國立臺北科技大學

資訊工程系碩士班

碩士學位論文

改善 FruityMesh 藍牙低功耗網狀網路傳輸效能的設計與實作

Design and Implementation of Performance Enhancements in FruityMesh Based Bluetooth Low Energy Mesh Networks

研究生：陳柏勳

指導教授：吳和庭博士

中華民國一百一十四年七月

「學位論文口試委員會審定書」掃描檔

審定書填寫方式以系所規定為準，但檢附在電子論文內的掃描檔須具備以下條件：

1. 含指導教授、口試委員及系所主管的完整簽名。
2. 口試委員人數正確，碩士口試委員至少 3 人、博士口試委員至少 5 人。
3. 若此頁有論文題目，題目應和書背、封面、書名頁、摘要頁的題目相符。
4. 此頁有無浮水印皆可。

審定書不用頁碼

摘要

關鍵詞：藍牙低功耗、物聯網、FruityMesh、BLE Mesh、網路拓撲、傳輸壅塞、重傳率、封包傳遞成功率

藍牙低功耗（Bluetooth Low Energy, BLE）有省電及低成本的特性，使得藍牙技術在物聯網（Internet of Things, IoT）中占據重要的角色。在物聯網的應用中，大量使用無線感測網路（Wireless Sensor Networks, WSN），會在環境中分布建立許多的節點，而節點不只有當作感知器測量環境的數據，常常還要當作中繼節點，負責轉傳發送端與目的端之間的封包。最終，將所有量測的數據匯集到 Sink 節點，以監控所有的節點數據，將數據儲存後，進行分析後並做出適當的處理，也可以透過分析數據預測環境的變化，並提前做出適當的處理。本論文針對 FruityMesh 網路建立流程進行改善，讓 Sink 節點在網路建立完成後成為整個網狀網路的根節點，並且確保 Sink 節點斷線後重新連線時，仍然是根節點的角色。如此一來，所有非 Sink 節點在傳送封包至 Sink 節點時，只需要往父親節點傳輸封包即可，有效的減少節點發送及轉發的次數。此外，本論文也透過調整 BLE 相關參數，緩解大量封包匯聚至根節點所造成傳輸壅塞的問題，進一步提升了整體 Mesh 網路的封包傳遞成功率（Packet Delivery Ratio, PDR）、降低了封包傳輸延遲及重傳率，進而改善網路效能。

Abstract

Keyword: Bluetooth Low Energy,Internet of Things,FruityMesh,BLE Mesh,Network Topology,Transmission Congestion,Retransmission Rate,Packet Delivery Ratio

Bluetooth Low Energy （BLE）is characterized by its low power consumption and cost-effectiveness, making it a key technology in the development of the Internet of Things （IoT）. In IoT applications, Wireless Sensor Networks （WSNs）are widely used, where numerous sensor nodes are distributed throughout the environment. These nodes not only collect environmental data but often function as relay nodes, responsible for forwarding packets between the source and destination. Ultimately, all collected data is aggregated at the Sink node, which serves as a central point for monitoring, storage, and analysis. This enables the system to make informed decisions and even predict environmental changes based on the analyzed data. This thesis proposes an enhancement to the FruityMesh network formation process, ensuring that the Sink node becomes the root node of the mesh network once the network is established. Furthermore, it guarantees that the Sink node retains its root node role even after disconnection and reconnection. With this configuration, non-Sink nodes can transmit packets toward the Sink node by simply forwarding them to their parent node, effectively reducing the number of transmissions and retransmissions required. Additionally, by tuning specific BLE parameters, the proposed method alleviates network congestion caused by high packet traffic toward the root node. The experimental results show that the proposed improvements increase the Packet Delivery Ratio （PDR）, reduce transmission delay and retransmission rates, and ultimately enhance overall network performance.

致謝

所有對於研究提供協助之人或機構，作者都可在誌謝中表達感謝之意。

目錄

圖目錄

表目錄

第一章　緒論

1.1　研究背景

科技的快速進步，讓人們的生活更加便利，物聯網（IoT）的應用已經與日常生活密不可分，包含了醫療及工業的應用，無所不在，[1] 根據日商環球訊息有限公司（GII）調查，物聯網（IoT）市場規模預計從 2024 年到 2029 年，將從 1.17 兆美元增加至 2.37 兆美元，年均複合成長率（CAGR）為 15.12%，如圖1.1所示。

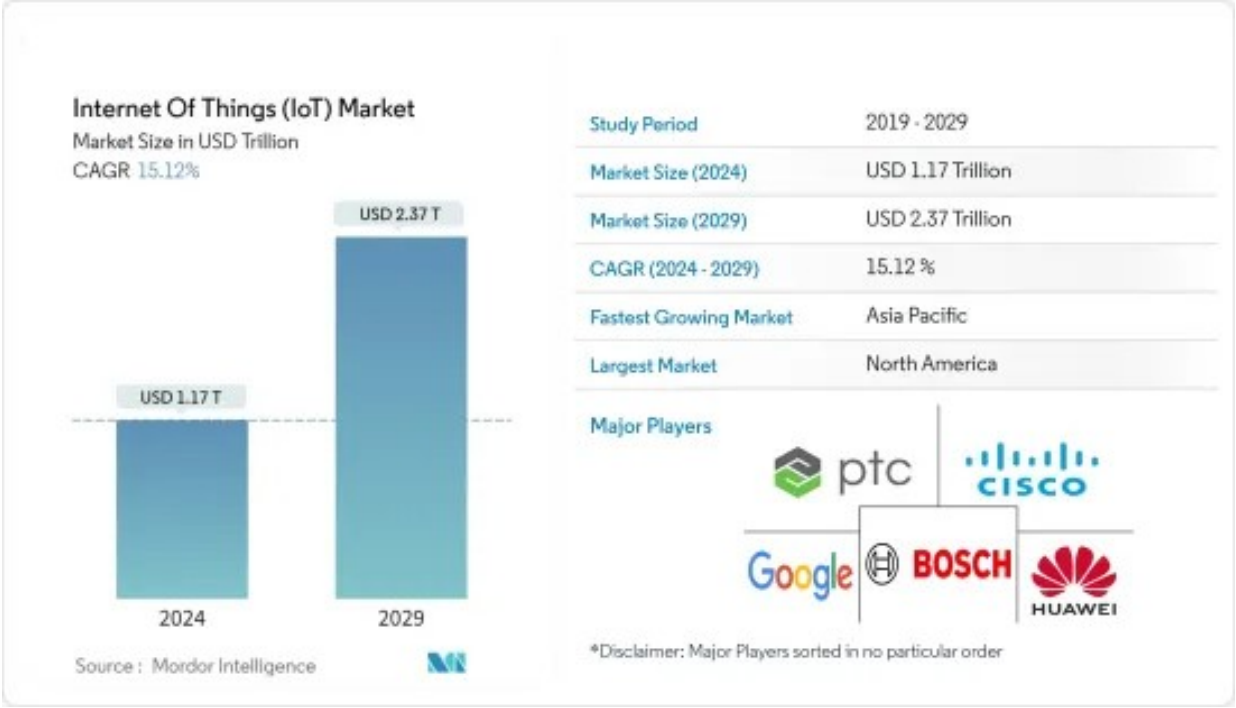

圖 1.1 IoT 市場規模評估 [1]

2010 年 6 月藍牙技術聯盟（Bluetooth Special Interest Group）提出了低功耗藍牙（Bluetooth Low Energy, BLE），BLE 省電及低成本的特性，使得藍牙技術在物聯網（IoT）的應用種佔據了不可或缺的角色，例如: 目前市面上的無線設備包括藍牙耳機、藍牙鍵盤及藍牙滑鼠。物聯網（IoT）的應用中，大量使用無線感測網路（Wireless Sensor Networks, WSN），會在環境之中分布許多的節點，而節點不只有當作感知器測量環境的數據，常常還要當作中繼節點，轉傳發送端與目的端的封包，最終將所有量測的數據匯

集到終端節點，以監控所有的節點數據，將數據儲存後，進行分析後並做出適當的處理，也可以透過分析數據預測環境的變化，並提前做出適當的處理。

藍牙網狀網路（Bluetooth Mesh）架構的實現，讓 BLE 更具有可靠性（Reliability）及擴展性（Scalability），可以允許多個 BLE 相互連接並形成網狀結構，讓封包可以在多個裝置或節點之間進行傳輸，讓傳輸距離不會受到單一裝置的傳輸範圍限制，解決了節點之間裝置連接數量的限制以及傳輸距離不足的問題，在物聯網（IoT）的應用，例如: 智慧建築、智慧工業、智慧城市、智慧家庭.. 等等，BLE 都已經扮演重要也不可或缺的角色。

在物聯網（IoT）與藍牙網狀網路（Bluetooth Mesh）中，對整個系統架構做出適當的評估，在不影響裝置效能的情況下，設計多個藍牙裝置之間的分流機制，因為在整個系統中，流量可能會有所起伏，為了讓每個裝置可以有一樣的傳輸品質，且系統可以發揮最好的吞吐量。

1.2 研究動機與目的

根據文獻 [2] 中針對 Multi-Hop 的提出 DOT 及 DOST 排程設計，研究發現在原生 FruityMesh 上封包傳輸是採用廣播的方式，向相鄰的所有節點發送封包，這種設計雖然簡單，卻容易導致大量冗餘封包的產生，這會導致節點之間傳輸不必要的封包，最終導致整個 BLE Mesh 出現廣播風暴。

[2] 作者針對 FruityMesh 傳輸時產生的廣播風暴，提出了 DOT（Destination Oriented Transmission）排程的機制，將 BLE Mesh 導入樹狀結構，對整個系統進行分層管理，讓封包傳輸只會向正確的路徑上傳輸，避免了廣播風暴的產生，但是作者在研究過程中，也發現 BLE Multi-Hop 網狀網路節點會擁有 Master 以及 Slave 兩種角色，而在同一個時間下，節點同時會是 Master 和 Slave 的狀態，而這種狀態上的衝突會導致的封包傳輸產生遺失現象。

[2] 作者為了解決角色衝突的問題，在 DOT 的架構下增加啟用以及禁用的機制，提出 DOST（Destination Oriented Switch Transmission）的排程設計，利用 TDMA 讓裝置在同一個時間只會扮演 Master 或 Slave，解決在同一個時間點節點可能會因為扮演 Master 及 Slave 角色衝突而導致的封包傳輸遺失的問題；在研究 [2] 中作者在 FruityMesh 中導入

DOST 設計，改善了大部分的封包傳輸延遲以及封包重傳率，但仍存在些許效能瓶頸，因為在 FruityMesh 中，所有節點的封包傳輸都是透過廣播的方式進行，這會導致在多個節點同時傳輸封包時，可能會產生封包碰撞的問題，導致封包傳輸延遲增加。

此外，文獻指出，若要在 DOST 架構下進一步提升效能，需確保拓樸建立時，Sink 節點為整體網路的根節點。然而原研究並未針對拓樸控制機制提出具體解法。若未能確保 Sink 位於樹根，當其斷線並重新加入網路時，可能無法恢復為原先的根節點角色，進而影響整體傳輸路徑的穩定性與效能。

在 [2][3] 中，針對了 single hop mesh 及 multi hop mesh 的網路拓樸進行探討，當使用合適的封包連接參數可以有效降低封包傳輸延遲及重傳率，但是數據實際上傳輸時，每個封包大小並不固定，[4] 中，探討了不同封包大小可以算出最適合的連接參數，在傳輸過程動態調整連接的 CE(Connection Event) 及 CI(Connection Interval)，達到更好的吞吐量。

本計畫將研究中在 DOT 的排程設計下，仍然會有些許因為封包遺失，而產生的封包重傳的問題，以及在藍牙網狀網路（Bluetooth Mesh）架構，執行 TDMA 的排程機制，結合調整連線數據，透過修改 Connection Interval 數值，確保傳輸品質及確保系統可以產生最大吞吐量，並修改 FruityMesh 拓樸方法，讓 BLE Mesh 建立的過程確保 Sink 點在整個樹狀結構的根節點，也確保 Sink 點如果斷線後重新連線後依然是整個 BLE Mesh 樹狀結構的根節點。

1.3 論文架構

本論文共分為五章，第一章說明本研究的背景與動機，並闡述本論文所要解決的問題與研究目的。

第二章整理並探討本研究所依據的基礎知識與技術背景，包括藍牙低功耗（BLE）通訊技術、藍牙網狀網路（BLE Mesh）架構、BLE 中的排程與連線機制（如 CI、CE）、以及目的地傾向傳輸策略。此外，也介紹本研究所採用的開發平台，包含 Nordic Semiconductor 所推出的 nRF52840 晶片與 SoftDevice 堆疊，以及基於該硬體的 FruityMesh 開源框架，並進一步說明 FruityMesh 的 Cluster 與 Self-Healing 機制。

第三章分析現有 FruityMesh 在拓樸建立與封包傳輸上可能出現的問題。接著提出改

良的拓樸建立演算法與設計，包括以 Master 或 Slave 身分進行最佳連線選擇，以及改善自我修復（Self-Healing）行為的拓樸重建機制。此外，本章亦設計改善封包傳輸效能的對應策略，以確保在多節點環境下依然能維持高效通訊。

第四章說明本研究之實驗設計與評估方式，分別建置實體測試平台與模擬平台（CheerSim）以模擬不同節點規模下的網路拓樸生成與封包傳輸行為。實驗部分以多組節點數模擬進行測試，並設計多項效能評估指標，包括拓樸建立時間、自我修復時間、平均 HopsToSink 數值、封包傳輸延遲、封包抵達率與封包重傳率，藉此全面評估所提出機制的實際效能表現。

第五章總結本研究的成果與貢獻，並探討未來在 BLE Mesh 網路設計中可延伸的研究方向與改進空間，期望能為低功耗網狀通訊領域提供可參考之實作架構與演算法設計依據。

第二章　相關技術與背景

2.1　藍牙低功耗技術

藍牙技術聯盟（Bluetooth Special Interest Group）在 2010 年 6 月發布了可以短距離數據交換和低功耗的藍芽低功耗技術（Bluetooth Low Energy, BLE）。而藍芽低功耗技術被發布後，就被物聯網（IoT）廣泛的應用，包括了家庭娛樂、醫療保健、運動健身、安防以及信標等領域。

藍牙低功耗技術（Bluetooth Low Energy, BLE）是一種功耗極低的技術，這一個技術讓裝置在大部分的時間都在休眠模式，只有在需要使用該裝置時，才會快速喚醒進行工作，這讓 BLE 裝置僅需要一顆鈕扣電池就可以運作數月甚至數年之久，這讓 BLE 生產成本更低，且保留了傳統藍芽（Classic Bluetooth）類似的通訊範圍，且一樣相容於手機、平板電腦等設備。

BLE 運作在 2.4G 的 ISM 頻段，利用 FDMA（Frequency Division Multiple Access），將 2402MHz 至 2480MHz 分成 40 個 Channel，又將這些 Channel 又分成兩種傳輸模式，廣播模式（Advertising Mode）及連線模式（Connection-Oriented Mode），其中廣播模式使用了 Channel 37、Channel 38、Channel 39，工作頻率分別是 2403MHz、2426MHz、2480MHz，剩餘的 37 個 Channel 為連接模式使用，如圖2.1所示。

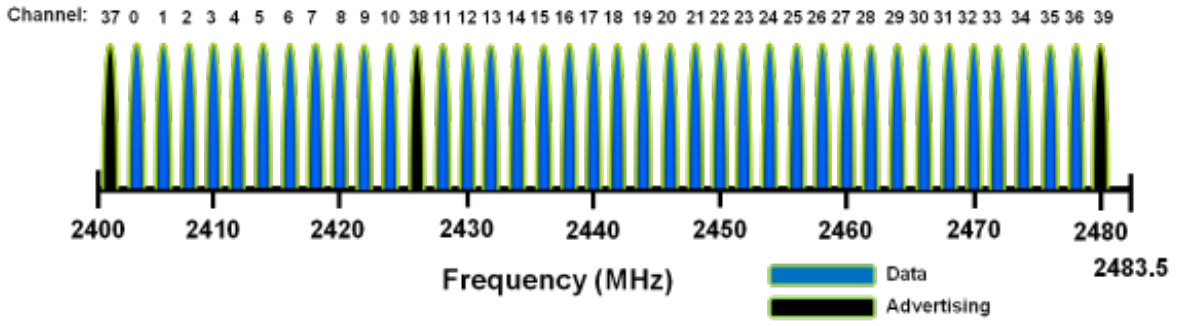

圖 2.1 廣播頻道與資料頻道示意圖 [5]

廣播模式和連接模式的運作機制由 BLE 的控制層（Controller Layer）狀態機管理，包括 Standby（等待）、Advertising（廣播）、Scanning（掃描）、Initiating（初始化）、

Connection（連接）五種狀態，如圖2.2所示。

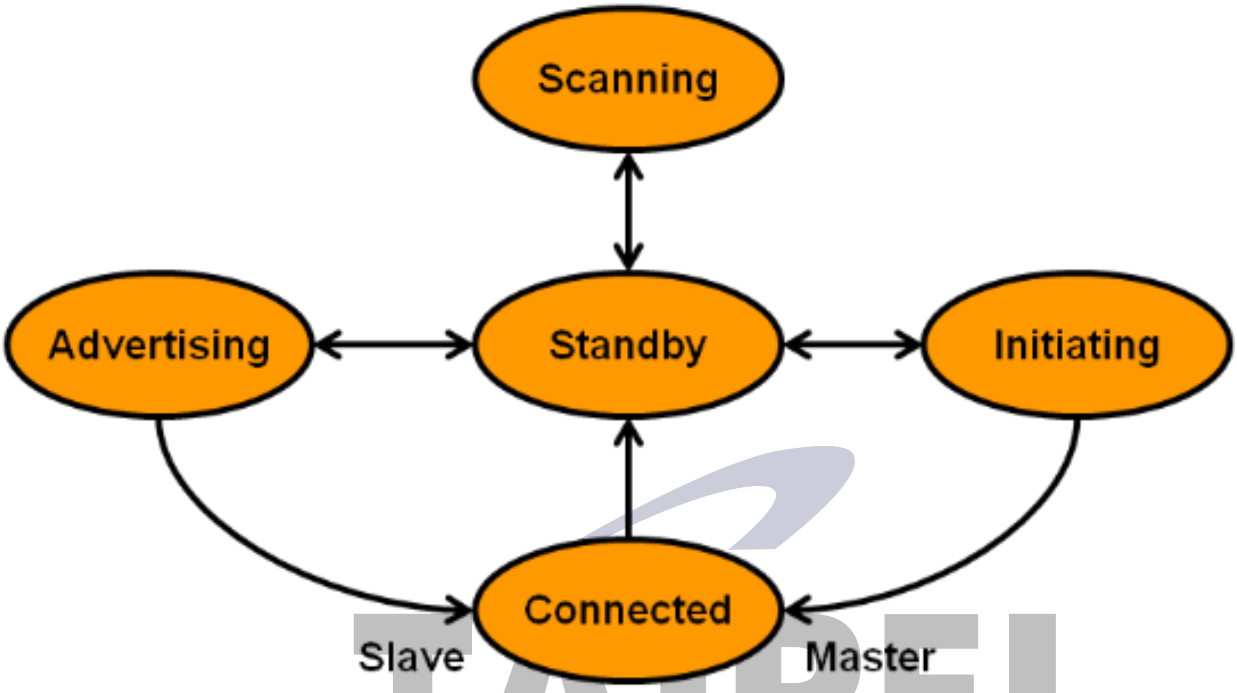

圖 2.2 Controller layer 狀態機示意圖 [5]

2.1.1 廣播模式

在廣播模式（Advertising Mode）中，會使用 Channel 37、Channel 38、channel 39 這三個廣播頻道，主要用於掃描裝置、建立通訊頻道和廣播的傳輸，其中廣播者（Advertiser）是由待機狀態（Standby Status）進入廣播狀態（Advertising Status），廣播者會在三個廣播頻道輪流發送廣播封包，讓掃描者（Scanner）可以檢測到其存在，並提供基本的數據，例如: 裝置名稱或狀態。

掃描者（Scanner）是由待機狀態（Standby Status）進入掃描狀態（Scanner Status），掃描者會輪流掃描三個廣播頻道的廣播封包，已接收範圍內的所有廣播的資訊，掃描收集數據後，準備與廣播設備建立連接，如圖2.3所示。

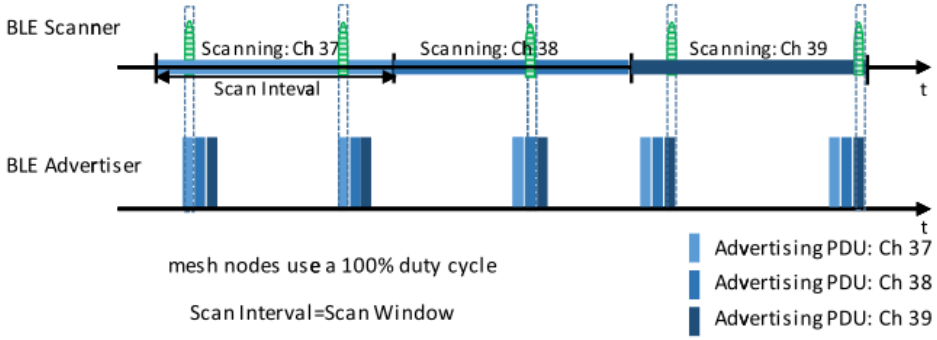

圖 2.3 廣播模式示意圖 [6]

2.1.2 連接模式

連接模式（Connection-Oriented Mode）中，設備需要首先透過廣播模式建立連接，其中廣播者（Slave）負責發送連線請求的廣播封包，而掃描者（Master）檢測到廣播封包後，從 Standby 進入 Scanning，再進入 Initiating 狀態，開始與廣播設備握手（Handshake）。

當握手成功後，雙方進入 Connection 狀態。連接建立後，Master 與 Slave 可通過剩餘的 37 個數據頻道進行數據傳輸；Master 負責與多個 Slave 的連線管理，分配專屬的時間槽（Time Slot）給各連接設備，即使沒有數據需要傳輸，系統仍會保留固定的時間槽以確保系統的穩定性，避免通訊衝突。圖2.4為連線模式示意圖。

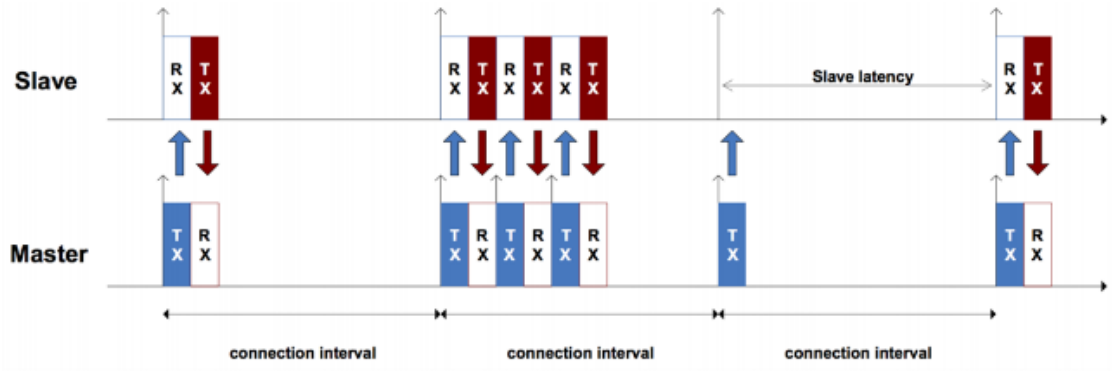

圖 2.4 連線模式示意圖 [6]

2.1.3 BLE 排程機制

廣播者（Advertiser）和掃描者（Scanner）會在建立連線後，進行數據的交換，當掃描者在三個掃描頻道掃瞄並偵測到廣播者發送的封包後，掃描者會在約 150 微秒（T_{IFS}）後回應一個 CONNECT_IND 封包。CONNECT_IND 封包包含了多項管理連接的參數，例如：影響錨點時間（Anchor Point, AP）的 WinSize 以及 WinOffset。

建立連線之後，Connection Interval （CI）、Anchor Point （AP）和 Connection Event（CE）對於裝置之間的排程機制影響非常的大，在一個 Connection Interval（CI）的時間內，一個 Slave 裝置只會與 Master 裝置有一個 Connection Event（CE）傳輸時間來進行資料的交換，所以 Connection Interval（CI）的時間影響 Connection Event（CE）的發生頻率，這直接影響到整個系統的效能及吞吐量，圖2.5表示了 Master 連接建立過程和連接事件。

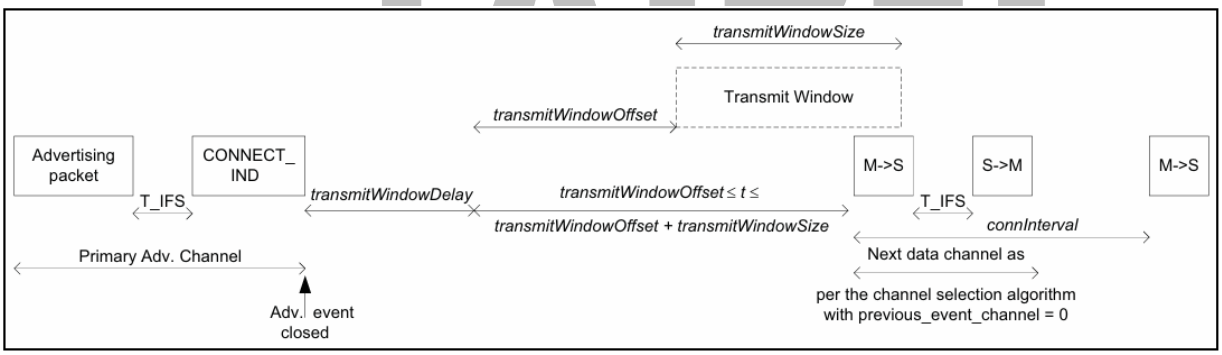

圖 2.5 Master 連接建立過程和連接事件 [7]

CONNECT_IND 封包會在 t_{ind} 時間內完成傳輸，Master 會在2.1式至2.2式的時間內設置第一個錨點（Anchor Point, AP），其中 transmitWindowDelay 通常為 $1.25\,ms$，用來同步 Master 與 Slave。transmitWinOffset 與 transmitWinSize 分別由2.3式與2.4式算出。WinOffset 代表錨點（Anchor Point, AP）的偏移量；WinSize 代表錨點（Anchor Point, AP）可能發生的時間範圍 [8]。

$$t_{ind} + transmitWindowDelay + transmitWinOffset \tag{2.1}$$

$$t_{ind} + transmitWindowDelay + transmitWinOffset + transmitWinSize \tag{2.2}$$

$$transmitWinOffset = WinOffset \times 1.25\,ms \tag{2.3}$$

$$transmitWinSize = WinSize \times 1.25\,ms \tag{2.4}$$

2.2 藍牙網狀網路

藍牙網狀網路（Bluetooth Mesh）是由藍牙技術聯盟（Bluetooth Special Interest Group）於 2017 年發布的標準，旨在解決傳統藍牙技術在物聯網（IoT）應用中的限制。藍牙網狀網路允許多個 BLE 裝置之間建立一個可靠且可擴展的網狀結構，這使得裝置之間可以進行多跳傳輸，從而擴大了傳輸距離和覆蓋範圍。

Bluetooth Mesh 是基於藍牙低功耗（Bluetooth Low Energy, BLE）的網狀網路技術，是多個裝置進行多對多（many-to-many）連接的拓樸技術，可以將資料從 BLE Mesh 中的其中一個節點，發送至 BLE Mesh 中的任何一個節點，且任兩的節點間的傳輸路徑，並不會只有一條，因為這種多重路徑（multi-path）的特性，讓 Mesh 中如果有一個節點故障，也部會因此癱瘓整個系統。此技術有效解決了 BLE 在長距離和多設備連接上的限制，擴展了 BLE 的應用範圍，特別適用於無線感測網路（Wireless Sensor Networks, WSN）和物聯網（IoT）環境。

Bluetooth Mesh 保有 BLE 的優點，並改善 BLE 傳輸範圍有限的問題，Bluetooth Mesh 的網路架構分為 Flooding 模式及 Routing 模式，Flooding Mesh 是大多數 BLE Mesh 協議採用的傳輸模式，利用廣播方式傳送訊息，無需建立節點間的連接，節點將訊息以廣播模式傳遞給通訊範圍內的所有節點，節點收到訊息後再次廣播，直到訊息傳遞至目標節點為止。

Routing Mesh 使用連接模式，節點需在訊息傳輸前建立節點間的連接，確保訊息傳遞有序且可靠。節點之間先建立連接，資料通過預設路徑逐步傳遞至目的節點，並根據

需求動態調整路由，因為資料傳輸通過固定路徑，減少碰撞和干擾，讓傳輸更穩定。也因為訊息傳遞通過固定的路徑，降低重複廣播次數，節省了不必要的電能消耗。Routing Mesh 資料傳輸的方式確保資料完整傳遞，即使網路繁忙也能正常運作。相對的缺點也顯而易見，因為資料傳遞時，節點間需建立連接，設計上比較複雜而增加實現難度。且建立連接需耗費額外時間，當路徑上的節點完成所有連接後才會開始傳輸，所以網路啟動比較慢。也會因為接點間連接路徑較長，讓資料傳遞時間增加，最後產生較高的傳輸延遲。

2.3 分時多工與 BLE 中的 CI 與 CE 機制

分時多工（Time Division Multiple Access, TDMA）如字面上的意義，它一種將時間分割的通訊技術，讓多個裝置可以高效的共享同一傳輸介質進行傳輸。TDMA 將時間劃分為多個時槽（Time Slots），每個裝置只能在指定的時槽內進行傳輸，這確保了同一時刻，只有一個裝置進行傳輸，其他的裝置在同一個時間下（相同 Time slot），都處於等待的狀態，可以確保訊號穩定且不重疊，有效避免了訊號重疊和干擾而導致的數據丟失。

在 BLE 中，TDMA 的理念被具體實現在連線模式下的 CI 與 CE 設計中。BLE 通訊雙方在建立連線後，會協商出一個固定的 CI，作為接下來通訊週期的基本單位。每個 CI 之內，雙方僅在特定的 CE 中進行資料交換，其餘時間則處於睡眠或非活躍狀態。

這樣的設計與 TDMA 的概念密切相關：每個 BLE 連線設備實際上都在預定的時槽（即 CE 時間內）進行資料交換，彼此間在時間上分離，以避免干擾與資源衝突。尤其在多裝置共存環境中，例如一個主裝置（Central）與多個從裝置（Peripheral）連線時，主裝置會根據各連線的 CI 設計，輪流與不同的裝置進行資料交換，實現一種類似 TDMA 的排程與存取控制機制。

2.4 目的地傾向傳輸

在 FruityMesh 的傳輸機制中，訊息從一個節點發送至中繼節點時，中繼節點會廣播訊息給所有相鄰節點，相鄰的節點又會在廣播給其相鄰的節點，直到目標節點，而廣播風暴會造成節點間多餘不必要的資料傳輸，而增加網路的負擔。為了解決廣播風暴問題，

目的地傾向傳輸（Destination-Oriented Transmission, DOT）[2] 機制，通過引入方向性和目的傾向的傳輸策略，優化封包的傳輸過程，減少網路負載與封包重傳，提升傳輸效率與穩定性。

DOT 機制利用樹狀架構中的 Level 層級概念，為每個節點計算其所屬層級，使用 FIND_LEVEL 封包進行層級標記，傳輸方向封包只需傳遞至 Level 較低的節點，避免無意義的廣播。DOT 在中繼節點限制封包的傳輸方向，僅向 Level 較低的節點傳輸封包，這種方式有效減少了網路流量，降低碰撞率，並提升了傳輸可靠性。

2.5 目的地傾向切換傳輸

目的地傾向切換傳輸（Destination Oriented Switch Transmission, DOST）[2] 機制進一步解決角色身份重疊與傳輸重疊的問題，通過啟用與禁用連線的方式，降低網路負載並避免封包重傳，解決 BLE 網狀網路節點可能因同時扮演 Master 和 Slave 角色衝突而導致的封包傳輸遺失現象。

啟用與禁用功能，節點傳輸 INIT_STATE 封包，用於標記和控制各連線的啟用或禁用狀態，確保所有節點對於連線的處理具有一致性，節點在某一連線傳輸封包時，禁用其他連線以避免通訊重疊，每個 Connection Interval 結束後，切換連線的啟用與禁用狀態，雖然可能導致封包延遲一個 Connection Interval，但有效降低了網路負載，提升傳輸效率，如圖2.6所示。

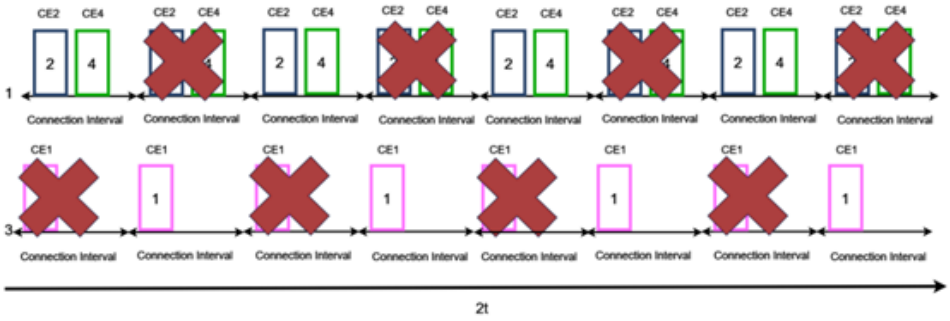

圖 2.6 傳輸機制的節點時序示例 [2]

2.6 Nordic Semiconductor nRF52840

nRF52840 是 nRF52 系列中最先進的成員，專為應對需要協議並行處理、豐富外設和複雜應用挑戰的需求而設計。它提供充裕的快閃記憶體與 RAM 空間，能滿足高性能應用的需求。nRF52840 支援多協議並行運作，包含低功耗藍牙、藍牙 Mesh 網狀網路、Thread、Zigbee、IEEE 802.15.4、ANT 及 2.4 GHz 專有協議，讓其在多種應用場景中均表現出色。

此 SoC 採用 32 位元 ARM® Cortex™-M4 處理器，具備浮點運算單元，主頻達 64 MHz。內建 NFC-A 標籤，可簡化裝置配對或用於支付應用。此外，晶片搭載 ARM TrustZone® CryptoCell 安全加密單元，能在 CPU 之外高效處理加密演算法，提供強大的安全性。

nRF52840 也整合多種數位周邊和介面，包括高速 SPI 與 QSPI，能連接外部快閃記憶體與顯示裝置；PDM 與 I2S 接口用於連接數位麥克風與音訊設備。此外，內建全速 USB 控制器，支援資料傳輸功能，亦可作為電池充電的電源來源。

透過先進的片上自適應電源管理系統，nRF52840 實現了極低功耗，適合各類高效能且低能耗的應用需求，所以選用 Nordic Semiconductor 的 nRF52840-DK 開發板 [9] 作為本次的硬體設備，如圖2.7。

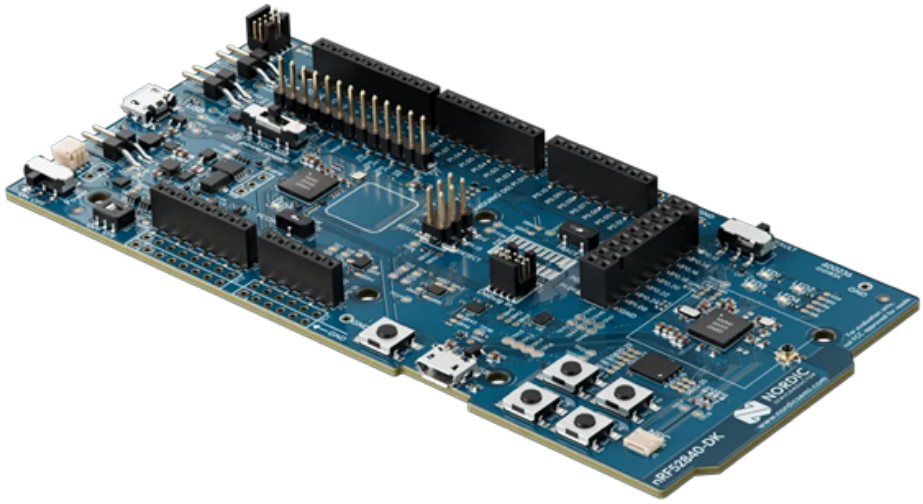

圖 2.7 nRF52840-DK 開發板

2.7 FruityMesh 開發平台介紹

FruityMesh [10] 專為 BLE Mesh Network 設計的開源通訊協定框架。該平台主要針對 Nordic Semiconductor 所推出之 nRF52 系列晶片（例如 nRF52832、nRF52840）進行最佳化，提供節點之間低功耗、穩定且具擴展性的無線通訊能力，廣泛應用於物聯網（IoT）場景。

FruityMesh 架構底層結合 Nordic SoftDevice 的藍牙堆疊，搭配實時任務管理與連線資源控管模組，能夠有效分配 Mesh 內部的連線插槽（Connection Slots）與角色權限。此外，其內建的 Clustering 與 Routing 機制可支援點對點封包傳遞與多跳路由（Multi-hop），達成資料有效分散與收斂的設計目標。

本研究即基於 FruityMesh 為開發基礎，進一步修改其拓樸建立、排程控制、連線優化與自我修復（Self-Healing）機制，以因應實際多節點部署中所可能遭遇的延遲、壅塞與節點斷線等問題。

2.7.1 Nordic SoftDevice

SoftDevice 是由 Nordic Semiconductor [9] 提供的一組預編譯、經過認證的無線通訊協議堆疊 (Protocol Stack)，專為 nRF 系列（如 nRF52840、nRF52832）等 Bluetooth Low Energy(BLE) 與 ANT 無線晶片設計。SoftDevice 主要負責 BLE 通訊協議的完整實作，並且以軟體庫的形式提供，方便嵌入式開發者在不需理解底層協議的情況下進行無線應用開發。

2.7.2 FruityMesh Cluster

在 FruityMesh [10] 架構中，Cluster（叢集）機制扮演了關鍵的角色。該機制允許網路中的節點根據功能或地理位置進行分組，以降低整體網路中不必要的廣播與封包重傳，進而提升通訊效率與穩定性。每個 Cluster 可被視為一個子網，其內部節點之間擁有穩定的連線關係，並可視需求與其他 Cluster 建立通訊橋接，使整體網路具備良好的可擴展性與模組化管理能力。

當一個節點首次啟動時，會主動進入 High Discovery 狀態，並開始廣播 JOIN 封包以

尋找可連接的節點。隨著時間推進，鄰近節點會逐步聚合形成數個 Cluster，並根據連線策略逐步合併成為一個完整的網狀結構。在 FruityMesh 的預設演算法中，節點在選擇連線對象時會優先考量 Cluster 的節點數量，傾向加入節點較多的 Cluster。因此，整體網路會自然形成「大 Cluster 併吞小 Cluster」的現象，最終形成一個包含所有節點的 Spanning Tree 拓樸結構。

2.7.3 FruityMesh Self-Healing

為強化網路穩定性，FruityMesh [10] 同時設計了 Self-Healing（自我修復）機制。當網路中某個節點因掉電、移動或干擾等原因導致斷線，系統會自動啟動恢復流程。具體而言，斷線的節點會立即廣播連線請求封包至其通訊範圍內的所有鄰近節點。一旦鄰近節點接收到該封包，便會與之進行連線協商程序，使該孤立節點得以重新加入原有網路拓樸中。

此機制基於 FruityMesh 支援的多跳傳輸設計，確保即使部分節點失效，網路仍可自動尋找可行的替代路徑傳遞資料。Self-Healing 機制大幅提升了 BLE Mesh 系統在實際部署環境中的容錯性與可靠性，亦減少了人工維護與手動重連的需求，對於物聯網應用中的大規模部署具有重要意義。

第三章　BLE Mesh 拓樸建立機制設計

3.1　問題分析

3.1.1　BLE Mesh 拓樸建立問題

在原生 FruityMesh 架構中，節點的建立與連線並未針對資料匯集或匯出端進行特別設計，因此整體網路並無明確的 Root 節點。其封包傳輸方式主要採用廣播機制，即每當節點產生封包時，會向所有相鄰節點廣播傳送，期望藉由鄰近節點的重傳將封包推進至目的地。雖然此方法具備一定的自我修復能力，但在多節點、大範圍的 Mesh 網路中，極易導致封包重複傳送與碰撞，進而產生所謂的「廣播風暴」現象，嚴重影響整體網路效能與穩定性。

為了解決此問題，[2] 研究中引入 DOT 機制，將原先的無向廣播傳輸改為目的導向的封包路由方式，透過樹狀拓樸的建立使資料傳輸路徑更具方向性與效率性。然而，導入 DOT 排程機制的前提之一，是整個 BLE Mesh 網路需具備清楚的階層結構，而 Sink 節點（資料匯集端）必須作為整體樹狀拓樸的根節點。唯有如此，封包才能沿著既定的父子節點路徑，自節點有效匯流至 Sink 節點，達成 DOT 排程所需的單向、無冗餘傳輸目標。

然而，由於 FruityMesh 原生設計並非以 Sink 節點為中心的拓樸為出發點，現有拓樸建立流程尚無法保證 Sink 節點必然成為根節點。在未進行拓樸控制調整的情況下，可能出現 Sink 位置處於樹葉節點甚至中繼節點等非理想情境。因此，若要在 FruityMesh 架構中有效實作 DOT 排程機制，勢必需重新設計拓樸建立邏輯，強制指定使 Sink 節點始終位於拓樸樹的根部，即使在斷線並重新連線的情況下，亦能恢復其根節點角色，確保網路穩定性與傳輸效能。

3.1.2　BLE Mesh 封包傳輸問題

在基於 FruityMesh 的藍牙低功耗網狀網路傳輸機制設計中，[2] 研究已針對封包延遲與封包抵達率等品質指標進行優化，並取得初步成效。該研究藉由導入 DOT 與 DOST 排

程機制，成功改善了大部分的傳輸延遲與資料完整性。然而，即使在優化機制下，網路中仍不時出現封包遺失（掉封包）與重傳現象，顯示目前的傳輸機制尚存在進一步提升的空間。

造成封包掉落與重傳的關鍵因素之一，在於 BLE Mesh 網路的流量集中特性。由於整體網路皆以 Sink 節點作為封包的最終目的地，所有資料封包最終皆會匯集至該節點，導致越接近 Sink 的節點需承擔更多的中繼與轉發任務。這種負載不均的現象將使中樞節點在短時間內接收到大量傳輸要求，導致中樞節點的 Sent buffer 資源迅速耗盡。一旦緩衝區爆滿，不僅無法即時處理新進封包，還可能造成排程延遲、封包掉落，進而引發重傳機制啟動，進一步加重網路負載。

為有效緩解此類壅塞現象，需從連線參數層級進行調整，特別是針對 CI 與 CE 進行優化配置。透過適當調整 CI 間隔，可有效控制節點可傳輸封包的節奏與頻寬分配，進一步降低高負載節點的壓力，提升整體封包處理效率。結合連線參數調整機制，可以有效改善封包壅塞問題，以提升 BLE Mesh 網路在多節點、高流量情境下的服務品質。

3.2 BLE Mesh 拓樸建立機制設計

本論文提出改善 FruityMesh 的拓樸建立機制，確保 Sink 節點始終位於整體網路的根節點，並在斷線後能夠自動恢復其角色。此設計為後續的 DOT 排程機制提供穩定的基礎，如圖3.1。

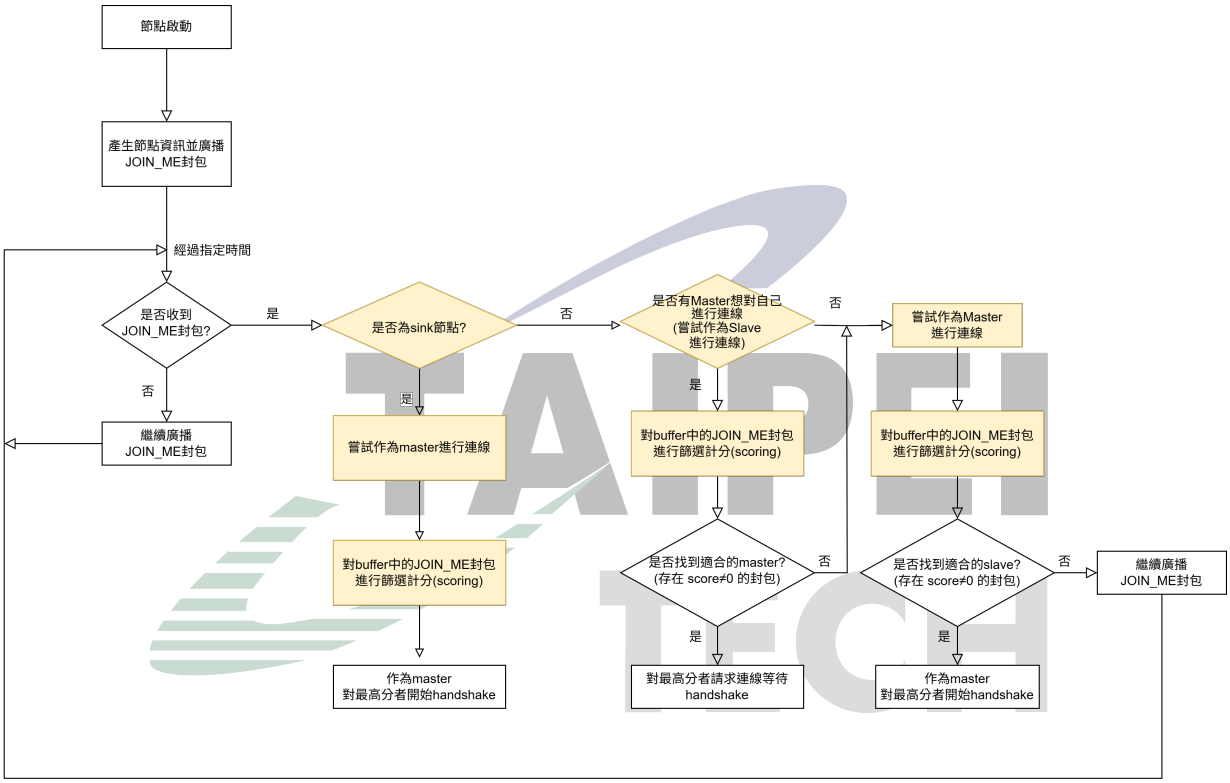

圖 3.1 提出的拓樸建立流程圖

在拓樸建立流程啟動時，每個節點在開機後會廣播包含自身節點資訊的 JOIN_ME 封包。節點收到其他節點所廣播的 JOIN_ME 封包後，會將該封包儲存於本地緩衝區中，並根據封包內容進行評分（scoring），以評估對方作為連線對象的適合程度。此評分機制可根據鄰近程度、RSSI 訊號強度、是否為 Sink 節點等參數進行加權計算，從而判斷潛在連線對象的優先順序。

若節點本身為預先指定的 Sink 節點，則會主動嘗試以 Master 的角色發起連線，以確保其成為拓樸的根節點。另一方面，對於非 Sink 節點，為了確保拓樸架構的根節點為 Sink 節點，節點在評估過程中會優先以 Slave 身分尋找合適的 Master 進行連線。只有當

未能找到合適的 Master 節點（例如評分皆為 0 或未收到有效的 JOIN_ME 封包）時，該節點才會轉而作為 Master 嘗試連線至其他尚未建立連線的節點，進一步擴展網路的拓樸結構。

若節點在轉為 Master 之後，仍未能找到合適的 Slave 節點（即緩衝區中所有 JOIN_ME 封包的評分皆為 0 或無法建立穩定連線）時，該節點將回到初始狀態，繼續定期廣播 JOIN_ME 封包，以等待後續更合適的連線機會出現。此重試機制能避免節點因短期內找不到連線對象而陷入孤立狀態，提升整體拓樸建立的成功率與網路的自組織能力。

3.2.1 決定最好的群組加入

Algorithm 1及 Algorithm 2及 Algorithm 3描述了節點在藍牙低功耗網狀網路（BLE Mesh）中，決定其與其他節點建立連線方式的邏輯流程。其核心目標是依據當下網路狀態與節點角色，判斷該節點應該主動發起連線（作為 Master）或是被動接受連線（作為 Slave）。

Algorithm 1 DetermineBestClusterAvailable - Part 1

```
Initialize result as NO_NODES_FOUND
Get device type and assign to deviceType
if deviceType is SINK and outbound connections are available then
    bestClusterAsMaster ← DetermineBestClusterAsMaster()
    if bestClusterAsMaster ≠ null then
        Adjust connectionIv based on peer’ s device type
        Attempt to connect as Master
        if connection succeeds then
            Update connection attempt time and count
        end if
        Set result.result ← CONNECT_AS_MASTER
        Set result.preferredPartner ← bestClusterAsMaster.sender
        return result
    end if
end if
```

Algorithm 1流程一開始會檢查當前節點是否為 SINK 且具備可用的對外連線資源。若節點為 Sink 節點，則優先考慮作為 Master 嘗試連線，並透過 DetermineBestClusterAs-Master() 函式選出最合適的連線目標節點。若成功取得候選節點，系統會根據對方裝置的類型調整連線參數（如 CI），並嘗試建立連線。若連線成功，系統會更新連線時間與次數統計，並回傳連線成功的決策結果。

Algorithm 2 DetermineBestClusterAvailable - Part 2

```
Reset currentAckId to 0
bestClusterAsSlave ← DetermineBestClusterAsSlave()
if deviceType is SINK then
    bestClusterAsSlave ← null
end if
if bestClusterAsSlave ≠ null then
    currentAckId ← bestClusterAsSlave.clusterId
    if meshMaxInConnections == 1 then
        Check if any fresh connection exists (handshake not expired)
        if no fresh connection and freeMeshInConnections == 0 then
            if clusterSize ≠ bestClusterAsSlave.clusterSize OR random trigger passed then
                Force disconnect other mesh connections
                Reset cluster size to 1
                Generate new clusterId
            end if
        end if
    end if
    Update JoinMe packet
    Set result.result ← CONNECT_AS_SLAVE
    Set result.preferredPartner ← bestClusterAsSlave.sender
    return result
end if
```

Algorithm 2若節點角色並不是 Sink 節點，則會優先作為 Slave 接受他人連線。此時

會重設回應欄位（currentAckId），並透過 DetermineBestClusterAsSlave() 選出合適的候選節點。並再次檢查當前節點如果為 SINK 節點，則不允許 Sink 節點作為 Slave 進行連線。當節點作為 Slave 進行連線時，若符合條件的 Master 節點存在，系統則會建立連線。

Algorithm 3 DetermineBestClusterAvailable - Part 3

```
bestClusterAsMaster ← DetermineBestClusterAsMaster()
if bestClusterAsMaster ≠ null and outbound connections are available then
    Adjust connectionIv and attempt to connect
    if connection succeeds then
        Update connection info
    end if
    Set result.result ← CONNECT_AS_MASTER
    Set result.preferredPartner ← bestClusterAsMaster.sender
    return result
end if
Log 'no cluster found'
Set result.result ← NO_NODES_FOUND
return result
```

Algorithm 3當節點作為 Slave 進行連線沒有找到適合的 Master 時，則該節點會再度嘗試以 Master 身分建立連線，重複執行候選節點評估與連線流程，若成功，則回傳連線結果。最後如果所有嘗試皆沒有辦法成功連線，系統會記錄「未找到叢集」的訊息，並回傳 NO_NODES_FOUND 的決策結果。

3.2.2 以 Mater 身分選擇最佳的 Slave

Algorithm 4 CalculateClusterScoreAsMaster

1: Retrieve device type from device configuration
2: **if** packet is too old **then return** 0
3: **end if**
4: **if** packet.clusterId == this.clusterId **then return** 0
5: **end if**
6: **if** packet has no free inbound connections **then return** 0
7: **end if**
8: **if** packet.ackField $\neq$ this.clusterId and $\neq 0$ **then return** 0
9: **end if**
10: **if** packet.clusterSize $>$ current cluster size **and** current device is not SINK **then return** 0
11: **end if**
12: **if** packet is temporarily blacklisted **then return** 0
13: **end if**
14: **if** already connected to packet.sender **then return** 0
15: **end if**
16: **if** packet.rssi $<$ threshold **then return** 0
17: **end if**
18: **if** current device type == LEAF **then return** 0
19: **end if**
20: $rssiScore \leftarrow 100 + packet.rssi$
21: $score \leftarrow packet.freeMeshOutConnections + rssiScore - (packet.hopsToSink \times 1000)$
22: Modify score using preferred partner policy
23: **return** score

Algorithm4說明了節點在嘗試成為 Master 時，如何針對鄰近的叢集候選節點進行評分。該評分機制用來決定是否值得主動與某個節點建立連線。

一開始，節點會先取得自身的裝置類型，若為 LEAF（葉節點）則無法發起連線，直接回傳 0 分。接著系統會篩選掉以下不合適的候選節點：

- 候選節點的封包太舊
- 與候選節點的叢集相同
- 候選節點無 inbound slot 的數量可以使用
- 與候選節點的 ack 欄位不符
- 候選節點不是 Sink 節點且對方叢集較大
- 候選節點在短時間內已多次連線失敗（暫時黑名單）
- 已與該節點連線中
- 候選節點的 RSSI 過低

通過初步篩選後，系統會以 RSSI 為基礎計算連線品質分數 rssiScore，再根據對方節點的連線資源與至 Sink 節點的跳數（hopsToSink）進行加權計算整體分數。該分數越高，代表對方節點越適合作為連線對象，最後，系統還會根據偏好節點（preferred partner）進行微調後，回傳最終評分結果。

3.2.3 以 Slave 身分選擇最佳的 Master

Algorithm 5 CalculateClusterScoreAsSlave

Retrieve device type from configuration
if deviceType == SINK **then return** 0
end if
if packet.hopsToSink < 0 **then return** 0
end if
if packet.freeMeshOutConnections == 0 **then return** 0
end if
if packet is too old **then return** 0
end if
if packet.clusterId == current.clusterId **then return** 0
end if
if packet.clusterSize < current cluster size **and** packet.deviceType $\neq$ SINK **then return** 0
end if
if packet.rssi < threshold **then return** 0
end if
$rssiScore \leftarrow 100 + packet.rssi$
$score \leftarrow 0$
if packet.deviceType == SINK **then**
    $score \leftarrow score + 10000$
end if
$score \leftarrow score + (packet.hopsToSink \times -1000) + (packet.clusterSize \times 100) + (packet.freeMeshOutConnections \times 100) + rssiScore$
Modify score based on preferred partner logic
return score

Algorithm5描述了節點在考慮作為 Slave 加入其他叢集時，如何針對潛在的 Master 節點進行評分。

首先系統會排除不適合的情況：

- 若目前節點為 SINK 節點，則無法作為 Slave 加入其他叢集。
- 候選節點無 outbound slot 的數量可以使用。
- 候選節點的封包太舊。
- 當前節點與候選節點為同一叢集。
- 候選節點並非 SINK 節點且候選節點叢集規模小於當前節點叢集。
- 候選節點的 RSSI 過低。

若符合條件，則會計算 RSSI 分數，並依據以下項目計算總分：

- 候選節點是否為 SINK 節點，弱勢 Sink 節點會獲得額外的 10000 分。
- 對方距離 Sink 節點的跳數（hopsToSink）乘以-1000。
- 對方叢集大小（clusterSize）t/6u3 乘以 100。
- 可用的 outbound slot 數量（freeMeshOutConnections）乘以 100。
- 實際 RSSI 品質分數（rssiScore）

最終分數會再次根據偏好節點機制進行修正，作為節點選擇最佳 Master 的依據。

3.2.4 Self-Healing 機制的拓樸修復改進

原生 FruityMesh 提供了 Self-Healing 機制，使節點在斷線後能回到 HIGH_DISCOVERY 狀態並尋找可連線的鄰居節點，且會由小的叢集合併進大的叢集，從而重新建立連線。然而，此機制下若斷線的是 Root（Sink）節點，重新連回後將失去 Root 身份，可能導致整個網路拓樸失效或性能下降。

為解決此問題，本研究修改了 CLUSTER_WELCOME 訊息的握手流程。在握手判斷邏輯中，若發現對方為 SINK 且其 Cluster 比自己小（或等於），則避免讓原本為 Root 的

節點退位為普通節點，並強制原 Cluster 切斷其他連線，重新啟動拓樸建立，使原本的 SINK 節點能保有 Root 身份，維持拓樸穩定性與網路一致性，如 Algorithm6。

Algorithm 6 Topology Self-Healing Handshake Logic

Require: CLUSTER_WELCOME packet received

```
1:  if packet size invalid then
2:      Log and ignore packet
3:  else
4:      Save partner handle and enter HANDSHAKING state
5:      Backup local cluster ID and size
6:      if remote cluster ID == local cluster ID then
7:          Disconnect: SAME_CLUSTERID
8:      else if remote cluster size < local size and remote is not SINK then
9:          if connection is inbound then
10:             Disconnect: WRONG_DIRECTION
11:         end if
12:     else if network ID mismatch then
13:         Disconnect: NETWORK_ID_MISMATCH
14:     else if connection is not preferred and preferences ignored then
15:         Disconnect: UNPREFERRED_CONNECTION
16:     else
17:         Accept connection as Root
18:         Send CLUSTER_ACK_1 with hopsToSink = 0 if self is SINK
19:         Disconnect all other mesh connections
20:         Reinitialize local cluster with size = 1 and new ID
21:     end if
22: end if
```

Algorithm6為了強化 FruityMesh 在節點斷線重連後的拓樸穩定性，本研究在原有 Self-Healing 機制之上，進一步修改其握手邏輯。當節點收到 CLUSTER_WELCOME 封包後，將進入拓樸重組的判斷流程。具體邏輯如下：

- 首先，節點會確認封包的大小是否正確，若不正確則忽略該封包。
- 接著，進入 Handshaking 狀態並備份目前的 Cluster ID 與 Cluster 大小，用以後續比對使用。
- 若對方節點的 Cluster ID 與自己相同，代表已屬同一個 Cluster，這在初始階段是不應該出現的情況，因此直接斷線處理。
- 若對方的 Cluster 大小比本地端小，且其並非 SINK 節點，則本節點認定自己應作為主導方，若該連線是 inbound，則代表方向錯誤，同樣會中止連線。（freeMeshOut-Connections）乘以 100。
- 若兩節點的 network ID 不一致，或連線對象並非偏好的節點，也將被強制中止以避免不必要的 Mesh 錯誤連線。

在以上檢查皆通過的情況下，代表本節點應接受對方節點加入自己的 Cluster。若本節點為 SINK 節點，則會在 CLUSTER_ACK_1 回覆中指定 hopsToSink = 0，表明自己為 Mesh 根節點，並強制中斷其他 Mesh 連線，重新以自己為中心建立拓樸結構，確保在重新連線的情況下仍能維持 SINK 節點的 Root 地位。

3.3 BLE Mesh 封包傳輸機制設計

在 BLE Mesh 網路中，封包傳輸的效率與穩定性對整體系統效能具有關鍵影響。尤其在具備多跳（multi-hop）特性的網狀架構中，位於拓樸上層的節點（例如接近 Root 的節點）需承擔來自下層節點的大量封包轉送任務，因此其封包處理壓力遠高於葉節點。

若未妥善配置這些節點的傳輸參數，將容易造成 sent buffer 過載、傳輸延遲增加、甚至出現頻繁重傳等問題，進而影響整體系統穩定性與效能表現。

為了改善上述問題，本研究提出一種基於節點層級調整連線間隔參數（Connection Interval, CI）的機制，如圖 3.2 所示。設計理念為：根據節點於拓樸中的層級深度，賦予不同的 CI 值，越靠近 Root 的節點，其連線間隔設定越短，以提供更頻繁的傳輸機會。此方法能夠有效緩解中繼節點所面臨的頻寬壓力，減少傳輸阻塞與重傳情形，進一步提升整體 BLE Mesh 網路的穩定性與資料傳遞效率。

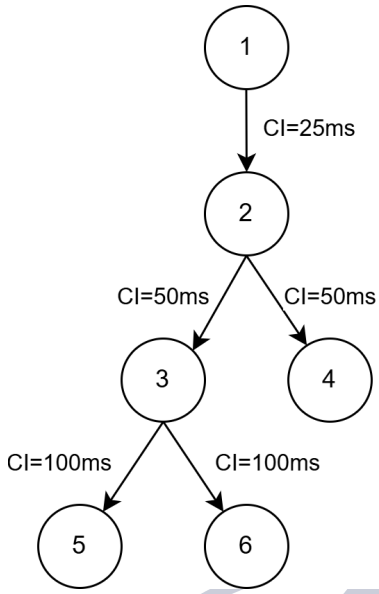

圖 3.2 BLE Mesh 封包傳輸機制設計節點架構圖

圖 3.2 顯示一個包含六個節點的 Mesh 拓樸，其中節點 1 為 Root。依據拓樸深度，每層的 CI 設定分別為 25 ms、50 ms 與 100 ms，形成由上而下頻率遞減的調整策略，有效反映節點所承擔的資料轉發負載，透過此層級化的參數配置機制，本研究期望在不增加硬體資源的前提下，提升多跳 BLE Mesh 網路在長時間運行中的穩定性與封包抵達率。

第四章　實驗結果與分析

4.1　實驗環境與參數設計

本研究實驗環境包含實體硬體測試平台與模擬環境平台兩部分，搭配修改後之 FruityMesh 協定，用以驗證所提出之拓樸生成與傳輸優化機制在不同環境下的效能表現。

4.1.1　實體實驗平台

實體網路建構基於 Nordic nRF52840 SoC 開發板，相關配置如表4.1所示。

表 4.1 實體實驗平台配置

項目	說明
晶片型號	Nordic Semiconductor nRF52840（支援 BLE 5.0）
節點數量	共 6 個節點（含 1 個 Sink 節點）
資料傳輸通道	使用 UART 作為 Log 回傳通道
電力供應方式	透過 USB 供電進行情境部署
節點與節點距離	小於 1 公尺

本研究所進行的實體實驗平台建置於 Nordic Semiconductor 所推出的 nRF52840-DK 開發板上。整體平台共由 6 個 nRF52840 節點組成，分別模擬 Sink 節點與多個子節點，透過實際部署驗證所提出之拓樸建立機制與封包傳輸策略在真實硬體環境下的效能表現。

各節點皆搭載 Nordic 所提供之 SoftDevice BLE 堆疊，並以 FruityMesh 為基礎韌體框架進行修改與擴充。根據不同實驗目的，調整拓樸建立參數、連線策略與角色排程機制，並燒錄至各 nRF52840-DK 板上進行測試。透過 UART 紀錄各節點狀態與封包交換資訊，如圖4.1，觀察節點之間的拓樸變化與網路重建過程。

實體節點部署於室內空間中，模擬感測器佈建情境。節點間距為小於 1 公尺。所有實驗皆重複進行多次以確保結果穩定與具參考性，並與模擬平台結果進行對比分析，以佐證演算法之可行性與實用性。

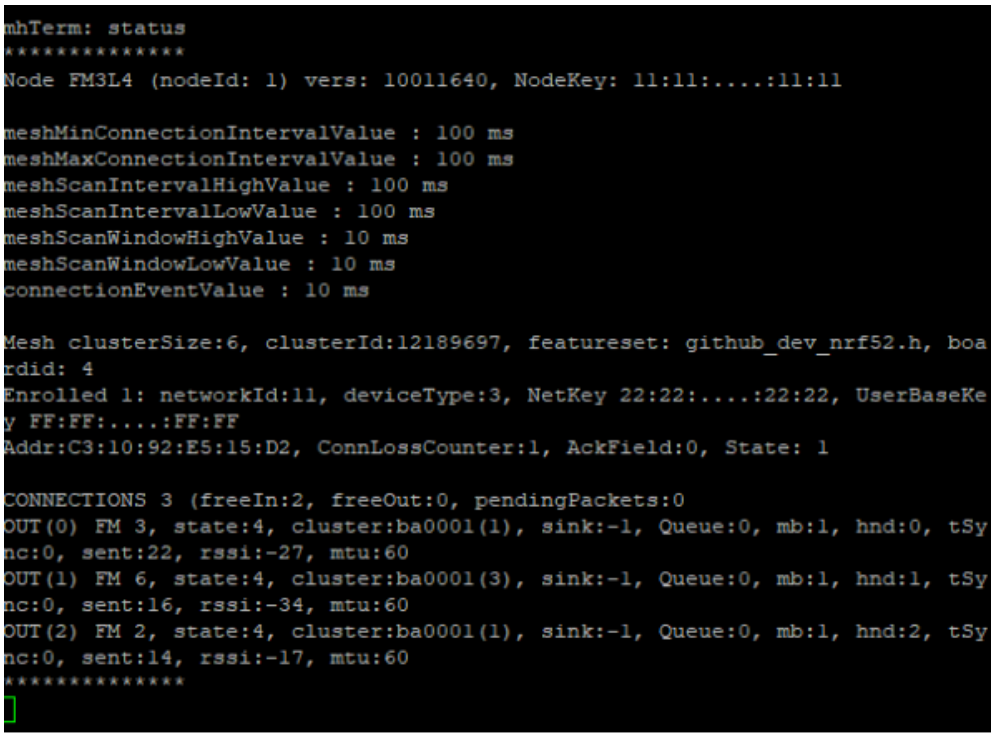

圖 4.1 nRF52840-DK 日誌紀錄

4.1.2 模擬實驗平台（CheerSim）

為進一步驗證系統在大規模節點下的可擴展性與穩定性，本研究亦使用 CheerSim 模擬器進行測試。CherrySim 作為模擬 BLE 網路的主要工具。CherrySim 是由 FruityMesh 開發團隊所提供的官方模擬器，能夠在不需要實體硬體的情況下，模擬基於 Nordic nRF52 系列晶片及其 SoftDevice 堆疊的 BLE 節點行為。CherrySim 支援完整的連線協議模擬，包含封包交換、連線建立、訊號強度 (RSSI)、連線參數（如 CE/CI）等，並提供豐富的除錯與日誌紀錄功能。

此外，CherrySim 內建網路狀態視覺化介面，可清楚顯示模擬過程中節點的地理分布與連線拓樸，並即時更新節點之間的連線變化。研究人員可透過該介面直觀掌握拓樸建立流程、連線演化過程與自我修復行為。

4.1.2.1 模擬實驗流程

本研究於 CherrySim 中設計三組模擬情境，分別模擬節點數為 11、21 及 31 的藍牙網路環境，每種情景分別使用 3 種不同 seed 進行模擬。每組模擬皆包含一個 Sink 節點，

並配置節點 4 的位置為 Sink 點，模擬其拓樸自動生成與穩定過程，模擬觀察拓樸建立完成所需時間、自我修復機制及平均 HopsToSink 數值。

其中，11 節點網狀節點配置式意圖，如圖4.2，21 節點網狀節點配置式意圖，如圖4.3，31 節點網狀節點配置式意圖，如圖4.4。

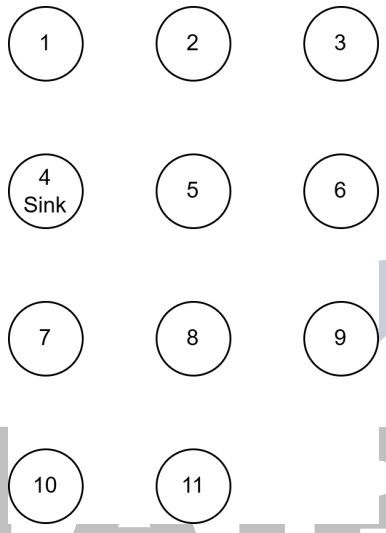

圖 4.2 模擬實驗 11 點網狀節點配置示意圖

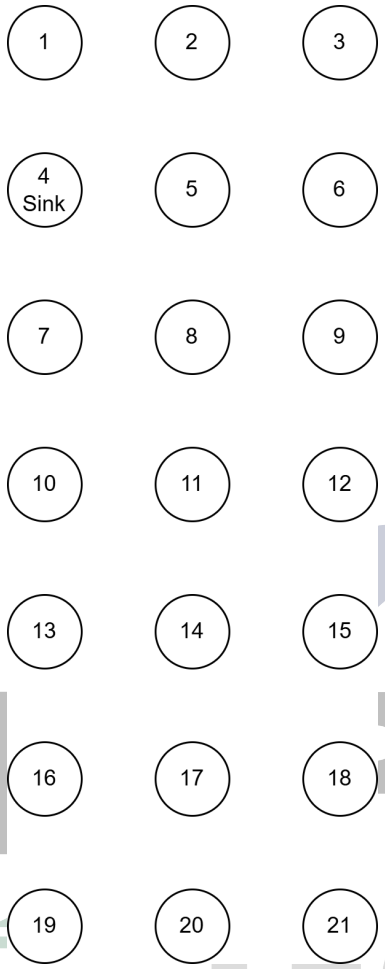

圖 4.3 模擬實驗 21 點網狀節點配置示意圖

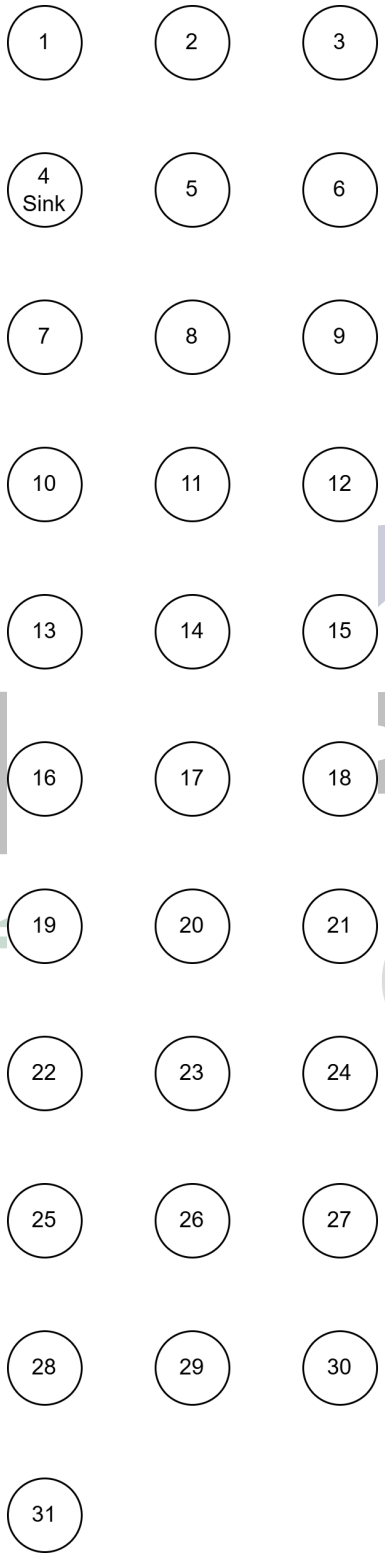

圖 4.4 模擬實驗 31 點網狀節點配置示意圖

4.1.2.2 模擬實驗參數設定

模擬實驗中，所有節點相關參數如表4.2所示。表中包含 Simulated Device、Node Deployment、Distance between nodes、節點數量、Sink 節點位置、Seed、Connection Interval 及 Connection Event 等參數設定。

表 4.2 模擬實驗參數設定表

參數名稱	設定值
Simulated Device	nRF52DK
Node Deployment	Grid
Distance between nodes	3m
節點數量	11, 21, 31（inculding 1 sink node）
Sink 節點位置	節點 4
Seed	1, 10, 100
Connection Interval	100 ms
Connection Event	10 ms

4.2 效能評估指標

本研究針對所提出的拓樸生成與傳輸優化機制，設計以下數項效能評估指標，以量化系統在不同模擬情境下的表現。

4.2.1 拓樸建立時間

拓樸建立時間是指節點成功建立 Mesh 拓樸所需的時間。此指標反映系統在節點加入網路時的效率，時間越短表示系統越能快速適應新節點，具備更佳的擴展性與部署靈活性，如4.1式。

$$\text{TopologyEstablishmentTime} = Topology_{finish} - Topology_{init} \tag{4.1}$$

其中：

- $Topology_{finish}$ 為拓樸建立完成的時間點。
- $Topology_{init}$ 為拓樸建立開始的時間點。

4.2.2 自我修復時間

自我修復時間指的是節點在斷線後，能夠重新加入網路並恢復傳輸功能所需的時間。此指標評估網路面對節點失效或路徑中斷時的容錯能力與恢復效率，能顯示出 Mesh 結構的穩定性，如4.2式。

$$\text{SelfHealingTime} = Topology_{finish} - Node_{disconnection} \tag{4.2}$$

其中：

- $Topology_{finish}$ 為拓樸建立完成的時間點。
- $Node_{disconnection}$ 為節點斷線的時間點。

4.2.3 平均 HopsToSink 數值

平均 HopsToSink 表示封包從任意節點傳送至 Sink 節點所需的平均跳數。此數值越低代表網路拓樸越扁平，資料傳輸的路徑越短，能有效降低延遲並提升整體效率，如4.3式。

$$\text{AvgHopsToSink} = \frac{\sum HopsToSink}{N_{nodes}} \tag{4.3}$$

其中：

- $\sum HopsToSink$ 為所有節點到 Sink 節點的跳數總和。
- N_{nodes} 為網路中節點的總數，不包含 Sink 節點。

4.2.4 平均封包傳輸延遲

此指標衡量封包從來源節點產生到 Sink 節點成功接收所經歷的時間總和，反映網路整體的即時性與處理能力，此值越低，代表網路回應時間越短、傳輸效率越高。如4.4式

$$\text{AvgPacketDelay} = \frac{\sum Delay_{end-to-end}}{\sum N_{sink\_received\_packet}} \tag{4.4}$$

其中：

- $\sum Delay_{end-to-end}$ 為所有成功接收封包從來源節點產生至 Sink 節點接收之總延遲時間。
- $\sum N_{sink\_received\_packet}$ 為所有成功接收封包的總數。

4.2.5 平均封包抵達率

平均抵達率（Average Packet Delivery Ratio, PDR）表示成功接收封包的總數，佔來源節點原本應發送封包總數的比例，當網路品質下降或傳輸負載過高時，PDR 會明顯降低，如4.5式。

$$\text{AveragePDR} = \frac{\sum N_{sink\_received\_packet}}{\sum N_{source\_send\_packet}} \times 100\% \tag{4.5}$$

其中：

- $\sum N_{sink\_received\_packet}$ 為所有成功接收的封包數量。
- $\sum N_{source\_send\_packet}$ 為所有來源節點發送的封包數量。

4.2.6 平均封包重傳率（Average Retransmission Rate）

平均重傳率衡量封包在網路中被重複傳送的比例，反映網路壅塞或角色衝突導致的傳輸效率問題，重傳率越低，代表網路越穩定、資源使用效率越高，如4.6式及4.7式。

$$\text{AvgRetransmissionRate} = \frac{\sum RetransmissionPacket}{\sum Conn_{expect\_send\_packet}} \times 100\% \quad (4.6)$$

$$\sum RetransmissionPacket = \sum Conn_{actual\_send\_packet} - \sum Conn_{expect\_send\_packet} \quad (4.7)$$

其中：

- $\sum Conn_{expect\_send\_packet}$ 為所有節點預期應發送的封包數量。
- $\sum Conn_{actual\_send\_packet}$ 為實際發送的封包總數（包含重傳）。
- $\sum RetransmissionPacket$ 為實際重傳封包的總數。

4.3 實驗結果與分析

本研究針對所提出的拓樸生成與傳輸優化機制，進行實體與模擬環境下的效能評估。以下分別呈現實體實驗平台與模擬平台的實驗結果，並針對各項效能指標進行分析。

4.3.1 模擬提出之網路拓樸建立

模擬實驗中，使用 CheerSim 平台進行三組不同節點數量的拓樸建立測試。每組模擬均包含一個 Sink 點，並配置節點 4 位置當作 Sink 節點。實驗結果顯示，所提出的網路拓樸演算法能夠確保 Sink 節點，也就是節點 4 的位置，位於拓樸的根節點，其中，圖4.5、4.6及4.7，分別表示節點數量為 11 點時，Seed 分別為 1、10、100 的模擬結果；圖4.8、4.9及4.10，分別表示節點數量為 21 點時，Seed 分別為 1、10、100 的模擬結果；圖4.11、4.12及4.13，分別表示節點數量為 31 點時，Seed 分別為 1、10、100 的模擬結果。

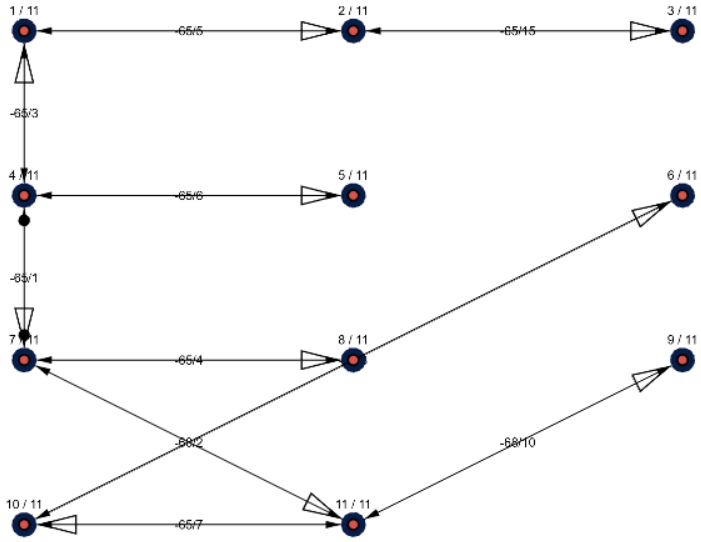

圖 4.5 模擬節點數量 =11 點，Seed=1

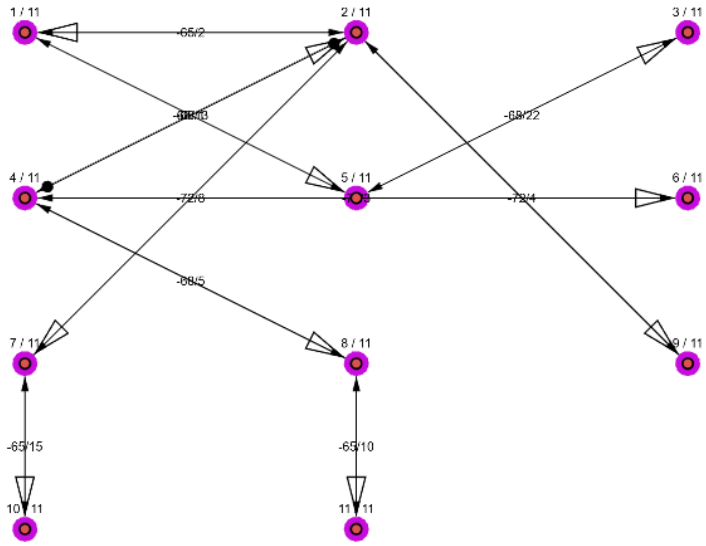

圖 4.6 模擬節點數量 =11 點，Seed=10

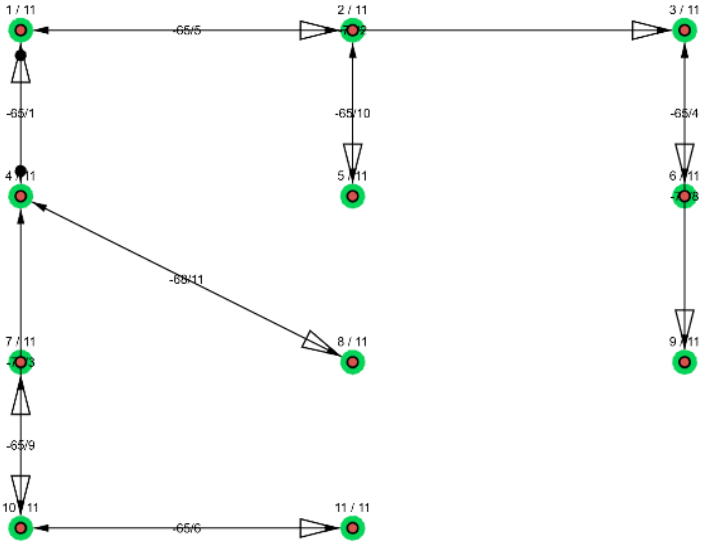

圖 4.7 模擬節點數量 =11 點，Seed=100

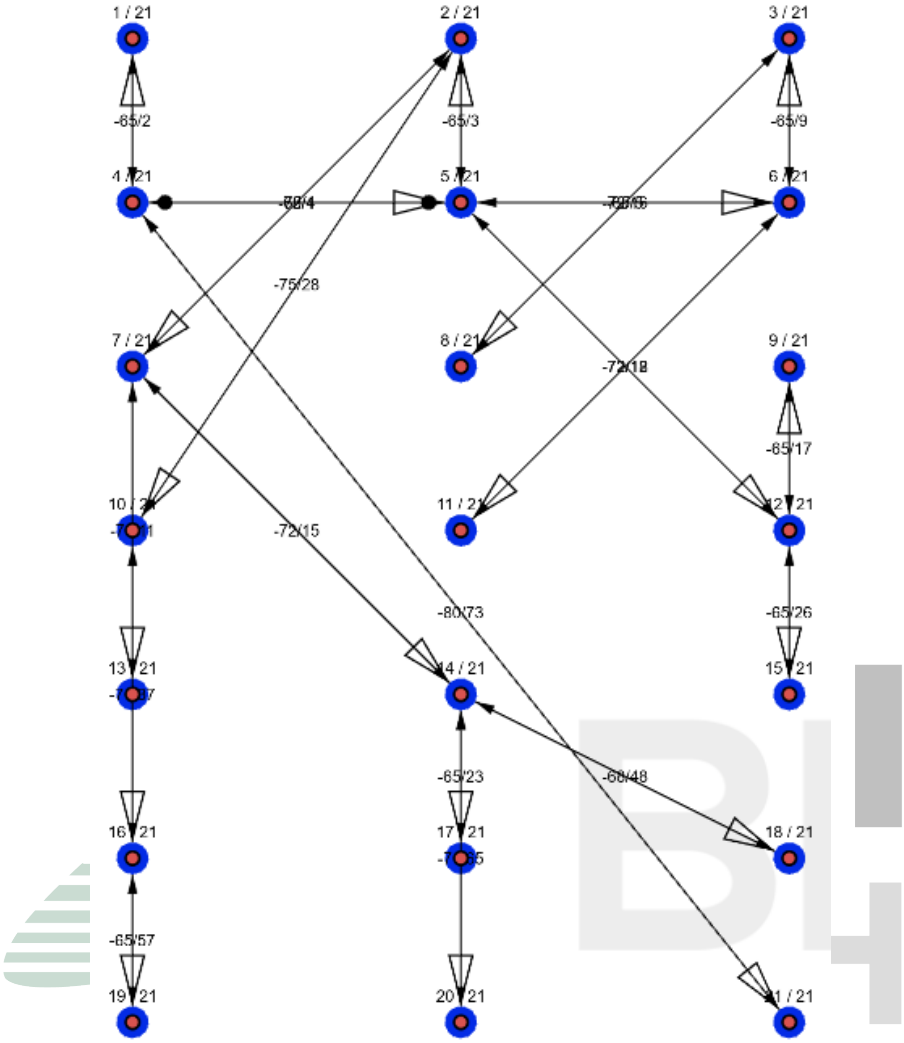

圖 4.8 模擬節點數量 =21 點，Seed=1

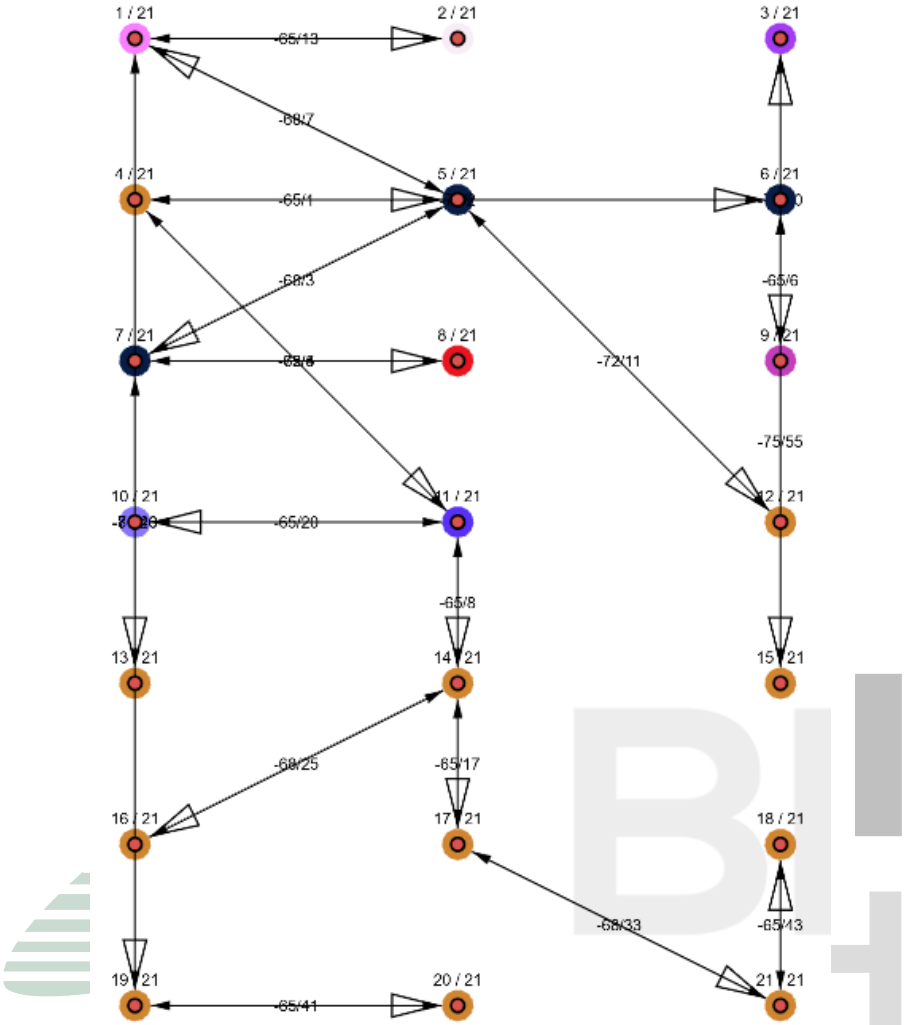

圖 4.9 模擬節點數量 =21 點，Seed=10

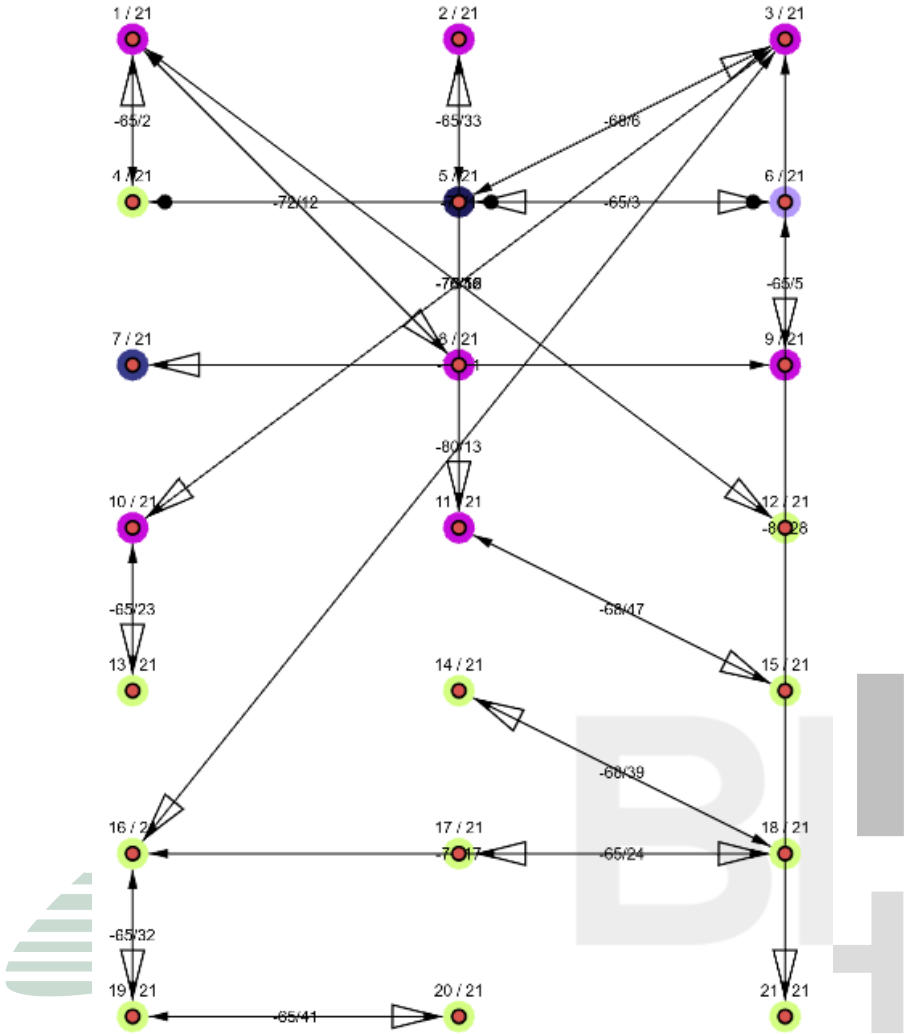

圖 4.10 模擬節點數量 =21 點，Seed=100

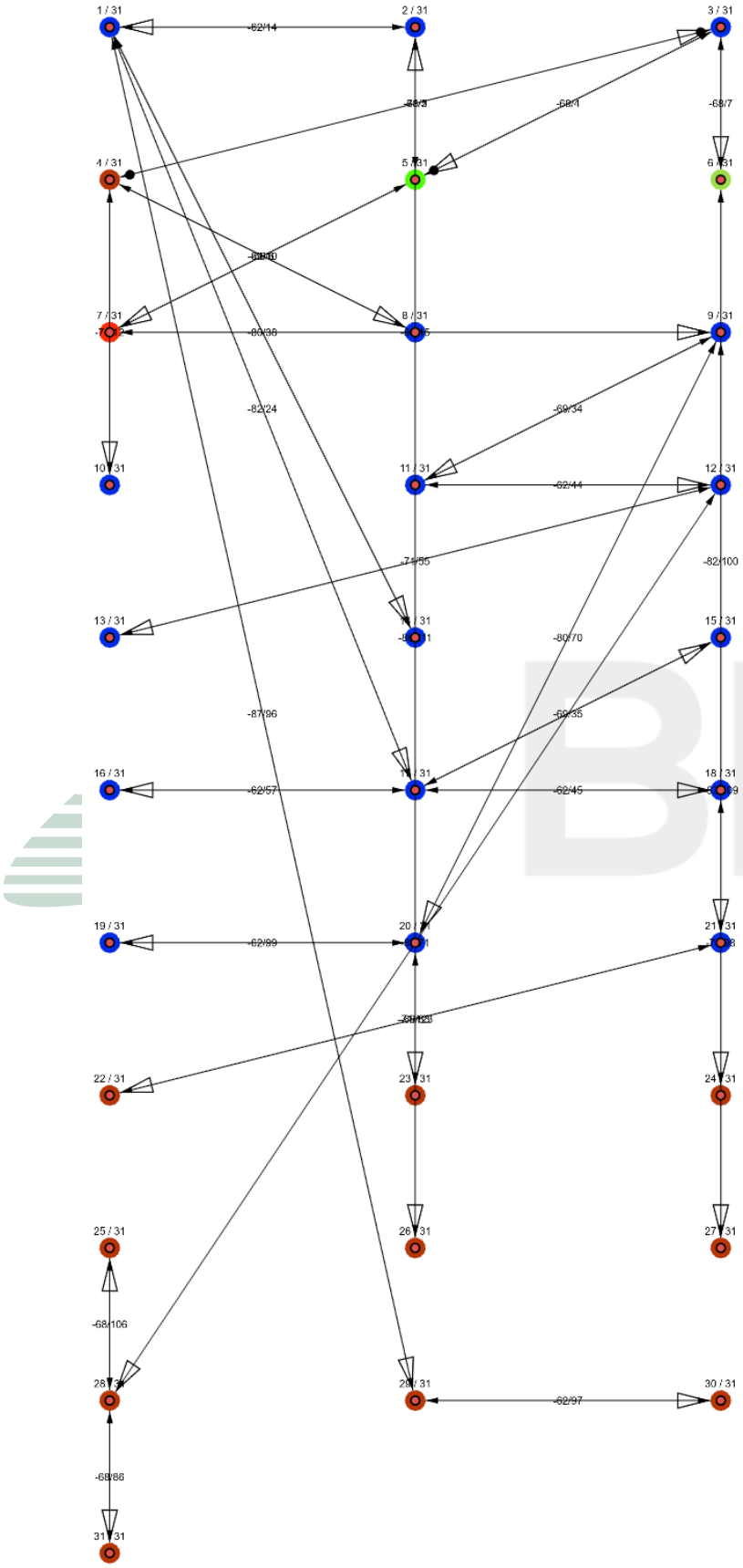

圖 4.11 模擬節點數量 =31 點，Seed=1

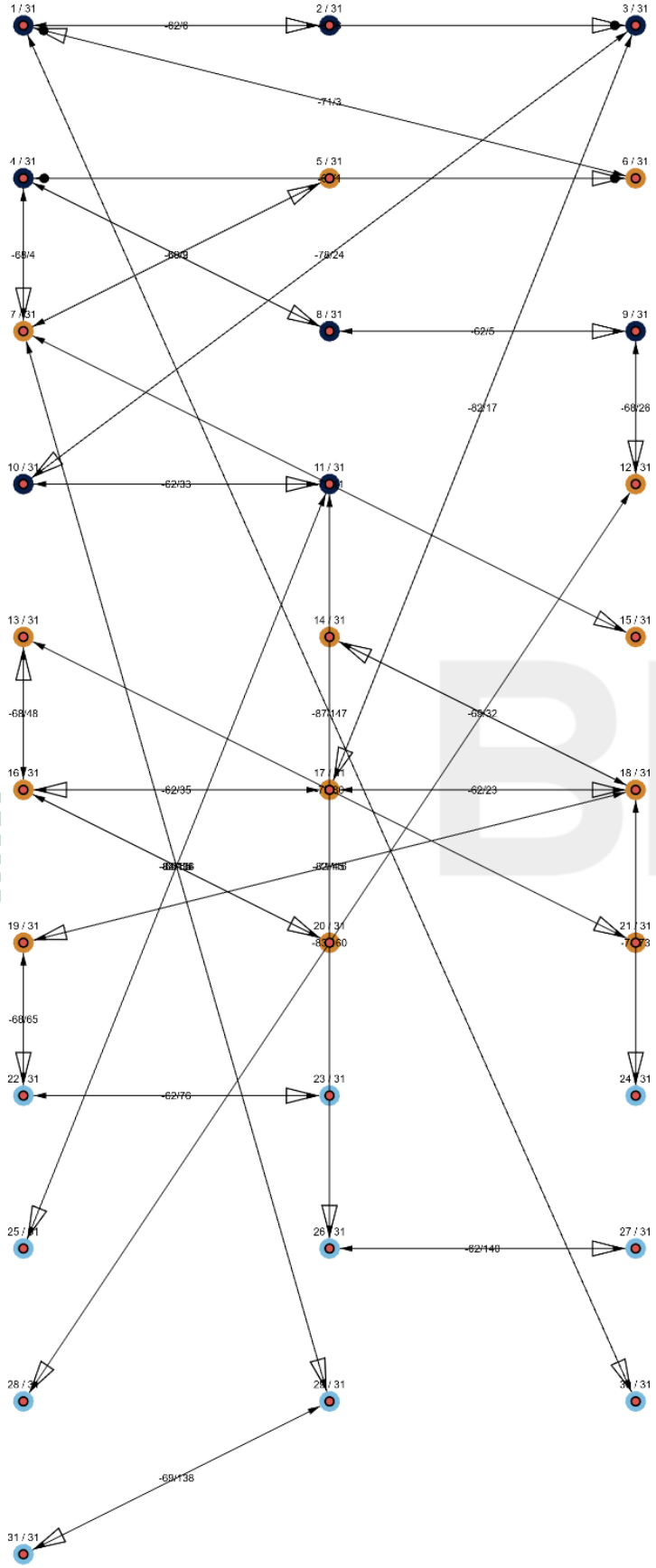

圖 4.12 模擬節點數量 =31 點，Seed=10

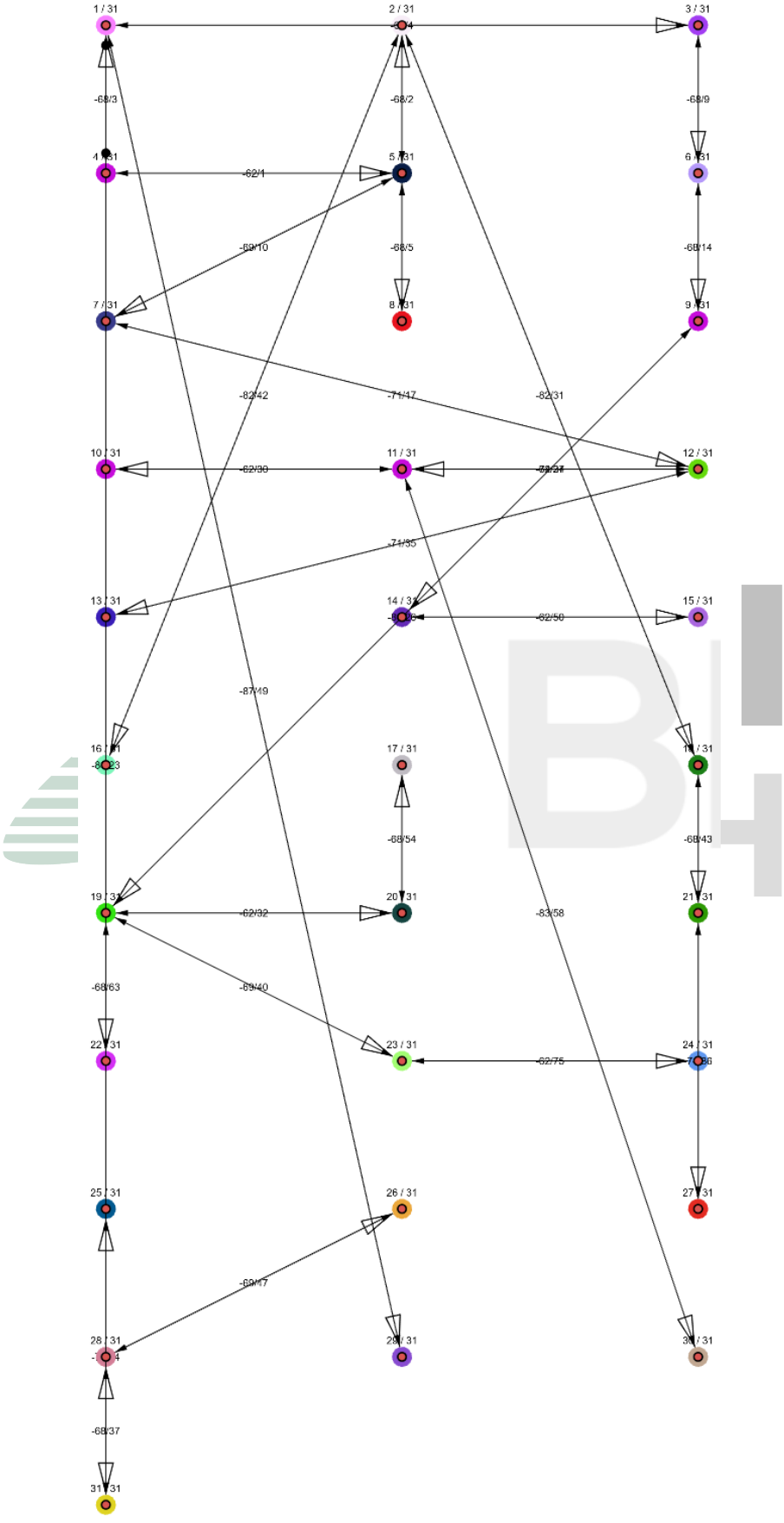

圖 4.13 節點數量 =31 點，Seed=100

4.3.2 模擬提出之網路拓樸 Sink 節點斷線後拓樸重建機制

模擬實驗中，會將 Sink 節點斷線後，Mesh 系統能夠自動重新建立拓樸並確保 Sink 節點仍然是根節點。分別以節點數量 11、21 及 31 實驗 10 次，圖4.14、4.15及4.16分別顯示了節點數量為 11、21 及 31 時，Sink 節點（節點位置 4）斷線後重新連線後的結果。可以觀察到，即使在 Sink 節點斷線後，Mesh 系統仍能恢復並保持 Sink 節點（節點位置 4）在拓樸的根節點。

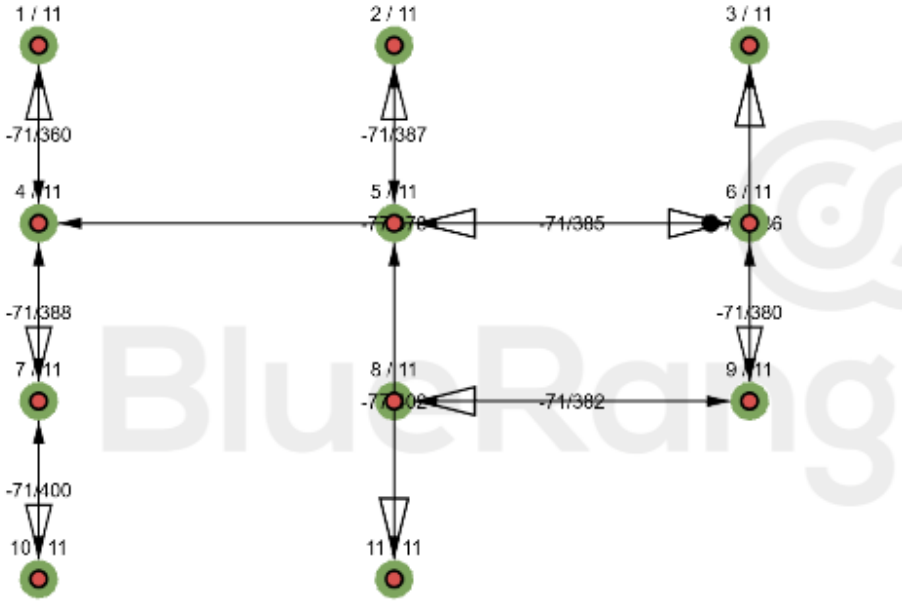

圖 4.14 模擬實驗 11 點 sink 斷線後連回結果

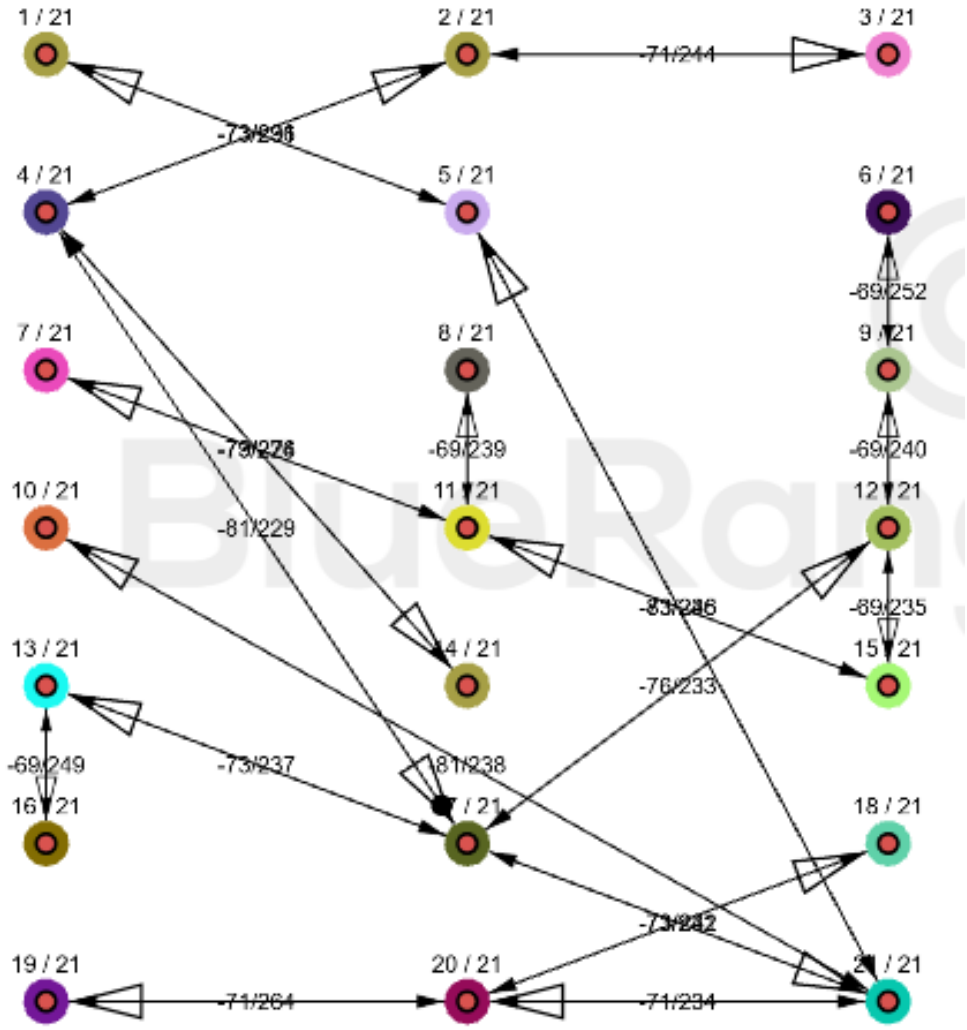

圖 4.15 模擬實驗 21 點 sink 斷線後連回結果

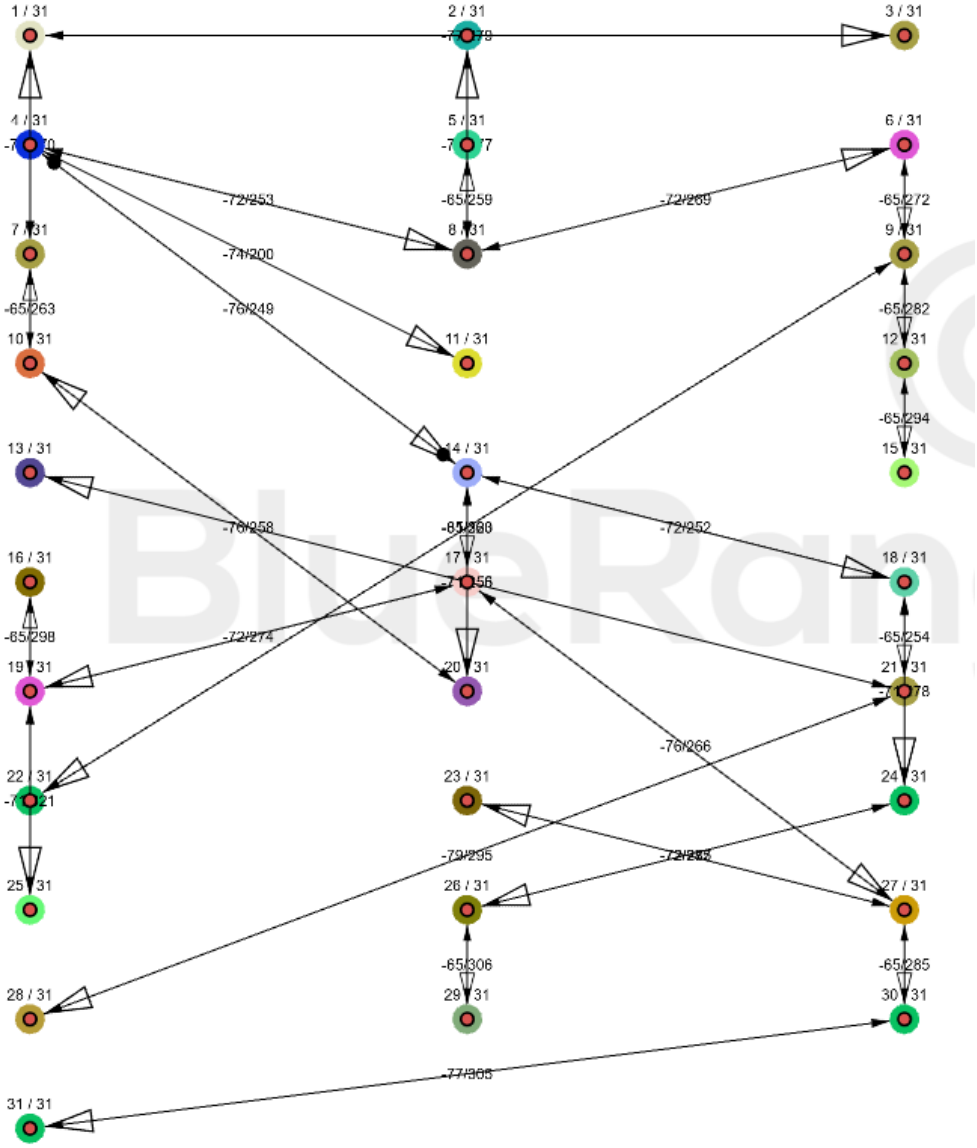

圖 4.16 模擬實驗 31 點 sink 斷線後連回結果

4.3.3 模擬節點重啟後之拓樸重建時間分析

在本研究中，為驗證所提出機制於節點重啟（reset）後的拓樸重建效率，設計了不同網路規模下的模擬實驗。實驗分別針對節點總數為 11、21 及 31 的網路拓樸進行，並比較兩種節點類型的重啟情境，分別為 Sink 節點（根節點）與非 Sink 節點（一般中繼節點或邊緣節點）。

每一種網路規模與節點類型情境下，皆進行 100 次重啟測試，記錄節點自啟動至成功連入 Mesh 網路所需之時間，並據此計算其平均連線時間與標準差。透過大量重複實

驗，能更準確評估系統在拓樸變動下的穩定性與恢復能力，並觀察不同節點角色對 Mesh 重建效率的影響。

本實驗分別於 11 點、21 點與 31 點之網路拓樸規模下，針對 Sink 節點與非 Sink 節點進行模擬測試。每組情境皆重複執行 100 次實驗，統計其 Mesh 成功建立所需的平均時間與標準差，結果如圖 4.17 所示。

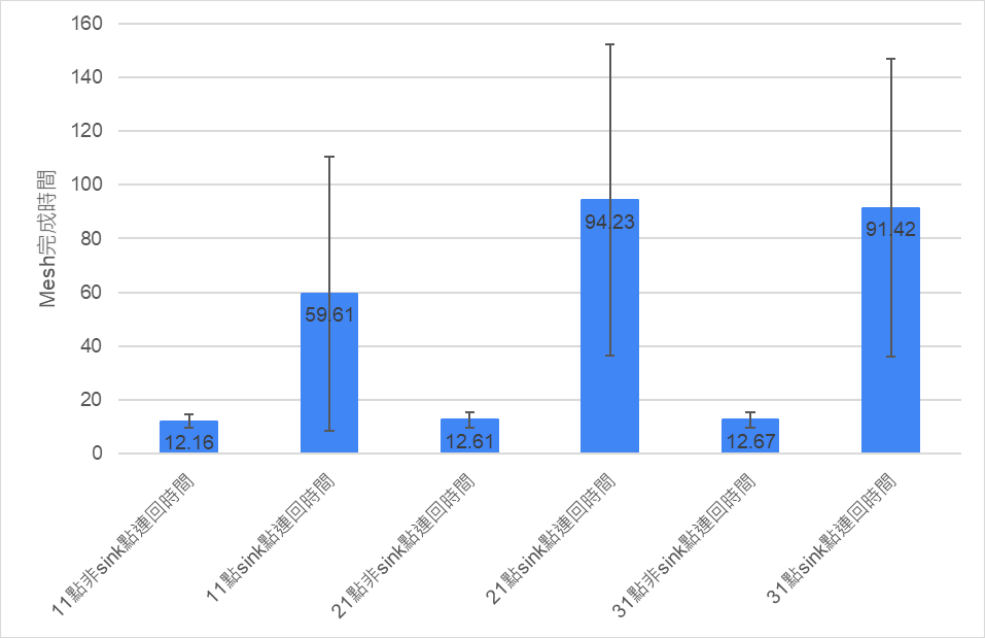

圖 4.17 模擬節點重啟後之拓樸重建時間分析

從實驗結果可觀察到，在三種網路規模下，非 Sink 節點的重啟連線平均時間皆維持在約 12.1～12.7 秒之間，且標準差相對較小（約 2.3～3.0），顯示其連線行為穩定且具一致性。而相較之下，Sink 節點的重啟平均時間顯著較高，分別為 59.61 秒（11 點）、94.23 秒（21 點）與 91.42 秒（31 點），且標準差也明顯較大（達 50～57），顯示在重建根節點角色與拓樸過程中，其恢復行為具有較高的變異性與延遲風險。

此現象可歸因於 Sink 節點在拓樸中需重新扮演根節點角色，並建立樹狀結構的主幹連線路徑，因此重啟後需耗費較多時間進行握手、選擇路徑與節點重連。此外，實驗亦證實所提出之拓樸修復機制能在重啟後正確辨識 Sink 節點並讓其重新成為 Mesh 拓樸的根節點，保持網路層級的一致性與穩定性。

4.3.4 模擬不同節點數與隨機環境下拓樸平衡性分析

本節實驗針對原生 FruityMesh 與本研究所提出之修改後演算法在拓樸平衡性上的表現進行比較。拓樸平衡性的衡量方式以節點連向 Sink 節點所需跳數（HopsToSink）作為評估依據，平均值與最大值越低表示拓樸越平衡、傳輸路徑越有效率。

為驗證演算法於不同隨機環境下之穩定性，實驗設計採用三組隨機種子，seed = 1、10、100，並分別在節點數為 11、21 與 31 的情境中進行測試，觀察各版本之平均 HopsToSink 與最大 HopsToSink 指標，如表4.3、4.4及4.5。

從實驗數據觀察，節點數為 11 點時，本研究提出之演算法在三組 Seed 中皆能有效降低平均 HopsToSink 數值。例如，Seed = 10 時原生 FruityMesh 的平均跳數為 3.27，而修改演算法僅為 1.91，最大跳數也從 5 降至 4，顯示修改機制能使節點更有效地連接至 Sink 節點，拓樸結構更為平衡。

在 21 點節點數下，Seed = 10 與 Seed = 100 的實驗中亦呈現類似趨勢。修改演算法可使平均 HopsToSink 明顯下降（例如 Seed = 10 時由 4.14 降為 2.43），最大跳數亦由 7 降為 5。此結果顯示在中等節點密度下，修改後之選擇連線機制具備更強的拓樸優化能力。

針對 31 點情境，雖然在 Seed = 1 的實驗中平均跳數略升（由 4.35 增為 4.55），但在其他兩組 Seed（Seed = 10 與 100）中，修改演算法仍可降低平均 HopsToSink（由 4.03 降為 3.65；由 5.10 降為 3.39），顯示本機制在高節點密度下仍能維持穩定之優化效果，並有效壓低拓樸樹的深度與最大跳數。

整體而言，修改後的演算法在大多數情境下均能降低節點到 Sink 的平均跳數，證實其在拓樸平衡性與傳輸效率上的改進成效。尤其在節點數越多的情境中，平均跳數的改善更為明顯。

表 4.3 模擬 Seed = 1 時的平均 HopsToSink 與最大 HopsToSink

	節點數量	平均 HopsToSink	最大 HopsToSink
原生 FruityMesh	11	2.55	5
本研究所提出之演算法	11	1.82	3
原生 FruityMesh	21	2.86	6
本研究所提出之演算法	21	2.90	7
原生 FruityMesh	31	4.35	8
本研究所提出之演算法	31	4.55	8

表 4.4 模擬 Seed = 10 時的平均 HopsToSink 與最大 HopsToSink

	節點數量	平均 HopsToSink	最大 HopsToSink
原生 FruityMesh	11	3.27	5
本研究所提出之演算法	11	1.91	4
原生 FruityMesh	21	4.14	7
本研究所提出之演算法	21	2.43	5
原生 FruityMesh	31	4.03	7
本研究所提出之演算法	31	3.65	7

表 4.5 模擬 Seed = 100 時的平均 HopsToSink 與最大 HopsToSink

	節點數量	平均 HopsToSink	最大 HopsToSink
原生 FruityMesh	11	2.64	5
本研究所提出之演算法	11	1.82	3
原生 FruityMesh	21	3.05	6
本研究所提出之演算法	21	3.48	6
原生 FruityMesh	31	5.10	9
本研究所提出之演算法	31	3.39	6

4.3.5 實作提出之網路拓樸建立

實體實驗中，使用 6 塊 Nordic nRF52840-DK 開發板進行小規模拓樸建立測試，表4.6為相關參數設定。每個節點均配置為 FruityMesh 協定堆疊，並透過 UART 進行日誌紀錄。

表 4.6 實作提出之網路拓樸建立參數設定

參數名稱	設定值
節點數量	6（包含 1 個 Sink 節點）
Sink 節點編號	1 / 5
Connection Interval	100 ms
Connection Event	10 ms

4.3.5.1 實作初始化拓樸中 Sink 點之 Root 角色實驗

為驗證本研究所提出的拓樸建立機制是否能正確地將指定之 Sink 節點設定為整個 BLE Mesh 網路的 Root（根節點），本實驗設計兩組不同的節點配置情境進行觀察。每組實驗均包含 6 個 nRF52840 節點，其中僅指定其中一節點為 Sink，其餘皆設定為 Static 類型，並透過 UART 回傳每個節點之拓樸連線資訊，以視覺化方式呈現網路結構。

在第一組實驗中，節點編號為 1（nodeId = 1）者被指定為 Sink 節點，初始化完成後之網路拓樸如圖 4.18 所示。從拓樸圖可觀察到，所有節點皆透過多層結構連接至節點 1，且未出現多個中心節點或分離子網情形，顯示該節點成功成為整體 Mesh 的根節點。

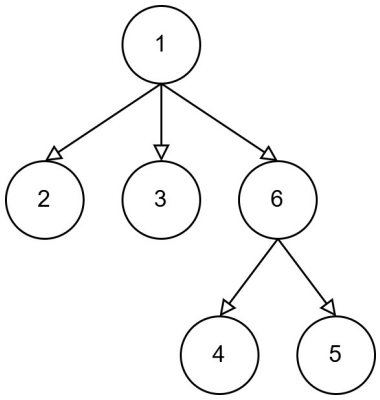

圖 4.18 實作 sink=1 初始化拓樸圖

於第二組實驗中，改由節點 5（nodeId = 5）作為 Sink，其他節點仍為 Static。初始化後之連線關係如圖 4.Y 所示，同樣可觀察到所有節點皆以節點 5 為樞紐，逐層建立向外擴展之拓樸，符合樹狀結構設計原則，Sink 成功成為網路中心。

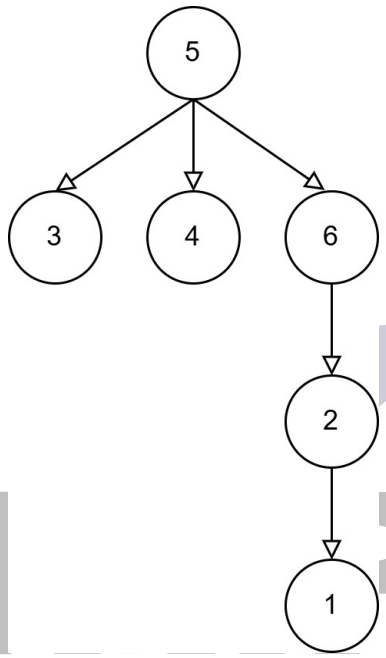

圖 4.19 實作 sink=5 初始化拓樸圖

此兩組實驗皆證實，本研究所實作之拓樸建立演算法在初始化階段能正確識別 Sink 節點，並將其設為網路的 Root。即使在不同位置指定 Sink 節點（如節點 1 或 5），皆可以達到以 Sink 為中心之拓樸結構。

4.3.5.2 實作重啟拓樸中 Sink 點之 Root 角色實驗

在本實驗中，針對已建立之 BLE Mesh 拓樸進行節點重啟測試，以驗證系統在節點重啟後能否正確恢復拓樸結構並保持 Sink 節點的 Root 角色。實驗中，分別將節點 1 及節點 5 設定為 Sink 節點，並透過 UART 日誌紀錄其連線狀態。

在節點 1 重啟後，系統能夠自動重新建立拓樸，並確保節點 1 仍然是整個 Mesh 網路的根節點。圖 4.20 顯示了重啟後的拓樸結構，從中可見所有節點仍然以節點 1 為根節點，完成拓樸建立。

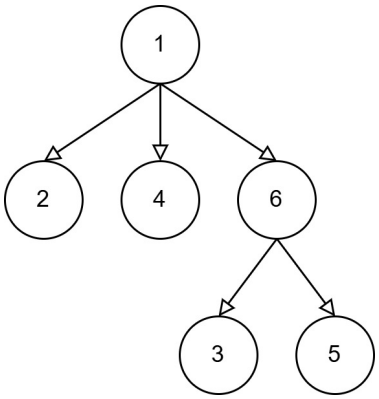

圖 4.20 實作 sink=1 重啟拓樸圖

同樣地，當節點 5 重啟後，系統亦能正確識別其為 Sink 節點，並重新建立拓樸結構。圖 4.21 顯示了節點 5 重啟後的拓樸結構，所有節點仍然以節點 5 為根節點，完成拓樸建立。

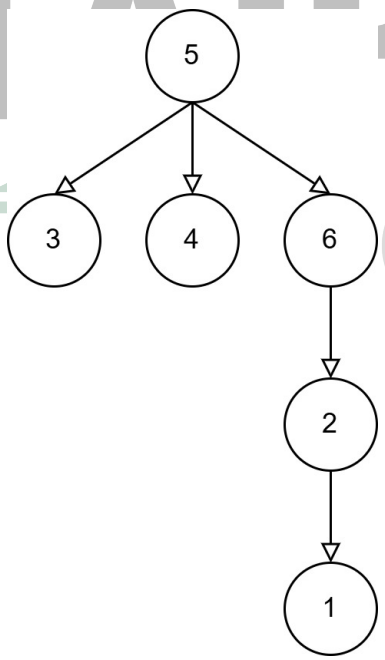

圖 4.21 實作 sink=5 重啟拓樸圖

4.4 調整 Connection Interval 對網路穩定性的影響評估

(Todo)

第五章　結論與未來工作

5.1　結論

5.2　未來工作

參考文獻

[1] Mordor Intelligence. *Internet of Things (IoT) - Market Share Analysis, Industry Trends & Statistics, Growth Forecasts 2024–2029*. Accessed: 2025-06-04. Jan. 2024. URL: https://www.gii.tw/report/moi1403099-internet-things-iot-market-share-analysis-industry.html.

[2] 温淇淼. "基於 FruityMesh 之藍牙低功耗網狀網路傳輸機制的設計以改善 QoS". PhD thesis. 台北市, 2024, p. 41. URL: https://hdl.handle.net/11296/7u74xv.

[3] 吳俊諺. "基於 FruityMesh 之 BLE TDMA 傳輸機制的設計與實現". PhD thesis. 台北市, 2022, p. 48. URL: https://hdl.handle.net/11296/a3g254.

[4] 郭家瑋. "基於 Nordic SoftDevice 之 BLE TDMA 傳輸機制的設計與實現". PhD thesis. 台北市, 2021, p. 55. URL: https://hdl.handle.net/11296/5c352v.

[5] MICROCHIP Developer Help. *Bluetooth Link Layer*. https://microchipdeveloper.com/xwiki/bin/view/applications/ble/introduction/bluetootharchitecture/bluetooth-controller-layer/bluetooth-link-layer. Accessed: Nov. 2023. Nov. 2023.

[6] ángela Hernández-Solana et al. "Bluetooth Mesh Analysis, Issues, and Challenges". In: *IEEE Access* 8 (2020), pp. 53784–53800. DOI: 10.1109/ACCESS.2020.2980795.

[7] Bluetooth SIG. *Bluetooth Core Specification Version 5.0*. https://www.bluetooth.com/specifications/specs/core-specification-5-0/. Accessed: 2025-06-04. 2016.

[8] Eunjeong Park, Hyung-Sin Kim, and Saewoong Bahk. "BLEX: Flexible Multi-Connection Scheduling for Bluetooth Low Energy". In: *Proceedings of the 20th International Conference on Information Processing in Sensor Networks (Co-Located with CPS-IoT Week 2021)*. IPSN '21. Nashville, TN, USA: Association for Computing Machinery, 2021, pp. 268–282. ISBN: 9781450380980. DOI: 10.1145/3412382.3458271. URL: https://doi.org/10.1145/3412382.3458271.

[9] Nordic Semiconductor. *SoftDevices*. https://infocenter.nordicsemi.com/topic/ug_gsg_ses/UG/gsg/softdevices.html. Retrieved November, 2023. 2023.

[10] FruityMesh. *FruityMesh Documentation*. https://bluerange.io/docs/bluerange-mesh-community/Quick-Start.html. Retrieved November, 2023. 2023.