南　華　大　學

資訊管理學系

碩士論文

無線廣播環境下的動態排程演算法

A Dynamically Schedule Algorithm for Wireless Broadcast

研 究 生：林士颺

指導教授：吳光閔　博士

中　華　民　國　九　十　五　年　六　月　二　十　三　日

南　華　大　學

資訊管理研究所

碩士學位論文

無線廣播環境下的動態排程演算法

研究生：（林士閎）

經考試合格特此證明

口試委員：陳峰

吳光閎

蔡德謙

指導教授：吳光閎

系主任(所長)：資訊管理學系吳光閎系主任

口試日期：　中華民國　95　年　6　月　6　日

誌　　謝

研究所兩年的期間過的非常快，記得印象中才剛入學沒多久，轉眼已完成論文，即將完成碩士學位的攻讀。就讀研究所時期，非常感謝元安、建磐、美綸、閔浩、乾訓等學長姐的教導，讓我在踏入此陌生領域時不會徬徨無助，可以很快的適應。非常感謝同學小童、嘉明、小吟、淑玲、昭勳、坤錨及全班同學在這兩年時間內的支持與指教，讓我可以順利完成研究所的課業。

最特別感謝指導教授吳光閔主任與蔡德謙老師的細心指導，雖然讓您花費許多時間在學生身上，但透過您的教導讓我成長許多，將來希望能更有成就以不負您諄諄教誨之苦心。這兩年內有太多太多的事要感謝您，誠心的向您說聲：老師謝謝您。

無線廣播環境下的動態排程演算法

學生：林士颺　　指導教授：吳光閔

南　華　大　學　資訊管理學系碩士班

摘　　要

在無線網路的拉式架構之下，客戶端向伺服器端請求播送資料，為了解決冷、熱門資料不同點播率及客戶端請求之後等待時間過久的問題，伺服器端同時考慮資料項的被點播率與客戶端請求資料的等待時間之後，再決定播送的資料項。在真實的環境之下，資料項的被點播率與客戶端請求的等待時間會隨著時間而不斷改變。若同時考慮上述兩個因素，伺服器端在每次廣播前必需重算全部資料項的權重值，再決定廣播的資料項，這是非常耗資源且低效率的方法。為了解決這樣的問題，我們提出動態排程演算法(DYSA)來解決上述的問題。DYSA 使用資料項的權重值來維護最大堆積樹，以供快速廣播資料項。此外，我們利用索引的技巧來加速堆積樹裡資料項的存取。在最後的模擬實驗結果顯示，在動態環境的要求下，DYSA 比一般演算法有更好的執行效能。

關鍵字：無線廣播、排程、堆積、索引

A Dynamically Schedule Algorithm for Wireless Broadcast

Student：Shih-Yang Lin Advisors：Dr. Guang-Ming Wu

Department of Information Management
The M.B.A. Program
Nan-Hua University

ABSTRACT

In the pull-based wireless network, clients request data items from a server. In order to solve cold and hot data, and reduce clients' waiting timer, server considers hit rate of data items and waiting time of requests for broadcasting. The hit rate of data items and the waiting time of requests always changed along with time elapsed. Therefore, server has to calculate overall weight value of data items and decides maximum weight value of data items before broadcasting. In order to increase performance of server, we propose a dynamically schedule algorithm (DYSA) for wireless broadcast. The DYSA uses weight value of data items and maintains them in a maximum heap tree for quickly broadcasting. Furthermore, we present a indexing for accessing a data item's information faster. The experiment results show that our method has more effectively than a common algorithm in the dynamic wireless environment.

Keyword：Wireless Broadcast, Schedule, Heap, Indexing

目　　錄

表 目 錄

圖 目 錄

第一章 序論

隨著社會和科技的發展，我們享受到進步帶來的便利與舒適。在20世紀末期，因為科技的發展使得電腦普及化，而電腦普遍之下帶動附產業—網路的蓬勃發展。網際網路隨著寬頻時代的來臨，已慢慢的融入我們的日常生活之中，現代的人經常透過網路來找尋資料、聯絡情感、消費等等，它早已不再只是單純的學術或軍事用途。

早期的網路速度慢、頻寬窄，在使用上有相當多的限制。而現在進步的科技，讓我們可以很方便與快速的使用高速無線網路(Wireless Network)。有越來越多的科技產品支援無線網路技術，人們在使用網際網路時將不再只局限於有線網路，更可以透過無線技術來快速使用所需的服務。

第一節 研究背景與動機

現代的人們幾乎已離不開網路的世界，科技的發展讓我們可以方便的使用網路，接下來的世代將是無線網路為主的時代。無線網路可以應用在相當多的產業上，相對於在不同的領域上會有不同

的需求與問題。隨著社會的進步，有越來越多的產品使用無線網路的技術，而人們也將會慢慢的由有線網路轉換到以無線網路為主的環境。因此，無線網路仍有很大的發展空間，相對於有線網路，它有著更多的便利但也相對的有較大的限制。

在無線網路的發展上，無線技術仍受限於頻寬、傳播媒介等因素。在有限的頻寬下如何將資料完整的傳送將是很值得探討研究的。有很多的專家學者分別討論不同的技術並應用在無線網路領域上，使得無線網路的頻寬可以在有限的條件下容納更多的資料。除了廣播資料項的討論外，我們還面臨一個重要的考慮因素：資料項的被點播率。在真實的環境之下，資料項會隨著時間而有不同的被點播率。因此，如何以最短的時間決定播送的資料亦成為重要研究。本研究將提出高效能的演算法，快速決定播送的資料項，以降低伺服器的負載。

第二節　研究主題

在無線網路的大環境下可以區分成許多不同的種類，分別用在特定的無線網路環境裡，而不同的無線環境其運作與特性也不相同。本研究將專注於在伺服器端的廣播排程領域，提出高效能排程

演算法來降低伺服器端的負載。在過去許多的廣播排程文獻中，有很多的研究者分別提出不同的方法來有效的將資料置放於適當的廣播位置上[4][7][9][12][15][16][17][18][19][20]。但是多數的研究報告都未考慮到一個現實環境的因素[4][7][12][15][18][19][20]：資料項的被點播率應會隨時改變，而不應是一個固定的被點播率。

因此，我們將以解決動態頻率問題為主要研究方向，提出一個動態排程演算法(Dynamically Schedule Algorithm, DYSA)能有效解決動態環境的問題，及避免客戶端請求冷門資料項而等候過久的饑餓現象。

第三節　研究限制

本研究並不是適用於所有的無線網路，因此將其限制說明如下：

1. 著重於伺服器端的無線廣播排程演算法。
2. 排程演算法適用於單頻道廣播系統，未考慮多頻道資料配置問題。

3. 適用於拉式廣播系統(Pull System)。

4. 固定長度的資料項。

5. 有限的無線廣播頻寬。

第四節　研究架構

本研究將以有結構化的三個不同階段循序進行。第一階段為起始期，我們首先瞭解無線網路的背景環境及其運作方式，接著再進行相關文獻的討論。在對無線環境有一定程度的瞭解之後，擬定我們的研究主題與方向。本研究將朝此研究方向探討。完成了第一階段後，接著進入第二階段的發展期。在發展時期引用第一階段的知識及研究方向，針對研究內容提出問題討論，提出本研究專注及解決的問題。接著再探討各種可行方案。可行方案透過評估之後，選定一最佳方案並結束第二階段。最後，是研究的驗證階段。在此階段裡將以程式的呈現來模擬無線廣播排程的實驗，核定所提出的方案是否能達成較佳的效果。透過模擬實驗的數據來進行分析比較之後，再提出本研究的結論。

透過結構化的研究步驟，可以較完整的呈現我們的研究過程。本研究主要是基於無線網路的環境下，針對拉式(Pull)廣播系

統的動態環境提出合理且理想的解決方案。此方案將可以適用於真實的環境下，因為它符合真實動態環境的要求。此外，我們的方法更可以有效解決資料項因動態機率產生等待時間不公平等問題。

第五節　簡介

本研究裡首先介紹無線網路下的環境架構，說明無線廣播與資料排程的運作方式。接著，在Pull廣播架構之下，我們發現目前所提出的演算法常有下列兩個缺點：1.大多數無線廣播排程演算法[4][7][12][15][18][19][20]均考慮固定的客戶端點播率，這在真實的環境之下是不合理的假設。2.多數排程演算法只嘗試一味降低客戶端的等待時間，造成客戶端請求冷門資料而產生等候過久的饑餓現象。

在真實的動態環境之下資料項(Data Item)的被點播率應隨著進入伺服器(Server)的請求(Request)而改變。為了符合動態環境的要求，Server 應能夠隨著每個 Request 到達時，重新計算資料項的加權值以做為廣播的依據。如果 Server 在每次 Request 到達時皆需重算所有 Data Item 的加權值，這將會是相當耗資源的。另一個動態環境的問題是饑餓現象的產生。饑餓現象的問題發生於客戶端

(Client)所產生的Request若是較冷門的資料，那麼由於Data Item的被點播率較低導致Client必需等待很久才能接收到此冷門的Data Item。

為了解決上述的兩個因素，我們提出動態排程演算法(Dynamically Schedule Algorithm, DYSA)來解決上述的問題。DYSA運作在Pull架構之下，而且可以有效提高Server的運作效能。DYSA主要的運作理念是：利用堆積樹(Heap Tree)的特性，Heap Tree 總是將最大(小)值維護在樹根(Root)，我們利用此性質來快速的決定Server即將廣播的Data Item，而不需透過繁雜的計算程序。在DYSA裡，每個 Data Item 都會依客戶端送進的 Request 而被計算出一個總重值(Total Weight Value)，此Total Weight Value用以決定此Data Item在Heap Tree裡的相對應位置，DYSA只需每次在Request進入或廣播(Broadcast)之後維護此 Heap Tree 和相關陣列(Array)的正確性。運用這樣的方式將比全部重新計算資料項目的 Total Weight Value，再決定最大值做為廣播的方式快上許多。

此外，由於在 Heap Tree 內資料搜尋的效率不佳，為了能快速有效的增加 Heap Tree 的搜尋，動態排程演算法導入 Index 的技術。每個Data Item都會利用 Index 的方式記錄本身目前所在Heap

Tree 裡的位置。當 Request 進入而需更改相關資料時，將可以快速的直接指向 Heap Tree 的相對位置而進行更改，故能有效的節省在 Heap Tree 裡資料搜尋的時間。

DYSA 亦能避免冷門資料等待過久的問題，所以在計算 Total Weight Value 時同時加入等待時間的考量因素，我們希望等越久的 Data Item 能有較高的優先權排入廣播頻道裡。

在模擬實驗結果顯示我們提出的演算法(DYSA)有較佳的執行效能。動態排程演算法可以快速的處理進入 Server 的 Request，有效提高 Server 的運作效能及減少排隊在 Server 端佇列裡的 Request 數量。此外，模擬結果也顯示在 100 個資料項且 Arrival Rate 低於 25/sec 時，動態排程演算法幾乎可以完全運算且讓 Server 裡的 Queue 經常保持在空閒狀態，這現象表示它有著相當好的效能。

本論文有系統的介紹剩餘的部份：第二章將討論在無線網路環境下的相關文獻探討；第三章則是描述透過文獻探討之後，我們所發現的問題；接著，在第四章詳細介紹說明動態排程演算法的概念、程序及運作；我們將模擬實驗的結果介紹於第五章，此章節將詳細說明模擬環境、參數及數據分析結果；最後第六章是本研究的結論。

第二章　文獻探討

隨著科技的進步，人們使用無線網路的技術已經越來越普遍了。美國電機電子工程學會(IEEE)在1990年11月召開802.11委員會，而在1997年6月公佈 IEEE802.11 的標準規範，使得 IEEE802.11 協定成為無線網路的另一個代名詞，而 802.11 協定裡亦規範了許多不同的需求與規格。

所謂的無線網路(Wireless Local Area Network, WLAN)，主要是由用戶端(Client)的需求透過設備的轉換，而以電波的方式在空氣媒介中傳遞，到達伺服器應用端(Server)。伺服器端亦以同樣的方式將資料傳送給用戶端。簡單的來說，就是客戶端透過無線設備，例如：無線網路卡(Wireless Card)與伺服器端的無線設備，例如：無線基地台(Base Station, BS)或存取點(Access Point, AP)等，來進行連線的動作，進而可以交換資料、連接網際網路等。

無線網路在另一個普遍的應用為手機電話的普及。在現在的社會大眾中，手機似乎已成為人手一機，更多的是一人使用多機的情況。手機通訊實際上也是一種無線網路的應用。把手機視為一移動式的用戶端，在待機或通訊時則是透過無線電波與基地台進行連線的資料傳遞；相同的若手機位於基地台電波範圍之外亦無法進行通訊的運作。

無線網路已深深的應用在許多日常生活之中，接下來本研究將討論相關的文獻探討，並加以進一步說明本研究的背景環境。無線網路的主要架構可以分為兩個部份：有基礎架構的無線網路(Infrastructure WLAN)和無基礎架構的無線網路(Noninfrastructure WLAN)[6]。接著，再分別區分為PUSH、PULL與Table Driven、On Demand等不同的環境。無線網路環境架構如圖 1 所示。

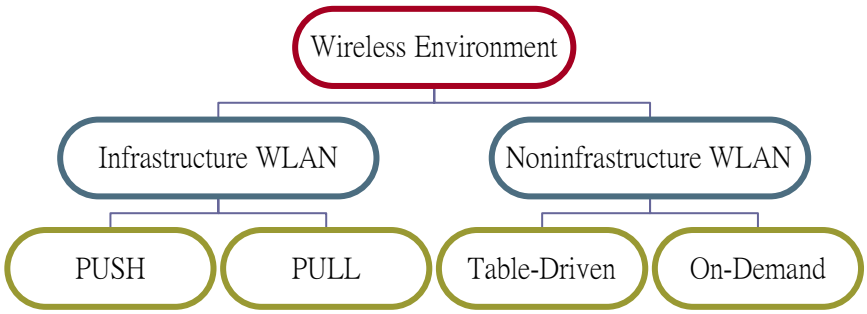

圖 1　無線網路架構分類圖

第一節　背景環境

隨著時代的進步，網路的使用早已融入我們的日常生活之中，網路的使用已成為人們生活的必需品。而藉著科技的發展，網路技術也由有線網路慢慢的轉成以無線網路為主的架構，且有越來越多的人們和越來越多的產品都在使用無線網路，使得無線網路已

經非常普遍了。

在Infrastructure WLAN裡，是以基地台(Base-Station, BS)為主，BS和BS之間可以透過有線網路來連線，但是在BS廣播覆蓋範圍下的移動性主機(Mobile Host, MH)可以透過無線的方式來與Server進行連線。在這個架構下的廣播排程方式，主要又可分為推式(Push)和拉式(Pull)[12]。

以推式為基礎的運作方式，是 Server 端將資料推送給 Client 端，也就是由 Server 端不斷廣播所有的資料，而 Client 端則一直監聽從 Server 端播送出的所有資料，如果側聽到自己需要的資料，則再將資料截取下來，這樣的方式不需主動送出任何詳細的要求(Request)。而以拉式為基礎的廣播方式，是由 Client 送出對所需資料項的請求廣播，經由無線網路的傳送而到達 Server 端，然後 Server 嘗試以較佳的排程演算法將此Client端所請求的資料項排入廣播排程來滿足 Client 端的請求。

不論Push或Pull的廣播排程方式都有其優缺點，所以Navrati Saxena等人在 2004 年的 Proceedings of the 18th International Parallel and Distributed Processing Symposium 上提出一種結合兩種方式優點的混合式排程演算法[6]。混合式排程演算法允許

Base Station(BS)和Mobile Host(MH)可以同時使用Infrastructure WLAN和Noninfrastructure WLAN。MH可以直接和兩個Hop範圍內的另一個MH直接連線傳遞資料，而在兩個Hop範圍之外的連線則可透過BS的輔助。結合兩種協定的運作將可以更有效率的使用網路資源。

另一方面，在Noninfrastructure WLAN裡，主要的架構我們稱之為Ad hoc WLAN，它與Infrastructure WLAN剛好相反，在Ad hoc的環境下，沒有Base-Station的存在。它是以Mobile Host彼此的廣播覆蓋面做為通訊的方法，也就是MH透過另一個或多個MH的跳躍(Hop)傳遞，將資料傳送至目的地，每個MH除了做為一般的主機之外，也必需扮演路由器(Route)的角色。因此，像是BS的集中式管理或定期標準化服務的支援是不存在這種環境之下的。在Ad hoc的系統裡，MH透過相鄰的MH將資料傳送到目的地，因此，在這樣的情況之下傳送路徑的選擇(Routing)則變成相當的重要，因為較佳的Routing選擇可以讓我們能更有效率的使用Ad hoc網路。Ad hoc的Routing方法，主要可分為兩種方式：Table-Driven Routing和On-Demand Routing[1][2]。

所謂的Table-Driven方式，是指在Ad hoc環境下每個節點(node)本身都存有Routing的資訊，並且週期性的去更新在這個路由

表(Routing Table)內的資訊，這是屬於主動式更新的方法。它固定每隔一段時間就必需去維護整個網路的拓撲(topology)，因此較適合使用於靜態的Ad hoc環境。Table-Driven方式的缺點為需要不斷地更新及維護(Maintain)使用的路由表，此表格將會佔據不少空間。另一個缺點為需要週期性的更新路由表並廣播此資訊，這導致網路頻寬的使用率降低。較常見的Table-Driven方式有DSDV,FSR,WRP和OLSR等等[3]。

Ad hoc 環境下的另一種 Route 方式為 On-Demand 模式。在On-Demand 方式中，只有在有需要時，Mobile Host 才發送尋找路徑的封包，這是屬於被動式的 Routing 方式，它亦可以降低在Table-Driven 方式下的網路 overhead 問題，適合使用在動態的 Ad hoc 環境下。而 On-Demand 方式通常可以區分為兩個階段，第一個階段為找尋路徑(Route Discovery)，第二個階段為資料傳送(Data Transfer)。由於 On-Demand 方式沒有週期性的更新與廣播，只有在有需要時才去找尋路徑，因此在網路頻寬的使用上較 Table-Driven 有效率，但在 Route Discovery 上卻比 Table-Driven 方式多花費一些時間。較常見的 On-Demand 演算法有 AODV,DSR,ABR 和 LMR 等等。

在 Ad hoc 環境下的無線網路，MH 是透過彼此之間的連線來達

成將資料送達目的地的方式，因此，每個 MH 除了是 Host 的角色外，必須擔任其它 MH 資料的轉送者，也就是 Route 的角色。在這樣的情況之下，選擇一個個較穩定、效率高的傳輸路徑(Routing)即為 Ad Hoc 網路中相當重要的一環。不論是 Table-Driven 或 On-Demand 方式都一直有專家學者在不斷的研究發表更新、效能更好的方法來使 Routing 最佳化。

DSR的繞徑協定是由Johnson和Maltz提出的[8]，它只可以在雙向的通訊環境下運作。因此Tomonori Asano等人提出另一個Routing協定，稱為LBSR[5]。LBSR不論是在單向還是雙向的通訊環境，都比原本的DSR協定有較好的效能，作者以更完整的演算法和附加更多的旗號(Flag)讓Route形成一個Loop。Routing時不用從來源的路徑往回，而是找尋另一條可靠的路徑回到起點，這可有效解決DSR只能在雙向連結的環境下運作。同時，實驗結果也顯示LBSR在許多不同的實驗中皆能獲得比DSR方法較高的效能。

除了Routing的協定外，也有一些學者在討論透過已建立好的路徑來傳送封包而產生的封包遺失問題。為了有更高的封包傳遞率，Chiu-Kuo和His-Shu等人提出一個AODV-HPDF(AODV routing protocol with high packet delivery fraction)的機制[9]，來提

高封包的傳遞率。AODV是由Perkins和Royer兩位學者所提出的[10]，它適用在On-Demand的網路型態上。AODV在Route Discovery時，主要是由Source端向Destination端發出建立連線請求，此時Source端會發出一個RREQ的封包給所有網路裡的節點，節點在第一次接收此封包之後再將此封包轉送出去。直到Destination端接收到RREQ封包時，會依紀錄在RREQ封包裡的Path資訊選擇一條合適的Route，並回送RREP封包給Source端。當Source端接收到RREP封包時，Route Discovery階段便已算完成。但是當Routing建立完成後，網路裡的節點亦有可能會隨著時間移動，而使的Route的連結失敗。因此，此篇的作者提出一個AODV-HPDF的運作程序來解決這樣的問題。最後的實驗結果亦顯示，運用AODV-HPDF的方法比AODV[10]、DSR[8]和TORA等方法有更好的封包傳送率。

Shiow-Fen和Chyi-Ren等人提出一種以分群為基礎的Routing Discovery協定，而且可運用在有多個Source端的環境中[14]。在MANET(Mobile Ad hoc Network)的環境下，一般的討論都是以單一個Source端與單一個Destination端之間的Route建立，但是在同一個On-Demand網路裡，可能同時會有多個Source node與多個Destination node，且可能經由演算法所建立的Route會有重疊的部

份，因此造成在Routing時網路資源的浪費。為了解決這個問題，Shiow-Fen 提出一個新的演算協定，稱為 Cluster-based Multi-source multicast routing protocol(CBMRP)。這個演算法首先將網路內所有的節點分群，每個群組有個Clusterhead，負責管理與監控 2 個hop內的成員。接著，以Cluster-based的方式，將網路內所有的Source與Destination的節點全部串接起來，如此一來，節點在傳送資料時便不用重新尋找路徑，有效的降低整個網路的overhead。在Mobile Ad hoc環境裡，CBMRP亦考慮到在Route的維護與Route的重建(Reconstruction)。以整個網路的效能來看，CBMRP在多樣的測試條件中，效能大多比原本的ODMRP來的高。

在MANET的環境下，來源端透過中繼節點(Relay node)傳遞資料給目的地端，通常在傳送資料之前，Source必需和Destination建立一條路徑用來傳送資料，此階段稱為Route Discovery 階段。除了Route Discovery之外，Route在建立之後的維護(Maintenance)也是相當的重要的。不過，如果透過重建delivery tree的方式，其成本將會是很大的。在這樣的問題下，Fumiaki Sato和Tadanori Mizuno等人提出一個新的局部修復Delivery Tree的方法[11]。這個方法應用在ODMRP上，並且加入一個”Supporting Group”的技術來達到局

部Route的重建，而不需重建整個路徑。如此一來，將可以有效減少許多在網路上的控制封包，進而提升網路上的資料傳送率。對照於舊有的ODMRP方法，Fumiaki Sato所提出的方法不論是在資料傳送率或是降低控制封包的數量上，都有更佳的效能。

第二節　相關文獻

在 Wireless 的架構之下可以分為 Infrastructure WLAN 和 Noninfrastructure WLAN 兩大部份，在上一節的部份已簡略的介紹在 Noninfrastructure WLAN 架構下相關文獻的討論。接著在本節的部份將說明在 Infrastructure WLAN 下的相關文獻。

Infrastructure WLAN的架構下又可分為兩個不同的運作模式，分別是推式系統(Push System)和拉式系統(Pull System)。在Push架構之下，Server透過演算法排程來決定廣播的資料項。Haw-Yun 等人提出一個建構於Push的廣播模式之下並可適用於多頻道的廣播排程方案(PinWheel Scheduling scheme, PWS)[13]。PWS用在即時的(Real Time)廣播資料和非即時的(Non Real Time)廣播資料的排程。透過模擬實驗結果可以清楚的瞭解運用PWS方案能有效

的降低延遲(delay)和劇跳(jitter)，且在非即時資料的封包佇列上運用PWS方案能有較短的佇列長度。

即時資料封包的傳遞必需在每一個封包所設定的Deadline之前完成。為了在無線廣播系統的封包傳遞中同時滿足即時資料和非即時資料的平均回應時間，Hsung-Pin等人便提出了有效解決此問題的排程演算法[17]。在此排程演算法裡可以同時處理即時資料與非即時資料的排程，讓每一即時資料能在Deadline之前完成，且讓非即時資料的平均回應時間降低。[17]是在廣播頻道裡先劃分固定的(Fixed)時槽(Time slot)數目，專用於廣播即時資料的時槽與一般即時和非即時資料共用的時槽。考慮到即時資料的Deadline，如果此資料項排入某一專屬的時槽時未能滿足此資料項的Deadline，則將此資料項向前調整到一般的時槽中廣播，以符合Deadline的要求。透過這樣的運作方式將能夠滿足即時資料的有效性，並降低客戶端非即時資料的平均等待時間。

多頻道的廣播排程可以容許更多的資料，即在更短的時間內廣播較多的資料，也就是說在一個廣播週期內將資料置於不同的頻道中同時廣播，以節省Client的等待時間。在 1997 年 6 月，美國IEEE公佈 802.11 無線網路的通訊標準協定規範，此後 802.11 便成為無

線網路的另一個代名詞。由於802.11的通訊協定的效率不能滿足一般的需求，Juki等人便提出了一個可以適用於802.11高速無線網路下的高效能廣播排程協定，稱為Out-of-Band signaling, OBS[16]。OBS將廣播頻道分為信號頻道(Signaling Channel)以供競爭頻道所使用；而資料頻道(Data Channel)供傳遞資料所使用。其中OBS為了能更有效的節省頻寬的使用，特別將資料和回訊(ACK)封包結合在一起，如此一來可以更減少在等待閒置時槽(Idle Timeslot)的時間。在模擬實驗的結果顯示隨著客戶端不斷增加，透過OBS的運作亦能有效的處理客戶端的資料請求。在Data Rate的模擬實驗中，隨著Data Rate的增加，使用OBS協定的方式亦可達到比使用IEEE802.11協定更好的效能。

在Push架構下，Server重覆廣播資料給Client時可能導致浪費許多資源，為考慮資料之間的關聯性以降低平均回應時間(average response time)，Etsuko等人因此提出Correlation-Based Scheduling(CBS)[20]的排程策略。CBS將廣播資料中較相關的資料集中一起廣播，以有效減少客戶端平均回應時間。例如資料項的關聯：A→C→D→F，在廣播排程時便將此四個項目依序排入廣播週期裡，接著再排另外的資料項。此外，為了能更加速廣播的運作，在

此方案裡亦加入不同於傳統快取(Cache)的方式。模擬實驗結果顯示，運用CBS策略可以有效減少平均回應時間。

另一方面在Infrastructure WLAN的架構下的Pull運作模式中Server針對單一Client所發出對資料項的請求(Request)，將請求的資料項依所用的排程演算法排入廣播頻道中。因為一般排程演算法只針對客戶端的平均回應時間做考量，Weiwei等人提出了適用於On-Demand架構下的排程演算法，稱為Largest-Delay-Cost-First, LDFC[15]。LDFC除了考量客戶端的平均回應時間外，亦考慮平均成本(Average Cost)裡包含的存取時間成本(Access Time Cost)、調協時間成本(Tuning Time Cost)、請求交換失敗成本(cost of handling failure Request)等三個項目。LDFC主要是以資料項的Delay Cost做為廣播排程的依據，以最大Delay Cost的請求有最高的優先權排入廣播頻道裡，運用這樣的方式來降低平均成本(Average Cost)。模擬實驗結果顯示，LDFC與其它的排程演算法相比較，LDFC花費較少的成本，而且即使在Push-Based廣播架構下仍然能有較高的可靠度。

在許多之前的研究裡，假設每一個Mobile Client在請求資料時只需請求一個資料項，而實際上Mobile Client有可能發出的請求

裡包含多個資料項，且可能同時有多Mobile Clinet發出請求。Ye-In等人使用Query-Set-Based的技巧結合查詢擴張法(Query Expansion Method, QEM)和挖掘關聯法則(Mining association rules)提出一個高效能的關聯排程演算法[7]。透過Mining Association Rules的技巧可以全面找出多數Client端頻繁發出請求的資料項集合(Data Item Sets)，再將之排入廣播頻道裡。透過這樣的運作方式，將可以快速的回應Client所發出的請求。

Navrati等人提出了適用於不同環境下的動態混合式排程演算法(Dynamic Hybrid Scheduling Algorithm)[11]。它結合Push和Pull的運作，將較為熱門的資料(較多Client請求的資料)以Push的方式廣播，而較不熱門的資料(較少Client請求的資料)以Pull的方式排入廣播頻道裡。動態混合式排程演算法假設在已知道的資料項存取率上(Data Access Probability)運作，且在Push和Pull的切換點上使用動態的方式來依需求調整，動態調整的方式可以將等待時間(Waiting Time)最小化。透過模擬實驗顯示，動態混合排程演算法的存取時間(Access Time)皆小於使用其它演算法。

在無線廣播架構下的Pull架構是很值得討論的議題。在這樣的架構下許多研究均單獨討論個別的或是全部的網路效能。個別的

網路效能指的是某一Mobile端或是Client端本身的執行效能，而全部的網路效能指的是整個網路下的傳輸效能。因此，我們需要一種能夠平衡個別與全部網路效能的排程演算法，且能衡量資料集合的大小、用戶端的總數、和網路的頻寬等因素。Demet等人提出了R*W的排程演算法來解決上述的問題[4]。R*W排程演算法主要考慮兩個不同的因素，分別是R(Request)和W(Wait)。R表示用戶端的請求，W表示在一段廣播週期內某資料項尚未被廣播前的等待時間。利用計算R*W之後得到總重值，演算法選擇最大值之資料項做為廣播。使用R*W演算法可以平衡資料項的請求數與等待時間，讓兩者可以取得一個較佳的平衡點。

Yiqiong等人提出不同於R*W的排程演算法，亦能夠平衡個別與全部的網路效能，稱為LTSF (Longest Total Stretch First)[19]。LTSF允許運作在不同的環境之下，且允許資料項不是固定的長度。在LTSF裡，Server維護每一個資料項的佇列(Queue)，透過計算，每個資料項都有個延伸值(*Stretch*)，廣播時選擇延伸值最大者的資料項做為廣播。透過這樣的方式將可以快速的選擇廣播的資料項，減少浪費的運算時間。

另一個能平衡個別與全部網路效能的排程演算法，同時能有

效解決資料項的饑餓現象是由Xiao等人提出Preemptive Request Stretch(PRS)演算法。它分析了FCFS(First Come First Served), MRF(Most Requests First), LWF (Longest Wait First), RxW, STOBS(Summary Tables On-Demand Broadcast Scheduler), SSTF (Shortest Service Time First), SRST (Shortest Remaining Service Time), PLWF (Preemptive Longest Wait First), LTSF (Longest Total Stretch First)等方式之後提出PRS排程演算法[18]。PRS計算所有資料項的R*S的總重值做為廣播排程的依據(R表示等候廣播資料項的數量，S表示較早之前的請求而未廣播的資料項)，較大總重值的資料項則優先廣播。實驗結果顯示雖然PRS在個別或全部的網路效能均無法達到最佳的效能，但是它可以在平衡兩者之間有較佳的執行效能。

第三章　問題描述

在前一章節的部份，我們已清楚的介紹相關的文獻著作。接著本章節將會詳細說明在文獻討論之中，所發現常被忽略的問題。本章節有系統的分別介紹符號的定義、無線廣播的運作、動態環境、資料廣播的問題。

第一節　符號定義與說明

為了能更清楚的說明問題與本研究的內容呈現，表 1 定義本研究所使用的符號。

表 1　符號定義表

符號	說明
d_i	第 i 個資料項，最基本的資料項單位(data item)
D	Server 端裡資料項的集合，$D=\{d_1,d_2,d_3,...,d_n\}$
r_k	某一 Client 端產生的廣播請求
$d(r_k)$	$d(r_k)$ 代表 r_k 所請求的資料項 $d(r_k)=d_i$ ：第 k 個 request 請求播送 d_i

$d^{-1}(d_i)$	d_i 第一次被請求的 request $d^{-1}(d_i)=r_k$: 第一次請求 d_i 的 r_k
R	Request 的集合，$R=\{r_1,r_2,r_3,...,r_m\}$
T_i	T_i 為時間點單位
$T(d^{-1}(d_i))$	d_i 第一次被請求的時間
NH_i	d_i 的被請求次數(點播權重值)
W_i	d_i 的總重值(Total Weight Value)
I_i	d_i 的索引值，存放 d_i 在 Heap Tree 內的位置
BS_i	d_i 的時間權重值，$BS_i=1/(T(d^{-1}(d_i))-T_0)$

在表 1 中，廣播系統最基本的廣播單位為資料項，d_i 表示指的是第 i 項資料項，而 D 則是資料項的集合，標記成 $D=\{d_1,d_2,d_3,...,d_n\}$。由 Client 所發出的請求(Request)標記成 r_k (k 表示 Request 的號碼)，而 $d(r_k)$ 表示此 Request 所要求廣播的資料項。R 表示為此請求資料的集合，記錄為 $R=\{r_1,r_2,r_3,...,r_m\}$。

在系統的運作上需要說明時間單位，因此將時間符號記錄成 T_i，系統起始時間則記錄為為 T_0，而 $T(d^{-1}(d_i))$ 表示 d_i 第一次被 r_k 請求的時間($d(r_k)=d_i$)。所有 Client 所發出的的 Request 會先進入

Server 的佇列(Queue)裡等待廣播，起始值預設為空集合。在演算法運作時，需要陣列資料結構的輔助，分別說明如下：NH_i 表示 d_i 的被點播次數；I_i 表示 d_i 在 Heap Tree 的索引位置；W_i 為經過計算後 d_i 的總重值。 W_i 的方程式如下：

$$W_i = \alpha * NH_i + (1-\alpha) * BS_i \text{, where} \begin{cases} 0 < \alpha < 1 \\ BS_i = 1/(T(d^{-1}(d_i)) - T_0) \end{cases} \quad (1)$$

為了實現等候較久的資料項有較高的優先權廣播，必需設立起始時間 T_0 來給進入系統的 Request 運算時所使用。演算法在運算時，必需計算 d_i 的時間權重值。d_i 的時間權重值標記為 BS_i，表示此為 d_i 的時間權重值($BS_i = 1/(T(d^{-1}(d_i)) - T_0)$)。

第二節　無線廣播的運作

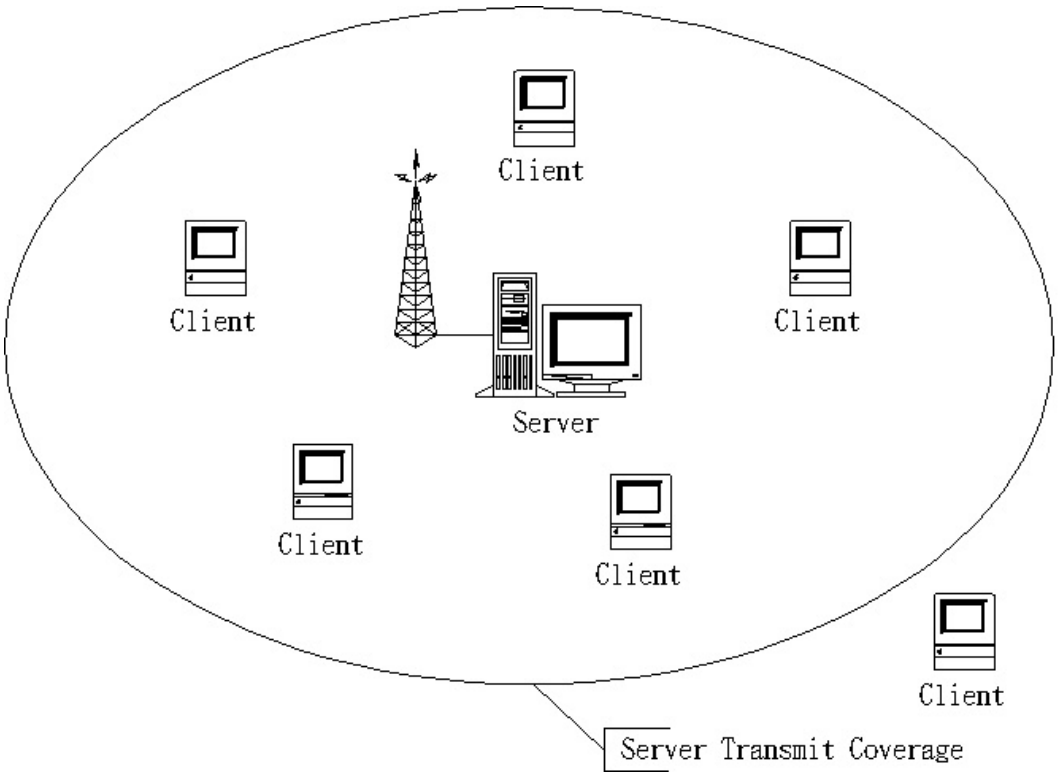

圖 2　無線網路架構圖

在本文前述的部份，已清楚描述說明無線網路的整個大環境。在有基礎架構下的無線網路系統運作以基地台為主，用戶端透過基地台涵蓋的無線電波範圍與基地台進行通訊的運作，而用戶端與用戶端之間一般是無法直接通訊，是必需透過基地台的轉送資料來進行交換資料的動作(如圖 2 所示)。

Push 系統簡單來說是廣播伺服器本身將所有資料排入廣播週期裡，利用週期性的廣播不斷重覆播送全部的資料；當用戶端側聽

到自己所需要的資料項時，再將其收取下來。圖 3 顯示 Push 廣播系統的架構；在伺服器裡只需透過排程演算法將資料項排入廣播週期裡。

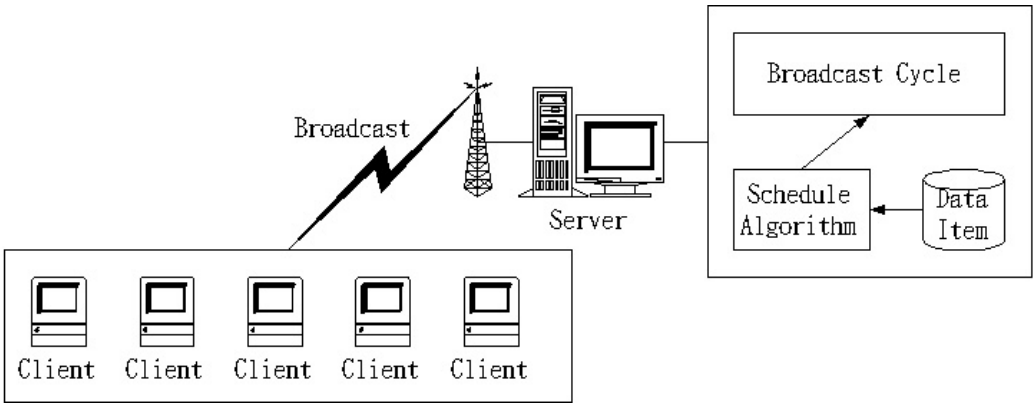

圖 3　Push 系統運作圖

另一方面的 Pull 系統則是依據由用戶端產生的請求，廣播伺服器再依此被點播的資料項排入廣播頻道裡。隨著演算法技巧不同，排入的先後順序也不盡相同。Pull 廣播方式的優點為較符合用戶端的需求，且較熱門的資料項可以較早發送，以節省用戶端的等待時間。圖 4 顯示 Pull 廣播系統架構；用戶端透過無線電波與伺服器端通訊，傳送請求給伺服器；伺服器在收到來自於用戶端的請求後，透過排程演算法與資料項之間的計算，然後再將資料項排入廣播頻道裡。

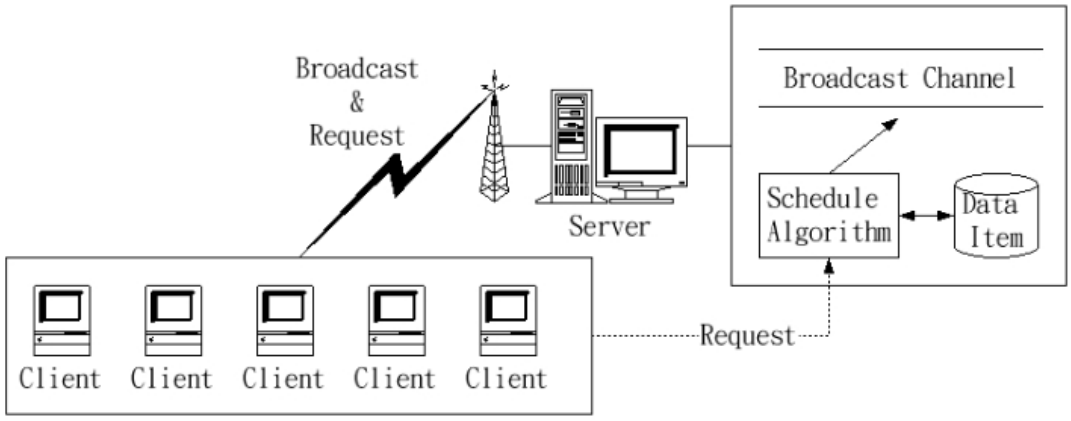

圖 4　Pull 系統運作圖

第三節　動態環境

在我們現實的生活之中，有許多環境的因素是呈動態的變化，在無線廣播環境裡亦相同。為了能讓所提出的方案更適用於真實環境中，我們應該努力將環境模擬與現實環境相似。因此，不論是 Push 或是 Pull 廣播系統，其伺服器端內的資料項被點播率應該與真實環境中相似，以符合動態環境的要求。

本研究所謂的動態環境，指的是：在 Pull 系統裡，每個在伺服器(Server)中的資料項(Data Item，d_i)，都有自己本身的被點播率(NH_i ，表示此資料項目前被請求的次數)。每個資料項的被點播率，應隨著進入伺服器的請求(Request)而相對變化，如此一來更能符合真實的動態環境，因為此資料項的被點播率不應為一固定的機

率值。舉例來說(如圖 5 所示)，假設 Server 內有 n 個資料項，而第 68 個資料項(d_{68})在 T_1 之前本身的被請求次數為 24。在 T_1 這個時間點上有個用戶端請求 $d(r_k)=d_i$ (假設 $i=68$)，因此，d_{68} 的被請求次數在經過 T_1 的時間點之後應相對應的更改為 25。

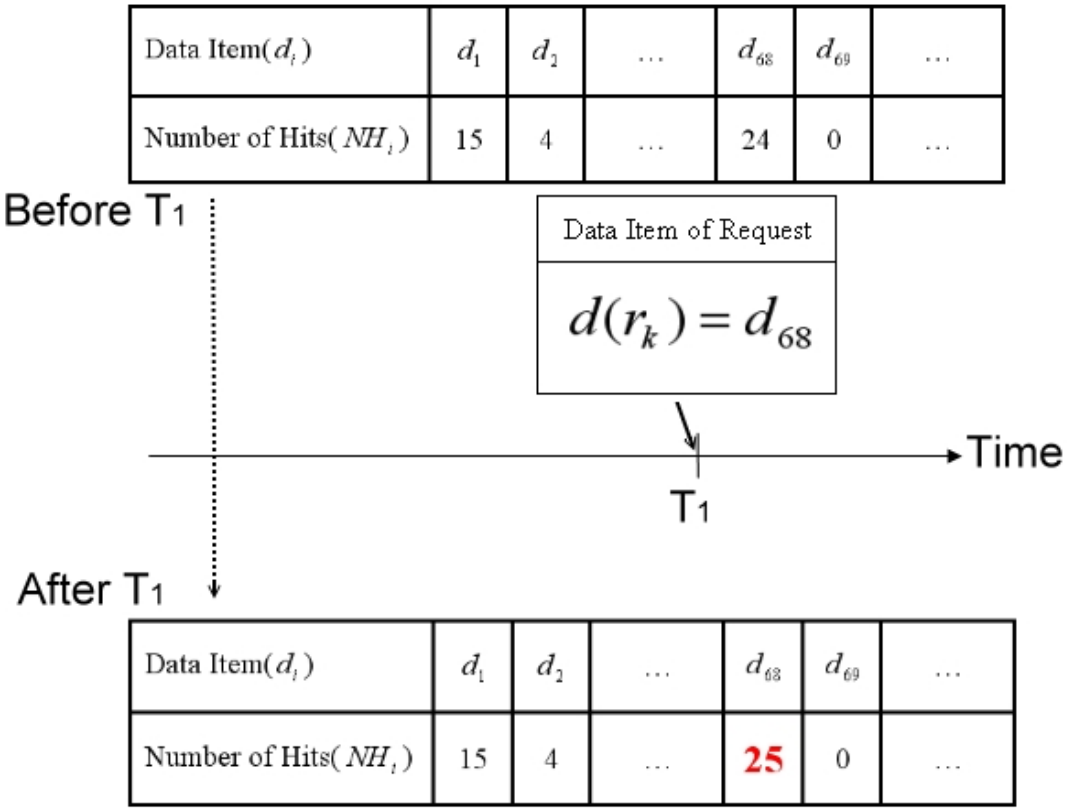

圖 5　動態環境圖

第四節　資料廣播的問題

在無線環境下伺服器透過廣播的方式將資料傳給用戶端，隨

著Push與Pull不同方式的運作而有著不同的適用方案。本研究討論以Pull為主的廣播方式[4][7][12][15][18][19][20]，也就是說當用戶端需要某資料項時，才發出請求(Request)給伺服器。然後伺服器再透過演算法的運算，適當的將請求的資料項排入廣播裡。用戶端在側聽伺服器端的廣播資料時，若遇到自己需求的資料才將此資料項收取下來。

一般的 Pull 系統基本運作方式為了能有效解決冷門和熱門資料被點播率的不同而提出各種不同的演算法。在 Pull 系統的運作之下，用戶端可以在較短的時間內就可收到較熱門的資料。但是，在上述許多排程演算法的發展之際，未能考慮到一個問題：動態環境的問題(The Problem of Dynamically Environment, PDE)。

我們發現許多演算法推演的假設中，常設定資料項為固定的被點播機率。例如說：某個資料項設定被點播機率為 0.24，那麼在演算法的運作之下，此資料項被請求的機率值便為 0.24，也就是說當系統在運作時，在 100 個 Request 裡會有 24 個 Request 要求播送同一資料項(不符合真實動態環境的要求)；實際上資料項的請求次數不應為固定值，應以實際的情況而定。

因此，在符合動態環境的要求且同時考慮資料項的請求次數

與等待時間之下，若是每個請求r_k在進入伺服器時，都需重新計算資料項的總重值來決定廣播的資料項，那麼將耗費很大的成本在資料項的運算上。此外，若是廣播前透過演算法重新計算廣播的資料項，亦會造成伺服器端額外的負擔，且浪費過多的資源。圖 6 說明此問題的情況。

在 Pull 架構下，當 Client 產生 Request 而傳送至 Server 端時，Server 未能即時處理，因此先置放於 Server 的佇列(Queue)裡。待 Server 所使用的演算法將處理資料項時，再從佇列裡取出，然後透過演算法的計算，將適合的資料項擺放至合適的廣播位置。在這樣的環境之下，每當廣播時，Server 的運算是非常耗資源的。因為在符合動態環境的要求下，一般演算法(Common Algorithm)需全部重算每個資料項的數值才能決定廣播那一個資料項。

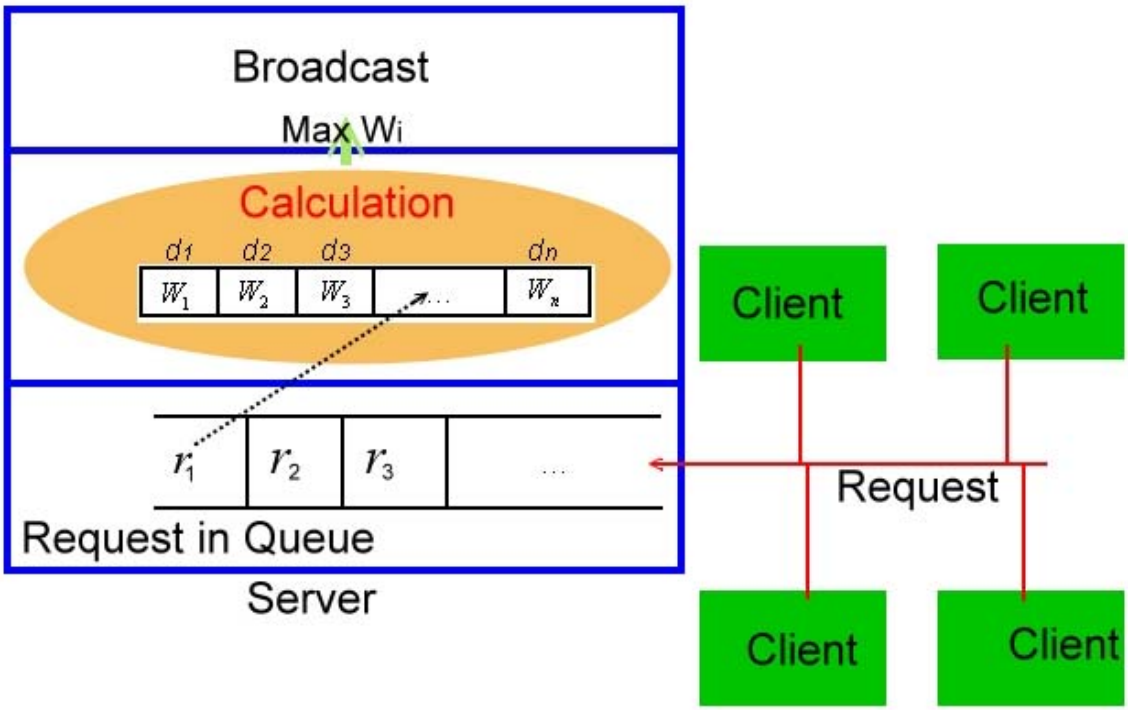

圖 6　動態環境的問題

在圖 6 裡，假設$d(r_1)=d_{12}$、$d(r_2)=d_2$、$d(r_3)=d_{35}$，表示 Client 端的r_1、r_2、r_3分別請求廣播d_{12}、d_2、d_{35}。而這些請求將會進入 Server 的排隊佇列中，等待計算後排入廣播頻道裡。為了符合動態環境的要求，且同時考慮點播權重值與時間權重值，一般演算法(Common Algorithm)在決定廣播資料項之前，必需先計算全部資料項裡每個資料項的總重值，再決定 Max(W_i)的資料項。這樣的方式將浪費大量的資源在運算上。下列為 Common Algorithm 廣播程序的虛擬碼。

- Common_broadcast

  Loop

    For i = 1 to n

      Calculation W_i

    Broadcast Max(W_i)

  End loop

在這樣的情況之下，我們希望演算法可以有效、快速的廣播隨時進入的 Request，以降低排隊在 Server 佇列裡 Request 的數量。隨著演算法的運算方式不同，效能高的演算法將可以有較好的效能解決此一問題，讓 Server 更有效率的運用。

另一個在動態環境下所衍生的問題是用戶端等待所請求資料項的等待時間問題(The Problem of Dynamically Waiting Time, PDW)，也就是資料項的饑餓現象。同樣的在動態環境之下，不同的用戶端可能在不同的時間點產生相同的資料項請求，對應於請求的資料項可以區分為冷門資料和熱門資料。

同時考慮到資料項的被點播率之外，希望用戶端在等待廣播所請求的資料項時，可以不用耗費太多的時間。也就是說，希望等

待越久的資料項可以有較高的優先權廣播，如此一來，將可以平衡冷門資料等待過久而未廣播的問題，有效解決廣播資料時的饑餓現象。

考量動態環境與上述的問題之後，我們提出一個動態排程演算法(Dynamically Schedule Algorithm, DYSA)。動態排程演算法可以有效解決在動態環境下的兩個問題 PDE 與 PDW，且降低伺服器端的運作負載。此外，更可以適用於真實的無線網路環境中。接下來將詳細說明動態排程演算法。

第四章　動態排程演算法

動態排程演算法可以有效解決在動態環境下所隱藏的問題(PDE 與 PDW)，本研究藉由提出此演算法有效提高伺服器端的運作效能。下列將完整詳細介紹說明動態排程演算法的運作。

第一節　主要概念

如前所述，演算法的目的除了要滿足動態環境的要求外，亦須能有效解決我們所發現一般演算法所忽略的兩個問題(PDE 與 PDW)。最重要的，是演算法必需能有很好的排程效能，有效降低伺服器的執行負載，進而提高伺服器的運作效能。

為了解決上述的問題，動態排程演算法同時考慮資料項的被點播率(點播權重值)與等待時間(時間權重值)，以計算每個資料項的總重值(Total Weight Value)作為排程依據。每個資料項經過方程式(1)的計算之後都有自己的總重值，演算法依此總重值建立最大堆積樹(Max-Heap tree)。總重值最高者為樹根(Root)。當用戶端的請求進入伺服器端時，伺服器只需重新計算此資料項的總重值，並

適當的更新或插入堆積樹的資料，而不需重算全部資料項之數值。

我們知道最大堆積樹在刪除頂端節點時是非常快速的，其時間複雜度為$O(1)$；但是在資料的更新或插入時是比較耗時間的，其時間複雜度為$O(\log n)$。為了讓堆積樹更有效率的運作，我們使用索引(Index)的技巧來加速堆積樹的搜尋。

在建立最大堆積樹時，除了依總重值建立之外，同時也建立每個資料項的索引值，用來記錄此資料項在堆積樹裡的位置。如此一來，在資料搜尋或更新時，只需找到資料項內的索引值(I_i)，即可快速的得到資料項在堆積樹內的位置。因此，在更新或插入新資料時，將可以是很有效率的。

動態排程演算法只需在每個用戶端的請求到達時，計算資料項的總重值和適當的更新或插入到堆積樹裡，並維護此堆積樹的穩定性。而在廣播的時後可以快速的抽出堆積樹頂端的資料項，不經過繁雜的運算，將可以有效提高伺服器的運作效能。動態排程演算法除了維護堆積樹之外，也將維護後的資訊寫回資料項的索引值裡，以確保資料一致性與可靠性。圖 7 說明動態排程演算法概念。

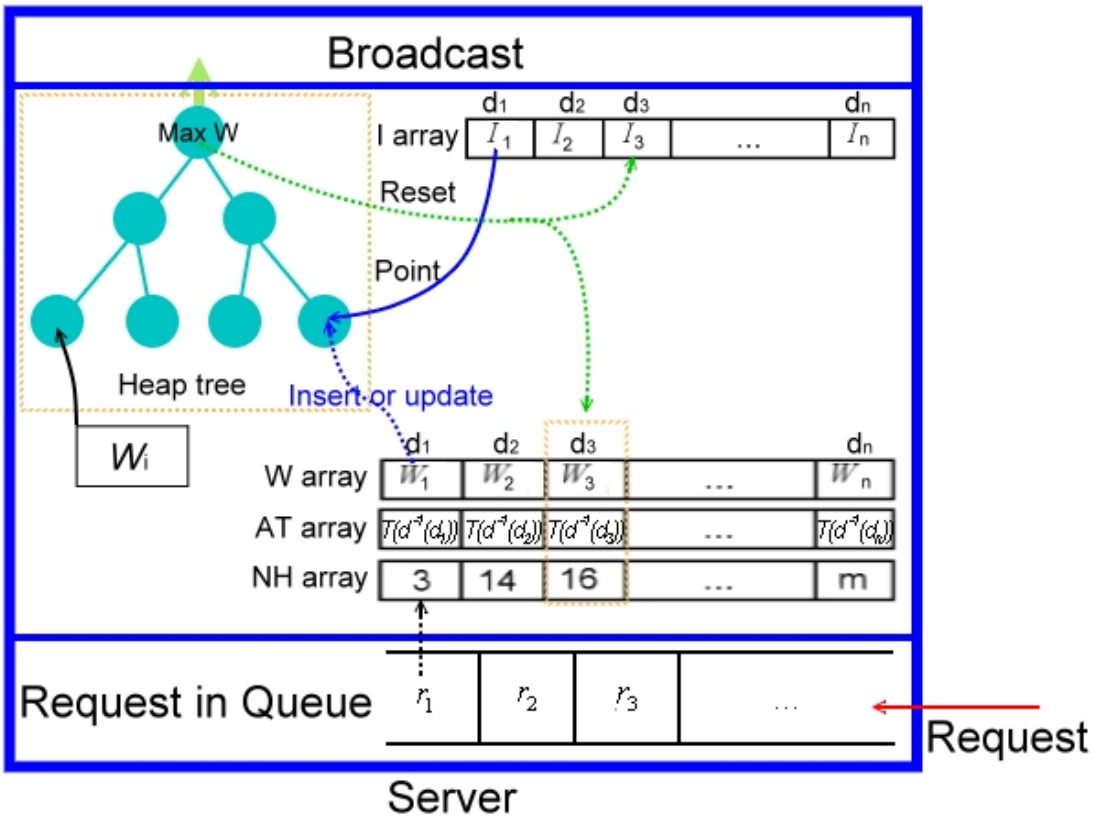

圖 7　動態排程演算法概念圖

用戶端(Client)的 Request r_k 透過無線網路傳遞至伺服器端(Server)而進入伺服器的佇列裡(如圖 7 所示)。Server 藉由 $d(r_k)=d_i$ 計算被請求的資料項，接著透過方程式(1)計算之後，每個被點播的資料項(d_i)將會產生自己的總重值(W_i)，再以此資料項之總重值排入堆積樹裡。當 Client 產生 Request 時，會先排入 Server 的 Queue 裡。若 Server 的演算法效率高，在 Queue 裡的 Request 數量則會相對的減少，甚至為空的(Null)。舉例來說，在圖 7 裡，當系統處理

Queue 裡的r_1請求時，會先找出r_1所要求廣播的資料項($d(r_k)=d_i$，假設 i=1)，接著判別$NH_1 \neq 0$(表示d_1已被請求過但尚未廣播的情況)，接著累加d_1的 NH array 內的數值，再重新計算d_1的W_1值。計算完W_1值後，依照I_1的索引位置找出在 Heap Tree 內的W_1值並適當的更新，最後藉由更新 Heap Tree 內的W_1值並維護 Heap Tree 的正確性。此時，若是有節點位置需變動的，需在變動後寫回 I array 內以維護 Index array 的正確性。

動態排程演算法每次提取一個 Request，並由$d(r_k)=d_i$得知 Client 請求播送d_i，接著計算d_i的總重值(Total Weight Value，W_i)。這裡分為兩種情況：NH_i為 0 或非 0。

當資料項d_i的被點播率為 0 時($NH_i=0$)，表示此資料項未曾有 Client 請求過，或是經由廣播此資料項之後將之重設為 0。此時，動態排程演算法更新相對應的 NH array($NH_i=NH_i+1$)和 AT array($T(d^{-1}(d_i))=now$，表示被請求廣播資料項第一次進入的時間)。AT array 只有在當此資料項第一次被點播時(此時$NH_i=0$)才會記錄，且取時間差的倒數做為此資料項的時間權重值($BS_i=1/(T(d^{-1}(d_i))-T_0)$)，因為希望較早進入而未廣播的資料項能有較高的時間權重值做為廣播的依據。

舉例來說，(如圖 8 所示) r_{25} 進入排程演算法的時間為 T_1，假設 $d(r_{25})=d_{12}$，因此 d_{12} 時間權重值為 BS_{12} ($BS_{12}=1/(T(d^{-1}(d_{12}))-T_0)=1/(T_1-T_0)$)；假設 $d(r_{26})=d_2$，則 d_2 時間權重值為 BS_2 ($BS_2=1/(T(d^{-1}(d_2))-T_0)=1/(T_2-T_0)$)。我們取此段時間差的倒數便可以使 BS_{12} 的數值大於 BS_2，此方法可以讓越早進入而未廣播的 d_i 有較高的時間權重值。

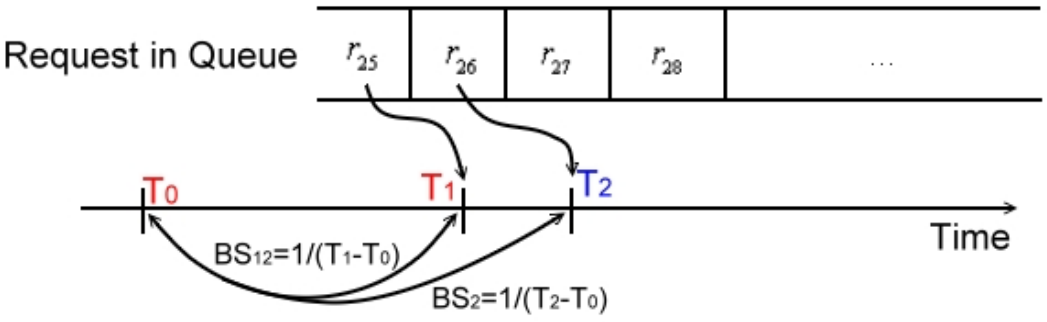

圖 8　時間權重值示意圖

另一種情況為資料項的被點播率為非 0($NH_i \neq 0$)，也就是說曾經有用戶端請求過，但未廣播的情況。此時動態排程演算法累加 d_i 的 NH_i ($NH_i = NH_i + 1$)，接著重新計算總重值(但不更動 d_i 的 AT array，使其保持在較大的數值)，然後依據 d_i 的索引(I_i)可以快速的找到此資料項在堆積樹裡的位置，並更新資料後重新維護堆積樹。此外，由於堆積樹節點的位置與索引有關，因此在維護堆積樹之後

亦需更新索引內的數值。

在動態排程演算法的廣播程序裡，是相當快速的，只需廣播堆積樹樹根(Root)對應的資料項。在廣播之後，必需寫回所對應資料項的陣列裡，使其重設為 0，以表示此資料項已廣播(假設 Root 對應於 d_i，則 $NH_i = 0$, $T(d^{-1}(d_i)) = 0$, $W_i = 0$, $I_i = 0$)。在圖 7 裡，假設 Heap Tree 的 Root 對應於 d_3，因此當廣播之後需將 d_3 的相關 array 重設為 0($NH_3 = 0$, $T(d^{-1}(d_3)) = 0$, $W_3 = 0$, $I_3 = 0$)。

當資料項的被點播次數為 0 時(Number of Hit，$NH_i = 0$)，且新的 Request 進入後，資料項的 AT array 和 NH array 都必需重新計算。當資料項的被點播次數為非 0 時(Number of Hit，$NH_i \neq 0$)，亦需重新計算此資料項目的 Total Weight Value，但此時只累加 NH array 裡相應的資料項($NH_i = NH_i + 1$)，而不更動此資料項的 AT array。同時，在廣播之後的維護時，必需寫回 I array 或資料更改相對應的 array 內，這將可以維持資料與 Heap Tree 的一致性。

透過上述的方式，我們所提出的 DYSA 可以快速的運作，以提高 Server 端的效能。

第二節　運作程序

當系統開始運作時，所有資料項的數值皆為空的。首先，依資料項的數目建立相同的陣列，分別為 NH array、AT array、W array、I array 等四個陣列。NH array 用來存放每個資料項的被點播次數。當 Request 進入時，所要求的資料項便會累加，每次增加 1 個單位值，數值越高表示此資料項被請求的次數越多。AT array 記錄此d_i第一次被請求播送的時間$T(d^{-1}(d_i))$，以供計算時間權重值(BS_i)所用。時間權重值為一段時間間隔的倒數($BS_i = 1/(T(d^{-1}(d_i)) - T_0)$)，若此資料項進入演算法的時間較晚，則相對的時間權重值較小，這是因為考慮較早請求的資料項應給予較高的加權值。此時間權重值只會在d_i的被點播率為 0 時才會記錄。也就是說第一次請求此資料項或當資料項被廣播後且未曾有 Request 請求此資料項時，才會記錄此 Request 的到達時間。

除了廣播之後的重設動作會將對應資料項的 AT array 重設為 0 之外，沒有其它程序會更改此資料項的 AT array。W array 用以存放經過計算後的總重值(Total Weight Value)。

在經過計算之後，每個資料項有自己相應的W_i值，便存放於 W array 裡。I array 則是用來記錄此資料項在堆積樹內的位置，用以

加速堆積樹的搜尋。

在介紹完動態排程演算法的概念後，接下來描述說明演算法的完整運作程序。動態排程演算法分為三個部份在運作，分別是插入、更新、廣播等三個不同的運作程序。以下將分別介紹各程序的運作。

- 插入

當 Client 的 Request 透過無線傳遞的方式到達 Server 端時，會先排入 Server 的 Queue 裡。動態排程演算法此時先取出在 Queue 裡的 Request(r_k)，此 Request 請求廣播某一資料項 d_i（$d(r_k)=d_i$）。如果 d_i 的被點播率為 0（$NH_i=0$），表示此項目未曾有 Request 請求廣播過，或是經由廣播後重設為 0。因此，必需先將 Request 進入的時間記錄（$T(d^{-1}(d_i))=now$），然後記錄於 AT array 裡，以供計算時間權重值時所用。接著使用方程式(1)來計算 d_i 的 W_i 值（$W_i=\alpha * NH_i+(1-\alpha)* BS_i$）。接著在 Heap Tree 插入總重值為 W_i 的新節點，同時將插入後的相對位置寫回 I array 裡。

- 更新

在另一方面，如果d_i的被點播率為非 0($NH_i \neq 0$)，表示d_i曾有 Client 請求過，且未廣播。然後我們只需將d_i的被點播率加 1($NH_i = NH_i + 1$)，並重新計算W_i值(此時，不更新此$T(d^{-1}(d_i))$數值)。接著，再於 I array 裡找到d_i堆積樹內的相對位置資訊(I_i)，然後再更新堆積樹內的W_i值，並適當的維護堆積樹的完整性與寫回相對位置資訊至 I array。

透過這樣的方式，每次只需更新一個資料項的資訊，且可以快速的計算及搜尋維護堆積樹，這使動態排程演算能很有效率的運作。

- 廣播

在動態排程演算法的主要訴求之一為快速，因此，在廣播時可以非常快速的決定廣播那一個資料項。因為演算法維護堆積樹的完整性，所以在廣播時只需廣播堆積樹 Root 對應的資料項，然後維護 Heap Tree 的正確性。同樣的，在堆積樹的維護完之後都必需寫回 I array 以確保相對位置資訊的正確性。此外，假設廣播d_i，在廣播程序完成後，亦需將d_i的 NH array、AT

array 、 W array 、 I array 內的值重設為 0($NH_i = 0$, $T(d^{-1}(d_i)) = 0$, $W_i = 0$, $I_i = 0$)。表示 d_i 目前沒有任何 Client 請求，直到有 Client 請求而進入插入程序才會再次更改相對陣列內的數值。

第三節　動態排程演算法

我們所提出的動態排程演算法可以有效提高伺服器端的效能，減少排隊等候在佇列內的請求。接著下列為動態排程演算法的虛擬碼。

- Update

When Request arrival(r_k):

$d(r_k) = d_i$

If $NH_i \neq 0$

$NH_i = NH_i + 1$

$W_i = \alpha * NH_i + (1-\alpha) * BS_i$

Call Heap_Tree_Update(W_i)

Maintain I array

End If

- Insert

When Request arrival(r_k):

$d(r_k) = d_i$

If $NH_i = 0$

$NH_i = NH_i + 1$

$T(d^{-1}(d_i)) = now$

$W_i = \alpha * NH_i + (1-\alpha) * BS_i$

Call Heap_Tree_Insert(W_i)

Maintain I array

End If

- Broadcast

Loop begin

d_i=root

Call Heap_Tree_Delete(root)

Reset $NH_i = 0$, $T(d^{-1}(d_i)) = 0$, $W_i = 0$, $I_i = 0$

End loop

第四節　堆積樹的運作

DYSA 主要是透過 Maximum Heap Tree 的方式來快速的決定廣播的資料項，每個 Data Item(假設為第 i 個 Dtaa Item，d_i)透過演算法的運算而得到自己本身的W_i值，Heap Tree 內 node 的運算值便以此W_i值做為運算元。本節將詳細說明 DYSA 內 Heap Tree 的運作方式。

堆積樹的特性：

1. 堆積樹是一個完整二元樹。
2. 每一個節點之鍵值大於或等於其子節點之鍵值。
3. 樹根節點之鍵值是堆積樹中之最大者。

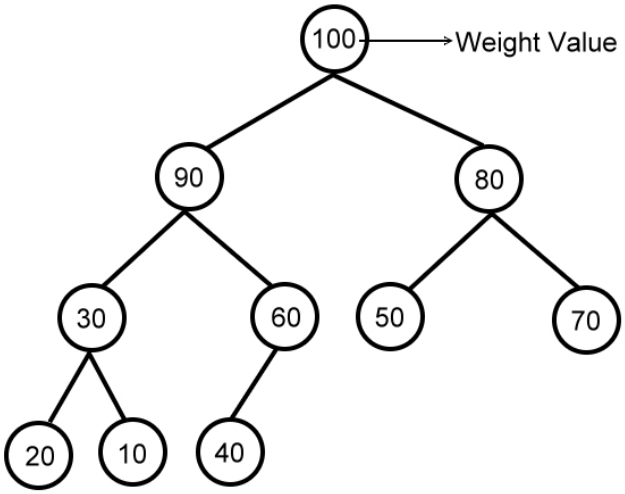

圖 9　堆積樹狀圖

圖 9 顯示 Heap Tree 的範例。在 Heap Tree 裡，最大值 Node 會總是保持在樹根。而在進行廣播(刪除樹根節點)時，最底層的最右邊 Node 移至樹根，接著比較本身和子節點。若子節點的值大於自己本身的值，則會選擇兩個子節點之中最大值者進行位置交換(如圖 10 所示)。如此重覆執行，直到子節點的值未大於自己本身的值。

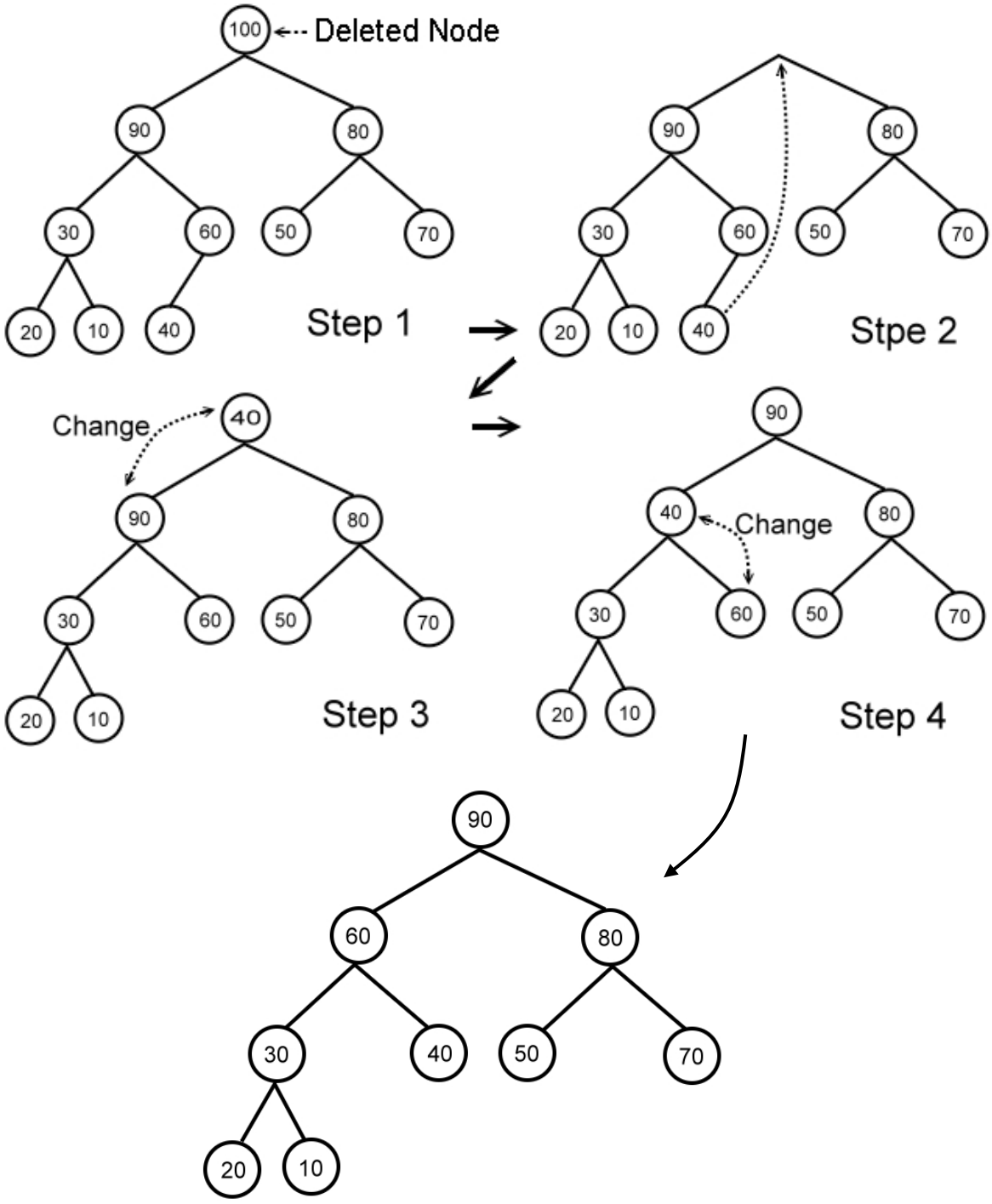

圖 10　堆積樹刪除樹根步驟圖

另一方面，當有新的 Node 插入 Heap Tree 裡時，會先安插至

Heap Tree 最後的位置上，然後與自己本身的父節點比較。若本身的值大於父節點，則會與父節點進行交換的動作，直到父節點的值比本身大，或是到達樹根。圖 11 顯示在 Heap Tree 裡執行插入的動作程序。

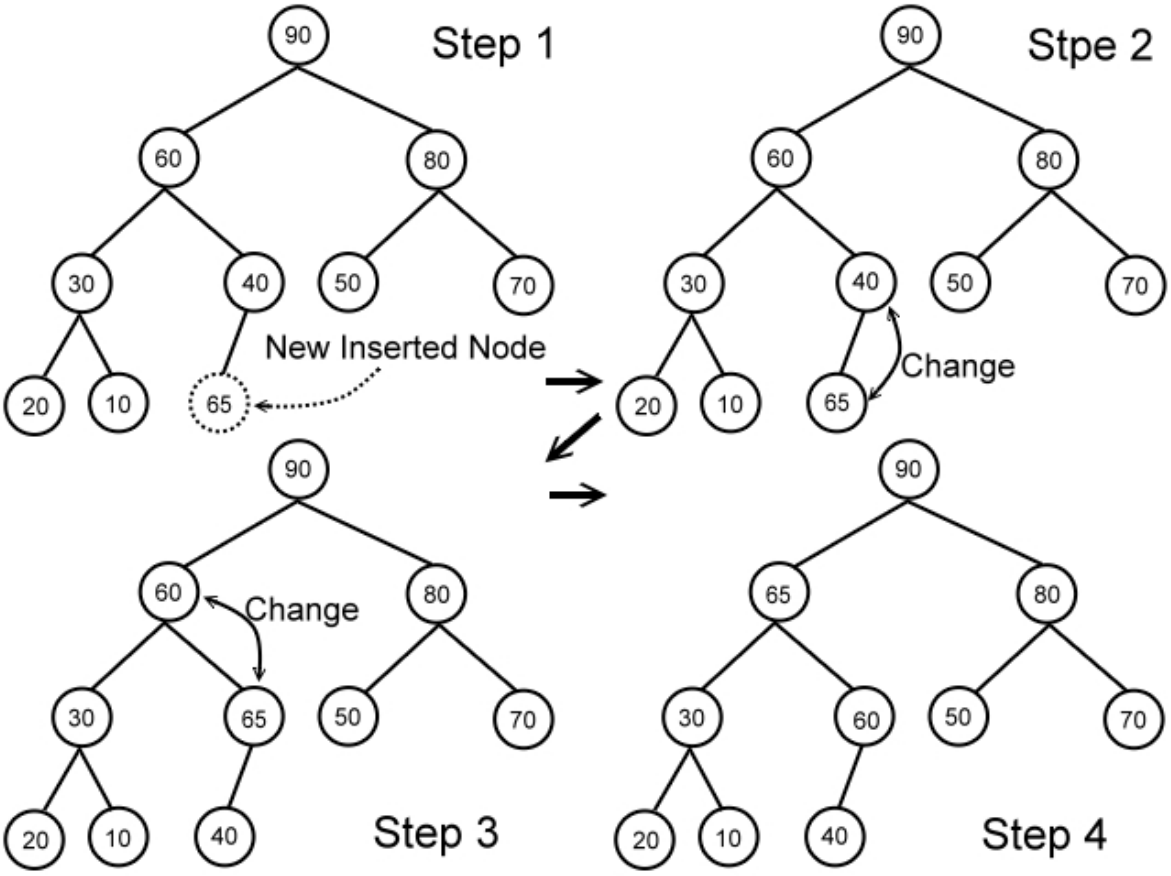

圖 11　堆積樹插入節點步驟圖

在 DYSA 演算法裡的 Heap Tree 每一個 node 代表自己本身的 Data Item。而 Data Item 的 Total Weight Value 可能隨時改變，因此仍需要在更動 Node 值之後維護 Heap Tree 的完整性。如同插入

程序，若節點的值更改，只需與父節點的值進行比較。若本身的值大於父節點，即進行調換的動作。直到父節點的值大於自己本身的值或是到達樹根。圖 12 說明修改 Node 內的W_i值的運作程序。

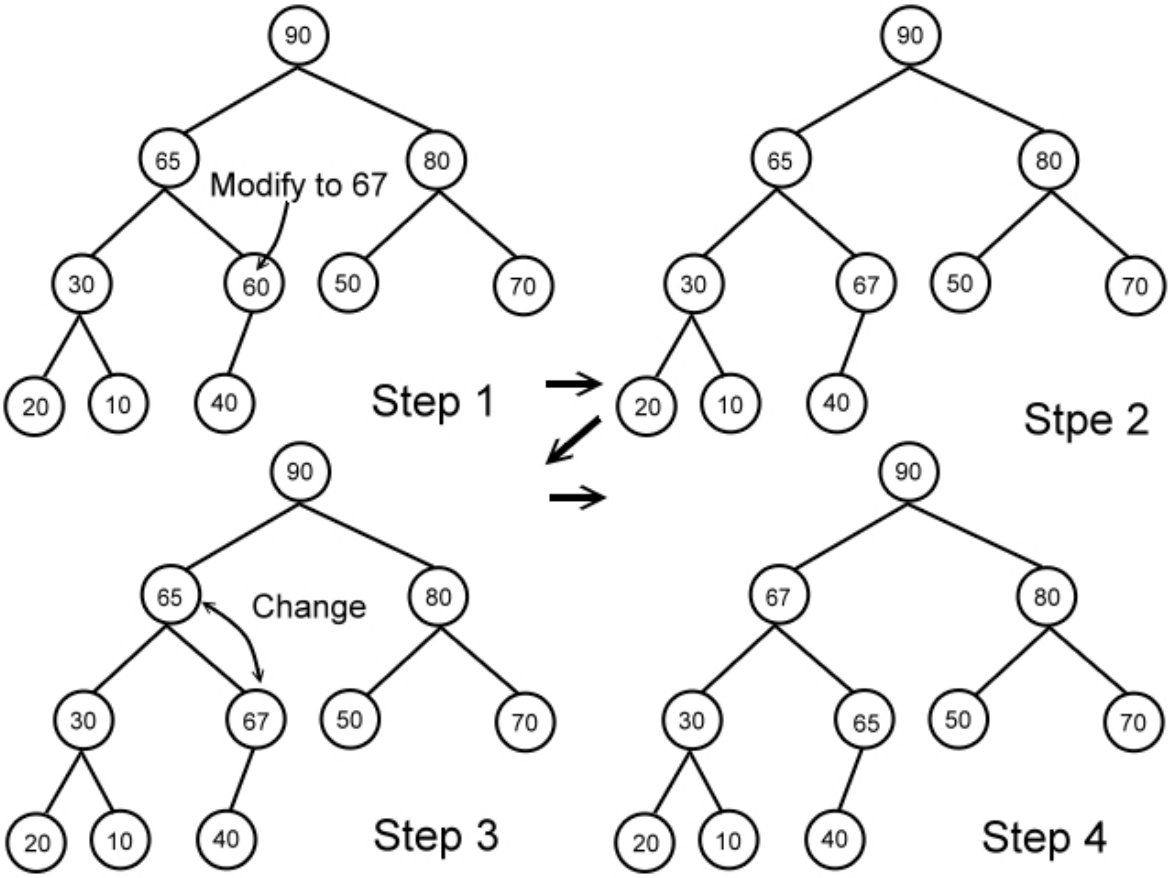

圖 12　堆積樹節點更改步驟圖

在系統裡，將 Heap Tree 以線性串列的結構呈現。假設目前 Heap Tree 裡有 k 個節點，且以 $Heap\_Tree[i]$ 表示第 i 個節點。下列為

Heap Tree 運作程序的虛擬碼。

- Heap_Tree_Insert(node=W_i)

```
If Tree is Empty
    Insert node to be root
ELSE
    Insert the node to Heap_Tree[k+1]
    While the father node weight less than the inserted node
    weight
        Exchange the father node and the inserted node
        Return changed nodes
    End While
END IF
```

- Heap_Tree_Delete()

```
Remove root
Move Heap_Tree[k] to Heap_Tree[0]
While one of children nodes weight greater than itself
```

weight

Exchange max(Left_Child, Right_Child) with itself

Return changed nodes

End While

- Heap_Tree_Update(node=I_i)

Set the node weight with the W_i

While the father node weight less than the modified node weight

Exchange the father node and the modified node

Return changed nodes

End While

第五節　時間複雜度

時間複雜度通常用以衡量程式、演算法的執行效能。本章節的部份將比較我們所提出的DYSA與一般演算法(Common)的時間複雜度，藉以此衡量來顯示演算法的效能。

在計算演算法的時間複雜度時，忽略微不足道的細項，例如

載入 Query 的程序，只針對各個演算法最重要的部份進行運算。表 2 顯示計算之後的結果。

表 2　時間複雜度分析表

		插入	刪除	更新
最好的情況	COMMON	O(n)	O(n)	O(n)
	DYSA	O(1)	O(1)	O(1)
最差的情況	COMMON	O(n)	O(n)	O(n)
	DYSA	O(log n)	O(log n)	O(log n)
平均的情況	COMMON	O(n)	O(n)	O(n)
	DYSA	O(log n)	O(log n)	O(log n)

註：COMMON 表示一般演算法

時間複雜度的分析分別區分為將請求(Request)的資料項(Data Item)加入演算法的運作裡，記錄為插入；經演算法運算後決定廣播的資料項，記錄為刪除；Client 請求廣播 d_i 且 $NH_i \neq 0$ 的程序，記錄為更新。此外，分別顯示演算法之間最好、最差、及平均的情況。

從表 2 得知，一般演算法在進行插入動作時，最好與最差的

情況下時間複雜度皆為 O(n)，因為只要當 request 進入時，皆需全部重算所有資料項的總重值。DYSA 在最好的情況之下時間複雜度為 O(1)，因為不需重整 Heap Tree 的正確性，而 DYSA 最差的情況為 O(log n)。

在廣播(刪除)的部份是演算法最重要的部份，較高的效率可以使演算法可以有較快速的運作效能。在此部份，演算法透過計算而決定每個資料項目的 Total Weight Value，並選取最大 Total Weight Value 值對應的資料項做為廣播。從表 2 來看，一般演算法不論是在最好的情況或是最差的情況，在廣播之前都必需對所有資料項目進行運算，因此其時間複雜度為 O(n)，而平均情況的時間複雜度亦為 O(n)。相對於 DYSA，由於資料項在進行插入後，已計算完總重值(Total Weight Value)，因此在廣播時隨時可選擇 Max Total Weight Value 的資料項(Heap Tree Root)做為廣播，而不用再重新計算。最好的情況為不需重整 Heap Tree，其時間複雜度為 O(1)；而最差的情況則需完全重整 Heap Tree 的正確性，因此最差情況下時間複雜度為 O(log n)。

在資料更新的部份，最好的情況下一般演算法仍只有 O(n)的效能，而 DYSA 卻能有 O(1)的效能。在另一方面，一般演算法在最差

的情況下時間複雜度亦為 O(n)，DYSA 為 O(log n)。由於 DYSA 使用 Heap Tree 的資料結構做為演算法的基礎，為了改進 Heap Tree 較沒效率的搜尋，在 DYSA 演算法裡皆使用了 Index 的方式來加速在 Heap Tree 內資料的搜尋。雖然加速 Heap Tree 的搜尋，但在 Heap Tree 的 Maintain 上仍無法突破 O(log n)的限制。雖然如此，DYSA 的效能仍是高於一般演算法許多。

透過時間複雜度的詳細分析，我們所提出的 DYSA 在各種情況之下皆能有較好的效能，接下來將以實驗的數據來實際比較不同演算法之間的效能。

第五章　模擬實驗

為了驗證動態排程演算法確實有較好的效率，及能解決在動態環境下的問題，我們以模擬實驗的方式呈現。模擬結果將以數據、圖表顯示。以下將分別說明模擬環境、參數和最後的實驗結果。

第一節　模擬環境

模擬環境使用爪哇(Java)語言做為模擬程式的撰寫語言，考量的因素為 Java 提供的多執行緒、物件導向等功能，較符合程式設計的需求。

首先，撰寫模擬的動態環境。在動態環境下，隨時會有 Client 端產生的 Request 進入 Server 端，每個 Request 會請廣播某一資料項。另外，Client 產生的 Request 則以不同的 Arrival Rate 做為模擬。

實驗模擬動態排程演算法和一般演算法的運作，比較兩個不同演算法在經過某段固定時間後，仍然排列在 Queue 裡的數量。背景環境相同，而 Queue 裡數量較少者表示其演算法有較好的處理效

能。此外，模擬實驗也比較在不同的 Arrival Rate 下兩個演算法的處理量(Throughput)。

當 Request 進入 Server 後會待在 Queue 裡等待演算法的處理。為了符合動態環境的要求，演算法都必需隨著處理的 Request 而重新計算 Weight 值。一般的演算法(Common)在這個部份因每個進入的 Request 都必需將所有的資料項重新計算，因此耗費大量的運算時間在計算 Total Weight Value 上。而動態排程演算法只須計算 Request 相應的資料項，並適當的維護堆積樹的完整性。

兩個演算法 Request 進入的方式都相同，廣播的方式亦相同，以求公平呈現演算法的效能差異。程式模擬相同的 Request 進入到 Server 的 Queue 裡，也透過相同方程式(1)的計算公式來計算 Total Weight Value，並以此 Total Weight Value 做為廣播的依據。

在 Client 所產生的 Request 部份，使用兩種不同的情況分別模擬。一種為資料項隨機(Randomize)分配產生的 Arrival Rate，也就是請求(Request)資料項為隨機分配，沒有次序的。另一個模擬 Request 產生 Arrival Rate 的部份使用 Zipf 的分配，也就是說資料項的被點播率呈 Zipf 分配，其機率值與θ傾斜角度相關。θ越大表示資料項被點播的機率呈較垂直的分配，反之則表示資料項的被點

播率越趨近相等。Zipf 分配被普遍的使用在許多模擬程式的設定上，它主要為資料項的被點播率有其相對的機率關系，就像一般文章裡英文字母出現的機率一樣有高有低。因此，為了能讓實驗結果更有可信度，加入以 Zipf 分配的資料項產生 Request 的比率。在本實驗中， Zipf 分配的θ偏斜值若無特定說明，則皆設定為 0.5。

第二節　模擬參數

在模擬環境的參數，使用下列的參數。

- α：使用者自訂參數
- Number of Data Item
- Arrival Rate
- Execute Time
- Number of Request
- Zipf 分配的θ偏斜值
- Throughput

表 3　隨機分配的參數

Number of Data Item	100
Arrival Rate	20/sec 25/sec 30/sec
α 值	0.6

表 4　Zipf 分配的參數

Number of Data Item	100
Arrival Rate	20/sec 25/sec 30/sec
α 值	0.6
Zipf θ 值	0.5

表 5　處理量的實驗參數

Number of Data Item	100
Arrival Rate	5/sec~30/sec

α 值	0.6
Zipf θ	0.5

第三節　實驗結果

實驗模擬分成三個不同的部份。第一與第二部份皆為比較留在 Server 端 Queue 裡的數量。不同的是第一部份使用隨機分配產生的 Request，請求播送資料項的機率為隨機分配。第二部份請求播送資料項的機率為 Zipf 分配。

第三個部份比較不同的 Arrival Rate 下，兩個演算法的處理量。系統執行時間為 30 秒，Arrival Rate 為每秒產生 5 個 Request 到每秒產生 30 個 Request。

從圖 13、14、15 來看，我們所提出的動態排程演算法(DYSA)有很好的效能，在符合動態環境的要求且同時考慮資料項點播次數與等待時間下，DYSA 的執行效能皆高於 Common Algorithm。表 3 為圖 13、14、15 的實驗參數。圖 13 與圖 14 裡的動態排程演算法在 Arrival Rate 低於 25/sec 前，能全部處理 Client 所產生的 Request。在這樣的情況下，Server 若仍有 Request 的等待是因為產

生的 Request 與演算法所用的執行緒不相同，理論上來看是平行處理，實際上仍是有一小段誤差。若此誤差值小於 Request 產生的速度，則可以將這個誤差視為 0。這也表示動態排程演算法在 Data Item=100 時，Arrival rate 小於 25/sec 的情況下仍有空閒的時間。

圖 15 表示 Arrival Rate 為 30/sec 的情況。從圖來看，動態排程演算法雖然有未處理完的 Request 仍停留在 Server 的佇列裡，但是它的累積上升弧度仍是很小的。這表示動態排程演算法有很好的效能，能有效加速 Server 處理動態廣播排程的問題。

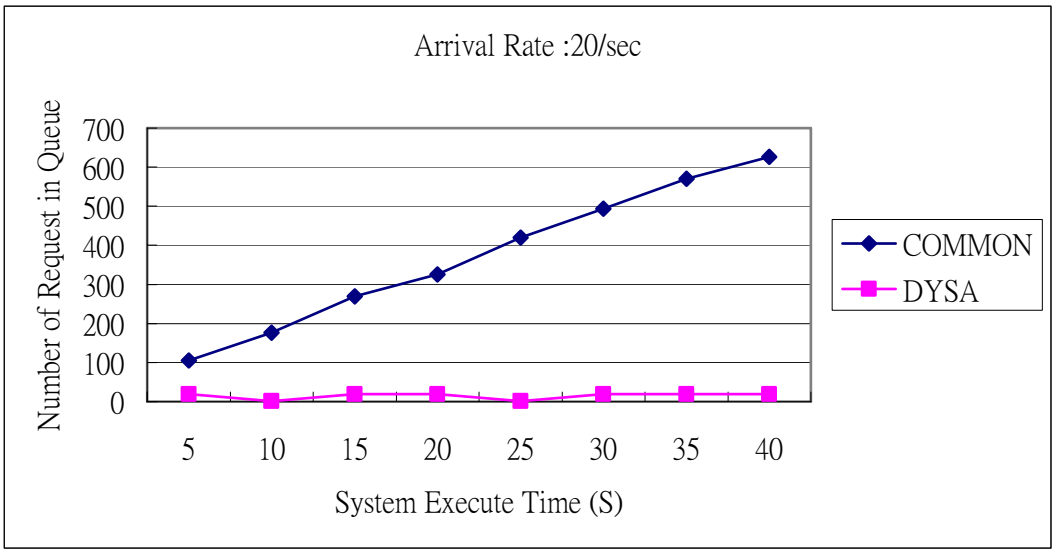

圖 13　實驗結果(Randomize 分配)_Server 端 Request 數量(1)

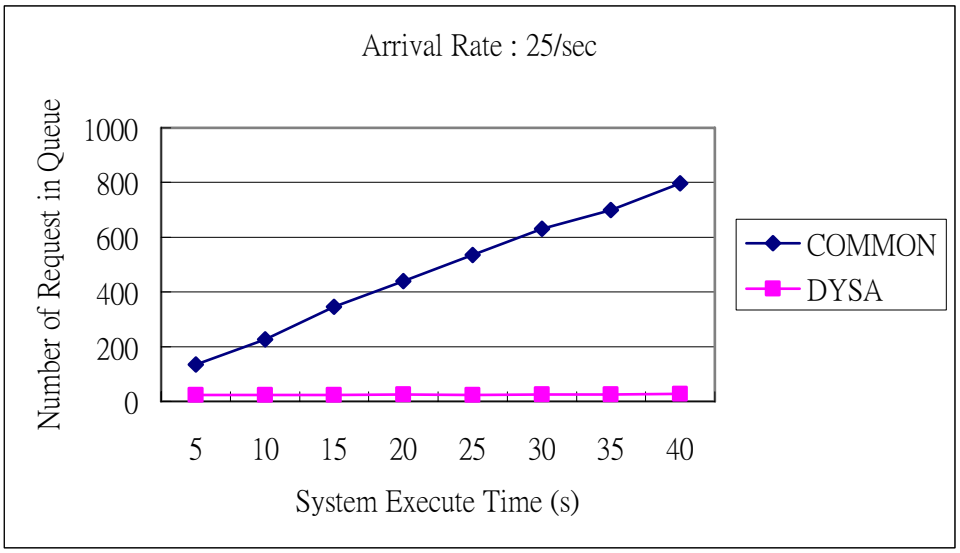

圖 14　實驗結果(Randomize 分配)_Server 端 Request 數量(2)

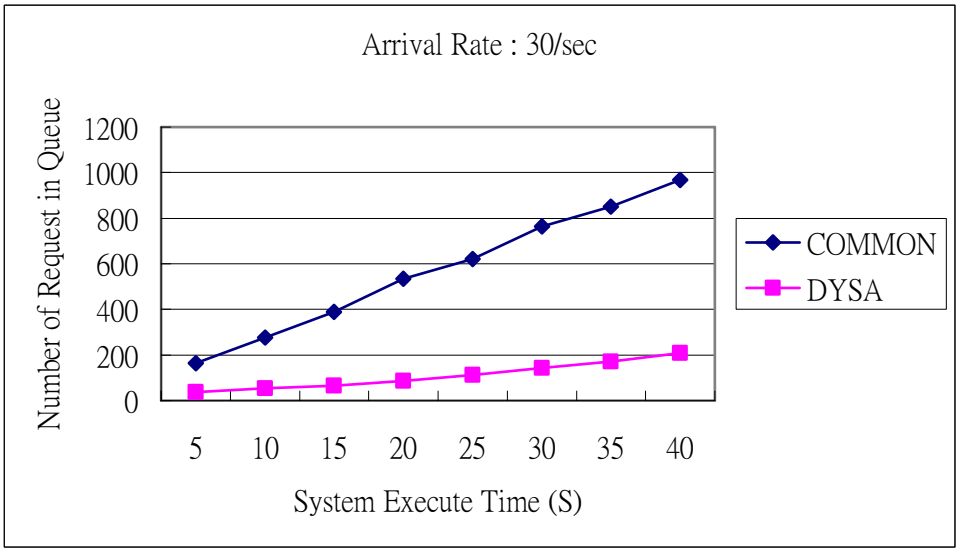

圖 15　實驗結果(Randomize 分配)_Server 端 Request 數量(3)

在另一個模擬的環境下，資料項請求的機率為 Zipf 分配，表 4 說明圖 16、17、18 的模擬參數。由於未能真實的使用多工系統來

模擬，因此在比較 Zipf 分配與隨機分配的數值，會發現使用 Zipf 分配之下，留在 Queue 內的 Request 較 Randomize 分配稍微較多一些。理由是因為單工系統模擬雙工的作業環境，加入 Zipf 分配的運算後將稍微多佔用一些系統的運算資源。因此，導致演算法的數值皆上升一些。這是合理的現像。同樣的兩個演算法的比較說明已在上述說明過，只差別在於使用 Zipf 分配。此部份的模擬結果參閱圖 16、17、18。

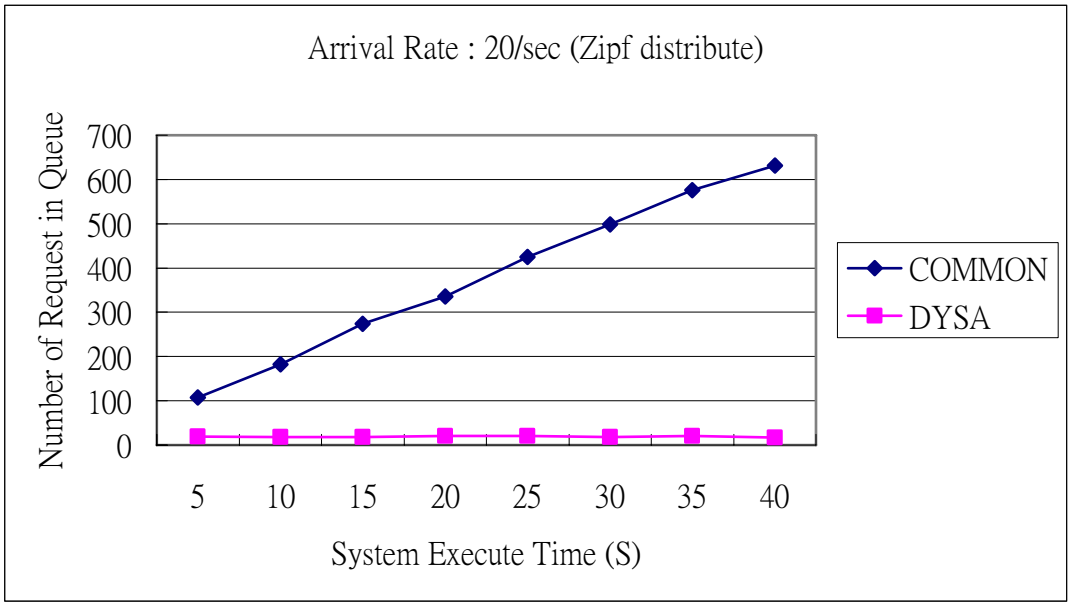

圖 16　實驗結果(Zipf 分配)_Server 端 Request 數量(1)

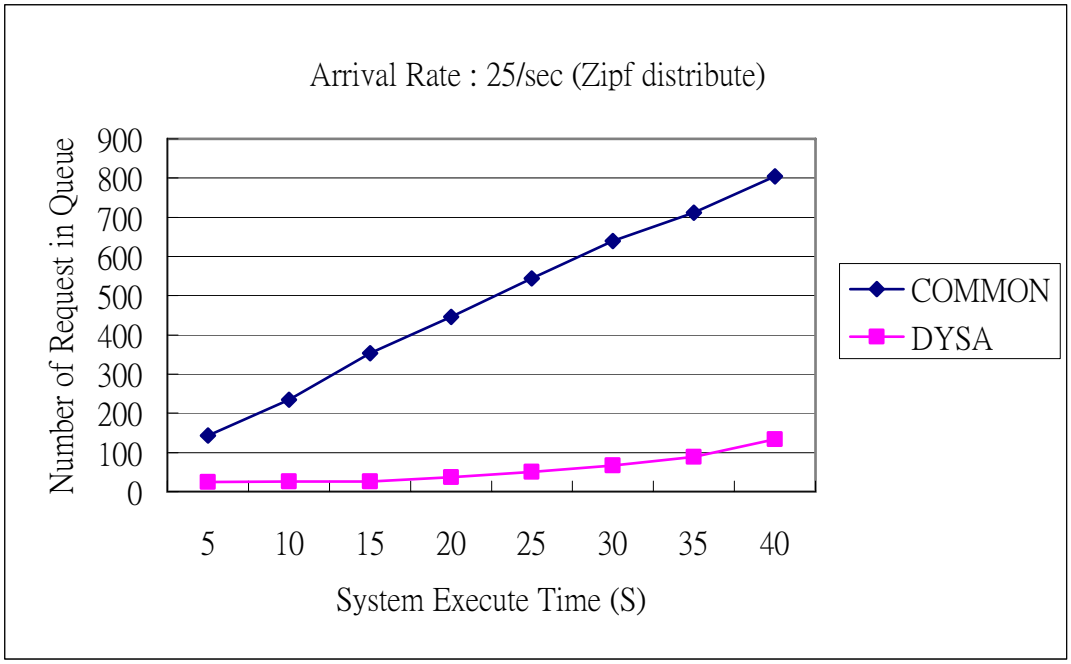

圖 17　實驗結果(Zipf 分配)_Server 端 Request 數量(2)

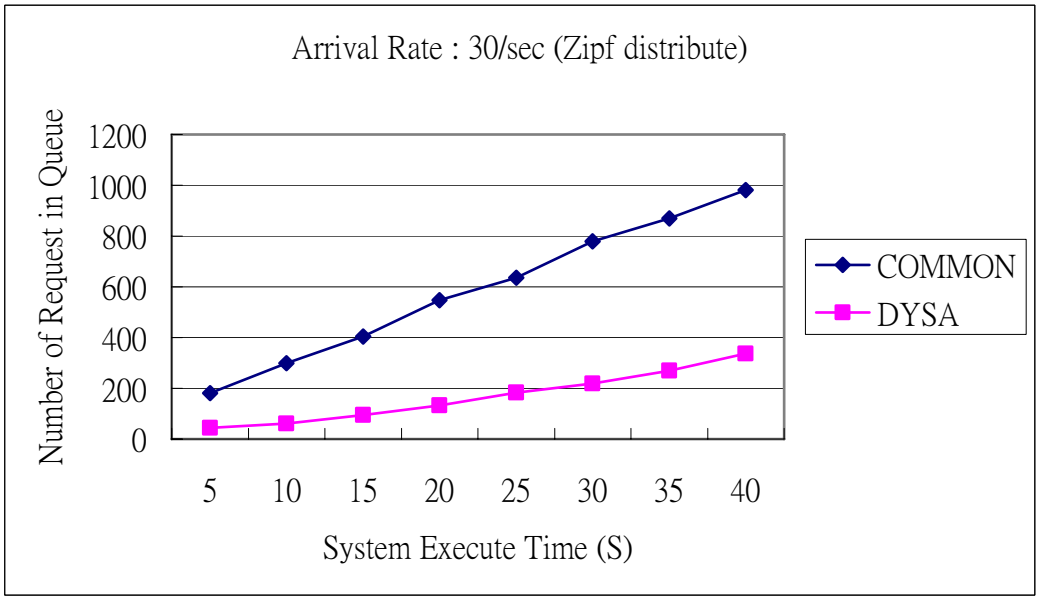

圖 18　實驗結果(Zipf 分配)_Server 端 Request 數量(3)

另一個模擬實驗為處理量的運算。分別比較有無使用 Zipf 分配的情況。計算處理量的方程式如下列公式：

(Handle/(Arrival rate*sec))*100%　　(2)

符號說明：

- Handle：Number of Broadcasted Data items
- Arrival_rate：Arrival Rate
- sec：System Execute time

模擬程式的執行時間為 30 秒，Arrival Rate 分別從 5/sec 遞增 5/sec，直到 Arrival Rate 為 30/sec(其餘參數參照於表 5)。圖 19 顯示動態排程演算法在 Arrival Rate=25/sec 前能 100%的將 Request 快速的處理，而傳統一般的演算法因為每個 Request 進入時皆需重算全部資料項的總重值(為了符合動態環境的要求)，因而耗費大量的運算時間，所以導致較差的執行效能。即使在 Arrival Rate 高於 25/sec 後，在 Queue 裡 request 的數量仍是緩慢的上升。

圖 20 顯示資料項請求機率為 Zipf 分配的結果，DYSA 效能仍

明顯高於一般演算法。演算法的處理量在 Zipf 分配時皆低於隨機分配，原因是因為採用 Zipf 分配時，多耗費些許的運算資源，而導致演算法效能稍微的下降。

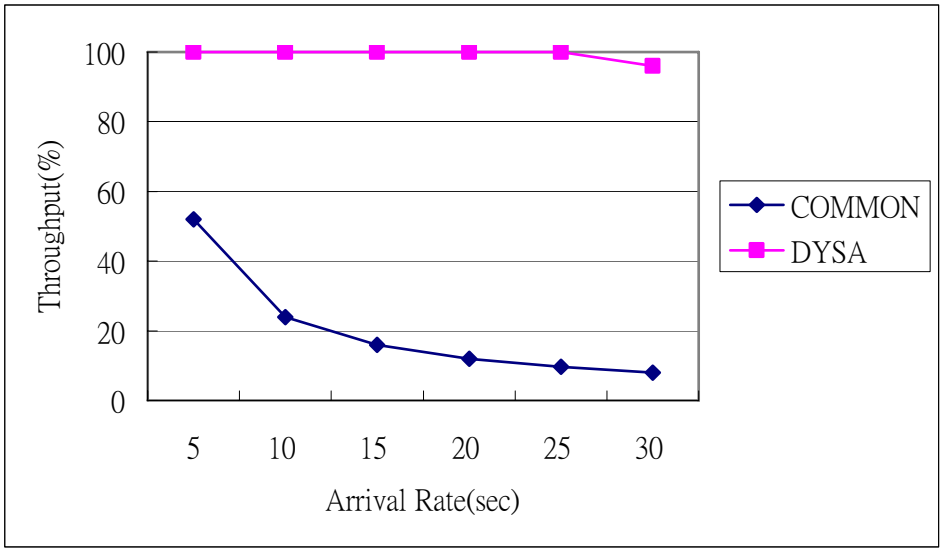

圖 19　實驗結果_系統處理量(Randomize 分配)

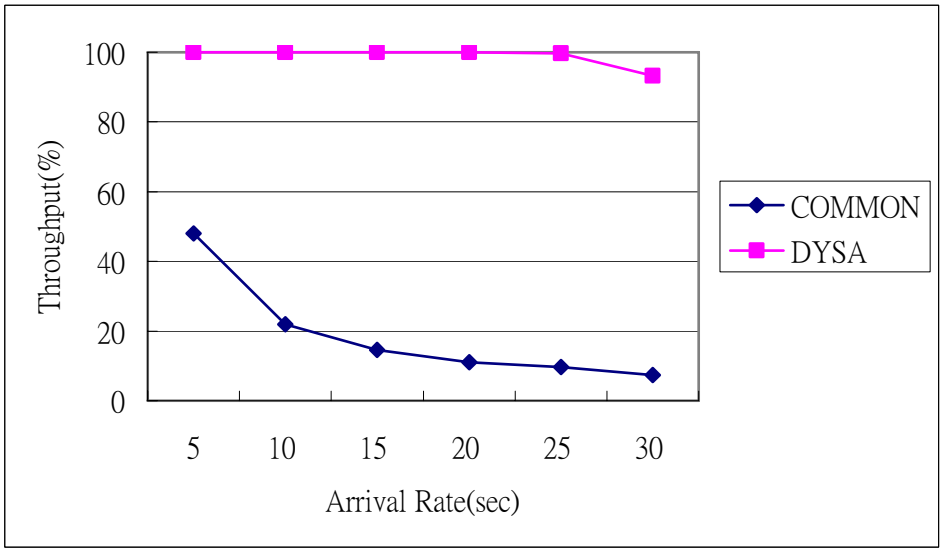

圖 20　實驗結果_系統處理量(Zipf 分配)

不論是一般演算法或是 DYSA 演算法，為了同時考慮資料項的點播率與等待時間，皆透過方程式(1) $W_i = \alpha * NH_i + (1-\alpha) * BS_i$ 來計算每個資料項的總重值，再廣播最高總重值之資料項。上述的模擬實驗，我們皆設定 $\alpha = 0.6$。其原因為資料項的被點播率較 Request 的等待時間為重要，因此資料項的被點播權重值與時間權重值之比值設定為 0.6:0.4。

為了能更廣範的呈現真實的環境，另外模擬 α 值分別從 0.9 遞減至 0.1，以求得更真實的數據。圖 21、22 顯示不同 α 值的實驗結果。

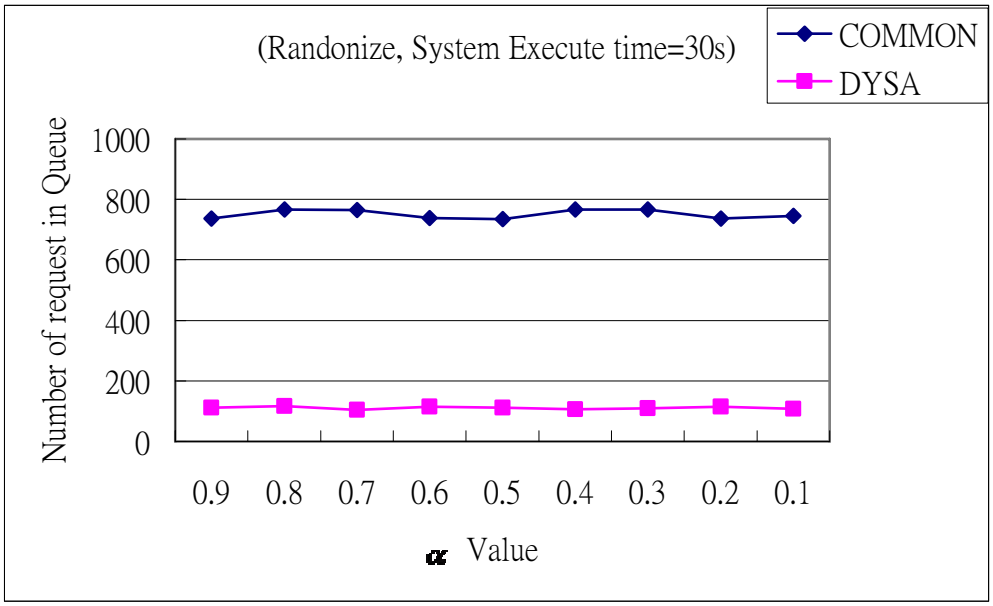

圖 21　實驗結果_Server 端 Request 數量分佈圖(Randomize 分配)

圖 22 顯示使用固定的 Arrival Rate，不同α值的情況。在模擬裡，使用固定的 Arrival Rate=30/sec，資料項的總數為 100，也就是說每秒由 Client 產生 30 個隨機分配的 Request 送達 Server 端，程式執行的時間為 30 秒。

從圖 22 裡，發現演算法用以計算總重值的方程式 $W_i = \alpha * NH_i + (1-\alpha) * BS_i$，其不同之$\alpha$值對演算法的效能幾乎無影響。不同$\alpha$值將只會影響資料項的總重值，也就是說只會影響廣播的資料項，而較不會影響演算法的效能。

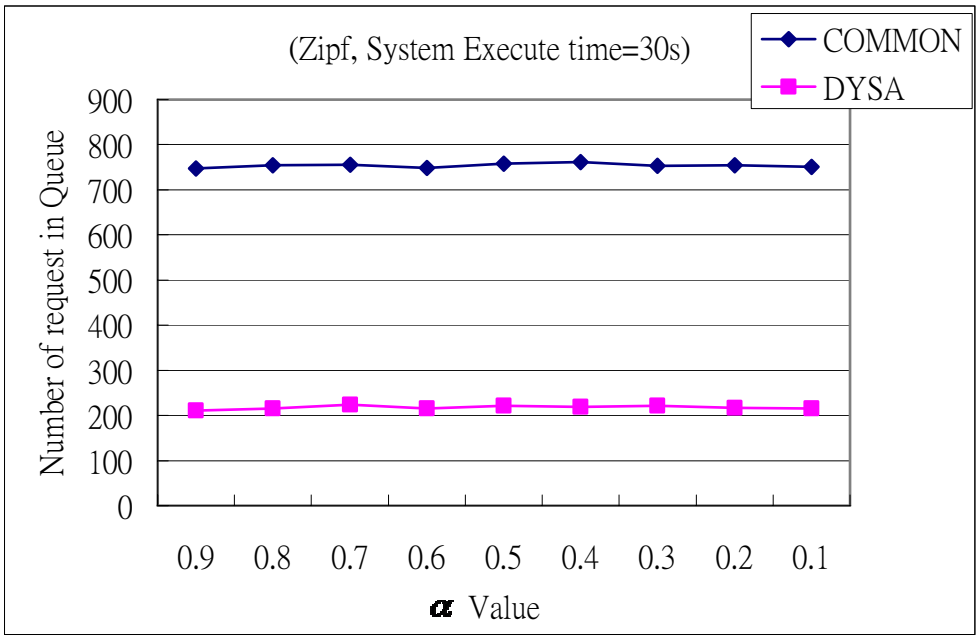

圖 22　實驗結果_Server 端 Request 數量分佈圖(Zipf 分配)

圖 21 裡與圖 22 顯示相同的資訊。圖 21 裡 Request 資料項目的模擬環境使用隨機分配，而圖 22 則是使用 Zipf 分配。使用隨機分配或 Zipf 分配，DYSA 的效能皆遠勝於一般演算法。

比較圖 21 與圖 22，發現使用 Zipf 分配時，兩個演算法(Common 與 DYSA)在 Server 端內 Queue 的 Request 數量皆稍微高於使用隨機分配，因為在使用 Zipf 分配時，系統多花費一些資源在產生資料項為 Zipf 的分配上，但這對演算法而言是較無影響的。

本章節透過完整的模擬實驗以及本文內的時間複雜度分析，分別評估三個不同演算法的效能。上述的數據顯示，在符合動態環境的要求且下，動態排程演算法(DYSA)的效能皆遠勝於一般演算法。

第六章　結論

從無線環境的介紹，到相關文獻的探討，都很清楚在本研究裡描述說明無線網路的架構。透過研究發現兩個在動態環境下的問題；資料項的被點播率會隨時改變及客戶端發出請求的等待時間過久而形成的饑餓現象。本研究裡提出高效能的動態排程演算法(DYSA)，來解決所發現的問題。最後的實驗結果顯示，我們的演算法(DYSA)在動態環境要求下，比一般演算法有更好的運作效能。此外由於動態模擬的環境較接近真實的網路環境，因此表示動態排程演算法除了效能更好外，也能更適用於真實的動態環境中。

本研究在實驗模擬的部份比較排列在佇列裡請求的數量及演算法的處理量，未來的工作可以加入與 FCFS 演算法平均等待時間的比較。雖然 FCFS 可能在排程的部份有較快的速度，因為它不用經過運算，但是若考慮冷門、熱門資料的相關性，及平均等待時間的比較，相信動態排程演算法仍然會有較好的效能。此外，除了應用在單一頻道的廣播之外，亦可將動態排程演算法融入多頻道廣播的應用，以期更能增加適用性。

參考文獻

中文部份

[1] 林士揚, “適用於無線隨意網路下之分散式差異性服務”, 碩士論文, 七月, 2003.

[2] 周雨韻, “以電力能源為基礎的蜂巢式階層管理”, 碩士論文, 五月, 2005.

[3] 夏銘君, “在移動隨意網路下支援多對多群播及廣播服務”, 碩士論文, 七月, 2003

西文部份

[4] Demet Aksoy and Michael Franklin, “R*W: A Scheduling Approach for Large-Scale On-Demand Data Broadcast,” IEEE/ACM Transactions on Networking, vol. 7, no. 6, pp.846-860, 1999.

[5] Tomonori Asano, Hiroyuki Unoki and Hiroaki Higaki, “LBSR:Routing Protocol for MANETs with Unidirectional Links,” IEEE, Proceedings of the 18th International Conference on Advanced Information Networking and

Application, vol. 1, pp.219-224, 2004.

[6] Ruay-Shiung Chang, Wei-Yeh Chen and Yean-Fu Wen, “Hybrid Wireless Network Protocols,” IEEE Transactions on Vehicular Technology, vol. 52, no. 4, pp.1099-1109, 2003.

[7] Ye-In Chang and Wu-Han Hsieh, “An Efficient Scheduling Method for Query-Set-based Broadcasting in Mobile Environments,” IEEE, Proceedings of the 24th International Conference on Distributed Computing Systems Workshops, pp.478-483, 2004.

[8] David B. Johnson and David A. Maltz, “Dynamic Source Routing in Ad Hoc Wireless Networks,” IEEE Transactions on Mobile Computing, vol.353, pp.153-181, 1996.

[9] Chiu-Kuo Liang and Hsi-Shu Wang, “An Ad Hoc On-Demand Routing Protocol with High Packet Delivery Fraction,” IEEE, International Conference on Mobile Ad-hoc and Sensor Systems, pp. 594-596, 2004.

[10] Charles E. Perkins and Elizabeth M. Royer, “Ad Hoc On-demand Distance Vector Routing,” IEEE, Workshop on Mobile Computing

Systems and Applications, pp. 90-100, 1999.

[11] Fumiaki Sato and Tadanori Mizuno, “A Route Reconstruction Method Based on Support Group Concept for Mobile Ad hoc Networks,” IEEE, Proceedings of the 19th International Conference on Advanced Information Networking and Applications, pp. 84-89, 2005.

[12] Navrati Saxena, Kalyan Basu and Sajal K. Das, “Design and Performance Analysis of a Dynamic Hybrid Scheduling Algorithm for Heterogeneous Asymmetric Environments,” IEEE, Proceedings of the 18th International Parallel and Distributed Processing Symposium, pp.223-229, 2004.

[13] Haw-Yun Shin, Jean-Lien C. Wu and Yi-Hsien Wu, “A Packet Scheduling Scheme for Broadband Wireless Networks with Heterogeneous services,” IEEE, Proceedings of the 18th International Conference on Advanced Information Networking and Application, vol. 2, pp.355-358, 2004.

[14] Yi-Yu Su, Shiow-Fen Hwang and Chyi-Ren Dow, “An Efficient Multi-Source Multicast Routing Protocol in Mobile Ad Hoc

Networks," IEEE, International conference on parallel and Distributed Systems, vol. 1, pp.8-14, 2005.

[15] Weiwei Sum, Weibin Shi, Bole Shi and Yijun Yu, "A cost-efficient scheduling algorithm of on-demand broadcasts," Manufactured in The Netherlands, Wireless Networks 9, pp.239–247, 2003.

[16] Juki Wirawan Tantra, Chuan Heng Foh, and Bu Sung Lee, "An Efficient Scheduling Scheme for High Speed IEEE 802.11 WLANs," IEEE, Vehicular Technology Conference, vol. 4, pp.2589-2593, 2003.

[17] Hsin-Wen Wei, Pei-Chi Huang, Hsung-Pin Chang and Wei-Kuan Shih, "Scheduling Real-Time Information in a Broadcast System with Non-Real-Time Information," IEEE, Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp.289-292, 2005.

[18] Xiao Wu and Victor C. S. Lee, "Preemptive Maximum Stretch Optimization Scheduling for Wireless On-Demand Data

Broadcast," IEEE, Proceedings of the International Database Engineering and Applications Symposium, pp.413-418, 2004.

[19] Yiqiong Wu and Guohong Cao, "Stretch-Optimal Scheduling for On-Demand Data Broadcasts," IEEE, Proceedings of the IEEE International Conference on Computer Communications and Networks, pp.500-504, 2001.

[20] Etsuko Yajima, Takahiro Hara, Masahiko Tsukamoto and Shojiro Nishio, "Scheduling and Caching Strategies for Broadcasting Correlated Data," Symposium on Applied Computing archive Proceedings ACM, pp.504-510, 2001.