Ad hoc 無線網路上的有效繞路演算法
An Efficient Routing Algorithm in Ad Hoc Networks

林 嘉 彬
Chai-Bing Lin
逢甲大學資訊工程學系
Department of Information Engineering,
Feng Chia University
cblin@plum.iecs.fcu.edu.tw

黃 秀 芬
Shiow-Fend Hwang
逢甲大學資訊工程學系
Department of Information Engineering,
Feng Chia University
sfhwang@fcu.edu.tw

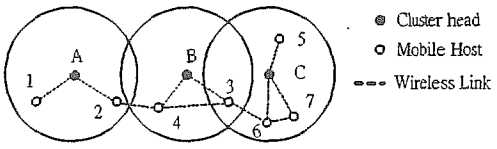
摘 要

本論文利用叢集(Cluster)和虛擬骨幹(Virtual backbone)的觀念，在 ad hoc 無線網路上，設計一個有效的繞路演算法。我們首先提出了標貼叢集演算法(Label-Clustering Algorithm)，此演算法能在 O(\|V\|)時間內將整個網路分成數個叢集，並且所得的叢集個數，在平均上會小於等於已知的叢集演算法，如高連結度叢集演算法(High-Connectivity Clustering Algorithm)；接著，我們在 O(logn)的時間裡，將叢集的管理者(Cluster head)架成一個虛擬骨幹，其中 n 為叢集管理者的個數，而此骨幹的點數最多為 3n。最後，藉由此緊緻的虛擬骨幹及良好的資料儲存，達到有效的繞路。

關鍵字：Ad hoc 無線網路, cluster head, clustering, routing, mobile host.

一。 簡 介

在無線網路中，行動主機(mobile host)是可任意移動的，故其繞路問題的複雜度也較傳統有線網路為高，所以有線網路繞路演算法常常無法直接適用於無線網路上，特別是在 ad hoc無線網路中，由於沒有基地台的支援，行動主機常常必須紀錄一些網路資訊來維持路徑的暢通，但這些資訊卻隨著行動主機的移動而常常變動，而追蹤這些變動點，一般來講可以分成兩個方法，第一種為利用路徑表，就是只要有一個行動主機變動位置(網路結構改變)，整個網路上所有的行動主機就必須更新他們的路徑資料庫([17,18])，這種行為不僅會使行動主機負擔過重，還會造成網路頻寬擁塞及需要大量記憶空間等缺點，而好處就是每個行動主機所紀錄的資料都是正確且最新的，當需要路徑時，可以直接從記憶體中尋找，立刻建立連線；另一種方法就是需要路徑的時候再開始去尋找，所以會有路徑搜尋時間較長的缺點，其優點就是不必一直維護路徑表，並且不需用到太多的記憶空間來紀錄網路資訊，這兩種方法各有優缺點，為了結合這兩個方法的優點，就有學者就利用叢集(cluster)的觀念([4,6,7,8,9,10,12,15,19])來改善這些問題。他們將整個網路劃分成一個個叢集，並從中選出 cluster head，由這些 cluster head 來負責管理叢集裡的成員和交換重要的相關資訊，並紀錄整個網路的架構，但非 cluster hea 的點就不必紀錄這些資訊，如此一來，整個網路的每個點所要紀錄的總資料量就可以大大的減少，在路徑搜尋速度方面，非 cluster hea 點如果需要路徑時，就直接向所屬的 cluster head 要求路徑，所以可以得到不錯的路徑搜尋速度。如圖一，每個點都只紀錄其相鄰點的資訊，而不必把自己的路徑表再傳給其他點，只要傳給自己所屬的 cluster head，如此一來就可以減少傳輸路徑表時所造成的資料量氾濫。



圖一、 Cluster 的架構

我們所知在 cluster 的劃分方法上，有 M. Gerla 和 J.T.C. Tsai ([4])提到的"最低識別碼叢集法(Lowest ID clustering)"和"高連結度叢集法(Hig -connectivity clustering)"。這兩個方法的特點都是簡單且快速，但卻都會使得在路徑維護時較麻煩。比較特別的是 B. Das 和

V. Bharghavan([6,7])提出一種以"最小連通控制集(Minimum connected dominating set)"的方法來做叢集，這個方法是利用最小連通控制集來形成虛擬骨幹(Virtual backbone)，讓 cluster head 間的訊息溝通，都經過虛擬骨幹，如圖二，進而把每個叢集給串聯起來，並使得在路徑的維護上有較好的效率，但要找出一個最小連通控制集卻是一個 NP-Complete 的問題，雖然他們用了一些近似法(Approximation Metho 來求解，但 其 誤 差率高達 $2H(\Delta)$，其中 $H(\Delta)=\sum_{i=1}^{\Delta}1/i\le \ln\Delta+1$。

二. 繞路演算法

在 ad-hoc 網路上，每個點(行動主機)都紀錄著相鄰點的訊息，我們要在這些點中找出一些適當的點當 cluster hea ，並確保除了 cluster hea 外，每一個點都必須至少被一個 cluster head 管轄，當整個網路的 cluster head 都被找出後，再利用 shortest path tree，找出這個網路的虛擬骨幹，方便 cluster head 之間互相傳遞訊息用，有了虛擬骨幹之後，cluster hea 就可以不需紀錄整個網路的連結狀態，而只要紀錄 cluster head 之間的路徑還有和在同一個 cluster 裡所有成員的資訊即可，當有點需要和其他點連結時，即可直接向他所屬的 cluster head 要求路徑，如果目的點不在同一個叢集，此時 cluster hea d 就藉由虛擬骨幹向其他的 cluster head 詢問目地點的位置，當目的點的 cluster hea 傳回路徑資訊後就可以建立連線。

2.1 Cluster Head 選擇

在 ad hoc 網路中，我們要在所有的點中選出一些適當點當 cluster head ，以方便管理。一般常用的演算法，如 Lowest ID 或是 Hig -connectivit 他們得出來的 cluster head 可能是階度(degree)為"1"的端點，而造成多餘的 cluster head，再加上 cluster head 之間的互斥性，而使得這些方法所畫出來的虛擬骨幹長度也會較長，而我們利用最小控制集(minimum dominating set)的觀念，設計了一個標貼叢集演算法(Label-Clustering Algorithm)，使劃出之叢集，更適合用於虛擬骨幹之上，其主要概念就是，在做 clustering 時，對每一個點當時的狀態貼一個標籤(label)，以方便相鄰點互相了解彼此的狀態，當每個點都了解彼此的狀態下，做 clustering 時就可以依此標籤來決定誰當 cluster hea 會比較適合，而用這種方式法求得的 cluster head 的個數在平均上不會多於 Hig -connectivi t 等演算法，並且由於這種 cluster head 個數少且不互斥的特性，更會使得虛擬骨幹的長度較短，而更適合用在虛擬骨幹上，以下我們將說明標貼叢集演算法的運作方式。

步驟一：

在網路起始狀態，或是沒被 cluster head 給管轄時，每個點的狀態皆設定為 B (Bounded)，此時開始查看相鄰點和自己的狀態，如果自己的階度為"1" (圖二的 4、7 號點)，，且相鄰的的階度不為"1"時，則把相鄰點的狀態設成 R(required)，也就是 cluster head(圖二的 3、8 號點)。

步驟二：

如果相鄰點有階度為"1"的點存在，而自己的階度不為"1"的時候，則把自己的狀態改成 R，所有相鄰點，除狀態為 R 的點之外，皆改成 F (Free)的穩定狀態。

步驟三：

當自己的階度不為"1"且無階度為"1"的相鄰點時，則表示相鄰的狀態，不是為 B 就是為 F，此時，決定哪個相鄰點或自己該成為 R 的條件，就是看誰擁有較多的狀態為 B 的相鄰點，如果不行，則再比彼此間的階度，最後則是比 ID，而這個步驟就是為了讓較好的點成為 R。

以下就是標貼叢集演算法的詳細步驟，其中 N(x)是指 x 所有相鄰點所成的集合，而 N[x]=N(x)∪ {x}.

標貼叢集演算法:

Label- clustering algorithm()

Any node "x" initial status is B

While L(x)=B Do

If deg(x)=1 (let y∈ N(x)) Then

  If deg(y) ≠ 1 Then

    L(y)=R

    $\forall z\in N(y)$: If L(z) ≠ R Then L(z)=F

  Else (deg(y)=1)

    If ID(x)<ID(y) Then

L(x)=R

L(y)=F

Else

L(y)=R

L(x)=F

Else (deg(x) ≠ 1)

If ∃y∈ N(x) with deg(y)=1 Then

L(x)=R

∀ z∈ N(x) : L(z) = F

Else

If ∃y∈ N(x) with L(y)=R

L(x)=F

Else

∀ z∈ N[x]: $d_1(z)=|\{y\in N[z] \mid L(y)=B\}|$

$t_1=\max\{d_1(z) \mid z\in N[x]\}$

$S_1=\{z\in N[x] \mid d_1(z)=t_1\}$

$t_2=\max\{\deg(z) \mid z\in S_1\}$

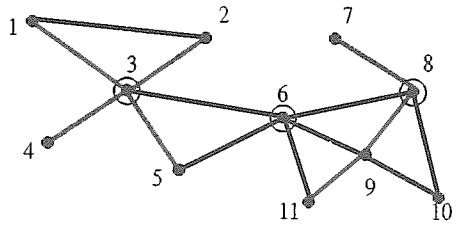
$S_2=\{z\in S_1 \mid \deg(z)=t_2\}$

Choose ∈ S_2 with ID(y)=min{ID(z) | z∈ S_2}

L(y) =R

∀ w∈ N(y):If L(w) ≠ R Then L(w)=F

EndWhile



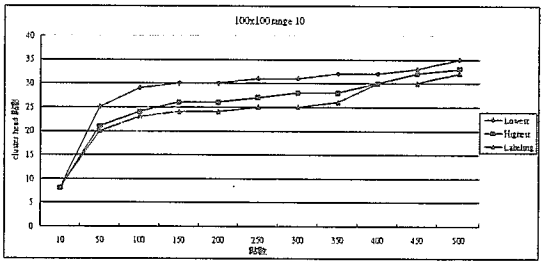
圖二、 Cluster hea 的形成

在我們的方法裡，一個點，可以同時屬於多個 cluster hea ，也就是 cluster 間有重疊的特性，這種特性雖然會多紀錄一些的資料，但這種特性也會使得 cluster 在往後的維護上較方便。

標貼叢集演算法的方法找出來的 cluster head 不僅個數少，同時 cluster head 也更密集地靠近在一起，這個特性，可以減少非 cluster hea 的點加入 虛擬骨幹的機會，而使得虛擬骨幹的長度縮短，以降低 cluster head 的通訊時間和傳輸成本。

標貼叢集演算法的特性就是不強求以最高階度的點或是以 ID 大小來當 cluster head ，基本上是先判斷本身位置是否合適，然後再選出最高階度的點看是否有資格當 cluster head ，如果遇到相同階度的點，就以 ID 大小來比較誰可以當 cluster head，而另一個特性就是 cluster hea 之間不會有絕對的互斥性，像 Lowest ID 或 Hig -connectivity clusteri n 演算法 cluster head 之間必須有一個 gateway node，這種互斥性看起來好像會減少 cluster head 的個數，但這種 cluster head 互斥特性，卻會把一些 cluster head 逼到階度為"1"或是不適合 cluster hea 存在的位置，而造成多餘的 cluster hea 或增長虛擬骨幹的長度。

為了說明由標貼叢集演算法所劃出來 cluster head 在平均上會比 Lowest ID、Hig -connectivity clusteri n 演算法所得出之 cluster hea 少，我們在一個 100 平方的矩陣中做 clustering 模擬，而得到如圖三的結果。



圖三、 Clustering 演算法之比較

此外，此標貼叢集演算法所形找出的 cluster hea 亦有較緊密之特性，其對我們所欲架設之虛擬骨幹將有所助益。而整個演算法亦有不錯的複雜度，將分析於輔理 2 和定理 3。

輔理 1 由標貼叢集演算法所找出之每個 cluster head，到其最近的 cluster hea 之距離最多為 3。

證明:

假設有一個 cluster head 到其最近的 cluster hea 之距離 P 大於 3，因此可知每個 cluster hea 可以管轄所有距離為"1"的點，所以減去兩 cluster hea 所管轄之距離，則 P 還大於 1，則表示此兩 cluster hea 之間還有

點未被其他 cluster head 所管轄或表示此兩 cluster hea 之間還有一距離更近之 cluster head 存在，如此一來就與已知條件不合，且與假設矛盾，故得證。

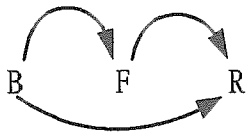
□

輔理 2 在標貼叢集演算法中，每個點最多只需送出 3 次 messag e即可完成 clustering。

證明:

在網路起始時，每個點把自己的 ID 和當時的狀態廣播給相鄰點，而當狀態由 B 轉換成 F 或 R 時，再把自己的狀態廣播給相鄰點。由於在 clustering 時，狀態的轉換為不可逆，如圖四，所以 clustering 動作，每個點最多只需送出 3 次 messag e即可完成。

□



圖四、 標貼叢集演算法狀態表

定理 3 在圖 G=(V,E)中，標貼叢集演算法的時間複雜度為 O(|V|)

證明:

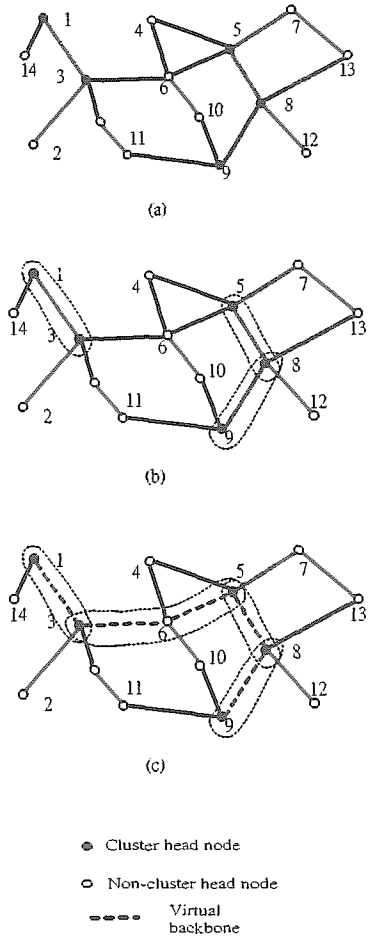
由輔理 2 可知，在標貼叢集演算法中，每個點最多被標貼 3 次，而每次決定為何標貼時，比較次數最多為階度(degree)，所以，整個演算法可在 O(|V|)的時間內完成。

□

2.2 虛擬骨幹的架設

我們架設虛擬骨幹的主要用意有二，一是節省整體節點對於路徑表的記憶空間，再來就是希望路徑可以較快的取得，為了達到這兩個目的，我們結合了 cluster hea 的特性，和虛擬骨幹的優點，以達到我們的需求，而虛擬骨幹的用途，是用來讓 cluster head 互相溝通用的，使得 cluster head 之間可以直接通訊，而不必再花費時間尋找彼此間的路徑。

因為我們要利用到虛擬骨幹，所以首先必須先定義哪些點必須被包含在虛擬骨幹裡，而這些點就是我們的 cluster head，我們把這些 cluster hea 連接起來，成為虛擬骨幹，在一開始沒有虛擬骨幹的情況下，彼此間不知道對方的位置，我們只需找每一個 cluster head 到距離它最近 cluster head 的路徑，由輔理 1 可以知道，這條路徑最多為"3"，且每個邊的成本為"1"，以至於每個 cluster hea 可以在 O(1)的時間內找到另一個 cluster head，所以完成這個動作的時間複雜度為 O(1)，再來就是要把這些最多 n/2 棵的 shortest path tree 連成一棵完整的樹，這個步驟可以由每一棵 shortest path tree 開始向外和其他棵 shortest path tree 相結合成更大棵的樹，直到只剩下一棵樹為止，由於每個邊的成本皆為"1"，所以可以省去搜尋兩棵樹之間最短路徑的時間，和第一個步驟一樣，最近的兩棵樹之間的距離不超過"3"，所以兩棵樹結合成一個樹的時間複雜度為 O(1)，重複合併動作，一直到只剩下一棵完整的樹為止，而完成一棵唯一樹的時間複雜度則為 O(log*n*)。



圖五、 虛擬骨幹的形成

以圖五(a)為例，共找到 cluster head 1、3、5、8、9

號點，我們要開始架設虛擬骨幹，首先第一步就是以這些 cluster head 為起始點，開始向外尋找離自己最近的 cluster head 點，而形成數棵 shortest path tree，就如圖五(b)所示，在第一步驟之後，整個網路形成{1，3}、{5，8，9}兩棵樹，而第二步驟就是把這兩棵樹在合併成一棵樹，當然也是找這兩棵樹中的最短距離，所以 3-6-5 這條路徑就被選擇，而形成{1，3，6，5，8，9}這棵樹，如圖五(c)所示，而這棵樹就是我們所要找的虛擬骨幹。

在我們找出來的這條虛擬骨幹中，裡面非 cluster hea 點的個數是有一定的上限。

輔理 4 在虛擬骨幹中，非 cluster head 點的個數，最多為 cluster hea 點個數的 2 倍

證明:

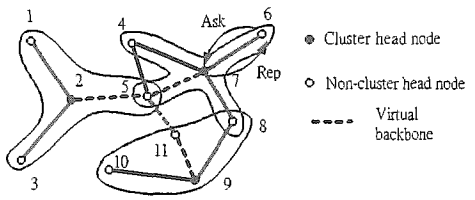
假設在虛擬骨幹中，cluster hea 點的個數為 n。因為虛擬骨幹的葉節點必為 cluster head 且由輔理 1 可以知道，兩距離最近的 cluster head，其最短路徑路徑裡，最多只有 2 個非 cluster head 點存在。又由於虛擬骨幹是一棵樹，所以 n 個 cluster hea 之間有 n-1 個路徑相連，每個路徑最多有兩個非 cluster head 點，由此可知，在虛擬骨幹中，非 cluster head 的點最多只有 2(n-1)個，事實上，再一步分析，更正確的上限是最多會有 $\left\lceil \frac{3}{2}n \right\rceil$ 個非 cluster hea 點存在於虛擬骨幹中。

□

2.3 路徑取得

有了虛擬骨幹之後，就可以開始為各個點提供路徑搜尋的服務，在我們的架構中，虛擬骨幹的功能主要是用來作為 cluster head 間的溝通管道，由於 cluster head 間的訊息通訊頻繁，所以除了 cluster hea 間的通訊外，我們希望其他訊息盡量不經過虛擬骨幹來溝通，所以我們就利用了代替道路的方法，避免一般通訊訊息經過虛擬骨幹傳輸，以減少虛擬骨幹的頻寬擁塞。

如圖六，在一個 cluster 中，cluster hea 紀錄著整個 cluster 成員的所有紀錄，假設 6 號點要和 10



圖六、 路徑取得

號點溝通，如果經過 7➔5➔11➔9 這段虛擬骨幹而到達 10 號點，則會增加虛擬骨幹的負擔，假如我們有第二條路徑，則可以使 6 號點到 10 號點的路徑改為 6➔7➔8➔9➔10 如此一來，不僅僅可以有較短的傳輸路徑，更可以減少虛擬骨幹上的資料量，使得處於虛擬骨幹的點不至於過於忙碌，而為了達到這種代替道路的特性，cluster hea 就必須多紀錄自己成員的一些資訊，也就是自己 cluster 裡的成員之中，gateway node 如 5 號點、8 號點，和哪個 cluster 相連，例如 7 號 cluster head 點紀錄 8 號點時，就順便紀錄 8 號點也是屬於 9 號 cluster hea 點；所以，當 6 號點向 7 號 cluster head 點要求到 10 號點的路徑時：

步驟一：7 號 cluster head 點藉由虛擬骨幹向其他 cluster hea 詢問 10 號點的所在位置

步驟二：9 號 cluster hea 點收到路徑要求的訊息後，要開始傳訊息回 7 號 cluster head 點，9 號 cluster hea 點會先看要求路徑的點是屬於哪個 cluster 並且經過虛擬骨幹的哪些點，在這裡，9 號 cluster hea 點知道這個要求路徑訊息是由 7 號 cluster hea 點所發出，並且經過 7、5、11、9 這些虛擬骨幹點。

步驟三：9 號 cluster head 點就開始查看自己的 cluster 成員，看是否有 gateway node 可以和 7、5、11 這些 cluster 相連，而且順序愈靠近 7 號 cluster hea 點的 gateway node 愈好。

步驟四：如果沒有，則把要傳到 7 號 cluster head 點的路徑訊息，交給 11 號點，由 11 號點繼續檢查自己成員是否有 gateway node 連到 7 號、5 號點，如此一直下去，直到路徑訊息傳回到 7 號 cluster head 點，如果這些步驟之後還是沒找到好的 gateway node，則 7 號 cluster head 點就收到只能走虛擬骨幹這條路的訊息，而我們這個例子剛好有一個 gateway node 連接 7 號 cluster hea 點和 9 號 cluster hea 點，所以 9 號 cluster head 點回應給 7 號 cluster hea 點的路徑則是 9➔8➔7，雖然這個訊息也會經過 11 號、5 號點直到 7 號 cluster head 點，但由於之間沒有更好的路徑可以取代 9➔8➔7 這條路徑，所以 7 號 cluster head 點就把這條路徑的訊息傳給 6 號點，由 6 號點經由這條路徑跟 10 號點溝通，而達到我們要分散資料流的目的。

在我們的方法裡，每個點皆會記錄其相鄰點的一些

基本資訊，例如相鄰點的型態(一般點、cluster head 或是 gateway node)、和哪一個 cluster 相連等資料。以圖六的 9 號點為例，它的路徑表為表一所示，有 8、10、11 三個相鄰點，其中 10、11 號點為一般點，它們沒有直接和其他的 cluster 相連，如表一中的 11 號點的紀錄表示，經過 11 號點，可以直接到達 2 和 7 兩個 cluster，但 11 號點卻沒有和 2、7 兩點直接相連，而表一中的 8 號點則是一個 gateway node，它和 11 號點不同的是，它是直接和 7 號 cluster head 點相鄰，所以我們在選擇路徑時，當然就是以 gateway node 為優先，然後才考慮一般點。而 cluster head 點所紀錄的路徑表和一般點一樣，只不過它還必須紀錄整條虛擬骨幹的連結狀態，以使得 cluster hea 之間可以迅速的交換資料。

Node	Type	NC
8	G	7
10	N	0
11	N	2,7

N: 一般點
G: Gateway node
CH: Cluster head
NC: near cluster head

表一、 圖六之 9 號點路徑表簡圖

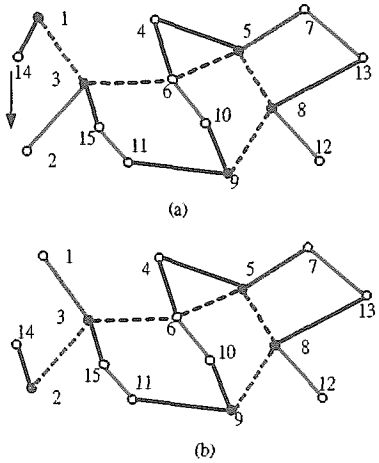
由於 cluster 的觀念，使得我們路徑表的更新動作不會太頻繁，因為如果有一個點移動，要更新路徑表的只有其相鄰的點動作即可，而不必像 DSDV 一樣必須整個網路的每個點都必須更新路徑表，而省下大量的傳輸和處理時間。

2.4 虛擬骨幹的維護

當虛擬骨幹架設起來後，由於 ad ho c網路的點可任意移動，這種特性會造成整個網路架構的變動，使得點與點之間的連結中斷或是新的連結出現，而對於這種現象，我們必須能保證整個網路還是能繼續運作，尤其是我們之前所建立起來的虛擬骨幹，也會因網路架構的變動而改變，例如虛擬骨幹 斷裂成幾部分，或是新的點加入虛擬骨幹等情況，我們將依移動點的特性來分類探討網路如何維護。

A、非虛擬骨幹點的移動

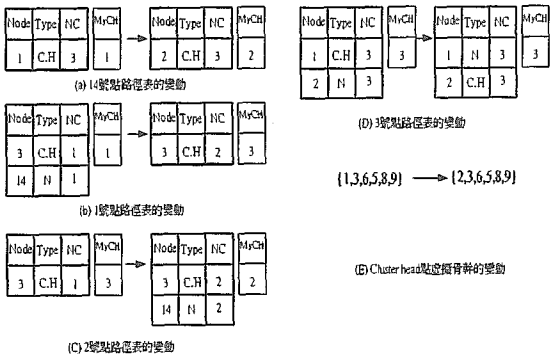
一般來講非虛擬骨幹的點，即非 cluster head 的點，這類點移動時的維護是最容易的，在之前提過，這類點皆有 cluster hea 所管轄，所以他們的狀態皆處於穩定狀態，所以這類點的狀態皆為 F(如圖七(a)的 14 號點)，而當這類點無法再收到原本的 cluster head 所發出的訊號時，它的狀態就會由於無 cluster head 管轄而變成無人管轄的狀態 B，而當一個點狀態為 B 時，就開始注意鄰點的訊息，並且執行標貼叢集演算法，如果相鄰點之中有 cluster hea 存在，就把自己的狀態設成 F，並且加入這個 cluster，或是依相鄰點的狀況，產生新的 cluster head(如圖七(b)的 2 號點)，而這個 cluster head 可能是其中的一個相鄰點，或是移動點本身，然後再加入虛擬骨幹的群組裡。



圖七、 非虛擬骨幹點移動

在資料異動方面，一般來講，由於叢集的區域性，所以受影響的只有移動點、跟移動點相連的點和 cluster hea 點這幾種，如圖七的 1、2、14 號點，和虛擬骨幹上的 cluster hea 點；在圖七(b)中，1 號點把原來的 14 號點資料去除，並且把自己的 cluster head 設定成 3 號點，而 2 號點由於 14 號點的加入，所以在自己的路徑表中加入 14 號點的資料，再加上 2 號點升級成 cluster head，所以向其他取得虛擬骨幹的資料，並且加入自己的 ID，送給其他 cluster head。而其他 cluster head {3,5,8,9 號點，收到 2 號點送出的新的虛擬骨幹資料後再把自己的虛擬骨幹資料更新，至於 14 號點的所更改的資料，就只要把相連點由原本的 1 號點改成 2 號點，並且設定 2 號點為 cluster head 即可，如表二，而其他點皆不受其影

響。



表二、 圖七中 14 號點移動後路徑表的變化

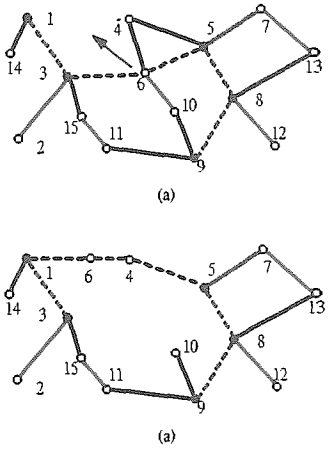
B、 虛擬骨幹點的移動

在虛擬骨幹中，所包含的點不僅僅只有 cluster hea ，還有一些非 cluster head 的點，這些點的移動時，雖然對其相鄰點影響不大，但往往會導致虛擬骨幹斷裂成數個部分，而對於 cluster hea 的移動，也會有這類的問題產生，我們將虛擬骨幹的點分成葉節點、非葉節點但為 cluster head 及非葉節點且非 cluster head 三類來討論其移動時如何維護。

case 1: 非葉節點且非 cluster head

存在虛擬骨幹內部，卻非 cluster head 的點，都是在求兩 cluster head 間的最短距離時，被包括進來的點，雖然這些點的移動對相鄰點影響不大，但對於虛擬骨幹而言影響卻很大，因為他們是處於兩個 cluster hea 間的中介點，所以當然也必定處於虛擬骨幹的中間骨幹上，所以只要他們一移動，就會造成虛擬骨幹的架構改變，甚至使得虛擬骨幹產生不連結的斷裂狀態，而分成數個群(group)，這些群之間可能還有其他通路連結，或是完全中斷，如果發生完全中斷的情形，就只有把網路分成不互相連通的數個群，群與群之間無法互相溝通，而這裡我們所關心的是當虛擬骨幹斷裂後，我們如何快速的再把虛擬骨幹給架設起來，讓原本斷裂的虛擬骨幹再成為完整的虛擬骨幹。

在圖八(a)中，由於 6 號點的移動，而使得虛擬骨幹分成{1,3,6}和{5,8,9}兩個部分，首先利用標貼叢集演算法，讓移動後受影響的點恢復成穩定狀態(R 或 F)，然後再利用之前所提到的架虛擬骨幹的方法，把這兩棵樹，連成一個完整的 shortest path tree，而形成圖八(b)的完整虛擬骨幹。

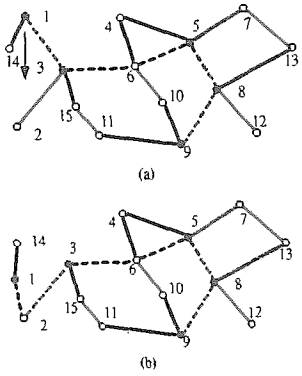


圖八、 在虛擬骨幹裡非 cluster hea 點的移動

在點移動後資料變化方面，除了移動點、相鄰點和 cluster head 點資料需要更新外，和之前所提的比較不同地方就是 4 號點，由一般點升級成虛擬骨幹的點，而這兩點所屬的 cluster hea 就把這兩點的 ID 加入虛擬骨幹內，再把新的虛擬骨幹資料送給其他 cluster head 即可。

Case 2: 葉節點

由於是虛擬骨幹的葉節點，所以此類的 cluster head 移動時，並不會有虛擬骨幹斷裂情況的產生(如圖九(a)的 1 號點)，所以我們可以不必考慮虛擬骨幹斷裂後重新連結的狀況，只需考慮到 cluster hea 移動後原本的連接點和新連結點狀態重新設定即可，對於舊的相鄰點，由於無法再收到 cluster head 所送出的訊息，所以這些點開始執行標貼叢集演算法，選出新的 cluster head，而此 cluster head 點移入新的 cluster 時，必須判斷是否該降級，我們利用一個簡單的演算法來決定。



圖九、 在虛擬骨幹裡葉節點的移動

假設 x 為移動的 cluster hea 點，並對所有點 z∈ N[x]定義一個 c 函數如下:

c(z)=| { w∈ N(z) | L(w)=R}|

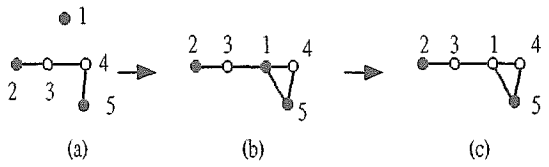
Down()

If ∃y∈ N(x) with L(y)=R & ∀y∈ N(x) with c(y)>1

Then

L(x)=F

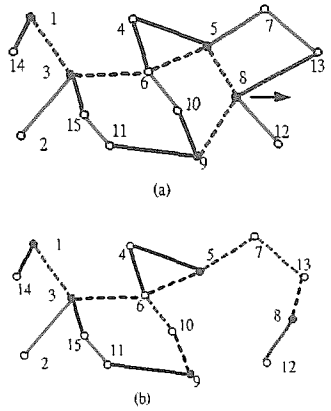
以圖十為例，1 號點往 3、4 號點之間移動時，就形成圖十(b)的狀態，此時我們就有必要讓 1 或 5 號點降為非 cluster head 的狀態 F，此時如果 1 號點先執行 Down() 的動作，由於它的相鄰點都被 1 個以上的 cluster hea 所管轄，則 1 號點就放棄 cluster hea 的地位，狀態就由 R 降為 F 而形成圖十(c)的狀態，當然最後也可能是 5 號點降級為一般點。



圖十、 Cluster hea 間的消長

Case 3: 非葉節點但為 cluster head

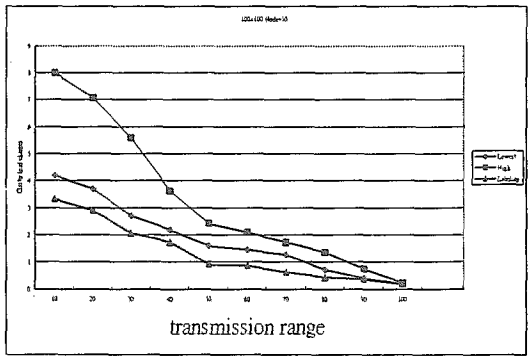
處於虛擬骨幹內部的 cluster hea 移動情況，跟處於虛擬骨幹裡的一般點類似，他們的移動皆會造成虛擬骨幹的斷裂，只是這類的 cluster hea 移動時影響較大，它不僅影響到虛擬骨幹的連續性，還會影響到它的相鄰點，一般來說，當這類 cluster head 移動時，它的相鄰點如果是一般點，它就必須檢查自己是否被其他的 cluster head 所管轄，然後執行標貼叢集演算法，決定自己的狀態，而 cluster hea 本身在移入其他 cluster 之後也必須執行 down 的演算法，決定是否降級成為一般點。以圖十一(a)來舉例，當 8 號點移開時，虛擬骨幹就會分裂成 {1,3,6,5}、{9}、{8}三部份，和之前的步驟一樣，先利用標貼叢集演算法，讓移動後受影響的點恢復成穩定狀態，然後再把這三顆樹連成一顆樹(如圖十一(b))，而形成新的虛擬骨幹。



圖十一、 虛擬骨幹中非葉節點但為 cluster head 移動

在維護方面，還有一個重要的問題要考慮，那就是 cluster head 的變動率，而這個 cluster head 的變動率則直接影響到我們虛擬骨幹的穩定度，當 cluster hea 變動率小的時候，我們的虛擬骨幹就更穩定，甚至還可以省去 re-clustering 的時間，所以我們就必須比較我們的標貼叢集演算法和其他 clustering 演算法之間 cluster hea 的穩定度。

為了要顯示我們的方法有比前兩種演算法的 cluster head 變動率低，所以我們在一個 100x100 的方陣中，灑 50 個可任意移動的點，依不同的通訊距離，測試移動 100 個單位時間後，cluster hea 的變動率，由所得結果可知我們的標貼叢集演算法具有較高 cluster hea 穩定性。



圖十二、 cluster hea 穩定度比較

三。 結論

本論文是針對 ad ho 無線網路之繞路方法，從叢集的劃分到路徑的架設與維護，提出一個整體性的有效解決方法。我們首先提出的標貼叢集演算法，它可以在 O(|V|) 的時間內把網路分成數個叢集，此方法所得之叢集個數不但少，且有集中之特性，這將使我們架設的虛擬骨幹更短且資料記錄量減少。其次，我們利用 shortest path tree 架成一個虛擬骨幹，因其緊緻且長度小，故能達有效之繞路。最後，我們也提出網路維護方法，使整個演算法更趨於完整。

由於本論文著重在理論的研究和模擬，在未來，如果可以實際配合 ad hoc 無線網路硬體架構方面的研究，必可使其更具應用價值。

參考文獻

[1] Josh Broch, D. A. Maltz, D. B. Johnson, Y. C. Hu and J. Jetcheva, "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols ", *ACM/IEEE MobiCom'98*, Oct 25-30 1998.

[2] Josh Broch, D. B. Johson, D. A. Maltz, "The Dynamic Source routing Protocol for Mobile Ad Hoc Networks" *Mobile Ad-hoc Networks (manet)*, http:// www.ietf.org/html.charters/ manet-charter.html, Aug. 1998.

[3] C. C. Chiang and Mario Gerla, "Routing and Multicast in Multihop, Mobile Wireless Network" In Proceedings of ICUPC '97.

[4] F. K. Hwang, D. S. Richards, P. Winter, "The Steiner Tree Problem", Elsevier Science Publishers B. V., 1992.

[5] T.H. Cormen, C.E. Leiserson, R. L. Rivest, "Introduction to Algorithms", McGraw-Hill Book Company,6th printing,1996.

[6] Bevan Das, V. Bharghavan, "Routing in Ad -Hoc Networks Using Minimum Connected Dominating Sets", in *IEEE International Conference on Communication (ICC '97)*, Jun. 1997.

[7] B. Das, R. Sivakumar and V. Bharghavan, "Routing in Ad Hoc Networks Using a Spine", in *Proc. IEE Computer Communication and Networks*, pp -39, 1997.

[8] M. Gerla and J.T.C. Tsai, "Multicluster, mobile, multimedia radio network", *ACM Journal on Wireless Networks*, pp. 255-265, 1995.

[9] Z. J. Haas and M. R. Pearlman, "The Zone Routing Protocol (ZRP) for Ad Hoc Networks", *Mobile Ad-hoc Networks (manet)*, http://www.ietf.org/html.charters/manet-charter.html, Nov. 1997.

[10] Ting-Chao Hou, Tzu-Jane Tsai, "Adaptive Clustering in a Hierarchical Ad Hoc Network", *ICS Proceedings of Workshop on Computer Networks, Internet, and Multimedia*, Dec, 1998.

[11] F. K. Hwang, D. S. Richards, P. Winter, "The Steiner Tree Problem", Elsevier Science Publishers B. V., 1992.

[12] M. Jiang, J. Li and Y.C. Tay, "Cluster Based Routin Protocol(CBRP) Functional Specification ", *Mobile Ad-hoc Networks (manet)*, http:// www.ietf.org/html.charters/ manet-charter.html, Nov. 1997.

[13] D. B. Johnson, D. 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.

[14] 李逸元, "無線區域網路之資訊安全控管機制", The 4th Computing Workshop,3/25~3/26 1998.

[15] Chunhung Richard Lin and Mario Gerla, "Adaptive Clustering for Mobile Wireless Networks," *IEE Journal on Selected Areas in Communication*, Vol. 15, No. 7, pp. 1265-1275, Sep. 1997.

[16] K. Mehlhorn, "A faster approximation algorithm for the Steiner problem in graphs", *Inf. Process. Lett.* 27 (1988) 125-128.

[17] Charles Perkins, E. M. Royer, "Ad Hoc on Demand Distance Vector (ADOV) Routing ", *Mobile A -hoc Networks (manet)*, http://www.ietf.org/html.charters/manet-charter.html, Aug. 1998.

[18] C. E. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing(DSDV) for Mobile Computers", *Proceedings of ACM SIGCOMM '94*, pp. 234-244, Oct.1994.

[19] Y.Rekhter, S. Hotz and D. E strin, "Constraints on forming clusters with link-stats ho -by-hop routing', *Technical Report* USC-CS-93-536, University of Southern California,1993.