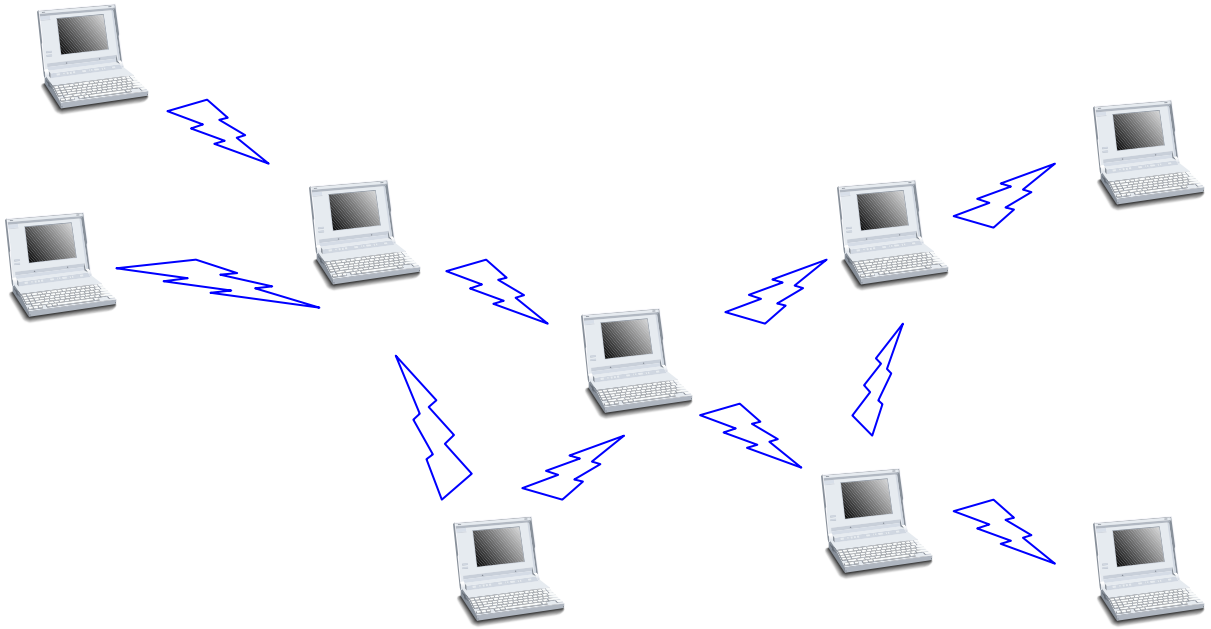
Chapter 8
無線隨意網路路由

簡介(一)

- 無線隨意網路：MANET = Mobile Ad Hoc Networks
  - a set of mobile hosts, each with a transceiver
  - no base stations; no fixed network infrastructure
  - multi-hop communication
  - needs a routing protocol which can handle changing topology



簡介(二)

- 無線網路不同於有線網路的特性：
  - 目的位址（Destination Address）不等於目的位置（Destination Location）。
  - 無線訊號強度會因處於不同環境而有不同程度之衰減 。
  - 無線網路具有動態網路拓撲之結構 。
  - 同屬一個無線網路集合將會共用同一個傳輸媒介 。
  - 資料傳輸過程未受保護，故易受外界干擾且保密安全性低。
  - 省電之課題
  - 具有分散式執行功能。
  - 無迴路（Loop-free）結構。
  - 具有多重中繼段（Multi-hop）路由。
  - 對動態鏈結作快速地適應。
  - 不可靠的廣播機制。

簡介(三)

- 無線網路路由通訊協定在功能與設計之考慮：
  - 路徑的建立是主動或被動 。
  - 是否需週期性更新路徑 。
  - 是否需維護多條路由 。
  - 網路架構為叢集式或平面式。
  - 是否提供安全機制。
  - 是否支援群播功能 。
  - 建立路徑所需花費之時間 。
  - 工作站電源管理的考量 。
  - 通訊頻寬的需求 。
  - 所需之參考資訊表格數量 。

簡介(四)

- 兩種路由方法
- 表格驅動式路由 (Table Driven Routing)
  - 主動式 Proactive 路由
- 需求式路由 (On-demand Routing)
  - 回應式 Reactive 路由

簡介(五)

- 表格驅動式路由 (Table Driven Routing)：每一個節點（工作站）都會各自維護自己的路由表（Routing Table），再利用各個節點週期性地廣播自己的路由資訊，來彼此交換與更新訊息並藉此得以主動發現所需之封包傳送路徑。
  - 循序的目的地距離向量路由協定（Destination-sequenced Distance-vector Routing，DSDV）。
  - 群首閘道交換路由協定（Clusterhead Gateway Switch Routing，CGSR）。
  - 無線路由協定（Wireless Routing Protocol，WRP）。

簡介(五)

- 需求式路由 (On-demand Routing)：當有某一節點需要一條路徑以傳送封包給目的節點時才開始建構。而各節點本身並不會週期性與主動性地傳送路由更新資訊給網路上其他節點 。
  - 輕量移動路由協定（Lightweight Mobile Routing，LMR）。
  - 暫時性順序路徑演算法（Temporally-ordered Routing Algorithm，TORA）。
  - 動態來源端路由協定（Dynamic Source Routing，DSR）。
  - 無基礎式需求距離向量路由協定（Ad Hoc On-demand Distance Vector Routing Protocol，AODV）。
  - 關聯性基礎路由協定 (Associativity-based Routing， ABR)。
  - 輔助定位路由通訊協定 (Location-aided Routing ，LAR)。

表格驅動式路由演算法

- 表格驅動式路由協定有兩類主要的演算法：
  - 距離向量路由（Distance-vector Routing，DVR）。
  - 鏈路狀態路由（Link-state Routing，LSR）。

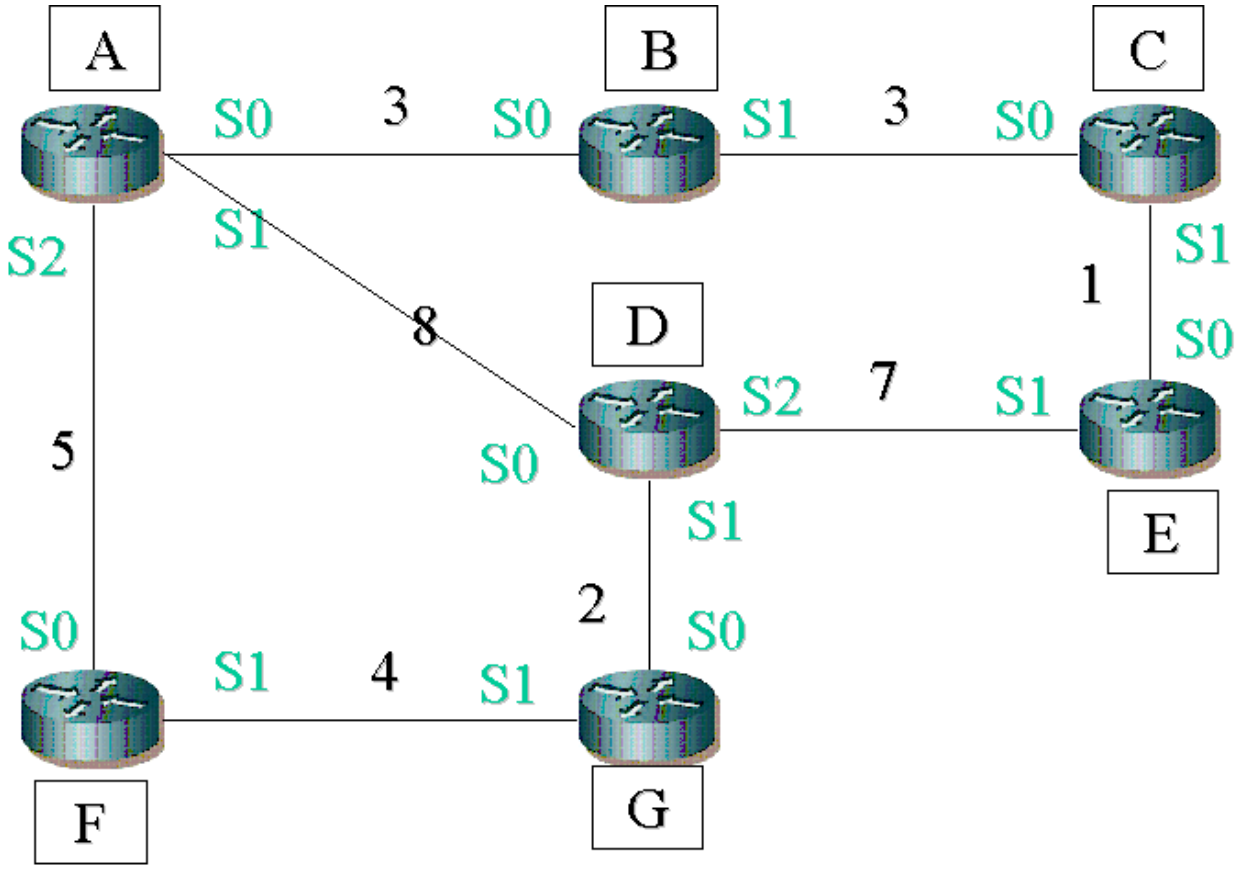
距離向量（Distance）與鏈結狀態（Link state）路由演算法

- 距離向量與鏈結狀態演算法最主要的差別：
  - 距離向量演算法需藉由其鄰近節點所告知之〝距離〞資訊做為其是否選擇該節點為路徑的依據
  - 鏈結狀態演算法則是每一節點經由鄰近節點的告知，獲得所有網路〝鏈結狀態〞的資訊（亦即擁有網路的全貌）後，再做為選徑的依據

距離向量演算法 (DVR)

- 使用距離向量的路由器都會自行維護一個路由表，他會記錄到每一個已知目的地的最佳距離到自己的路由表中
- 最典型的代表就是Bellman-Ford Routing Algorithm
- 當機器開機時，路由器中只有自己本身的路由資訊，然後會和鄰近的路由器以廣播的方式，彼此交換自己的路由表內的資訊，藉由這樣的交換訊息，很快的就可以得知整個網路的連結狀態

Example



Example

From A to	Interface	Metric
A	local	0

From B to	Interface	Metric
B	local	0

From C to	Interface	Metric
C	local	0

From D to	Interface	Metric
D	local	0

From E to	Interface	Metric
E	local	0

From F to	Interface	Metric
F	local	0

From G to	Interface	Metric
G	local	0

From A to	Interface	Metric
A	local	0
B	S0	1
D	S1	1
F	S2	1

From B to	Interface	Metric
B	local	0
A	S0	1
C	S1	1

From C to	Interface	Metric
C	local	0
B	S0	1
E	S1	1

From D to	Interface	Metric
D	local	0
A	S0	1
E	S2	1
G	S1	1

From E to	Interface	Metric
E	local	0
C	S0	1
D	S1	1

From F to	Interface	Metric
F	local	0
A	S0	1
G	S1	1

From G to	Interface	Metric
G	local	0
D	S0	1
F	S1	1

From A to	Interface	Metric
A	local	0
B	S0	1
D	S1	1
F	S2	1
C	S0	2
E	S1	2
G	S1	2

From B to	Interface	Metric
B	local	0
A	S0	1
C	S1	1
D	S0	2
F	S0	2
E	S1	2

From C to	Interface	Metric
C	local	0
B	S0	1
E	S1	1
A	S0	2
D	S1	2

From D to	Interface	Metric
D	local	0
A	S0	1
E	S2	1
G	S1	1
B	S0	2
F	S0	2
C	S2	2

From E to	Interface	Metric
E	local	0
C	S0	1
D	S1	1
B	S0	2
A	S1	2
G	S1	2

From F to	Interface	Metric
F	local	0
A	S0	1
G	S1	1
B	S0	2
D	S0	2

From G to	Interface	Metric
G	local	0
D	S0	1
F	S1	1
A	S0	2
E	S0	2

From A to	Interface	Metric
A	local	0
B	S0	1
D	S1	1
F	S2	1
C	S0	2
E	S1	2
G	S1	2

From B to	Interface	Metric
B	local	0
A	S0	1
C	S1	1
D	S0	2
F	S0	2
E	S1	2
G	S0	3

From C to	Interface	Metric
C	local	0
B	S0	1
E	S1	1
A	S0	2
D	S1	2
F	S0	3
G	S0	3

From D to	Interface	Metric
D	local	0
A	S0	1
E	S2	1
G	S1	1
B	S0	2
F	S0	2
C	S2	2

From E to	Interface	Metric
E	local	0
C	S0	1
D	S1	1
B	S0	2
A	S1	2
G	S1	2
F	S1	3

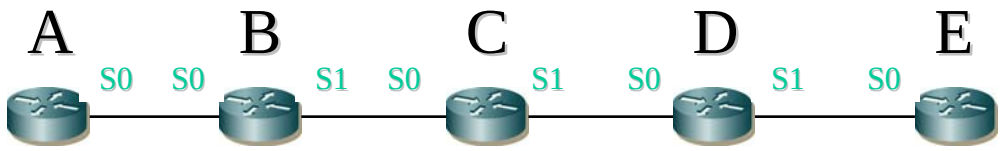
From F to	Interface	Metric
F	local	0
A	S0	1
G	S1	1
B	S0	2
D	S0	2
C	S0	3
E	S1	3

From G to	Interface	Metric
G	local	0
D	S0	1
F	S1	1
A	S0	2
E	S0	2
B	S0	3
C	S0	3

距離向量演算法-無限計數

- 因為每一網路節點不需掌握網路整體拓撲，因此在選徑時就有可能會有形成迴圈的現象，甚至有可能會使得路徑的選擇產生無限計數（Count to infinity）的問題

經一段時間後，路由器各自之完整的路由表如下：



From A to	Interface	Metric
A	local	0
B	S0	1
C	S0	2
D	S0	3
E	S0	4

From B to	Interface	Metric
A	S0	1
B	local	0
C	S1	1
D	S1	2
E	S1	3

From C to	Interface	Metric
A	S0	2
B	S0	1
C	local	0
D	S1	1
E	S1	2

From D to	Interface	Metric
A	S0	3
B	S0	2
C	S0	1
D	local	0
E	S1	1

From E to	Interface	Metric
A	S0	4
B	S0	3
C	S0	2
D	S0	1
E	local	0

假設A-B之間的鏈路斷了，B無法到達A，但是B卻從C得知有一條到A的路徑，於是B就更新自己的路由表，如下表

From B to	Interface	Metric
A	S1	3
B	local	0
C	S1	1
D	S1	2
E	S1	3

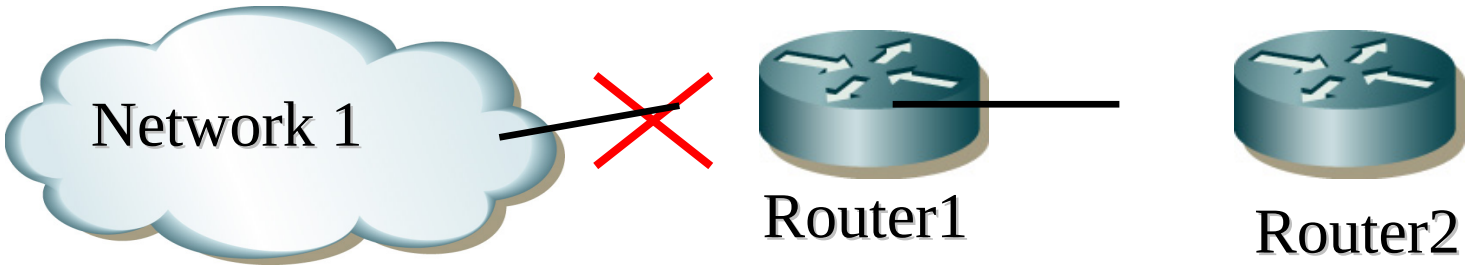
然而過一段時間，C 必須再次跟鄰居交換訊息，發現到 A 這個 entry 必須更新，於是 C 就更新自己的路由表，如下表

From C to	Interface	Metric
A	S0	4
B	S0	1
C	local	0
D	S1	1
E	S1	2

距離向量演算法

- 解決無限計數的方法：
  - **Maximum hop count**：限制其跳躍數
  - **Split horizon**：這個方法的原則是不回送資料，所以當A-B之間斷了，C不會再把A的訊息告訴B，因為到A的訊息是由B告訴C的
  - **Hold Down**： 利用Hold down timer，若hold down timer過期前的任何時間裡，從不同的路由器接到更新資訊，但指標較差，則不理會更新資訊。在hold down timer時效範圍內，不理會指標較差的更新資訊，則能有更多時間將中斷的改變資訊傳播到整個網路上。

Split horizon



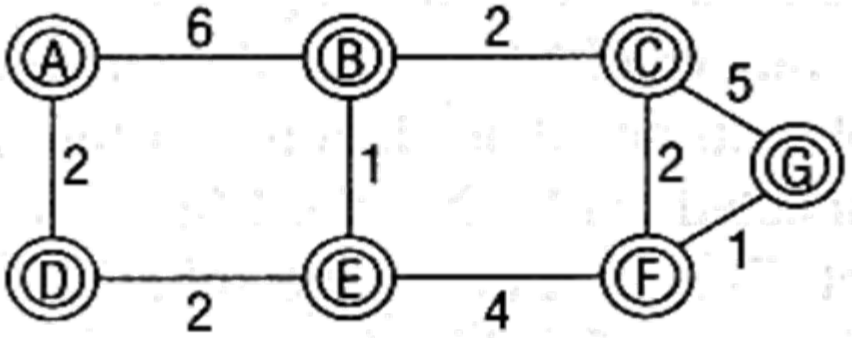
Router2不能告訴Router1有關Network 1的資訊，因爲那是Router 1告訴Router 2的。

鏈結狀態路由演算法

- 鏈結狀態演算法在一般圖形理論中也稱為最短路徑（Shortest Path First， *SPF*）演算法，它保持一份網路整體拓撲的資料庫。 再利用圖形理論中的最短路徑演算法找出該節點到各點之最短路徑
- 圖形理論中，最短路徑的演算法有許多種，但其中以Dijkstra演算法最有名，而Dijkstra演算法也被引用為OSPF（Open Shortest Path First）演算法中

鏈結狀態路由演算法：Dijkstra's Algorithm

Initial State: each host only knows its direct neighbors



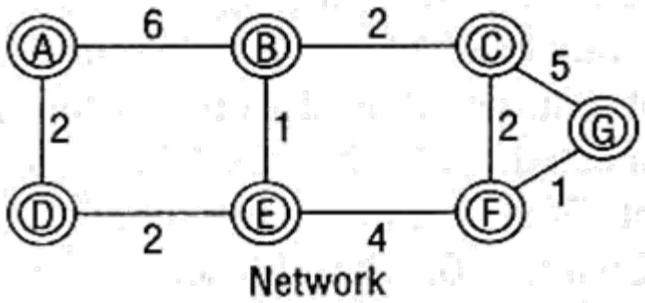
Network

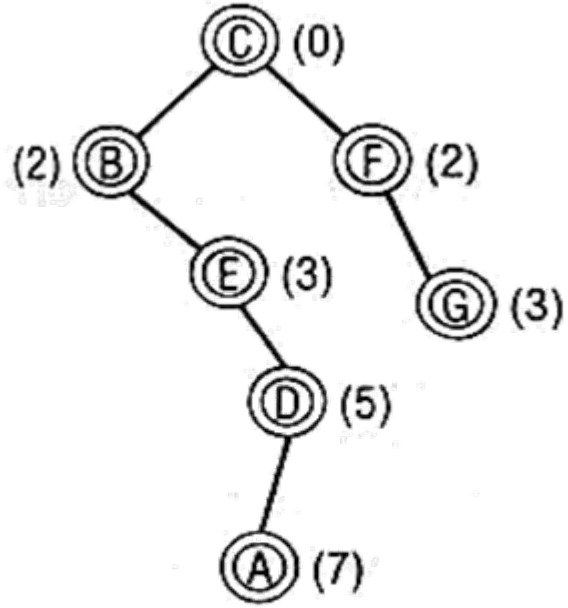
A	B	C	D	E	F	G
B/6 D/2	A/6 C/2 E/1	B/2 F/2 G/5	A/2 E/2	B/1 D/2 F/4	C/2 E/4 G/1	C/5 F/1

Link State Database

Figure 9.13

Evolution of States in C





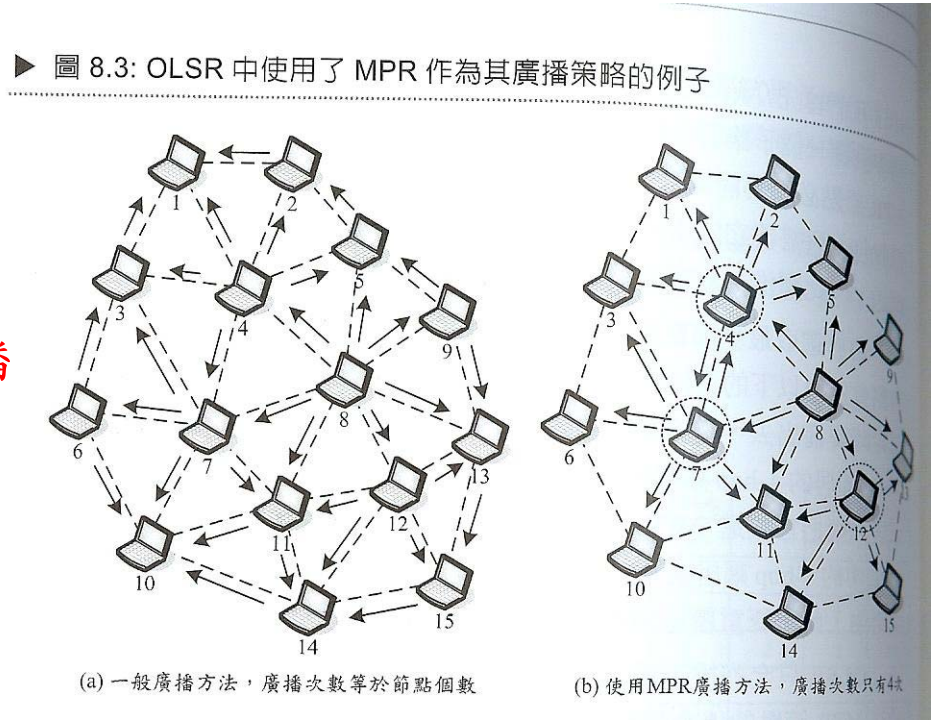
7. Place A in path.
   Examine A's LSP.
   No nodes left.
   Terminate.

- Comments: This is a centralized algorithm, not appropriate.

鏈結狀態路由演算法

- Optimal Link State Routing (OLSR)：
  - OSPF最佳化版本
  - 使用多重傳遞節點(MPR) 減少鏈結狀態更新的封包數量
  - MPR:挑選1-hop鄰居，使得能夠完全覆蓋全部2-hop鄰居

15次廣播



4次廣播

鏈結狀態路由演算法

- 鏈結狀態演算法有下列優點：
  - 無迴圈（Loopless）：路徑的決定是在擁有整個網路資訊後才決定，因此不會找出有迴圈的路徑
  - 多路徑（Multiple paths）到目的地：既然每一節點能獲知網路全貌，因此每一節點可依需要找出到目的地且不共用鏈結之多條路徑，而找出多條路徑的目的，可以是為了做路徑保護（備用路徑），或是分擔訊務使用

表格驅動式路由協定：循序的目的地距離向量路由協定 (DSDV)

- DSDV：針對DVR進行修正而得。
  - DSDV多了目的地循序號碼（Destination-sequenced Number）的記錄。利用目的地循序號碼來判斷路徑是否需要更新，避免因產生迴路效應而使路徑發生錯誤 。

表格驅動式路由協定：
循序的目的地距離向量路由協定 (DSDV)

目的地 Destination	下一個節點 Next Hop	路徑節點數 Metric	循序號碼 Sequence Number	第一次相連時間 Install

DSDV通訊協定中每一個節點內都必須存有一個路由表

- 目的地(Destination)：可到達之節點。
- 下一個節點(Next Hop)：在到達某一目的節點之路徑上的第一個相鄰節點。
- 所需節點數(Metric)：到達目的節點中間所會經過的節點個數。
- 循序號碼(Sequence Number)：每個節點進行週期性資訊時所附上之號碼。初始時之編號是隨機產生，而後每次進行更新時號碼會加二。
- 第一次相連時間(Install)：各個節點第一次與其他節點進行通訊時所記錄之時間。

表格驅動式路由協定：
循序的目的地距離向量路由協定 (DSDV)

- 當網路拓撲結構有改變的時候，皆透過週期性的傳送廣播封包來告知新的路由資訊 。
- 這些路由資訊中不會改變的部分包含每一個節點第一次與其他節點通訊之時間 (Install)。
- 當某一節點在進行路由表更新時，對於具有相同目的節點之數條路徑：
  - 此節點通常會選擇具有最新循序號碼之路徑。
  - 若是遇到具有相同循序號碼之路徑，則將會選擇具有較少中繼節點數者。

表格驅動式路由協定：循序的目的地距離向量路由協定 (DSDV)

- 在動態網路中，擁有鏈結（Links）的節點可能會移動。
- 其鄰近之節點若是在經過一段時間內沒再收到此一節點的更新訊息時，其鄰近節點會認為此一節點已經離開，則這些鄰近節點會將各自的路由表中凡與此移動節點有關之路徑的路徑節點數設成無限大，並建立一個新的循序號碼將其廣播出去。
- 一般節點所發出的循序號碼都會被設定為偶數，而擁有無限大路徑節點數的循序號碼會被設定為奇數。

表格驅動式路由協定：
循序的目的地距離向量路由協定 (DSDV)

- 基本上每一個節點都會同時維護兩個路由表:
  - 一個是自己建立路徑時所需之路由表。
  - 另一個則是要廣播路由更新資訊給相鄰節點用的路由表。
- 每一個節點為了確保自己可以收到最好的路徑，都會保留之前的路徑資訊。
- 在DSDV通訊協定中，採用具較少中繼節點數者來更新自己路由表之方式來防止路徑迴圈發生。
- 路由表中的節點都會是以最短路徑（Shortest Path）的方式到達目的端節點。

表格驅動式路由協定：
循序的目的地距離向量路由協定 (DSDV)

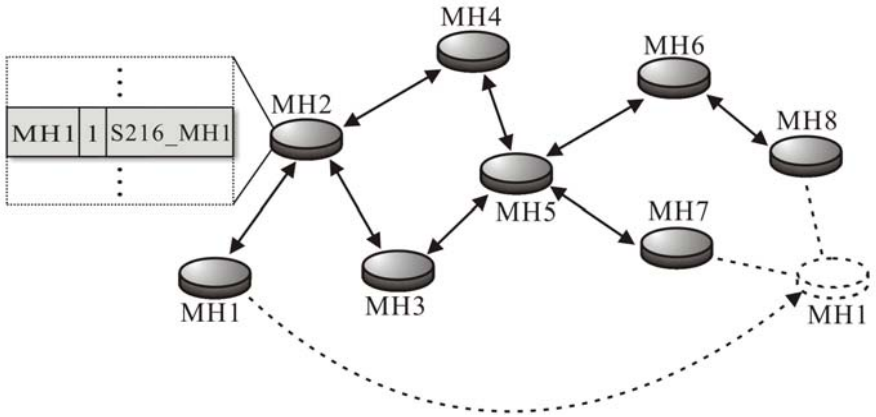
- 在DSDV通訊協定中，為防止各節點內之路由表不會常常被更改，而使網路較容易穩定下來 之作法：
  - 阻尼變動（Damping Fluctuations）：是描述如何去設置一個時間表，以防止路由表不斷的被改變。
    - 讓某些節點在接收到類似這樣的訊息封包後延遲將它送出。
    - 此延遲之時間最好大於兩個平均散佈之時間。
    - 所謂的平均散佈的時間為網路內每個節點傳送訊息給其它結點所需之時間平均。

表格驅動式路由協定：
循序的目的地距離向量路由協定 (DSDV)

- Example: 節點MH1在無線網路當中移動之情形。

目的地 Destination	下一個節點 Next Hop	路徑節點數 Metric	循序號碼 Sequence number	第一次相連時間 Install
MH_1	MH_2	2	$S216\_MH_1$	$T001\_MH_4$
MH_2	MH_2	1	$S328\_MH_2$	$T001\_MH_4$
MH_3	MH_2	2	$S546\_MH_3$	$T001\_MH_4$
MH_4	MH_4	0	$S602\_MH_4$	$T001\_MH_4$
MH_5	MH_5	1	$S312\_HM_5$	$T002\_MH_4$
MH_6	MH_5	2	$S056\_MH_6$	$T001\_MH_4$
MH_7	MH_5	2	$S128\_MH_7$	$T002\_MH_4$
MH_8	MH_5	3	$S056\_MH_8$	$T002\_MH_4$

MH4初始的路由表

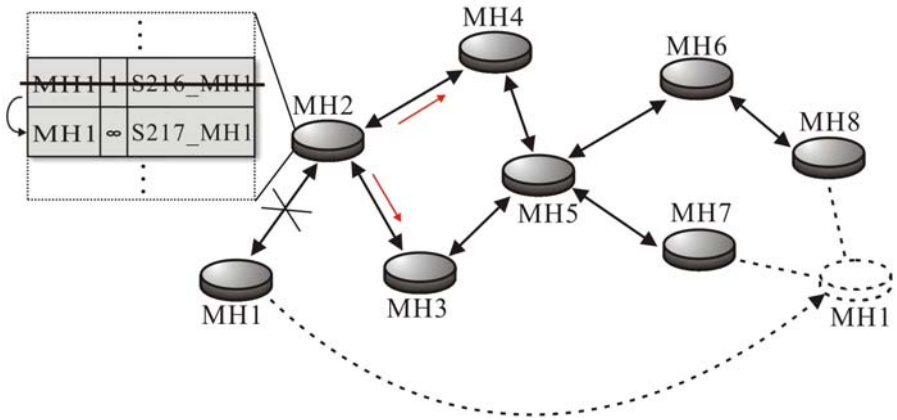


節點MH1在Ad Hoc網路中移動之情形

表格驅動式路由協定：
循序的目的地距離向量路由協定 (DSDV)

目的地 Destination	路徑節點數 Metric	循序號碼 Sequence number
MH_1	2	$S216\_MH_1$
MH_2	1	$S328\_MH_2$
MH_3	2	$S546\_MH_3$
MH_4	0	$S602\_MH_4$
MH_5	1	$S312\_HM_5$
MH_6	2	$S056\_MH_6$
MH_7	2	$S128\_MH_7$
MH_8	3	$S056\_MH_8$

MH4一開始所送出的更新路由資訊



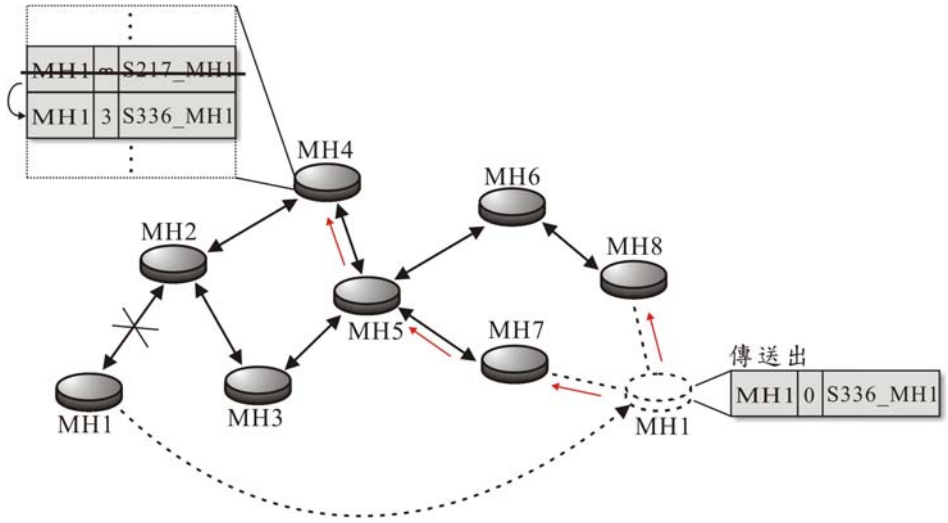
節點MH1移動至節點MH7與MH8附近

- MH2發現MH1離開後，其將自行更改到MH1的路徑節點數為無限大，並且更改其循序號碼為奇數。同時MH2會將這個消息廣播出去讓整個網路節點都知道MH2現在與MH1已經斷訊。

表格驅動式路由協定：
循序的目的地距離向量路由協定 (DSDV)

目的地 Destination	下一個節點 Next Hop	路徑節點數 Metric	循序號碼 Sequence number	第一次相連時間 Install
MH_1	MH_2	∞	$S217\_MH_1$	$T001\_MH_4$
MH_2	MH_2	1	$S328\_MH_2$	$T001\_MH_4$
MH_3	MH_2	2	$S546\_MH_3$	$T001\_MH_4$
MH_4	MH_4	0	$S602\_MH_4$	$T001\_MH_4$
MH_5	MH_5	1	$S312\_HM_5$	$T002\_MH_4$
MH_6	MH_5	2	$S056\_MH_6$	$T001\_MH_4$
MH_7	MH_5	2	$S128\_MH_7$	$T002\_MH_4$
MH_8	MH_5	3	$S056\_MH_8$	$T002\_MH_4$

節點MH4收到MH2發出與MH1中斷連線之封包後的更新路由表



節點MH1移動至MH7與MH8附近

表格驅動式路由協定：
循序的目的地距離向量路由協定 (DSDV)

目的地 Destination	下一個節點 Next Hop	路徑節點數 Metric	循序號碼 Sequence number	第一次相連時間 Install
MH_1	MH_5	3	$S336\_MH_1$	$T510\_MH_4$
MH_2	MH_2	1	$S448\_MH_2$	$T001\_MH_4$
MH_3	MH_2	2	$S666\_MH_3$	$T001\_MH_4$
MH_4	MH_4	0	$S722\_MH_4$	$T001\_MH_4$
MH_5	MH_5	1	$S432\_MH_5$	$T002\_MH_4$
MH_6	MH_5	2	$S176\_MH_6$	$T001\_MH_4$
MH_7	MH_5	2	$S238\_MH_7$	$T002\_MH_4$
MH_8	MH_5	3	$S176\_MH_8$	$T002\_MH_4$

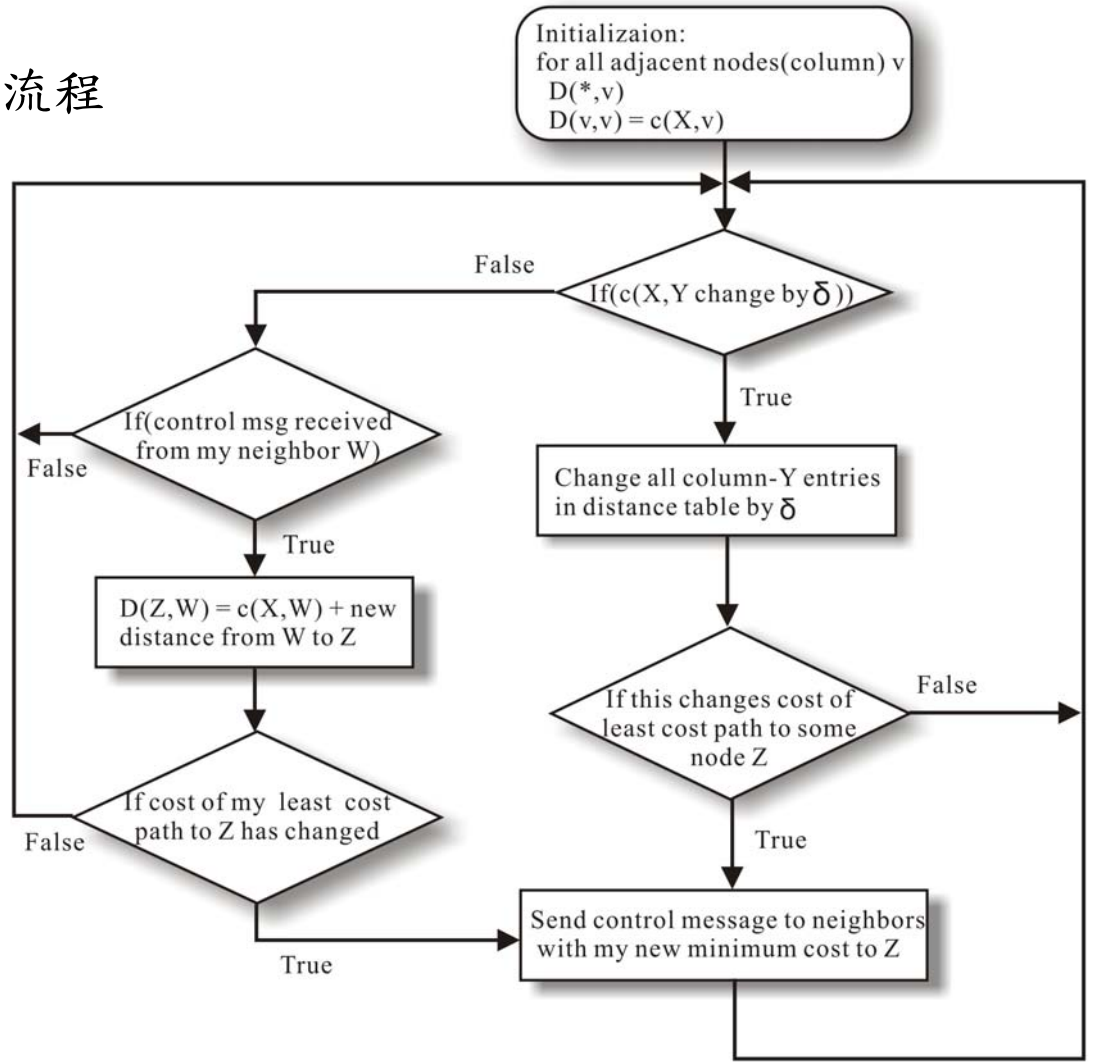
節點MH4再次更新後的路由表

目的地 Destination	路徑節點數 Metric	循序號碼 Sequence number
MH_4	0	$S722\_MH_4$
MH_1	3	$S336\_MH_1$
MH_2	1	$S448\_MH_2$
MH_3	2	$S666\_MH_3$
MH_5	1	$S432\_MH_5$
MH_6	2	$S176\_MH_6$
MH_7	2	$S238\_MH_7$
MH_8	3	$S176\_MH_8$

節點MH4在MH1移動至MH7跟MH8附近後所送出的更新路由資訊

表格驅動式路由協定：
循序的目的地距離向量路由協定 (DSDV)

DSDV協定的處理流程



表格驅動式路由協定：群首閘道交換路由協定 (CGSR)

- 以DSDV為基礎並結合叢集路由機制設計而成的。
- 在多重中繼段和移動式無線網路中，讓節點加入某一叢集中，並受到此叢集之群首（Clusterhead）的控制。
- 在叢集式架構中會使用分散式演算法（Distributed Algorithm），來選出某一個節點作為某一叢集之群首。
- 在一群首通訊範圍內的所有節點均可加入此一叢集，而成為此叢集之成員。
- 在叢集中之所有節點均可以與群首直接進行通訊。
- 一個叢集中之節點可分為三種角色：
  - 群首（Clusterhead）。
  - 閘道（Gateway）。
  - 一般節點（Node）。

表格驅動式路由協定：群首閘道交換路由協定 (CGSR)

- **演算法**：
  - 一個叢集的複雜度（Complexity）主要取決於「群首選擇」的方法：
  - 一般有兩種分散式叢集演算法可採用：
    - 最低識別碼叢集演算法（Lowest-ID Clustering Algorithm）。
    - 最高連通性叢集演算法（Highest-connectivity（Degree）Clustering Algorithm）。
  - 無線網路中，維護一個叢集最重要的準則是穩定性，倘若群首太常變更將不利於此叢集之效能。
  - 討論另一種演算法：LCC叢集演算法（Least Cluster Change Clustering Algorithm）。
  - 此演算法只有在兩種情況下才會導致群首的變更：
    - 當某一個群首進入另外一個群首的通訊範圍時。
    - 當某一個群首脫離所有叢集的通訊範圍時。

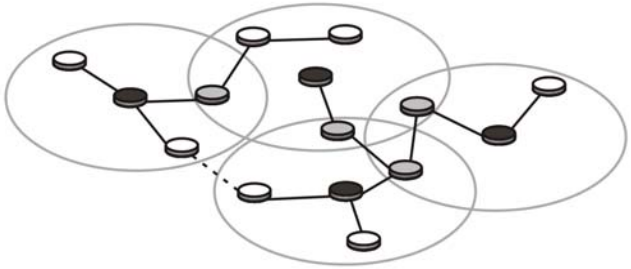
表格驅動式路由協定：群首閘道交換路由協定 (CGSR)

- LCC演算法：
  - 先採用「最低識別碼」（Lowest-ID）或「最高聯通性」（Highest-connectivity）叢集演算法，來創造一個初始的叢集。
  - 當某一個原屬於叢集A的非群首節點移動到叢集B時，則叢集A和B中之群首並不會改變。（只是叢集中之成員會變少或變多）
  - 若某一個非群首節點離開它的叢集，且沒有進入到其它叢集時（即指未進入其他叢集之通訊範圍內），則它將會成為一個新的群首，並形成一個新的叢集。
  - 當叢集A之群首節點離開叢集A並進入到叢集B時，則此一進入之群首節點將會和原本叢集B之群首節點互相競爭。再根據「最低識別碼」或「最高聯通性」叢集演算法（或其他較好的優先權方案）來競爭，勝出者將成為此叢集B的新群首。
  - 當叢集A之群首節點離開叢集A時，則此叢集A將會根據「最低識別碼」或「最高聯通性」叢集演算法，重新計算以產生新的群首。

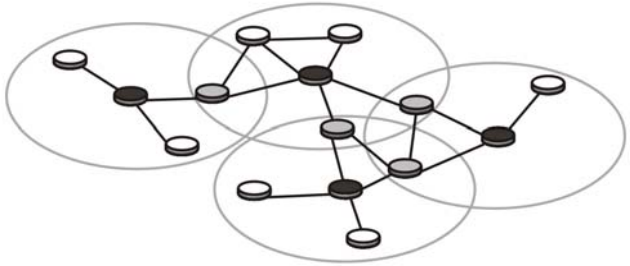
表格驅動式路由協定：群首閘道交換路由協定 (CGSR)

- 某兩個節點彼此停留在彼此的通訊範圍內且有穩定的訊號能源，則此兩個節點間會形成鏈結（Link）。
- 每個叢集都會有屬於自己的傳輸代碼（Transmission Code），且每一個節點都會記錄著自己所屬叢集的傳輸代碼。
- 兩個節點會因為彼此的傳輸代碼不同，而無法互相傳輸。
- 定義上述兩個使用不同代碼的節點，彼此間之鏈結為偽鏈結（Pseudo Links），並且從連線狀態中移除。
- 群首將會與叢集中的每一個成員進行鏈結 。

表格驅動式路由協定：
群首閘道交換路由協定 (CGSR)



在形成叢集之前的各節點鏈結情形



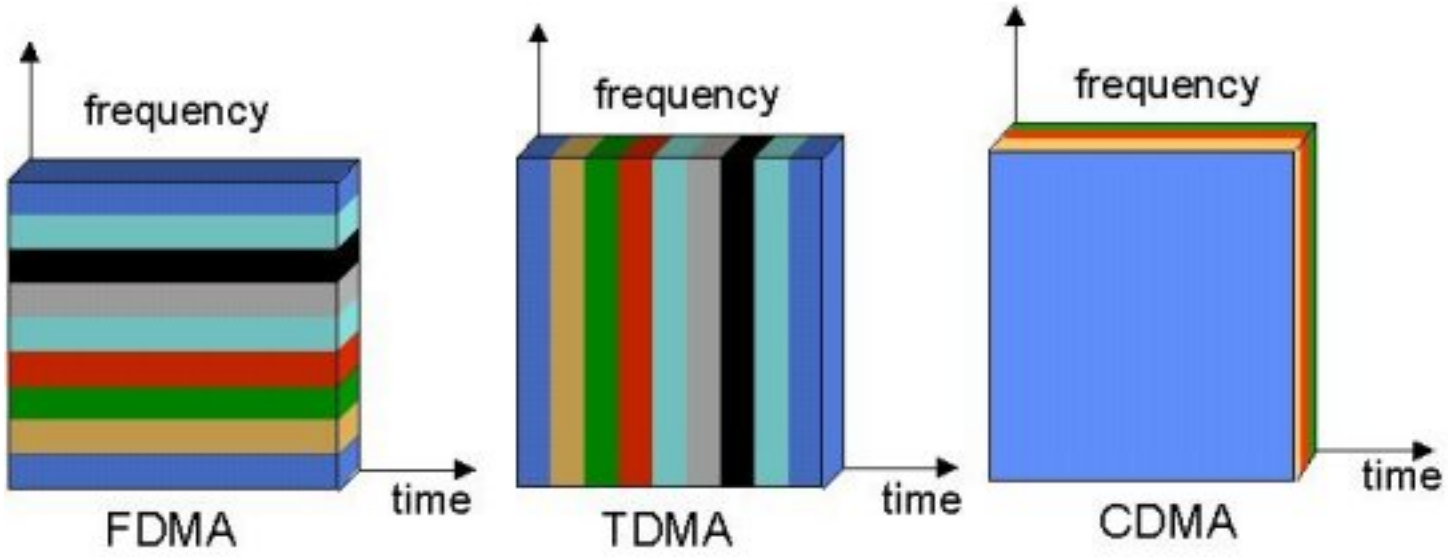
形成叢集後的各節點鏈結情況（黑點代表群首，灰點代表閘道節點，白點代表一般節點）

表格驅動式路由協定：群首閘道交換路由協定 (CGSR)

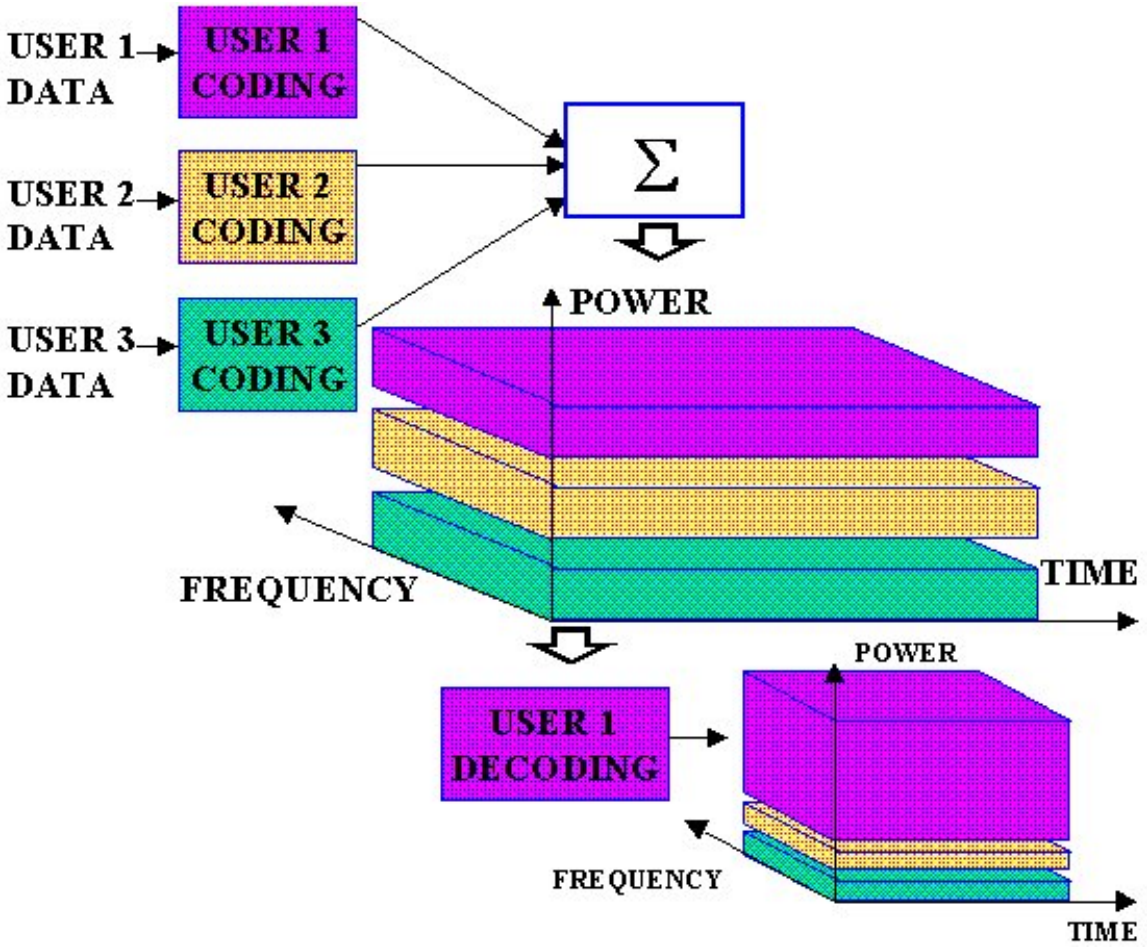
- 頻道存取演算法 ：
  - 相互交叉的叢集中，可藉由使用「分碼多重存取」（CDMA）之方式來增加相互交叉的叢集彼此間平行傳輸（Parallel Transmission）的效能。
  - 在叢集內部各節間點的運作，一般可使用「群首控制符記（Token）」的「輪詢協定」（即Polling）來依序分配頻道給其中某一個節點使用，以避免各節點間因傳輸競爭而產生的碰撞情況。
  - 安排給予群首傳輸優先權，使群首具有更多的傳送機會。
  - 每一個叢集中只有得到允許符記的節點可以使用被分配的傳輸代碼（即CDMA方法中的編碼）來存取頻道。
  - 群首需具有重新發出允許符記之能力，原因：
    - 在一些情況中允許符記可能會遺失。

CDMA

- In Code Division Multiple Access (CDMA), every communicator will be allocated the entire spectrum all of the time. CDMA uses codes to identify connections.

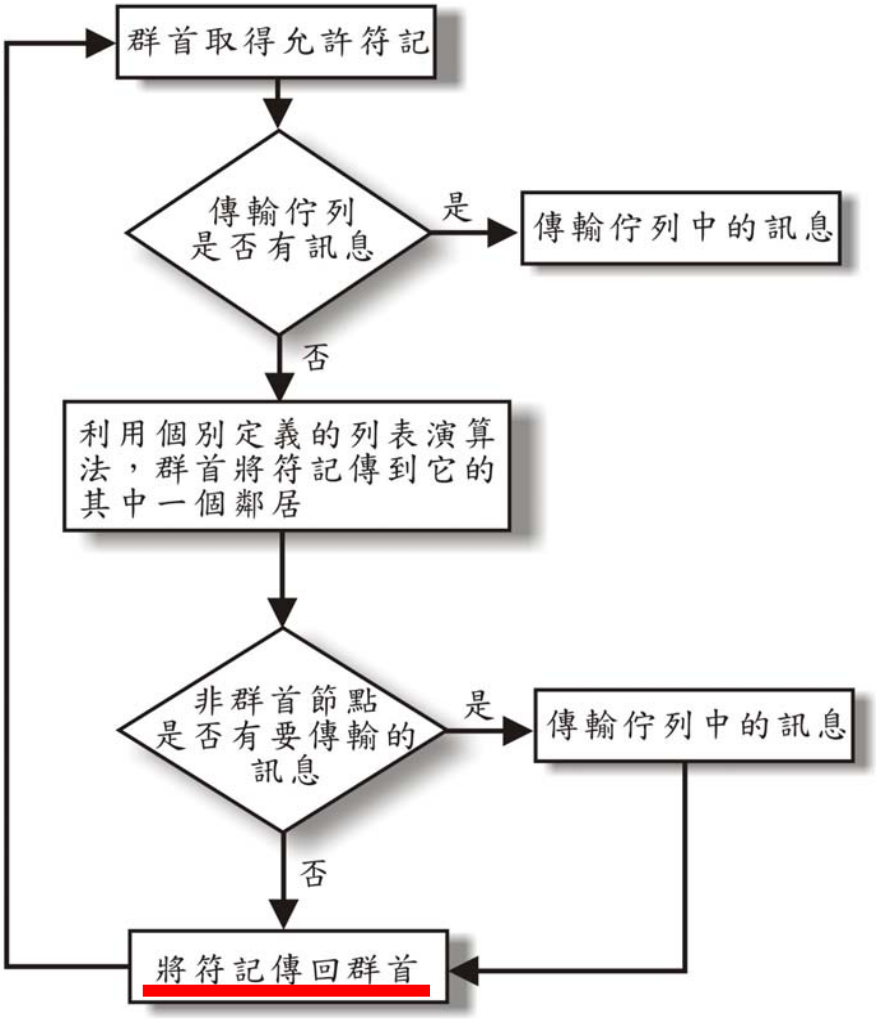


CDMA **(Cont.)**


USER 1 DATA
USER 1 CODING
USER 2 DATA
USER 2 CODING
USER 3 DATA
USER 3 CODING
Σ
POWER
FREQUENCY
TIME
USER 1 DECODING
POWER
FREQUENCY
TIME

表格驅動式路由協定：群首閘道交換路由協定 (CGSR)

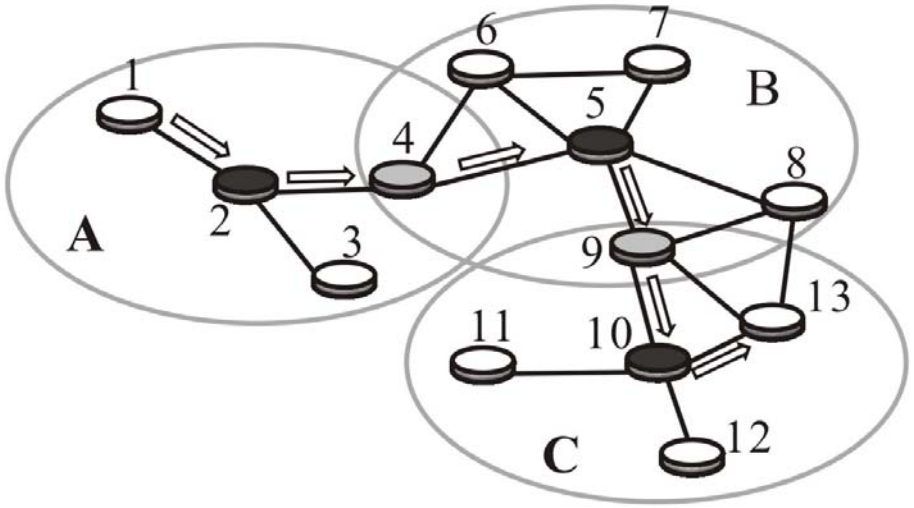
採用輪詢協定的頻道存取演算法



表格驅動式路由協定：群首閘道交換路由協定 (CGSR)

- CGSR路徑選擇：
  - 封包的傳送路徑只由群首節點與閘道節點交替建構而成。
    - C1, G1, C2, G2,… ,Cn
  - 最後到達目的地叢集的群首節點，再由目的地群首節點將封包轉送至目的地節點。
  - 其優點在於群首節點會有較多的傳送機會，而閘道節點則是唯一可在各叢集之間轉送封包的節點。
  - 每個節點有兩種數據結構：
    - 叢集成員表：
      - 描述每個目標節點所在叢集的群首。
      - 每個節點會使用DSDV協定週期性的與鄰近節點交換叢集成員表以更新表項內容，並且使用循序號碼來避免老舊路徑資訊的傳播。
    - 路由表：
      - 記錄了位於通往目的地群首路徑上的下一個節點。(群首或閘道)

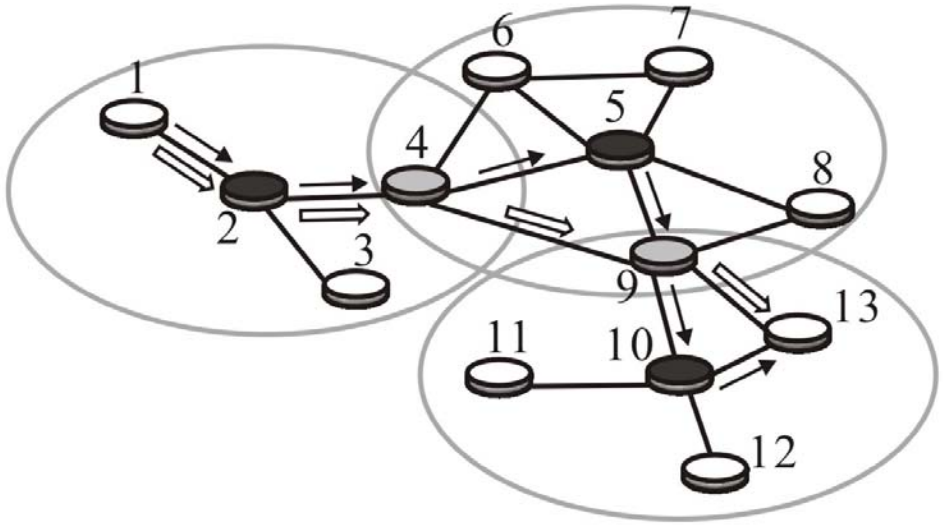
表格驅動式路由協定：
群首閘道交換路由協定 (CGSR)



CGSR的運作示意圖：由節點1傳送封包到節點13之範例

表格驅動式路由協定：
群首閘道交換路由協定 (CGSR)

- DSDV與CGSR的比較：
- 最主要的不同之處在於CGSR可以避免閘道對閘道的傳送。雖然增加路徑長度



- 不同方法會走不同路線，黑箭頭表CGSR方法，白箭頭表DSDV方法
- 模擬結果CGSR比DSVD有較少的平均延遲時間