^5$ | 01… |
| 2 | $2^3$～ $2^4$ | 001101,001111,001110,001001 |
| 3~6 | 0～ $2^3$ | 000001,000010,000101 |

表 2- 7 節點 1 的 Routing Table

節點 2 計算出與 Key 的距離/共同前綴數量，藉此找出對應的的 K-bucket，從 K-bucket 中取出最靠近 Key 的 K 個節點。節點 2 回傳節點 1,2,9,11 的資訊給節點 63。

| Table of node-000010 | | |
|---|---|---|
| **prefix** | **distance** | **nodes** |
| 0 | $2^5$～ $2^6$ | 1… |
| 1 | $2^4$～ $2^5$ | 01… |
| 2 | $2^3$～ $2^4$ | 001011,001101,001110,001001 |
| 3~6 | 0～ $2^3$ | 000010,000001 |

表 2- 8 節點 2 的 Routing Table

節點 63 收到資訊後，將目標節點序列重新整理後為: 1,2,5,6,9,22…。節點 63 從目標節點序列中找出尚未詢問過的 α 個節點，找到節點 5。節點 63 傳送出 FIND_NODE 指令給節點 5。

節點 5 計算出與 Key 的距離/共同前綴數量，藉此找出對應的的 K-bucket，從 K-bucket 中取出最靠近 Key 的 K 個節點。節點 5 回傳節點 2,5,6,9 的資訊給節點 63。



DOI:10.6814/NCCU202100314

| Table of node-000101 | | |
|---|---|---|
| **prefix** | **distance** | **nodes** |
| 0 | $2^5$～ $2^6$ | 1… |
| 1 | $2^4$～ $2^5$ | 01… |
| 2 | $2^3$～ $2^4$ | 001010,001101,001110,001001 |
| 3~6 | 0～ $2^3$ | 000101,000010,000110 |

表 2- 9 節點 5 的 Routing Table

節點 63 收到資訊後，將目標節點序列重新整理後為: 1,2,5,6,9,22…。已經對目標節點序列中的前 K 個節點 1,2,5,6 傳送過 FIND_NODE 指令，並且收到回覆。此時節點 63 已經找到了最靠近 Key 的 K 個節點，Lookup 在此終止。由節點 63 對 K 個節點傳送出 STORE(Data)指令。

## 五、 儲存資料

將資料存入 Kademlia 系統中的流程為:

以 Hash_Function 處理資料後得到 Key 值，對 Key 進行 Lookup，找出最靠近 Key 的 K 個節點後以 STORE 指令將<Key,Value>存入該 K 個節點中。
K 為系統參數，與 K-bucket 中的 K 值在 Kademlia 設計上為同一值，但應用時可視情況修改。

P2P 系統具有節點頻繁進出系統的特性，隨時都有大量的節點連線進入系統或離線離開系統，持有資料的節點亦可能離線，離線後便無法向 Kademlia 系統提供服務，為了避免持有資料的節點離線造成資料搜尋失敗，Kademlia 設計上讓同一筆資料保存在 K 個節點中，K 個持有資料的節點在一小時之內皆離線的機率低，藉此增加資料保存在系統中的機率。



DOI:10.6814/NCCU202100314

K 個持有資料的節點在系統中被視為互相靠近的小群體，但實際上位於不同的實體位置與 IP，當部分持有資料的節點遭到攻擊或是其他原因無法服務時，其他節點仍可提供服務。

## 六、 搜尋資料

找尋資料的流程與 Lookup 大致相同，傳送給其他節點的指令由 FIND_NODE 改為 FIND_VALUE，找到資料時停止運作，或是最靠近 Key 的 K 個節點都未持有資料時停止搜尋。

找到資料時，會對當下最靠近 Key 的無資料節點送出 STORE 指令，將資料存入節點。

每次平行詢問後，得到的回覆資訊能更靠近 Key 至少 1bit，藉此確保在 $\log_2 N$(N 為總節點數量)的平行詢問次數內尋找到資料。

圖 2- 4 搜尋資料時與 Key 的距離逐步減少


## 七、 Re-publish

持有資料的節點每間隔一小時會對該筆資料進行 Re-publish，搜尋最靠近 Key 的 K 個節點，隨後將資料存入最靠近 Key 的 K 個節點中。若持有資料節點在一小時內曾收到其他節點對於同一筆資料的 store 指令，代表此資料已經由其他節點進行過 Re-publish，此次的 Re-publish 暫停，避免浪費網路通訊資源。

P2P 系統中節點隨時可能離線，為了避免資料節點離線導致資料遺失在 Kademlia 系統中，藉由 Re-publish 確保在此資料在首次儲存到網路後的數小時後仍可被搜尋。



DOI:10.6814/NCCU202100314

經由多次 Re-publish 後，同一筆資料在網路中可能存有過多備份資料，造成儲存空間資源浪費。Kademlia 規範上設定資料有 24 小時的到期時間，資料初始發布的 24 小時後，持有資料節點不再對該資料進行 Re-publish，同時可該筆刪除資料，資料原始發布者被要求在 24 小時候重新發布資料。在不同應用上可斟酌調整到期時間，或是使靠近 Key 的節點有較長到期時間，遠離 Key 的節點到期時間較短。

## 八、 Refresh

每間隔一小時，節點必須對每個 K-bucket 範圍內的隨機 ID 進行節點 Lookup，若該 K-bucket 曾在一小時內藉由 Lookup(搜尋節點、Store 或 Search)更新過，則不須進行 Refresh。

Routing Table 內的節點可能在短時間內離線，為了避免 K-bucket 內的節點皆離線，需要定期更新 K-bucket。Refresh 可使 Routing Table 保持新鮮，確保 K-bucket 內的多數節點都在線。



DOI:10.6814/NCCU202100314

## 第二節　Kademlia 搜尋成功率

在 Kademlia 中搜尋 Key 時，節點會向其 Routing Table 中的節點進行詢問，藉由每一輪詢問後得到的回覆逐漸靠近 Key。由不同節點對同一個 Key 執行 Lookup 時所詢問的節點及其的回覆內容可能不同，導致最後找到的 K 個最靠近 Key 的節點不盡相同。當資料所存入的 K 個節點與搜尋時找到最靠近 Key 的 K 個節點完全不同時，將導致資料搜尋失敗。

先前對於 Kademlia 搜尋成功率的相關研究，都以 KAD 網路為研究對象，在實際運作的 P2P 網路中測試將資料存入網路後，由不同 ID 的節點搜尋該資料得到搜尋成功率。

### 一、　KAD 網路

KAD 網路是一種基於 Kademlia 運作的 P2P 網路。KAD 網路可使用 eMule 或 aMule 等客戶端與其相連，在熱門時期同時上線用戶多達一百五十萬用戶數。Kademlia 是技術名稱，KAD 網路則是依照 Kademlia 技術運作的 P2P 網路。

KAD 網路的參數與 Kademlia 有些微差異，KAD 網路的 ID 長度為 128 bit，節點收到 FIND_NODE 指令後，回傳的節點資訊數量為 4 與 2(儲存資料時與搜尋資料時)。Kademlia 回傳的節點資訊數量為 K。

KAD 網路搜尋資料時會分為兩個階段，第一階段為 Lookup，會找到距離 Key 最近的 K 個節點，過程中都是傳送 FIND_NODE 指令；第二階段為找到最靠近 Key 的 K 個節點後，會傳送 FIND_VALUE 指令。Kademlia 在整個過程中都會傳送 FIND_VALUE 指令。



DOI:10.6814/NCCU202100314

二、　搜尋成功率相關研究一

先前的研究(Hun J. Kang et al.,2009) 以探討 KAD 網路中搜尋失敗的原因為目標，在 KAD 網路中進行實驗並對遠端控制指令的傳送對象及頻率進行修改。此研究將儲存資料後持有資料的 K 個節點稱為 Replica Root，以每次搜尋能找到的 Replica Root 數量作為主要研究方式，其結果指出在儲存資料後立即進行資料搜尋能達到 92%的搜尋成功率，平均每次搜尋能找到 18%的 Replica Root。認為搜尋失敗的原因為持有資料的節點與搜尋到的節點不一致，有 45%的 Replica Root 不曾被搜尋到。

認為一致性過高的 Routing Table 導致 Lookup 的搜尋結果大量重複，最後蒐集到的 K 個節點只有少數是真正靠近 Key 的節點，其他節點距離 Key 過遠難以被搜尋到。若能使搜尋一致，將能以更少的節點儲存資料，達到相同的可靠程度，並處理被搜尋時附載不均問題。

為了使持有資料的節點與搜尋到的節點一致，提出三種修改方法:

1.調整參數。增加 K 值、修改回傳節點資訊量為 20 或增加 α 值。

2.找到最靠近 Key 的節點後，詢問該節點中最靠近 Key 的 20 個節點是否持有資料。在搜尋資料時使用，搜尋到最靠近 Key 的 Replica Root 機率為 90%，找不到該 Replica Root 的 10%在實施此方法後因為 Routing Table 有高達 70%的相似度，搜尋到該 Replica Root 的機率提升至 97%。

3.找到最靠近 Key 的 K 個節點後，K 個節點各自回傳最靠近自己的節點資訊。在儲存資料時使用，藉此處理只有少數最靠近 Key 的節點存有資料，其他持有資料節點都較遙遠的問題。

將在實驗結果章節中將此三種方法與本文提出的方法進行比較。



DOI:10.6814/NCCU202100314

三、　搜尋成功率相關研究二

隨後的研究(Bingshuang Liu et al.,2012)也以 KAD 網路為目標進行研究，並在 KAD 網路中進行實驗，以自製的爬蟲軟體 ANTHILL 紀錄節點儲存資料或搜尋資料的每個步驟，藉此分析 Lookup。

分析後認為搜尋失敗的原因可分為四個類型:

1.封包遺失

最初傳送 Request 給節點後未得到回覆，隨後重傳 Request 才成功得到回覆的案例視為 Packet loss。若連續傳送 4 次封包後，仍未得到回覆，視為該節點離線。

2.自私節點

部分節點只對 FIND_NODE 或 STORE 指令做出反應，而刻意拒絕 FIND_VALUE 指令。已經成功聯絡到節點，但節點不願意或無法回復 FIND_VALUE 指令，研究人員認為這類型的節點可能是由版權組織或其他人士所控制的自私節點。

3.指令因過期而未送出

資料搜尋者成功找到持有資料者，卻因為搜尋時間過長發生 Timeout 而終止行動，來不及傳送 FIND_VALUE。

4.無法找到資料節點

資料搜尋者無法找到資料持有者，研究人員認為是因為 KAD 網路的參數所導致。



DOI:10.6814/NCCU202100314

對應這四種原因提出三種改良方法:

1.封包重傳

在設定的 timeout 到達之前，若未收到回復便重傳指令，藉此處理 packet loss 問題。

2.相鄰節點搜尋(neighborhood lookup)

與 Key 有 θ 個共同 prefix 的所有節點被認為是 Key 的相鄰節點，把資料存在相鄰節點中的所有節點上，由這些節點共同處理 FIND_VALUE 指令。將在實驗結果章節中將此方法與本文提出的方法進行比較。

3.參數調整

將儲存資料與搜尋資料時回傳的節點資訊數量調整為 4 ，能夠減少無法找到資料節點的狀況並降低搜尋時間。



DOI:10.6814/NCCU202100314

## 第三節　網路擾動

網路擾動代表 P2P 網路中，節點頻繁進行網路連線與離線的現象。一般的網路擾動下，節點進出網路的頻率較低，網路結構不會被網路擾動所破壞，節點有充裕的時間能更新資訊；高度網路擾動指大量節點會在短時間內加入或離開網路(Zhiyu Liu et al.,2007)，舊有的網路結構或資訊可能變得不可靠。

churn 對於 P2P 系統的可靠程度有著極大的影響，若多數的節點連續上線時間長，離線頻率低且離線時間短，P2P 系統將具備可靠且穩定的服務能力；若多數的節點連續上線時間短，離線頻率高且離線時間長，P2P 系統的服務能力降低，服務請求容易失敗。

網路擾動現象不容易觀察及研究，有眾多要素需要納入考量，例如節點的連續上線時間長、連續離線時間長以及連線/離線頻率，即使透過監控得到這些要素的分布情況，將三者統合成完整系統有其難處。而透過監控得到這些要素的個別分布情況本身並不容易，可能出現許多偏誤，導致測量結果不夠準確。

早期的研究認為節點的平均上線時間呈現指數分布，後續的研究(Daniel Stutzbach et al.,2006)對大型 P2P 系統 BitTorrent ,Gnutella 及 KAD 進行研究，並推翻平均上線時間呈現指數分布的結果。BitTorrent 中節點會定期與 Tracker 聯絡，Tracker 能夠記錄節點個別上限的時間但無法得知節點離線的確切時間，研究者就以 Tracker 內部的紀錄進行研究。研究 Gnutella 及 KAD 時，研究者先藉由 Table 取得一定範圍內的所有節點訊息，並定期對節點傳送 Ping 指令，藉此研究節點狀態。研究者以對上述 P2P 系統的觀測資料，得出結論:

1.在不同 P2P 系統中，用戶上線離線的變動情況十分相似

2.用戶上線的時間長度並非呈現指數分布，而較接近韋伯分布

3.大部分活躍用戶的上線下線變化較穩定，其餘用戶具有高度變動性

4.用戶上線時間長度與連續現身的程度具有一定關聯性



DOI:10.6814/NCCU202100314

對於網路擾動研究不易進行，許多因素可能造成偏誤。上線時間長的節點較容易被紀錄也更容易搜尋到，常被選為觀測的節點，導致平均上線時間長度比真實時間久；BitTorrent 中，離線時不通知的節點無法準確紀錄時間；無法測量上線時間比觀測時間更久的節點；KAD 與 Gnutella 中的部分節點難以被搜尋，無法被測量；若節點在爬蟲的間隔期間離線，隨後重新上線，將被視為不曾離線；NAT 後若有眾多使用用戶，該節點可能產生永久在線的假象。有眾多的因素可能導致測量結果不夠精確，研究者能夠對網路擾動進行分析及模擬，卻不能保證與現實狀況一致。Churn 是否會隨著年代以及人們的網路使用習慣而改變，目前並沒有詳盡的相關研究。

有份研究(Octavio Herrera et al.,2007)對於研究 P2P 系統時，該如何模擬網路擾動提出建議。認為網路擾動的架構應該包含使用者、節點、網路及資源等層面。

使用者層面，將節點依照平均上線時間的長短分成三個類別:

Benefactor 平均上線時間以天數計算，會長期待在網路中。

Peers 平均上線時間以小時數計算，約為一場電影的時間。

Peerers 平均上線時間以分鐘數計算，下載完小型資料即離線。

實驗時，能改變使用者層面不同總類的占比。

節點層面，每個節點有各自的平均上線時間及平均離線時間，每個節點有各自的上線下線比利。

網路層面，使用 Churn Factor 平均離線比例 (Ch)與總用戶數量(N)計算當下在線的用戶數量。

資源層面，將資源依照不同資料大小、受歡迎程度以及儲存時的分布情況做分類，不同資源種類可能包含音樂、電影與遊戲等。

本研究採用其將節點依平均上線時間分類並調整比例的建議。



DOI:10.6814/NCCU202100314

# 第三章 研究方法

## 第一節　依距離分析 Kademlia 搜尋失敗情境

依照搜尋時找到的最靠近 Key 的 K 個節點位置(Found nodes)與持有資料節點(data nodes)位置兩群節點的位置關係分析 Kademlia 搜尋失敗情境。搜尋失敗時，Found nodes 與 Data nodes 兩群節點不重複，依據兩者的相對關係分為三個類型。

將 Found nodes 中距離 Key 最近的節點稱為 node-F，Data nodes 中距離 Key 最近的節點稱為 node-D。依照 node-F 與 node-D 的相對位置將搜尋失敗情境分為以下三種:

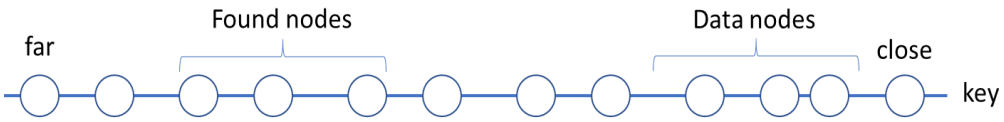
1.搜尋位置錯誤

CommonPrefix(node-F, Key) <= CommonPrefix(node-D, Key)

XOR(node-F, Key) > XOR(node-D, Key)

node-D 相比於 node-F，其與 Key 有更多共同前綴，距離 Key 更接近。這個情況代表 search 時無法順利接近 Key，而資料被存在相對正確(靠近 Key)的位置。對這筆資料搜尋失敗可能是一次性的，只會發生在這次的搜尋者，若由其他節點搜尋資料，仍有相當高的成功率。

圖 3- 1 搜尋位置錯誤






DOI:10.6814/NCCU202100314

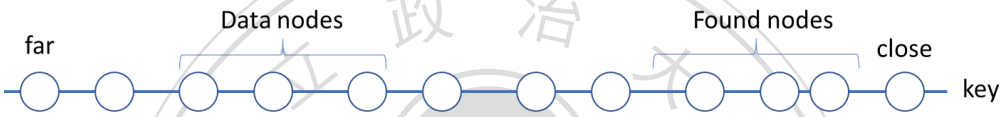
2.資料位置錯誤

CommonPrefix(node-F, Key) > CommonPrefix(node-D, Key)

XOR(node-F, Key) < XOR(node-D, Key)

node-D 相比於 node-F，其與 Key 有更少共同前綴，距離 Key 更遙遠。這個情況代表儲存資料時，無法順利找到靠近 Key 的節點，資料被存在相對錯誤(遠離 Key)的位置。在短時間內，對這筆資料搜尋幾乎都會失敗。隨著 Data nodes 進行 Re-publish 後，情況可能會改善。

圖 3- 2 資料位置錯誤




3.資料不存在網路中

Data nodes 不存在。代表 Data nodes 都已經離線，此時不可能搜尋到該資料。在任何 Data nodes 上線後，情況可能改善。



DOI:10.6814/NCCU202100314

## 第二節　模擬方法及環境

以 Python 撰寫的模擬環境，能依據使用者調整 K、α、Refresh frequency 等系統參數進行 Kademlia 模擬，能夠讀取時間序列中的節點上線、節點離線、存取資料等事件，並由持有資料的節點主動發起 Re-publish 事件。一切運作方式與先前介紹的 Kademlia 相同。

時間序列事件包含了節點上線、節點離線、儲存資料、搜尋資料、搜尋節點與 Re-publish 事件，這些事件由時間、節點、事件類型、Key、資料名稱、被動節點組成，並依時間順序排列為序列。程式藉由讀取序列的資訊進行模擬。

模擬結果主要以成功率、不同搜尋失敗類型與通訊成本來表示，藉此衡量 Kademlia 的效率以及修改方法的成效。

通訊成本分為三種:

Ping: 檢查 K-bucket 內的節點是否在線。

FindNode:進行 Lookup 時，傳送 Find_Node 指令。

ReturnNode:收到 Find_Node 後回傳資訊。

以這三種維護 Routing Table 的通訊成本為主要衡量標準。

模擬結果以下列方法表示:

Search success rate(%)，每小時的資料搜尋成功率

Found nodes wrong rate(%) ，搜尋位置錯誤發生率

Data node wrong rate(%) ，資料位置錯誤發生率

Data node lost rate(%) ，資料不存在網路中發生率

Cost Ping(1000times/hour)，檢查 K-bucket 內節點是否在線次數

Cost FindNode (1000times/hour) ，FIND_NODE 指令的傳送次數

Cost ReturnNode (1000times/hour) ，回覆 FIND_NODE 指令的次數



DOI:10.6814/NCCU202100314

模擬中會忽略掉一些現實環境可能遇到的問題，例如在資料搜尋失敗相關研究中提及的網路不通順、自私節點等問題，也不對搜尋時間等議題做探討，專注於是否能找到持有資料節點這件事。模擬中不會發生封包遺失或 NAT 的網路問題，不存在自私節點，節點皆有足夠效能的 CPU 及網路頻寬，節點收到遠端控制指令後一定會照指令回傳資料。

模擬時的總結點數量為 40000 個節點，由隨機 1000 個節點對 1000 筆資料進行儲存，每一筆資料儲存後隨即由一隨機節點對該資料進行第一次搜尋，此後每一小時都會由隨機節點對此 1000 筆資料進行搜尋，藉此測試搜尋成功率。

模擬時的平均節點上線時間以論文(Daniel Stutzbach et al.,2006)所提供的研究結果做模擬，因此使用韋伯分布，該論文提供數種分布參數，選用 $k = 0.59, \lambda = 41.9$ 作為分布模擬使用。相關論文中建議模擬 Churn 時依據平均上線時間的差異將節點做分類，分為 BeneFactor 平均上線時間達數小時至天，Peers 平均上線時間為一部電影的時間，Peerers 平均上線時間為數分鐘，模擬時可將三者以不同比例做調配。模擬時會將結點分為:

長時-平均上線時間大於等於三小時。

中時-平均上線時間小於三小時，大於等於 30 分鐘。

短時-平均上線時間小於 30 分鐘。

平均離線時間以常態分布模擬，平均 900 minutes，標準差為 150 minutes。

搜尋時的平均節點上線時間將以三種方式模擬，先以韋伯分布 $k = 0.59, \lambda = 41.9$ 作為節點個別的平均上線時間分布，隨後調整占比為:

1.長時節點 5%，中時節點 10%，短時節點 85%

2.長時節點 10%，中時節點 20%，短時節點 70%

3.長時節點 20%，中時節點 40%，短時節點 40%



DOI:10.6814/NCCU202100314

## 第三節　初步改良方式及其實驗結果分析

1. 估計離線時間

在系統中記錄節點的上線時間以及平均上線時間長度，藉由這兩者推估出節點的估計離線時間。估計離線時間將作為後須修改方案的主要應用概念，估計離線時間是藉由過去經驗估計出來的，節點有可能在達估計離線時間之前便離線，但此方法仍能發揮一定效用。

2. Find Re-publish node

資料不存在網路中，當持有同一筆資料的 K(或更多)個節點皆在在一小時內(re-publish 發生前)離線時，此資料將消失於網路中。

節點收到 store 指令後，持續上線一小時便會觸發 Re-publish 事件，若成功觸發 re-publish 事件，該筆資料不易消失於網路之中，不會成為資料不存在網路中的搜尋失敗案例；若無法觸發 Re-publish 事件，則必然形成資料不存在網路中的搜尋失敗案例。為了避免此情況發生，在此提出 Find Re-publish node 方法。

進行儲存資料時，找到最靠近 Key 的 K 個節點後，將資料存入節點中，此時會檢查這 K 個節點中是否有任何節點的估計離線時間在下達 Store 指令的一小時後。若 K 個節點的估計離線時間皆不到下達 Store 指令的一小時後，便認為不會觸發 Re-publish 事件，下次搜尋時這筆資料會以資料不存在網路中的情況搜尋失敗。在此時進行改良，向目標節點序列的第 k+1 個節點開始依序詢問其估計離線時間，尋找一個估計離線時間在下達 Store 指令的一小時後的節點，找到符合條件的節點後將資料存入該節點，並結束。在下次搜尋搜尋資料時，若離線時間較晚於估計離線時間或與其相同，不會造成資料不存在網路中的狀況。

估計離線時間有不準確的可能，依據估計離線時間運作的修改方式未必能



DOI:10.6814/NCCU202100314

起作用，但仍預期實驗後資料不存在網路中的次數會減少。

將此修改方法與原版的 Kademlia 方法作比較，在韋伯分布的平均上線時間下進行模擬，使用的參數皆為:ID 長度 160bits、k=10、α=3、FIND_NODE 指令後回傳 K 筆節點資料、Re-publish 的間隔為一小時、Refresh 的間隔為一小時。

3. Find re-publish node 實驗結果

| | 長時 5%，中時 10% | | 長時 10%，中時 20% | |
|---|---|---|---|---|
| 方法 | 原版 | Find RN | 原版 | Find RN |
| 成功率 | 80.5% | 80.8% | 91.5% | 91.6% |
| 搜尋位置錯誤率 | 143.7 | 147.7 | 81.2 | 80.1 |
| 資料位置錯誤率 | 0.75 | 0.5 | 0 | 0 |
| 資料遺失率 | 50.3 | 42.8 | 3.6 | 3.6 |
| Ping(千次/小時) | 106.2 | 106.2 | 155.0 | 155.3 |
| FindNode(千次/小時) | 564.8 | 565.5 | 691.5 | 691.9 |
| ReturnNode(千次/小時) | 130.7 | 131.0 | 178.1 | 178.2 |

表 3- 1 Find Re-publish node 實驗結果

Find Re-publish node 實驗中，期望在最靠近 Key 的 K 個節點無法觸發 Re-publish 時，從目標節點序列的第 k+1 個節點開始尋找到能觸發下次 Re-publish 的節點，藉此減少資料不存在網路中發生次數。

依據表 3-1 的實驗數據，在長時節點佔 5%的環境中，此方法能將資料不存在網路中的發生次數減少約 15%，搜尋成功率提高 0.3%，搜尋位置錯誤次數增加；在長時節點佔 10%的環境中，此方法並未改變資料不存在網路中的發生次數，但可能在 Re-publish 事件發生後改變 Routing Table，因此搜尋成功率有 0.1%的增長；在長時節點佔 20%的環境中不會出現資料不存在網路中的狀況，無須測試。



DOI:10.6814/NCCU202100314

Find Re-publish node 未達成預期成效，因此對於最靠近 Key 的 K 個節點無法觸發 Re-publish 的案例中所整理出的目標節點序列作調查。發現此狀況發生時，目標節點序列的總數量可能小於 K+1 個，因此 Find Re-publish node 在大部分案例中未被觸發，自然無法減少資料不存在網路中的發生次數。

4. 節點孤立

目標節點序列的總數量皆小於 K+1 個，有兩種可能原因:

1.Lookup 發起者的 Routing Table 內大部分節點皆已經離線，在線的節點數不足 K+1 個，這些節點收到 FIND_NODE 指令後會回傳的 K 個最靠近 Key 的節點，這些節點多數已經離線，導致 Lookup 所蒐集到的上線節點總數不足 K 個。

2.Lookup 發起者是 Routing Table 中唯一在線的節點，Routing Table 中的其他節點皆以離線。

在這兩個情況下，Lookup 發起者仍在線，卻失去與其他節點聯絡的方法，失去與 Kademlia 系統主動聯繫的方法。有可能在後來收到其他節點的指令，藉此更新 Routing Table，與 Kademlia 重新連線，但 Lookup 發起者在此時的所有行為不論是儲存資訊、搜尋資訊或搜尋節點會失敗。

進一步研究目標節點序列的總數量皆小於 K+1 個的情況，發現約 95%的案例中，目標節點序列的總數量為 1 個節點，該節點便是 Lookup 發起者本身。將節點本身是 Routing Table 中唯一在線的節點，Routing Table 中的其他節點皆以離線的情況稱作節點孤立。

若節點孤立發生在儲存資料時，儲存資料者本身是目標節點序列中的唯一節點，會將資料存在自己身上。儲存資料者的 ID 為最靠近 Key 的 K 個節點的機率非常低，因此資料被搜尋到的可能性低，容易搜尋失敗。其他用戶搜尋資料時，若是順利找到靠近 Key 的節點，Found nodes 會比 Data nodes 更靠近 Key，容易產生資料位置錯誤的搜尋失敗案例。若儲存資料者能與 Kademlia 重



DOI:10.6814/NCCU202100314

新連線，資料才可能被存放到靠近 Key 的節點上；若無法重新連線，在此節點離線後會轉回資料不存在網路中的搜尋失敗案例。

若節點孤立發生在儲存資料時，搜尋資料者本身是目標節點序列中的唯一節點，自然無法找到資料，導致搜尋失敗。搜尋資料者的 ID 為最靠近 Key 的 K 個節點的機率非常低，若此資料儲存時未發生節點孤立，Data nodes 應比 Found nodes 更靠近 Key，會產生搜尋位置錯誤的搜尋失敗案例。在這情況下，搜尋資料者搜尋任何資料都會失敗，除非自己持有該資料，但是對於搜尋同一筆資料的資料而言，這筆資料仍是可搜尋的。





DOI:10.6814/NCCU202100314

## 第四節　Kademlia 搜尋失敗原因

1. 大量節點連線時間過短

儲存資料後，存有資料的 K 個節點若在發生 re-publish 前就全部離線，會導致資料不存在網路中發生。節點連線時間過短會造成大量 table 中的節點或是收到的回傳節點大量離線，使系統將錯誤的節點（較遙遠的節點）當成最近的 k 個節點，導致節點只搜尋到局部的最靠近 Key 的 k 個節點，無法有效率地儲存或取得資料。更嚴重者會導致節點孤立現象，發生在儲存時，會使資料難以被搜尋，該資料同時處於消失於系統的風險；發生在搜尋面時，該次搜尋注定失敗。

此為導致搜尋失敗的主要原因，可能形成搜尋位置錯誤、 資料位置錯誤以及資料不存在網路中三種搜尋失敗類型。

2. 節點長時間離線

將資料存入 Kademlia 系統後，若存有資料的 K 個節點同時離線，此資料便會暫時消失在系統中，此 K 個節點離開系統的時間有多長，這筆資料就會從系統中消失多久的時間，這段期間內搜尋資料時總會以搜尋失敗作收。

節點長時間離線的特性使得造成資料不存在網路中的 Data Node 短期間內未重返系統，在 Data Node 回歸系統前對此資料進行搜尋，都會以資料不存在網路中作收。

3. 節點無法有效認識周遭節點

理想中的 Kademlia 網路中，節點會對於距離接近的節點有完全的認識，將資料存入 Kademlia 系統時，會找到整個系統中與 Key 最接近的 K 的節點，若在當下由任意節點搜尋資料，都能夠找到那 K 個節點，並且順利取得檔案。但現實並非如此，在資料儲存者與資料搜尋者都未發生節點孤立的情況下，Found



DOI:10.6814/NCCU202100314

node 與 Data node 都靠近 Key 時，仍可能發生資料搜尋失敗，便是節點無法有效認識周遭節點所造成的。

這些搜尋失敗原因並非彼此獨立，而是交互影響的。大量節點連線時間過短會造成 Routing Table 內的大量節點離線，或是 FIND_NODE 指令所回傳的 K 個節點大量離線，都會間接導致節點無法有效認識距離自己較近的其他節點；儲存資料時的節點孤立也會間接增加資料不存在網路中發生次數，隨後因大量節點長期離開 Kademlia 網路使得資料不存在網路中持續發生。





DOI:10.6814/NCCU202100314

## 第五節　改良方法

1. 長壽節點

為了處理節點孤立問題而提出的改良方法。

節點傳送出 FIND_NODE、FIND_VALUE 指令與收到指令的節點回覆時，會將自己的估計離線時間傳給彼此，節點會將估計離線時間最晚的 K 個其他節點的資訊及其估計離線時間保存下來，將這 K 個節點稱為長壽節點。

傳送出 FIND_NODE、FIND_VALUE 指令與回覆這些指令時，也會將自己的長壽節點一併傳送給對方。收到其他節點傳來的長壽節點後，與自己本身的長壽節點進行比較，將這 2*K 個節點中估計離線時間最晚的 K 個節點保留下來，作為新的長壽節點。

當節點進行 Lookup 時，若發生節點孤立的現象，此時將長壽節點加入目標節點序列，以 Lookup 搜尋自己的 ID，藉此與 Kademlia 重新連接，隨後重新執行原本的 Lookup 指令。

長壽節點未必能記錄到系統中估計離線時間最晚的 K 個其他節點，與節點本身和其他節點的來往頻率及對象有關，而估計離線時間本身存在準確度的問題，同時長壽節點也有出現節點孤立現象的可能性。基於以上原因，此方法並非在每一次都能處理節點孤立問題。

2. 遠距搜尋

為了解決接點無法認識所有鄰近節點的問題而提出的改良方法。

假設存在兩群靠近 Key 的節點群 A 以及節點群 B 如圖 3- 3，節點群 A 中所有節點的 Routing Table 內都不存在節點群 B 的任一節點資訊，節點群 B 中所有節點的 Routing Table 內亦不存在節點群 A 的任一節點資訊，節點群 A 及節點群 B 將無法藉由 Refresh 機制或是 Lookup 來認識彼此。節點群 A 的節點進行 Refresh 或是 Lookup 以更新靠近 Key 的節點時，會先以距離最近的 K-bucket 為



DOI:10.6814/NCCU202100314

詢問目標，而該 K-bucket 內的節點即為節點群 A 中的節點，依據這兩群節點互不在對方 Routing Table 內的假設，這兩群節點不會因為 Refresh 或 Lookup 而互相接觸。

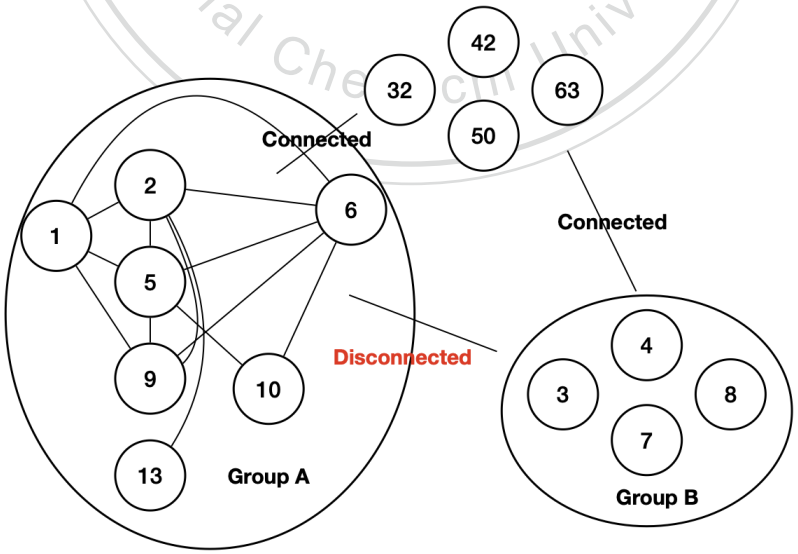
Refresh 或 Lookup 靠近自己的節點，能夠確認舊節點是否在線並將新加入系統的節點加入自己的 Routing Table 中，無法使兩群相近而無交集的節點群相連。

儲存資料時，存入的節點可能為節點群 A 或節點群 B 的節點，若搜尋資料時找到不同節點群，將導致資料搜尋失敗。

Refresh 後，進行一次對節點自身 ID 的 Lookup，只將共同前綴數量為 0 的 K-bucket 加入目標節點序列，而非加入整個 Routing Table 的節點，目標節點序列同樣依照 FIND_NODE 的回傳資訊更新。

以較遙遠的節點開始向 Key 搜尋，此方法有可能找到與原本無交集的節點群，但並不保證會有成效，可能會找到原本的節點群而非無交集的節點群。相近節點群無交集的情況並不多，此方法能改善的效率有限。

圖 3- 3 遠距搜尋示意圖






DOI:10.6814/NCCU202100314

## 第六節　Kademlia 搜尋失敗原因及改良方法統整

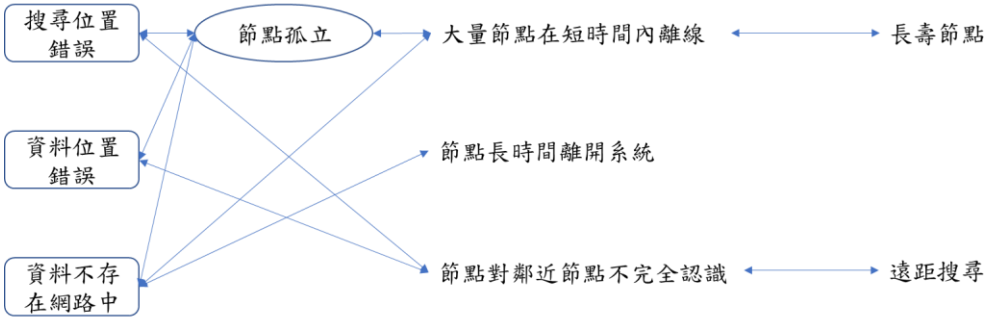
搜尋失敗的主要原因為大量節點連線時間過短，導致持有資料節點皆離線以及節點孤立的情況發生。持有資料節點皆離線的情況需要藉由調整系統參數來改善。節點孤立發生在搜尋者時，該次搜尋必定失敗，造成搜尋位置錯誤的發生次數增加；節點孤立發生在資訊儲存者時，該筆資料被成功搜尋的可能性微乎其微，造成資料位置錯誤的發生次數增加；而發生節點孤立的資訊儲存者離線時，該筆資料會消失於網路中，造成資料不存在網路中的搜尋失敗類型增加。

長壽節點能使發生節點孤立的節點與長壽節點重新連線(假如有至少一個長壽節點在線，同時該節點未發生節點孤立)，使搜尋位置錯誤、資料位置錯誤與資料不存在網路中的發生次數減少。

搜尋失敗的次要原因為大量節點長期離開 Kademlia 網路，導致形成資料不存在網路中的資料會長期離開系統，資料不存在網路中會持續發生。必須藉由減少資料不存在網路中發生的次數改善，調整系統參數以及處理節點孤立問題能夠減少長期資料不存在網路中發生次數。

搜尋失敗的第三原因為節點無法有效認識距離自己較近的其他節點，導致 Lookup 無法順利找到最靠近 Key 的 K 個節點，造成搜尋位置錯誤與資料位置錯誤發生。假設存在兩群靠近 Key 但無交集的節點群，而這兩群節點無法藉由 Refresh 及 Lookup 找到彼此，可藉由遠距搜尋解決此問題。

圖 3-4 搜尋失敗原因及改良方法關係圖






DOI:10.6814/NCCU202100314

# 第四章 實驗結果與分析

## 第一節　將改良方法與 Kademlia 進行比較

原版:以 Kademlia 運作，參數為 160bits、K=10 與 α=3。隨後未特別提及者，皆以同樣參數運作。

1. 長時 5%，中時 10%的實驗結果

依據表 4-1 的實驗數據遠距搜尋將搜尋成功率提升 1.1%，資料不存在網路中的發生次數減少 1.5%，進行 Lookup 時傳送更少的 FIND_NODE 指令，同時得到較多的回覆次數。長壽節點將搜尋成功率提升 8.6%，搜尋成功率的增幅主要源於搜尋時節 點孤立次數降低使搜尋位置錯誤減少，搜尋成功率提升時傳送 FIND_NODE 指令的次數僅增加不到 6%。同時實施遠距搜尋與長壽節點，相較於只實施長壽節點時，搜尋成功率提升 0.9%達到 90%，搜尋位置錯誤與資料不存在網路中的發生次數皆明顯下降。

| 方法 | 原版 | 遠距搜尋 | 長壽節點 | 遠距搜尋+長壽節點 |
|---|---|---|---|---|
| 成功率 | 80.5% | 81.6% | 89.1% | 89.8% |
| 搜尋位置錯誤率 | 14.4% | 14.6% | 6.6% | 6.1% |
| 資料位置錯誤率 | 0.1% | 0.4% | 0.1% | 0.1% |
| 資料遺失率 | 5% | 3.5% | 4.2% | 4% |
| 搜尋時節點孤立率 | 14.2% | 13.8% | 6.6% | 6.2% |
| Ping(千次/小時) | 106.2 | 111.6 | 115.7 | 122.2 |
| FindNode(千次/小時) | 564.9 | 564.7 | 597.1 | 600.3 |
| ReturnNode(千次/小時) | 130.7 | 131.1 | 144 | 144.5 |

表 4-1 改良方法與 Kademlia 比較結果(長時 5%，中時 10%)



DOI:10.6814/NCCU202100314

2. 長時 10%，中時 20%的實驗結果

依據表 4-2 的實驗數據遠距搜尋未使搜尋成功率提升，但資料不存在網路中的發生次數減少 0.2%。長壽節點將搜尋成功率提升 6.7%，搜尋成功率的增幅主要源於搜尋時節點孤立次數降低使搜尋位置錯誤減少，搜尋成功率提升時傳送 FIND_NODE 指令的次數僅增加約 5%。同時實施遠距搜尋與長壽節點，相較於只實施長壽節點時，搜尋成功率沒有提升，傳送 FIND_NODE 指令的次數降低。

| 方法 | 原版 | 遠距搜尋 | 長壽節點 | 遠距搜尋+長壽節點 |
|---|---|---|---|---|
| 成功率 | 91.5% | 91.5% | 98.2% | 98.1% |
| 搜尋位置錯誤率 | 8.1% | 8.3% | 1.4% | 1.5% |
| 資料位置錯誤率 | 0% | 0% | 0% | 0% |
| 資料遺失率 | 0.4% | 0.2% | 0.4% | 0.4% |
| 搜尋時節點孤立率 | 8.1% | 8.3% | 1.5% | 1.5% |
| Ping(千次/小時) | 155.0 | 163.9 | 165.7 | 175.0 |
| FindNode(千次/小時) | 691.5 | 691.8 | 726.2 | 726.7 |
| ReturnNode(千次/小時) | 178.2 | 178.4 | 193.5 | 193.4 |

表 4- 2 改良方法與 Kademlia 比較結果(長時 10%，中時 20%)



DOI:10.6814/NCCU202100314

3. 長時 20%，中時 40%的實驗結果

依據表 4-3 的實驗數據遠距搜尋未使搜尋成功率提升，但傳送 FIND_NODE 指令的次數降低。長壽節點將搜尋成功率提升 3.4%，搜尋成功率的增幅主要源於搜尋時節點孤立次數降低使搜尋位置錯誤減少，搜尋成功率提升時傳送 FIND_NODE 指令的次數僅增加不到 3%。同時實施遠距搜尋與長壽節點，相較於只實施長壽節點時，搜尋成功率提升 0.2%，進行 Lookup 時傳送更少的 FIND_NODE 指令，同時得到較多的回覆次數。

遠距搜尋在長時節點占比低時能夠明顯提升搜尋成功率，在長時節點占比高時搜尋失敗的主因為搜尋時的節點孤立，因此遠距搜尋對於提升搜尋成功率沒有顯著的成效，但有時能降低傳送 FIND_NODE 指令的次數。長壽節點成功降低搜尋時的節點孤立，在三種環境中都能替搜尋成功率帶來顯著的增長，傳送 FIND_NODE 指令的次數僅增長不到 6%。同時實施遠距搜尋與長壽節點時總是能達成更高的搜尋成功率，或以更少的傳送 FIND_NODE 指令次數完成 Lookup。以上數據指出遠距搜尋與長壽節點兩者同時實施時，能夠有效改良 Kademlia 的搜尋成功率。

| 方法 | 原版 | 遠距搜尋 | 長壽節點 | 遠距搜尋+長壽節點 |
|---|---|---|---|---|
| 成功率 | 96% | 96% | 99.4% | 99.6% |
| 搜尋位置錯誤率 | 4% | 4% | 0.6% | 0.4% |
| 資料位置錯誤率 | 0% | 0% | 0% | 0% |
| 資料遺失率 | 0% | 0% | 0% | 0% |
| 搜尋時節點孤立率 | 4% | 4% | 0.6% | 0.4% |
| Ping(千次/小時) | 249.5 | 265.1 | 259 | 275.8 |
| FindNode(千次/小時) | 891.6 | 890.2 | 915.6 | 915.2 |
| ReturnNode(千次/小時) | 267.3 | 267.3 | 280.4 | 280.9 |

表 4-3 改良方法與 Kademlia 比較結果(長時 20%，中時 40%)



DOI:10.6814/NCCU202100314

## 第二節　將改良方法與其他方法進行比較

原版:以 Kademlia 運作，參數為 160bits、K=10 與 α=3。隨後未特別提及者，皆以同樣參數運作。

My Method:長壽節點以及遠距搜尋。

My Method(K=15):長壽節點以及遠距搜尋，將 K 值修改為 15。

Other-1:Why KAD Fails 所提的方法-Fix1，找到最靠近 Key 的節點後，由該節點回傳其周遭 20 個節點的資訊，像此 20 個節點傳送 FIND_VALUE 指令。

Other-2:Why KAD Fails 所提的方法-Fix2，找到最靠近 Key 的節點後，由目標節點序列中的前 K 個節點回傳自己的周遭節點資訊。

Other-3:Why KAD Fails 所提的參數調整方法，將 K 值修改為 20。

Other-4: Revisiting Why KAD Fails 所提的方法-Neighborhood Lookup，將與 Key 擁有一定共同 prefix 數量的節點當作 Neighborhood 的成員，由整個 Neighborhood 負責儲存同一檔案並回覆其 FIND_VALUE 指令。

1. 長時-5%,中時-10%,短時-85%

依據表 4-4 的實驗數據，Other-1 與 Other-2 的搜尋成功率相較於原版只有微幅成長，效果不顯著；Other-3 有效降低搜尋時節點孤立及資料不存在網路中的次數使搜尋成功率成長 10.4%，傳送 FIND_NODE 指令的次數增加了超過 100%；Other-4 有效降低資料不存在網路中的次數使搜尋成功率較原版提升了 4%，傳送 FIND_NODE 指令的次數增加約 25%。本文的修改方法在 K 值為 10 時成功率略低於 Other-3，若將 K 值調整為 15，搜尋成功率將達到 94.6%，較高於 Other-3 的 90.9%，同時傳送 FIND_NODE 指令的次數低於 Other-3。

以資料不存在網路中的次數進行探討， Other-3 與 Other-4 將降低了 90%以上的資料不存在網路中的次數達到 4 次。在實施本文的方法時將 K 值調整為 15，資料不存在網路中的次數將降為 9.7 次，仍然高於 4 次，說明了要減少資



DOI:10.6814/NCCU202100314

料不存在網路中發生的次數，增加備份數量是最有效的方法。

| 方法 | 原版 | My Method K=10 | My method (k=15) | Other-1 | Other-2 | Other-3 | Other-4 |
|---|---|---|---|---|---|---|---|
| 成功率 | 80.5% | 89.8% | 94.6% | 81.0% | 80.7% | 90.9% | 84.5% |
| 搜尋位置錯誤率 | 14.4% | 6.1% | 4.4% | 14.3% | 14.5% | 8.6% | 15.1% |
| 資料位置錯誤率 | 0.1% | 0.1% | 0% | 0% | 0.1% | 0% | 0% |
| 資料遺失率 | 5% | 4% | 1% | 4.6% | 4.8% | 0.4% | 0.4% |
| 搜尋時節點孤立率 | 14.2% | 6.2% | 4.4% | 14.2% | 14.1% | 8.5% | 14.2% |
| Ping (千次/小時) | 106.2 | 122.2 | 159.5 | 106.2 | 110.7 | 187.6 | 147.9 |
| FindNode (千次/小時) | 564.8 | 600.3 | 1008.9 | 564.8 | 634.3 | 1421.1 | 709.6 |
| ReturnNode (千次/小時) | 130.7 | 144.5 | 205.1 | 130.7 | 138.0 | 250.6 | 163.8 |

表 4-4 改良方法與其他方法比較結果(長時 5%，中時 10%)



DOI:10.6814/NCCU202100314

2. 長時-10%,中時-20%,短時-70%

依據表 4- 5 的實驗數據，Other-1 與 Other-2 的搜尋成功率相較於原版無顯著差異；Other-3 有效降低搜尋時節點孤立及資料不存在網路中的次數使搜尋成功率成長 4.29%，傳送 FIND_NODE 指令的次數增加了超過 100%；Other-4 的搜尋成功率較原版無顯著提升，傳送 FIND_NODE 指令的次數增加約 45%。

以資料不存在網路中的次數進行探討， Other-3 與 Other-4 皆不再發生資料不存在網路中的狀況。實施本文的修改方法時，將 K 值調整為 15，資料不存在網路中的狀況亦將停止發生。

以搜尋時的節點孤立次數進行探討，長壽節點成功發揮作用，使搜尋時的節點孤立次數降低約 80%，Other-3 則降低了 50%，因此本文的改良方法達成較高的搜尋成功率。





DOI:10.6814/NCCU202100314

| 方法 | 原版 | My Method | My Method (k=15) | Other-1 | Other-2 | Other-3 | Other-4 |
|---|---|---|---|---|---|---|---|
| 成功率 | 91.5% | 98.1% | 99.0% | 91.6% | 91.4% | 95.8 | 91.9% |
| 搜尋位置錯誤率 | 8.1% | 1.5% | 1% | 8.1% | 8.2% | 4.2% | 8.1% |
| 資料位置錯誤率 | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| 資料遺失率 | 0.4$ | 0.4% | 0% | 0.3% | 0.4% | 0% | 0% |
| 搜尋時節點孤立率 | 8.1% | 1.5% | 1% | 8.1% | 8.1% | 4.2% | 8% |
| Ping (千次/小時) | 155.0 | 175.0 | 222.8 | 155.0 | 160.0 | 264.9 | 225.9 |
| FindNode (千次/小時) | 691.5 | 726.7 | 1177.7 | 691.5 | 767.0 | 1679.5 | 901.8 |
| ReturnNode (千次/小時) | 178.1 | 193.4 | 266.1 | 178.1 | 187.0 | 332.0 | 230.6 |

表 4- 5 改良方法與其他方法比較結果(長時 10%，中時 20%)



DOI:10.6814/NCCU202100314

3. 長時-20%,中時-40%,短時-40%

依據表 4-6 的實驗數據，Other-1 與 Other-2 的搜尋成功率相較於原版無顯著差異；Other-3 在成功率達提升了 2.1%，傳送 FIND_NODE 指令的次數增加了超過 100%；Other-4 的搜尋成功率較原版無顯著差異，傳送 FIND_NODE 指令的次數增加了 25%以上。

在長時 20%,中時 40%的環境中資料消失於網路中的狀況幾乎不再發生。

以搜尋時的節點孤立次數進行探討，長壽節點成功發揮作用，使搜尋時的節點孤立次數降低約 90%；Other-3 則降低了 50%。在長時 20%,中時 40%的環境中搜尋時的節點孤立成為搜尋失敗的主要原因，而長壽節點是因應此狀況而提出的方法，因此本文的改良方法能達到較高的搜尋成功率。

前人所提的方法在本實驗中成效較不理想，可能是由於先前的研究大多針對 KAD 的搜尋失敗狀況進行探討，KAD 本身存在著參數設計缺陷，這些方法可能解決了此問題，而 Kademlia 並不存在同樣的參數設計問題，因此實施在 Kademlia 時沒有顯著的成效。

本文的改良方法在三個不同的網路擾動環境中進行實驗，與原版的 Kademlia 及前人所提出的共 4 種方法進行比較，皆以有限的通訊成本增長使得搜尋成功率顯著提升。與多數的方法相比，達成了更高的搜尋成功率；搜尋成功率略輸時，若調整 K 值，能在通訊成本較低的情況下達到更高的搜尋成功率。由於上述的實驗結果，認定本文所提出改良方法相較於原版以及前人的修改方法皆能達到更好的成效。



DOI:10.6814/NCCU202100314

| 方法 | 原版 | My Method | My Method (k=15) | Other-1 | Other-2 | Other-3 | Other-4 |
|---|---|---|---|---|---|---|---|
| 成功率 | 96.0% | 99.6% | 99.8% | 96.0% | 95.9% | 98.1% | 95.9% |
| 搜尋位置錯誤率 | 39.5 | 0.4% | 1.8 | 39.3 | 39 | 18.5 | 40.7 |
| 資料位置錯誤率 | 0 | 0% | 0 | 0 | 0 | 0 | 0 |
| 資料遺失率 | 0 | 0% | 0 | 0 | 1 | 0 | 0 |
| 搜尋時節點孤立率 | 39.5 | 0.4% | 1.8 | 39.5 | 38.8 | 18.5 | 40.6 |
| Ping (千次/小時) | 249.4 | 275.8 | 335.8 | 249.4 | 255.1 | 400.6 | 347.1 |
| FindNode (千次/小時) | 891.5 | 915.2 | 1347.3 | 891.5 | 953.8 | 1969.4 | 1139.8 |
| ReturnNode (千次/小時) | 267.2 | 280.9 | 389.1 | 267.2 | 276.4 | 468.9 | 336.1 |

表 4- 6 改良方法與其他方法比較結果(長時 20%，中時 40%)



DOI:10.6814/NCCU202100314

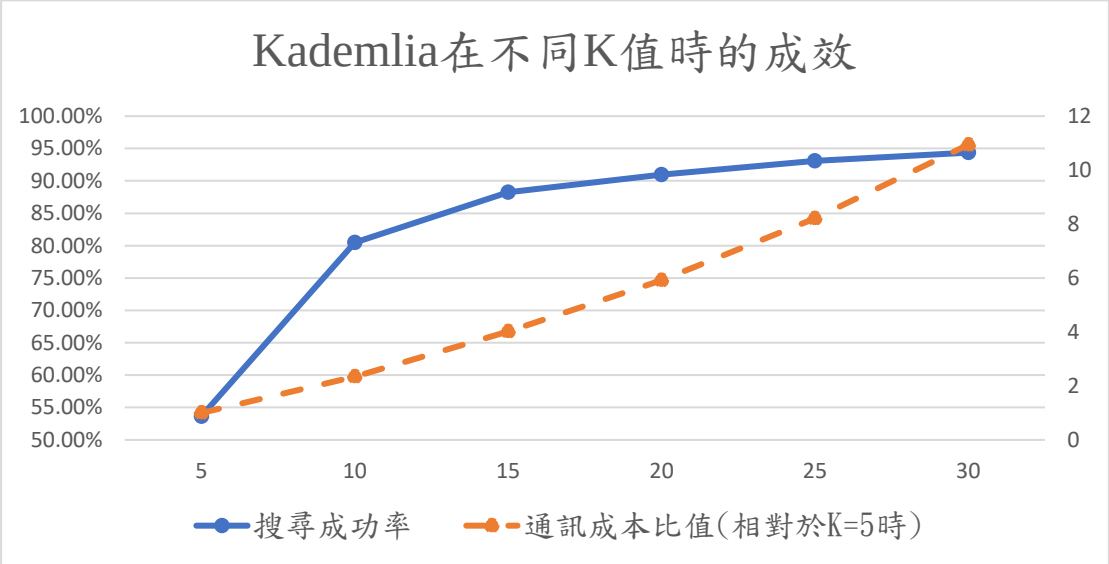
## 第三節　其他調整方式及其實驗結果

1. 調整 K 值

K 值表示 Routing Table 之中，每個 K-bucket 中所儲存的節點資訊數量，同時影響接點收到 FIND_NODE 或 FIND_VALUE 指令後所回傳的節點資訊數量，也代表每次儲存資料時的備份數量。增加 K 值將降低節點孤立的發生率，同時減少資料遺失率，應能使資料搜尋成功率大幅提升。

在長時-5%,中時-10%,短時-85%的網路擾動環境中測試原版 Kademlia 在不同 K 值下的搜尋成功率以及 FIND_NODE 指令傳送次數(通訊成本)。

圖 4- 1　Kademlia 調整 K 值成效




將 K 值從 5 調整至 10 時，搜尋成功率由 53.7%大幅提升至 80.5%，通訊成本由 K 值為 5 時的 1 倍增加至 2.3 倍；將 K 值從 25 調整至 30 時，搜尋成功率由 93.1%微幅提升至 94.4%，通訊成本由 K 值為 5 時的 8.2 倍增加至 10.9 倍增加。增加 K 值可使搜尋成功率顯著提升，效果會隨著 K 值增加而趨緩；通訊成本的增幅則會隨著 K 值增加而擴大。調整 K 值確實能提升搜尋成功率，但是考量到通訊成本時，調整 K 值並非最符合效益的方法。



DOI:10.6814/NCCU202100314

2. 在網路中加入超級節點

若在網路中加入 K(10)個超級節點(永遠不離線，能夠同時處理大量請求並且回覆)，作為節點孤立時的詢問對象，是否能再提升搜尋成功率？

將超級節點加入網路以取代長壽節點，在三種網路擾動環境中測試以超級節點代替長壽節點的成效。

| 網路擾動程度 | **長-5%,中-10%, 短-85%** | | **長-10%,中-20%, 短-70%** | | **長-20%,中-40%, 短-40%** | |
|---|---|---|---|---|---|---|
| 方法 | 長壽節點 | 超級節點 | 長壽節點 | 超級節點 | 長壽節點 | 超級節點 |
| 成功率 | 89.8% | 95.8% | 98.1% | 99.5% | 99.6% | 99.9% |
| 搜尋位置錯誤率 | 6.1% | 0.1% | 1.5% | 0.1% | 0.4% | 0.1% |
| 資料位置錯誤率 | 0.1% | 0.1% | 0% | 0% | 0% | 0% |
| 資料遺失率 | 4% | 4% | 0.4% | 0.4% | 0% | 0% |
| 搜尋時節點孤立率 | 6.2% | 0% | 1.5% | 0% | 0.4% | 0% |
| Ping (千次/小時) | 122.2 | 234.1 | 175.0 | 348.3 | 275.8 | 579.7 |
| FindNode (千次/小時) | 600.3 | 760.2 | 726.7 | 916.8 | 915.2 | 1117.9 |
| ReturnNode (千次/小時) | 144.5 | 193.1 | 193.4 | 262.4 | 280.9 | 404.2 |

表 4- 7 長壽節點與超級節點成效比較



DOI:10.6814/NCCU202100314

在三種網路擾動環境中，若存在超級節點作為節點孤立時的連線對象，將能使搜尋成功率進一步提升。相比於在純 P2P 的環境以不確定上線時長的節點作為節點孤立時的連線對象，若加入少量的伺服器成為的混合式 P2P 系統會更加可靠，效果在網路擾動嚴重時更加顯著。

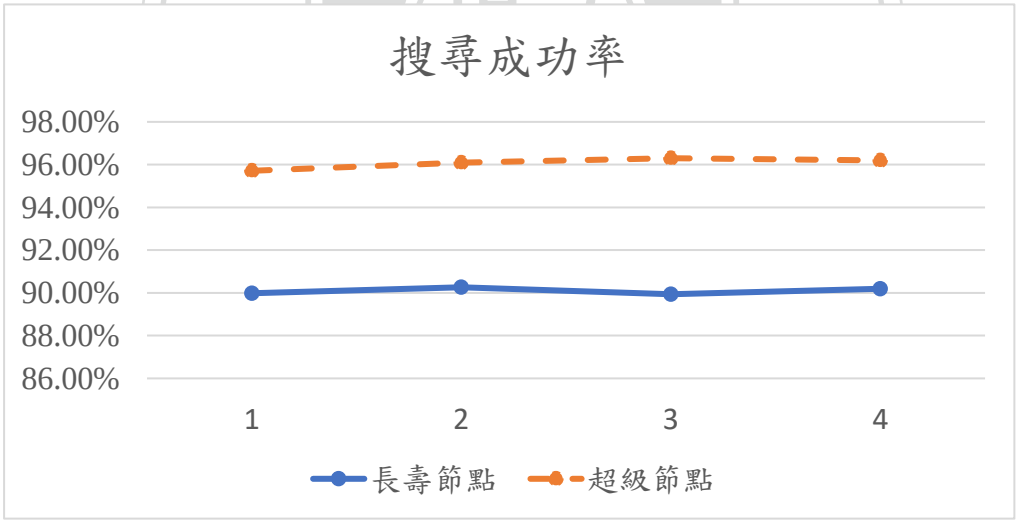
長壽節點能以不依靠伺服器的方式提升資料搜尋成功率，在網路擾動較輕微時能以約八成的通訊成本，達到與實施超級節點相近的資料搜尋成功率。

3. 長壽節點/超級節點的數量調整

調整長壽節點/超級節點的數量否會對搜尋成功率產生影響？

在長時-5%,中時-10%,短時-85%，K=10 的環境中測試調整長壽節點/超級節點的數量對於的搜尋成功率以及 FIND_NODE 傳送次數(通訊成本)的影響。

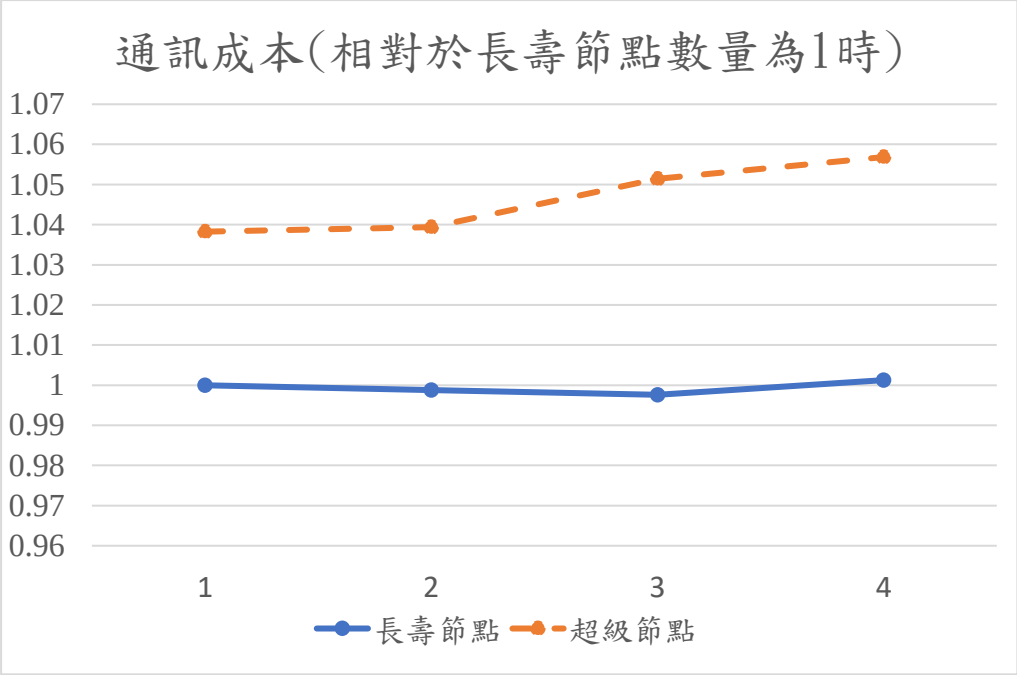
圖 4- 2 調整長壽節點/超級節點後的搜尋成功率






DOI:10.6814/NCCU202100314

圖 4-3 調整長壽節點/超級節點後的通訊成本




長壽節點/超級節點的數量與搜尋成功率之間無明顯關聯，可能是由於長壽節點會將預計在線最久的節點資訊保存，該節點在線機率較其他可接觸到的節點都高，若紀錄更多在線機率較低的節點並無實際作用；單一的超級節點能夠負擔所有其他節點的重新連線需求，同時沒有離線的風險，增加數量無顯著效用。



DOI:10.6814/NCCU202100314

# 第五章 結論與未來展望

## 第一節　結論

本文認為大量節點連線時間過短是資料搜尋失敗的主要原因，會造成節點孤立，進而使資料搜尋失敗，隨後提出長壽節點的修改方法，在不同網路擾動環境中皆使搜尋成功率顯著成長，並達成相較於其他方法更好的成效，此結果也驗證了先前對於搜尋失敗原因猜想的正確性。

網路擾動的情況變化多端，有無數個可能出現的情境，在觀察網路擾動的實際情況時，須處理多種可能發生的偏誤，要得到貼近事實且詳盡的網路擾動資訊並不容易。前人觀測並提供了網路擾動的分布情形，但網路擾動亦可能隨著不同的應用環境而改變，現在與網路連線較過往容易許多，手機成為人們主要的網路裝置，這些情況都會改變網路擾動，因此無法斷言本文所提出的修改方法在所有網路擾動環境中總是能達到較好的效果，但我相信研究中提出的 Kademlia 搜尋失敗原因及對應的修改方案能夠對將來的 Kademlia 相關應用系統或是其他 P2P 系統做出貢獻。

本文注意到 P2P 系統中存在 Routing Table 失效的問題並加以改善，未來的 P2P 系統開發人員或研究人員可對此部分多加留意，將提升 P2P 系統的可靠程度，並以此為基礎擴展 P2P 系統的應用可能。不論是搜尋引擎、公共資料庫、分散式運算服務、影音串流服務、資料驗證服務、通訊服務、社群服務等領域，或是目前尚未形成的新形態網路服務，P2P 應用都具備極大的發展潛能，期待將來能夠出現更注重個人隱私，保障人們言論自由以及視聽選擇權的網路世界。



DOI:10.6814/NCCU202100314

## 第二節　未來展望

本研究以長壽節點作為發生節點孤立時的對應方法，在節點意識到孤立現象後向長壽節點發出詢問請求，藉此與 P2P 系統重新連線，藉此減少大部分搜尋時的節點孤立現象，使搜尋成功率提升。由於估計離線時間未必準確，仍可能在發現搜尋時的節點孤立現象後無法與系統重新連線，導致搜尋失敗；發現節點孤立現象後與系統連線，並再次進行資料搜尋需要耗費較多時間。實施長壽節點後在高度網路擾動環境中 Kademlia 仍存在以上兩項不足之處。

在實驗中假設網路中存在少數用戶願意擔任超級節點(平均上線時間極長，且能夠接受並回復大量指令)，在此環境中超級節點容易因為平均上線時間長的特性被選為長壽節點，進一步減少因搜尋時的節點孤立現象形成的搜尋失敗，使得搜尋成功率再提升。此方法亦無法避免發現節點孤立到重新進行資料搜尋的期間過長所造成的延遲問題。

若能避免節點孤立現象發生就會減少復原時所造成的延遲，未來能夠對於如何預防節點孤立現象進行研究。以類似邊緣伺服器的小型伺服器對於具有不同前綴的 ID 進行管理。在小型伺服器的管理下，將可掌握節點來去網路的資訊，在節點孤立發生前更新 Routing Table；進行 Lookup 時，若發起的節點與 Key 過於遙遠，可先向小型伺服器聯絡，藉由小型伺服器間通訊找到更接近 Key 的節點後，再進行 Lookup 程序，將能減少接近 Key 所造成的延遲。



DOI:10.6814/NCCU202100314

# 參考文獻


[1] aMule, Retrieved January 2021, from: amule.org

[2] Bittorrent, Retrieved January 2021, from: https://www.bittorrent.com/

[3] D.Stutzbach and R.Rejaie. (2006). Improving Lookup Performance Over a Widely-Deployed DHT. *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*. doi:10.1109/INFOCOM.2006.329

[4] D. Stutzbach and R. Rejaie. (2006). Understanding Churn in Peer-to-Peer Networks. *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, 189~202. doi:10.1145/1177080.1177105

[5] Gnutella, Retrieved January 2021, from: https://en.wikipedia.org/wiki/Gnutella

[6] J. Falkner, M. Piatek, J. John, A. Krishnamurthy, and T. Anderson. (2007). Profiling a Million User DHT. *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. doi: 10.1145/1298306.1298325

[7] Kad network, Retrieved January 2021, from: https://en.wikipedia.org/wiki/Kad_network

[8] Kang H, Chan-Tin E, Hopper N, Kim Y. (2009). Why kad lookup fails. *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*, 121–130. doi: 10.1109/P2P.2009.5284547

[9] Liu B, Wei T, Zhang J, Li J, Zou W, Zhou M. (2012). Revisiting why kad lookup fails. 2012 IEEE 12th International Conference on Peer-to-Peer Computing (P2P), 37–42. doi: 10.1109/P2P.2012.6335808

[10] Liu Z, Yuan R, Li Z, Li H, Chen G. (2006). Survive under high churn in structured P2P systems: Evaluation and strategy. *Computational Science – ICCS 2006*, 404−411. doi: 10.1007/11758549_58




DOI:10.6814/NCCU202100314

[11] M. Steiner, E. W. Biersack, and T. En-Najjary. (2007). Actively Monitoring Peers in KAD. *IPTPS 2007*.

[12] Maymounkov P, Mazières D. (2002). Kademlia: a peer-to-peer information system based on the xor metric. *IPTPS '01: Revised Papers from the First International Wor*kshop on Peer-to-Peer Systems, pp 53–65.

[13] O. Herrera and T. Znati. (2007). Modeling Churn in P2P Networks. 40th Annual Simulation Symposium (ANSS'07). doi: 10.1109/ANSS.2007.28

[14] Official eMule-Board, Retrieved January 2021, from: forum.emule-project.net





DOI:10.6814/NCCU202100314