使用新的快取記憶體架構增進以雜湊表為基礎的 IP 查表演算法之效能

吳其政
龍華科技大學電子工程系
講師
jesse@mail.lhu.edu.tw

黃彥涵
龍華科技大學電子工程系
研究生
ryan.smalla@xuite.net

吳東旭
龍華科技大學電子工程系
副教授
wds@mail.lhu.edu.tw

摘要

路由器主要是查詢 IP 封包的目的位址來決定封包的下一站。由於 CIDR 的使用在決定 IP 的下一站時必須在路徑表找出符合的最長前置位元。本文使用線性雜湊搜尋法、二元雜湊樹和非對稱的二元雜湊樹等三種方法來執行路由查表的工作。結果發現以非對稱的二元雜湊樹效率好，平均的 hash 次數只有 1.0882 次。為了增快搜尋的速度，我們加入了記憶體快取，使用的映射函數為直接和集合關聯二種，置換演算法有 FIFO、隨機和 LRU 三種，實驗的結果顯示 LRU 置換演算法的效果最好，且集合關聯愈大時快取記憶體的失誤率越低。除此之外，我們使用新的快取方法，不僅儲存封包的目的位址且儲存前置長度到快取記憶體中，這個方法可以增加存在快取記憶體每一筆資料的範圍，因此提高了快取擊中比(hit rate)。模擬的結果顯示新的快取方法能有效率的使用快取記憶體且大幅的增進路由查表效能。

關鍵詞：IP 路由查表，雜湊搜尋，記憶體快取。

Abstract

With the increasing routing table size, explosive growth of traffic, and higher speed links, the router requires performing routing lookup to forward the packet up to 10Gbps or above. Since the use of CIDR, IP routing lookup must find out the longest matching prefix with the IP packet destination address in the routing table. This paper presents three schemes that use hashing to perform lookup: linear hash search, binary hash tree and asymmetric binary hash tree. In addition, we add the cache memory to increase speed of routing lookup. We evaluate the mapping functions for direct mapping and set-associative mapping and the replacement algorithms for FIFO, random and LRU. Our experiments show that using LRU and higher level of set-associative can improve the cache performance. We also propose a new mechanism to further improve the cache performance. Our major ideas are that we not only cache the destination address of the packet but also cache the prefix length. The prefix length information can extend the ranges of each entry in the cache and thus increase the cache hit rate. From the simulation results, we believe our scheme can achieve caching effectiveness and improve the IP routing lookup performance significantly.

Keywords： IP routing lookup, hash search, memory cache.

1. 簡介

因為網路的普及化和多媒體的廣泛應用，使得網際網路的通信量(traffic)快速的增加。再者，光纖線路的普及，連線的速度大幅提升，例如 OC-192 光纖線路的速度為 10Gb/s，因此網際網路流量(traffic)的主要瓶頸已由網路線路轉移至路由器(Router)。傳統的路由器主要是查詢 IP(Internet Protocol)封包的目的位址(Destination Address)來決定封包的下一站(Nexthop)。由於 CIDR[1]的使用，在查詢下一站時必須在路徑表(Routing table)中找出符合的最長前置位元(Longest Prefix Matching)，因此，IP 查表可包含二個步驟：首先，在路由表中尋找和封包的目的位址相符合的集合。再來，從這些相符合的集合中找出符合最長的前置長度。例如，表 1 為一個簡單的路徑表，假設當一個封包進來時，它的目的位址為 00100111，查尋路徑表可以找到 *B*(001*)和 *C*(0010*)等二項跟目的位址相符合，依照 CIDR 的定義必須找出符合的最長前置位元，所以它的下一站為 *C*(0010*)。

目前已有許多關於加速 IP 查表方面的研究，這些研究可略分為以軟體和硬體方法兩方向，就軟體而言，它大多以 trie 資料結構為基礎如[2 - 7]，硬體部分則有[8 - 11]等相關研究。

表 1 簡單的路徑表

prefix	prefix length	nextHop
000*	3	A
001*	3	B
0010*	4	C
00111*	5	D
11010001	8	E
110111*	6	F
11000*	5	G
10*	2	H
11010000	8	I
00101001	8	J

圖 1 是依照表 1 所建造的一個簡單的二元樹架構(Binary trie)，建造的方法是根據路徑表中的每一個項目，從左至右逐一讀取位元，首先建立樹根(Root)節點，當讀取的位元為 0 時，就往左建立子節點；為 1 時就往右建立子節點，直到讀取到“*”時，就將下一站存至該節點，我們用黑點加以區別。例如，建立 *H*(10*)時，第一個位元為 1，就往右建立子節點，第二個位元為 0，接下去往左建立子節點，因為接下來為*，所以就在該節點存入下一站“*H*”，並用黑色節點表示，如圖 1 中實線箭頭所示。

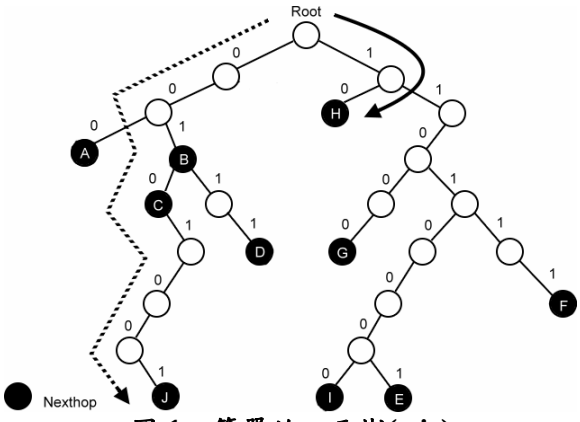

圖 1 簡單的二元樹(trie)

二元樹的搜尋是依照 IP 目的位址的位元從左到右逐一往下搜尋，首先從樹根進入，假設讀取的位元為 0 時，就往左子樹(Left Subtree)搜尋；為 1 時就往右子樹(Right Subtree)搜尋，當遇到黑色節點時(表示已找到相符合的前置位元)，則將它的下一站儲存起來，若尚有子樹，則繼續往下搜尋是否有相符合的更長前置位元，如此可找出相符合的最長前置位元。例如有一個 8-bit 的 IP 目的位址(00101001)，從左到右依序往下搜尋，可以發現相符合的前置位元長度有 *B*（001*）、*C*（0010*）和 *J*（00101001*）等三項跟 IP 目的位址相符合，如圖 1 中虛線箭頭部份所示，我們找到了符合 IP 目的位址（00101001）的最長前置位元並將下一站 *J* 輸出。

本論文主要是參考[4]使用雜湊方法(Hash)來製作和查尋路徑表，雜湊的索引稱做為鍵(Key)，其對應的資料，稱為值(Value)，雜湊就是由一對對的 “鍵/值” 組合而成。換言之，就是把某一套資料對應到另一套資料。雜湊首先的工作就是建立雜湊表(Hash Tables)，建立雜湊表的方式則是視使用的雜湊函數(Hash Function)而定，雜湊函數為一種數學的運算，理想的狀態下，在搜尋時，只需要經過一次的雜湊函數計算，便能在建好的雜湊表中搜尋到所要的值。我們將在下一節中做詳細描述。

本篇論文會在第二節介紹路由查表的方法，第三節介紹快取的方法，第四節是模擬的結果與分析，第五節是結論。

2. 路由查表的方法

我們使用了三種雜湊方法來搜尋 IP 的下一站位址，第一個為線性雜湊搜尋法(Linear Hash Search) ，第二個方法是使用二元雜湊樹(Binary Hash Tree)，第三個方法為非對稱二元雜湊樹，詳細的介紹如下。

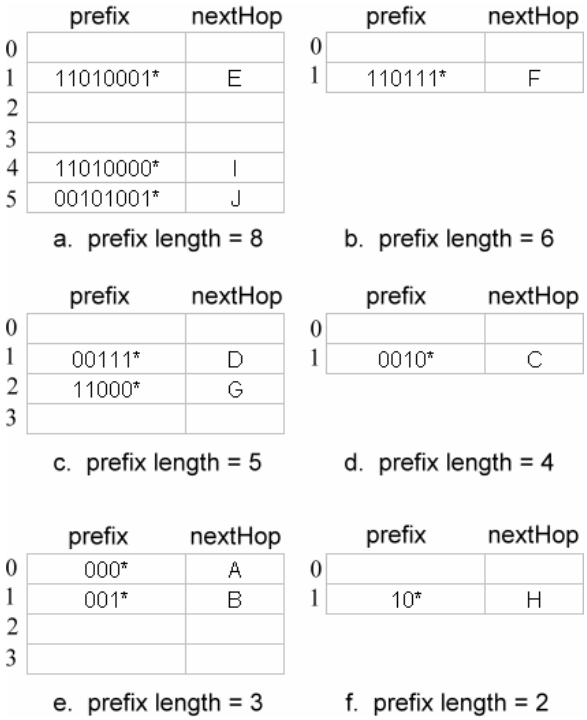

圖 2 根據表 1 所建立之各種不同前置位元長度的 hash table

2.1 線性雜湊搜尋法

首先我們根據各種不同的前置位元長度分別建立雜湊表，如圖 2 是根據表 1 所建立的各種不同前置位元長度之雜湊表。在查表時，則由前置位元長度較長的雜湊表逐次往下搜尋，直到找到為止。因為是從最長的前置位元雜湊表開始搜尋，所以如果找到就代表是找到了符合最長的前置位元，就可以直接將下一站輸出。例如，IP 的目的位址為 11011100，首先至前置位元長度為 8(如圖 2a)的雜湊表搜尋，沒找到則至前置位元長度為 6(如圖 2b)的雜湊表搜尋，找到了則將它的下一站 *F* 輸出。

圖 3 為我們查表時所找到各種不同長度前置位元分佈圖，橫座標為前置位元長度，縱座標為找到的前置位元長度個數(取$\log_{10}$)。由圖顯示，根據我們的路徑表和trace資料，我們查表所找到的下一站，大部份集中在前置位元長度為 16 的項目，共有 60872 筆，其次為長度 20 有 114 筆。線性雜湊搜尋法平均的hash次數大約有 16 次。

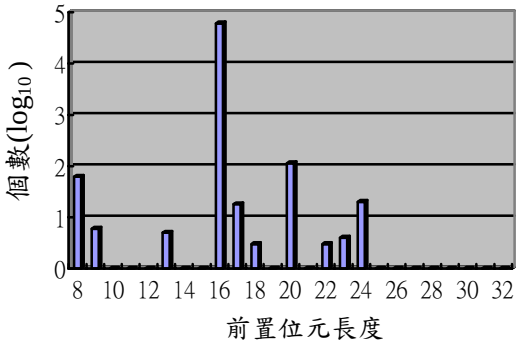

圖 3　找到各種不同長度前置位元分布圖

2.2 二元雜湊樹

為了減少 hash 的次數，第二個方法是使用二元雜湊樹來決定 hash 的順序，首先，我們算出中間的前置位元長度當做二元雜湊樹的根節點，再把較小的前置位元長度放到左子樹，較長的前置位元長度放到右子樹，如此循環可建立二元雜湊樹，在這個方法中，我們必須將較長的前置位元節點，在其根節點上建立標記(Marker)，表示右子樹上還有更長的前置位元資料， Marker 設定為 1 表示該項資料尚有更長的前置位元。例如，我們從表 1 可以知道前置位元長度為 2 到 8，所以位於中間的前置位元長度就為(2+8)/2 = 5 設為根節點，再將長度大於根節點的放入右子樹；長度小於根節點的放入左子樹建立二元雜湊樹。如圖 4 為根據表 1 所建立的二元雜湊樹，圖 5 為其所對應之各種不同前置位元長度雜湊表，在搜尋的時候，先從前置位元長度為根節點的進行查表，若找不到符合的前置位元，就往左節點搜尋；若有找到且不是最長的前置位元(此時 Marker 為 1)就往右節點搜尋。例如有一個 IP 目的位址為 00100111，查表時，是從根節點前置位元長度為 5 開始搜尋(圖 5d 的雜湊表)，找不到就往左節點為 3 搜尋(圖 5f 的雜湊表)，找到了但是不是最長的前置位元(Marker 為 1)，所以繼續往右節點為 4 搜尋(圖 5e 的雜湊表)，找到了且符合最長的前置位元(Marker 為 0)就將下一站 *C* 輸出。二元雜湊搜尋演算法如圖 6 所示。

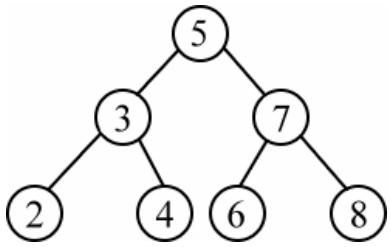

圖 4　根據表 1 建立的二元雜湊樹

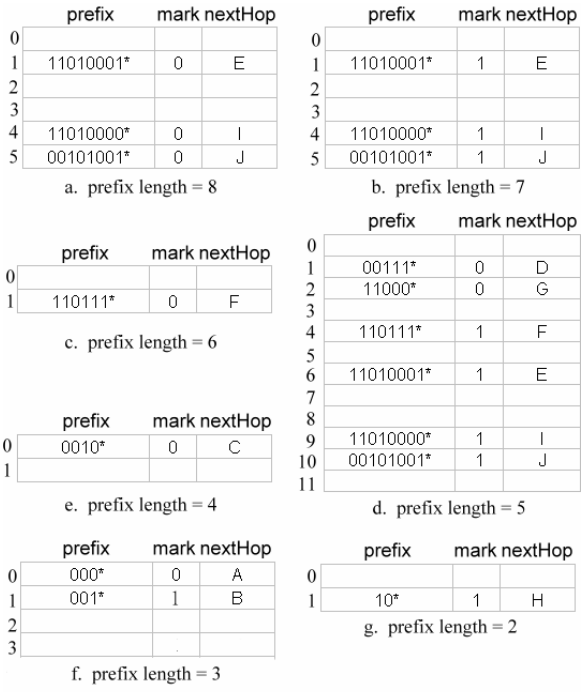

圖 5　根據表 1 所建立之各種不同前置位元長度的 hash table

根據我們所得到的路徑表資料建出的二元雜湊樹如圖 7 所示，因為在我們的路徑表中，前置位元的長度是從 8 到 32，所以位於

中間的前置位元長度就為(32+8)/2 = 20，就為根節點，再將長度大於根節點的放入右子樹；長度小於根節點的放入左子樹建立我們的二元雜湊樹。最後根據我們的路徑表和 trace 資料，得到的平均的 hash 次數約為 5 次。

```
Procedure binaryHashSearch (ip)
  low =  length of shortest prefix length in routing table
  high =  length of longest prefix length in routing table
  while low <= high do
    mid = (low + high) / 2;   //計算位於中間的前置位元長度
    (mark, nextHop) = searchHash (mid, ip);
    case
    Mark = 0 and nexthop != 0 : return nexthop; //將Nexthop輸出
    Mark = 0 and nexthop = 0  : high = mid-1; //找左節點
    Mark = 1 and nexthop != 0 : low = mid+1; //找右節點
    Mark = 1 and nexthop = 0  : low = mid+1; //找右節點
    end
  end
  return "not found"
end
```

圖 6　二元雜湊搜尋演算法

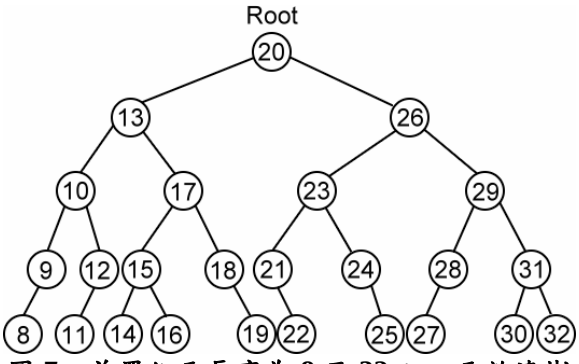

圖 7　前置位元長度為 8 至 32 之二元雜湊樹

2.3　非對稱的二元雜湊樹

我們從圖 3 不同前置位元長度分布圖可以發現，根據路徑表找到的前置位元長度幾乎都出現於前置位元長度為 16 的項目，我們將出現機率最高的前置位元長度 16 放在根節點，再將前置位元長度大於根節點的放入右子樹；前置位元長度小於根節點的放入左子樹。依照這個方法，將出現機率高的前置位元長度放在根節點，如此往下遞回可建立非對稱二元雜湊樹，如圖 8 所示。最後根據我們的路徑表和 trace 資料，得到的平均的 hash 次數有 1.0882 次。

3. 快取的架構

快取記憶體(cache memory)主要是希望查表的速度能夠趨近最快速的記憶體，通常快取記憶體會拷貝部分主記憶體的內容。圖 9 是快取架構的基本模型，主要包含快取記憶體和路徑表二個部份，當封包從輸入端近來時，會直接到快取記憶體去搜尋，若有找到就直接將封包的下一站輸出；若沒找到就到路徑表去搜尋，再將資料回存至快取記憶體並且將找到的封包下一站輸出。

不同的快取方法對快取架構的性能有很大的影響，在這節中，我們將對以下二大方面來探討其對快取性能的影響：快取關聯(cache associatives) 和 快 取 置 換 演 算 法 (cache replacement algorithms)。

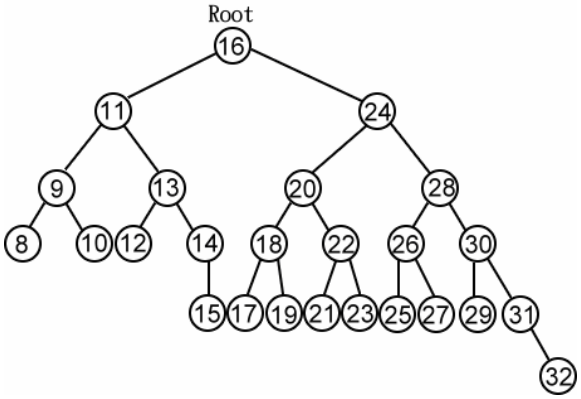

圖 8　非對稱的二元樹

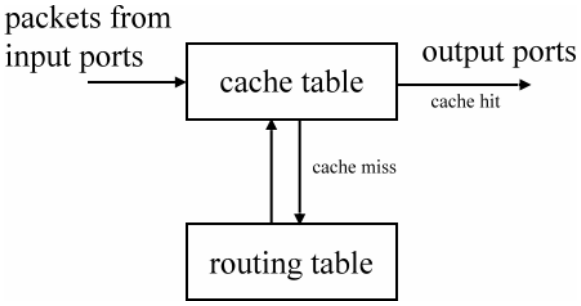

圖 9　快取的基本模型

3.1　快取關聯

增加關聯度已經廣泛使用於 CPU 記憶體快取技術中來增加擊中比。在本節中，我們將探討 IP 查表快取技術，通常關聯有三種技術分別是直接(direct)映射、關聯(associative)映射和集合關聯(set associative)，本論文使用的方法為直接映射和集合關聯。

3.1.1 直接映射

直接映射的方法是直接將主記憶體內的每各區塊映射到唯一的快取記憶體，其架構如圖 10 所示，我們將 IP 的目的位址分為標籤(tag)和索引(index)二個欄位，當封包到達時，取出其 IP 目的位址，根據它的索引位址到快取記

憶體搜尋，比較 tag 是否相同，假如相同就將 IP 的下一站輸出；若不同就必須到路徑表搜尋，找到了就將資料存至快取記憶體並且將下一站輸出。假設有一個 IP 的目的位址為 11000110，它的標籤為 11000，索引值為 110，在搜尋時首先到快取記憶體中索引值為 110 的位址搜尋，比較標籤是否相同，若相同就將下一站輸出；若不同就到路徑表中搜尋，找到了就將標籤 11000 和下一站 *G* 存至快取記憶體中的索引 110 位址並且將下一站 *G* 輸出。

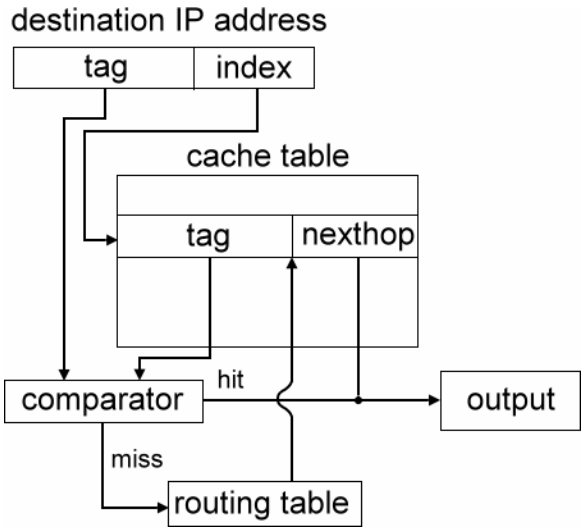

圖 10　直接映射架構

我們提出一個新的方法是將原本快取記憶體的欄位加入 length 和 longest 欄位，其中 length 欄位為前置位元長度，longest 欄位代表該項是否為最長的前置位元長度，如圖 11 所示。在搜尋的時候，根據 IP 目的位址的索引值到快取記憶體中取出 tag 值，若 IP 之 tag 值等於快取記憶體中的 tag 值，就將下一站輸出；若不相等，根據快取記憶體 length 的欄位指示將標籤做遮罩再比較，如果相等而且 longest = 1，就將下一站輸出，若不符合就到路徑表中搜尋。根據我們的模擬，將直接映射和加入 length 和 longest 欄位分別比較在快取記憶體大小為 512、1K、2K 和 4K 的結果如圖 12 所示，直接映射的失誤率在 0.246249% 到 0.0857648%之間，加入 length 和 longest 欄位的失誤率在 0.088154%到 0.0405669%之間，模擬結果發現加入 length 和 longest 欄位的能很有效率地降低快取失誤率(miss rate)。

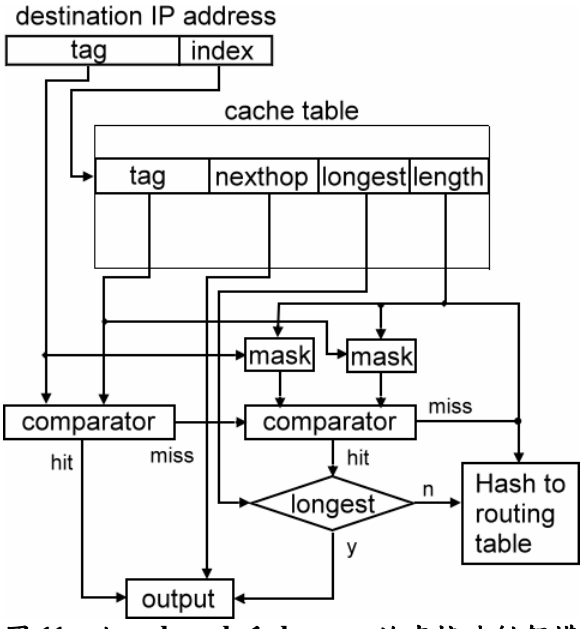

圖 11　加入 length 和 longest 的直接映射架構

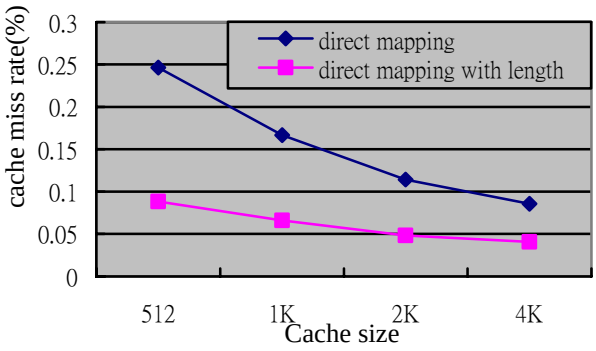

圖 12　比較各直接映射在不同的記憶體大小下的失誤率

3.1.2 集合關聯

集合關聯映射是將快取記憶體的區塊分成 *m* 向(*m*-way)集合關聯映射，如圖 13 所示為 4 向的集合關聯，將快取記憶體在索引的位址分成 4 個空間儲存，當封包到達時，取出其 IP 目的位址，計算出索引位址，根據索引位址至快取記憶體中比對每一個標籤，若找到符合的就將下一站輸出；若沒找到就至路徑表搜尋再將標籤和下一站存至快取表中後將下一站輸出。假如我們以表 1 路徑表為例，當 IP 目的位址為 11010001 的封包到達時，首先依照它的索引 001 到快取表搜尋，依序比對 tag 是否相同，若相同就將下一站輸出；若不同就到路徑表中搜尋，找到了就將下一站 *E* 和標籤 11010 存入快取表並且將下一站 *E* 輸出。

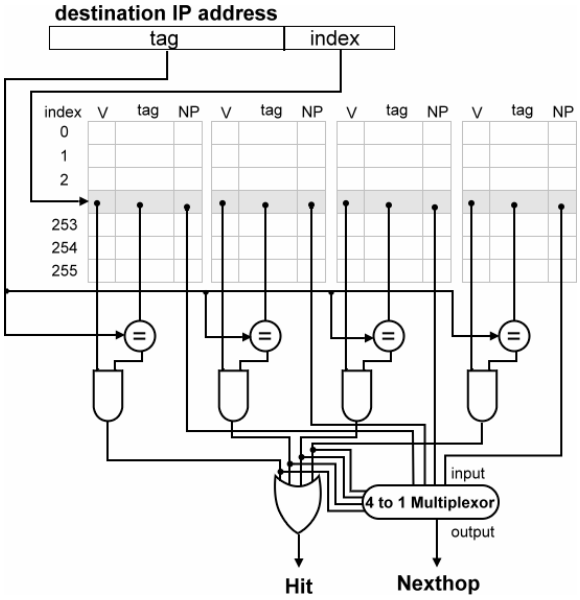

圖 13　4 向集合關聯映射架構圖

根據上述將快取表加入 length 和 longest 欄位的架構，我們同樣的應用在集合關聯映射，如圖 14 所示為加入 length 和 longest 欄位後的 4 向集合關聯映射的架構圖，當封包到達時，取出其 IP 目的位址，計算出索引位址，根據索引位址至快取記憶體中比對每一個標籤，若找到符合的就將下一站輸出；若不相等，根據快取表 length 欄位的值將標籤做遮罩再比較，如果相等而且 longest = 1，就將下一站輸出，若不符合就到路徑表中搜尋。

3.2　置換演算法

當快取記憶體載入了新的標籤和下一站就必須更換新的標籤和下一站。對直接映射而言，只有一條快取線所以直接更換不需置換演算法(replacement algorithms)，但是對集合關聯映射而言，它的快取線有 *m* 個，就需要置換演算法決定需要更換的位址，在本論文使用的置換演算法為先進先出 (first-in-first-out，FIFO) 、隨機 (random) 和最近最少用 (least-recently used，LRU) 置換演算法。

FIFO 置換演算法是將標籤和下一站依序放入快取表，當快取表滿了就將最先存入的標籤和下一站更換。隨機置換法是在候選的標籤和下一站隨機選取置換的位址。最後一種方法是 LRU 置換法，這個方法是將最少用到的位址替換。

4. 模擬結果與分析

根據我們的路徑表和 trace 資料，模擬 1、2、4 和 8 的集合關聯，每一個集合關聯的快取記憶體項目大小分別為 512、1K、2K 和 4K，使用的置換演算法分別為 FIFO、隨機和 LRU，我們將在下面做詳細的說明。

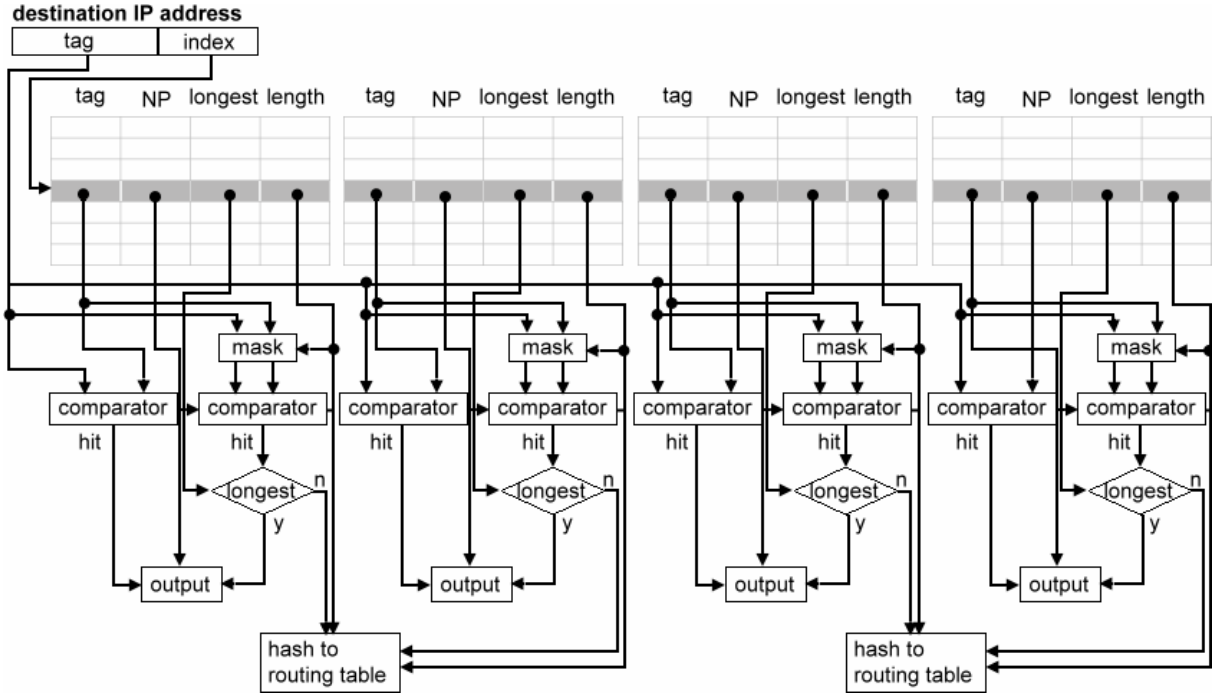

圖 14　加入 length 和 longest 欄位後的 4 向集合關聯映射的架構圖

圖 15 是使用 FIFO 的置換演算法，橫座標為集合關聯，縱座標為快取失誤率，統計的結果失誤率在 0.246249%到 0.0252827%之間，圖 16 是加入 length 和 longest 欄位後使用 FIFO 的置換演算法，失誤率在 0.088154%到 0.0198498%之間。圖 17 是使用隨機置換演算法，失誤率在 0.246249%到 0.0251518%之間，圖 18 是加入 length 和 longest 欄位後使用隨機置換演算法，其失誤率在 0.088154%到 0.0212407%之間。圖 19 是使用 LRU 的置換演算法，失誤率在 0.246249%到 0.0251518%之間，圖 20 是加入 length 和 longest 欄位後使用 LRU 的置換演算法，失誤率在 0.088154%到 0.0205207%之間。圖 21 是快取記憶體大小為 1K，集合關聯分別為 2、4 和 8 的比較圖，橫座標為使用的置換演算法，縱座標為失誤率，我們可以看出以 LRU 為最佳。圖 22 是加入 length 和 longest 欄位，快取記憶體大小為 1K，集合關聯分別為 2、4 和 8 的比較圖，在加入 length 和 longest 欄位後，LRU 的效能仍最佳。圖 23 是在相同的快取記憶體下比較各集合關聯的快取失誤率，集合關聯為 1 快取記憶體大小為 4K、集合關聯為 2 快取記憶體大小為 2K、集合關聯為 3 快取記憶體大小為 1K 和集合關聯為 4 快取記憶體大小為 512 時失誤率的比較圖，圖 24 是加入 length 和 longest 欄位後在相同的快取記憶體下比較各集合關聯的快取失誤率，集合關聯為 1 快取記憶體大小為 4K、集合關聯為 2 快取記憶體大小為 2K、集合關聯為 3 快取記憶體大小為 1K 和集合關聯為 4 快取記憶體大小為 512 時失誤率的比較圖。在圖 23 和圖 24 中我們發現當快取記憶體大小相同時，集合關聯越多，失誤率也會相對的下降，且 LRU 的效率最好，加入 length 和 longest 欄位也會使失誤率更低。從圖 15 到圖 22 的結果，我們發現當快取記憶體愈大和集合關聯愈多時，失誤率越低，且將快取記憶體加入 length 和 longest 欄位也能有效率的降低失誤率。

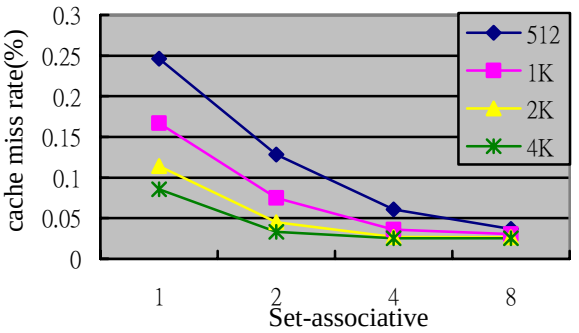

圖 15　FIFO 置換演算法快取失誤率

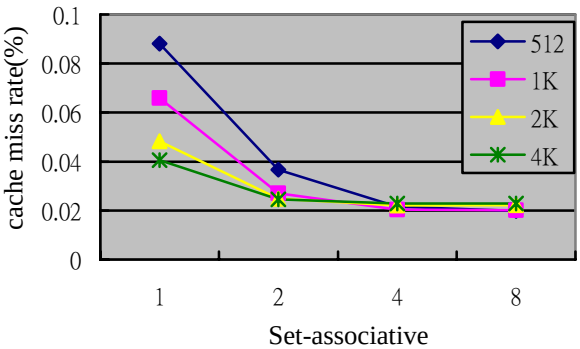

圖 16　加入 length 和 longest 欄位後的 FIFO 置換演算法快取失誤率

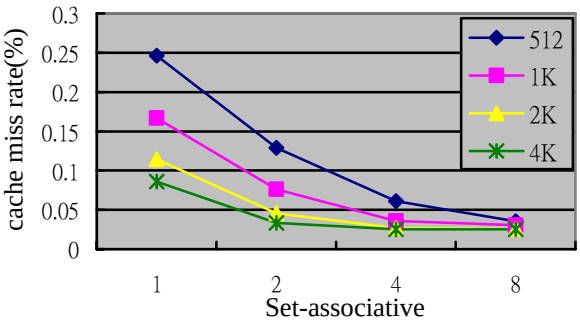

圖 17　Random 置換演算法快取失誤率

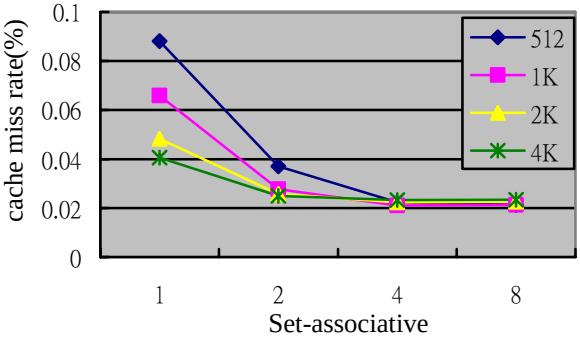

圖 18　加入 length 和 longest 欄位後的 Random 置換演算法快取失誤率

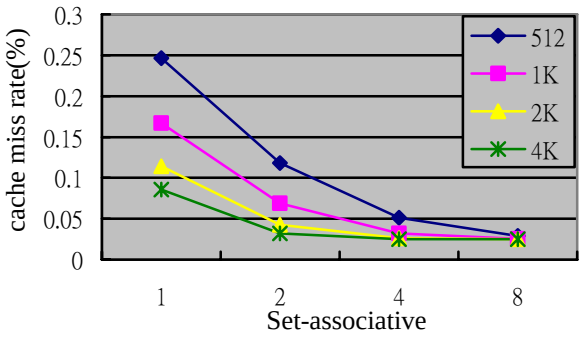

圖 19　LRU 置換演算法快取失誤率

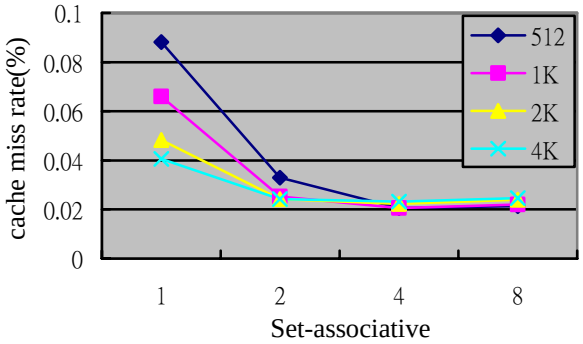

圖 20　加入 length 和 longest 欄位後的 LRU 置換演算法快取失誤率

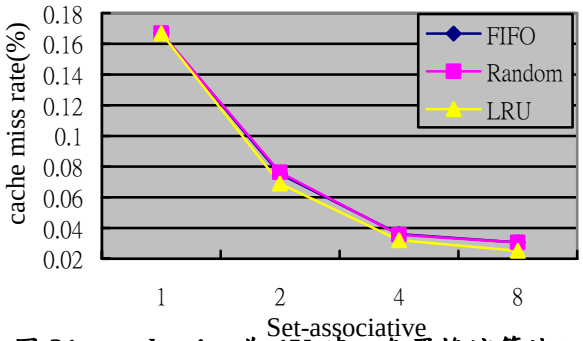

圖 21　cache size 為 1K 時，各置換演算法之快取失誤率比較圖

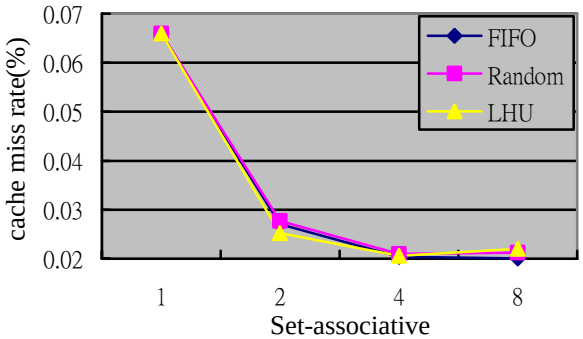

圖 22　加入 length 和 longest 欄位後 cache size 為 1K 時，各置換演算法之快取失誤率比較圖

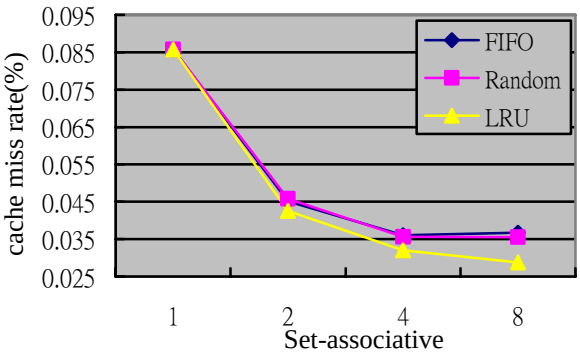

圖 23　在相同的快取記憶體下比較各集合關聯的快取失誤率

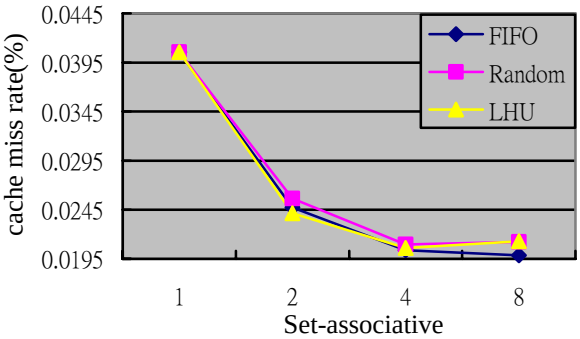

圖 24　加入 length 和 longest 欄位後在相同的快取記憶體下比較各集合關聯的快取失誤率

5. 結論

這篇論文中我們根據前置位元長度建立各自的雜湊表，使用線性搜尋、二元雜湊樹和非對稱二元雜湊樹等三種方法搜尋 IP 的下一站位址，結果非對稱的二元雜湊樹效率最好，平均的 hash 次數只有 1.0882 次。為了加快搜尋的速度，我們加入了記憶體快取，使用的映射函數為直接和集合關聯，置換函數為 FIFO、隨機和 LRU，經過比較後發現 LRU 的效率最好且關聯集合越大時失誤率越低，平均的失誤率在 0.246249%至 0.0251518%之間(集合關聯為 1 至 8)。我們也提出一新的快取架構，不僅儲存封包的目的位址且儲存前置長度到快取記憶體中，這個方法可以增加存在快取記憶體每一筆資料的範圍，因此降低了快取失誤率，實驗結果顯示加入 length 和 longest 欄位後失誤率在 0.088154%至 0.0198498%之間(集合關聯為 1 至 8)。可見，新的快取方法能有效率的使用快取記憶體且大幅的增進路由查表效能。

參考文獻

[1] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless Inter-Domain Routing(CIDR): an Address Assignment and Aggregation Strategy. ***RFC1519***, Sep 1993.

[2] Gene Cheung, and Steve McCanne. Optimal Routing Table Design for IP Address Lookups Under Memory Constraints. In Proceedings of the ***INFOCOM 1999***, volume 3, pages 1437-1444, 1999.

[3] Henry Hong – Yi Tzeng, and Tony Przygienda. On Fast Address-Lookup Algorithms. ***IEEE Jouranl on Selected Areas in Communications***, 17(6):1067-1082, June 1999.

[4] Marcel Waldvogel, George Varghese, Jon Turner, and Bernhard Plattner. Scalable High Speed IP Routing Lookups. In ***Proceedings of ACM SIGCOMM 1997***, pages 25-36, 1997.

[5] Stefan Nilsson, and Gunnar Karlsson. IP-Address Lookup using LC-Tries. ***IEEE Jouranl on Selected Areas in Communications***, 17(6):1083-1092, June 1999.

[6] V. Srinivasan and G. Varghese. Fast Address Lookups using Controlled Prefix Expansion. ***ACM Transactions on Computer Systems***, 17(1):1-40, Feb 1999.

[7] Willibald Doeringer, Gunter Karjoth and Mehdi Nassehi. Routing on Longest-Matching Prefixes. ***IEEE/ACM Transactions on Networking***, 4(1): 86-97, Feb 1996.

[8] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink. Small Forwarding Tables for Fast Routing Lookups. In ***Proceedings of the ACM SIGCOMM 1997***, pages 3-14, 1997.

[9] Nen-Fu Huang, and Shi-Ming Zhao. A Novel IP-Routing Lookup Scheme and Hardware Architecture for Multigigabit Swiching Routers. ***IEEE Jouranl on Selected Areas in Communications***, 17(6): 1093-1104, June 1999.

[10] Pankaj Gupta, Steven Lin, and Nick McKeown. Routing Lookups in Hardware at Memory Access Speeds. In ***proceedings of INFOCOM 1998***, pages 1240-1247, 1998.

[11] Pi-Chung Wang, Chia-Tai Chan, and Yaw-Chung Chen. A Fast IP Routing Lookup Scheme. In ***Proceedings of the ICC 2000***, volume 2, pages 1140-1144, 2000.