應用基底排序之多播網路

詹景裕*[a]、王家祥[a]、高誌陽[a]、何嘉玲[b]
國立臺北大學電機工程學系[a]
臺北城市科技大學行銷與流通管理系[b]

摘要 —因網路普及後，使用者使用多播的機率增加，容易造成網路之壅塞，高吞吐量之多播網路設計亦行重要。本文提出之多播網路架構使用基底排序，以多播標籤(Multicasting Tag)及複製碼(Duplication code)的搭配，不需額外電路控制即達到複製封包與傳繞目的地的功能。本架構優點為控制機制與節點架構簡單易於實作，且顧及封包競爭公平性，並無排程問題造成輸出端閒置，因此輸出可達最高的吞吐量(Throughput)。單一封包繞徑之時間複雜度為 $O(\log N)$，單組封包時間複雜度最差為 $O(N)$，空間複雜度皆為 $O(N^2)$，其中 N 為輸入(出)之埠數。

一、 簡介

交換器常必須處理由一個發送源傳送資料給兩個以上的目的地，此傳送方式稱為多播(Multicast)，為了因應許多網路多媒體的應用，交換器不僅要能夠達到多播的功能，而且單一接收端也要能接收來自不同發送端的資料，具有此功能的交換器稱為多播封包交換器(Multicast switch) [3,8,10]。在多播式傳送的環境下必須考慮兩個問題，一個是封包複製(Duplicate)時機，在愈靠近輸出端做複製動作就愈能減少頻寬的浪費；另一個是有效快速地繞徑(Route)至目的地。在短時間內來自不同輸入端的封包湧向同一輸出端的時候，勢必會有碰撞(Block)的情況發生，必須透過適當的輸出排程來處理，這一直是利用多播方式所面對的重要議題。在資料持續不斷地進入交換器中，在有大小限制的頻寬內要複製和繞徑，使得不同時間進入交換器的資料需要同時處理，勢必要有適當的機制加以管理讓交換器能在最節省硬體成本、最短時間內以及最低封包遺失率完成工作。

本文利用基底排序的特性搭配樹狀節點建構基底多播網路架構，雖在節點複雜度上達 $O(N^2)$，但少了排程器、控制器以簡化系統控制機制，並大幅加快處理速度，從輸入端至輸出端之深度為 $O(\log N)$。

本文於第二節文獻探討說明主要的多播網路架構分類，第三節則介紹基底多播網路的架構與方法，並舉實際的範例作說明，第四節作效能分析說明其時間及空間複雜度，最後結論作總結。

二、 相關研究

本節將介紹幾個多播封包交換器之相關文獻回顧，分別針對其節點複雜度、時間複雜度和佇列所需空間來比較探討。節點複雜度考慮架構中所有的交換節點數目；時間複雜度計算封包進入交換器到輸出端的時間，其中若有額外的控制電路將影響處理速度；佇列所需空間是用來儲存競爭輸出失敗之封包以降低遺失率。

多播封包交換器以佇列型態來分類，可分輸入佇列交換器[1,9]、輸出佇列交換器[11]、共享佇列交換器[5]。輸入佇列交換器在有競爭輸出發生時，封包儲存在輸入佇列導致後來的封包即使沒有競爭輸出也會被擋住，這種狀況稱為線頭壅塞(Head-of-line blocking)。改善方法為加入排程器解決堵塞問題，因輸入佇列交換器需在輸入端做排程工作再採用分離式複製網路(Copy network)與繞徑網路(Routing network) [7]，因犧牲在排程器處理上的時間很多，時間複雜度為 $O(N)$以上，節點複雜度為 $O(N\log N)$，佇列所需的空間為 $O(N)$。

輸出佇列交換器儲存封包是在複製之後，佇列所需空間為 $O(N^2)$，它避開在輸入端的阻礙，也省去了排程器所耗的時間，時間複雜為 $O(\log N)$，但爲避免複製過程的碰撞需較多節點，所以節點複雜度到達了 $O(N^2)$。

Knockout [11]的架構讓所有輸入端跟輸出端都有專門的連結管道，它利用廣播(Broadcast)的方式將封包傳送到所有輸出端，輸出端的判斷元件再判斷所傳達的封包是否接收，再將封包傳送到集中器參與競爭輸出，透過移位器將同一時槽內可進入輸出佇列的封包限制在 L 個($<<N$)，配合重傳機制可將佇列空間減少為 $O(LN)$。

共享佇列(Shared queue)交換器[5]在做複製動作之前，透過中央控制器將輸入端的封包儲存至共享佇列內，控制器再為儲存在佇列內的封包產生資訊檔，此資訊包含儲存佇列的位址、封包為多播或單播(Unicast)和目的輸出端等等，然後再將封包傳送給共享匯流排，最後傳達輸出端完成工作。因為需要中央控制器來處理，在處理上會犧牲許多時間，時間複雜度為 $O(N)$以上，節點複雜度為 $O(N)$ 還要加上額外的中央控制器，佇列所需求空間為 $O(N)$。

縱橫式多播交換器[6]的左方的任一輸入端同一時槽內只能有一固定長度封包進入輸入佇列，接著等候依照間隔時槽數再進入架構內部的節點來執行複製與繞徑的動作，最後在由下方的輸出端輸出。因此架構運作在時槽控制的機制下，輸入封包之形式需為固定長度，所以必須將封包的長度切成一樣，再透過多播標籤產生器給予原始封包所附有之繞徑目的地的資訊，繞徑方式採用自我繞徑(Self-routing)，簡單來說一開始輸入的原始封包只會利用往右的路徑傳送，而複製後的封包則利用往下行走的路徑傳送到輸出端為止，而此方法為了避免碰撞造成遺失的方法，會將不同方向傳遞的封包設定不同的優先權，而優先權較低者將會被儲存至節點內的緩衝器等待下一個時槽傳送，其節點複雜度為 $O(N^2)$，時間複雜度則為 $O(N)$。

Corresponding author:roger2234@gmail.com
DOI:10.6159/IJSE.2014.(4-2).65

三、 網路架構

3.1 應用基底排序之多播網路架構

本文在構想此新多播架構時使用了由詹景裕等人所提出的基底排序(Radix sorting)的架構[4]，成功地降低封包在架構中運行的時間，該文僅用該架構進行一項運算即排列(Permutation)。本文之作法將利用此架構來達到單一跟單組封包的多播，在該架構每個節點改變複製及繞徑兩種運算來達成多播網路架構功能，未來將改進此概念架構得以讓我們建構一個新的網路多播系統，整體架構如圖 1 所示，圖 2 為 4 輸入端之架構範例。

基底排序之多播網路架構(Multicasting Networks Base on Radix Sorting) 由左方 *N* 個輸入端以及右方 *N* 個輸出端組成，每一條路徑行經至節點後將會根據其自身所帶有的多播標籤(MT, Multicasting Tag) 來決定在此節點進行的動作，在左半邊的階段將決定是否複製封包，在右半邊的階段將會決定是否繞徑。左方的任一個輸入端同一時間內只能有一個資料或一組資料進入，依間隔時槽數進入架構，在左右的節點執行複製與繞徑，在中間進行路徑規劃，最後從右方的輸出端輸出。

此架構運作藉由時槽機制來控制，其輸入封包形式為一固定長度，故須使封包的長度一樣再透過多播標籤附上目的地資訊才能得知目的地輸出端位址，再經由自我繞徑才能完成傳輸。如圖 3 所示，Activity Bit 為 1 表示該時槽內有封包到達，多播標籤為 *N*-bits，表示該複製幾個封包傳送到哪些目的地輸出端， 以二進位表示為 b_{N-1}.....b_1b_0，如 b_0 為 1 表示有封包要傳送到輸出端 0。

在有了多播標籤和控制封包進入架構的機制後，一開始將準備進入輸入端的封包放入輸入端佇列根據設定的間隔時槽控制進入架構。封包進入架構後每過一個時槽就往右方一個節點前進，直到最右方節點為止。在封包每進入一個新的節點，多播標籤必須與節點的位置標籤對照，作為封包是否該複製或繞徑的判斷依據。

簡單來說，一開始輸入的原始封包將會根據複製碼來決定每個階段節點內的動作，如在這個階段中，代表要複製的 bit 為 1 的話則進行複製。在架構中路徑分成上下兩半部，故每個節點會有 Upper half 以及 Lower half 兩個數值，根據這兩個數值的不同將決定是否複製以及行走的路徑。依此法炮製每個階段則最後傳送到輸出端為止，所以在此架構中原始封包跟複製後的封包在繞徑過程中是各自獨立的，不會有碰撞的現象發生。

3.2 節點架構

在上述交換器的架構中，節點設計內大部分具有切換器、佇列或多個暫存器，因此這些架構中有許多的控制動作必須在節點內完成，而本文的架構節點也扮演一個重要的角色，不同於其他類型的交換器架構，本架構由於設計簡化，成本相對減少，而封包經過節點內的運作也不需經過多餘的單位像是佇列或者是暫存器，滯留時間也相對減少，這也就是本架構中一個非常重要的優勢，利用相對少的節點內部單位就達成所要的目標。

本文所述的架構圖中每一分歧或是滙入點都有一個節點。在架構左半每一條路徑都將一分為二，即在這半邊的節點為一進二出的架構，資料進到節點會根據本身所帶有的多播標籤來決定在這個節點裡面是否複製資料，決定之後便帶著資料往下一階段前進。而右半邊每

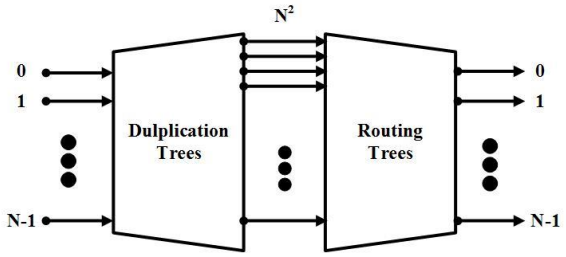


圖 1. Architecture of a *N*-input radix multicasting network

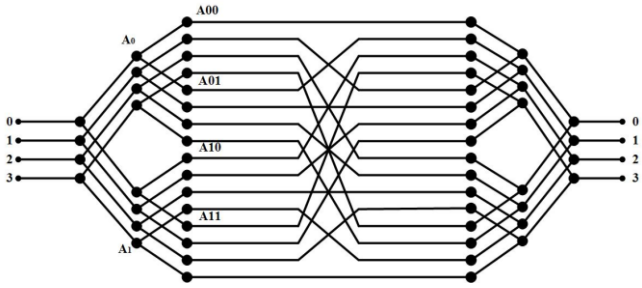


圖 2. Architecture of a 4-input radix multicasting network

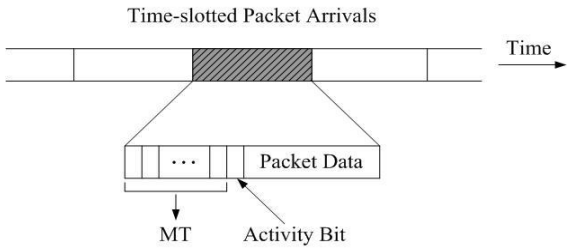


圖 3. 輸入封包之形式

一條徑都將由兩條路徑滙集成一條最後滙成輸出端，故這半邊的節點架構為二進一出的架構，資料進到這的節點一樣根據本身所帶的多播標籤來決定是否繞徑及將行走的路徑。而在二進一出的節點內有個 Switch 的架構，Switch 在上下同時都有封包的時候會根據上一回合最後允許的路徑是哪一條路徑來決定這一回合另一路徑優先權比較高，以第一回合為例，由於沒有上一回合所以此時優先權的控制為空，於是節點會使上方的封包優先權較高而使其先通過，通過之後立刻切換使下方的封包通過，如此上下交替能使得每一條路徑都被公平地傳輸；但當 Switch 前的兩條路徑只有一方有封包要進入時，將直接切換至該路徑，使得封包立即通過不需等待。也就是當需要排隊的時候 Switch 會公平地在兩條路徑中切換，不需排隊時會立刻讓封包通過。即封包在節點進行自我繞徑而不需中央控制，節點能自行切換決定放行方向。此做法好處是不需額外中央控制單位，省下額外的成本，不需等候中央控制單位發送控制訊息的時間。由於此設計使得在傳送封包時能夠快速且有效率地運用任何空間，而這樣的設計將不會造成一方節點要求通過時切換器正允許沒有封包的另一方路徑通過的情形，而能使節點使用率達到最高同時兼顧公平性，兩種節點結構如圖 4 所示。

3.3 多播標籤(Multicasting Tag)

多播標籤是個能夠將封包的目的地清楚表達且能夠簡化

架構運作的一組標籤。藉著將封包資訊放進多播標籤內，配合預先處理(Preprocessing)產生出複製碼(Duplication code)，將能夠大量節省每個封包在每一階段經過節點要決定方向時所耗費的大量時間。而多播標籤為一組 N-bit 的 Tag，N 表示輸出與輸入的數目，該 bit 為 0，則表示該封包沒有要送到該目的地，反之若 bit 等於 1，則該封
包將會多播至該目的地，且該組多播標籤將為第一組複製碼再經由 OR 運算產生出其他的複製碼，圖示 5 為 8 bits 的範例。

3.4 複製碼 (Duplication code)

繼 3.3 節多播標籤所述，將封包的資訊放至多播標籤內後，便可用 OR 運算得到一組複製碼(Duplication code)，複製碼將會決定每個封包在每一階段節點所進行的行為，以下將介紹複製碼的運作過程。

為了計算出每一個封包的複製碼，必須從多播標籤著手取得封包的目的地資訊，其中 b_{n-1} ... b_1b_0 為輸入封包的多播標籤，當 $b_i = 1$ 時表示這個封包將被傳送至輸出端 i，利用這組 n-bit 的多播標籤，我們將可以產生一組長度 (2n-2)-bit 的複製碼。

其中在第 j 個階段及 u_i^j 與 l_i^j 在區間 $0 \le i \le 2^j - 1$，$0 \le j \le (\log_2 n) - 1$ 時，$u_i^j = b_{n/2-1}$ OR … OR b_1 OR b_0，而 $l_i^j = b_{n-1}$ OR … OR $b_{n/2+1}$ OR $b_{n/2}$ 為在第 j 個階段時從左至右、從上至下的第 i 個分配器的複製碼。也就是說此封包在 $b_{n/2-1}…b_1b_0$ 這段標籤內的值不全為零時，將被繞徑至節點的上半部，也就是在 $u_i^j = 1$ 的時候。

同理這個封包在 $b_{n-1}…b_{n/2+1}b_{n/2}$ 這段標籤內的值不全為零時，也就是 $l_i^j = 1$ 時，將被繞徑至節點的下半部。從第一個階段到最後一個階段，重複做這段多播標籤的編碼，我們可以得到一組長度為(2n-2)-bit 的複製碼：$u_0^0l_0^0/u_0^1l_0^1u_1^1l_1^1/u_0^2l_0^2u_1^2l_1^2u_2^2l_2^2u_3^2l_3^2/$…也就是每一個封包將會自帶一組長度為 n-bit 的廣播標籤，並預先進行每一個階段的 OR 運算，使得每個階段都減少邏輯運算時間。

3.5 繞徑樹 (Routing tree)

本文架構的右半邊，如節點設計所述，封包在此階段進行自我繞徑，即節點不需透過中央控制來決定封包行走的路徑，而是封包所自帶的 Tag 以及節點決定。當封包經過左半邊的節點決定該封包要複製至哪些路徑後會經過架構中間的路徑規劃，接著進入繞徑樹階段，每一節點如同二元樹的根(root)，又同時是右邊節點的子樹(subtree)，由於此階段封包在進行自我繞徑，故每個二元樹就如同多個繞徑樹。封包行經節點時將藉由節點架構中所提出之方法來決定封包行走路徑或者是否等待。

3.6 實例說明

3.6.1 單一封包範例

以 $N = 4$ 為例，假設一封包 P_A 要多播到第 1、2、3 輸出端，多播標籤為 1110，此多播標籤同時也為第一組複製碼、第二階段複製碼；將第一組複製碼作 OR 運算，將得到：$u_0^0 = b_0$ OR $b_1 = 1$，$l_0^0 = b_2$ OR $b_3 = 1$，由此可知此範例複製碼為：$u_0^0l_0^0/u_0^1l_0^1u_1^1l_1^1 = 11/0111$。

該封包由第一個輸入端出發進入架構後遇到的第一個節點，將會在此時判斷複製碼為第一階段複製碼為 11，表示上下半部都有封包輸出，即在此將複製封包，

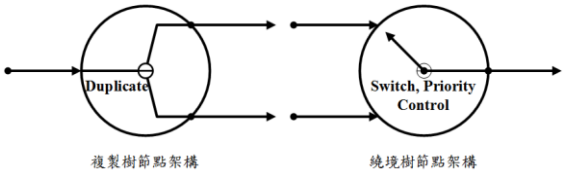


圖 4. 內部節點結構

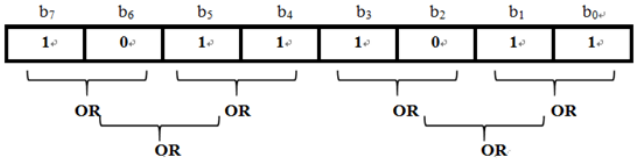
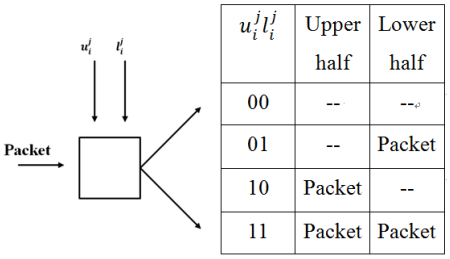


圖 5. 多播標籤格式及運算規則範例



$u_i^j l_i^j$	Upper half	Lower half
00	--	--
01	--	Packet
10	Packet	--
11	Packet	Packet

圖 6. 複製碼運作說明

然後將輸出到上下兩條路徑。接著封包各行經至第二階段，原行走在上半部路徑的封包將會根據 $u_0^1l_0^1 = 01$ 此複製碼判斷將要複製及行走的路徑，即上半部不會有封包通過，而下半部有封包通過；而原行走在下半部路徑的封包將根據 $u_1^1l_1^1 = 11$ 來判斷。此時此封包已完成左半的複製階段，將通過中間的路徑規劃到右半邊的繞徑樹部分，一進入繞徑樹封包沿原路徑進入節點，將依照節點架構中的方法來決定封包如何行走。在此例，P_A 封包經過了左半邊複製碼過程複製三份，經中間路徑規劃後進入右半邊的繞徑樹階段，會出現在第 2、3、4 個節點後進入輸出端也就是第 2、3、4 個輸出端，單一封包將不會有等待排隊的情形，便完成封包傳遞且無任何資料遺失，其結果如圖 7 所示。

3.6.2 單組封包範例

單組封包資料運作的過程，如同單一封包運作，但各個封包同時平行運作，如圖 8 所示。值得注意的是在進入繞徑時因各封包平行進行，便可能會產生等待。在節點設計所述，封包在節點將會根據複製碼來決定路徑，當出現同時有兩封包要求進入時，節點將會自行決定此時使哪條路徑之封包通過，如圖 9 所示。

在此例共 4 個封包將要複製 10 次到達 4 個輸入端，只需三個回合便能將封包全部達目的地而沒有任何封包遺失，這便本法在單組封包資料運作所能達到的優勢。

四、 效能分析

本文目前僅考慮單一及單組資料，得到下列定理：

定理 1：本架構在輸入端及輸出端皆為 N 時，其單一封包多播時間複雜度為 $O(\log N)$而單組封包多播時間複雜度為 $O(N)$。

證明：在單一資料時無任何等待，封包經過每個節點的時間為左半邊複製樹的 log N 加上右半的 log N，共最多 2log N 時間，即單一封包多播時，時間複雜度最差為

$O(\log N)$。預先產生複製碼的階段，資料 N-bit 時僅需 $\log N$ 的時間即產生複製碼，時間複雜度= $O(\log N)$。單

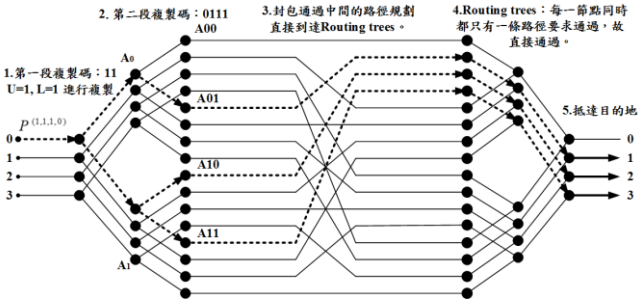


圖 7. 單一封包複製及繞境流程

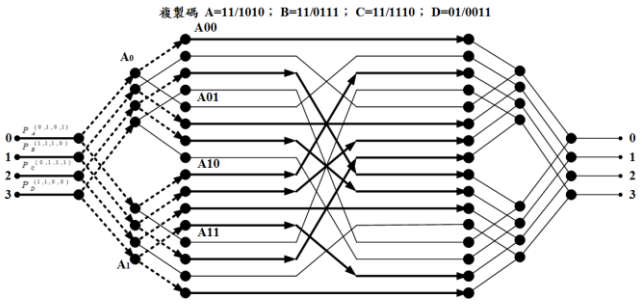
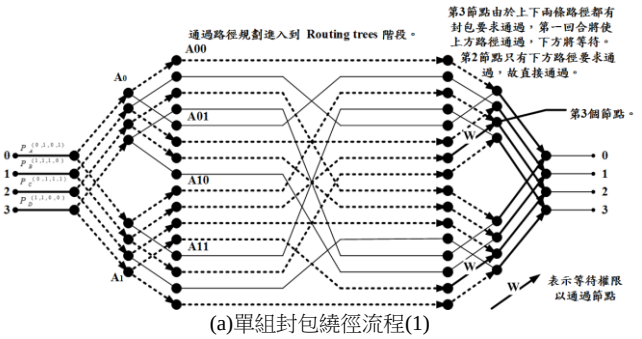
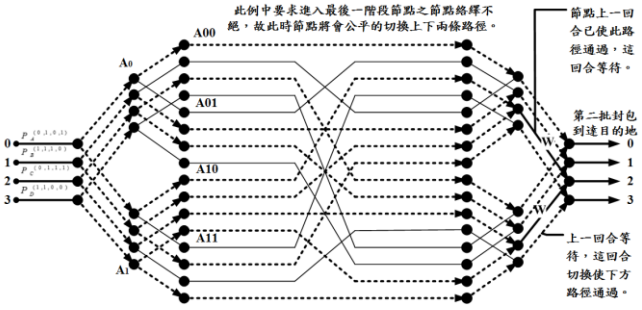


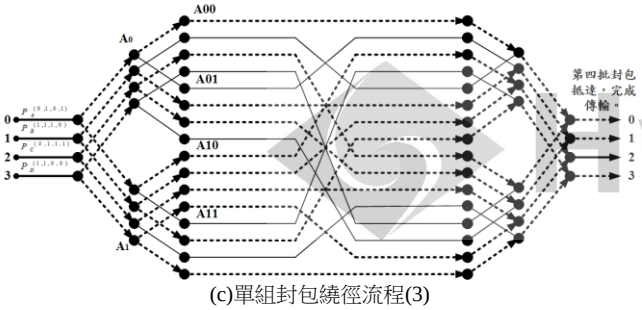
圖 8 單組封包複製流程



(a)單組封包繞徑流程(1)



(b)單組封包繞徑流程(2)



(c)單組封包繞徑流程(3)

圖 9. 單組封包繞徑流程

組多播在最差情形下需要輸出時間為：N^2 個 Dulplication / N 個輸出端 = N，即在 N 個時間內消化完所有封包。

定理 2：本架構在輸入端及輸出端皆為 N 時，其空間複雜度為 $O(N^2)$。

證明： N 個輸入端在架構中需 N^2 個節點，在前級的節點數需 $2N$ 個，即左半邊節點數需 $2N+N^2$ 個，右半邊亦然，故空間複雜度為 $2(N^2 + 2N) = O(N^2)$。

觀察 1：根據 Throughput(吞吐量)的定義[2]：輸入端流量滿載時，平均總輸出佔平均總輸入的比例。在本文觀察之結果，單組封包多播滿載時，不會因封包競爭而造成輸出端出現停頓，其 Throughput 為：

$$\frac{\text{最多}N^2\text{個 Dulplication (輸入端)}}{N\text{個輸出端}} = N$$

依據 Maximum Throughput 定義，為輸出端的最大利用率，輸出端為 N，在單組封包情況， N 個輸出端滿載時，因本架構不會有任何的閒置故輸出端將魚貫不絕地輸出，其值為 N，最大利用率為：

$$\frac{N\text{個封包輸出}}{N\text{個輸出端}} = 100\%$$

由於在單組亦或是未來發展至多組封包傳輸時，若使用節點數為 $N \log N$ 之網路架構則其 Data loss rate 將會極高不敷多播產生 $O(N^2)$個封包之使用，故本作法將節點複雜度提升至 $O(N^2)$，來避免這樣的情形發生。且本架構 $\log N$ 之網路深度以及其接踵而至之封包的傳輸方式可有效改進 Throughput。

五、 結論

本文方法與前面介紹的架構相比，優點在控制機制與節點的架構簡單容易實行，封包的競爭公平性佳，沒有排程問題造成輸出閒置，在單組封包多播時，不會浪費等待時間，其輸出最大利用率為 100%，網路深度為 $O(\log N)$，為多播網路中最短者。

參考文獻

[1] M. Andrews and L. Zhang, “Achieving Stability in Network of Input-Queued Switches,” *IEEE/ACM Trans. on Networking*, vol. 11, pp. 848-857, Oct. 2003.

[2] H. Jonathan Chao, and Bin Liu, *High Performance Switches and Routers*. Wiley-Interscience, New Jersey, 2007.

[3] W. T. Chen, C. F. Huang, Y. L. Chang, and W. Y. Hwang, “An Efficient Cell-Scheduling Algorithm for Multicast ATM Switching Systems,” *ACM /IEEE Trans. on Networking*, vol. 8, no. 4, pp. 517-525, Aug 2000.

[4] C. Y. (G. E.) Jan and A. Y. Oruç, “Fast self-routing permutation switching on an asymptotically minimum cost network,” *IEEE Transactions on Computers*, vol. 42, pp.1469-1479, 1993.

[5] J. Garcia-Haro and A. Jajszczyk, “ATM shared-memory switching architectures,” *IEEE Network* , vol 8, no. 4, pp. 18-26, July 1994.

[6] G. E. Jan, S.-W. Leu, W. Liou, and S. I. Chen, “FPGA implementation of a multicasting crossbar switch,” 2009 *International Conference on Microelectronics*, pp. 177-180, 2009.

[7] T. T. Lee, “Nonblocking Copy Networks for Multicast Packet Switching,” *IEEE JSAC, 1988*.

[8] Li, Y. Li, J. Wu, S. Su, and J. Yu, “ESM: Efficient and Scalable Data Center Multicast Routing,” *IEEE/ACM Trans. on Networking*, vol. 20, no. 3, pp. 944-955, June 2012.

[9] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, “Multicast traffic in input-queued switches: optimal scheduling and maximum throughput,” *ACM/IEEE Trans. on Networking*, vol. 11, no. 3, pp. 465-477, June 2003.

[10] Jonathan S. Turner, “Design of a Broadcast Packet Switching Network,” *IEEE Trans. on Communications*, vol. 36, June 1988.
[11] Y. S. Yeh, M. Hiuchyj, and A. Acampora, “The Knockout Switch: A Simple Modular Architecture for High-Performance Packet Switching,” *IEEE Journal on Selected Areas in Comm.* , vol. 5, pp. 1274-1283, Oct 1987.