封包分類器之二維衝突偵測

饒昕芳　　張毓騏　　方斐仙　　王丕中
國立中興大學 資訊科學與工程學系
pcwang@nchu.edu.tw

摘要

封包分類是利用事先建立的規則對封包來進行分類，它的應用包含防火牆或服務品質保證等。規則衝突檢測及衝突解析演算法是影響安全性及服務品質的關鍵。例如防火牆有著成千上萬的規則，其複雜性可能會增加配置錯誤的可能性，如果網路系統管理員不具備足夠技能和專門知識來正確配置它們，將可能對內部網路造成一定程度的威脅。因此，使用一個有效率且能夠自動檢測、發現規則衝突和潛在問題的演算法，才有便於管理人員針對產生衝突的規則做進一步的處理，以保障內部網路的安全性。本文對封包分類過濾規則之間的關係進行分析，並進一步針對規則進行分群，然後在封包分類規則分群的基礎上提出了一種衝突檢測演算法。實驗證明該演算法可以比現有的演算法有更好的時間與空間效能。

關鍵詞：封包分類、服務品質保證、防火牆、規則衝突、衝突檢測

1. 緒論

為了提供進階的網路服務，如防火牆、差異化服務[3]等，路由器須具有封包分類的功能。封包分類器是透過路由器接收封包後，擷取其表頭各欄位資訊，查詢在分類規則資料庫中預先定義的分類規則，對封包執行相對應的動作[5][16] [19][20]00。現有許多的封包分類演算法通常會預先假設在沒有任何分類規則衝突的情況下進行研究，然而在現實情況中，分類規則衝突經常發生 0。分類規則發生衝突是兩個或多個分類規則產生重疊，造成不明確的封包分類，並可能造成網路安全的漏洞、錯誤的使用者服務品質或資料傳輸[7]0。例如封包過濾(Packet Filtering)技術根據封包特性決定是否允許特定的封包通過，然而，在判斷的過程中，規則可能存在著互相牴觸的關係，尤其現今防火牆規則數目不斷增多，大型企業防火牆通常包含幾千條甚至上萬條規則。當新增規則時，新規則往往會出現與現有規則相衝突的情形，而這種衝突將可能造成潛在的安全性漏洞[6]0。因此，規則間的衝突所造成的網路安全、使用者品質服務及傳輸正確性等方面的問題，對整體網路效能和可靠度皆有影響。

規則衝突問題無法以單純的規則排序來達成預期的結果[7]，必須透過新增解析規則才能完全避免錯誤。而未來網路傳送的速度和資料量不斷地再增加，除了加快封包分類的執行速度之外，衝突偵測演算法在處理新增或是刪除分類規則時也必須有效率的運作，否則將成為路由器效能上的瓶頸。

目前針對規則衝突檢測的相關研究，因其在封包偵測時，常需要新增輔助的資料結構來加速封包分類的處理速度，隨著處理的分類規則數越多，記憶體的使用量也會隨之上升，規則的新增或是刪除也將變得複雜，連帶著規則衝突檢測的演算法效率也變差許多。而以目前網際網路發展的趨勢來看，規則資料庫所包含的規則將會逐漸增加，現有的衝突檢測演算法在檢測時間和記憶體空間上，將有很大的改善空間。而被廣泛利用的封包分類過濾器是依據來源與目的 IP 位址進行分類，管理單一及群體網際網路上的資料傳輸。以 IP 位址作為過濾器考量的欄位，可以使得路由器維持規則資料庫的循序性，並且可透過 IP prefix 進行有效的規則分類。而選用 port number 作為考量的欄位，雖然可以保持資料庫中的循序性，但以 port number prefix 進行規則的分類，所得到的效益不一定與 IP prefix 一樣高。而選用 protocol 作為考量的欄位，無法保持資料庫中的循序性，也無法將 protocol 直接按照 prefix 進行規則的分類。

本文針對依據來源與目的 IP 位址所定義的二維封包分類規則提出快速衝突偵測演算法，該演算法能在給定的規則中進行快速檢測，找出規則衝突的部分。本演算法將規則進行分群，並考量規則的幾何關係來大量減少所需要的規則比對次數。與現在的演算法相比，檢測速度更快、時間複雜度更低，實作該演算法的軟體能夠顯著簡化封包分類規則的衝突偵測，以達成有效率的規則管理和消除規則衝突。

2. 相關文獻

Hari 等提出一個基於 Grid-of-Tries 的衝突偵測演算法[7]，此演算法需要以 source address 和 destination address 欄位建立出兩個完整的遞迴樹來輔助偵測所有的衝突情形，並進而增加額外的解決分類規則解決衝突。在一個分類規則資料庫中找出所有發生衝突的分類規則，Hari 等利用 Grid-of-Tries 的資料結構和切換指標(Switch Pointer)來進行衝突偵測[10]，偵測時間為 O(nW+S)，其中 n 為規則數目，W 為最長前置字元的長度(在 IPv4 下，W=32；

在 IPv6 下，W=64)，S 為分類規則產生衝突的對數，不包含建立 Grid-of-Tries 資料結構的偵測時間複雜度為 $O(nW^2)$，空間複雜度為 O(nW)。若是採用最單純的方式則每個分類規則須逐一判斷是否跟其他分類規則有發生衝突，其時間複雜度為 $O(n^2)$。

Baboescu 和 Varghese 在前置規則(prefix filter)之間提出利用位元向量(Bit Vector, BV)的資料結構[1]，在基於聚集位元向量(Aggregated Bit Vector, ABV) [2]封包分類演算法的架構下提出了基於 ABV 的衝突偵測演算法[8]，採用聯集(Union)和交集(Intersection)運算來執行分類規則的衝突偵測。由於此作法無法預先得知需要存取的位元向量個數，因此最差情況下必須存取所有的位元向量，為了進一步改善效能，作者進一步提出新的位元向量定義，每個節點儲存兩個不同的位元向量，一個是針對該欄位包含的規則，一個則是針對該欄位被包含的規則，透過事先計算的方式，大幅降低運算的成本。假設所要判斷的分類規則欄位為來源端位址與目的端位址兩個欄位，此演算法在離線衝突偵測時間複雜度為 $O(n^2)$，空間複雜度為 $O(n^2)$。他們利用壓縮(Compressed)的二元 tries(每一個欄位需要使用一個 trie)，以及 n 位元向量。而在上線衝突偵測應用中，在時間複雜度 O(nW)內可判斷新增加的分類規則跟原有的分類規則是否發生衝突，而動態新增刪除分類規則所需的時間複雜度為 O(W)。若是採用[12]中的方式，則需要 $O(n^2)$的時間，因為每一個位元向量的大小都必須改變以加入新的分類規則。除此之外，此篇的演算法也可以運用在多維度(d > 2)的分類規則上執行分類規則的衝突偵測。

Lu 和 Sahni 沒有利用 Hari 等所提出來的定理，而是藉由偵測矩形的線段交叉點(crossing)來判斷是否發生規則衝突[9]。由於 Lu 和 Sahni 所提出的平面掃描演算法只能在空間中偵測所有存在的 Perfectly Crossing， 其演算法不能在空間中判斷是否有 Imperfectly Crossing 的存在，但 Imperfectly Crossing 的情況也會造成分類規則的衝突。為了解決這個問題，Lu 和 Sahni 定義一個操作(Operation)—擴大(Magnify，簡稱 mag)。將所有的分類規則(Trivial and Nontrivial)轉換成 Nontrivial Filter，也就是說，將所有線段重疊的重疊情況轉換成一般面積重疊的問題。分類規則擴大後仍保有衝突性，假設一個二維的分類規則 f = ([x1,x2], [y1,y2])，經由擴大的轉換後，其表示方式為 mag(f) = ([x10, x21],[y10, y21])，其中 x10 為在 x1 後面加上一個有效位元 0，x 和 y 分別代表不同維度的前置字元。Lu 和 Sahni 不僅將平面上所有 Trivial Filter 擴大轉換成 Nontrivial Filter，當任兩個分類規則發生衝突時，也會在二維平面上產生 Perfectly Crossing，如此可用來偵測任兩個分類規則是否發生衝突。而平面掃描衝突演算法所需的時間複雜度為 O(nlogn+s)，s 為分類規則發生衝突的個數或 Perfectly Crossing 的個數，空間複雜度為 O(n)。

近年來，衝突偵測領域也另外延伸出新的研究方向，例如透過兩兩規則之間沒有發生衝突，但是某些規則的聯集會跟其他規則的聯集產生衝突，或是出現某些規則完全不會被比對到的情況[11][14][15]，由於這些議題多為目前議題之延伸，在此研究中，我們先不予討論。

3. 封包分類規則與衝突

封包規則多具有以下欄位，protocol 指的是該封包所使用的網路協定類型，例如： ICMP 或 TCP/UDP 等等。來源位址(Source IP Prefix, src_ip)指的是發送端的 IP 位址，長度為 32 bit。目的地位址(Destination IP Prefix, dst_ip)指的是接收端的 IP 位址，長度為 32 bit。Port 是電腦與電腦之間的通信埠。Action 欄位所代表的是封包是否通過檢查，若接受(ACCEPT)表示可進入本機取得資源。封包分類規則之範例見表 1，表中的規則是應用在防火牆，根據事先定義好的過濾規則，檢查封包中的標頭資訊，由封包的來源位址、目的位址、來源端埠號、目的端埠號等欄位來決定是否允許資料封包的通過。

表 1 防火牆規則舉例

Order	Protocol	Source		Destination		Action
		IP Prefix	Port	IP Prefix	Port	
1	TCP	140.192.37.20	Any	*.*.*.*	80	Deny
2	TCP	140.132.37.*	Any	*.*.*.*	80	Accept
3	TCP	*.*.*.*	Any	161.120.33.40	80	Accept
4	UDP	140.192.37.*	Any	161.120.33.40	80	Deny
5	TCP	140.192.37.30	Any	*.*.*.*	21	Deny
6	TCP	140.192.37.*	Any	*.*.*.*	21	Accept

當一個封包同時符合兩個以上的規則時，常接以第一個符合的規則做為封包分類的結果。但是這個做法衍伸出規則之間的幾何關係與管理人員期望得到的結果出現落差。

當兩個分類規則的表頭內容 (來源位址、目的位址) 發生部分重疊，此時稱兩個分類規則互相發生衝突[8]。假設擁有分類規則 A 和 B。分類規則在二維空間上的幾何關係表示可為 R (A,B)，依類別大略可分為五種關係。

1. Disjoint：當兩個分類規則沒有發生交集 EX R(A,B)=disjoint,為 FS(A) ∩ FS(B)= ∮ 參照 圖 1(a)所示
2. Inside：當分類規則 B 完全被分類規則 A 所包含，R(A,B)=inside，為 FS(B) ⊂ FS(A) 參照 圖 1(b) 所示
3. Contain：當 A 分類規則被包含在 B 分類規則時，R(A,B)=contains 為 FS(B) ⊂ FS(A) 參照 圖

1(c) 所示

4. Equal：當分類規則 A 和 B 完全相等時，R(A,B)= equal，為 FS(A)=FS(B)參照 圖 1(d)
5. Overlap：當 A 和 B 兩個分類規則發生部分重疊的時候，或是幾何關係不同於上面所述之四種關係時，R(A,B)=overlap 參照 圖 1(e)

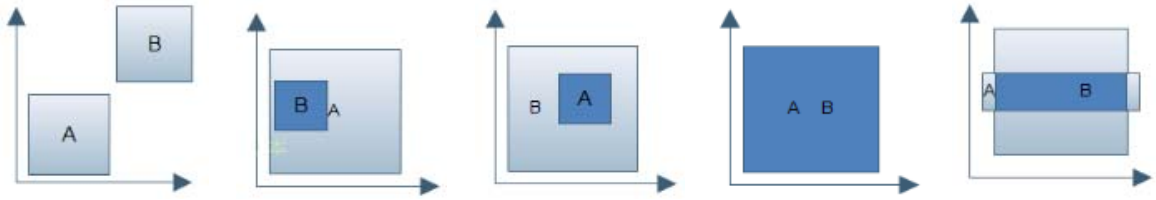

圖 1(a) Disjoint **圖 1(b) Inside** **圖 1(c) Contain** **圖 1(d) Equal** **圖 1(e) Overlap**

由以上所描述的衝突情況中，本論文所要探討的是第五種情形如 圖 1(e)所示，即兩個分類規則發生部分重疊的情況。雖然第 2 到 4 種關係也會有重疊的情況，當一個排序在後的規則能符合的所有封包，也能被一個排序在前的規則先符合到，那麼這個排序在後的規則將永遠無法被應用，如此將導致一些原先應該允許的封包被拒絕，或者應該阻擋的封包被允許進入[5]。如規則 4 被包含在規則 3 中，而規則 3 的順序優於規則 4，造成規則 4 起不了阻擋的作用，但是因為可透過調整分類規則的比對優先順序，而解決這類衝突的情形，因此在偵測衝突關係的演算法中，不把規則間包含的關係列入衝突的情形。此外，在某些應用中，規則只包含兩個欄位，因此只需要依據兩個維度的欄位進行衝突偵測，如 Quality of Service 和 Virtual Private Network[18]，本文將針對 scr_ip 及 dst_ip 兩個欄位的規則所發生的衝突情形作探討，並提出一個高速的演算法。

4. 二維衝突偵測演算法

當要偵測分類規則是否和資料庫中的其他分類規則產生衝突時，時常會與不必要的分類規則比對是否發生衝突，此時勢必會增加分類規則衝突偵測所需的時間。例如：在某些情況時 wildcard(*)太多，會造成偵測衝突的時間增加，亦增加過多的記憶體使用量[7]。根據二維分類規則的特性，prefix 之間可以簡單歸納成包含、完全不相交及被包含等三種幾何關係。因此將分類規則先做分群，分群的條件應滿足同群內規則彼此互不重疊，接下來只需再分別偵測各群間的規則衝突，進而減少不必要的衝突判斷，便可將上述的問題加以修正[7] 。

二維分類規則可以解決部分衝突發生的問題，但是在空間上還可以有改善的空間。以文獻[7] 所提之演算法為例，當偵測到衝突數量很多時，資料結構就會變得較為複雜。因此，本文所提出之演算法 Conflict Detecting using Enclosure Regions(以下簡稱 CDER)，將會把研究目的著重在簡化資料結構特性及減少空間的應用上，並提出以來源位址、目的位址兩者長度為概念的衝突偵測演算法。

本文所提出的演算法主要可分為三個步驟：

步驟一：依序讀取資料庫中每筆規則的 source IP prefix 及其長度, destination IP prefix 及其長度，再將每筆規則依據兩個長度值作為索引，存入已建立的 vector maskV[33][33]，完成分類的動作。此步驟完成後，即可將規則依據 src_ip 與 dst_ip 欄位的長度分群，依據前述的說明，同群的規則彼此之間僅可能會相同或是互斥兩種關係[16]，因此可滿足我們方法的要求。

步驟二：利用迴圈依序選定 group A, group B 在 maskV 中的 index 值傳入 Function pairnum() 進行衝突數量偵測，若選定的規則群組有其中一個群組的兩個欄位長度皆短於或等於另一個群組的兩個欄位，則不做偵測。因此在此情況下，規則之間的關係指可能是包含或是被包含其中之一。我們可以歸納出需要判斷衝突的群組之間關係如下:

1. (A.src_len > B.src_len) && (A.dst_len < B.dst_len)

2. (A.src_len < B.src_len) && (A.dst_len > B.dst_len)

當滿足上述任一種情形，必須進行衝突偵測。

步驟三：計算 smin = min(A.src_len, B.src_len) 及 dmin = min(A.dst_len, B.dst_len)， 作為需擴展的範圍大小(smin,dmin)，依據 smin 與 dmin 分兩種情形討論:

1. smin==0 && dmin==0，則表示群組 A 與群組 B 所有的規則都發生衝突，因此可以直接回傳 AB 間的衝突數為 A.size*B.size。
2. 若情形 1.不成立，則將群組 A 中所有規則分別取出 smin 與 dmin 的位元數，此作法等同於將群組 A 中的規則進行擴展，將擴展後得到的規則放入雜湊表中，依據擴展後的規則產生的 hash key 作為 index，放入雜湊表中。再將群組 B 的規則依同樣方式擴展，並以相同的 hash 函式得到 index，如果雜湊表中對應的位置內並無儲存任何規則，

亦即 hash table(index).size=0，則表示 B 群組中的此規則與所有 A 群組規則皆無衝突。如果 hash table(index).size 不等於 0，再依序與 hash table(index)中的規則比較 s.prefix 前 smin 位元與 d.prefix 前 dmin 位元是否完全相同，相同時，則兩個對應的規則為衝突，衝突累計數(counter)加 1。

接下來我們提出詳細演算法，在主函式中分別取兩兩個規則群組，並判斷此兩個群組是否符合需要進一步執行衝突偵測，如果需要時，則進入 pairnum 函式，依前述作法執行兩個群組規則的衝突偵測。我們的函式虛擬碼如下：

```
//需要判斷衝突的情形
for( select groupA ) {
  for( select groupB ){
   if( ( A.src_len > B.src_len) && (A.dst_len < B.dst_len )
    || ( A.src_len < B.src_len) && (A.dst_len > B.dst_len ))
      pairnum();
   else
      select another group as group B
  }
}

void pairnum(){
 //宣告 hash table
 vector<Connection> borden[37];
 //取得需擴展的範圍大小
  s_min( a.src.len, b.src.len );
  d_min( a.dst.len, b.dst.len );
 //處理一定發生衝突的 case
  if ( s_min == 0 && d_min == 0 )
     counter += ( groupA.size * groupB.size );
  else{
     for( each rule in groupA ){
     //擴展 src_IP
       for ( i = 0 to s_min )
        src_ip *= random_S;
  //擴展 dst_IP
       for ( i = 0 to d_min )
         dst_ip *= random_D;
        HT_index = (src_ip ^ dst_ip) % HT.size();
  }
   for( each rule in groupB ){
    //擴展 src_IP
     for ( i = 0 to s_min )
  src_ip *= random_S;
       //擴展 dst_IP
       for ( i = 0 to d_min )
          dst_ip *= random_D;
       HT_index = (src_ip ^ dst_ip) % HT.size();
       //偵測衝突
          if ( HT_index.size > 0 ){
           for ( each element in HT ){
            for ( each groupB.element_src )
             if ( groupB.element _src != HT.element _src    ){
                conflict_flag = 0;
              break;
             }
            else
               continue;
            for ( each groupB.element _dst )
              if ( groupB.element _dest != HT.element _dst ){
                 conflict_flag = 0;
                 break;
                }
              else
                 continue;
              if ( conflict_flag != 0 )
                 counter++;          }
          }
        }
     }
}
// HT: Hash Table
```

當兩個群組的規則需要進一步的偵測衝突時，需要執行的動作分別為第一個群組規則執行插入雜湊表以及使用第二個群組的規則至雜湊表查詢是否有衝突的規則，因此執行的時間複雜度為 O(|A|+|B|)，|A|與|B|分別表示群組 A 與 B 的規則數。比起基本的一對一規則比對所需要的複雜度 O(|A|*|B|)，新的雜湊表方式可以大幅減少時間複雜度。

5. 效能分析

本文所設計的實驗，係根據前段所述之衝突偵測演算法，計算程式執行所需的時間，並同時分析執行所需要的記憶體空間。執行的時間包含建立資料庫與執行衝突偵測，我們進一步針對兩段時間進行分析。

5.1 模擬環境與參數設定

本論文在模擬衝突實驗中所需的分類規則由資料庫利用 Taylor 和 Turner 所提供的 ClassBench[13](用來測試封包演算法的工具)。此工具可以模擬出現實分類規則資料庫(Real Filter Sets)的特性。在此基礎下可以產生出三類接近於現實的封包分類規則資料庫。分別是防火牆(Firewalls, FW)、存控制列表(Access Control Lists, ACL)、和分類規則鏈(IP Chains, IPC)。透過 ClassBench 提供的十二種不同的資料庫特性檔(Seed)，十二組的規則數目約為 16K，共包含五個取控制列表規則資料庫 ACL1、ACL2、ACL3、ACL4 及 ACL5，五個防火牆規則資料庫 FW1、FW2、FW3、FW4 及 FW5，以及兩個分類規則鏈規則資料庫 IPC1 及 IPC2。實驗環境於 Windows 7 作業系統及 CPU 為 INTEL XEON 2.33 GHz 的個人電腦上進行實驗模擬。

利用前述 ClassBench 所產生出不同類型的 16K 規則，每一組待測試的資料庫分別利用位元向量快速衝突偵測演算法(SBV)[8]，以及本論文提出之以 prefix 長度分群的衝突偵測演算法，進行各十次的模擬實驗，分別記錄每次執行的時間，取平均值後做為本次實驗執行偵測所需時間。而記憶體空

間則以實際分類規則後，以及本身程式所需的變數及資料結構，計算此實驗記憶體空間的使用量。

5.2 實驗結果分析

我們先列出 CDER 演算法的詳細數據資料，表 2 為所有資料庫偵測衝突所得到的衝突數量以及所需之時間。平均偵測時間表示一個規則執行衝突偵測所需的平均時間，時間單位為毫秒(ms)，平均建立時間代表從讀取資料到建立完資料結構所需的時間，為評估建立時間與規則數量的關係，我們亦採用將全部建立時間除以規則數目來得到平均每規則的建立時間，單位為毫秒(ms)。由表 2 的數據中，我們可以看出建立時間與規則衝突的數量並無直接相關，但是偵測時間卻與衝突數量呈現負相關，原因在於CDER 演算法在減少衝突偵測需要判斷的數量時有相當明顯的改善，因此當衝突數量多時，CDER 可以得到很好的效能表現(如 fw1~fw5 資料庫)。但是當規則資料庫中分群的組數較多時(如 acl1、acl5 與 ipc1 資料庫)，因為需要建立較多的雜湊表來分群，連帶需要判斷的群組對數也會增加，進而導致平均偵測時間拉長。

表 2 CDER 偵測衝突數量與所需之時間

資料庫	衝突數量	平均偵測時間(ms)	平均建立時間(ms)
acl1	294	5.49286	0.103
acl2	1286422	0.00920	0.095
acl3	1065847	0.01809	0.089
acl4	889072	0.02507	0.095
acl5	3070	0.87446	0.064
fw1	32846051	0.00008	0.089
fw2	16729400	0.00002	0.093
fw3	36463926	0.00002	0.086
fw4	17527431	0.00001	0.082
fw5	40550883	0.00003	0.082
ipc1	704463	0.03281	0.092
ipc2	15156444	0.00001	0.098

我們進一步與 SBV 演算法比較衝突偵測時間及記憶體空間的使用量。圖 2 顯示兩個方法在平均偵測時間的比較。可以看出 SBV 演算法的偵測時間相對穩定，但是在大多數情況下，都需要遠比 CDER 演算法要多的偵測時間，尤其當產生的衝突數量很多時，CDER 與 SBV 的差距也會跟著拉大。

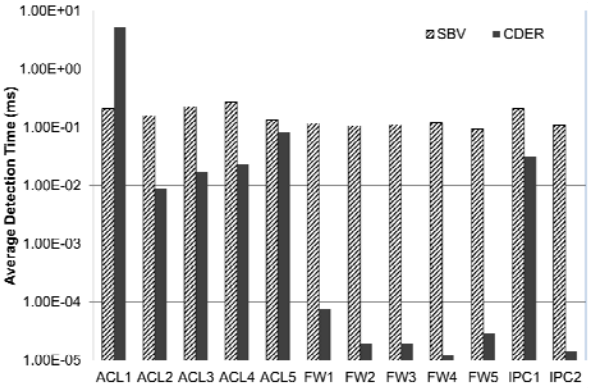

圖 2 平均偵測時間

接下來我們比較兩個方法在資料結構建立所需的時間。圖 3 顯示兩個方法在平均建立資料結構所需的時間，對兩個演算法來說，這部分所需時間都與規則衝突的數量無關。整體而言，CDER 只需要遠短於 SBV 的時間即可完成資料結構的建立。因此表示路由器不須隨時維護此資料結構，僅在有新規則插入時再建立即可。可以大幅減少路由器維護資料結構的成本。

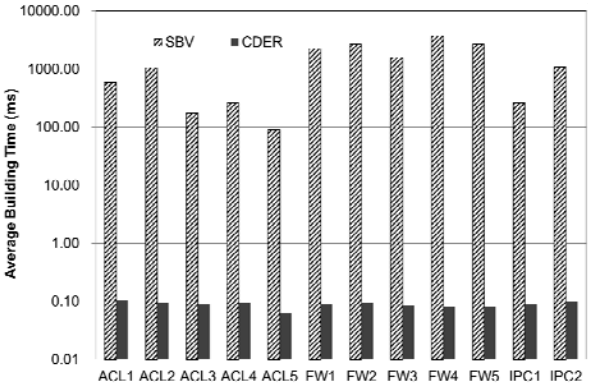

圖 3 平均建立資料結構時間

最後我們比較兩個方法資料結構所需的記憶體。如圖 4 所示，CDER 需要的空間相對十分穩定，而且與規則衝突的數量無關。對於 SBV 演算法來說，其所需的空間遠大於 CDER，由於 SBV 演算法的空間取決於兩個欄位不同欄位值的個數，加上其使用的位元向量長度等於所有規則的數量，因此當規則數增加時，所需的記憶體空間亦隨之快速增加。因此，在空間部分的表現，CDER 的效能遠勝於 SBV。

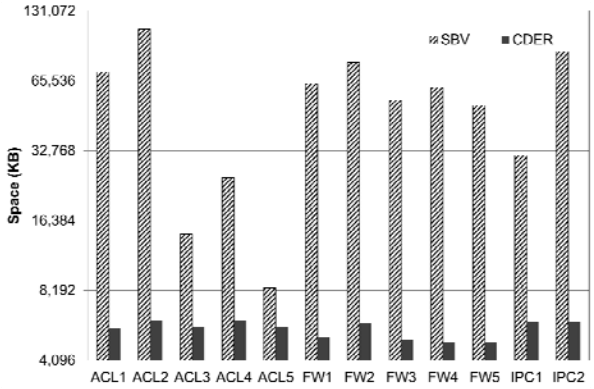

圖 4 記憶體空間

6. 結論

衝突偵測對於封包分類器的管理扮演了重要的角色。本文首先對防火牆過濾規則進行分類，並提出了過濾規則的五種關係，定義各規則間產生衝突情形。然後在此基礎上提出了防火牆規則衝突檢測之演算法。該演算法利用雜湊表來進行快速偵測，透過偵測規則衝突的數量，協助網路管理人員找出衝突的規則，並進行調整，以確保整體網路的安全。由實驗結果得知，本文 CDER 演算法針對產生較少衝突數量的資料庫，效能較不顯著。然而在產生較多衝突數量的資料庫中，能彰顯本文 CDER 演算法的效能。對於現今大型企業所使用之大型防火牆資料庫中，本文 CDER 演算法能以更快的速度，找出大量衝突的情形。

在未來的研究中，我們期望不論資料庫產生衝突數量的多寡，皆能展現本文 CDER 演算法的效能，也希望能將目前僅考慮兩個維度的欄位:來源端位址與目的端位址，擴展成多個維度的欄位，並依據每個欄位的性質來做相對應的分群，以增加本文 CDER 演算法的運用彈性，更能符合目前普遍網路的需求。

7. 參考文獻

[1] S. Blake, D. Black, M. Carlson, “An architecture for differentiated services,” IETF RFC2475, 1998.

[2] P. Gupta and N. McKeown, “Packet Classification on Multiple Fields,” in ACM SIGCOMM, September 1999, pp.147–160.

[3] V. Srinivasan, G. Varghese and S. Suri, “Packet Classification using Tuple Space Search,” in ACM SIGCOMM, September 1999, pp. 135–146.

[4] Pi-Chung Wang, Chia-Tai Chan, Shuo-Cheng Hu, Chung-Liang Lee and Wei-Chun Tseng, “High-Speed Packet Classification for Differentiated Services in NGNs,” IEEE Transactions on Multimedia, Vol. 6, No. 06, December 2004, pp. 925-935.

[5] Pi-Chung Wang, Chia-Tai Chan, Chun-Liang Lee and Hung-Yi Chang, “Scalable Packet Classification for Enabling Internet Differentiated Services,” IEEE Transactions on Multimedia, Vol. 8, No. 06, December 2006, pp. 1239-1249 .

[6] Pi-Chung Wang, Chun-Liang Lee, Chia-Tai Chan and Hung-Yi Chang, “Performance Improvement of Two-Dimensional Packet Classification by Filter Rephrasing,” IEEE/ACM Transactions on Networking, Vol. 15, No. 04, August 2007, pp. 906-917.

[7] Pi-Chung Wang, “Scalable Packet Classification with Controlled Cross-producting,” Computer Networks, Vol. 53, Issue 6, 2009, pp. 821-834.

[8] Alex X. Liu. Firewall policy change-impact analysis. ACM Trans. Internet Technol. 11, 4, Article 15 (March 2008), 24 pages .

[9] A. Hari, S. Suri, and G. Parulkar, “Detecting and resolving packet filter conflicts,” in Proc. IEEE INFOCOM, 2000.

[10] M. A. Benelbahri and A. Bouhoula, “Tuple Based Approach for Anomalies Detection within Firewall Filtering Rules,” IEEE/ISCC, 2007.

[11] V. Srinivasan, G. Varghese, S. Suri and M. Waldvogel, “Fast and Scalable Level Four Switching,” in ACM SIGCOMM, September 1998, pp. 191–202.

[12] T.V. Lakshman and D. Stidialis, “High Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching,” in ACM SIGCOMM, September 1998, pp. 203–214.

[13] F. Baboescu and G. Varghese, “Scalable packet classification,” in ACM SIGCOMM ’01, August 2001, pp. 199–210.

[14] F. Baboescu, G. Varghese, “Fast and scalable conflict detection for packet classifiers,” Computer Networks, vol. 42, no. 6, pp. 715–735, August 2003.

[15] H. Lu and S. Sahni, “Conflict Detection and Resolution in Two-Dimensional Prefix Router Tables,” IEEE/ACM Transactions on Networking, December 2005.

[16] Y. Yin, Y. Katayama, and N. Takahashi, “Detection of Conflicts Caused by a Combination of Filters Based on Spatial Relationships, ″ Journal of Information Processing, August 2008.

[17] A. X. Liu and M. G. Gouda, “Complete Redundancy Removal for Packet Classifiers in TCAMs,” IEEE Transactions on Parallel and Distributed Systems, April 2010.

[18] A. X. Liu, C. R. Meiners, and Y. Zhou “All-Match Based Complete Redundancy Removal for Packet Classifiers in TCAMs,” in Proc. IEEE INFOCOM, 2008.

[19] Vijay P. Kumar, T. V. Lakshman and Dimitrios Stiliadis, “Beyond Best Effort: Router Architectures for the Differentiated Services of Tomorrow’s Internet,” IEEE Communication Magazine, vol. 36, no. 5, pp. 152–164, 1998.

[20] David E. Taylor and Jonathan S. Turner, “ClassBench: A Packet Classification Benchmark,” in Proc. IEEE INFOCOM ’05, March 2005.