實作十六埠樹狀多播封包交換器

Implementation of a 16-port Tree-based Multicast Packet Switch

詹景裕

國立台灣海洋大學

資訊工程學系

基隆市北寧路二號

Email:b0199@mail.ntou.edu.tw

TEL:(02)24622192 轉 6211

羅天一

國立台灣海洋大學

電機工程學系

基隆市北寧路二號

Email:kb8@ind.ntou.edu.tw

摘 要

本篇論文針對輸出佇列(output queues)之多播(multicast)封包交換器，基於複製(duplication)與競爭(contention)的觀念提出 2 種設計。設計 1 使用分時多工(timing division multiplexing)與管線式(pipelining)方法減少節點複雜度，其佇列所需求記憶空間為 $O(N^2)$，時間與節點複雜度分別為 $O(N)$ 與 $O(N\sqrt{N})$。設計 2 目的在限制每個輸出埠在一個時槽內可接受的封包數目為變數 $L(<< N)$ 個，並以新增之高級集中器(hyperconcentrators)來減少設計 1 佇列集的封包緩衝器(buffers)數目，其封包複製方法與設計 1 相同，佇列所需求記憶空間為 $O(LN)$，時間與節點複雜度分別為 $O(L\sqrt{N})$ 與 $O(LN)$。

在 VLSI 實作方面，我們使用硬體描述語言(VHDL)來完成晶片設計。實作模擬受限於 FPGA 元件 APEX 20K400EBC – 1X 大小，設計 1 實作 16 埠之多播封包交換器，晶片工作頻率為 50 MHz，每個埠可達到 95 Mbps 封包傳輸率。設計 2 則尚無實作。

關鍵詞：多播、封包交換器、輸出佇列、高級集中器、VLSI、VHDL、FPGA

一、緒論

近年來，網路上許多應用程式(如遠端視訊會議 video conferencing)，需要讓發送者傳送封包(packet)給一群接收者，此稱為多播(multicast)，以往這類傳送受限於網路中交換器(switch)，或路由器 (router)不具備多播功能，導致網路頻寬的浪費。為了改善此現象，實有必要發展具備多播功能的交換器，它能夠讓封包在傳輸過程中維持一份數量，等到達接收者的時候再行複製封包，如此便不會浪費網路頻寬。因此許多研究人員提出許多種 (N, N) 多播封包交換器[1-3, 6-18]。小括號 (N, N) 表示意義為：第一個變數表示輸入埠數目，第二個變數表示輸出埠數目。

當我們研讀這些已被提出的多播封包交換器時，依佇列放置於交換器中的位置來分類，可分為輸入佇列(input queues)、輸出佇列(output queues)以及共享佇列(shared queue)之交換器。

本篇論文針對輸出佇列之多播封包交換器，基於複製與競爭的觀念，封包先經過複製後暫存在佇列，再從佇列經競爭輸出，於是我們提出 2 種設計，分別為設計 1 (又稱為縮小樹狀多播封包交換器(Shrunk Tree-based Multicast

Packet Switches))與設計 2 (又稱為含集中器之縮小樹狀多播封包交換器(Shrunk Tree-based Multicast Packet Switches with Hyperconcentrators)。

在實作方面，受限於 FPGA 元件 APEX 20K400EBC – 1X 大小，設計 1 預計實作 (16,16) 多播封包交換器，而設計 2 則無實作，然後設計方法為使用硬體描述語言 VHDL (Very High Speed Integrated Circuit Hardware Description Language)來撰寫這設計 1 交換器之硬體電路，最後利用 FPGA (Field Programmable Gate Array)技術實作出來。

二、設計 1：縮小樹狀多播封包交換器

設計 1 使用分時多工與管線式方法減少設計 1 節點複雜度。分時多工意義為選擇樹從多個輸入封包選擇其一進入到複製樹，達成多個封包共同使用相同複製樹，減少節點複雜度，管線式則可加快封包複製速度，還有為避免封包因等待被選擇樹選擇而遺失，我們使用佇列集來暫存這些在等待的封包。設計 1 架構如圖 1 所示，組成元件包含了左邊的 n 個選擇樹(Left Selection Trees，簡稱 $LST_k\,(0 \le k \le n-1)$)、左邊的 n 個複製樹(Left Duplication Trees，簡稱 $LDT_k\,(0 \le k \le n-1)$)、右邊的 n 個選擇樹(Right Selection Trees，簡稱 $RST_k\,(0 \le k \le n-1)$)、右邊的 n 個複製樹(Right Duplication Trees，簡稱 $RDT_k\,(0 \le k \le n-1)$)、n 個佇列集(Set of Queues)，與 N 個輸出佇列。m 為 LST_k 輸入埠數目。

2.1 設計方法

設計 1 讓多個輸入封包共同使用相同複製樹，以便降低複製樹節點數目。複製樹可分成 LDT_k 與 RDT_k 兩部份，LST_k 從 m 個輸入封包選擇其一進入到 LDT_k，而 RST_k 則從暫存在佇列集中的封包選擇其一進入到 RDT_k。佇列集為利用 n 個各自獨立的佇列，可同時對來自 LDT_k 所複製出來的 n 個封包做暫存，佇列集需要 mn 個封包緩衝器，才能避免封包因等待著被 RST_k 選擇而遺失。輸出佇列則為對抵達目的輸出埠封包做暫存，輸出佇列可暫存的封包越多，封包因輸出競爭而遺失的機率越小。

定理 1：設計 1 佇列集之佇列若有 m 個封包緩衝器，封包便不會因等待被 RST_k 選擇而遺失。

證明：在一個時槽內，LST_k 最多只能選擇 m 個封包進入到 LDT_k，再經由 LDT_k 到達佇列集之佇列，所以佇列集之佇列只要有 m 個封包緩衝器，封包便不會因等待著被 RST_k 選擇而耽誤到傳送至下一級的時間，最後遺失。

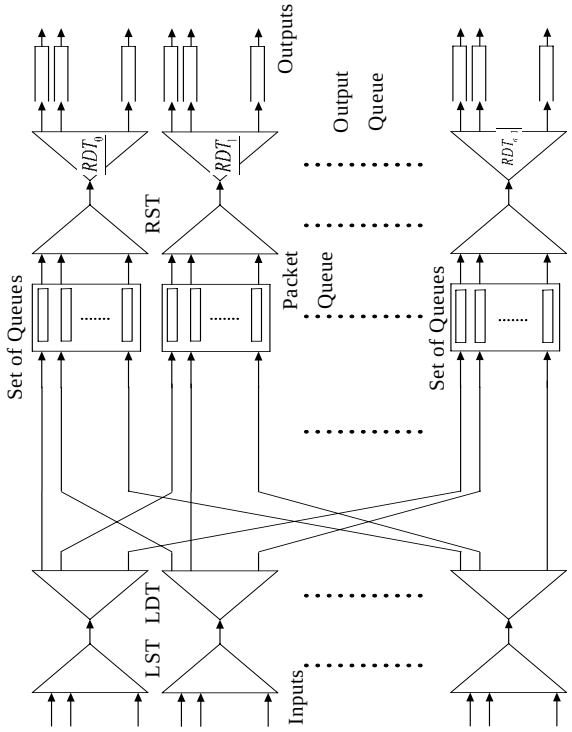

圖 1　(N, N) 設計 1

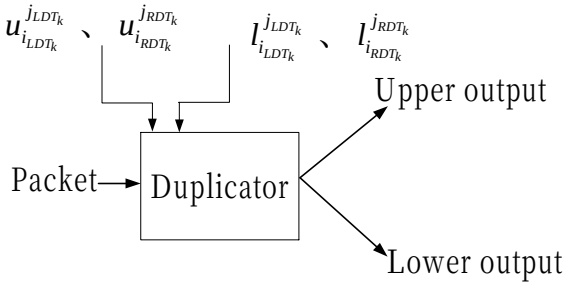

圖 2　複製碼

這邊要注意到，在設計 1 採用管線式傳繞運作下，多個封包會同時往 LST_k、RST_k 之根節點前進而導致阻塞，為避免封包因此遺失，LST_k、RST_k 之內部節點都需利用緩衝器來暫

存封包。

2.2 元件介紹

LST_k 為二元樹圖形架構，可從 m 個輸入封包選擇其一進入到 LDT_k。LST_k 之內部節點具有二對一多工器和緩衝器，二對一多工器可從上方輸入與下方輸入中擇其一，來往 LST_k 之根節點傳送封包。當上方、下方輸入同時有封包要進入時，則採用循環的方式，來決定上方、下方輸入優先順序，而緩衝器為 1 個封包大小，當發生阻塞時可用來暫存封包。

LDT_k 為二元樹圖形架構，負責完成 $\log n$ 級數的封包複製。LDT_k 之內部節點具有一對二複製器，此複製器受控制於複製碼 $u_{i_{LDT_k}}^{j_{LDT_k}} l_{i_{LDT_k}}^{j_{LDT_k}}$，如圖 2 所示，其中 $u_{i_{LDT_k}}^{j_{LDT_k}}$ 為 1 表示複製器選擇上方輸出傳送封包，$l_{i_{LDT_k}}^{j_{LDT_k}}$ 為 1 表示複製器選擇下方輸出傳送封包，j_{LDT_k} 為 LDT_k 從根節點到樹葉節點之中第幾個級數，i_{LDT_k} 為 j_{LDT_k} 從上到下之中第幾個節點。$u_{i_{LDT_k}}^{j_{LDT_k}} l_{i_{LDT_k}}^{j_{LDT_k}}$ 為 LDT_k 之內部節點對封包中之 MT 做 OR 運算所產生，如下式(1)與式(2)所示：

$$u_{i_{LDT_k}}^{j_{LDT_k}} = b_{\frac{(i_{LDT_k}+\frac{1}{2})}{2^{j_{LDT_k}}}N-1} \text{OR.....OR} b_{(1-\frac{2^{j_{LDT_k}}-i_{LDT_k}}{2^{j_{LDT_k}}})N} \quad (1)$$

$$l_{i_{LDT_k}}^{j_{LDT_k}} = b_{\frac{(i_{LDT_k}+1)}{2^{j_{LDT_k}}}N-1} \text{OR....OR} b_{\frac{(i_{LDT_k}+\frac{1}{2})}{2^{j_{LDT_k}}}N} \quad (2)$$

如此封包經過 $\log n$ 級數複製後，便可到達佇列集。

佇列集由 n 個各自獨立之佇列所組成，佇列採用先進先出架構，可儲存 m 個封包大小，整個佇列集合共可儲存 nm 個封包。

RST_k 為二元樹圖形架構，可從暫存在佇列集之封包選擇其一進入到 RDT_k。RST_k 之內部節點具有二對一多工器和緩衝器，二對一多工器可從上方輸入與下方輸入中擇其一，來往 RST_k 之根節點傳送封包。當上方、下方輸入同時有封包要進入時，則採用循環的方式，來決定上方、下方輸入優先順序，而緩衝器為 1 個封包大小，當發生阻塞時可用來暫存封包。

RDT_k 為二元樹的圖形架構，負責完成 $\log n$ 級數的封包複製。RDT_k 之內部節點具有一對二複製器，此複製器受控制於複製碼 $u_{i_{RDT_k}}^{j_{RDT_k}} l_{i_{RDT_k}}^{j_{RDT_k}}$，如圖 2 所示，其中 $u_{i_{RDT_k}}^{j_{RDT_k}}$ 為 1 表示複製器選擇上方輸出傳送封包，$l_{i_{RDT_k}}^{j_{RDT_k}}$ 為 1 表示複製器選擇下方輸出傳送封包，j_{RDT_k} 為 RDT_k 從根節點到樹葉節點之中第幾個級數，i_{RDT_k} 為 j_{RDT_k} 從上到下之中第幾個節點。$u_{i_{RDT_k}}^{j_{RDT_k}} l_{i_{RDT_k}}^{j_{RDT_k}}$ 為 RDT_k 之內部節點對封包中之 MT 做 OR 運算所產生，如下式 (3) 與式 (4) 所示：

$$u_{i_{RDT_k}}^{j_{RDT_k}} = b_{\frac{((i_{RDT_k}+2k)+\frac{1}{2})}{2^{(j_{RDT_k}+\log n)}}N-1} \text{OR.....OR} b_{(1-\frac{2^{(j_{RDT_k}+\log n)}-(i_{RDT_k}+2k)}{2^{(j_{RDT_k}+\log n)}})N} \quad (3)$$

$$l_{i_{RDT_k}}^{j_{RDT_k}} = b_{\frac{((i_{RDT_k}+2k)+1)}{2^{(j_{RDT_k}+\log n)}}N-1} \text{OR....OR} b_{\frac{((i_{RDT_k}+2k)+\frac{1}{2})}{2^{(j_{RDT_k}+\log n)}}N} \quad (4)$$

如此封包經過 $\log n$ 級數複製後，便可到達目的輸出埠。

輸出佇列則為採用先進先出架構，可儲存之封包容量越大，發生輸出競爭時，封包遺失機率越小。

2.3 效能分析

計算設計 1 節點數目時，因設計 1 採用輸出佇列來設計，不需額外的控制電路來防止封包發生輸出競爭，所以只需考慮 LST_k 節點數目為 $2m-1$、LDT_k 節點數目為 $2n-1$、RST_k 節點數目為 $2n-1$、RDT_k 節點數目為 $2n-1$ 與佇列集節點數目為 mn，因此設計 1 總共所需節點數目為 $n(2m-1)+n(2n-1)+n(2n-1)+n(2n-1)+mn^2 = mn^2+6n^2+2mn-4n$。

計算設計 1 處理多播封包所花時間，需考慮 LST_k 從 m 個輸入埠選擇封包進入 LDT_k 所花費時間為 m，RST_k 從佇列集 mn 封包之中選擇其一進入 RDT_k 所花時間為 mn，LDT_k、RDT_k 複製封包所花時間皆為 $\log n$，故設計 2 總共花費時間為 $m+\log n+mn+\log n = mn+m+2\log n$。

計算設計 1 佇列所需求記憶空間，需考慮在一個時槽內，每個輸出埠都可接收到 N 個封包，所以設計 1 佇列所需求記憶空間為 $N \times N = N^2$。

定理 2：在使用管線式傳繞方法下，設計 1(如圖 1)處理多播封包所花的時間複雜度為 $O(mn)$，節點複雜度為 $O(mn^2)$。

特例 1：當 $n = m = \sqrt{N}$ 時，設計 1 節點複雜度達最佳化為 $O(N\sqrt{N})$，此時設計 1 時間複雜度 $O(N)$。

2.4 電路設計與實作模擬結果

本節主要是介紹使用設計 1，其中 $n = m = \sqrt{N} = 4$，完成 (16,16) 多播封包交換器之電路設計。設計 1 硬體規格為內部資料線寬度 32 bits，封包長度為固定 64 bytes，因此在設計 1 內兩個節點之間，每個封包需花費 16 個系統時脈(Clock)來傳送，其中第 1 個系統時脈所傳送的封包資料需包含 16-bit MT。此外，因實作輸出佇列需大量記憶體，所以在這裡不考慮實作。

2.4.1 電路設計

LST_k、RST_k 之內部節點決定上方、下方輸入優先順序都採用循環的方式，因此電路都相同，包含了暫存器、控制器、緩衝器、二對一多工器、與計數器。緩衝器是使用 Quartus II 軟體所產生出來 IP，容量為 1 個封包大小。當封包開始進入節點，計數器便開始由 1 至 16 計數。控制器是控制封包寫入與讀出佇列，當緩衝器不為滿時，依上方、下方輸入優先順序允許輸入封包寫入緩衝器，並發出請求往下一個節點傳送封包，還有計數器為 16 時，也須做上方、下方輸入優先順序選擇。暫存器為把封包做延遲，讓佇列寫入正確封包資料。

LDT_k、RDT_k 之內部節點都是使用 DC 來控制一對二複製器，因此兩者電路除了控制器外皆都相同，包含了暫存器、控制器、計數器與一對二複製器。當封包開始進入節點，計數器便開始由 1 至 16 計數。當計數器為 1 時，LDT_k 之內部節點控制器讀取封包資料獲得 MT 後，再利用式(1)與式(2)便可獲得 DC，而 RDT_k 之內部節點控制器讀取封包資料獲得 MT 後，則利用式(3)與式(4) 來獲得 DC。暫存器為把輸入封包資料做延遲，讓控制器能夠來得及設定一對二複製器。一對二複製器則受控於 DC 來完成封包複製，如圖 2 所示。

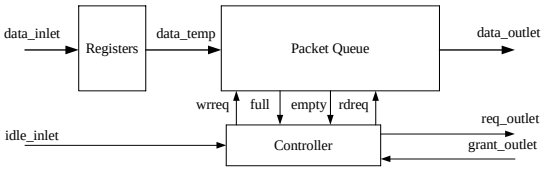

圖 3　佇列集之佇列電路

佇列集是由 4 個各自獨立的佇列所組成，其中每一個佇列電路架構，如圖 3 所示，包含了暫存器、控制器與佇列。佇列是採用 Quartus II 軟體所產生出來的 IP，架構為先進先出的方式，容量為 4 個封包大小。控制器是控制封包寫入與讀出佇列。暫存器為把封包資料做延遲，讓佇列寫入正確封包資料。

2.4.2 實作模擬結果

我們從 (16,16) 多播封包交換器之輸入埠 0 輸入測試封包，第 1 個系統時脈所傳送封包資料為 0000003C(MT 為 003C)，預期在輸出埠 2、3、4 與 5 能接收到輸入埠 0 測試封包；從輸入埠 2 輸入測試封包，第 1 個系統時脈所傳送封包資料為 000000E8(MT 為 00E8)，預期在輸出埠 3、5、6 與 7 能接收到輸入埠 2 測試封包；從輸入埠 8 輸入測試封包，第 1 個系統時脈所傳送封包資料為 0000C000(MT 為 C000)，預期在輸出埠 14 與 15 能接收到輸入埠 8 測試封包；從輸入埠 13 輸入測試封包，第 1 個系統時脈所傳送封包資料為 0000CC00(MT 為 CC00)，預期在輸出埠 10、11、14 與 15 能接收到輸入埠 13 測試封包。整個實作模擬在 Quartus II 下，使用 APEX 20K400EBC – 1X 來完成的 Post-Layout Level Simulation，模擬結果與我們預期相符。

FPGA 最大工作頻率為 50 MHz (=20 ns)，時槽計算方式從每個輸入埠輸入廣播封包開始，到每個輸出埠接收 16 個封包便結束，這期間共需花費 270 個系統時脈，因此時槽為 $270 \times 20 = 5400$ ns，每個埠每秒處理的封包資

料為(64*8)/5400*10^9 =95Mbps。整個晶片的特性如表一。

表一　設計 1 晶片特性

Chip name	Design 1
FPGA	APEX EP20K400EBC652 - 1X
Number of I/O ports	16 × 16
Data width of per port	32-bit
Data rate of per port	95 Mbps
Control	Self-routing
Packet size	64 bytes
Time slot	5400 ns
Max frequency	50 MHz

三、設計 2：含集中器之縮小樹狀多播封包交換器

設計 2 目的在限制每個輸出埠在一個時槽內可接受的封包數目為變數 $L(<< N)$ 個，並以新增加高級集中器(hyperconcentrators)來減少設計 2 佇列集封包緩衝器(buffers)數目，封包複製的方法與設計 1 相同。設計 2 架構如圖 4 所示，組成元件包含了左邊的 n 個選擇樹(Left Selection Trees，簡稱 $LST_k(0\le k\le n-1)$)、左邊的 n 個複製樹(Left Duplication Trees，簡稱 $LDT_k(0\le k\le n-1)$)、右邊的 n 個選擇樹(Right Selection Trees，簡稱 $RST_k(0\le k\le n-1)$)、右邊的 n 個複製樹(Right Duplication Trees，簡稱 $RDT_k(0\le k\le n-1)$)、n 個佇列集(Set of Queues)、n 個高級集中器(Hyperconcentrator)，與 N 個輸出佇列。m 為 LST_k 輸入埠數目。

3.1　設計方法

設計 2 在複製封包方面相同於設計 1，可查看 2.1 和 2.2 節有詳細說明，然而與設計 1 不同之處，為限制了每個輸出埠在一個時槽內，可接受的封包數目為變數 L 個，在此條件下設計 2 利用高級集中器，把 LDT_k 所複製出來的大量封包連續集中，再儲存到佇列集，此時佇列集只需要 nL 個封包緩衝器，封包便不會因等待被 RST_k 選擇而遺失。佇列集為 n 個各自獨立之佇列所組成，佇列採用先進先出架構，具有 L 個封包緩衝器。

我們可根據 Knockout 論文中提到，當每個輸出埠可接受的封包數目被限定為變數 $L(=12)$，封包發生超過 $L(=12)$ 的機率為 $\le 10^{-10}$，這點說明絕大部分情況下，要同時抵達相同輸出埠的封包不會超出 $L(=12)$。

定理 3：在限定每個輸出埠在一個時槽內，可接受的封包數目為變數 L，設計 2 藉由高級集中器把封包連續地集中再儲存入佇列集，此時佇列集之佇列只需有 L 個封包緩衝器，封包便不會遺失。

證明：在一個時槽內，RST_k 從佇列集中選擇封包進入到 RDT_k，再由 RDT_k 複製封包到達 n 個目的輸出埠，因此在每個輸出埠可接受的封包被限定為 L 個的條件下，最多只有 nL 封包需被暫存在佇列集內，所以佇列集只需要 nL 個緩衝器封包，封包便不會遺失。高級集中器可讓封包連續地集中，避免超過 L 個封包要儲存在佇列集之同一個佇列，導致封包遺失。

3.2　高級集中器

高級集中器為 Jan 所提出[4]，如圖 5 所示，組成元件包含了 Prefix Ranking Tree[5]、並聯加法器(Parallel Adders)、立方網路(Indirect Binary Cube Network)、立方網路前面的 q 個選擇樹(Selection Trees，簡稱為 $HST_k(0\le k\le q-1)$)、立方網路後面的 q 個分佈樹(Distribution Trees 簡稱為 $HDT_k(0\le k\le q-1)$)。

這邊要注意到，在設計 2 採用管線式傳繞運作下，多個封包同時往目的輸出埠傳繞，高級集中器內 HST_k 與立方網路會發生阻塞，為避免封包因此遺失，立方網路與 HST_k 內部節點需利用緩衝器來暫存封包。以下對高級集中器各組成元件做介紹。

Prefix Ranking Tree 目的為產生由 1 為起始值的 monotonic ranks，運作原理為：當輸入路徑有封包輸入時，以 1 為 Prefix Ranking Tree 的處理輸入值；當輸入路徑無封包輸入時，以 0 為 Prefix Ranking Tree 的處理輸入值，然後在

經 Prefix Ranking Tree 處理後，其產生輸出為由 1 為起始值的 monotonic ranks， r 為輸入高級集中器封包數目， $0 \le r \le n$ 。

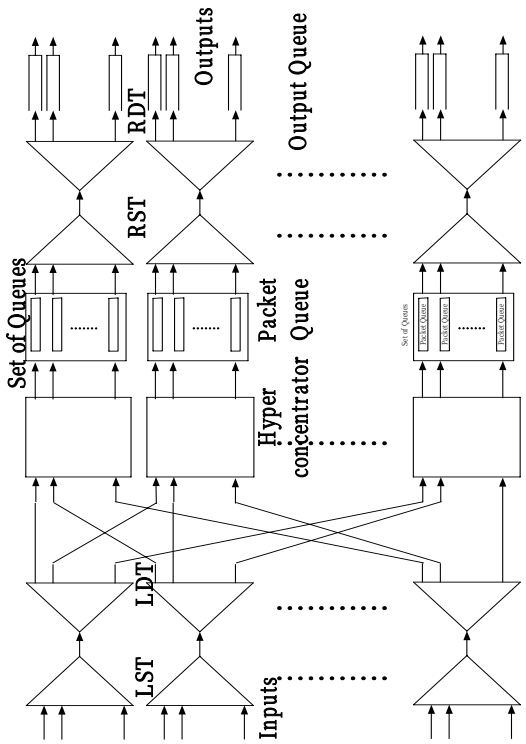

圖 4　(N, N) 設計 2

並聯加法器主要為把 monotonic ranks 一同做加起始值(offset)的運算，並對無封包輸入的相對輸出路徑之 rank 設定為 null，如此便可得到立方網路和 DT_k 所需的 routing ranks，起始值為上次 Prefix Ranking Tree 所產生的 monotonic ranks 中之最大值並減 1，減 1 為產生高級集中器輸出埠 0 之 routing rank。

HST_k 為二元樹的圖形架構，可從 p 個輸入封包選擇其一進入到立方網路。HST_k 之內部節點具有一個二對一多工器和緩衝器，二對一多工器可從上方輸入與下方輸入中擇其一，來往 HST_k 之根節點傳送封包。當上方、下方輸入同時有封包要進入時，則採用循環的方式，來決定上方、下方輸入優先順序，而緩衝器為 1 個封包大小，當發生阻塞時可用來暫存封包。

立方網路為 $\log q$ 級數的 $(2,2)$ 轉轍器(switch elements)所組成，如圖 6 所示，每個 $(2,2)$ 轉轍器內都有緩衝器可暫存封包。立方網路處理封包傳繞的方式為 LSD (least- significant-digit) radix sorting，從封包中 routing rank 的最左邊 $\log q$ bits 由低位元往高位元，分別做為立方網路中從左到右之各級數 routing rank。當 routing rank 為 0， $(2,2)$ 轉轍器選擇上方路徑；routing rank 為 1， $(2,2)$ 轉轍器選擇下方路徑。如果 $(2,2)$ 轉轍器內發生兩個輸入埠要到達相同輸出埠時，則採用循環的方式，來決定輸入埠的優先順序，而緩衝器為 1 個封包大小，當發生

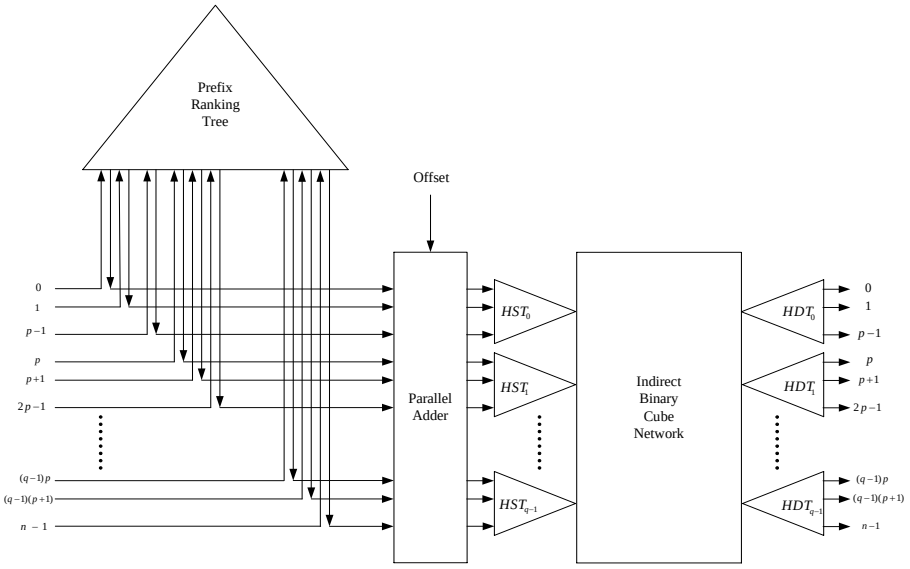

圖 5　(n, n) 高級集中器

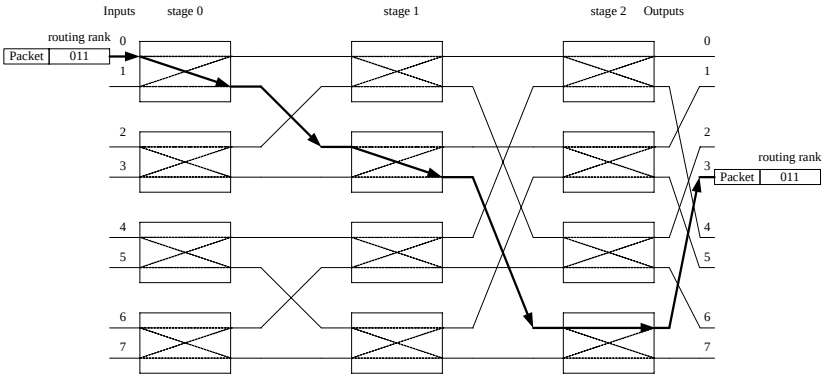

圖 6　(8,8) 立方網路

阻塞時可用來暫存封包。

HDT_k 為二元樹的圖形架構，處理封包傳繞的方式為 MSD (most-significant-digit) radix sorting，從封包中 routing rank 的最右邊 $\log p$ bits 由高位元往低位元，分別做為 HDT_k 中從根節點到樹葉節點之各級數的 routing rank。HDT_k 之內部節點具有一個一對二解多工器 (1-to-2 demultiplexer)，如 routing rank 為 0，解多工器選擇上方輸出傳送封包；routing rank 為 1，解多工器選擇下方輸出傳送封包。

3.3　效能分析

計算設計 2 節點數目時，因設計 2 採用輸出佇列來設計，不需額外的控制電路來防止封包發生輸出競爭，所以只需考慮 LST_k 節點數目為 $2m-1$、LDT_k 節點數目為 $2n-1$、RST_k 節點數目為 $2n-1$、RDT_k 節點數目為 $2n-1$、佇列集節點數目為 nL 與高級集中器節點數目為 $4n+2q\log q$，因此設計 2 總共所需節點數目為 $n(2m-1)+n(2n-1)+n(2n-1)+n(2n-1)+Ln^2+4n+2q\log q = Ln^2+6n^2+2mn\ +\ 2q\log q$。

計算設計 2 處理多播封包所花時間`，需考慮 LST_k 從 m 個輸入埠選擇封包進入 LDT_k 所花費時間為 m，RST_k 從佇列集 Ln 個封包之中選擇其一進入 RDT_k 所花時間為 Ln，LDT_k、RDT_k 複製封包所花時間皆為 $\log n$，高級集中器所花費時間為 $4\log n+1$，故設計 2 總共花費時間為 $m+\log n+Ln+\log n+4\log n+1 = m+Ln+6\log n+1$。

計算設計 2 佇列所需求空間，需考慮限定每個輸出埠在一個時槽內，可接受的封包數目為變數 L，所以設計 2 佇列所需求記憶空間為 $L\times N = LN$。

定理 4：在使用管線式傳繞方法下，設計 2(如圖 4) 處理多播封包所花的時間複雜度為 $O(m+Ln)$，節點複雜度為 $O(Ln^2+mn+q\log q)$。

特例 3.1：當 $n=m=\sqrt{N}$、$p=q=\log n$ 時，設計 2 節點複雜度達最佳化為 $O(LN)$，此時設計 2 時間複雜度為 $O(L\sqrt{N})$。

若考慮在同一時間某些 port 有大量 input 時的情形，因此會有 HOL (Head of line) blocking 出現，則需利用 QoS (Quality of Service) 的額外機制，外加控制器來控制 input 的流量管制，或是把 input queue 加長，來減低 packet drop 的情形，而這也無可避免讓成本增加，但會使得效率相對的提高。

四、結論

本篇論文所提出兩種多播封包交換器，在考量封包傳輸率，具有時間複雜度為 $O(\log N)$ 設計 1 最佳；在考量實作硬體成本，具有節點複雜度為 $O(N\sqrt{N})$ 設計 1 最佳;在 $N >> L$ 的條件下，具有時間複雜度為 $O(L\sqrt{N})$ 與節點複雜度為 $O(LN)$ 設計 2 最佳。在 VLSI 實作方面，受限於 FPGA 元件 APEX 20K400EBC – 1X 大小，設計 1 實作 (16,16) 多播封包交換器，每個埠可達到 95 Mbps 封包傳輸率；設計 2 則無實

作。未來預計實作可採用 ASIC (Application Specific IC)技術，來完成封包傳輸率更快與埠數更多的交換器。

五、參考文獻

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

[2] J. L. Derome, D. Al-Khalili, M. H. Rahman, "VLSI Implementation of a Copy Network for a Multicast ATM Switch," *Canadian Conference on Electrical and Computer Engineering*, vol. 1, pp. 328-331, Sept 1994.

[3] A. Huang and E. Arthurs, "Starlite: A Wideband Digital Switch," *Proc. IEEE Globecom '84*, pp. 121-125, 1984.

[4] G. E. Jan, "Composite Interconnection Networks for Parallel Computers," *Ph.D. Dissertation*, U. of Maryland, USA, 1992.

[5] D. Koppelman and A. Y. Oruc, "A self-routing permutation network," *Journal of Parallel and Distributed Computing*, no. 10, pp. 140-151, 1990.

[6] Andreas Kirstadter, "Contention Resolution for Different Traffic Categories in Large Input Buffered ATM Switches," *Proc. IEEE ATM Workshop*, 1997.

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

[8] X. Liu, H. T. Mouftah, "Design of a High Performance Nonblocking Copy Network for Multicast ATM Switch," *IEEE International Conference on Communications*, pp. 317-324, Oct 1994.

[9] Tsern-Huei Lee, Shun-Jee, "Multicasting in a Shared Buffer Memory Switch," *IEEE TENCON'93*, vol. 1, pp. 209-212.

[10] Jin Li and Chuan-lin Wu, "Design and Implementation of a Multicast-Buffer ATM Switch," *IEEE International Conference on Network Protocols*, pp. 84-91, Nov 1995.

[11] Nick McKeown, "Fast Switched Backplane for a Gigabit Switched Router," Cisco Systems, May 1998.

[12] Y. M. Matcheswala, "An Architecture and Algorithms for Multicasting Data in Computer Communication Networks," *Comp. Commun. J.*, March 2001.

[13] Nader F. Mir, "A Survey of Data Multicast Techniques, Architectures, and Algorithms," IEEE Communications Magazine, pp. 164-170, Sept 2001.

[14] Jonathan S. Turner, "Design of a Broadcast Packet Switching Network," *IEEE Transactions on Communications*, vol 36, June 1988.

[15] Jonathan S. Turner, "An Optimal Nonblocking Multicast Virtual Circuit Switch," *Proc. IEEE INFOCOM'94*, pp. 298-305, June 1994.

[16] 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 Communications*, vol 5, pp. 1274-1283, Oct 1987.

[17] K. Yoshigoe and K. J. Christensen, "A Parallel-Polled Virtual Output Queued Switch with a Buffered Crossbar," *2001 IEEE Workshop on*, 2001.

[18] Wen De Zhong, Yoshikuni Onozato, Jaidev Kaniyil, "A Copy Network with Shared Buffers for Large Multicast ATM Switch," *IEEE International Conference on Communications*, vol. 2, pp. 722-726, May 1993.