國立交通大學

網路工程研究所

碩士論文

ZigBee 網路之調控者輔助的分散式負載平衡機制

A Controller-Assisted Distributed Load Balancing Scheme for ZigBee Networks

研 究 生：楊宗義
指導教授：曾建超　教授

中 華 民 國　九 十 七　年　七　月

ZigBee 網路之調控者輔助的分散式負載平衡機制
A Controller-Assisted Distributed Load Balancing Scheme for ZigBee Networks

研 究 生：楊宗義　　Student：Tsung-Hsi Yang
指導教授：曾建超　　Advisor：Chien-Chao Tseng

國 立 交 通 大 學
網 路 工 程 研 究 所
碩 士 論 文

A Thesis
Submitted to Institute of Network Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science

July 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年七月

ZigBee 網路之調控者輔助的分散式負載平衡機制

研究生： 楊宗義　　指導教授： 曾建超 教授

國立交通大學資訊學院網路工程研究所

摘　要

本論文針對多重協調器（Multiple Coordinators）之 ZigBee 網路提出一套協調器負載平衡機制。ZigBee 網路是由好幾個 PAN 組成，其中一個協調器負責一個 PAN 的資料接收，並且往後端的伺服器發送；當網路規模日益龐大之時，大量的負載常常集中於少數協調器。本機制由網路中的調控者(可能是後端伺服器等元件)輔助，與網路中的其他網路節點共同合作，決定出位於負載較重 PAN 中哪些節點需要切換到負載較輕的 PAN，以達成負載平衡的目的。

本論文針對具有多個協調器(Coordinator)之 ZigBee 網路提出一套協調器負載平衡機制。ZigBee 網路可以由好幾個個人區域網路(Personal Area Network;PAN)組成，其中一個協調器負責一個 PAN 的資料接收，並且往後端的伺服器發送。當網路規模日益龐大之時，如果網路節點(或流量)分配不均，會導致大量的負載集中於少數協調器。因此本論文提出一套平衡協調器負載的機制，藉由調控者(controller;一般是網路後端的伺服器) 的輔助，網路中的網路節點會共同合作，決定出位於負載較重的 PAN 中，哪些節點需要切換到負載較輕的 PAN，以達成負載平衡的目的。

目前各方所提的各種負載平衡機制，可以簡單分成兩大類，分別是集中式處理與分散式處理。如果採用集中式處理的方法，網路節點需要將負載資訊上傳至伺服器，這些負載更新訊息會造成網路很大的負擔；反之，如果採用分散處理的方法，

網路節點則不需要上傳負載資訊到伺服器，但網路節點間卻需要交換較多的訊息才能完成負載平衡的動作。

在無線感測網路的架構下，對於協調器的負載情形，後端伺服器是較容易知道的，但是網路拓樸以及各自的負載情形卻是散佈於整個網路之中，這些資訊都是負載平衡動作所需要的，因此不論是集中式或分散式作法都需要許多的訊息來傳遞這些參數。本論文提出的這套機制就是希望能夠讓掌握資訊的角色各司其職與共同合作，以減少所需要的訊息。這套機制首先建立子樹負載資訊維持樹（Sub-tree based load information maintenance tree），使得每一節點能夠知道自己底下子樹的總負載量，同樣的伺服器也知道各個協調器的總負載量；之後再由掌握協調器負載情形的伺服器決定該轉換 PAN 的節點個數，至於該從何處，也就是到底哪些節點應該要轉換 PAN，則由各個網路節點跟據自己的子樹總負載量與伺服器決定的量做個比較來作判斷，一次以一個子樹為單位來做切換 PAN 的動作。因此我們的機制是一套介於集中式與分散式處理的作法。

根據模擬結果顯示，我們提出的機制能夠達到跟集中式作法相同的良好平衡表現，在資料傳輸量（Throughput）和資料傳輸延遲（End to end delay）方面表現也良好，但是所需要的訊息量卻遠較集中式與分散式作法都還要來的少。

關鍵詞：無線感測網路、負載平衡、閘道節點、ZigBee

A Controller-Assisted Distributed Load Balancing Scheme for ZigBee Networks

Student : Tsung-Hsi Yang Advisor : Dr. Chien-Chao Tseng

Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University

Abstract

In this thesis, we propose a load-balancing mechanism for ZigBee networks with multiple coordinators. In general, a ZigBee network may consist of a number of Personal Area Networks (PANs) and each of which has a coordinator that is responsible for transmitting the data from the PAN to the fusion center or vice versus. As the size of a ZigBee network grows, some coordinators may be overloaded if too many sensor nodes join the same PANs. In order to overcome the multi-coordinator load unbalance problem of ZigBee networks, we propose a Controller-Assist distributed (CAD) load balancing scheme. In CAD, a controller, possibly a server on the network side, decides the amount of traffics of an overloaded PAN should be reduce, and other ZigBee nodes of the overloaded PAN determine autonomously which nodes should switch to which PAN.

In general, we can classify the load-balancing mechanisms into two categories: centralized and distributed approaches. The centralized approach can achieve better load balance under the expense of more uplink traffic overhead for the ZigBee nodes to update load information in a centralized server. On the contrary, the distributed approach does not rely on a centralized server to collect load information from ZigBee nodes but requires each ZigBee node to exchange load information with its neighbors periodically, and may have ping-pong effects in balancing loads of multiple coordinators.

Neither centralized nor distributed approach best fits the need of multiple coordinator load balance problem. According to the characteristics of ZigBee networks,

the server can know the loads of coordinators easily, but not the loads of all other nodes or information delivery paths. On the other hand, each ZigBee node can know the loads of the nodes that have a direct connection with it, but not the global view of the loads of all PANs. Therefore, it is necessary to exchange many message with the load and topology information in both centralized and distributed approaches.

In light of the above characteristics, the CAD scheme we proposed in this thesis makes each node in a ZigBee network play its role in accordance of its own knowledge of load information and cooperate with other nodes to archive multi-coordinator load-balancing. First, we established a sub-tree based load information maintenance tree so that each node knows the total loads of its sub-tree. Second, the server decides and informs an overloaded PAN how many loads it needs to reduced. Third, each node in the overloaded PAN determines, autonomously in a distributed manner, if it needs to switch to another PAN in accordance of the amount of traffic it needs to reduce and the load of its sub-tree.

The simulation results show that the CAD scheme can achieve the same performance as the centralized approach does in balancing the loads of multiple coordinators, while incurring fewer control messages than both centralized and distributed approaches. Furthermore, it is also very effective in terms of throughputs and end-to-end delays.

Keywords: Wireless sensor network、Load balancing、Gateway、Sink、ZigBee

誌　謝

本論文能順利的完成，首先要感謝我的指導教授—曾建超博士，在過去這兩年來對我的指導與口試期間的細心指點，指導我正確的研究方法與態度，引領我走向正確的研究方向，並且提供一個完整且自由的研究環境。在這學習的過程中，也讓我體認到積極奮發與創意思考為研究之要務。

也感謝我的口試委員：嚴力行博士、蔡文能博士與曹孝櫟博士，感謝他們細心的審查我的論文，並提供寶貴的意見，讓我得以更完善我的論文。另外，也很感謝王瑞堂學長肯在他百忙之中陪同我一起討論與研究，在研究的過程中給予了我很多的建議與啟發，讓我得以完成本篇論文。還要感謝賴俊羽與吳昭男同學陪我患難與共，共同度過數個腦力激盪的夜晚。最後感謝實驗室所有的學長姐、同學、學弟妹們的幫忙與支持，讓我在這兩年之中過得很充實，謝謝你們。最後，我要感謝我的父母與朋友們的陪伴，給予我精神層面上的鼓勵，讓我得以繼續堅持下去。

僅以此論文獻給我親愛的家人，以及所有關心我的師長和朋友們

目　錄

圖目錄

表目錄

第一章　緒論

1.1 研究動機

隨著無線技術蓬勃發展，以及嵌入式硬體技術日益成熟，使得許多體積微小的電子裝置可以內嵌精密感測、計算及通訊等多樣化功能。此類感測器(Sensor Nodes)不但能偵測及感應環境的變化，更能分析所蒐集到的資訊，且透過無線通訊的功能將資料傳回至後端的處理單元供其處理。近年來眾多研究單位以及產業界人士相當看好這類微小裝置的未來前景以及其所能夠發展之應用，比方環境測量、災害偵測或車機載具等應用，甚至也開始考慮在人或其他生物上裝設感測器，例如醫療看護、動植物防護等等[1][2]。

通常一個無線感測網路的架構如 Figure 1-1 所示，首先將大量的感測器佈於待感測區域來蒐集各種資料，再藉由無線網路將蒐集的資訊透過無線資料蒐集器傳回後端以作處理。這些感測器將會自行組成一個無線感測網路，並且具有資料繞送的能力以將資料傳回後端[2]。

由 Figure 1-1 可見整個無線感測網路的資料最後都需要由閘道器(Gateway)來負責接收並轉送後端，若是網路規模漸大勢必需要多個閘道器來分攤過大的負擔。考慮到無線感測網路的應用往往對於即時(Real time)較為要求，一個閘道器負擔較為平均的網路，將會使得資料傳遞較為順暢，且封包不易遺失。因此我們希望能夠研究出一套有效率的機制能夠有效的平均分攤負載，使得無線感測網路的資料能夠即時有效的傳送到後端。

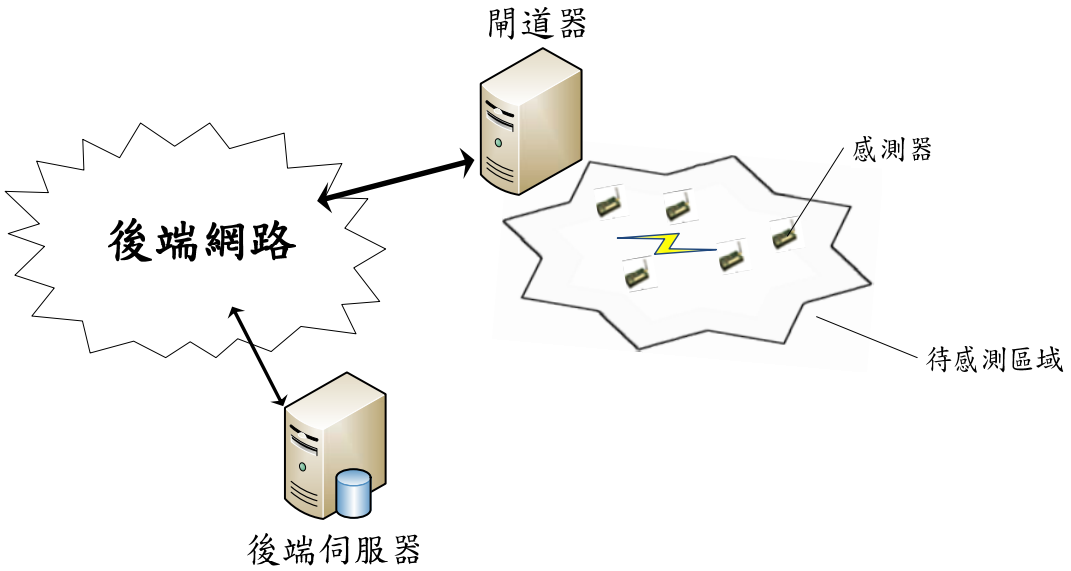


Figure 1-1 無線感測網路系統基本架構圖

1.2 研究目的

我們以目前在無線感測網路應用最為廣泛的 ZigBee/IEEE 802.15.4 技術[3][4]作為主要研究對象，依據標準的定義，整個網路的架構大致仍與前一小節所敘相同，其中閘道器又稱為協調器(Coordinator)，另外所有的 ZigBee 節點可依據是否具有繞送(routing)功能分為可繞送節點與不可繞送節點，顧名思義，可繞送節點具有幫別人轉送封包的功能，而不可繞送節點亦稱為終端節點(End Device)，不具有轉送封包之功能。一個協調器將會負責組成一個個人區域網路(Personal Area Network, PAN)，在同一個 PAN 下的節點擁有相同的網路識別碼(PAN ID)，並且運行於同一個頻道(channel)上。一個無線感測網路則可能由好幾個 PAN 所組成，因此會有數個協調器各自負責不同的 PAN，並且同樣連向共同的後端伺服器，Figure 1-2 為示意圖。

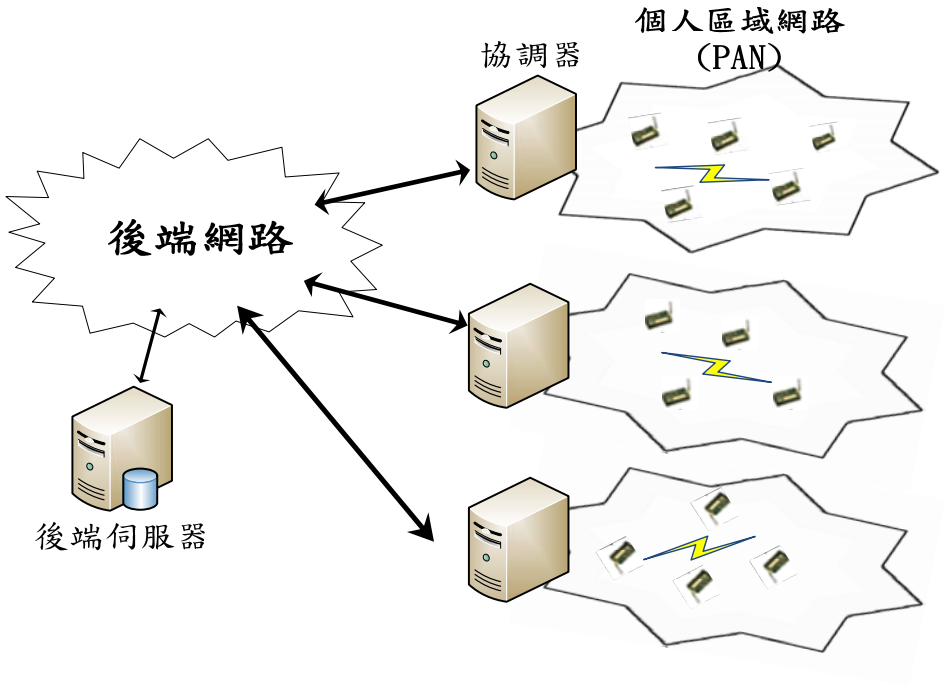


Figure 1-2 ZigBee 網路架構示意圖

現有的 ZigBee 機制中，允許選擇兩種繞送機制中的其中一種作為網路層(Network Layer)繞送協議。這兩種繞送機制分別是 AODV 與 Tree Routing，使用 AODV 即成為一個網狀拓樸網路(mesh)，而使用 tree routing 顧名思義將形成樹狀拓樸網路。這兩種機制分別具有其優缺點，AODV 具有點對點之間能找到較短路徑的特性，但需要的額外負擔較大，且因為路徑建立的較慢使得對於即時性的要求較為不足；相對的 Tree Routing 在節點加入網路取得識別碼(ID)之後，根據識別碼即具有繞送能力，因此資料能夠較能快速傳送(而不需等待繞送路徑建立才傳送)，且無繞送路徑建立的額外負擔，缺點則是在點對點傳送情況下，傳送的路徑可能較長(須要先送至共同祖先再往下傳送)。

考慮到大多數的無線感測網路的應用環境，幾乎大多數的資料流都是以上行至協調器為主，其次是下行各個節點的資料流(如一些控制訊息等等)，點對點的情況非常少見，因此 Tree Routing 成為最適合的選擇，再者其機制本身也相當簡單，易於實作，更利於在無線感測網路這樣的環境下使用。

因此我們主要針對 Tree Routing 繞送機制作研究，發現在樹狀拓撲環境下，各個節點在新加入網路的時候，只考慮距離因素而並沒有考慮到其他像負載情形等因素，即每個新加入的節點將會在各個頻道作掃描，而取得一份有能力做為自己的父節點之清單，在這清單中選出真正的加入連線對象的依據，僅以深度作為考量。參考 Figure 1-3，舉出四個例子作為說明，紅色節點為新加入網路的節點，由於左右兩邊發現到可加入的節點深度皆為二，因此兩邊有相同的機率會被選為父節點作為加入的對象，若是選擇加入右邊的 PAN，如(a)與(b)所示可能加入到節點數較多的 PAN，或者如(c)所示加入到耗電量較多的 PAN，或者如(d)所示加入到資料流量較大的 PAN，遂產生負載分部不平均的現象。

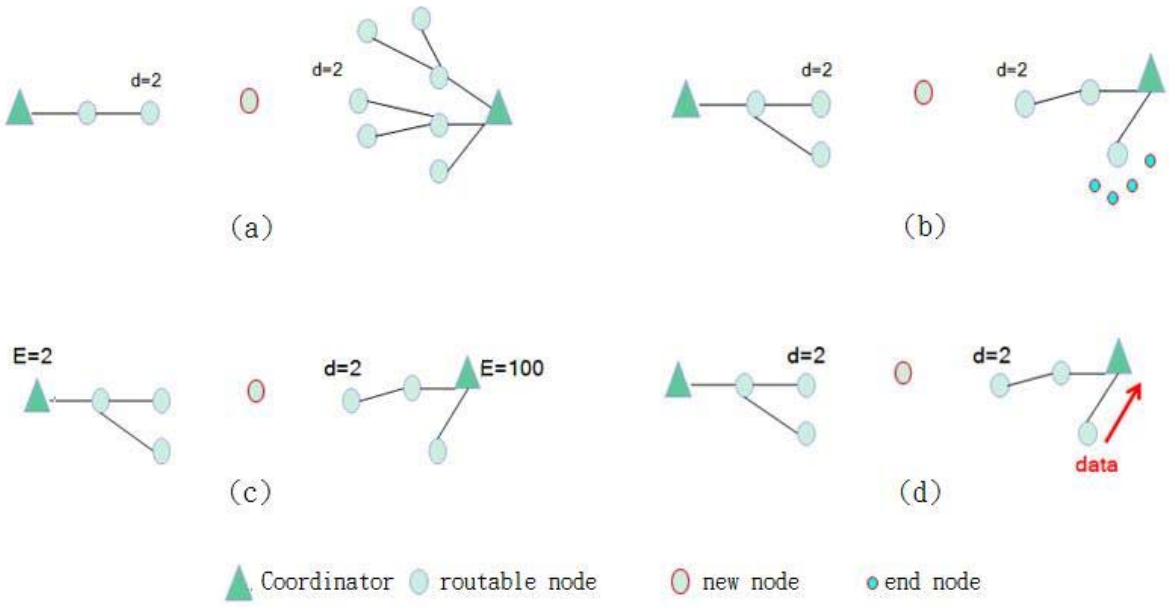


Figure 1-3 ZigBee 新節點加入所遇到之問題示意圖

因此，在原來的 ZigBee 樹狀拓樸網路下，多個協調器容易出現負載過份集中於其中少數的情形，由於協調器本身需要負責整個 PAN 的資料接收，本身負載已是相當大，協調器的負載分佈不均將使得網路擁塞的情形更為嚴重，使得許多即時性且重要的資料較慢且較不易地傳達至後端伺服器。

總結以上，我們希望基於 ZigBee 樹狀拓樸環境之上，設計出一套完整的協調器負載平衡機制，能夠使得負載平均分散於各個協調器之上，其中我們必須要能夠克服與滿足以下需求：

- 能夠讓節點在運行不同頻道的 PAN 之間切換，也就是從負載較重的 PAN 退出轉而加入負載較輕的節點。
- 可以針對不同的管理需求使用不同的參數依據作為負載的指標，例如資料流量(traffic load)、耗電量等等，並且能夠同時將其他因素如繞送路徑距離等一齊作考量。
- 由於無線感測器本身所配備之電源、記憶體以及運算能力均受到極大的限制，因此我們所設計出來的機制必須本身不能帶來太多額外負擔，必須能夠簡單有效的達到負載平衡的目的。

1.3 章節簡介

關於這篇論文接下來各章節之內容簡述如下：第二章研究背景與相關論文研究會先針對本論文的基礎網路環境 ZigBee 這項技術做個說明，另外再介紹以往對於負載平衡這方面的問題所做過的研究與大家提出的機制方法，這些方法將會依照其作法的特性分成集中式與分散式處理兩大類，最後我們也將會分別列出這兩大類作法各自的優缺點。第三章 Controller-Assisted Distributed (CAD) load balancing scheme 就開始介紹我們所設計的負載平衡機制，首先會先說明一下基本的想法以及想要達到的目標，然後才會介紹真正的基本機制，之後再針對這些機制的一些細節處做些探討，提出一些方法來讓這套機制能夠較快較有效率的達到負載的平衡。第四章

Simulation 裡我們針對我們的方法做模擬，並且與其他的方法做個比較。最後第五章結論與未來工作的部分將對此篇論文作一個總結，並對未來能夠繼續研究的方向提出一些心得與看法。

第二章 研究背景與相關論文研究

2.1 ZigBee Network

前面提到 ZigBee 有兩種網路拓樸與繞送機制，分別是網狀拓樸，使用的是 AODV 演算法，以及樹狀拓樸，利用識別碼的特定分配方式達到繞送功能。本篇論文所作的研究主要針對樹狀拓樸，因此這裡針對樹狀拓樸的新節點加入、識別碼分配以及繞送機制分別做個說明。

2.1.1 新節點加入

一個 ZigBee 節點在加入網路之前，需要先使用 passive scan 或者 active scan 以得知目前所在之處在各個頻道上有哪些網路節點的存在，使用 passive scan 表示節點會在每個頻道間切換，每次停留某一特定的時間，這段時間內節點會監聽此頻道中是否有其他網路節點發出 beacon 封包。若使用 Active scan，則節點同樣在個頻道切換，但不同處在於節點將會發送一個詢問的封包(probe request)，在此頻道中的其他網路節點收到這個詢問封包，必須馬上回應一個 beacon，因此 active scan 將不需要在每個頻道逗留到像 passive scan 這麼久的時間。網路節點所發的 beacon 將會攜帶許多其本身的資訊。其他部分細節與底層的 MAC 層與 PHY 層有較大的關連，和我們的研究較無相關，於 IEEE 802.15.4 standard 有充分之說明，在此便不再贅述。

於各個頻道掃描完成後，接下來的工作便是從這份清單中選出一個節點作為加入的對象(association)，如同前面有提過的，選擇的依據是根據這些節點在樹狀拓樸中的深度作為考量，深度的資訊是由這些節點所發出的 beacon

中所攜帶，參考 Figure 2-1，每個節點上面的數字表示其在樹狀拓樸中的深度，每個 PAN 以協調器作為樹根，往下形成一棵樹，樹根的深度定義為零，往下遞增，圖中的藍色節點 b 代表新加入的節點，經過掃描之後發現有節點 a 與節點 c 的存在，此時根據深度發現是節點 c 比較淺，因此節點 b 將選擇節點 c 作為父節點開始進行連線動作(association)，之後便會加入到 PAN2。

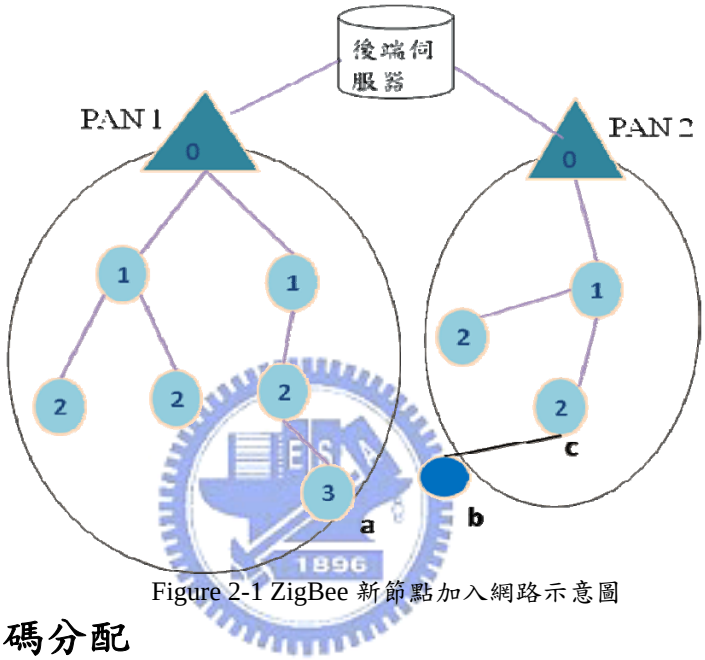


Figure 2-1 ZigBee 新節點加入網路示意圖

2.1.2 識別碼分配

前一小節提到新節點會與現存的網路節點進行連線動作，在這連線動作過程中，新節點將會傳送一個連線請求封包(association request)給父節點，接下來就會進行識別碼的分配動作。

ZigBee 網路位址分配方法，在確定每個 PAN 的最高連線數及連線層數等限制後，會利用運算公式求出每個節點可以分配的識別碼，藉由這些有規律的識別碼分配將可以達到繞送封包的效果。

首先介紹幾個需要事先設定好的參數，分別有規定每個節點最多能夠有幾個子節點的參數 Cm (nwkMaxChildren)，以及整個樹狀拓樸的最大允許深度

Lm (nwkMaxDepth)，還有每個節點最多允許有幾個可繞送節點作為其子節點的 Rm (nwkMaxRouters)。藉由這些參數就可以代入一個特地的函數 Cskip(d)，其中 d 表示深度。其所得到的值其實就是代表深度為 d+1 的節點，以其作為樹根之下的整顆子樹最多能夠連結的節點個數，Cskip(d)函數的定義如下：

$$Cskip(d) = \begin{cases} 1 + Cm \cdot (Lm - d - 1) & , \text{if } Rm = 1 \\ \dfrac{1 + Cm - Rm - Cm \cdot Rm^{Lm-d-1}}{1 - Rm} & , \text{otherwise} \end{cases} \quad (1)$$

一個新節點向父節點發出連線請求(association request)的時候，父節點先跟據自己的深度求出 Cskip 值，接著根據此新節點屬於可繞送節點或者終端裝置(End Device)而有不同的分配策略。若屬於可繞送節點，假設此新節點是屬於第 n 個加入自己的可繞送節點，令其應該被分配到的識別碼為 R_n，父節點的識別碼為 A_{parent}，將根據底下公式 2 求得此值：

$$R_n = A_{parent} + Cskip(d) \cdot n \quad (2)$$

其用意在於為每個可繞送子節點預留一段識別碼位址空間，因為這些可繞送子節點將來可能會有其他新節點陸續加入到其子樹下。

至於終端裝置的識別碼分配規則，將被分派在所有的可繞送節點的識別碼位址之後，假設其應該被分配到的識別碼為 R_n，根據底下公式 3 將求得此值：

$$R_n = A_{parent} + Cskip(d) \cdot Rm + n \quad (3)$$

在完成識別碼的分配之後，新節點與父節點都需要記錄彼此的識別碼，

之後繞送封包時便需要這些資訊。

2.1.3 繞送機制

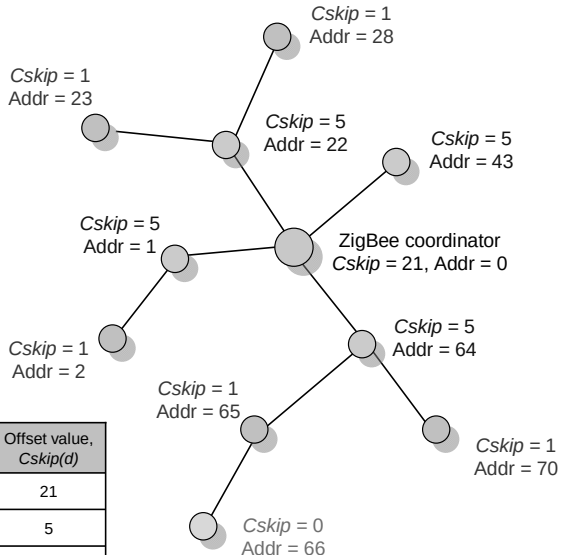
使用樹狀拓樸的最大好處就是在繞送封包的時候，不需要像 AODV 等演算法那樣啟動繞送路徑建立(routing path setup)程序，而是只需要根據識別碼就能夠知道往哪裡轉送封包。

參考 Figure 2-2，假設其參數設定為 nwkMaxChildren = 4, nwkMaxRouters = 4, and nwkMaxDepth = 3，一開始協調器的識別碼一定是零，依照前一小節所提，根據深度算出 Cskip 值為 21，因此被協調器分配出來的識別碼分別是 1, 1+21, 1+21*2,然後直到超過最大的位址空間(address space)，或者已經超過 nwkMaxChildren，其他節點依同樣的方式分配識別碼。

每個節點在繞送封包的時候遵循的一個簡單的規則：判斷目的識別碼是否在自己的子樹底下，如果是就往子節點送，如果不是就往父節點送。判斷是不是位於自己子樹底下很簡單，如果目的識別碼的值大於自己的識別碼，且小於自己的識別碼加上上一層的 Cskip 值時，就表示介於預先分配給自己的位置空間中，所以位於自己的子樹底下。以 Figure 2-2 為例，假如識別碼 22 的節點收到一個欲送往 28 的封包，由於 22<28<(22+21)，因此得知這封包將是要送往自己底下子樹的封包。

若是位於自己的子樹底下，下一步就是要找出該往哪一個子節點送。每個 ZigBee 節點都會有一個表格記錄自己底下的各個子節點相關資訊，運用相同的公式，每個子節點其底下的子樹的位址空間為子節點的識別碼加上自己的 Cskip 值，例如 Figure 2-2 中識別碼 22 的節點有兩個子節點，其底下的子

樹位址空間分別是 23~27(23+5-1)以及 28(23+5)~32(23+5+5-1)。同樣的根據封包中的目的識別碼座落於哪一個區段中，就知道究竟目的地位於哪一個子節點以下的子樹，封包便往那個子節點傳送。



Depth in the network, *d*	Offset value, *Cskip(d)*
0	21
1	5
2	1
3	0

Figure 2-2 ZigBee 樹狀拓樸與識別碼分配示意圖

2.2 相關的無線網路負載平衡機制

底下將以往許多針對無線網路負載平衡所提出的機制，根據不同處理方式分為兩大類，分別是由各個節點各自決定的分散式處理方式，以及統一交由一個中央的管理單元來控管的集中式處理方式。

2.2.1 分散式處理

網路中的節點在繞送封包時，如果能夠讓資料流能夠分散於不同路徑，就可以讓這些瓶頸節點的負載較為平衡，因此底下再根據不同的繞送策略分為兩類:

- Request on Demand

這類的作法所使用的繞送策略是 on-demand 式的，就是只在要送封包時才開始尋找與建立繞送路徑。許多的作法是藉由廣播的方式(flooding)，意即發送端(source)發送繞徑發現詢問(route discover request)封包時，這些封包在網路中會被廣播出去。在這過程中，建立繞送路徑的方法有兩種不同的方法，一種是中間轉送的節點(intermediate node)會把自己的位址放入封包中，然後繼續轉送下去，因此最後這個封包上就會記錄有路徑中每個節點的位址，例如 DSR[5][10]。另一種是中間節點收到這些封包，會建立一個繞送表格記錄是誰送給自己，因此有別於 DSR 是完整知道整個路徑，這類作法只會知道封包要送往的下一個節點，透過每個節點的繞送表格，就會順利的送到目的地，例如 AODV[8]。

當目的端(destination)收到此詢問封包後，會回應一個繞徑發現回應(route discover reply)封包，然後依著發送時建立的路徑延路送回發送端，當發送端收到此回應封包即完成整個流程。

因此沿途中幫忙轉送這些詢問與回應封包的節點若將自己負載的資訊(如耗電量、資料流量等)夾帶於封包之中，當發送端收集好這些回應封包建立起路徑時，也已經知道目前不同路徑各自的負載狀況，遂可依此作為依據作路徑選擇(path selection)，達到負載平衡。

DLAR[9]算是在 AODV-like 的作法中較具代表性的一個負載平衡繞送演算法。基本的精神大同小異，有幾點不同之處：

I. 只有目的節點能夠回應繞送發現封包

這其實是許多負載平衡繞送演算法的特性，傳統繞送演算法為了加快繞送路徑建立，並且減少繞送訊息，因此允許中間節點在收到繞送發現封包時，若發現自己暫存(catch)有相關的繞送資訊，就可以自己回應繞送回應封包，就不需要再繼續廣播下去。然而這些被暫存的資訊已經是過時(stale)的資訊，為了跟據最新最即時的負載情形來做繞徑選擇，因此只准許目的節點能夠回應，讓封包能夠完整經過路徑中的每一個節點，記錄最新的負載資訊。

II. 負載定義

對於負載的定義在各家研究中各有不同，在 DLAR 這裡它以每個節點目前的傳送封包佇列大小代表負載情形。每條路徑的節點加總負載就代表這條路徑的負載情形。

III. 動態更換路徑

在路徑建立完成後，各個節點傳送封包時，可以把自己最新負載資訊附加在資料封包中(piggyback)，由目的地節點負責監控，當此路徑的負載超過一個門檻值(threshold)，便重新選定另一條路徑。至於如何重新選一條路徑的方法，可以由目的地節點啟動一個針對來源節點的繞送路徑發現程序，也就是如同前面所提的發送繞送路經發現封包，並且透過廣播的方式送達來源節點。此時來源節點不需要回應封包，因為此時已經建立好來源至目的間的繞送路徑了。

在本論文最後的實驗部分，分散式的負載平衡演算法，即 DLAR 將會是我們的一個比較對象，然而我們對於負載的定義則是取各個節點所接收到的資料量。

- Table Driven

相反的，另一類策略則是平時就會維護(maintain)繞送的資訊。最常見的方式是利用坡度繞徑演算法[11][12]的觀念與方法，由這些易成為瓶頸的閘道器等節點主動向外發出訊息，這些訊息會被廣播至整個網路(flooding)，讓整個網路之節點皆可藉此以建立起到不同閘道器的路由與繞送所需的花費。如 Figure 2-3 中兩個閘道器各自發出建立路由資訊的封包，並且在網路中不斷的廣播，每當收到封包後會將封包中的 hop count 數加一然後繼續轉送下去，因此可藉由收聽這些封包來建立到閘道器的最短路徑，圖中各個節點中的數字表示節點透過聽取這些封包所取得的最淺的深度(距離閘道器的 hop count)，如果閘道器也把自己的負載情形也放入此封包之中，其他節點就可以藉此來選擇適當閘道器完成負載的平衡。

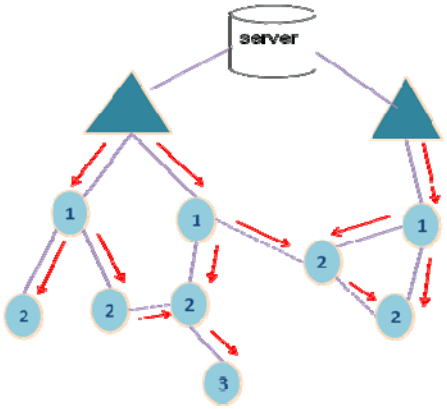


Figure 2-3 坡度繞境演算法示意圖

另有研究[14]提出一種採由底至上(bottom-up)的平衡生成樹建構方式，從最底層開始，每一個節點都會把自己以及自己的子節點的連接狀況與負載狀

況資訊傳送給自己的父節點，由父節點來負責幫自己與自己的兄弟節點之間調整出一個平衡的狀態，因為父節點如前所述的已經由自己與自己的兄弟節點們上傳的資訊，而得知底下的資訊，父節點藉由仲裁其孫節點(即自己的子節點)該重新選擇其他新節點作為父節點的方式，完成負載平衡的動作。

以 Figure 2-4 為例，(a)部分的左半部黑色虛線框起來的部分可以看到是一個不平衡的狀態，此時當訊息送交到紅色的節點時，如(b)的部份所示，透過底下節點的更換父節點動作，新的樹是一個較為平衡的生成樹。同樣的動作繼續往樹根方向重複的往上做，直到抵達樹根時，當樹根做完自己的子節點間的平衡之後，就得到一棵完整的負載平衡樹了。這個做法的主要缺點除了需要在網路中互相交換非常多控制訊息(Control Message)外，由於採用以底至上的貪婪演算法方式(greedy method)，因此會呈現底部節點彼此之間維持平衡，但上層節點與上層節點之間卻較不平衡的情況，然而就真實的考量面來看，通常最容易成為瓶頸的卻是較靠近樹根的這些上層節點，因此較無法提供這些部分的負載平衡成為這類作法的最大問題(如 Figure 2-5)。

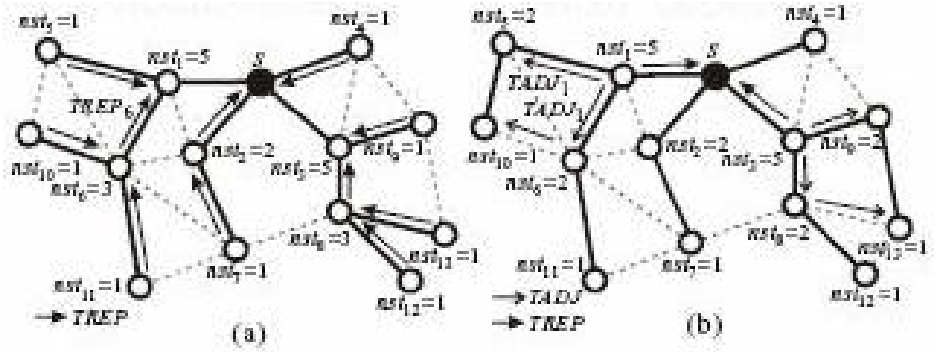


Figure 2-4 分散式負載平衡樹建立示意圖 (引自[15])

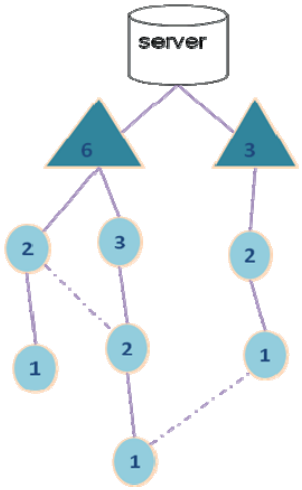


Figure 2-5 分散式負載平衡樹缺點示意圖

上述的各種分散式處理方式，優點是對於行動節點之網路有較佳的支援性，即適合變動頻繁的網路。

然而其都有個共同的缺點，就是由於需要資訊的傳遞，造成網路因傳遞訊息造成額外的負擔。且當網路規模很大的時候，要散播這些資訊就會需要較久的時間，通常會有反應較慢，無法即時動態反應之問題。另外像坡度繞徑演算法那類的作法，由於讓每個節點自行選擇閘道器，有可能導致乒乓效應(ping-pong effect)的問題，也就是在這一刻有許多節點同時選擇閘道器 A 作為出口，因此使得閘道器 A 的負載瞬間增加，因此在下一刻大家察覺到了以後，同時一齊選擇閘道器 B 作為出口，一直這樣循環下去，使得網路一直無法達到一個穩定平衡的狀態。

另外當網路有各個子區域分屬不同頻道(channel)的情況時(如 ZigBee 不同的 PAN 可使用不同 channel)，就很難單靠繞送演算法來發現跨越頻道的路徑。

2.2.2 集中式處理

另外一類作法[13][14]是利用一個中央集中式的控管方式來達到負載平衡，因此需要各個節點將各自的資訊送交至一個統籌管理單元，以供其處理

與控管各個節點的繞送行為。

網路上的節點把自己的能力資訊、周圍鄰近的節點資訊等上傳至一個統一的伺服器，伺服器在收到這些資訊以後，便有能力掌握整個網路的狀態，能夠計算出網路的拓樸情形，根據這些被上傳的負載資訊生成一個平衡的生成樹，然後將此計算出來的樹狀拓樸廣播給網路的各節點，之後各節點根據此拓樸來繞送，因而達到負載平衡的效果。

集中式處理的優點，因為交由一個集中的角色來統籌分配，因此較能達成最佳解(optimal solution)，且不會有分散式處理的乒乓問題。

然而其缺點就是整個網路都要上傳許多資訊給中央伺服器，是故仍無法避免網路中過多的訊息傳遞，若是節點的具有移動性時，需要上傳更新資訊的頻率將更為頻繁，另外集中式的處理也造成這些負責控管的單元之負擔，當網路規模龐大時可能成為瓶頸(bottleneck)。

2.3 總結

我們將負載平衡演算法大致上分為兩大類，分別是分散式與集中式演算法，分散式的作法不需要一個集中管理的伺服器，通常是利用各個節點在繞徑選擇時選出負載較輕的路徑；而集中式則需要一個能夠掌控整個網路的伺服器，並求由伺服器來決定各個節點如何繞徑。

底下分別條列整理出兩種作法的優缺點：

- 分散式
  - 優點
    1. 不需要伺服器管理，彈性較大
    2. 對於節點可能會移動的無線感測網路，使用分散式作法較為合適
  - 缺點
    1. 需要的控制訊息(control message)非常多，例如像 DLAR 那樣需要在整個網路廣播訊息(flooding)
    2. 有些作法可能導致乒乓效應(ping-pong effect)
- 集中式
  - 優點
    1. 由於交由統一伺服器管理，在最後負載平衡的效果上可能可以得到較好的表現
    2. 不會有乒乓效應問題
  - 缺點
    1. 完全的集中式處理導致需要上傳的訊息也非常多，包括自己的

負載資訊、周圍鄰居節點資訊等等。另外當節點移動，也需要重新上傳資訊。

2. 伺服器可能隨著網路規模越大而無法負荷

第三章 Controller-Assisted Distributed (CAD) load balancing scheme

3.1 基本精神與目標

觀察前一小節集中式與分散式作法，整理出問題的癥結點：

- 通常形成瓶頸的節點大多位於較上階層的節點(因為需要負責較多的節點)，因此對於這些需要負載平衡目標節點的負載資訊，是位於上層。(所謂上層可以理解為樹狀拓樸中較靠近樹根的節點)
- 對於整個網路的連通情形以及各自對網路產出的負載等資訊是散佈於整個網路的各個節點上的。

是故我們思考能否採用一種介於集中式與分散式之間的方式，讓上述的這些各種負載平衡所需的各個資訊儲存於各自適當的地方，並且讓大家各司其職互相合作來完成負載平衡的動作。首先採用集中式的伺服器啟動負載平衡動作，由於在無線感測網路架構中，伺服器本就位於那些閘道器之後，因此要知道這些閘道器的負載情形並不是什麼難事。伺服器觀察到閘道器的負載有不平均的現象時，便開始啟動負載平衡機制，我們並不讓所有決定權都集中於伺服器上，既然網路的連通情形與個節點的負載情形是散佈於整個網路之中，因此強求集中於伺服器處理就會有集中式處理的缺點，也就是大家需要上傳許多資訊。我們希望能夠設計出一套由伺服器決定啟動負載平衡機制，並且由伺服器與底下各個網路節點共同決定該如何切割網路。

同時我們希望設計出來的機制能夠根據網路管理者的不同管理需求，將網路切割成若干適當大小之 PAN。所謂不同的管理需求，比方說應用於較著重即時性的應用，就考慮將資料流量大小作為負載依據；如果在意的是系統存活時間(System Lifetime)，那就考慮將耗電量作為負載依據。也就是我們只是提出一個能夠完成閘道器負載平衡這個目的之機制，套用資料流量、耗電量或節點個數等參數皆適用。

3.2 基本機制

我們所設計的機制基於 ZigBee 的樹狀拓樸機制之上，也就是一開始節點仍需要選擇父節點並且取得識別碼，在整個樹狀拓樸形成之後，再利用我們的方法完成協調器的負載平衡。在此我們為方便解說，先以平衡每個協調器下所管轄的節點數為例，並且我們先以兩個協調器及其所屬網路間的平衡負載做說明，在後面章節將會說明多個協調器之間如何做平衡負載。首先本機制會建立一顆子樹負載資訊維持樹(Sub-tree based load information maintenance Tree)，在此樹中每個節點會帶有目前自己以及自己的子樹之總共的負載，在此例即為節點數，事實上這棵樹就是 ZigBee 機制下所形成的樹，只是我們多記錄了負載的資訊。如 Figure 3-1：

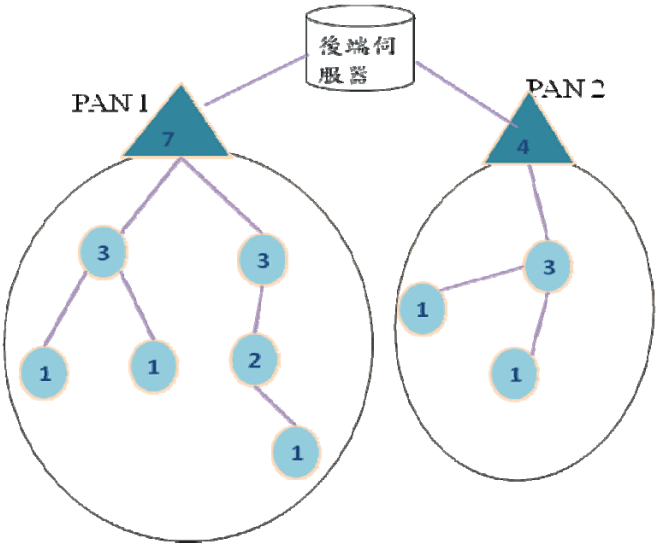


Figure 3-1 負載資訊維持樹示意圖

Figure 3-2 表示如何建立與維護這顆樹，圖中最底下假設有一個新加入的行動節點出現，當它依據先前的 ZigBee 協定加入網路以後，便送出一更新的訊息往後端伺服器，沿路經過的節點在收到此訊息後就將自己所帶的值增加，並且繼續往上傳送。

同樣的，隨著不同情況與不同的參數考量，這個更新的訊息所帶的欲增加的值便會不同，例如電量可能就是耗電量，或是資料量大小等。

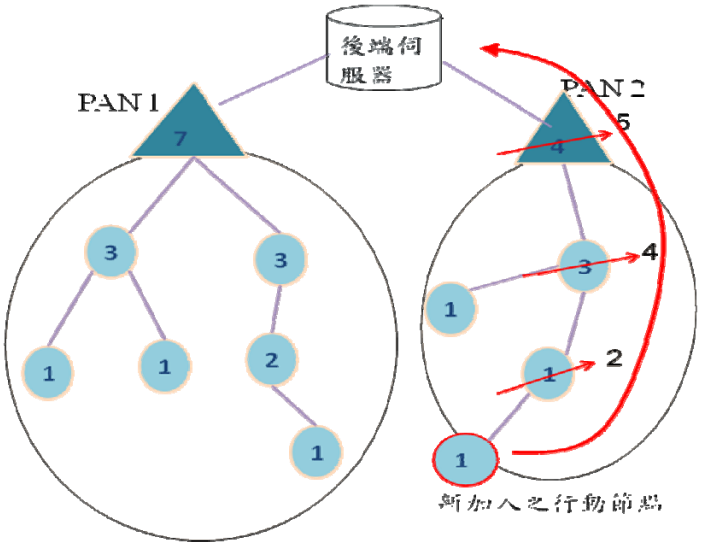


Figure 3-2 負載資訊維持樹之更新示意圖

接下來見 Figure 3-3，如果希望能夠讓負擔較重的 PAN 1 把一些網路節點

轉換到 PAN 2 的話，就還需要掌握網路中的拓樸連通資訊，如果能夠知道哪些節點有機會可以連上不同閘道器所屬的兩個個人網路(PAN)，例如 Figure 3-3 底下黑色框框所圈起來的兩個節點（節點 a 與節點 b），伺服器其實並不需要知道全部的拓樸資訊，而只需要記住這些節點就好，所以額外的負擔大大的減少。

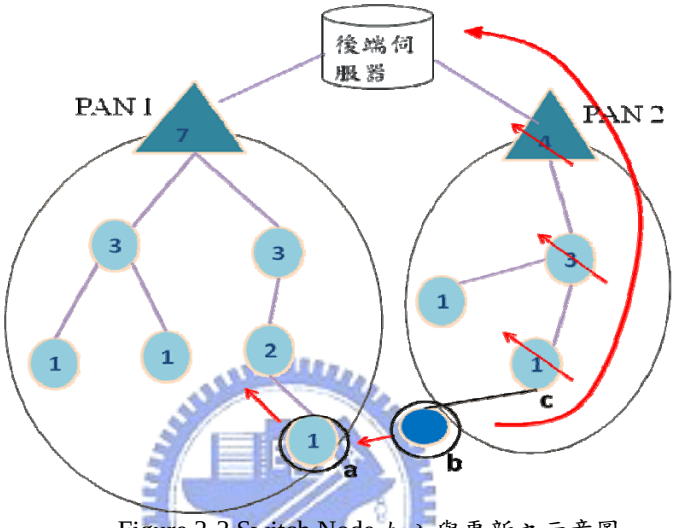


Figure 3-3 Switch Node 加入與更新之示意圖

整個動作可以從 Figure 3-3 來看，底下藍色實心的節點表示新加入的節點，當它掃描完各個頻道後，發現目前有兩個網路可以連入，接著便選擇加入 PAN2(在 ZigBee 網路環境下會選擇樹狀拓樸中深度較淺的節點加入)，則第一步在它加入之前會先使用 PAN 1 的頻道送一個通知的訊息告知節點 a，節點 a 被告之後亦會通知其父節點以便之後容錯(Fault Tolerance)之用。

接下來第二步即選擇節點 c 作為父節點加入 PAN 2。

第三步則如同前面的建立與更新資訊維護樹之程序一般，送一更新訊息至後端伺服器，只是此時再多夾帶一資訊為節點 a 與節點 b 的資訊，以及它們各自所屬的 PAN ID (一個識別不同閘道器所屬的網路的標識符)，還有這兩

個 PAN 所屬的頻道(channel)，藉此後端伺服器便可記錄這些節點，對於這些節點我們稱之為轉接點(Switch Node)，它們將被成對地記錄在伺服器上。

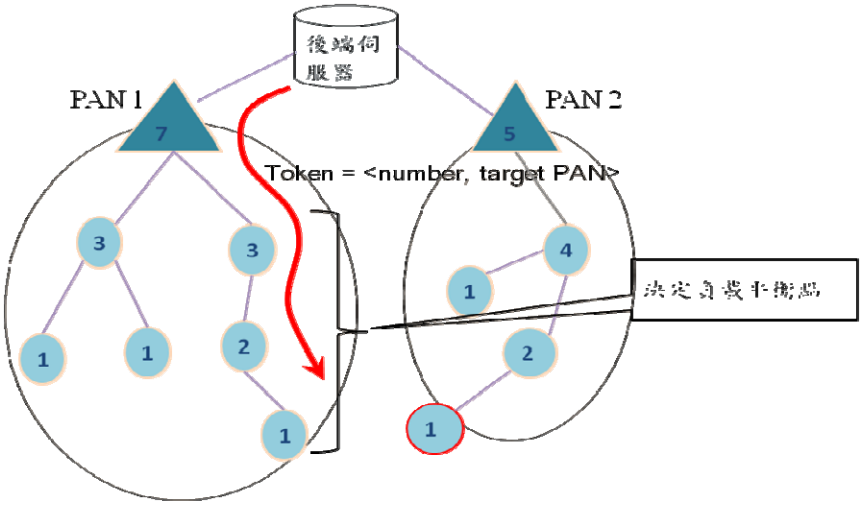


Figure 3-4 伺服器發命令啟動網路調整之示意圖

至此後端伺服器已可知道各個閘道器目前的負載，當負載不平衡時，伺服器可判斷較重的一方需要切除多少個節點轉移至較輕的一方。如 Figure 3-4，伺服器可查詢先前記錄的轉接點中，找出一對轉接點是分屬於此兩個不平衡網路的，接著發出一命令（token）送往位於較重網路中的轉接點，此標記中帶有一值表示伺服器判斷目前網路應切除多少個節點數。接下來就是如何找到負載平衡點的動作，並且以這個負載平衡點為界線將網路切割，將數個節點切換到另一個網路，藉此讓兩個網路的負載較為平衡。利用此標記在隨著樹狀拓樸沿路傳遞中，沿途的每個節點收到此標記時，便會將自己所維護的數值與標記中的數值作比較，由先前之說明可知，這些節點所維護的數值及代表其包括自己及其子樹的總共負載，因此若此數值小於或等於標記中的數字，即表示從此節點以下之子樹正是適合轉換到另一個負載較輕的網路的部分，亦即所謂的負載平衡點。透過這樣的技巧便可快速有效的找出負載平衡點並且切割調整網路，我們稱之為調控者輔助的分散式負載平衡機制。

切割調整的作法可由此節點往其下之子樹廣播一通知訊息，要求大家轉換至另一網路，並且同時此節點需要回復一個 ACK 訊息給伺服器，告知伺服器目前此網路已切除多少負載（即等於此節點所維護的數值，在此例中即為實際切除多少的節點數），此訊息將會沿著資訊維持樹之拓樸往樹根送去，最後送至伺服器，沿途經過的節點幫忙轉送之餘，也順帶的根據此訊息更新自己所維護的數值，伺服器也藉此更新各個網路的最新負載情形。接下來這些節點如何加入新網路便根據不同網路技術而不同，既然我們是基於 ZigBee 網路環境之上，這些節點接下來就是根據先前章節所述的新節點加入網路之程序，先轉換到另一個網路所屬的頻道，並且開始收聽此網路下的節點所發出的訊標封包(beacon)，這些訊標封包中會帶有發送者在整個樹狀網路拓樸下的深度，接著選擇深度最淺的節點作為自己的父節點，開始進行加入的程序(association)。

我們以底下 Figure 3-5 說明整個機制的運作過程，圖中藍色實心點表示心加入網路的節點，當它在掃描時發現同時有兩個 PAN 的節點可供加入，因此得知自己可成為轉接點。當它選擇加入底下的 PAN 時，發出更新訊息上傳至伺服器，除了讓沿途的節點更新負載資訊之外，也告知伺服器目前存在的轉接點資訊。伺服器根據兩個協調器的負載情形發現處於不平衡的狀態，由於存在轉接點表時能夠做網路切割調整的動作，因此伺服器便發出一個命令 Token(藍色箭頭)目的地為黑色圈圈的轉接點，Token 中的數值為 2 表示希望切除 2 單位的負載(也就是兩個節點)，這個 Token 首先經過負載量為 5 的節點，由於負載量大於 Token 中的數值，因此繼續往下轉送，當負載量為 2 的節點收到 Token 之後，便決定自己是適合切除的點，意即負載平衡點之所在，

因此它首先回覆一個 ACK 封包(綠色箭頭)，其中包含自己的負載量(2)，告知伺服器實際切除的負載量，在 ACK 送至伺服器之沿途中的轉送節點，也跟據此 ACK 封包更新自己的負載資訊。

接著負載量為 2 的這個節點便往自己底下的子樹廣播一個要求大家切換網路的封包，指定轉換到底下的另一個 PAN，接著大家便如同新節點加入網路一搬的動作，開始掃瞄並加入底下的 PAN，只不過不同之處在於此時是有指定要加入哪一個 PAN 的，而非單純根據深度來做選擇。

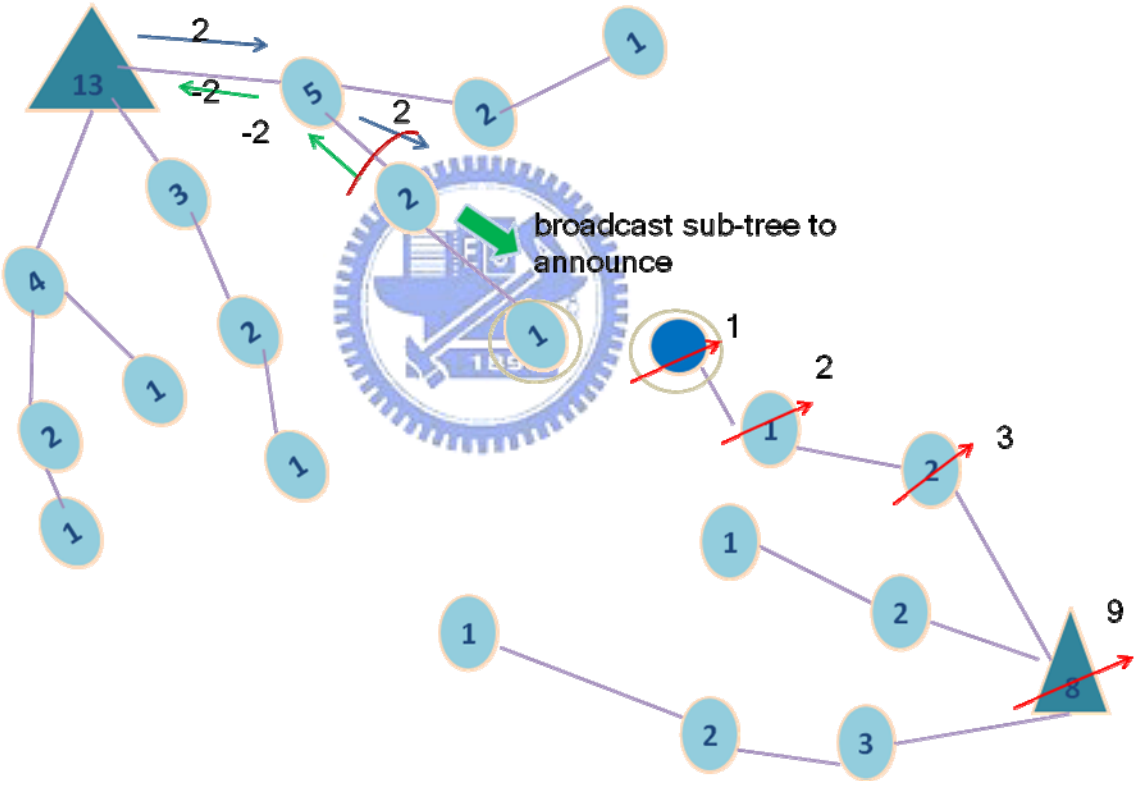


Figure 3-5 完整負載平衡機制運作示意圖

3.3 細節討論

接下來進一步探討一些細節部分。首先我們可以很容易發現，在本機制中轉接點(Switch Node)扮演著非常重要的角色，能否平衡負載全看是否存在轉接點介於兩個網路之間。此外由於無線感測網路節點可能會因為移動或者電

力用盡，使得原本的轉接點可能變成普通節點，或者新的轉接點產生，因此被記錄在伺服器上的資訊就需要有更新的機制。

另外我們也將在此探討如何加快整個負載平衡的流程使得網路能夠更快的達到平衡的狀態，以及最後會討論當整個無線感測網路包含兩個以上的 PAN 的情況。

3.3.1 Switch Node 記錄之更新

關於轉接點資訊的維護部分，由於節點的可移動性或節點的折損，或是由於作平衡負載的切割調整動作所造成的變動，皆會使得伺服器所記錄的資訊成為錯誤的資訊，因此需要有一些機制來做更新，底下分為兩種情況來做解說。

I. 由於負載平衡機制之切割動作所造成的變動

當切割動作發生後，原為轉接點之節點可能會變成普通的節點，如 Figure 3-6。由於不同網路的轉接點具有成對的關連性，只需要其中一個轉接點的資訊，伺服器就有辦法找出相對應的轉接點有哪些，因此只要讓那些原本是轉接點的節點，在加入新網路之後，把自己舊有的識別碼隨著要上傳伺服器的更新訊息一起攜帶，伺服器在收到這個就識別碼之後，就能跟據此註銷舊識別碼所對應的那些轉接點記錄。

另外隨著切割動作，會使得有些節點成為新的轉接點，如 Figure 3-7。這個情況其實可以把它視為新節點加入網路的情況，因為這些被要求轉換網路的節點，也是依照先前的加入網路的程序加入指定

的網路的。這些節點在各個頻道掃描之後，若發現舊有頻道上存在的那些舊鄰居節點，因此就知道自己將屬於新的轉接點，於是在加入新網路之後就如同先前所述的在更新訊息中放入自己以及相對應的轉接點資訊。

這邊為了加速加入新網路的速度，伺服器在發出命令的同時，可以同時告知目的網路所處的頻道等資訊，因此這些節點在掃瞄時，只需要掃描自己原來的頻道(為了知道舊鄰居有哪些，如果平時就有追蹤鄰居節點的訊標(beacon)封包的話就不需要)以及新網路的頻道，因此加入新網路的速度可以較快。

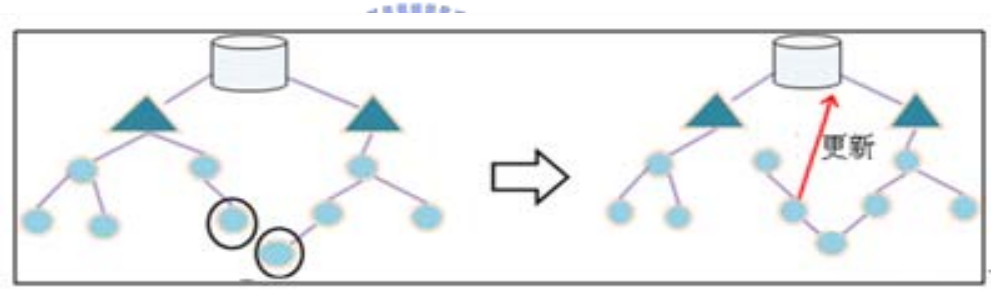


Figure 3-6 轉接點更新成為普通節點之示意圖

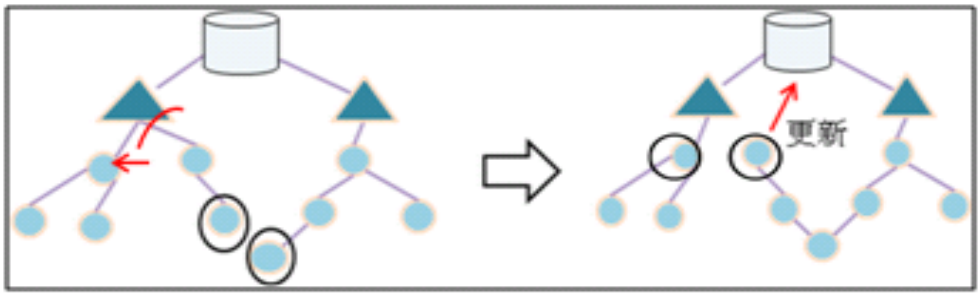


Figure 3-7 新轉接點隻更新示意圖

II. 由於節點之折損或移動所造成的變動

以 Figure 3-8 為例，當節點 A 因為電力消耗等原因而折損無法運作，或者因為移動的關係，導致節點 A 所屬的這對轉接點(圖中以

黑色圓圈所圈的兩個節點)已經不復存在，此時就必須要有一個機制來讓伺服器中儲存的相對應紀錄清除，我們讓成為轉接點的節點多了一個通知自己父節點的動作，讓轉接點的父節點可以透過監聽封包的方式(比方說聽取對方發出的訊標(beacon)封包)來判斷轉接點是否還存在，當 Figure 3-8 中節點 B 偵測到節點 A 已經不存在時，就需要負責發送一個更新的訊息通知伺服器。注意節點 B 需是節點 A 位於同一個網路的父節點，而不能是一個任意的相鄰節點。

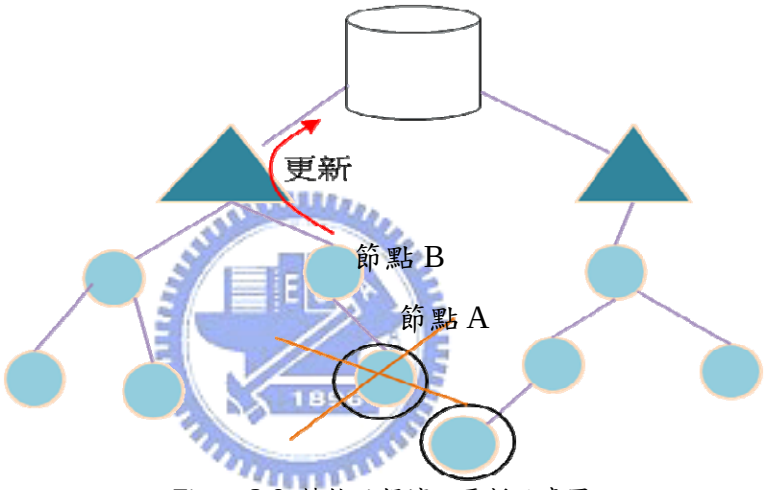


Figure 3-8 轉接點損壞之更新示意圖

3.3.2 存在多個轉接點時的選擇

前面的例子中，一個網路中都只有一個轉接點，然而實際情況通常是會同時存在許多的轉接點，當伺服器要開始進行網路切割調整動作時，要選擇哪一個轉接點作為 Token 的目的地，就會是一個問題。

原則上我們會希望被切割的節點在加入新網路之後，能夠有較淺的深度。因為較深的深度表示到達協調器的路徑較長，這樣會使得資料送達的時間較久，且使得資料遺失的機率大大的提高。因此伺服器在選擇轉接點時，原則上是希望能選擇一個轉接點使得節點在加入新網路後能以較淺的深度加

入新網路。

以 Figure 3-9 為例，當伺服器想要將 PAN1 切除若干節點至 PAN2 以平衡兩邊的負載時，發現有兩個轉接點可供選擇(節點 a 與節點 c)，節點 a 的對應轉接點節點 b 的深度為 1，而節點 c 的對應轉接點節點 d 深度則為 2，因此可以粗略的判斷，如果選擇節點 a 的話，在調整過後被切換到 PAN2 的節點應該能夠有較淺的深度，因此伺服器便發出一個以節點 a 為目的地的命令 Token 進行負載平衡的動作。

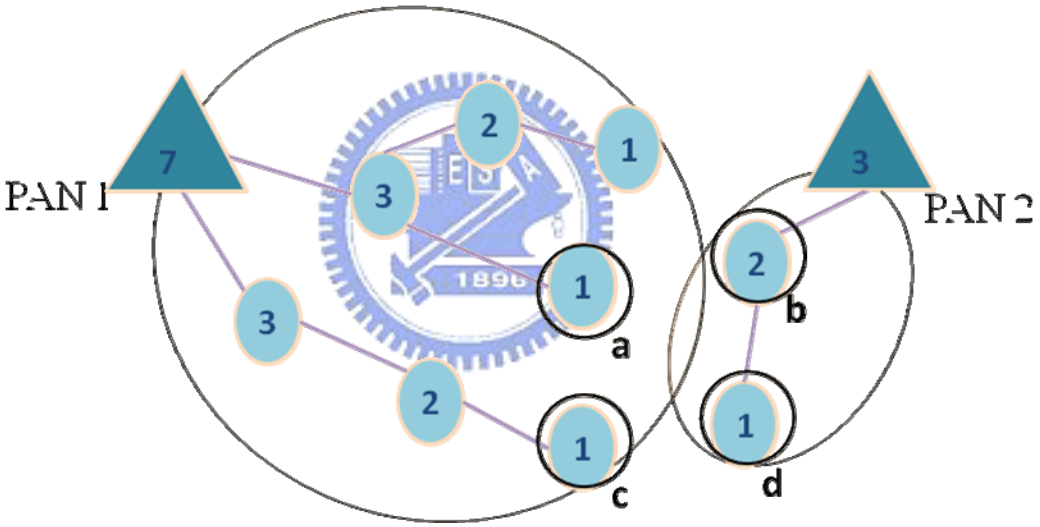


Figure 3-9 多個轉接點之選擇示意圖

讓我們回顧一下前面已述的新節點加入網路後的更新動作，當新節點在掃瞄後得知自己將為轉接點時，需要在上傳伺服器的更新訊息中，攜帶轉接點相關的資訊，這些資訊包括自己以及對應轉接點的識別碼以及網路識別碼(PAN ID)，除此之外，還需要附上深度這項資訊，以供之後伺服器在選擇轉接點的時候判斷之用。

3.3.3 加快平衡速度之伺服器快取機制

到目前為止我們所說明的都僅止於伺服器發出命令後如何的作網路切割調整的動作，以及後續的轉接點更新等動作，我們所設計的這套負載平衡機制可以說完全是以這個伺服器所發出的命令所啟動的。由於每次發出命令所得到的實際效果不見得如預期(因為只有當負載數量小於或等於命令中的數才會作切割)，因此往往需要不只一個命令，分別發向不同的轉接點才能真正完成負載的平衡，此時就有需要探討的地方，如果依照最單純的作法，應該是發出命令後，等待 ACK 封包收到，並且等待被切割的節點都加入新網路之後，再依照最新的負載情形再發出新的命令繼續下去，但很明顯的可以看出，這樣的作法將會需要非常久的時間才能達到負載的平衡。

以 Figure 3-10 為例，此時伺服器根據兩邊的負載情形判斷 PAN1 需要切除兩個節點至 PAN2，假設伺服器先選擇了轉接節點 a 作為 Token 的目的地，Token 中的負載數為 2，此時將發現只能切除一個節點，也就是節點 a(因為之前的節點的負載數都大於 2)，依照前面所提的作法，此時伺服器須等節點 a 回覆 ACK 封包，並且成功加入 PAN2，當伺服器發現兩邊的負載情形變成 6 比 4 時，才繼續發下一個 Token，此時目的地為節點 c，當節點 c 加入 PAN2 時遂完成整個負載平衡的動作。

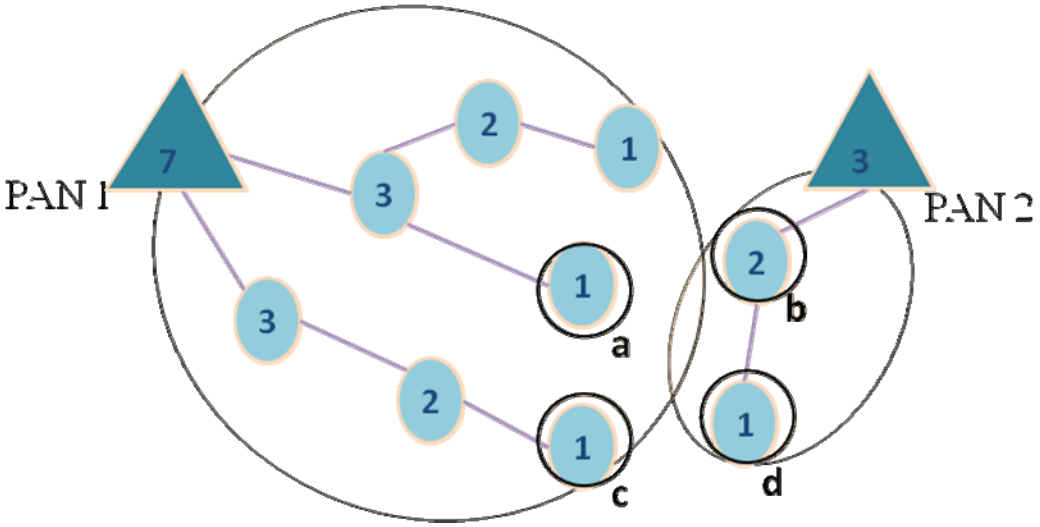


Figure 3-10 需要發出兩個以上 token 之示意圖

觀察這個例子，如果伺服器能夠在發出 Token 至節點 a 之後，收到 ACK 得知只切除了一個節點，就趕快發下一個 Token 至別的轉接點，也就是節點 b 的話，就可以節省非常多時間，因為等待被切除的節點加入新網路是一個非常花時間的事情。

因此我們提出一個在伺服器上簡單的快取機制，伺服器自己記錄目前各協調器的負載，這個快取的記錄將會定期跟據實際協調器的負載作更新，而伺服器在判斷負載是否平衡以及決定切除調整的數量，改為根據這個快取的記錄。觀察同樣一個例子，當伺服器根據快取發現兩個網路的負載為 7 比 3，決定切除兩個節點至 PAN2，當節點 a 回覆 ACK 告知實際只切除了一個節點時，伺服器的快取此時就調整為 6 比 4(而不是等待節點 a 加入 PAN2 才得知是 6 比 4)，此時再根據快取決定切除一個節點，因此第二個 Token 此時便送往了節點 c 了。

由於 ACK 的時間絕對比節點加入新網路的時間快上很多，因此這個機制將會非常有效的提高效率。

3.3.4 Multiple PAN 的快速平衡

接下來我們來看兩個以上的協調器的情況下，該如何做切割調整動作。其實多個協調器仍可以看作是兩個協調器(兩個 PAN)的延伸情形，伺服器可以針對兩兩協調器之間作平衡的動作，這些動作都跟前面所述的一樣，至於要切割的數量就根據所有協調器的負載的平均與各個協調器目前的負載的差距即可得到，因此問題就在於伺服器該以什麼樣的順序在多個協調器中逐一地挑出兩個來做平衡的動作。

最簡單也最沒效率的作法是隨機地任選兩個協調器出來做平衡，很明顯地這樣的作法並不是個聰明的作法。我們希望伺服器能夠在一個 pass，也就是對所有協調器檢查一次，就能夠決定出各個網路要切除多少節點(或者等著別的網路切除節點轉換至自己的網路下)。

我們在這邊只提出一個基本且有效率的作法，然而這僅是許多作法中的一種，因為這部分也不是本論文的重點，因此我們也不打算去探討是否有更好的作法。

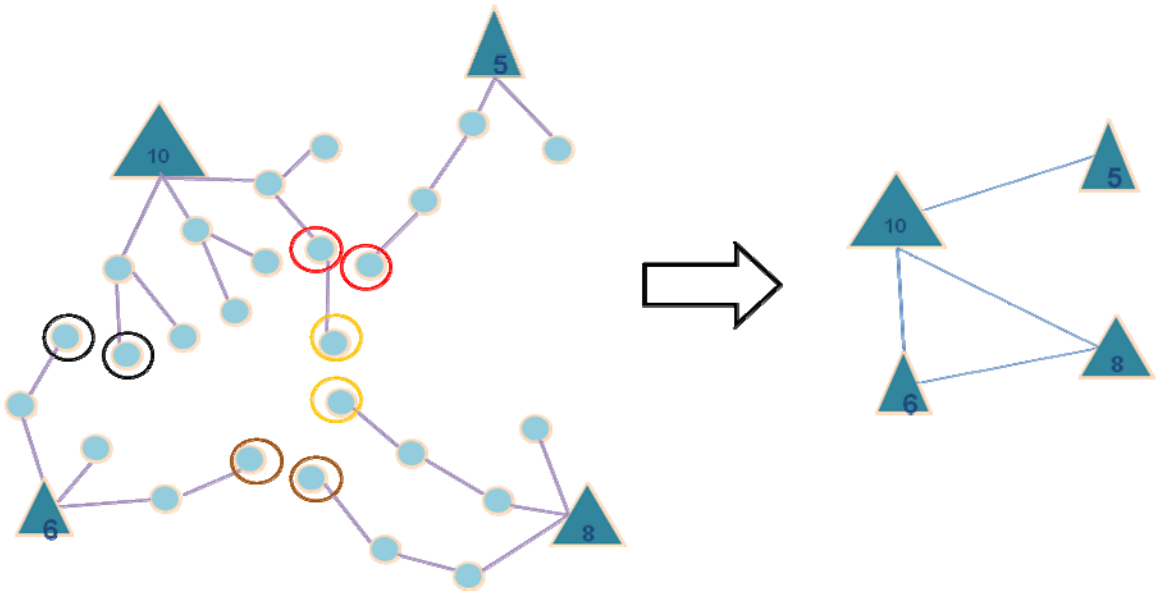


Figure 3-11 PAN Graph 示意圖

首先參考 Figure 3-11，左圖是實際整個無線感測網路的情形，總共有四個 PAN，三角形代表協調器，中間的數字表示其所屬的網路的負載量，另外圖中有一些節點被圈圈所圈起來的表示轉接點，注意到轉接點有成對的關係，當兩個網路存在有一對以上的轉接點時，就表示這兩個網路能夠使用我們的調整機制來平衡這兩個網路的負載。我們在此定義 PAN Graph，此 Graph 中的節點為無線感測網路中的協調器，而點與點之間有邊就代表實際網路中這兩個點相對應的 PAN 之間存在有轉接點，因此 Figure 3-11 中左邊的實際網路就可以化為右邊的 PAN Graph。

在我們的機制中，伺服器只知道各協調器的負載情形，以及轉接點的資訊，不過有了這兩個資訊，伺服器就能夠建構出一個 PAN Graph，接下來我們就開始說明伺服器如何藉由這個 PAN Graph 快速的決定各個 PAN 如何做切割調整。

首先藉由一些簡單的觀察推導出我們的演算法的基本精神，第一點是對於那些 PAN Graph 中只有一個邊連外的節點，在圖論中的術語就是所謂"out

degree 等於 1"的節點，這些節點在作切割調整動作時非常單純，一定是經由這唯一的邊和它的相鄰節點所代表的協調器作平衡，既然我們的目標是讓每個協調器最後的負載趨近於大家的負載平均值，因此我們可以在一開始就決定出這些 out degree 是 1 的節點和它的唯一的鄰居節點要如何作調整，Figure 3-12 為例，平均負載應該為 7，由於節點 b、節點 c 與節點 d 都只有一個邊，因此可以馬上確定的是節點 a 需要切除兩個單位的負載到節點 c，節點 a 需要切除一個單位的負載到節點 b，節點 d 需要切除三個單位的負載到節點 a。而這些動作是只需要針對每一個 out degree 等於 1 的節點掃一次就能夠做出的判斷，且是唯一的解不會再也其他的調整法。

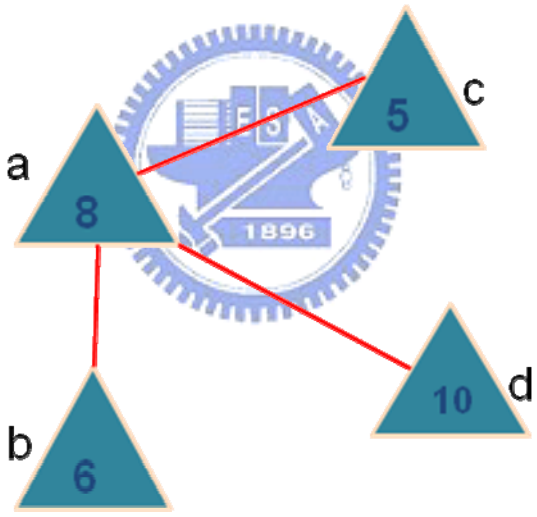


Figure 3-12 PAN Graph 之 Example 1

如果我們一開始就先把這些 out degree 是 1 的節點做好調整，調整完成後將他們自 PAN Graph 中移除，然後再繼續同樣的步驟，直到最後不再有這樣的節點時，接下來 PAN Graph 中剩下的都是具有兩個以上的邊的節點，這些節點怎麼和其他節點作平衡調整會有不止一種的調整法，我們直覺地認為應該先選取負載最大的節點開始，切除負載量到各個相鄰節點使其負載降至平均值，至於怎麼分配負載量到各個相鄰節點，可以完全把負載任意的切移到

隨便一個相鄰節點，或者依照各個相鄰節點的邊長度(即和各個相鄰節點所對應的 PAN 的轉接點深度)作等比例的分配，不過底下的例子 Figure 3-13 可以告訴我們事實上有些例子並不能任意的做切移，而需要根據 Graph 以及負載的分佈而有策略性的作調整。

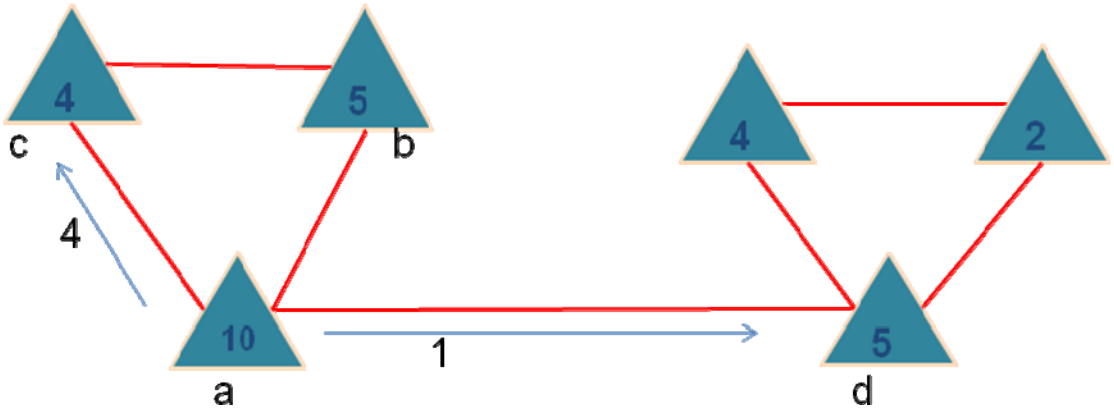


Figure 3-13 PAN Graph 之 Example 2

Figure 3-13 中理想的負載平均值是 5，節點 a 的負載最大因此從他開始調整，節點 a 總共需要移除 5 單位的負載，它總共有 3 個相鄰節點可供選擇，假設以圖中的分配方式，藍色箭頭與數字表示移除調整的方向，即節點 a 移除 4 單位的負載到節點 c，也就是伺服器將發送 Token 到節點 a 所代表的 PAN 中，Token 中的值為 4 並指定切換到節點 c 所代表的 PAN。可以很明顯的看出來當節點 a 以這樣的分配方式調整結束後，之後其他點的負載平衡調整如果不再透過節點 a 的話，將會無法達到平衡狀態(節點 b 與節點 c 負載變重，但除了透過節點 a，沒有管道可以把多的負載移到右邊的三個節點)。

因此我們的想法是，如果能夠讓所有的情形都能夠像 out degree 等於 1 的節點那樣的情況，就會非常的單純，於是我們想到一個資料型態能夠非常符合這樣的需求，那就是樹。如果我們對這個 PAN Graph 求出一個生成樹(spanning tree)，接著利用同樣的規則，先從只有一個邊的節點開始作平衡，

調整完後將這些節點移除，我們將發現永遠不會出現剩下的節點邊數都大於等於 2 的情況，這都是因為樹有一項特性：樹保證不存在迴圈。

底下我們展示的是演算法的 Pseudo-code：

Algorithm: Multiple PAN adjustment**(PAN Graph: G)**

```
1:  T ← Get-SpanningTree(G)
2:  Avg_Load ← average load of all coordinators
3:  while number of nodes in T != 1  do
4:      for all Ni ∈ out degree 1 nodes in T do
5:          adj_Ni ← adjacent node of Ni
6:          L ← load of Ni
7:          if L < Avg_Load  then
8:              Send(Token[Avg_Load-L, ID of SN in PAN_adj_N])to PAN_adj_N
9:              load of adj_Ni -= Avg_Load-L;
10:         else if L > Avg_Load  then
11:             Send(Token[L-Avg_Load, ID of SN in PAN_N])to PAN_N
12:             load of adj_Ni += L-Avg_Load;
13:         end if
14:         Remove Ni from T
15:     end for
16: end while
```

3.4 演算法 Pseudo-code

介紹完基本精神與機制之後，這裡要展示的是完整的演算法 Pseudo-code，我們根據功能性大致分為兩大類，分別是節點加入所引起的一連串更新程序，另一個則是真正負載平衡網路切割調整的相關動作。

Algorithm 1: Relay Nodes Join PAN

(A)Joining Node:

1: *msg* ← *Update-Msg(myLoad)*
2: **if** *re-join* **then**
3: *Add old-ID and old PANID to msg*
4: **end if**
5: **if** *switch node* t**hen**
6: **for all** *neighbors* N_i *in Neighbor Table belong different PAN* **do**
7: *Add (ID,PANID,depth) of* N_i *to msg*
8: **end for**
9: *Broadcast([SN-notify(myID,myPANID)])*
9: **end if**
10: *Send(parent,msg)*

(B) Relay Nodes: Received Message [SN-notify]

1: *Send(parent,SN-notify(“children is switch node”))*

(C) Relay Nodes: Received Message [Update-Msg]

1: *myLoad* ← *myLoad + Update-Msg.load*
2: *Send(parent,Update-Msg)*

(D)Server: Received Message [Update-Msg]

1: **if** *exist old ID information* **then**
2: *delete Switch-Node pairs record by old-ID*
3: **end if**
4: **if** *exist switch node information* **then**
5: add *Switch-Node pairs record*
6: **end if**
7: *update Load-Cache*

以上是加入網路後引發的一連串更新動作，(A)的部分是新節點的相關動作，主要就是三件事情需要作，第一個就是如果是因為移動或者切換網路造成需要新加入網路的話，則先把自己先前的舊資訊(識別碼、網路識別碼)包含於上傳的封包之中。第二個就是如果自己是轉接點的話，就把轉接點相關的資訊放入封包中，並且發一個廣播封包，通知鄰居節點自己以及它們將成為轉接節點，收到這個通知訊息的節點將會通知自己的父節點(B 部分)，讓父節點負責定期追蹤自己的存在與否。最後就是發送更新訊息封包，送往伺服器。

更新訊息沿著樹狀拓樸一路往樹根送，沿途的節點就跟據此來更新自己

的負載資訊，並且繼續往父節點轉送(C 部分)。

最後伺服器收到此訊息後，首先判斷是否攜帶有舊識別碼相關資訊，如果有就以此檢查轉接點紀錄表格中，是否有相匹配的記錄存在，如果有就把此記錄消除，完成轉接點記錄的註銷動作。

接著如果攜帶有轉接點相關資訊，則把這些資訊增加到伺服器的轉接點記錄表格之中。

最後和其他節點同樣的，更新自己的負載資訊，只不過伺服器要更新的是自己的協調器負載快取記錄。

Algorithm 2: Load Balancing Procedure

(A) Server: checking Balance (period)

```
1:  if not balance in Load-Cache then
2:      execute Algorithm in chapter 3.3.4
3:  end if
```

(B) Relay Nodes: Received Message [Token]

```
1:  if myLoad <= Token.num then
2:      Send(parent,[Token-Ack(myLoad)])
3:      Limit-Broadcast([Switch-PAN(target-PANID)])
4:      if not coordinator then
5:          re-join to target PAN as Algorithm 1
6:      end if
7:  else
8:      if ID != Token.dest then
9:          Forward(Token.dest, Token)
10:     end if
11: end if
```

(C) Relay Nodes: Received Message [Switch-PAN]

```
1:  re-join to target PAN as Algorithm 1
```

(D) Relay Nodes: Received Message [Token-Ack]

```
1:  update myLoad by Token-Ack
2:  Send(parent,[Token-Ack])
```

(E) Server: Received Message [Token-Ack]

```
1:  update Load-Cache by Token-Ack
2:  do checking-balance
```

(F) Server: Update Cache (period)

```
1:  for all coordinators do
2:      query load of coordinator and update cache
3:  end for
```

Algorithm2 則為網路切割調整的相關動作。基本上由伺服器定期的檢查自己的負載快取記錄，當發現負載不平衡的時候，便會發送 Token 來啟動這一連串的網路切割調整動作。

Token 以被伺服器選定的特定轉接點作為目的地，一路往下傳送，沿途中收到此命令的節點都會做一個判斷，那就是檢查自己的負載數與 Token 中的數值之大小，如果自己的負載數小於等於 Token 中的數值，就從自己開始進

行切割的動作，首先先回覆伺服器一個 ACK 封包，告知真正切除的負載量(即自己的負載量)，接著向自己底下的子樹廣播，告知大家開始切換到指定網路，接下來就遵循 Algorithm1 開始加入新網路的相關動作(B 部分與 C 部分)。

其他節點與伺服器在收到 ACK 封包後，都會根據封包中的數值更新自己的負載資訊(C 部分與 D 部分)，只不過伺服器要更新的是自己的負載快取記錄。伺服器在更新完後，會再一次的根據自己的快取記錄檢查是否負載平衡，如果不平衡就再做一次調整動作，也就是再執行 A 部分的動作一次。

第四章 Simulation

4.1 模擬環境與相關設定

本章節是關於利用模擬來比較我們的機制與其他負載平衡機制的優缺點。在第二章的 Relative Work 部分有提過我們大致將以往的作法分為兩大類，分別是分散式處理與集中式處理，因此我們各別選一個代表性的作法作為我們的比較對象，集中式的作法我們根據文獻[13]作為比較對象，當網路中所有的節點將資訊上傳至伺服器後，其提出一個有效的演算法能夠生成一個平衡的樹(balancing Tree)；而分散式的部分我們選用類似 AODV 的 Path selection 作法，參考的是 DLAR(文獻[9])作法，其中有些許的修改則已於 2.2.1 小節說明，這裡不多加贅述。

我們的模擬是在 NS2 上面進行的，這套免費的網路模擬器功能十分強大，也提供非常大的彈性供使用者發展自己的協定，由於 NS2 目前最新的版本(version 2.32)有提供 IEEE 802.15.4 的 MAC 與 PHY 層的模組，因此我們便可以直接選用這些模組作為我們的第一層與第二層協定。

Table 4-1 是整個模擬的環境與參數設定：

NS 2 版本	2.32
MAC 層協定	IEEE 802.15.4 MAC
PHY 層協定	IEEE 802.15.4 PHY
L3 routing 協定	ZigBee/AODV
ZigBee Cm	5
ZigBee Rm	5
ZigBee Lm	6
分布區域大小	小於 100m X 100m
訊號傳輸最大距離	10m
節點個數	10~120 (隨機分佈)

協調器(coordinator)個數	3
Traffic	70 bytes/s CBR

Table 4-1 模擬環境與參數設定

NS2 提供的一個模擬視覺化動畫工具名叫 NAM，能夠將模擬的情況利用圖形介面工具採用動畫的方式呈現，底下 Figure 4-1 為其執行中的一個截圖，圓圈代表網路節點，其中有三種顏色表示其屬於三個 PAN 中的哪一個 PAN，圈圈中間的數字代表節點的編號，也就是其 MAC 位址。節點 0、節點 1 與節點 2 可以發現較為特別，圈圈較大且為紅色圈圈，此三個節點正是網路中的三個協調器。

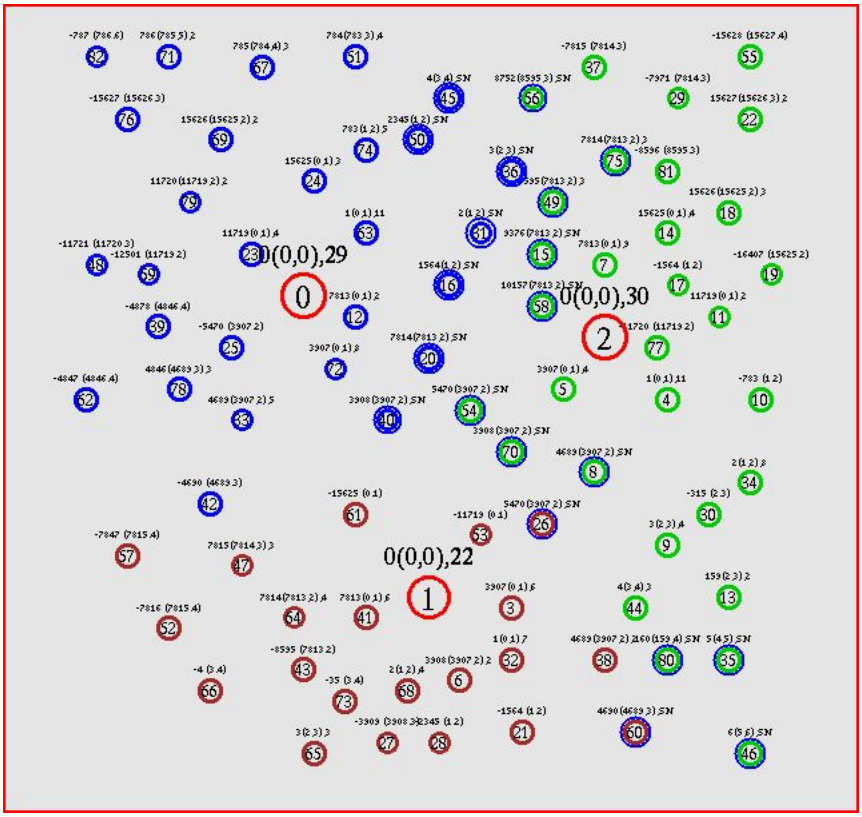


Figure 4-1 模擬實驗結果之截圖

在進入正式實驗與分析之前，先介紹一個用來評比負載的平衡是好是壞的一個方式，我們仿照其他相關研究[13]定義一個值，我們稱之為平衡係數(balance factor)，其定義如下：

$$Balance\ Factor\ (BF) = \frac{\left(\sum_{i=1}^{n} load_i\right)^2}{n * \sum_{i=1}^{n} load_i^2}\ ，n = number\ of\ \ coordinators\ at\ the\ network$$

其實這個 BF 值的意義就是利用標準差來評比負載平衡的程度，最平衡的情況下 BF 的值為一，越小表示越不平衡。下一小節將會針對幾個作為比較對象的演算法去分析觀察得到的 BF 情形。

4.2 模擬結果分析

首先我們把網路節點個數固定為 60、80 與 100，分別作三次模擬實驗，得到下面三張數據圖，其中 CAD 是我們的演算法的簡寫，Node-centric 就是取自[13]集中式的作法，L-AODV 則是如前所述修改自 DLAR(文獻[9])作法，其中有些許的修改則已於 2.2.1 小節說明，這裡不多加贅述。

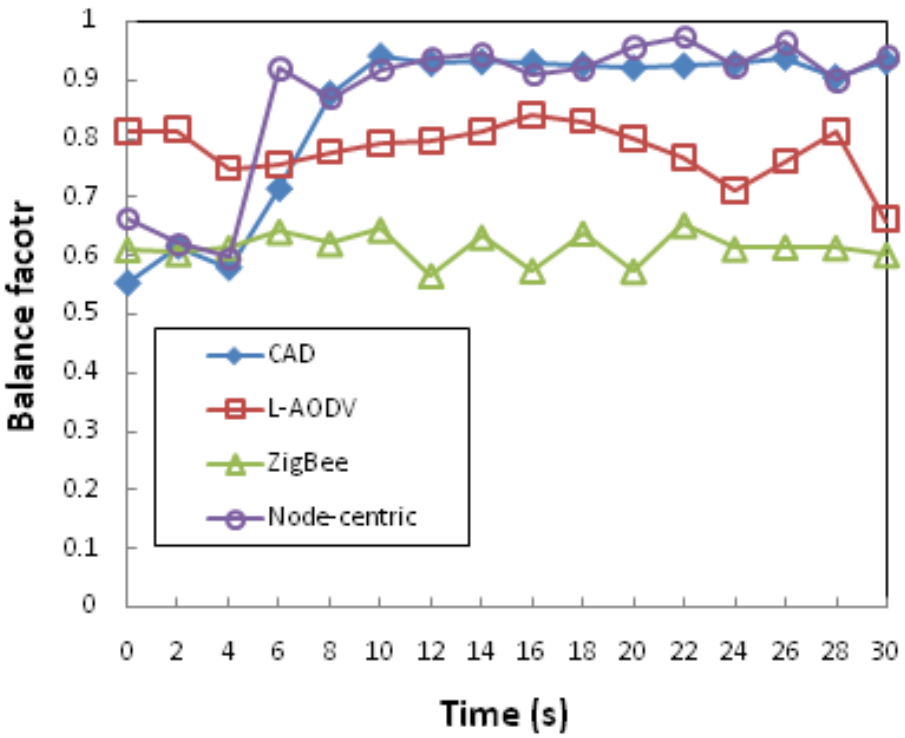


Figure 4-2 節點數 60 之 BF 變化圖

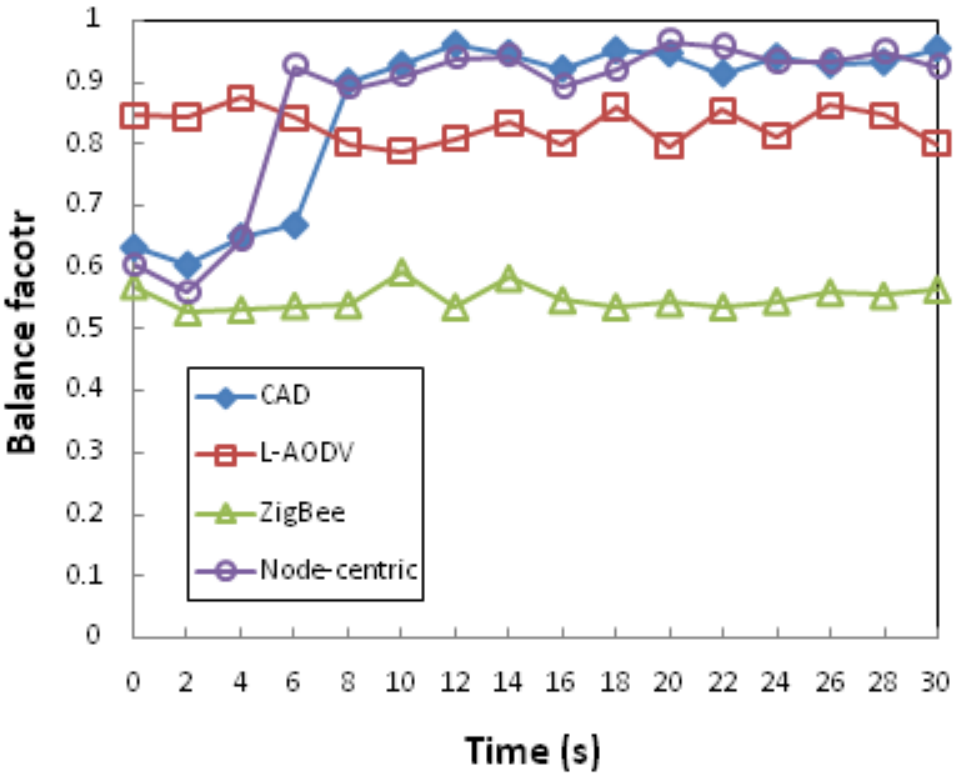


Figure 4-3 節點數 80 之 BF 變化圖

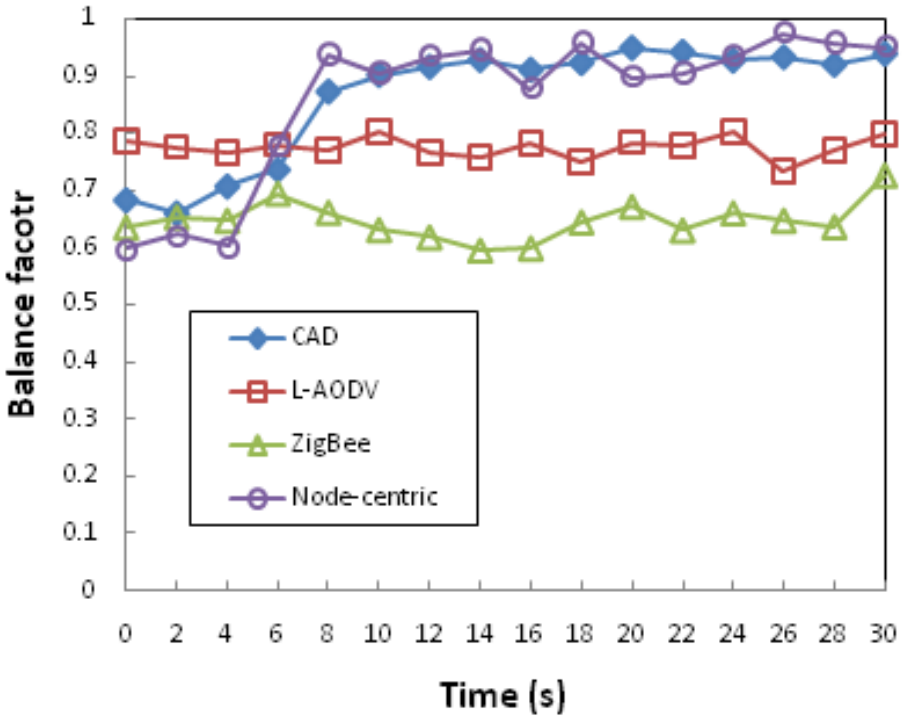


Figure 4-4 節點數 100 之 BF 變化圖

對於集中式的作法以及我們的作法，都需要一個中央的伺服器來處理，而不是像分散式處理的作法，每個節點進入網路就開始各自執行負載平衡的機制，我們讓

伺服器在模擬開始後的一段時間才開始動作，這段時間是為了讓整個網路先穩定下來，也就是所有新加入節點都順利加入網路之後，再由伺服器開始執行負載平衡的動作。之所以要這樣做有幾個原因，最主要是為了方便觀察出我們的演算法跟完全集中式的演算法到底有多快能夠完成負載平衡動作，屏除其他的影響因素。

觀察 Figure 4-2 到 Figure 4-4 這三張圖表，發現表現最好的是我們的作法跟集中式的作法，幾乎都可以達到九成的平衡效果。而 L-AODV 因為是分散式作法，無法達到跟集中式這麼好的效果，僅能達到大約八成的效果。而最差的就是完全沒有負載平衡動作的 ZigBee protocol，0.6 的平衡係數表現可以說是非常不平衡的。另外三張圖的 BF 變化的情形大致相同，似乎不管哪一種演算法，網路節點個數對於其 BF 的表現是不影響的，因此首先我們針對我們的作法 CAD，觀察節點數 60、80 以及 100 的 BF 變化情形如 Figure 4-5，發現節點個數的確不會影響。

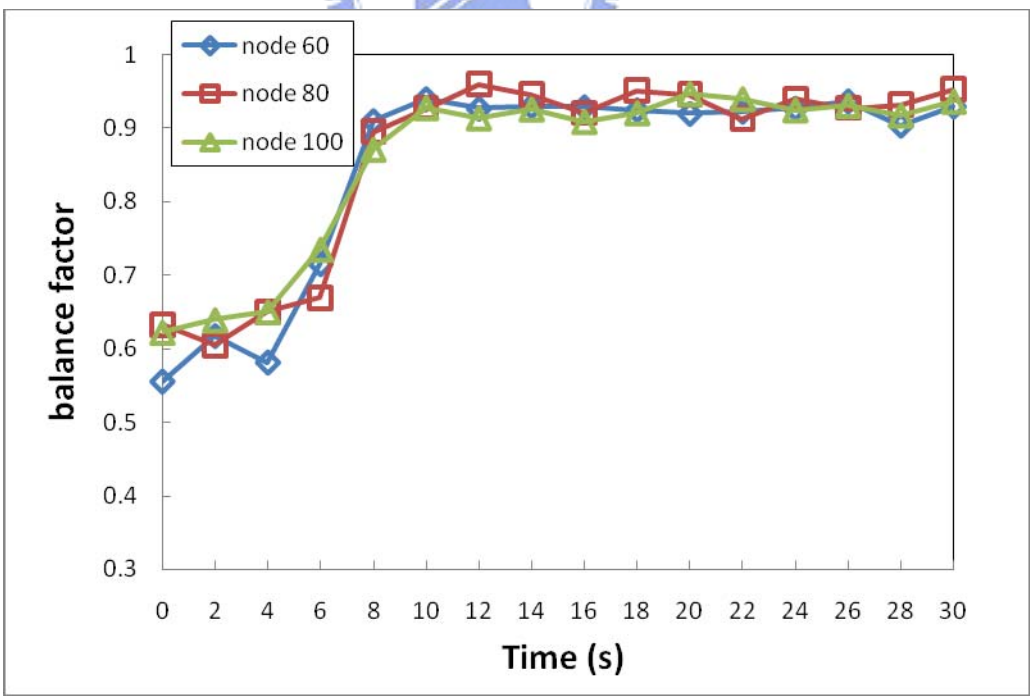


Figure 4-5 CAD 的 BF 變化情形圖

再來我們更改實驗方式，針對節點個數從 10 個到 100 個，觀察不同演算法最後穩定(stable)之後所得到的平均 balance factor 表現，得到 Figure 4-6，對於 CAD 跟

集中式作法，基本上還是一樣可以達到九成以上的穩定表現，而分散式的 L-AODV 作法雖然只能達到八成，但似乎也跟網路節點個數無關。至於僅運行 ZigBee 的情況下，發現當節點個數較少時，可能還有稍好(但也還不到七成)的表現，隨著節點數增多，負載越來越不平衡，當節點數超過 70 以上，就已經降至六成甚至六成以下的表現。

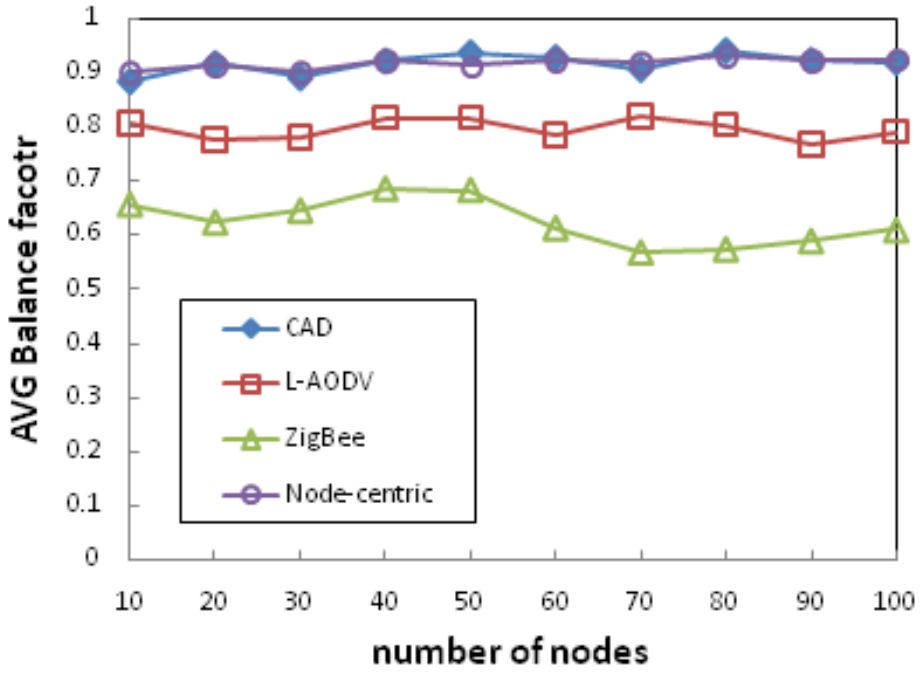


Figure 4-6 不同演算法對於節點個數的 BF 表現圖

既然得知我們的作法跟集中式的作法得到的效果差不多，接下來我們便想要觀察彼此之間的效率表現，我們針對不同演算法在達到 BF 效果九成以上時，所需要的訊息量作一個實驗，由於 L-AODV 頂多只能到達八成，因此無法一齊列入比較，但其實很明顯的可以知道分散式的作法需要的訊息量一定會比集中式的還要多。

底下 Figure 4-7 便為實驗的結果圖：

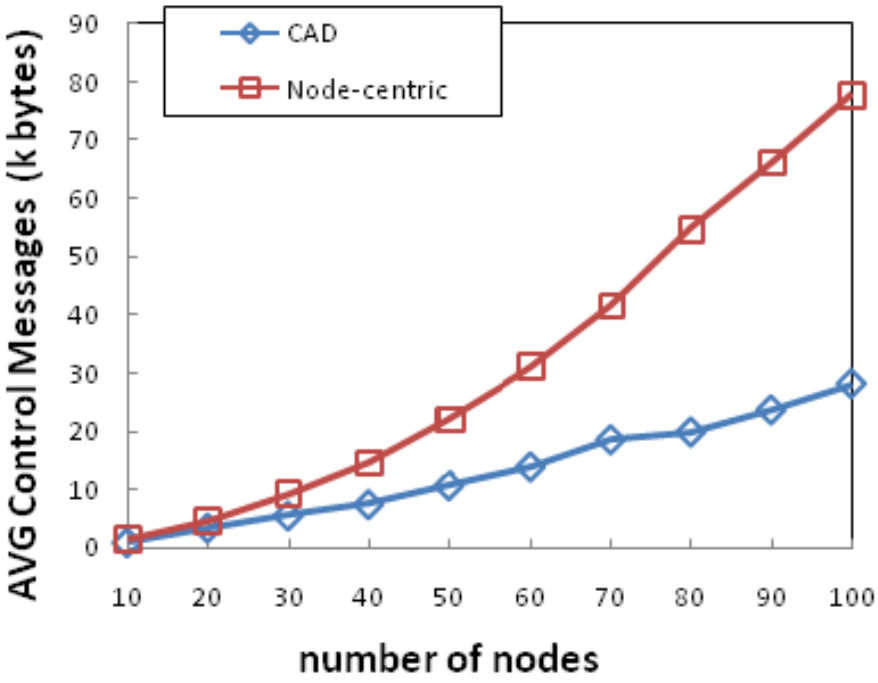


Figure 4-7 不同演算法所需的訊息量

可以看到我們的作法所需要的訊息量，隨著節點個數漸多，越顯得比集中式有效率，可見我們這個介於集中式與分散式之間的作法的確可以做到取長補短，讓負擔(overhead)大幅的減少。

Figure 4-8 則是觀察兩種作法從伺服器開始動作，到整個網路平衡係數到達 0.9 以上，這中間所花的時間。完全集中式的作法很單純，只要把資訊丟給伺服器，伺服器運算完再丟出來給網路各節點就完成的負載平衡的動作了，因此他所需要的時間比我們的作法還要短，不過當節點個數漸多，發現集中式的作法速度漸漸被我們的作法趕上，這主要是因為伺服器廣播出來的封包，隨著網路規模漸大，就需要花更多的時間才能讓所有節點都收到，尤其在無線的多步跳躍(multi-hop)環境下，網路規模漸大，要在其中作一個廣播的動作可想而知產生的干擾現象就更嚴重，後面的料吞吐量(Throughput)實驗也可以看出這個現象影響了集中式處理的表現。

從 Figure 4-7 與 Figure 4-8 兩張圖歸納出一個結論，就是我門的作法雖然需要

花的時間較集中式作法長，但是僅慢了約一秒多的時間，而且隨著節點個數越來越多，這個缺失會越來越不明顯，但在犧牲若干速度的情況下，我們的作法所需要的訊息量遠遠少於集中式作法所需要的訊息量，這在非常講究電量的無線感測網路應用之中，更能凸顯我們作法的好處。

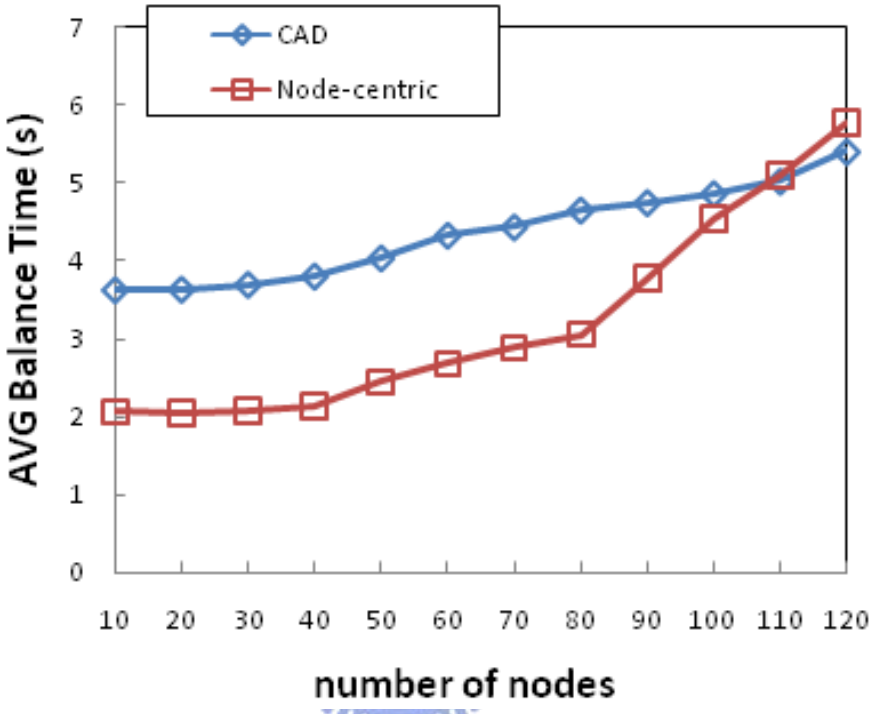


Figure 4-8 到達平衡所需要的時間圖

接下來我們比較各個演算法在資料吞吐量(Throughput)方面的表現情形，如 Figure 4-9 所示，三個機制中 L-AODV 表現最差，這是因為執行 L-AODV 演算法實網路的負載較不平衡，負載過於集中某些協調器的結果，導致網路壅塞嚴重，封包較容易遺失掉落。至於集中式的作法以及我們的作法，一開始表現差不多，但是可以發現當節點數超過 80 以上，集中式作法的表現開始往下掉，隨著節點數多而漸差，這就是因為前面所提到的，伺服器所廣播的繞送拓墣封包會因為網路規模大而容易遺失，模擬下來會發現當節點數很多的時候，很多節點因為這個廣播封包的遺失而沒有收到此封包，導致這些節點不知道如何繞送封包，我們稱這些節點為 un-connected 節點，Figure 4-10 是觀察不同演算法在不同網路節點個數出現的

un-connected 節點個數，集中式處理作法在節點數超過 80 以上之後就會出現很多 un-connected 節點，因此影響了集中式處理在資料吞吐量方面的表現。

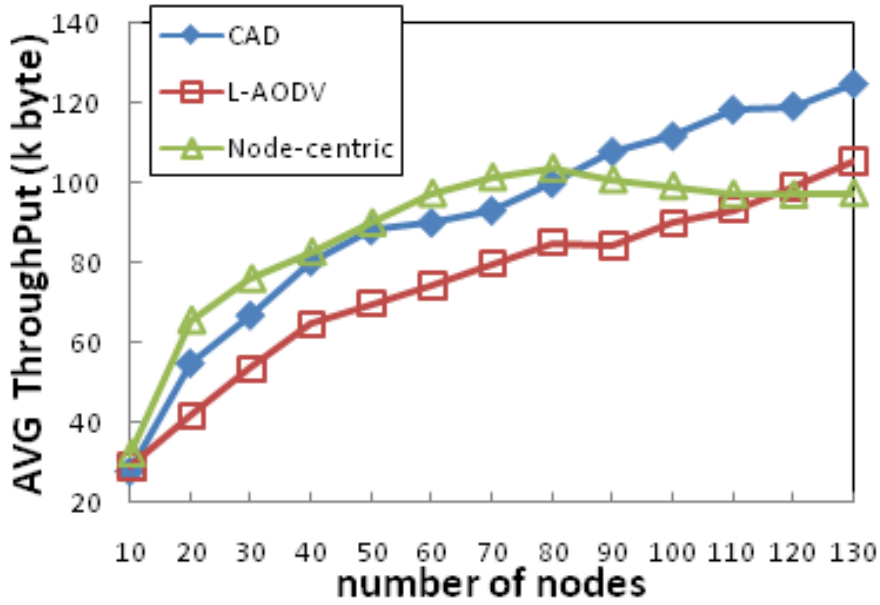


Figure 4-9 不同演算法之 Throughput 表現

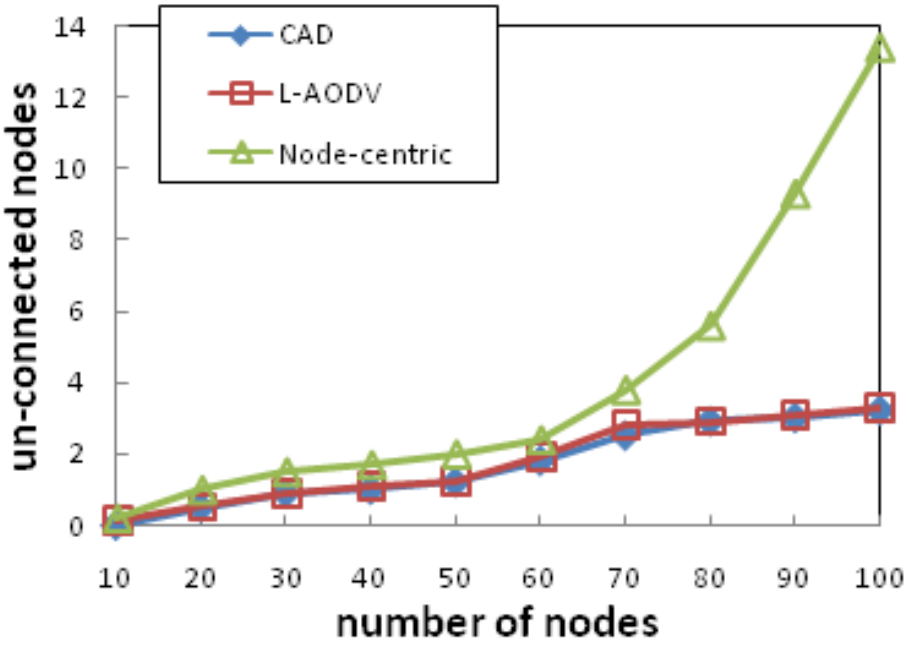


Figure 4-10 各演算法出現無法繞送節點之數據圖

最後我們來觀察三種作法的多步跳躍傳輸延遲(multi-hop end-to-end delay)表現，如 Figure 4-11 所示，可以看到較為平衡的集中式與我們的作法，表現出來的延遲都較短，雖然集中式處理在節點數多的時候會有資料吞吐量下跌的情形，但是其

它能夠繞送的節點仍然較為平衡，因此對於多步跳躍傳輸延遲仍較為快速。最不平衡的 L-AODV 所展現的多步跳躍傳輸延遲就較為延遲。這個實驗的結果顯示在許多要求即時性的無線感測網路應用環境中，我們的作法也能較為符合需求。

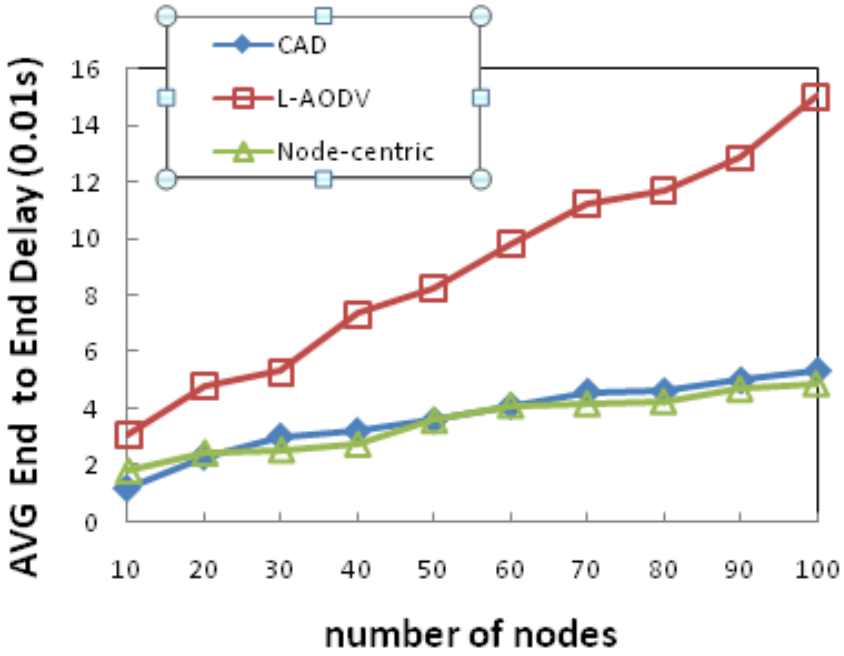


Figure 4-11 不同演算法之 End-to-end delay 表現圖

第五章 結論與未來工作

5.1 結論

在本篇論文中，我們針對無線感測網路因為規模漸大而有多個閘道器(協調器)，因而產生的閘道器之間的負載平衡問題，提出一套有效率的解決機制。本篇論文中探討的這套機制是以 ZigBee 的樹狀繞送機制為基礎，希望能夠把負載較重的 ZigBee 協調器切除部分節點，令其轉換到其他負載較輕的協調器，也就是轉換到另一個 PAN。

我們提出的這套機制採取一種由中央伺服器輔助，分散式調整的作法，因為這種介於集中式與分散式之間的作法，使得這套機制比兩者更加的有效率。首先在網路建立子樹負載資訊維持樹(Sub-tree based load information maintenance tree)使得每一節點能夠知道自己底下子樹的總負載量，同樣的伺服器也知道各個協調器的總負載量；再者我們讓每個節點在加入網路時判斷自己是否屬於轉接點(Switch node)，如果是的話便發送更新訊息讓伺服器能夠掌握這些轉接點的相關資訊；接著採用調控者(伺服器)輔助的分散式作法來切割調整各個 PAN，也就是讓伺服器決定需要切除的量，而究竟要從何處開始做切除轉換 PAN 的動作，則由底下的各個網路節點來自行決定。

另外本論文也針對轉接點因為轉換 PAN 的動作，或者是因為移動或是損毀，導致伺服器上所記錄的轉接點資訊過時不正確的現象，提出一些解決的辦法。另外針對如何加速伺服器與負載平衡的速度以及伺服器在多重協調器(超過三個以上)時，也提出一些相關的作法來解決。

第四章的實驗模擬結果，可以驗證我們介於集中式與分散式作法的這套機制，在負載平衡的程度上表現很好，以及在資料吞吐量還有多步跳躍傳輸延遲(multi-hop end-to-end delay)方面都有很好的效果，而所需要的訊息量卻遠比其他作法還要來的少。

5.2 未來工作

未來，我們希望能夠把切割調整的最小單位降至終端裝置(End device)，而不是目前的繞送裝置(Relay node)，由於目前應用大部分是使用者身上攜帶著終端裝置漫遊，終端裝置的能力較為受限，傳輸距離也較短，因此勢必需要一些調整才能將我們的這套機制套用在裝端裝置上。

另外，經過實驗模擬也發現，我們的機制在負載平衡的調整動作中，大部分的時間是花在那些被切除的節點要加入另一個指定的 PAN 時，所花的時間，這段時間包括從新掃瞄，並且加入網路取得新識別碼等步驟所花的時間，希望未來能夠有一套改進的機制來讓這段時間能夠縮短，以提升本論文的機制的效能。

最後，我們希望未來能夠把更多的參數納入切割調整時的考量，例如像跳躍步數(hop count)等，一般為了要將負載平衡，就需要轉移到其它路徑來傳送資料，因此往往為了顧及負載平衡，就會使跳躍步數變多，然後跳躍步數多表示到達終點的距離較遠，封包也會容易遺失，傳送所需的時間也較長。未來希望能夠顧及兩者取折衷，並且找出一個最佳化的加權方法。

Reference

[1] I. F. Akyildiz , W. Su , Y. Sankarasubramaniam , E. Cayirci, Wireless sensor networks: a survey, Computer Networks: The International Journal of Computer and Telecommunications Networking, v.38 n.4, p.393-422, 15 March 2002.

[2] Akyildiz, I.F., Weilian Su, Sankarasubramaniam, Y., Cayirci, E., "A survey on sensor networks," *Communications Magazine, IEEE* , vol.40, no.8, pp. 102-114, Aug 2002.

[3] ZigBee Standards Organization, “ZigBee Document 053474r06, Version 1.0,” De-cember 14, 2004.

[4] IEEE, "Draft Amendment to IEEE Standard for Information Technology - Telecommunications and Information Exchange between Systems - PART 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Sow-Rate Wireless Personal Area Networks (LR-WPANs): Amendment to add alternate PHY," *IEEE Std P802.15.4a/D6, Dec 2006* , vol., no., pp.-, 2006.

[5] David B. Johnson, David A. Maltz, “Dynamic Source Routing in Ad Hoc Wireless Networks”, Mobile Computing, edited by Tomasz Imielinski and Hank Korth, Chapter 5, pages 153-181, Kluwer Academic Publishers, 1996.

[6] Cedell Adam Alexander, Russell Eugene Gardo, Brahmanand Kumar Gorti, Olen Lee Stokes, “Method and system within a computer network for maintaining source-route information at a router bypassed by shortcut communication”, United States Patent US6452921 B1, September 17, 2002.

[7] Gutierrez, Jose A., Pereira, Luis R., “Source routing protocol for an ad-hoc communication network”, EUROPEAN PATENT APPLICATION EP1480387A1, April 26, 2004.

[8] C. Perkins, et al., “Ad Hoc On-demand Distance Vector (AODV) Routing,” IETF RFC 3561, July 2003.

[9] Lee, S.-J.; Gerla, M., "Dynamic load-aware routing in ad hoc networks," *Communications, 2001. ICC 2001. IEEE International Conference on* , vol.10, no., pp.3206-3210 vol.10, 2001.

[10] Hossam Hassanein , Audrey Zhou, “Routing with load balancing in wireless Ad hoc networks”, Proceedings of the 4th ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems, p.89-96, July 2001, Rome, Italy.

[11] C Intanagonwiwat and R Govindan and D Estrin, “Directed diffusion: A scalable and robust communication paradigm for sensor networks” in IEEE/ACM Mobicom, 2000, pp. 56-67.

[12] Shah, R.C. and Rabaey, J.M., “Energy aware routing for low energy ad hoc sensor networks” in Wireless Communications and Networking Conference, 2002. WCNC2002. 2002 IEEE, Volume 1, 17-21 March

2002 Page(s):350 - 355 vol.1.

[13] Hui Dai and Richard Han, “A node-centric load balancing algorithm for wireless sensor networks”in IEEE Global Telecommunications Conference, 1-5 Dec, 2003.

[14] Hock Guan Goh, Moh Lim Sim, Hong Tat Ewe, “Energy Efficient Routing for Wireless Sensor Networks with Grid Topology”, EUC 2006, LNCS 4096, 2006, pp. 834- 843.

[15] Tzung-Shi Chen and Hua-Wen Tsai and Chih-ping Chu, “Gathering-load-balanced tree protocol for wireless sensor networks” in proceedings of the IEEE international conference on sensor networks, ubiquitous, and trustworthy computing, 2006.