以 TCAM 實作二階段多重特徵字串比對

黃德成
中興大學
資訊科學與工程學系
huangdc@dragon.nchu.edu.tw

羅鴻政
虹晶科技
股份有限公司
hclo@socle-tech.com.tw

賴秉樑
中興大學
資訊科學與工程學系
phd9512@cs.nchu.edu.tw

陳育毅
中興大學
資訊管理學系
chenyuyi@nchu.edu.tw

摘要

由於網路入侵偵測系統的重要性與日俱增，對於多重特徵字串比對演算法處理速度上的要求也越來越高。以軟體實作的比對演算法，已不足以應付現今網路流量速度，爲改善軟體上處理速度的不足，本論文提出以三態內容可定址記憶體（Ternary Content Addressable Memory, TCAM）實作的二階段多重特徵字串比對方法。藉助 TCAM 本身所擁有的快速多元比對能力，對輸入字串進行第一階段快速過濾，再搭配靜態隨機存取記憶體（Static Random Access Memory, SRAM）用以儲存特徵字串表，從中抓取出完整的特徵字串，執行第二階段的再確認比對。透過此二階段的比對，可以快速且正確的處理特徵字串比對問題，根據實驗分析，執行第二階段 SRAM 搜尋比對的機率在 0.5023%以下，因此，整體的處理速度決定於 TCAM 的比對速度。此外，為加快特徵字串比對處理速度，我們將特徵字串平行儲存於 TCAM 中，讓 TCAM 每執行一次比對可同時處理多個位置，減少 TCAM 的總比對次數。預估使用 266MHz 的 TCAM，來處理 Snort 所擷取出的 2,082 筆特徵字串，且每筆特徵字串在 TCAM 中平行儲存四筆的條件下，使用 48,677Bytes 的 TCAM 記憶體空間，可達到 8Gbps 的處理速度。

關鍵詞：TCAM，入侵偵測，字串比對

Abstract

This paper presents a two pass multiple pattern matching method with Ternary Content Addressable Memory to improve the deficiency of software-based algorithm. At the first pass, we use TCAM to fast filter the input pattern of incoming packet, and then using Static Random Access Memory to store and fetch the intact pattern to perform re-compare at the second pass after the first. Based on the two pass processes, we can handle pattern matching problem fast and correct. Moreover, we add a specific queue between the two pass processes to ensure it can be performed in parallel to reach the best performance. By experimental results, we get that the matching probability of performing SRAM lookup is less than 0.523%. Therefore, the processing speeds of pattern matching mainly depend on the rate of TCAM lookup. In order to accelerate the processing speeds of pattern matching, we store patterns in TCAM more times to make TCAM be able to carry out multiple positions at one lookup. If we use 266MHz TCAM to deal with 2,082 Snort pattern, and each pattern is stored four times in TCAM, then the method can achieve 8Gbps with total 48,677 Bytes TCAM memory spaces.

Keywords: **TCAM, Pattern Matching, Intrusion Detection**

1. 前言

近年來由於網路的蓬勃發展，網路攻擊和病毒蠕蟲的種類也因此快速增加。現今的網路安全防護，大多是藉由使用者端以修補系統漏洞或是安裝安全防護軟體等方式來做防護。在使用者端被動式的防護並不能有效的解決問題，對於新式的網路攻擊，也缺乏立即性的應變能力。若可以在網路端就立即偵測出含有惡意攻擊的封包，並且即時過濾，將能夠更有效的做好網路安全防護。除此之外，在網路端對特徵字串和病毒碼資料庫進行更新能夠更快速的對新式攻擊做出應對防護。

網路入侵偵測系統（NIDS）就是以檢測網路封包是否帶有惡意攻擊為目標的系統，透過搜尋封包的標頭以及 Payload 中是否帶有某些特徵字串來偵測網路攻擊以及其他異常行為，並且對有問題的封包採取應有的防範措施。而 Snort [1]就是其中一種公開原始碼的網路入侵偵測系統。Snort 擁有許多惡意攻擊的規則，在這些規則中絕大部分都包含有某些特徵字串，可利用這些特徵字串來對封包做檢測。

因此，必須要有一個快速比對演算法來處理這數千筆特徵字串。目前 Snort 是以軟體來實作其比對演算法，比對速度約可達到 100Mbps 左右，但這速度仍然不足以應付現今的網路速度標準。有鑑於此，在本論文中希望藉助硬體快速的特性來加快特徵字串比對的處理速度。

而 TCAM 本身即具有快速以及多元比對的能力，正適合用來處理多重特徵字串比對問題。但是受限於 TCAM 寬度固定的特性，無法直接用來處理過長的特徵字串。因此，在本論文中提出了以 TCAM 實作的二階段多重特徵字串比對方法，以解決特徵字串長度的問題。在第一階段處理中，使用 TCAM 對輸入字串進行快速過濾，再搭配 SRAM 儲存特徵字串表，抓取出完整的特徵字串，對 TCAM 比對結果進行第二階段的再確認處理。透過此二階段的比對，可以快速且正確的處理特徵字串比對問題，並且利用佇列架構讓此二階段動作同時執行，以獲得 TCAM 可不間斷執行比對無須等待的好處。此外為加快整體特徵字串比對的處理速度，我們將特徵字串重複多筆平行儲存於 TCAM 中，讓 TCAM 在執行一次比對中可以同時處理多個位置，以獲得快速位移比對的效果。進而對於 TCAM 所使用記憶體空間加以考量，以抽取部份位元的方式來儲存特徵字串，減少因為平行儲存而增加的 TCAM 記憶體空間使用量，以期達到處理速度快以及 TCAM 使用記憶體空間少的雙重目標。

2.相關研究

在此章節中，將對過去已提出的幾個演算法做一些探討。如 Boyer-Moore、Knuth-Morris-Pratt（KMP）、Aho-Corasick、Wu-Manber、Bloom Filter 以及 TCAM-based 等等比對演算法。首先在 2.1 節，針對這些演算法做一些簡單的描述，對其稍有認知後，進而再去比較各演算法有何優缺點。接著在 2.2 節，則對 Snort 這套入侵偵測系統做些許介紹，以及針對 Snort 規則中的特徵字串做一些分析。

2.1 特徵字串比對演算法

Boyer-Moore[2]以及 KMP[3]此二種比對演算法都是藉由建立位移表來加快特徵字串比對的處理速度。對於長度 n 的輸入字串以及長度 m 的特徵字串，此二種演算法的時間複雜度分別為 $O(nm)$ 和 $O(n+m)$。不過二者都僅僅侷限於單一特徵字串比對，若要做到多重特徵字串比對，必須要付出相當程度的代價。假設要比對 k 筆特徵字串，時間複雜度可能成長為 $O(knm)$ 和 $O(k(n+m))$。Z. K. Barker 等人提出以改良的 KMP 演算法，搭配管線化技術（Pipeline）實作出多重特徵字串比對架構[4]。雖然可以做到快速多重特徵字串比對，不過處理速度仍然受到特徵字串數量多寡的影響。

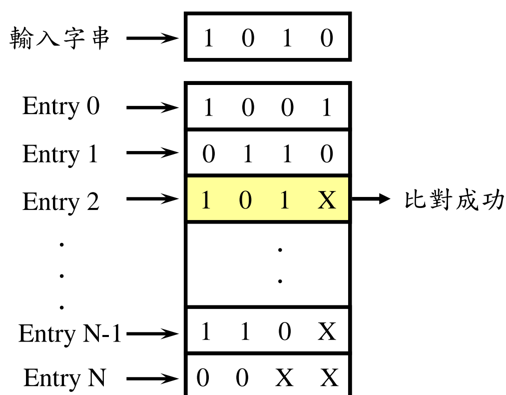
Aho-Corasick [5]是一個能夠有效同時比對多筆特徵字串的演算法。透過預先將需要比對的特徵字串建立一個有限狀態機，以及狀態之間不斷的轉換來對輸入字串進行比對，對長度 n 的輸入字串處理時間複雜度為 $O(n)$，可以快速的執行比對，但隨著特徵字串數量增加，有限狀態機的 state 也相對跟著增加。因此對記憶體的需求量相當大，當特徵字串數量多到某一程度以上時，可能造成記憶體不足的情形發生。C. J. Coit 等人將 Aho-Corasick 以及 Boyer-Moore 這二套特徵字串比對演算法合併，提出 AC_BM 演算法[6]，可加快特徵字串比對速度，不過此演算法只著重於速度上的考量因而使用更大量的記憶體空間。另外，Bro[7]此入侵偵測系統，使用類似有限狀態機的方法來實作特徵字串比對演算法，但由於產生出相當大量的 state，以致於只有少部分的 state 可存放於記憶體之中，執行比對時再動態延伸出其餘部分的 state 以解決記憶體不足問題，不過以這樣的處理方式似乎降低整體系統效能。

目前，Snort 所使用的比對演算法就是 Wu-Manber [8]，此演算法利用建立位移表快速搜尋特徵字串字尾部分，再對輸入字串進行比對。時間複雜度為 $O(bn/m)$，b 為雜湊函數計算時間，n 為輸入字串長度，m 為最短特徵字串的長度。此演算法的處理速度與最短特徵字串長度成正比，適用於長特徵字串集合的比對。根據實驗的分析，Snort 規則中擁有不少 1 到 4 Bytes 長度的特徵字串，若使用此演算法，較短的特徵字串可能需要另外處理。

Bloom Filter[9]，由 Burton Bloom 所提出，用來表示一個集合所包含的元素狀況 $A=\{a_1, a_2, \ldots, a_n\}$，以及快速判斷某一元素是否為集合成員。不過 Bloom Filter 有一定的機率將不屬於集合的元素錯誤判斷為屬於集合，稱之為 “False Positive ”。Dharmapurikar 等人提出以多個 Bloom Filter 同時處理不同長度的特徵字串[10]，可以在短時間內判斷輸入字串是否為特徵字串。但隨著特徵字串數量的增加，比對錯誤的機會也相對提高。此外，必須對各個可能長度的特徵字串建立一個 Bloom Filter，對於長

度分佈太廣的特徵字串集合並不適用。處理時間複雜度為 $O(bn)$，b 為 Bloom Filter 雜湊函數計算時間，n 為輸入字串長度。

內容定址記憶體（Content Addressable Memory, CAM）是一種可以進行高速多元比對的記憶體，每一次輸入可以同時對所有 Entry 內儲存的字串做比對。Entry 由固定長度的 Cell 所組成，每一個 Cell 可儲存一個位元的資料(0 或 1)。而三態內容定址記憶體(Ternary Content Addressable Memory, TCAM）如圖一所示，除了擁有以上 CAM 的種種特性外，比 CAM 更特別的是 TCAM 的每一個 Cell 可儲存三種狀態（0、1 及"X, Don't Care"）。



圖一、TCAM 比對示意圖

藉由"Don't Care"此狀態，TCAM 可以獲得以下幾種好處。第一，可直接處理小於 TCAM 寬度的字串，只需要把多餘的 Cell 設為"Don't Care"狀態。第二，可以不用考慮英文字母大小寫的差異性。在 ASCII 碼中，英文字母大小寫差距恰好為 32，因此在不考慮大小寫情況時，可以將字元的第五個位元設為"Don't Care"狀態。舉例來說"*A*"的 ASCII 碼為 65（0100 0001），而"*a*"的 ASCII 碼為 97（0110 0001），在不考慮大小寫情況時，能夠以（01*0 0001）的形式儲存在 TCAM 中。

M. Gokhake 等人利用 CAM 來處理特徵字串比對的問題[11]，對於 k 個 w Bytes 長的特徵字串，只需使用 kw Bytes 的儲存空間，即可在 $O(n)$ 的時間複雜度內，處理完長度為 n Bytes 的輸入字串，不過侷限於 CAM 的特性，只適用於固定長度為 w Bytes 的特徵字串比對。因此，許多研究轉而利用 TCAM 處理特徵字串比對問題，例如，[12,13]利用 TCAM 的"Don't Care"狀態，來處理封包分類問題。

Fang Yu 等人提出以 TCAM 來實作的多重特徵字串比對演算法[14]，可以處理多種狀況下的特徵字串。主要想法是將先比對到的 prefix 字串暫存，之後比對到後續部份，再藉由查表來判斷其組合是否可以對應到某一特徵字串。

此方法藉由 TCAM 的快速多元比對能力，處理速度可達到 1-2 Gbps 左右。但是此方法額外所需要的 Table 空間卻相當浪費，而且在速度上，一次僅僅處理一個位置，還有可以加速的空間。

2.2 入侵偵測系統 - Snort

Snort 是一公開原始碼的 NIDS，主要用來監視電腦與網路，以及偵測網路攻擊，並且在分析其偵測結果後回報管理者採取防禦措施。在目前 Snort 擁有 3,565 筆規則，包括 backdoor、DoS（Denial of Service）、DDoS（Distributed Denial of Service）等等入侵類型的規則。而 Snort 的規則可以分成規則標頭（Rule Headers）以及規則選項（Rules Options）二部份。規則標頭部份包含有此規則的動作、通訊協定、來源及目的 IP 位址和埠（Port）；而規則選項部分，可分成以下四類：

一、Meta-Data Rule Options：
　提供有關此規則的資訊，對封包偵測無任何影響。

二、Payload Rules Options：
　規定在封包 Payload 中須偵測的內容。

三、Non-Payload Rules Options。
　規定在封包 Non-Payload 部分須偵測的內容。

四、Post-detection Rules Options：
　用以規定當規則成立時要觸發的動作。

Snort 的規則中，絕大部份的規則都包含有 Payload Rules Options，也就是必須在封包 Payload 中搜尋某一特徵字串。根據我們對 Snort 所提供的規則（2.4 版）去做分析的結果，此版本共包含了 3,565 筆規則，我們從中擷取出 2,082 筆特徵字串作為我們的測試特徵字串。長度介於 1 Byte 至 107 Bytes 之間，而且絕大部分的特徵字串長度皆在 50 Bytes 以下，表一列出各長度特徵字串數量。由表一可以計算出此 2,082 筆特徵字串的平均長度為 15.87 Bytes，這說明了在 Snort 系統中以較短的特徵字串為多。

表一、各長度特徵字串的數量

Snort 各長度特徵字串分布表				
1 : 7	21 : 32	41 : 6	61 : 0	.
2 : 15	22 : 25	42 : 17	62 : 0	.
3 : 43	23 : 28	43 : 16	63 : 0	.
4 : 127	24 : 13	44 : 10	64 : 0	101 : 0
5 : 107	25 : 12	45 : 11	65 : 1	102 : 0
6 : 86	26 : 23	46 : 9	66 : 0	103 : 0
7 : 89	27 : 15	47 : 6	67 : 0	104 : 0
8 : 118	28 : 17	48 : 7	68 : 0	105 : 0
9 : 118	29 : 21	49 : 6	69 : 0	106 : 0
10 : 113	30 : 14	50 : 6	70 : 0	107 : 1
11 : 126	31 : 27	51 : 0	71 : 0	
12 : 134	32 : 15	52 : 2	72 : 0	
13 : 115	33 : 18	53 : 2	73 : 0	
14 : 83	34 : 23	54 : 0	74 : 0	
15 : 58	35 : 23	55 : 0	75 : 0	
16 : 67	36 : 18	56 : 1	76 : 0	
17 : 42	37 : 17	57 : 1	77 : 0	
18 : 46	38 : 14	58 : 0	78 : 0	
19 : 45	39 : 15	59 : 1	79 : 0	
20 : 47	40 : 21	60 : 1	80 : 1	

3. 系統架構

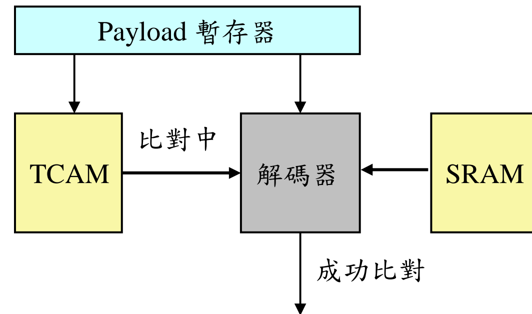
經過以上的相關研究分析，我們歸納出使用 TCAM 來處理多重特徵字串比對的二點訴求。第一點，快速的處理速度，第二點，少量的 TCAM 使用記憶體空間。在 3.1 節中，將針對使用 TCAM、加快比對處理速度以及減少 TCAM 使用空間所產生的問題加以討論並且提出解決方式，進而在 3.2 節提出本論文特徵字串比對方法的整體架構，並介紹整個架構的運作流程。

3.1 問題分析

首先在 3.1.1 節中，將介紹以二階段處理來解決長特徵字串無法直接儲存於 TCAM 中的問題。接著在 3.1.2 節，討論以平行儲存來加快比對處理速度的優缺點。在 3.1.3 節則針對平行儲存所造成 TCAM 使用空間大幅成長提出應對的解決方法。

3.1.1 二階段處理架構

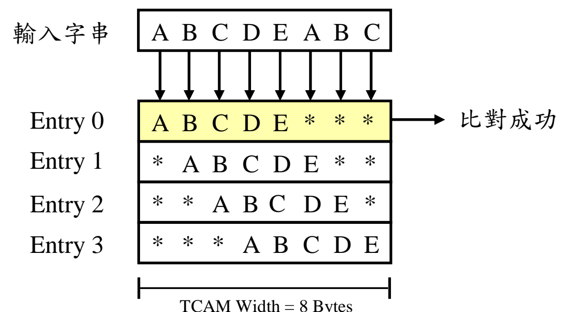
使用 TCAM 來處理特徵字串比對第一個會面臨到的問題，就是特徵字串長度的問題。由於 TCAM 寬度是有限的，長度大於 TCAM 寬度的特徵字串，無法直接儲存在 TCAM 中進行比對處理。爲解決此問題，我們來以二階段來處理長特徵字串，也就是 TCAM 只儲存長特徵字串前半部份固定長度字串，再搭配 SRAM 所儲存的完整特徵字串，進行第二階段處理。藉由 TCAM 對前半段固定長度字串進行過濾以及在 SRAM 中抓取出完整特徵字串執行再確認比對，此二階段的處理來解決特徵字串長度問題。當 TCAM 比對到長特徵字串所儲存的 Entry，將特徵字串索引值傳送至解碼器計算出此特徵字串在 SRAM 中的位置，抓取出完整特徵字串；再根據先前 TCAM 比對到位置從 Payload 暫存器中抓取出對應輸入字串，進行第二階段的再確認比對，如圖二所示。



圖二、二階段處理架構

3.1.2 平行儲存技術

一般來說，爲避免錯失任何位置比對到特徵字串的可能性，絕大部分的多重特徵字串比對方法都是一次位移一個位置來對輸入字串進行比對。爲加快執行特徵字串比對的處理速度，我們藉由平行儲存來減少 TCAM 比對次數，也就是說，將特徵字串重複 m 次儲存於 TCAM 中，每當 TCAM 執行比對等同於一次處理 m 個相異位置。以圖三為例，特徵字串重複四次儲存於 TCAM 中，每當 TCAM 執行一次比對可同時處理四個位置，總比對次數可以減少為原來的四分之一。當整體處理速度取決於 TCAM 比對次數時，平行儲存 m 次可將處理速度提升至 m 倍，也就是說以 TCAM 使用記憶體空間來換取時間。



圖三、平行儲存的方法

由於平行儲存會使得 TCAM 中的“Don’t Care”狀態增加，造成 TCAM 所使用的記憶體空間並非按照一定比例成長，因此，我們分析特徵字串平行儲存次數對“Don’t Care”狀態數量以及整體 TCAM 使用記憶體空間的影響。首先我們分析特徵字串平行儲存次數 m 與“Don’t Care”狀態數量 N_{DC} 之間的關係，由於每增加一筆儲存次數，“Don’t Care”狀態數量以二倍的等差數列成長，因此，可推導出“Don’t Care”狀態的數量 N_{DC} 與平行儲存次數 m 二者之間的關係如定理一：

定理一： “Don’t Care”狀態的數量 N_{DC} 與平行儲存次數 m 的關係為：

$$N_{DC} = m(m\text{-}1), \quad m \geqq 2$$

而特徵字串平行儲存 m 次之後，整體 TCAM 使用的記憶體空間如定理二：

定理二： 對於原本儲存於 TCAM 寬度為 w Bytes 的特徵字串，在特徵字串平行儲存 m 次之後，整體 TCAM 使用的記憶體空間 S_m 變為：

$$S_m = wm + m(m\text{-}1), m \geqq 2$$

因此，特徵字串平行儲存 m 次之後，對整體 TCAM 所使用記憶體空間成長倍率的影響如定理三：

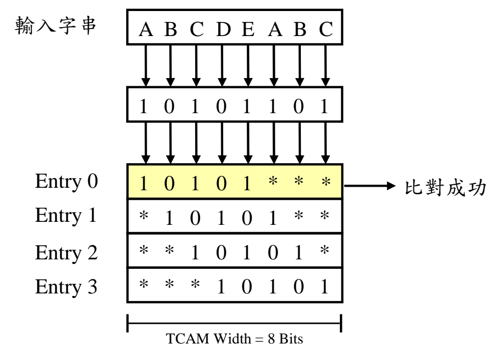
定理三： 特徵字串平行儲存 m 次於 TCAM 之中，對整體 TCAM 所使用記憶體空間成長倍率 T_m 的影響為：

$$T_m = \frac{wm + m(m\text{-}1)}{w}, m \geqq 2$$

特徵字串平行儲存雖然可以獲得在一次 TCAM 比對中可同時處理多個位置，以加快整體特徵字串比對速度的好處，但是相較於原來 TCAM 所使用的記憶體空間，特徵字串平行儲存 m 次也讓整體 TCAM 使用記憶體空間成長了 T_m 倍，而 T_m 絕對大於特徵字串平行儲存次數 m，且隨著平行儲存次數 m 的增加而跟著成長。使用 m 倍多的空間來換取 m 倍的速度並不符合效益。因此，爲節省 TCAM 所使用記憶體空間，進而考慮將特徵字串壓縮後再儲存於 TCAM 中，減少特徵字串在 TCAM 中佔用的記憶體空間。所採用的壓縮方式為抽取部份位元儲存，也就是特徵字串中的每一個字元只抽取出一個位元來做儲存。

3.1.3 減少 TCAM 使用記憶體空間

爲改善特徵字串平行儲存讓整體 TCAM 所使用的記憶體空間大幅成長的問題，我們改變特徵字串儲存於 TCAM 中的型態，採用的方式是將特徵字串每一個字元只抽取出一位元來儲存。舉例來說，如特徵字串“*ABCDE*”，只抽取其每一字元的最後一個位元值來做儲存，也就是將“*ABCDE*”以“10101”此五位元資料儲存於 TCAM 中，如圖四所示。採取此種儲存方式可以將 TCAM 所使用的記憶體空間減少為原來的八分之一。



圖四、特徵字串儲存於 TCAM 中的型態

此方法雖然可以有效的減少 TCAM 所使用的記憶體空間，不過也衍生出二個問題。第一，有機會出現比對錯誤的情形，也就是說當 TCAM 比對到某一筆特徵字串壓縮位元值，無法肯定是否真的比對到此特徵字串，有可能是其它同樣對應到此壓縮位元值的字串。第二，由於減少儲存的位元數，特徵字串被比對到機率也相對提高。

對於比對錯誤此問題，可以用 3.1.1 節所提出的二階段處理來解決比對錯誤的問題。也就是當 TCAM 比對到任何一個 Entry，將此 Entry 所對應的特徵字串索引值傳送到解碼器。再由解碼器計算出此特徵字串在 SRAM 中對應位置，至 SRAM 中抓取完整特徵字串進行第二階段比對，如此便可以解決比對錯誤的問題。

至於第二個問題，特徵字串被比對到機率提高，首先我們對特徵字串以每字元抽取一位元此種方式儲存之後被比對到的機率做分析。一個字元的特徵字串以一位元儲存，以機率來看，此特徵字串壓縮位元值被比對到的機會為二分之一。而二個字元的特徵字串以二位元儲存，則此特徵字串壓縮位元值有四分之一的機會被比對到。依此類推，當某一筆長度為 x 個字元的特徵字串以 x 位元儲存時，此特徵字串壓縮位元值被比對到的機率如定理四：

定理四：當一筆長度為 x 個字元的特徵字串以 x 位元儲存時，此特徵字串壓縮位元值被比對到的機率 $P(x)$：

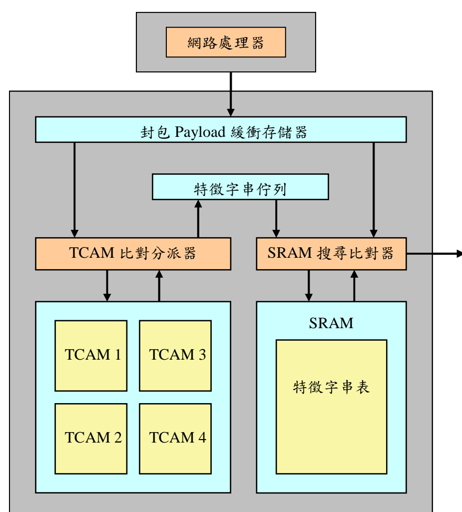
$$P(x)=\left(\frac{1}{2}\right)^{x},\ x\geqq 1$$

因此，特徵字串以每字元抽取一位元此方式儲存之後，被比對到的機會與其字串長度成反指數比，越短的特徵字串比對到的機會越高。

為解決短特徵字串壓縮位元值過易被比對到的問題，我們將特徵字串依其長度做分組，分別抽取不同的位元數儲存於 TCAM 之中，降低短特徵字串被比對到的機率。越短的特徵字串，抽取越多的位元數儲存於 TCAM 中，甚至儲存完整的特徵字串。在實驗中，將特徵字串依其長度分成四組（1-5 Bytes, 6-10 Bytes, 11-20 Bytes, 21 Bytes 以上），分別每字元抽取八位元（完整字串）、四位元、二位元以及一位元儲存於 TCAM 中。每次執行比對，則同時對四組 TCAM 所儲存資料進行比對。如此一來，便可以成功的降低短特徵字串被比對到機率，解決短特徵字串過於容易比對到的問題。

3.2 二階段多重特徵字串比對架構

經過 3.1 節的分析，我們提出了以 TCAM 實作的二階段多重特徵字串比對架構。此架構主要由六個部份所組成：封包 Payload 緩衝處理器、TCAM 比對分派器、四組 TCAM、特徵字串佇列、SRAM 搜尋比對器以及 SRAM 此六個區塊，如圖五所示，各部份的功能如下：



圖五、二階段多重特徵字串比對架構

一、封包 Payload 緩衝處理器：

儲存封包 Payload 供 TCAM 比對分派器以及 SRAM 搜尋比對器抓取字串進行比對。

二、TCAM 比對分派器：

自封包 Payload 緩衝處理器中抓取出輸入字串，分派至各個 TCAM 進行第一階段比對。且依據 TCAM 比對計算出第一階段比對到的特徵字串索引值，以及在 Payload 中比對到特徵字串的位置，將其儲存到特徵字串佇列中待第二階段處理。

特徵字串索引值=各 TCAM 特徵字串起始索引值+（TCAM Entry 索引值/m ）

比對到位置=目前位置+（TCAM Entry 索引值 mod m），m：平行儲存筆數

三、四組 TCAM：

分別以每字元抽取八位元（完整字串）、四位元、二位元以及一位元來儲存四組不同長度的特徵字串於各組 TCAM 中。

四、特徵字串佇列：

用以儲存在第一階段 TCAM 所比對到的特徵字串資訊，以供第二階段的 SRAM 搜尋比對器進行再確認比對。

五、SRAM 搜尋比對器：

自特徵字串佇列抓取第一階段比對資訊，依特徵字串索引值以及比對到特徵字串位置，分別自 SRAM 以及封包 Payload 緩衝處理器中抓取出對應特徵字串及輸入字串進行第二階段比對。

六、SRAM：

用以儲存第二、三、四組 TCAM 內壓縮位元值所對應的完整特徵字串表，以對第一階段比對結果進行再確認比對。

此架構的運作流程如下，封包經由網路處理器執行解碼，將封包的 Payload 部分儲存在封包 Payload 緩衝處理器之中。接著 TCAM 比對分派器自封包 Payload 緩衝處理器抓取輸入字串分派至各組 TCAM 中進行比對，並且依據 TCAM 比對結果計算出比對到的特徵字串索引值以及比對到特徵字串位置，儲存於特徵字串佇列之中，接著位移到下一個位置繼續抓取輸入字串進行比對，直到封包 Payload 結束，以上為第一階段的比對處理。

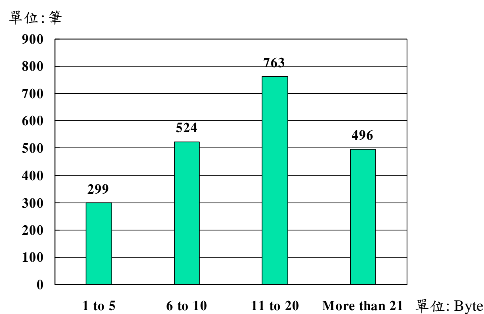
接下來 SRAM 搜尋比對器不斷去檢查特徵字串佇列是否為空，若不為空則抓取出佇列中所儲存的第一階段比對資訊執行第二階段比對處理。依據此資訊中的特徵字串索引值以及比對到位置，分別至 SRAM 及封包 Payload 緩衝處理器中抓取出相對特徵字串和輸入字串，執行再確認處理，若成功比對，則回傳比

對到的特徵字串索引值以及比對到位置。

此架構透過第一階段使用 TCAM 對輸入字串快速過濾以及第二階段的再確認動作，可以正確無誤的對封包 Payload 執行特徵字串比對。並且藉由特徵字串佇列此結構，讓此二階段處理同時動作；因此，TCAM 比對分派器可以不斷的自封包 Payload 緩衝處理器抓取輸入字串進行比對，同時 SRAM 搜尋比對器也不斷的自特徵字串佇列抓取第一階段比對結果進行比對動作。只要特徵字串佇列不是處於全滿的狀態，TCAM 就可以不斷的執行比對動作，整體的處理速度就決定於 TCAM 比對速度。

4. 實驗結果

實驗以 Snort 規則中所擷取出的 2,082 筆特徵字串作為測試特徵字串；另外，以二種形式的測試輸入資料對本方法進行測試，第一種為亂數產生的測試輸入資料，第二種為 MIT Lincoln 實驗室所提供的模擬攻擊測試檔[15]。首先，將 Snort 規則中抽取出的 2,082 筆特徵字串依其字串長度分成四組。由於考慮到每組最短特徵字串被比對到的機率以及整體 TCAM 所使用記憶體空間，因此將特徵字串分成 1-5 Bytes、6-10 Bytes、11-20 Bytes 和 21 Bytes 以上此四組，分別每字元抽取八位元（完整字串）、四位元、二位元以及一位元儲存於各組 TCAM 中，而每筆特徵字串平行儲存四筆。以此四組長度對特徵字串做分類，可將第一組以外的各組 TCAM 所儲存的每一筆特徵字串被比對到的機率降至百萬分之一以下。各組 TCAM 所包含的特徵字串數量如圖六所示。



圖六、各組 TCAM 所包含的特徵字串數量

4.1 節中將分析以上述二種輸入資料進行測試，執行第二階段 SRAM 搜尋比對的機率，以及因為抽取部份位元儲存而造成額外執行第二階段 SRAM 搜尋比對的機率。4.2 節中將對 TCAM 在儲存完整特徵字串以及儲存特徵字串部份位元，分析兩者使用記憶體的空間。

4.1 執行 SRAM 搜尋比對機率

首先，以機率的角度來估算，在理想情況下執行一次 TCAM 比對，發生額外第二階段 SRAM 搜尋比對的機率期望值 P 為：

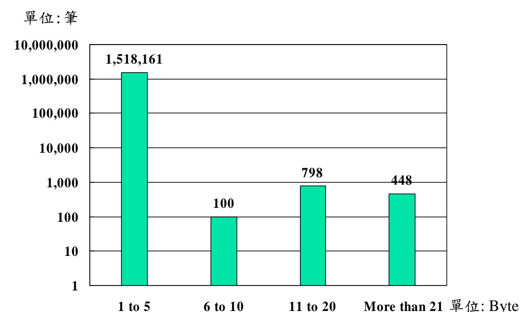
公式一： $$P = \sum_{pl=6}^{N} (\frac{1}{2})^{pl \times b} \times m$$

N 為第一組 TCAM 除外的特徵字串總數、pl 為字串長度、b 為抽取位元數、m 為平行儲存筆數。因此，根據計算結果，以上述四組長度對 Snort 所擷取出的 2,082 筆特徵字串作分類，以及每筆特徵字串壓縮位元值皆平行儲存四筆於 TCAM 中的情況下，執行第二階段 SRAM 搜尋比對的機率大約為 $P = 0.028\%$。也就是說在理想情況下，大約要執行 3,571 次 TCAM 比對才會出現一次額外比對到的情形。

接著以二種形式的測試資料來對本論文所提出的架構進行測試，第一種為亂數產生的測試輸入資料，第二種為 MIT Lincoln 實驗室[15]所提供的模擬攻擊測試檔。此測試檔為 Lincoln 實驗室在 1999 年 3 月 8 日至 3 月 12 日所錄製的實際封包資料，共包含十筆測試檔，在實驗中，我們抽取出此十筆測試檔封包 Payload 部份做為測試輸入資料。

4.1.1 亂數產生輸入資料

在第一組實驗中，以亂數產生共 100,000 筆長度為 200 Bytes 的輸入字串來進行測試，各組 TCAM 比對到的次數如圖七所示。由於第一組 TCAM（1-5 Bytes）包含過多短字串，造成亂數產生的測試資料對於這組特徵字串被比對到機會極高，但是第一組 TCAM 所儲存的是完整特徵字串，並不需要執行第二階段 SRAM 搜尋比對，因此不會影響到執行第二階段 SRAM 搜尋比對的機率。



圖七、第一組的實驗結果

執行第二階段 SRAM 搜尋比對機率 P 為第一組 TCAM 除外的各組 TCAM 比對到次數除以總比對次數。因此可以計算出亂數產生輸入測試資料執行第二階段 SRAM 搜尋比對機率 P 為：

P = (100+798+448) / (100,000*200/4) = 0.0002692 = 0.02692%

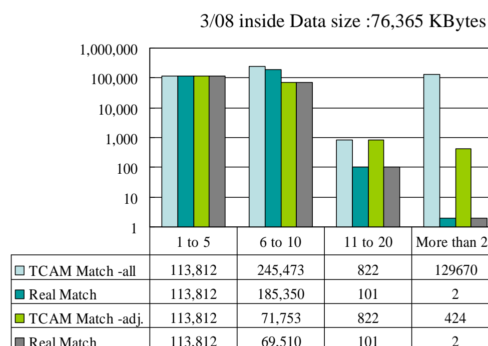
此機率略低於我們所估算的機率 0.028%，但差距不大，也就是大約執行 3,714 次 TCAM 比對，才會產生一次 TCAM 錯誤比對的情況。

4.1.2 MIT Lincoln 實驗室攻擊測試檔

第二組實驗是以 MIT Lincoln 實驗室所提供的模擬攻擊測試檔，共包含十組測試資料。在此實驗中，我們發現到在二種情況下會造成大量額外執行第二階段 SRAM 搜尋比對的機會，第一種情況為當某一特徵字串的壓縮位元值剛好對應到在 Payload 中常出現字串的壓縮位元值，此情況會造成此壓縮位元值在第一階段處理中被 TCAM 比對到的機會大幅提升。舉例來說，特徵字串“*ABCDE*”的壓縮位元值為“10101”，恰好對應到在 Payload 中常出現的字串“*EFGHI*”，每當此字串在封包 Payload 中出現，TCAM 就會比對到此位元值，也就必須多執行一次額外的 SRAM 搜尋比對。而第二種情況則為特徵字串抽取部份位元儲存時，對應到重複性的壓縮位元值，而封包 Payload 中恰好也是一連串重複性字元的輸入字串。例如，特徵字串“*AAAAAAAAAA*”其壓縮位元值為“1111111111”，當封包 Payload 中的輸入字串為一連串重複性字元且壓縮值同樣為“111111111111111111”時，便會造成大量的 SRAM 搜尋比對。

上述二種情況以第二種較為不利，因為當此情形發生時，會在短時間內造成 TCAM 比對到大量的特徵字串資訊，若第二階段的 SRAM 搜尋比對處理速度不足，造成特徵字串佇列處於全滿狀態，可能拖慢整體的比對處理速度。對於上述二種情況的解決方式如下，當第一種情況太嚴重時，可以將此特徵字串向前推移儲存至上一組 TCAM，增加此特徵字串抽取位元數，即可解決此問題。至於第二種情況無法以改變特徵字串儲存型態來改善，因此，我們增加一組 TCAM 專門用以儲存這類型的特徵字串來解決此問題。

接下來為以上述十筆模擬攻擊測試檔所做的實驗結果，將分別計算包含上述二種情況以及改善後執行第二階段 SRAM 搜尋比對機率。



	1 to 5	6 to 10	11 to 20	More than 21
TCAM Match -all	113,812	245,473	822	129670
Real Match	113,812	185,350	101	2
TCAM Match -adj.	113,812	71,753	822	424
Real Match	113,812	69,510	101	2

圖八、測試檔一的比對結果

圖八為測試檔一的比對結果，共有四筆資訊，分別為包含上述問題情況下，各組 TCAM 比對到次數和再確認後實際比對到次數，以及在改善之後各組 TCAM 比對到次數和再確認後實際比對到次數。我們分別計算在此二種情況下執行第二階段 SRAM 搜尋比對的機率以及額外執行第二階段 SRAM 搜尋比對的機率。執行第二階段 SRAM 搜尋比對的機率為：

Pa = TCAM(2, 3, 4)比對到次數/TCAM 總比對次數

而 TCAM 比對成功，但是經過第二階段 SRAM 搜尋比對再確認後，並非真正比對成功，此種由於抽取部份位元儲存而造成的額外執行第二階段 SRAM 搜尋比對的機率為：

Pe = (TCAM(2, 3, 4)比對到次數-實際比對到次數)/TCAM 總比對次數

因此，包含全部特徵字串情況下，執行第二階段 SRAM 搜尋比對的機率為：

Pa = (245,473+822+129,670) / (76,365K/4) = 0.0196930…= 1.9693%

而額外執行第二階段 SRAM 搜尋比對的機率為：

Pe = (60,123+721+129,668) / (76,365K/4) = 0.0099790…= 0.9979%

也就是平均約執行 100 次 TCAM 比對，就會造成一次額外的 SRAM 搜尋比對，此機率比我們所估算的 0.028% 高出許多。因此，我們分析各組 TCAM 中所儲存特徵字串被比對到的次數。

根據分析各組特徵字串被比對到的次數後發現，在第二組 TCAM（6-10 Bytes）中，有三筆特徵字串被比對到次數特別高，分別為：

“//////////”　：57,880
“/////////”　：57,960

"?????????? "： 57,880

其中 "*//////////*"及"*////////*"是實際比對到此特徵字串，而"?????????? "為對應到與"*//////////*" 此特徵字串相同的壓縮位元值"*FFFFFFFFFF*"（十六進位值）因而造成額外比對次數大幅增加。另外，在第四組 TCAM（21 Bytes 以上）同樣有三筆特徵字串被比對到次數特別高，分別為：

"*CCCCCCCCCCCCCCCCCCCCCCCC*"
：57,360
"*AAAAAAAAAAAAAAAAAAAAA*"
：57,486
"<><><><><><><><><><><><><><><><><>
<><><><><>" ：14,400

前二筆特徵字串"*CCC…*"及"*AAA…*"為對應到重複性字串"*///…*"的壓縮位元值"111…"所造成的額外比對。而第三筆特徵字串"<><>…"為對應到重複性字串"<00>…"的壓縮位元值"0000…"所造成的額外比對。在將此類字串進行額外處理之後，執行第二階段 SRAM 搜尋比對的機率為：

Pa = (71,753+822+424) / (76,365K/4) = 0.0038236…≒ 0.3824%

額外執行第二階段 SRAM 搜尋比對的機率為：

Pe = (2,243+721+422) / (76,365K/4) = 0.0001773…≒ 0.0177%

也就是平均約執行 5,640 次 TCAM 比對，才會造成一次額外的 SRAM 搜尋比對。此機率低於估算的 0.028%，這在合理的範圍之內。

3/08 outside Data size :56,486 KBytes

	1 to 5	6 to 10	11 to 20	More than 21
TCAM Match -all	204,784	243,623	782	177,466
Real Match	204,784	183,576	97	2
TCAM Match -adj.	204,784	69,903	782	252
Real Match	204,784	67,736	97	2

圖九、測試檔二的比對結果

圖九為測試檔二的比對結果，在未改善情況下，執行第二階段 SRAM 搜尋比對的機率為：

Pa = (243,623+782+177,466) / (56,486K/4) = 0.0298743…≒ 2.9874%

額外執行第二階段 SRAM 搜尋比對的機率為：

Pe = (60,047+685+177,464) / (56,486K/4) = 0.0168671…≒ 1.6867%

根據分析結果，在第二組 TCAM（6-10 Bytes）中，發生與測試檔一相同的情況，其中三筆特徵字串被比對到次數特別高，分別為：

"*//////////*" ： 57,880
"*////////*" ： 57,960
"?????????? "： 57,880

造成此三筆特徵字串被比對到的次數特別高的原因與測試檔一完全相同，因此不再加以說明。

而在第四組 TCAM（21 Bytes 以上）中，除了相同的三筆特徵字串被比對到次數特別高之外，還多了一筆特徵字串"FTP SERVER CHANGED TO"被比對到次數特別高，次數如下：

"FTP SERVER CHANGED TO"
：47,968
"*CCCCCCCCCCCCCCCCCCCCCCCC*"
：57,360
"*AAAAAAAAAAAAAAAAAAAAA*"
：57,486
"<><><><><><><><><><><><><><><><><>
<><><><><>" ：14,400

造成特徵字串"FTP SERVER CHANGED TO"被比對到次數特別高的原因為，此特徵字串壓縮位元值為"000011001001010110001"，恰好對應到在封包 Payload 中常出現的字串"LL LSAP INDIVIDUAL, S" 。因此，每當此字串出現在封包 Payload 中，TCAM 便會比對到此特徵字串壓縮位元值，而"LL LSAP INDIVIDUAL, S"在此測試檔中共出現 47,968 次，因此造成此特徵字串在 TCAM 中被比對到的次數為 47,968 次。為解決此特徵字串造成執行第二階段 SRAM 搜尋比對次數大幅成長的問題，我們將此特徵字串向前推移儲存至第三組 TCAM（11-20 Bytes），在改變其儲存組別後，此特徵字串就沒有再發生比對到的情形。

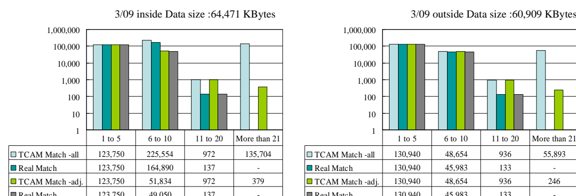
改善之後執行第二階段 SRAM 搜尋比對的機率為：

Pa = (69,903+782+252) / (56,486K/4) = 0.0050233…≒ 0.5023%

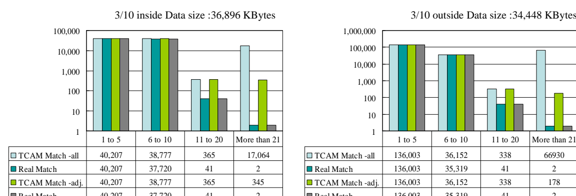
額外執行第二階段 SRAM 搜尋比對的機率為：

Pe = (2,167+685+250) / (56,486K/4) = 0.0002196…≒ 0.022%

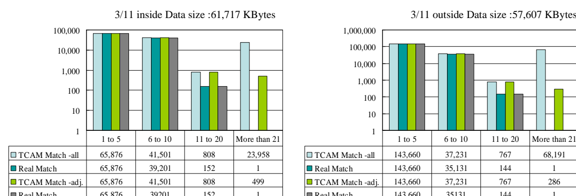
也就是說在此測試檔中平均約執行 4,545 次 TCAM 比對，才會造成一次額外的 SRAM 搜尋比對。圖十為第三到十的測試檔，發生 TCAM 比對次數大幅成長的狀況與測試檔一及測試檔二相同，因此不再詳細說明。



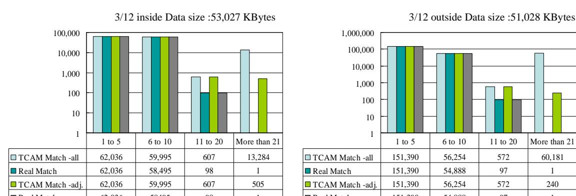
(a) 測試檔三　　(b) 測試檔四



(c) 測試檔五　　(d) 測試檔六



(e) 測試檔七　　(f) 測試檔八



(g) 測試檔九　　(h) 測試檔十

圖十、第三到十的測試檔的結果

以上的實驗測試數據，執行第二階段 SRAM 搜尋比對的機率是以平均值去做計算，並無法以此數據來判斷是否在一次 TCAM 比對過程中會出現大量的 SRAM 搜尋比對。因此，接下來將針對這一點做分析，判斷在各個封包 Payload 中是否會出現大量的 SRAM 搜尋比對，導致第二階段處理速度不及，造成整體比對處理速度延遲此情形發生。

根據實驗分析，在一組封包 Payload 中執行 SRAM 搜尋比對次數的機率如表二所示，需要執行第二階段 SRAM 搜尋比對的封包不到 4%，且其中約 94% 只需執行一次 SRAM 搜尋比對。實驗數據顯示出，在一組封包 Payload 中，最多須執行 7 次 SRAM 搜尋比對，但機率相當低，大約為 5.703e^{-5} %，而在一次 TCAM 比對中，最多造成 4 次 SRAM 搜尋比對，但同樣的機率也相當低。此外，根據我們的分析，SRAM 搜尋比對並非集中在某一區段。因此，可以判定在 TCAM 比對過程中並不會在短時間內造成大量的 SRAM 搜尋比對，導致整體比對處理速度延遲此情形發生。

表二、執行 SRAM 搜尋比對次數的機率

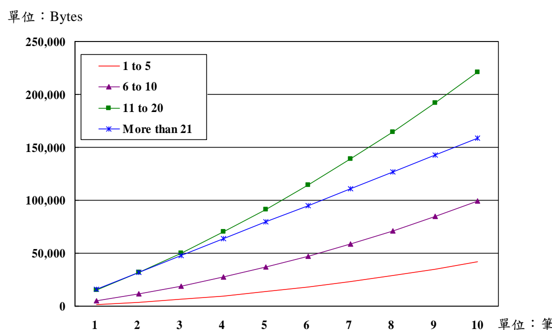
次數	機率
0	96.0548… %
1	3.742… %
2	0.194… %
3	6.559…e $^{-3}$ %
4	2.395…e $^{-3}$ %
5	9.125…e $^{-4}$ %
6	5.703…e $^{-5}$ %
7	5.703…e $^{-5}$ %
8	0 %
. . .	0 %

根據以上對 MIT Lincoln 實驗室所提供的十組模擬攻擊測試檔所作的測試結果，在改善上述二種情況所導致的問題之後，執行第二階段 SRAM 搜尋比對的機率為在 0.2658% 至 0.5023% 之間，也就是說平均大約執行 199 至 376 次 TCAM 比對，才需要執行一次第二階段 SRAM 搜尋比對；而因為抽取部份位元儲存而造成額外的第二階段 SRAM 搜尋比對的機率在 0.0152% 至 0.0248% 之間，也就是說平均約執行 4,032 至 6,578 次 TCAM 比對，才會造成一次額外的第二階段 SRAM 搜尋比對。所以 SRAM 搜尋比對器有足夠的時間去處理特徵字串佇列中的第一階段比對資訊，因此本架構整體的處理速度決定於 TCAM 比對速度，而 TCAM 執行比對為一固定常數時間，以 266MHz 的 TCAM 為例，執行一次比對的時間約為 4 ns，可計算出本架構的處理速度約為

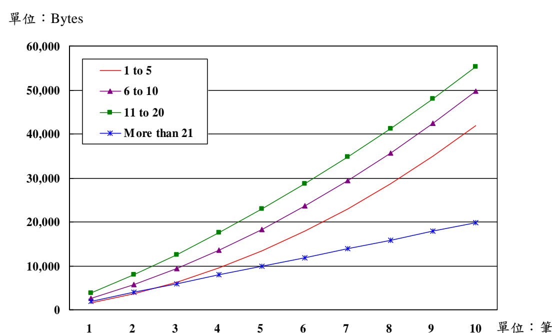
$(8 \times 4) \div 4^{-9} = 8$ Gbps。

4.2 TCAM 使用記憶體空間分析

首先對各組 TCAM 所使用的記憶體空間做分析，比較各組 TCAM 在儲存完整特徵字串與抽取特徵字串部分位元儲存此二種情況下，所使用記憶體空間情形。圖十一與圖十二為在上述二種情況下，各組 TCAM 在特徵字串平行儲存 M 次時，所使用的記憶體空間成長曲線圖。由於第一組 TCAM（1-5 Bytes）在二種情況下都是儲存完整的特徵字串，因此可以用第一組 TCAM 的成長曲線做為基準，比較二曲線圖之間的相互關係。



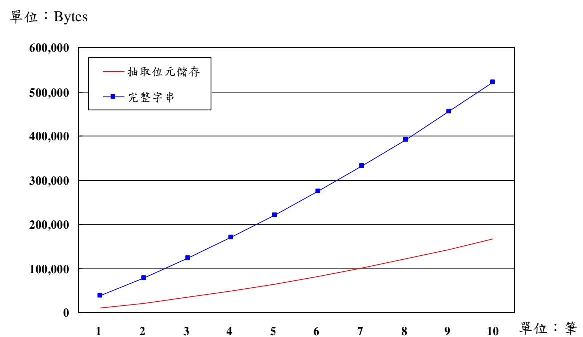
圖十一、儲存完整特徵字串的情況



圖十二、抽取特徵字串部分位元的情況

在圖十一及圖十二中可以明顯的觀察到，抽取特徵字串部分位元儲存可以有效減少長特徵字串所使用的 TCAM 記憶體空間。第二，三組 TCAM（6-10 Bytes 及 11-20 Bytes）使用空間的成長曲線明顯降低，更貼近第一組 TCAM 的成長曲線，第四組 TCAM（21 Bytes 以上）所使用的記憶體空間在平行儲存數筆後甚至可以降為四組 TCAM 中最低。因此，本論文所提出的架構，能夠有效的降低長特徵字串在 TCAM 中所佔用的記憶體空間，對於長特徵字串集合更能發揮出節省空間的效果。

接著，比較 TCAM 在儲存完整特徵字串與抽取特徵字串部分位元儲存此二種情況下，整體 TCAM 使用記憶體空間的情形。圖十三即為在此二種情況下整體 TCAM 使用記憶體空間的成長曲線圖。根據計算結果儲存特徵字串部份位元可以讓整體 TCAM 使用空間減少為原來的 26% 到 30%。



圖十三、整體 TCAM 使用的記憶體空間

5. 結論

在本論文所提出的二階段多重特徵字串比對處理架構中，第一階段使用 TCAM 快速的對輸入字串進行過濾，再藉由第二階段的 SRAM 搜尋比對，對 TCAM 比對結果進行再確認的動作。透過此二階段處理，可以確保不會有比對錯誤的情形發生。根據實驗結果，執行第二階段 SRAM 搜尋比對的機率在 0.5023% 以下。另外，由於此二階段動作可以同時執行，讓整體的處理速度決定於 TCAM 的比對速度，加上將特徵字串重複多筆平行儲存於 TCAM 中，減少整體比對次數。因此，對於長度 N 的輸入字串，以及在 TCAM 中每筆特徵字串平行儲存次數 M，時間複雜度為 $O(N/M)$。在 TCAM 使用記憶體空間方面，由於使用壓縮位元值來儲存特徵字串，可以讓整體 TCAM 使用記憶體空間減少為原來的 26% 到 30%，且對長特徵字串集合，更能發揮出節省空間的效果。預估在使用 266 MHz 的 TCAM，來處理 Snort 所擷取出的 2,082 筆特徵字串比對，且每筆特徵字串在 TCAM 中平行儲存四筆的條件下，使用 48,677 Bytes 的 TCAM 記憶體空間，可達到 8 Gbps 的處理速度。

參考文獻

[1] Snort system, http://www.snort.org

[2] R. S. Boyer and J. S. Moore, "A fast string searching algorithm", ***Communications of the ACM***, Vol.20, No.10 pp.762-772, Oct.1977.

[3] D. E. Knuth, J. Morris, and V. R. Pratt, "Fast Pattern Matching in Strings", ***SIAM Journal on Computing***, Vol. 6, No 2 pp.323-350, June 1977.

[4] Z. K. Baker and V. K. Prasanna, "Time and Area Efficient Pattern Matching on FPGAs", ***International Symposium on Field Programmable Gate Arrays*** (***FPGA***), pp. 223-232, Feb. 2004.

[5] A. Aho and M. Corasick, "Efficient string matching：An aid to Bibliographic search", ***Communications of the ACM***, Vol.18, No. 6 pp.333-343, June 1975.

[6] C. J. Coit, S. Staniford, and J. McAlerney, "Towards Faster String Matching for Intrusion Detection or Exceeding the Speed of Snort", ***DARPA Information Survivability Conference and Exposition*** (***DISCEX II***), June 2001.

[7] V. Paxon, "Bro：A System for Detecting Network Intruders in Real-Time", ***Computer Network***, 31(23-24), pp.2435-2463, Dec. 1999.

[8] Sun Wu and Udi Manber, "A fast algorithm for multi-pattern searching", ***Technical Report TR94-17, Department of Computer Science, University of Arizona***, May 1994.

[9] Burton Bloom, "Space/time trade-offs in hash coding with allowable errors", ***Communications of ACM***, pp.422-426, July 1970.

[10] S. Dharmapurikar, P. Krishnamurthy, T. Sproull and J. Lockwood, "Deep Packet Inspection using Parallel Bloom Filters", ***IEEE Micro***, pp.52-61, 2004.

[11] M. Gokhake, D. Dubois, A. Dubois, M. Boorman,. S. Poole, and V. Hogsett. "Granidt: Towards Gigabit Rate Network Intrusion Detection", ***International Conference on Field Programmable Logic and Applications*** (***FPL***), pp.404-413, 2002.

[12] E. Spitznagel, D. Taylor, and J. Turner, "Packet Classification Using Extended TCAMs", ***IEEE International Conference on Network Protocols*** (***ICNP***), 2003.

[13] P. Gupta and N. McKeown, "Algorithms for Packet Classification", ***IEEE Network***, pp.24-32, March 2001.

[14] Fang Yu, Randy H. Katz and T. V. Lakshman "Gigabit Rate Packet Pattern-Matching Using TCAM", ***IEEE International Conference on Network Protocols*** (***ICNP***), Oct. 2004.

[15] MIT Intrusion Detection Data Sets, http://www.ll.mit.edu/IST/ideval/index.html