Flow Redirection and Scatternet Restructuring for Congestion Control over Bluetooth Networks

Ho-Cheng Hsiao and Kwen-Shin Lin
Alethsia University
hocheng.hsiao@msa.hinet.net

Chih-Yung Chang
Tamkang University
cychang@mail.tku.edu.tw

摘要

Bluetooth 是隨建即連的網路系統，若二個需要通訊的 devices 分別位在不同 Piconet，則須為其建立通訊路徑以提供跨 Piconet 之通訊服務。在以往所發表的 routing protocols 中，當繞徑要求發起時，都分別針對 device 的耗電量、流量及 hop 數等參數的最佳化做考量，以達到最長生命週期或最短通訊路徑的目的。然而在分散式的環境執行了一段時間之後，對於位在整個 WPAN (Wireless Personal Area Network)中心或各 routing protocol 最佳決策點上的 device 會造成相當大的負擔。在這篇論文中，我們提出一個流量管理的協定，當頻寬負荷達到門檻後，使用 Piconet 的合併及分解運算來局部改變 scatternet 的結構，將部份工作量導向其他更適當的 devices 上，逐步縮減 hop 數，這樣的做法可以視需要讓經常通訊的 device 慢慢的合併至相同或相鄰的 Piconet 內。由實驗數據顯示，我們提出的做法可以有效的減少 transmission delay，改善 Scatternet 的效能。

關鍵詞：Bluetooth，Role Switch ， Scatternet，Routing，Traffic。

一、簡介

近幾年來，無線通訊技術正以非常快的速度蓬勃發展，並與人類的日常生活息息相關。隨著資訊產業的蓬勃發展以及日新月異的通訊產品相繼問世，以短距離無線通訊技術來連結不同的可攜式裝置，便顯得越來越重要。在眾多的短距離無線通訊技術中，藍牙(Bluetooth)儼然已成為低價位及低功率消耗的代表技術之一[6]。各個 Bluetooth Devices 可以運作在 ISM (Industrial Scientific Medical) 2.4GHz 的開放頻帶上。為了避免受到同一干擾源的影響，Bluetooth 使用 Time Division Duplex(TDD)的方式，將頻段劃分出 79 個頻寬為 1MHz 的頻道，採用每秒變換頻道 1600 次的跳頻展頻技術(Frequency Hopping Spread Spectrum, FHSS)，以避開同一頻道的 collision。在一個 Piconet 中，是由 Master 主動負責安排所有 Devices 資料傳輸的時間長度及順序。

為達到快速形成 Scatternet 的目的，許多研究[7][11]，以自由對頻的方式將所有 device 形成一個 connected Scatternet。但是這種做法僅能使所有 devices 互相連通，忽略了 devices 之間的通訊需求。若有許多必需時常通訊的 devices(如屬於同一個 PAN 或 Group 的 devices)分別加入不同的 Piconet 之中，則在 Scatternet 中必須採用 routing protocol[4][5] 來建立 routing path，以提供 devices 之間的通訊。如此，將會有許多參與 route 的 Bridge devices 為不屬於同一 Piconet 且正在通訊的 devices 代傳資料，使得其消耗頻寬並浪費電量。在這樣的網路架構下，當通訊持續了一段時間之後，將使位於 routing protocol 最佳決策點上的 device，因有多條 routing path 通過而形成網路瓶頸，使得電池提早耗盡，進而減短無線網路的存活時間。這樣的結果將會使 Piconet 形成 bottleneck，網路的傳輸效能亦因此而降低。

目前在 Bluetooth network 的研究中，以 Scatternet Formation 及 Scheduling Protocol 等研究議題較多，然而，大多數的 Scatternet Formation Protocol，其目的主要為形成一個 Connected Network 或一個適合 Routing 的結構[3][8]，並不考慮各個 Device 之間實際的通訊需求，因此前述因結構不良而造成 traffic congestion 的情況將經常發生；而 Scheduling Protocol 則以如何達到公平性[10]、最大頻寬利用率[1]或最小的 delay time[12]為目標，當一個 Piconet 的頻寬不足時，不論採用何種的排程方法，都無法疏解網路上的傳輸瓶頸。在研究[2]中，已經注意到這個問題，將 Piconet 中任二 device 間的流量納入考量，並以代數來建構出一個最佳的 Scatternet，但其計算過程較複雜，當 Network size 太大時，無法在 polynomial time 內完成計算，且其做法為集中式的協定，不適合使用在 Bluetooth 分散式的網路環境中。在這篇論文中，我們提出一個導流協定，以分散式的做法，在不影響 Scatternet 現有服務的情形下，利用 Role Switch 適當的變更部份 Scatternet 結構，使一些需要通訊的 Device 能逐漸調整至相鄰或同一個 Piconet 中，以達到縮減 hop 數及減少網路中 Forwarding ndoe 數目的目的，並將頻寬不足的 Piconet 視其通訊狀況將 Piconet 分裂開來，以加大其可用的傳輸頻寬。如此一來，不但可以節省 Bluetooth device 電量及頻寬的浪費，亦可因 routing path 的縮短，而減少 transmission delay。

本論文的後續章節安排如下，第二章將講述使用 role switch 執行 Piconet 的分裂合併的基本操作，第三章前半段定義了我們用來評估流量的表示法及成為 bottleneck 的 Piconet，後半段則提出我們的流量管理協定。最後，第四章將顯示我們所提出的做法與其他 routing protocol 的執行效能比較，而在第五章中將為我們的做法提出結論。

二、Background and Basic Concepts

在一 Bluetooth device 所形成的 Scatternet 中，當欲通訊的 Devices 落在不同 Piconet 時，由於其 hopping sequence 不同，即使二個 Devices 彼此的距離在通訊範圍內，亦無法直接通訊。若 Routing Protocol 採用 Bluetooth Network 來建立通訊路徑，將可能造成某些位於網路中央的 Piconets 同時有許多條 routing

path 通過，形成傳輸上的瓶頸。有效的導流不但可以疏解瓶頸，更可因參加 route 的 Devices 數量減少而達到省電及節省頻寬佔用的目的。本論文以分散式的方式，由每個 device 平時監控本身的流量，待瓶頸發生時才自動動本協定，在疏解瓶頸的同時，動態解決結構不佳的情形。

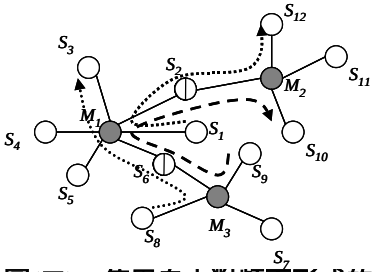
Problem Statement

在本論文中，主要欲解決的問題便是如何以分散式的方法自動偵測網路的瓶頸，並使用適當的技巧疏解網路瓶頸。

若 Scatternet 中已存在三條 routing path，如圖(二)所示。以表(一)的傳輸需求可知，M_1 需花費 0.4Mbps(0.2+0.2)的頻寬傳送 Route *1* 的資料、0.2Mbps(0.1+0.1)幫 Route *2* 傳送資料以及 0.6Mbps(0.3+0.3) 傳送 Route *3* 的資料。根據 Bluetooth Spec.定義的封包大小及傳輸方式，我們得知一個 Piconet 最多只有 1Mbps 的頻寬供 Piconet 內的成員使用，因此當這三條 routing path 同時存在時，M_1 所需提供的頻寬大小為 0.4+0.2+0.6=1.2Mbps，由此可知 M_1 提供的頻寬明顯不足，使得其他代傳的 Device B_1 及 B_2 將因等待 Master polling 而造成傳輸的 delay。

表(一)、在圖二的 Scatternet 內，需要通訊的 Device 及其通訊量。

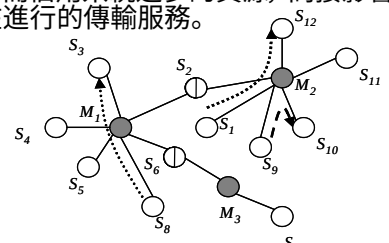
No.	Source	Destination	Require Bandwidth
1	S_1	S_{12}	0.2(Mbps)
2	S_4	S_{10}	0.1(Mbps)
3	S_3	S_7	0.3(Mbps)



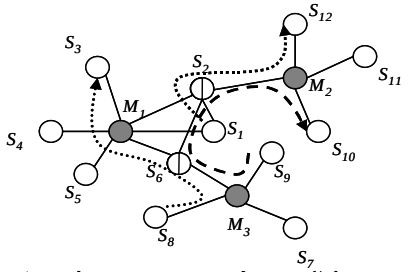
圖(二)、使用自由對頻而形成的 Scatternet 及已建立的 routing path。

本論文將依 Piconet 中流量的瓶頸程度，局部的改變 Scatternet 的結構，以新增及打斷連結的方式，一方面疏解 Master 或 Bridge 頻寬擁擠的情形，另一方面將通訊量較大的 Source 及 Destination Pair 逐漸調整至相近，避免其耗損電量及頻寬。圖(三)為以表(一)的傳輸需求採用合併分解機制，局部改善 Scatternet 結構所建構出較為理想的 Scatternet 結構。圖中的 M_1、S_2 及 S_6 原本是負責代傳資料的 Device，經過結構的調整後，要通訊的每對 Device 皆位在同一 Piconet 內，可省去這三個代傳 Device 的頻寬及電量消耗。比較圖(二)及圖(三)的 Scatternet 結構，我們可知以圖(二)的網路拓樸傳輸資料時，平均每條 routing path 的 hop 數為 4，且共有 5 個 devices 參與資料代傳；而圖(三)中，平均每條 routing path 的 hop 數為 2，且只需 2 個 devices 參與資料的代傳。以節省頻寬及電量的觀點來看，圖(三)的網路拓樸對於表(一)的傳輸需求有較好的效能。

以下，我們將以圖(二)的 topology 為例並提出一套簡單的運算方法，計算自己所屬的 Piconet 是否為一 bottleneck。以圖(二)為例，由於 S_6 擔任二條 routing path 的代傳點，若 M_1 沒有足夠的頻寬 polling S_6，則 S_6 會向 M_1 送出 ***Redirect Request***，當 M_1 收到這個訊息後，會根據 S_6 的資料流向，在 S_6 與 S_2 之間建立一直接的連線，而後續的資料便可由新的 link 傳送而不需經過 M_1。圖(四)所示為 S_6 發起 ***Redirect Request*** 之後的局部結果，若這樣的結構還無法滿足傳輸需求，則 device 會不斷執行我們的協定，直到圖(三)的 topology 形成為止。由於這種做法是當瓶頸發生時才進行資料流向的導流，由偵測到瓶頸點的 device 以 piggyback 的方式，來發起資料導流及重整結構的運算，可避免同一時間因必須執行多個 device 的導流工作而佔用系統過多的資源，間接影響網路上正在進行的傳輸服務。



圖(三)、以圖(二)的 Scatternet 結構為依據，視傳輸需求而調整結構後的 Scatternet。



圖(四)、由 device S_6 向 M_1 發起 *Redirect Request*，經過 Redirection 後的資料流向，S_6與 S_2建立一新的 link，使 hop 數縮短並解決 M_1的瓶頸。

為方便每個 Device 能適時偵測出 bottleneck 的產生，我們以四種狀態來管理此 Device 的情況及其該執行的動作：*Normal state*、*Busy state*、*Redirect state* 及 *Scheduling state*。*Normal state* 為 Master 及 Slave device 在傳送資料或等待 polling 時所處的 state，當某個 slave device 發現頻寬擁塞時，則進入 *Busy state*，並送出 ***Redirect Request***；而 Master 收到 ***Redirect Request*** 之後，便進入 *Redirect state*，負責協調該 device 所要連結的對象，這時送出 request 的 device 亦進入 *Redirect state*，並與指定的 device 連線。當完成連線後，所有連線更改的 device 均進入 *Scheduling state* 準備與 Master 執行排程的協調，而 Master 在收到連線完成的訊息之後，亦會進入 *Scheduling state* 規劃合併或分裂之後的工作排程，在第三章中，我們將闡述如何用這些方法，來解決在一個不適當的 Scatternet 結構中通訊所引發的問題。

三、流量管理協定

在本章中我們將敘述 Piconet 合併及分解的準則及其所用的技術，並定義後續章節所使用的符號，隨後再敘述我們所提出的流量管理協定，最後我們以一個例子來說明流量管理協定的執行過程。

為了加快連線的建立，我們所提出的導流協定將採用 Role Switch 為基本運算，省去 inquery/inquery scan 的步驟，直接以 page/page scan 的溝通方式來新建 link，以下我們便詳述 Role Switch 的執行過程及其一般的應用。

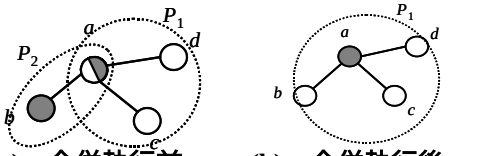
如前所述，為使整個 Bluetooth network 較

易達到 connected 的目的，Bluetooth device 在建立連線時大多使用自由對頻的方式[7][11]，因此無法保証都能以適當的角色與其他的 device 相連。但在 Bluetooth Network 中，某些特定的 device 必須根據自己提供的服務或用途來決定最適當的角色。Role switch 是由 Link Management Protocol layer 所發起的動作，當 Slave 想要發起 role switch 的請求時，便先向 Master 發出 LMP_slot_offset 訊息，告知二個 device 時槽間的 offset，接著送出 LMP_switch_req 訊息；若 Master 不同意 Slave 的請求，則回傳 LMP_not_accepted 並結束這次的程序，否則回傳 LMP_accepted，隨即進行 BD_ADDR 等訊息交換的動作，Slave 將 BD_ADDR 傳給 Master 後，Master 便根據此 BD_ADDR 計算出其跳頻序列，此後 Master 便依照舊 Slave 的 hopping sequence 跳頻，並在往後的通訊中得到舊 Slave 所給予的 AM_ADDR，自此便完成了 role switch 的操作。

Role Switch 的基本運算不僅可改變 device 的角色，亦可應用於局部改變 Scatternet 的結構。以下，將討論如何利用 Role Switch 的運算來達到 Piconet 合併、分解及接管等局部改變 Scatternet 結構的基本運算。

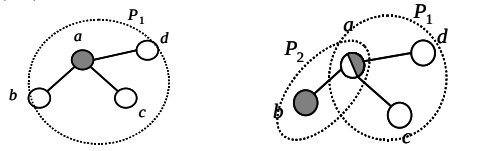
(1) Piconet 的合併

如圖(六)(a)所示， device *b* 可向 device *a* 發出 Role Switch 請求，變換彼此的身份以達成加入 Piconet P_2 並成為 Slave 的目的，改變後的結構如圖(六)(b)所示，使原本二 Piconets 合併為一個 Piconet。這將可利用來減少 Piconet 數量，並間接減少 routing path 的長度。



(a)、合併執行前　　(b)、合併執行後

圖(六)、執行 Piconet 合併之 Role Switch 運算



(a)、分裂執行前　　(b)、分裂執行後

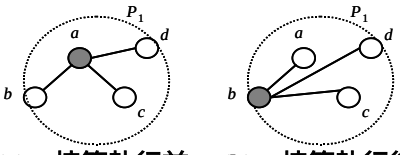
圖(七)、執行 Piconet 分裂之 Role Switch 運算

(2) Piconet 的分裂

如圖(七)(a)所示，device *b* 若想自 Piconet P_1 中分裂出去，形成一個新的 Piconet P_2，並扮演 Master 的角色，而由 device *a* 扮演 Slave 時，device *b* 就可以向 device *a* 發出 Role Switch 請求以達成此一目的。在 Role Switch 完成後其 Network 結構如圖(七)(b)所示，device *b* 成為 P_2 中的 Master，而 device *a* 則在 P_2 中扮演 Slave，在 P_1 中扮演 Master，成為同時兼具 Master 及 Slave 兩種身分的 M/S Bridge。使原 Piconet P_1 分裂成兩個 Piconet P_1 及 P_2。

(3) Piconet 的接管

如圖(八)(a)所示、若 Piconet P_1 中的 Slave *b* 欲取代原本的 Master *a* 而成為新的 Master 時，device *b* 可向 device *a* 發出 Role Switch 請求，當 device *a* 收到請求後，便要求處於 P_1 中其餘的 Slave *c* 及 *d* 進入 Page Scan State，並將它們的 BD_ADDR 及 Clock 傳給 device *b*。而 device *b* 便可利用此資訊進入 Page State 與處於 Page Scan State 的 device *c* 及 *d* 達成連結，並建構成一新的 Piconet P_2，如圖(八)(b)所示。而此一新的 Piconet P_2 包含 Master *b* 及其管理的 Slave *a*、*c* 及 *d*。由於是在已形成的 Piconet P_1 中透過 device *b* 來取代 device *a* 所扮演的 Master 身分，並接管其所有的 Slaves，這種 Role Switch 的動作我們稱其為"接管"。利用這個運算，我們可以讓通訊量較大的 device 成為 Piconet 的 Master，減少資料代傳的次數，並節省 Piconet 的頻寬。



(a)、接管執行前　　(b)、接管執行後

圖(八)、執行 Piconet 接管之 Role Switch 運算

以下所述為我們所提出的協定其執行合併分解的基本準則：

準則一：當一個 Piconet 流量大時，我們可以利用 Bluetooth 各自跳頻的優點，將一個 Piconet 分裂為二平行傳輸，以疏解 Master 的瓶頸。

準則二：若 Piconet 數分裂的太多，hopping sequence 因衝撞而增加資料傳輸碰撞的機會。收集流量較小的 Piconet，使其與另外的 Piconet 合併，將可減少 Piconet 的數量，藉此可減少 route 的長度，亦可平衡因準則一所新增的 Piconet 數。

3.1 流量測量方法(Flow Measurement)

為了方便及清楚描述導流管理協定，我們定義以下的符號：

Definition: *Scheduling Window* (*w*) 及 n_w

Scheduling window 乃由數個連續的 time slots 所組成，為 Master 執行 Scheduling 的最小單位。

Definition: *Numbers of neighbor* (*N (k)*)

*N(k)*為 device *k* 的 one hop 鄰居的集合，亦即與 device *k* 有建立直接連線的 device 集合。

Definition: *Numbers of neighbor* (N^2 *(k)*)

$N^2(k)$為 device *k* two hops 內的鄰居集合，每個 device 可藉由和其他 device 交換 *N(k)*的集合以獲得 $N^2(k)$的資訊。

Definition: *Packet Arrival Rate* (*a*)

Packet arrival rate 為每個 device 每秒收到的資料量，單位為 bytes/sec。

Definition: Bandwidth Requirement (r slots/w)

根據 device 的 *Packet arrival rate* (*a*)及其所使用的封包類型，將可計算出在一個 scheduling window 內，至少需要得到 Master 多少個 Slots 來服務。*r* 可由以下式子得知：

$$r = \frac{a}{p * n_w} \qquad (1)$$

其中 *p* 所代表的是平均每個 time slot 可傳送的資料量。為了表示在每個 scheduling window 內的 bandwidth requirement，我們以 r^j 來表示在第 *j* 個 scheduling window 內的 bandwidth requirement。

Definition: *Service Rate* (*s slots/w*)

對 Slave 及 Bridge 而言，在 scheduling window 內，受到 Master 服務的 slot 個數。以 s^j 表示在第 *j* 個 polling window 的 service rate。

Definition: *Buffered Data Length for device k* (b_k)

在某個時間 *t* 時，每一個 device 將資料放在 queue 中，準備傳給對另一個 device *k* 的 buffer 長度。我們以 b_k^j 表示在第 *j* 個 scheduling window 結束時，放在要送給 device *k* 的 buffer 其資料長度。

如下所述，每個 device 可以在第 *j* 個 scheduling window 結束後，以目前的 packet arrival rate 及 service rate，預測在第 *j+1* 個 scheduling window 結束時，累積準備送給 device *k* 的 queue 其資料長度：

$$b_k^{j+1} = b_k^j + p*(r^j - s^j)*n_w \qquad (2)$$

Definition: *Undirected Flow Matrix* (*F*)
Master 及 Bridge node 需記錄 scheduling window 內，Piconet 內任兩 device 間的通訊時槽數，我們以 Matrix *F(i,j)*來表示一 Piconet 中 device S_i 與 device S_j 在一個 scheduling window 的資料流量。

Definition: *Redirect Request Initiator* (*RRI*)
進入 *Busy state* 時，向服務的 Master 或 Bridge 發出 ***Redirect Request*** 的一個 device。

Definition: *Corresponding Node* (*C*)
我們將這些受到 ***Flow Redirect Scheme*** 影響的 devices 稱為 corresponding node 並以 *C* 表示 corresponding node 的集合。

Definition: Link utilization (u_k)及 Piconet utilization (u_{Pi})
u_k 為 Master 與 Slave *k* 之間的 link 利用率，若 u_k 的值小於門檻值，則 Master 將會考慮把與 device *k* 的 link 從網路中移除；而 u_{pi} 則代表 Master *i* 所在的 Piconet 頻寬利用率，若其小於某一門檻值，則我們將把此 Piconet 中的成員，合併至與其他 Piconet。

Definition: *Window scheduling table* (T_w)
Window scheduling table 是一個一維陣列，陣列大小為 scheduling window size，陣列中的值以 $T_w(i)$ 表示，*i* 表示在 scheduling window 的第 *i* 個 slot，如表(三)所示。Master 的 scheduling table 內容記載分配到這個時槽的 bridge 其 AM_ADDR，若 $T_w(i)$為 0，則表示此時槽不屬於任何一個 bridge，Master 可在此時槽 scheduling 任一 Slave；而 bridge 的 scheduling table 內容則記載分配到這個時槽的 Master ID，若 $T_w(i)$為 0，則表示 bridge 在此時槽內不需與任何 Master 溝通，可進入省電狀態。

表(三)、Window scheduling table

$T_w(1)$	$T_w(2)$	$T_w(3)$	$T_w(4)$	$T_w(5)$	$T_w(6)$	…	$T_w(w)$
0	0	0	0	0	0	…	0

3.2 流量導向策略(Flow Redirect Scheme)

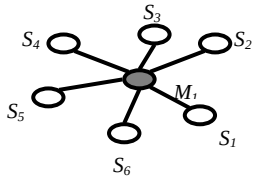
在介紹了我們所需要的定義與符號後，以下我們將以例子說明流量導向策略，其每個 device 將執行的正式協定將詳述其後。我們的協定中分為二個 phase：**Splitting Phase** 及 **Merging Phase**。以下我們便分別對這二個 phase 詳細描述其細節。

A. Splitting Phase

為了簡化描述我們的流量管理協定，我們以圖(二)局部的 topology 為例，先討論在一個 Piconet 內的結構變化，其 topology 如圖(九)所示，Master M_1 服務六個 Slaves，其在上一個 scheduling window 中任兩 device 的資料流量 *F* 矩陣如表(四)所示。由表中可知，M_1 與其他 device 之間的通訊並不頻繁，主要是其他 Slave 之間需做資料交換，因此 M_1 需花費二倍的頻寬將資料代傳至其他 device，而使 M_1 形成 bottleneck。

表(四)、圖(九)之 M_1 所記錄的 *F* 矩陣

	M_1	S_1	S_2	S_3	S_4	S_5	S_6
M_1	0	3	3	6	2	0	4
S_1	0	0	20	0	0	0	0
S_2	0	0	0	12	3	6	11
S_3	0	0	0	0	15	17	3
S_4	0	0	0	0	0	4	3
S_5	0	0	0	0	0	0	0
S_6	0	0	0	0	0	0	0



圖(九)、一個擁有六個 Slave 的 Piconet。

在我們提出的協定中，每個 device 在連線建立時，首先會進入 *Normal state*，並在每個 scheduling window 內，記錄 Master 的 service rate，之後 device 以公式(2)來預測在下一個 scheduling window 結束後，累積在 device 本身 buffer 內的資料量，若累積的資料量與 buffer size 的比例大於某一門檻值，則該 device 會進入 *Busy state*，通知服務該 device 的 Master 或 Bridge 開始執行 Flow Redirect，因此若 Master 或 Bridge 所提供的 service rate 無法滿足 device 目前的傳輸需求，該 device 會在下次與 Master 或 Bridge 交換資料時，以 piggyback 的方式，對 Master 或 Bridge 發出 ***Redirect Request***。

以下我們便定義在 **Splitting phase** 中所用到的門檻值：

Definition: *Buffer size threshold* (α)
為了決定 ***Redirect Request*** 發起的時機，我們定義一個門檻值，其表示 buffer 內的資料與 buffer size 的比例，當 buffer 內的資料達到這個門檻值時，device 便會進入 *Busy state*，並發出 ***Redirect Request***。

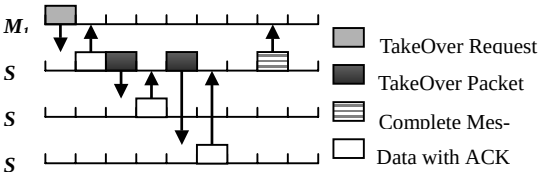
Definition: *Redirection threshold* (β)
為了避免 device 經常的發起 ***Redirect Request***，我們定義一個 redirection threshold，讓與 *RRI* 通訊量大於(最大通訊量*β)的 device 成為 corresponding node。

以圖(九)及表(四)為例，假設 S_2 的 buffer size 為 20kb，目前有 16kb 的資料位於 buffer 中，bandwidth requirement 為 20 *slots/w*，service rate 為 18 *slots/w*，而每秒有 4 個 scheduling window，且發起 ***Flow Redirect Scheme*** 的 threshold α為 0.9，此時 S_3 以公式(2)計算出在下一個 scheduling window 結束後，buffer 中將累積 16+0.339*(20-15) *4=18.712kb 的資料，由於 18.712kb/20kb≥0.9，已大於發起 Flow Redirect 的門檻值。因此 S_3 會進入 *Busy state*，當下次收到 M_1 的 polling packet 時，便會對 M_1 發出 ***Redirect Request***。

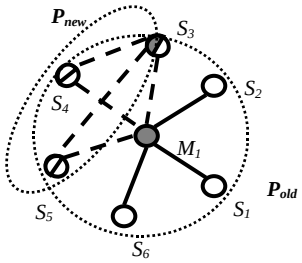
當 M_1 收到 S_3 發出的 ***Redirect Request*** 後，便進入 *Redirect state*，並根據本身所記錄的 *F* 矩陣，找出與 S_3 通訊量最大的 device，並以門檻值β為標準，將通訊量大於這個比例的 device 一起與 S_3 建立 link。若門檻值β 為 0.8，則當 device 與 *RRI* 的通訊量大於最大通訊量的 0.8 倍時，該 device 便會被設定為 corresponding node，這些 corresponding node 與 S_3 共同將 traffic 轉移至新的 Piconet。以表(四)為例，目前與 S_3 通訊中的 device，以 S_2 與 S_5 之間的通訊量 17 為最大，M_1 便查詢本身所維護的 *F* 矩陣，將 $F(S_3,*)/17 \geq 0.8$ 及 $F(*,S_3)/17 \geq 0.8$ 的 devices 選定為 corresponding node。以表(四)可知，由於 $F(S_3,S_4)/17=15/17>0.8$，且 $F(S_3,S_5)/17=17/17>0.8$，因此 M_1 將會把 S_4 及 S_5 設定為 corresponding node，隨後便將其資料流向直接由 M_1 重新導向至 S_3。這樣的做法可以將與 S_3 通訊頻繁的 devices，一次從所處的 Piconet 中移開其流量，除增加其可用頻寬外，還可避免其他欲和 S_3 通訊的 device 又發出 ***Redirect Request***，如此便可兼具減少 control overhead，及增加系統的效能的好處。

為了節省 Page/Page Scan 冗長的連線方式，我們以 slot leasing[9]的方法，S_3 在 Master 所使用的偶數時槽，對先前成為 corresponding

node，分別送出 *TakeOver Packet*，其中包括 S_3本身的BD_ADDR、clock及欲成立新Piconet的各成員其 AM_ADDR，如此在 *C* 內的devices，將會參與二個以上的 Piconets。圖(十)所示為使用 slot leasing 的方式與其他 devices 通訊的流程圖。



圖(十)、Slave 使用 slot leasing 機制在偶數時槽與其他 Slaves 通訊的流程。



圖(十一)、針對圖(九)所示的 Piconet 而言，device S_3 執行 Take Over 後的 Scatternet。

為了方便說明結構改變後的資料導向，我們定義以下符號：

Definition: *pre (S)*
對一條 route 而言，*pre(S)*表示在 device *S* 的 routing table 中，位於 previous hop 的 devices。
Definition: *next (S)*
對一條 route 而言，*next(S)*表示在 device *S* 的 routing table 中，位於其 next hop 的 devices。

當 Master M_1 收到 *complete message* 後，便進入 *Scheduling state*，開始協調與 *RRI* 及 corresponding nodes 之間切換 Piconet 的原則。Master M_1 會根據 routing table 的資訊，將與 *RRI* 有關的資料流，由本身的 *F* 矩陣中移除，並通知 *pre*(M_1) (*next*(M_1))中需要經由 Master M_1 送至 *next*(M_1) (*pre*(M_1))的 device，將資料流直接送往 next(M_1)(*pre*(M_1))，使 *RRI* 與 *C* 可藉由新的 link 直接通訊。表(五)所示為 Master M_1 分別設定 $F(M_1,S_3)=0$、$F(M_1,S_4)=0$ 及 $F(M_1,S_5)=0$，新的 *F* 矩陣狀態。

表(五)、經過接管之後 M_1 新的 *F* 矩陣值

	M_1	S_1	S_2	S_3	S_4	S_5	S_6
M_1	0	3	3	6	2	0	4
S_1	0	0	20	0	0	0	0
S_2	0	0	0	12	3	6	11
S_3	0	0	0	0	0	0	3
S_4	0	0	0	0	0	0	3
S_5	0	0	0	0	0	0	0
S_6	0	0	0	0	0	0	0

在 *Scheduling state* 中，Master 會對 *RRI* 及 *C* 中的 device 執行排程。Master 會以新 *F* 矩陣的值，計算出與 *RRI* 及 *C* 中 device 的通訊時間。下式表示 Master 計算 *RRI* 及 *C* 中各 device 分與其他 devices 所需的通訊量：

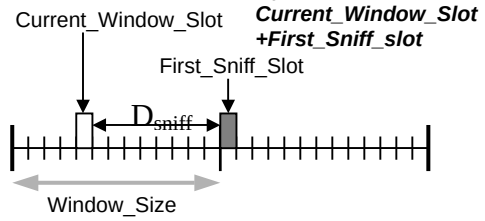
$$\sum_{i=0}^{N(M_{old})} F(S_k, S_i) + \sum_{j=0}^{N(M_{old})} F(S_j, S_k) \quad (4)$$

其中 S_k 為 *RRI* 及 *C* 中各 device。我們將其所需的通訊量視為所需頻寬，亦即 time slots 個數。

為了讓每個 device 能週期性的與 M_1 溝通，因此在與 *RRI* 的規劃中，我們採用 sniff mode 來使 *RRI* 及 *C* 中各 device 在不同的二個 Piconet 中切換。Sniff mode 為 Bluetooth spec. 所制定的三種省電模式之一，其有三個重要的參數：T_{sniff}、D_{sniff} 及 $N_{sniff\_attempt}$。當 Master 計算出各 devices 所需的 time slots 數後，便設定各 device $N_{sniff\_attempt}$ 的值，並根據其維護的 window scheduling table (T_w)，找出一段可用的 time interval，將其第一個時槽設定為 First_Sniff_Slot，亦即排定給該 device 的第一個可用時槽，再根據 Master 目前在 scheduling window 中所處的 time slot，計算出 D_{sniff} 值，再以 Window_Size 設定 T_{sniff} 的值，並逐一通知 *RRI* 及其 *C* 中各成員進入 *sniff mode*，此後這些分裂出去的 *RRI* 及 *C* 中各成員只要在和 Master 所約定的 Sniff_Slot 切換回原 Piconet，等待 Master polling 即可。由公式(4)可知，S_3 與原先參與的 Piconet 中其他 devices 的 traffic 量為 6+12+3=21 個 time slots/*w*，因此 Master 將 S_3 的 $N_{sniff\_attempt}$ 設定為 21，並依此方式分別計算出其他 S_4 及 S_5 的 $N_{sniff\_attempt}$ 分別為 2+3+3=8 及 6。之後 Master 會根據 window scheduling table 中，找出一可用的時槽區段，並將該區段的第一個時槽定為 First_sniff_slot，並以下列公式配置一段時間給各 Bridge。

$$T_w(\textit{First\_sniff\_slot of } C_i : \textit{First\_sniff\_slot of } C_i + T_{C_i} - 1) \leftarrow M_1 \textit{ 與 } C_i \textit{ 通訊} \quad (5)$$

其中 C_i 表示 *C* 集合內的成員，即 corresponding node i。以本例來說，Master 可查詢其 T_w，得知 S_3 第一個可用的時間區段，其第一個可用時槽為 slot 0，而 S_3 以公式(4)計算得知其需 21 個 time slot 與 P_{old} 內的 devices 通訊，因此 Master 會將第 1 個至第 21 個時槽配置給 S_3，其以 $T_w(1{:}21) \leftarrow S_3$ 表示。此後 Master 和 bridge 便在約定的時間進行資料交換。圖(十二)所示為 D_{sniff} 的計算方式。



圖(十二)、D_{sniff} 的計算方式。

由於目前 T_w 中所有的 time slot 並無排定任一 device，因此 Master 由 window scheduling table 可知 S_3 的 First_Sniff_Slot 之值為 1，並將 21 個 time slot 指定給 S_3。而對於其他的 *C* 中的 device，Master 亦使用相同的方式，各別計算出通訊所需的時間 $N_{sniff\_attempt}$ 及 Sniff 週期 T_{sniff}：

$$T_w(1{:}21) \leftarrow M_1 \textit{ 與 } S_3 \textit{ 通訊}$$
$$T_w(22{:}29) \leftarrow M_1 \textit{ 與 } S_4 \textit{ 通訊}$$
$$T_w(30{:}35) \leftarrow M_1 \textit{ 與 } S_5 \textit{ 通訊}$$

若我們使用的 Window_Size 為 400 slots，且目前已執行至 scheduling window 中第 300 個 time slot，則 Master 可計算出 S_3 的 D_{sniff} 的值為 400-300+1=101，且依相同的方法計算出 S_4 及 S_5 的 D_{sniff} 參數，並分別傳送給 S_4 及 S_5。由於 bridge 同時參與二個以上的 Piconet，若此時 Master 送達的 sniff 參數與其他 Picoent 的排程相衝突時，bridge 會根據本身的 scheduling table，仿照 Master 的方法，計算出新的 D_{sniff} 參數，傳送給 Master，當 Master 收到後，若 Master 可接收該參數，則將原先配

置給該 bridge 的時槽釋放，改為配置由 bridge 所指定的時槽；若 Master 無法接受，則 Master 將原先配置給該 bridge 的時槽設為-1，表示該時槽無法使用，再由剩下的時槽重新設定 D_{sniff} 參數，如此由 Master 及 Bridge 互相協調，直到找合適的時間區段。

B. Merging Phase

在 **Splitting Phase** 中，當一個 Piconet 成為瓶頸時，是以 *RRI* 為新的 Master，並和其他 *C* 中的 device 建立連線，使其能直接通訊，而如此的做法會使 *RRI* 及 *C* 中的 device 同時參與二個 Piconet，因此必須經常切換 Piconet 而造成 guard time 的浪費，Master 亦必須為 *RRI* 及 *C* 中的 device 保留頻寬以便將資料與新成立的 Piconet 交換，另外也會使得 Scatternet 存在太多的 Piconet，增加資料碰撞的機率，因此當 link 的 utilization 降低到一定的門檻值後，將觸發 merging event，執行 **Merging Phase** 的協定。我們在 **Merging Phase** 中，針對這二種狀況，提出 *Cut Rule* 及 *Merge Rule* 以改進上述缺點，以下我們便分別對這二種情況加以說明。

***Cut Rule*：**

若新建連線後，大部份的流量可以由新的 link 直接送至目的 device，則原本的 link 可以考慮將其中斷，而將所有流量全部轉移至新的 link 傳送，如此以減少 device 所參與的 piconet 數，並避免因 guard time 及 Master 保留時槽而造成頻寬浪費。 以下為了方便說明 Cut Rule 的發起時機，我們定義一個門檻值：

Definition: *Utilization threshold (γ)*

為避免 Scatternet 中，存在太多頻寬利用率低的 link。我們定義一個 threshold，將頻寬利用率低於門檻值的 link 切斷，以減少 device 所參與的 Piconet 數，減少切換的 guard time。

為了有效利用時槽及減少 guard time，Master 在每個 scheduling window 結束後會統計每條 link 的頻寬利用率，若頻寬利用率小於系統定義的 utilization threshold γ，則 Master 將考慮移除此 link。在 Master M_{old} 中，S_i 與原 Master M_{old} 的頻寬利用率我們可以下的式子測得：

$$u_{S_i} = \sum_{k=1}^{N(M_{old})} (F(S_i, S_k) + F(S_k, S_i)) \quad (6)$$

另外，為了防止連線中斷後形成 disconnected networks，device S_k 平時必須和其他 device S_i 交換彼此的鄰居資訊 $N(S_i)$，當 device S_k 收到由 device S_i 送來的鄰居資訊時，會將其加入 $N^2(S_k)$，如此每個 device 便可得到 two hops 的鄰居資訊，而當 Master 想要切斷與 S_k 的 link 時，下列二個條件必需同時成立：

$$u_{S_k} < \gamma ;$$

(b) $\exists S_i \in N^2(M_{old}) \ni N(S_i) \cap N(M_{old}) \neq \phi$。

其中 u_{S_k} 代表 Master 與 S_k 之間 link 的頻寬利用率，而γ為我們定義的一個門檻值。以下我們將分別對這二個條件加以敘述。當 Master 準備將一條 link 切斷時，會根據規則*(b)*，判斷欲切斷 link 的 device S_i，是否和 Master 有相同鄰居。若有，則表示 link 切斷後，還存在另一條路徑可到達該 device；若不存在相同的鄰居，則代表當 link 切斷後，將可能使 Scatternet 形成 disconnected network。以圖(十一)的 topology 為例，M_1、S_3、S_4 及 S_5 的 *N(k)*資訊分別為：

$N(M_1) \leftarrow \{S1, S2, S3, S4, S5, S6\}, N(S_3) \leftarrow \{M_1, S_4, S_5, N(M_1)\}, N(S_4) \leftarrow \{M_1, S_3, S_5\}, N(S_5) \leftarrow \{M_1, S_3, S_4\}$

而當Slave S_k與Master交換資訊之後，Slave可在$N^2(S_k)$中得知整個Piconet內的device資訊，對S_3而言，其$N^2(S_3)$內容為$\{M_1, S_4, S_5, N(M_1)\}$。若 Master M_1欲切斷S_4及S_5之間的連線時，M_1可藉由本身所維護的$N^2(M_1)$得知，S_4及S_5都與M_1存在一個共同鄰居S_3，因此以規則*(b)*判斷，M_1 可切斷與 S_4 及 S_5 之間的 link，而不會導致 disconnected network的發生。

當 link 切斷後，鄰居狀態有改變的 device S_k 首先必須更新自己的鄰居資訊 $N(S_k)$，再將 $N(S_k)$與其鄰居 device S_i 交換，當 device S_i 收到更新的 $N(S_k)$時，會同時更新其二步鄰居 $N^2(S_i)$ 的資訊，在這裡我們利用平常維護鄰居資訊的方式，來減少 *Cut Rule* 的資料交換時間，且 Master 亦會更新其所維護的 *F* 矩陣，表(六)為 Master M_1 更新後的 *F* 矩陣。

表(六)、修剪 link 之後，M_1所維護的 *F* 矩陣。

	M_1	S_1	S_2	S_3	S_6
M_1	0	3	3	6	4
S_1	0	0	0	0	0
S_2	0	0	0	0	0
S_3	0	0	0	0	3
S_6	0	0	0	0	0

***Merge Rule*：**

當 Master 計算本身所處 Piconet 的 u_{p_k} 小於門檻值時，Master 會採用 Merge Rule 將 Piconet 與其他 Piconet 合併。 為了說明 Merge Rule 的發起時機，我們定義一個符號：

Definition: *Merge threshold (δ)*

為了適時的減少 Scatternet 中 Piconet 的數量，因此每個 Master 都必須統計在每個 scheduling window 中總頻寬的使用量。若小於 *Merge threshold* δ，則我們將考慮將此 Piconet 合併至其他 Piconet 中。

為了適當的減少 Scatternet 中的 Piconet 數量，我們希望每個 Piconet 都能有最佳的利用率，因此每個 Master 都必須統計在每個 scheduling window 時段中的總頻寬利用率。我們可以下列公式計算：

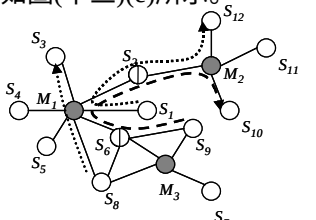
$$u_{p_k} = \sum_{i=1}^{N(M_k)} \sum_{j=1}^{N(M_k)} F(S_i, S_j) \quad (7)$$

當一個 Piconet 的 u_{p_k} 值小於我們定義的 *merge threshold* δ時，Master 會向其所有 Bridge nodes 發出 *utilization investigate* 的 message，請 Bridge nodes 詢問其他 Piconet 中的 Master，其所剩餘的頻寬，並將結果回報給發出 message 的 Master。回報後的其他各 Piconet 頻寬剩餘量，Master 會將其記錄於 *capacity table*。

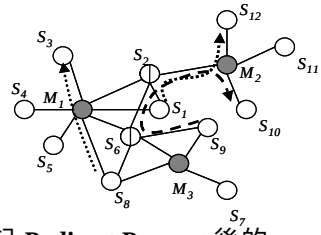
當選定欲合併的 Piconet 後，Master 便會向連接該 Piconet 的 birdge node 送出 ***Merge Request***，當 bridge 收到後，會為二個 Piconet 中的 Master M_i 及 M_j 建立一直接的連線，爾後 Master M_i 便向 Master M_j 送出 *Merge_List*，其內容包括屬於該 Piconet 的 devices 資訊，使 Master M_i 能以 Role Switch 運算接管其下成員，成為新的 Master。

最後，我們以圖(二)的網路拓樸為例，說明我們協定的完整運作方式。在圖(二)中，由於 Master M_3 有二條 path 通過，因此若在傳輸時因 bridge 切換或 Master scheduling 不佳，而使得 M_3 無法即時的將資料傳送至其他 Piconet 時，S_8 及 S_9 便將對 M_3 發出 ***Redirect Request***，M_3 將建立 S_8 至 S_6 及 S_9 至 S_6 的連線，將資料流導向新的連結，其網路拓樸如圖(十三)(a)所

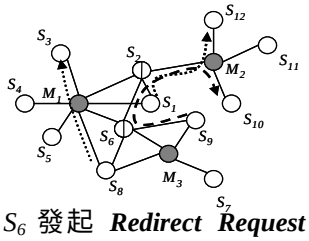
示。而由於 bridge S_6 參與了多個 Piconet，因此頻寬容易不足，加上又負擔了二條 routing path 的傳輸, 因此 S_8 隨後會向 S_6 發出 ***Redirect Request***，而將資料導向 M_1，如圖(十三)(b)所示。而由於 M_1 負擔了三條 routing path 的傳輸，因此亦可能發生瓶頸，此時可能由 S_6 及 S_1 向 M_1 發起 ***Redirect Request***，將資料直接傳送至 S_2，如圖(十三)(c)。最後，由於 bridge S_2 參與多個 Piconet，加上代傳了大量的資料，因此 S_1 及 S_6 隨後可能又將發起 ***Redirect Request***，將資料直接傳送至 M_2，如圖(十三)(d)所示。而在之前所建立的舊 link, 可以 **Merging Phase** 中的 *Cut Rule* 予以切斷；而以 S_9 及 S_6 組成的 Piconet，亦可以 **Merging Phase** 中的 *Merge Rule* 予以合併至 M_2 所主持的 Piconet，其最後結果如圖(十三)(e)所示。



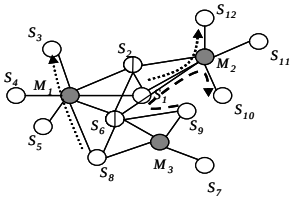
(a)由 S_8 及 S_9 發起 ***Redirect Request*** 後的 topology。



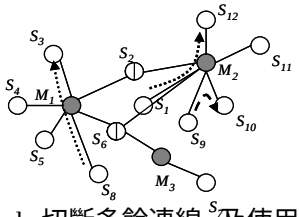
(b)由 S_6 發起 ***Redirect Request*** 後的 topology。



(c)由 S_1 及 S_6 發起 ***Redirect Request*** 後的 topology。



(d)S_1 及 S_6 再次發起 ***Redirect Request*** 後的 topology。



(e)使用 Cut Rule 切斷多餘連線, 及使用 Merge Rule 合併 Piconet 之後的結果。

圖(十三)、Flow Control Protocol 的運作過程。

四、實驗

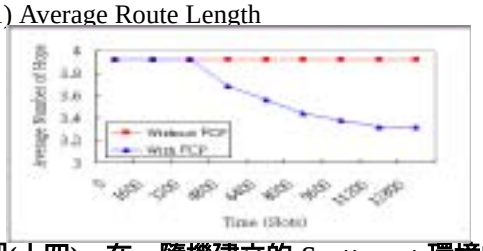
我們以一個隨機建構的 Connected Bluetooth Scatternet 的環境來評估本論文所提出 FCP(Flow Control Protocol)演算法的效能。在我們的實驗中，我們以隨機的方式，在 30 個 Bluetooth 中選擇 10 對的 Source-Destination pair, 每個通訊對以 RVM[6]的演算法建立通訊路徑，以評估其效能。我們模擬環境中其他的參數設定如表(九)所示：

表(九)、模擬實驗所使用之參數。

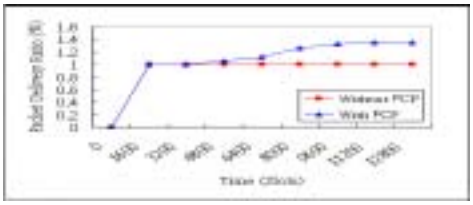
參數	值
Scheduling Window size	400 (slots)
Buffer size threshold (α)	0.9
Redirection threshold (β)	0.8
Utilization threshold (γ)	15 (slots)
Merge threshold (δ)	100 (slots)
Buffer size	100 (kbit)

根據以上的模擬環境，我們以下列事項來說明 FCP 所帶來的績效：

(1) Average Route Length



圖(十四)、在一隨機建立的 Scatternet 環境中配置 30 個 Blutooth Device，其 routing path 長度和時間的關係圖。



圖(十五)、在一隨機建立的 10 條 routing path 中，其 packet delivery ratio 和時間的關係圖。

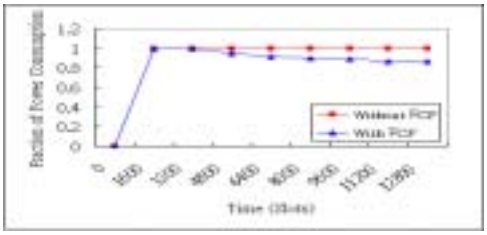
圖(十四)顯示，RVM 所建構的 Routing Length 不會隨時間改變；而 FCP 則會視傳輸需求，繞開瓶頸節點，以減少 hop 數。其改變程度不明顯的關鍵在於，由於 device 只有在 buffered data 到一定的比例時才會發起 FCP，因此若產生的 10 條 routing path 並無瓶頸節點發生，則我們的演算法並不會起作用。

(2) Packet Delivery Ratio

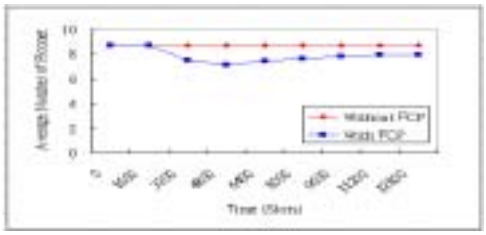
承圖(十四)之結果，當 10 條 routing path 同時在通訊時，受限於 Piconet 頻寬的影響，沒有採用 FCP 演算法的 Scatternet，會因結構不良而產生瓶頸節點，傳輸效能因此而受限；而採用 FCP 演算法的 Scatternet，因其遇到瓶頸節點時，會發起 Flow Redirect Scheme，使 path 長度縮短並避開瓶頸節點，使得單位時間內的 packet delivery ratio 能慢慢提升。如圖(十五)所示，由於採用 FCP 之 Scatternet 能夠適時避開瓶頸節點，並縮短 hop 數，因此其傳輸效能較隨意建立的 Scatternet 略高。

(3) Degree of Power Consumption

如圖(十六)所示，我們可得知不採用 FCP 的 Scatternet，其電量的消耗均維持一定量的比率，而採用 FCP 做法的 Scatternet 其消耗的電量約為不採用 FCP 方法的 0.88 倍，較為省電。



圖(十六)、在一由 30 個 Bluetooth device 組成的 Scatternet 環境中，隨機產生 10 條 Routing Path，其有無採用 FCP 之平均耗電量的比率關係圖。



圖(十七)、在一存在 10 條 routing path 的 Scatternet 中，我們提出的協定對於 Piconet 數的影響。

(4) Number of Piconet

如圖(十七)所示，由於受到 **Merging Phase** 中的 Merge Rule 的影響，在 Scatternet 一開始運作時，會將沒有路徑通過，亦即流量較少的 Piconet 與其他 Piconet 合併，因此在一開始時 Piconet 數快速減少；隨後又因 **Splitting Phase** 的作用，將成為瓶頸的 Piconet 分裂以加大頻寬，因此 Piconet 數隨後又稍微增加。由圖(十七)可知，我們提出的協定可以適時減少頻寬利用率較少的 Piconet，留下網路中主要的 Piconet 來主持資料的傳輸。

五、結論

本論文主要提出一個流量管理協定，以完全分散式的做法，來改善因 Scatternet 結構不佳所造成的網路瓶頸問題，其中主要分為二個部份：**Splitting Phase** 及 **Merging Phase**。在 **Splitting Phase** 中，我們利用新增連線來避開傳輸的瓶頸，若其與原本 Piconet 傳輸量少於一個門檻值時，再將其連線切斷，以分裂為二個 Piconet，並對分裂後的 bridge nodes 規劃其 sniff；而在 **Merging Phase** 中，我們亦提出了一個流量測定方法，將頻寬利用率低的 Piconet 予以打散合併，避免因不斷分裂而造成 Piconet 過多所引發的資料衝撞問題。如此可改善 Bluetooth 自由對頻所形成的不良結構，使其依照目前的傳輸需求，動態的變更 Scatternet 結構，以縮短 routing path length、減少代傳點的電量耗費，並適當的對 Piconet 予以合併，減少 bridge ndoe 切換 Piconet 時產生的 guard time，及 end-to-end transmission delay，以增加傳輸效能。

六、參考文獻

[1]. Andras Racz, Gyorgy Miklos, Ferenc Kubinszky, Andras Valko, “A Pseudo Random Coordinated Scheduling Algorithm for Bluetooth Scatternet,” *ACM MobiHOC*, Long Beach, California, USA, October 5, 2001.

[2]. Marco Ajmone Marsan, Carla Fabiana Chiasserini, Antonio Nucci, Giuliana Carello, Luigi DeGiovanni, “Optimizing the Topology of Bluetooth Wireless Personal Area Networks,” *IEEE Infocom - The Conference on Computer Communications*, New York, NY (USA), June 2002.

[3]. M. Sun, C.K. Chang and T.H. Lai, “A Self-Routing Topology for Bluetooth Scatternets," *The International Symposium on Parallel Architectures (I-SPAN)*, Manila, Philippines, May 2002.

[4]. Pravin Bhagwat, Adrian Segall, "A Routing Vector Method (RVM) for Routing in Bluetooth Scatternets,” *The Sixth IEEE International Workshop on Mobile Multimedia Communications (MOMUC'99)*, pp. 375-379, Nov. 1999.

[5]. Rohit Kapoor, Mario Gerla, “A Zone Routing Protocol for Bluetooth Scatternet,” 3808 H, Boelter Hall, UCLA, *Wireless Communications and Networking Conference (WCNC)*, 2003.

[6]. The Bluetooth Specification, *http://www.bluetooth.org* 1.0b & 1.1.

[7]. T. Salonidis, P. Bhagwat, L. Tassiulas, and R. LaMaire, “Distributed Topology Construction of Bluetooth Personal Area Networks,” *IEEE Infocom - The Conference on Computer Communications*, April 2001.

[8]. Ting-Yu Lin, Yu-Chee Tseng, Keng-Ming Chang, Chun-Liang Tu, “Formation, Routing, and Maintenance Protocols for the BlueRing Scatternet of Bluetooths,” Hawaii *International Conference on System Sciences (HICSS’03)*, pp. 313a, Big Island, Hawaii, January 2003.

[9]. W. Zhang, H. Zhu, and G. Cao, "Improving Bluetooth Network Performance Through A Time-Slot Leasing Approach," *IEEE Wireless Communications and Networking Conference (WCNC)*, March 2002.

[10]. Wensheng Zhang and Guohong Cao, “A Flexible Scatternet-wide Scheduling Algorithm for Bluetooth Networks,” *IEEE International Performance Computing and Communications Conference (IPCCC)*, Embassy Suites Phoenix North Phoenix, Arizona, April 2002.

[11]. Yoji Kawamoto1, Vincent W.S. Wong, and Victor C.M. LeungA,” Two-Phase Scatternet Formation Protocol for Bluetooth Wireless Personal Area Networks,” *IEEE Wireless Communications and Networking Conference (WCNC’ 2003)*, pp. 1242-1247, New Orleans, Louisiana, March 2003.

[12]. Rachid Ait Yaiz and Geert Heijenk, “Providing Delay Guarantees in Bluetooth,” *International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pp. 722, May 2003