實現具有速寫演算法來源識別碼回溯機制之 OpenFlow 網路流量變異偵測系統

黃柏勳 [1]　賴裕昆 [1,2]
中原大學通訊工程碩士學位學程 [1]
中原大學電機工程研究所 [2]
{ g10079031, ylai }@cycu.edu.tw

摘要

本研究使用 OpenFlow 架構，實現具有速寫演算法回溯機制之網路流量變異偵測系統於 NetFPGA-1G 開發平台。本系統於邊緣網路節點中，配置量測模組及 OpenFlow 交換器，量測模組藉由速寫演算法的硬體實現，快速摘要網路流量；透過 IPFIX 傳輸協定，將各個量測模組摘要結果，送至單一 OpenFlow 控制器進行分析及偵測，即時辨識並且回溯異常流量來源識別碼(Flow Key)、並通知交換器改變封包轉送的流向，適時對異常封包做相關處理，達到有效防止異常流量影響正常封包傳送的目標。本論文除了使用流量紀錄檔測試系統效能，並在機房佈建此系統，驗證真實網路環境系統之功能性。

關鍵詞：OpenFlow、NetFPGA、IPFIX、網路流量變異偵測。

1. 前言

近年來網路攻擊事件層出不窮，如端口掃描、蠕蟲、分散式阻斷攻擊等，此類攻擊造成短時間內網路流量的巨變，且攻擊者不一定為單一來源、其攻擊源頭通常來自於電腦，因此探討網路流量變異的議題，就必須收集多個觀測點的流量資訊，而大部分的防護，偵測設備或防禦設備擴充性不足，造成成本的提高，且防禦的範圍也僅限於此地區，各個區域所蒐集到的流量資訊無法互相交流或利用，造成在進行偵測或防禦機制上面的不足，因此最佳的解決方式，是以各觀測點蒐集流量資訊，並提供至集中式的分析中心或防禦系統內進行處理，當偵測出某個觀測點或區域遭受或進行攻擊行為，可從各地區蒐集的流量資料掌握整個攻擊的行為及方向，並即時的做適當的防禦動作[1]。

要準確量測高速網路流量並將收集到的流量整合，可仰賴速寫演算法來達成，速寫演算法可以使用少量的記憶空間，快速摘要大量資料，透過雜湊函數(Hash Function)使用，將資料儲存至固定的儲存空間。資料收集完成後，可以預估特定封包流的統計資訊，而其線性加總的特性，可將不同節點所得到的速寫演算法陣列整合，因此非常適合應用於高速網路流量量測系統。雖然此類演算法皆有快速資料摘要的特色，但演算法因雜湊函數(Hash Function)的使用無法回復異常流量來源識別碼(Flow Key)，需要有額外的儲存空間，記錄流量識別碼(Flow Key)，才能進行查詢統計結果，會需要增加額外的硬體資源[2] 。

Fast Sketch 速寫演算法[3]透過二維陣列的使用，具有優異的查詢速度及少量之記憶空間之特性，可將流量摘要及流量識別碼資訊儲存於此陣列，透過回溯運算即可查詢異常流量識別碼。因此，本論文使用 OpenFlow 架構實現網路流量變異偵測系統，將 Fast Sketch 演算法之硬體以 Verilog HDL 實現於 NetFPGA-1G 開發平台，透過硬體加速的特性，將網路流量資訊摘要至精簡的速寫演算法陣列，此系統不但具有網路分流器(Network Traffic Tap)之功能，更可以接著透過 IPFIX 傳輸協定，將各量測節點所量測到的流量資訊，傳送至由 OpenFlow 控制器，進行網路流量變異偵測，藉由流量變異偵測結果，回溯異常流量來源識別碼，透過 OpenFlow 控制器下達異常流量處理規則，將異常來源進行即時流量管制。

2. 背景與相關研究

2.1 NetFPGA

2007 年，美國史丹佛大學(Stanford University)研究團隊開發出以可程式邏輯閘陣列(Field Programmable Gate Array，FPGA)為硬體為系統核心的網路開發平台-NetFPGA-1G[4]，其建構於 Linux 作業系統環境下，設計者透過 Verilog 硬體描述語言的程式設計，可在 NetFPGA 開發平台上實現或實驗出各式網路架構，如防火牆、編碼加密演算法、網路路由器或交換器等系統。2010 年，設計出第二代的 NetFPGA-10G，不同於 NetFPGA-1G，除了提高硬體效能外，透過模組化程式設計，讓使用者能更容易開發出不同的網路架構。

2.2 OpenFlow

2008 年史丹佛大學基於軟體定義網路的概念，

提出 OpenFlow[5]架構，有別於傳統交換器或路由器，OpenFlow 交換器或路由器可由開發人員所設計的控制端來決定交換(Switching)以及路由(Routing)的方式，劃分成控制器(OpenFlow Controller)及交換器(OpenFlow switch)，控制器定義封包傳送規則，並透過安全通道(Secure Channel)將傳送規則送至交換器；交換器內具有一個封包流路由表(Flow Table)接收傳送規則，當封包進入交換器後，便可根據表格內所定義的規則進行轉送或傳送。

2.3 Fast Sketch 演算法

目前速寫演算法分類中，包含量測不同總類封包流總數(Distinct Flow)及封包流個數或流量大小(Packet/Size Per Flow)兩類，其中以計算封包流個數或流量大小為例，常見演算法包含 CM-Sketch (Count Min Sketch)[6]、PMC-Sketch(Probabilistic Multiplicity Counting)[7]，雖然此類演算法皆有快速資料摘要的特色，但皆無法回復異常來源的功能，需要有額外的儲存空間，記錄流量識別碼(Flow Key)，才能進行查詢統計結果。

Fast Sketch 演算法由 Yang Liu 等人所提出[3]，其演算法可計算個別封包流的流量大小，其最大的特色在於透過速寫演算法陣列，可回復原來的流量識別碼(Flow key)，不須使用額外儲存資源，其演算法可分為封包更新(Update)以及查詢(Query)兩部分。Fast Sketch 作者在論文中比較與現存演算法其空間與時間比如下**表 1**，由表格可得知 Fast Sketch 可使用少量空間達到快速更新及查詢的效果。

表 1 Fast Sketch 空間效能比較[3]

	Space	Update Time	Query Time
Modular Hashing	$O(n^{\frac{1}{\log\log n}}\log\log n)$	$O(\frac{\log n}{\log\log n})$	$O(kn^{\frac{1}{\log\log n}}\log\log n)$
Combinatorial Group Testing	O(k log n)	O(log *n*)	O(k log n)
Random Projection	O(k)	O(1)	O(n)
Chinese Reminder Theory	$O(n^{1/2})$	O(1)	$O(k^2)$
Fast Sketch	$O(k\log\frac{n}{k})$	$O(\log\frac{n}{k})$	$O(k\log\frac{n}{k})$

2.3.1 封包更新

Fast Sketch 使用一個二維的計數器陣列，其個數為 l*(1+log(n/l))，當封包進行摘要時，使用 N 個雜湊函數計算，映射到相對應的計器陣列，將此封包的長度(Length)累加至此計數器陣列當中，其中 n 為封包流的總類(例如以來源端 IP 為封包流分類：n = 2^32)，如**圖 1** 所示。

$$h(f) = (f \bmod l) \oplus h'(\lfloor f/l \rfloor) \quad (1)$$

$$h'_j(x) = ((\alpha_j x + \beta_j) \bmod P) \bmod l \quad (2)$$

$$q(f) = \lfloor f/l \rfloor \quad (3)$$

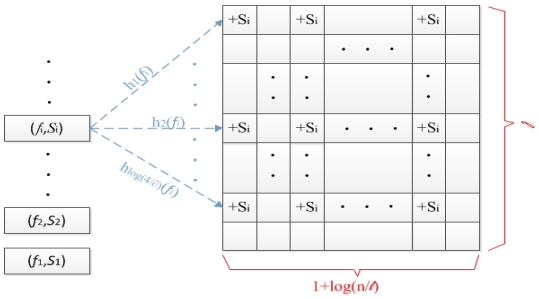

圖 1 Fast Sketch 封包更新

公式 (1)為雜湊函數表示式，而雜湊函數個數大小會影響查詢結果準確度，雜湊函數個數越多，代表準確度越高，因此可表示為$N = \log(4/\delta)$，其中 δ 為錯誤率。

公式(2)中，$h'_j(\bullet)$為 2 階 Universal 雜湊函數(2-Universal Hash)、P為大於 l 的質數，而α_j及β_j為介於{1,…P-1}的隨機數。

公式(3)中，$q(\bullet)$為商值(Quotient Function)，取決於f除l的商數，f代表封包流，以本系統為例，封包流以來源端 IP 位址分類，可表示為$f = \{\text{SrcIP}\}$。

舉例來說，當$f = 55, l = 5$，便可計算出商值$q(f) = 11$，其二進位表示為 b'1011，因此被填入的位置如圖 2 所示。

圖 2 Fast Sketch 位元示意圖

2.3.2 封包查詢

Fast Sketch查詢

1:for$a = 0, \ldots, l-1$ do
2: if ($\left|\triangle C^{all}_{a,0}(t)\right| > \frac{\theta}{2} \triangle \hat{V}(t)$)
3: $x \leftarrow 0,\ r \leftarrow 1$
4: for $b = 1, \ldots, log_2(n/l)$ do
5: if ($\left|\triangle C^{all}_{a,0}(t)\right| > \frac{\theta}{2} \triangle \hat{V}(t)$)then
6: $x \leftarrow x + r$
7: $r \leftarrow r \times 2$
8: end for
9: for $j = 1, \ldots, log(4/\delta)$ do
10: $f_j = \varphi(x, a)$
11: if $(median_P \triangle C^{all}_{p(f_j),0}(t) > \frac{\theta}{2} \triangle \hat{V}(t))$ then
12: Report f_j as anomalous;
13: end for
14:end for

演算法 1 Fast Sketch 查詢虛擬碼[3].

當封包更新完成，開始對第一行(First Column)的每一個計數器(Counter)進行比對，比較兩個區間的計數器變化量($\Delta S_o = S_o(t) - S_o(t-1)$)是否大於上一個區間所計算之門檻值，如果小於門檻值，

則往下一列繼續搜尋；反之，則進入此列對每一個計數器進行比對，比較計數器是否大於門檻值，並設定x初始值為 0，r初始值為 1，其中x值代表商值(Quotient Function)，r 值代表位元數(Bit Number)。

由 Fast Sketch 所使用的雜湊函數公式，可以衍生出反雜湊函數(Inverse Universal Hash)，反雜湊函數可以透過商值x及列值(Row)y的給定，可回復原來的流量識別碼f。

$$\varphi(x,y) = xl + y \oplus h'_j(x)$$
$$= \lfloor f/l \rfloor \bullet l + (f \bmod l) \oplus h'_j(\lfloor f/l \rfloor) \oplus h'_j(\lfloor f/l \rfloor)$$
$$= \lfloor f/l \rfloor \bullet l + (f \bmod l)$$
$$= f \quad (4)$$

當計數器超過門檻值，則將 x 做 r 的累加，藉此便能回復可疑流量的商值 x。由於不知道是哪一個雜湊函數(Hash Function)映射到此列，因此必須將商值 x 帶入每一個反雜湊函數(Inverse Universal Hash)回復可疑流量識別碼(Flow Key) f_j。

$$f_j = \varphi(x,y)\ for\ j = 1, \ldots, log\ (4/\delta) \quad (5)$$

為了減少誤判率(False Positive)的發生，使用 CM-Sketch 的方式，將每一個回復後的流量識別碼 f_j，再帶入雜湊函數的運算，比對映射到的位置的計數器平均值是否超過門檻值，如果超過則回報此流量識別碼f_j為異常來源。

$$\left|\text{median}_p \Delta C^{all}_{p(f_j),0}\ (t)\right| > \frac{\theta}{2}\Delta\hat{V}(t) \quad (6)$$

$$\Delta V(t) = \sum_f |v_f(t) - v_f(t-1)| \quad (7)$$

3. 系統實現

本論文應用 OpenFlow 架構及 NetFPGA-1G 開發平台，實現基於 OpenFlow 架構的網路異常流量偵測系統，此架構分為量測模組、傳輸模組、異常偵測模組、以及異常處理模組，如下**圖 3** 所示。

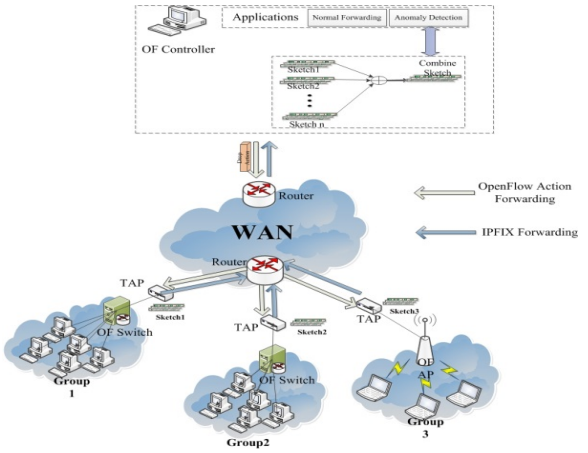

圖 3 網路異常流量偵測系統架構圖

量測模組：使用 Fast Sketch 演算法，實現於 NetFPGA-1G 開發平台架構中的網路分流器(Network Traffic Tap)，將流經量測模組的封包進行流量摘要。

傳輸模組：使用 IPFIX 協定傳輸，分為輸出模組(Exporter)及收集模組(Collector)，輸出模組實現於網路分流器中；收集模組實現於監控中心(OpenFlow 控制器)，將流量資料由輸出模組送至收集模組。

異常偵測模組：使用 Fast Sketch 演算法進行異常流量偵測，實現於監控中心(OpenFlow 控制器)，將摘要後的流量資料分析，找出異常流量來源。

異常處理模組：使用 NOX Application，實現防禦模組(Defense)於監控中心(OpenFlow 控制器)，將偵測模組提供的異常流量來源建立異常流量傳送規則。

當封包進入量測單元後，會使用速寫演算法紀錄並累加此封包長度，儲存至速寫演算法陣列，每隔一段觀測時間結束，所有量測單元會將摘要後的速寫演算法陣列，透過傳輸模組送往集中的監控中心(即 OpenFlow 控制器)，控制器便利用速寫演算法具有的線性加總特性，將接收後的速寫演算法陣列整合，並根據異常偵測模組的實現，判定是否有異常流量的出現，如果有，便透過監控中心所記錄的異常來源端 IP 位址，將此封包流增加異常流量處理規則，送至所有連接的 OpenFlow 交換器，防止異常流量影響正常使用者可使用的網路頻寬。

3.1 量測模組實現

本論文基於 NetFPGA-1G 開發平台下，實現以硬體為主的網路資料分流器(Network Traffic Tap)，在每一個不同的觀測區間內，量測流經觀測點中個別封包流的總量(Size of Flow)。

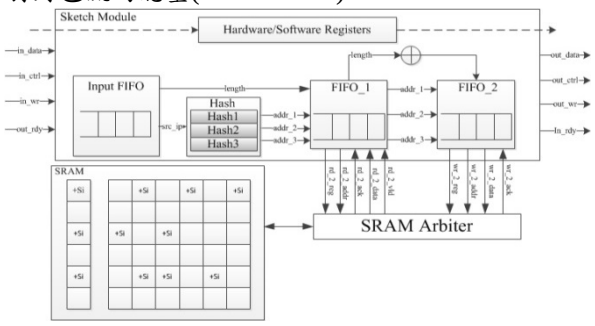

圖 4 速寫演算法硬體架構圖

本論文以 NetFPGA-1G 的網路卡參考設計模組(Reference NIC)為基礎，加入基於 Fast Sketch 封包更新的量測階層(Sketch Module)，包含雜湊函數實現及計數器陣列實現，其模組架構如**圖 4** 所示。由於 Fast Sketch 使用 2 階 Universal 雜湊函數(2-Universal Hash)，需要做餘數(mod)運算，以硬體設計角度而言，會消耗龐大的資源及運算時間，因此本論文使用 CW-trick[8]方式，實現平行化的雜湊函數運算，透過乘法器及位移的方式即可計算出映射值(Value)；而計數器陣列以 NetFPGA 開發平台中

的 SRAM 來實現，系統實現兩組相同大小的計數器陣列，藉由軟體暫存器(Software Register)的控制訊號(Update)做區間切換，如此一來便能即時更新封包流量，並將前一個區間統計後結果送至傳輸模組。

3.2 傳輸模組實現

透過傳輸模組的實現，可將分散各區域的量測單元所蒐集到的速寫演算法陣列，傳送到單一的偵測單元進行異常偵測，本系統將速寫演算法陣列資料結構透過 IPFIX (IP Flow Information eXport)[8][10] [11]的傳輸協定從量測單元輸出器(Exporter)送往位於偵測系統 OpenFlow 控制器的收集端(Collector)。

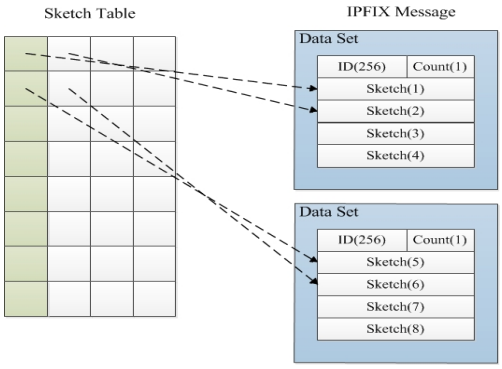

圖 5 計數器陣列轉換成 IPFIX 資料

每隔一個時間區間，輸出器會藉由 NetFPGA 軟體所提供的*Ioctl( )*函式將 SRAM 所暫存的計數器陣列讀出，並將計數器包裹成 IPFIX 的資料模板，進行傳輸，如**圖 5**；收集端開啟監聽(Listen)埠口後，進入輪詢(Poll)機制中，藉由接收輸出器傳送的封包，解析其傳送之 IPFIX 訊息，轉換成軟體的計數器陣列。

3.3 異常流量偵測模組實現

當 IPFIX 收集器(Collector)在第 N 個區間結束後，將所有速寫演算法陣列從輸出器(Exporter)收集完成，完成線性加總動作，得到此觀測區間的速寫演算法陣列S_o。有了這些帶有流量資訊的速寫演算法陣列，便可透過異常偵測模組，找尋可疑的異常來源，此部分系統使用 Fast Sketch[3]的流量變異偵測演算法結合移動平均(Moving Average)的概念實現。

得到線性加總後的觀測區間速寫演算法陣列S_o後，開始對第一行(First Column)的每一個計數器 "$S_o[a][0]$" 進行比對，比較計數器變化量S_e 是否大於由上一個區間所計算的門檻值，如果小於門檻值，則往下一列繼續搜尋；反之，大於門檻值，則進入此列對每一個計數器 "$S_o[a][0]$" 進行比對，比較計數器是否大於門檻值。由於網路流量的起伏是非常隨機且凌亂的，使用兩個區間所觀測到的速寫演算法陣列，以總流量相減$S_o(t) - S_o(t-1)$來計算門檻值，可能造成誤判率(False Positive)的提高，因此在於門檻值的處理，系統使用 K-ary Sketch [12]所提出的計算方式。異常偵測模組會設定一個預測陣列(Forecasting Array) S_f及視窗大小W(Window Size)，其功能為預測下一個區間特定封包流流量，陣列大小為$(l-1)$。

3.4 異常處理模組實現

當異常流量偵測模組偵測並列出此區間的異常流量識別碼(即為來源端 IP 位址)後，會將結果送至由 OpenFlow 控制器(OpenFlow Controller)執行的應用程式中，對此類封包流(Flow)建立丟棄規則，並將此規則送往各個 OpenFlow 交換器(OpenFlow Switch)，增加封包流路由表(Flow Table)的路由規則，因此當異常來源端 IP 位址再度進行惡意攻擊行為時，發送封包進入到 OpenFlow 交換器時，會直接被 OpenFlow 交換器丟棄，使其無法繼續影響正常封包的傳送，完成流量阻擋的動作，其示意圖如下。

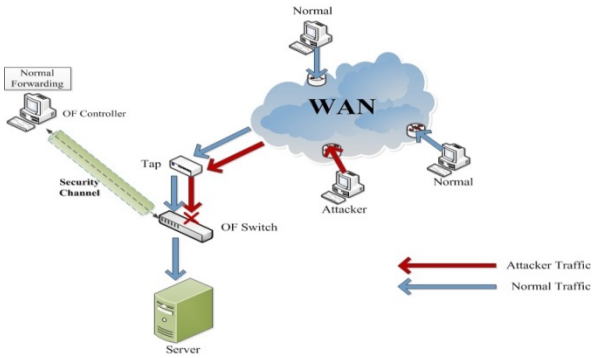

圖 6 異常流量處理概念圖

本論文使用 NOX 核心應用程式提供的 testhub 模組進行修改，當封包進入函式(*packet_in_callback*)時，開啟由異常流量偵測模組所記錄的異常流量識別碼檔案，查詢進入的封包是否被記錄為異常，如果是，則會建立一個規則，其封包流優先權(priority)設為最高，並將封包流動作(Action)設為丟棄，將此規則送往各個 OpenFlow 交換器中，當攻擊者再次傳送此類封包流，便會在 OpenFlow 交換器中對應到丟棄的傳送規則，達到阻擋此類惡意攻擊行為流量的效果。

4. 系統效能及探討

此節針對本系統實現之效能及功能進行分析及討論，首先探討 Fast Sketch 演算法以及精確計數(Real Count)的準確度，接著探討實現 Fast Sketch 的量測模組於硬體上，影響硬體效能的比率。最後以實機測試，將網路異常流量偵測系統佈建於不同的網路環境，探討是否能找出流量異常，並查詢異常流量的來源端 IP 位址，最後對此類流量異常進行阻

擋，佐證系統的功能性。

4.1 Fast Sketch 準確度測試

使用 Fast Sketch 演算法對流量做摘要會減少系統判定異常流量的準確度。此測試以軟體實現 Fast Sketch 演算法，並使用 MAWI[13]提供的流量記錄檔(200302270000.dump)，此流量檔擷取至跨太平洋 100Mbps 的傳輸通道(Trans-Pacific Line)，共有 51,778 個不同來源端 IP 位址所組成，總長度為 15 分鐘；軟體擷取流量檔中每個封包的時間戳記(Timestamp)，每個觀測時間區間 60 秒、速寫演算法陣列l=65535，雜湊函數個數=3，進行分析結果如下。門檻值不同，其誤判率(False Positive)與精確計數方式也會有所差異。

表 2 Fast Sketch 準確度測試

Threshold	*0.1*	*0.05*	*0.01*	*0.008*	*0.005*	*0.001*
False Positive	0.85	1.71	1.99	2.0	1.95	2.03

雜湊函數個數與誤判率有直接相關[3]。因此原作者提出使用多個雜湊函數及 CM-Sketch 來降低誤判之情況。但由本系統測試發現，使用越多雜湊函數情況下，是可以大幅降低誤判率，但卻造成漏判率的大幅提升。

誤判率造成原因是因為雜湊函數發生碰撞，使 Fast Sketch 陣列回溯錯誤，回溯出不存在的流量識別碼，以本測試為例，使用雜湊函數個數為 3 時，有高達 75%的流量識別碼不存在於原本的流量之中。

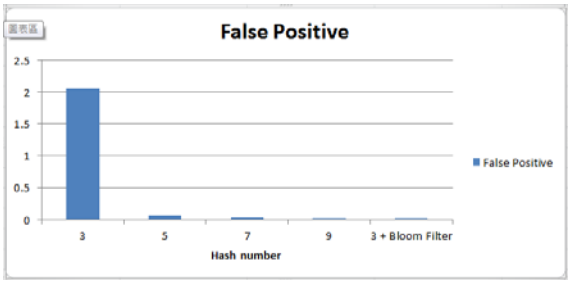

圖 7 準確率比較圖

為了降低誤判率，本系統加入布隆濾波器(Bloom Filter)[19]，當封包進入系統時，流量識別碼會被記錄至布隆濾波器之中，於 Fast Sketch 回溯流量識別碼時，便可再次查詢此流量識別碼是否存在於布隆濾波器之中，藉此便可過濾不存在的流量識別碼，大幅降低誤判率。本系統加入一個 16KB 大小的布隆濾波器(k=3, m=131072 bits)，設定門檻值參數為 0.01，於軟體模擬測試結果由圖 7 中可看到，當同樣使用 3 個雜湊函數加入布隆濾波器，可等同使用 9 個雜湊函數大幅降低誤判率的結果。

因為 Fast Sketch 所使用的計數器陣列實現於靜態記憶體 SRAM 之中，每個封包更新皆須對記憶體進行讀、加、寫之動作，使用越多個雜湊函數，需要更多更新次數，會增加系統處理時間。

使用 CM-Sketch 或布隆濾波器皆可改善誤判率，但以相同誤判率而言，如**表 3** 所示，使用布隆濾波器所需消耗之記憶空間較小，更新後也可送至控制中心進行線性加總並查詢，非常適合於本論文實現的硬體模組中。

表 3 相同誤判率之下使用記憶空間的比較

	0.01	*0.1*	*0.2*
CM-sketch	12KBytes	6.2KBytes	4.3KBytes
Bloom Filter	0.6KBytes	0.3KBytes	0.2KBytes

4.2 量測模組功能測試

在量測模組中，計數器陣列數值更新的讀寫時間長短會影響系統的效能，透過 Modelsim 模擬 Fast Sketch 模組進行封包更新，對靜態記憶體進行讀、加、寫的波型，如圖 8 所示。圖 8 中紅色區間為本系統以 2-Universal 雜湊函數進行摘要處理所得到的記憶體地址，黃色區間為本系統根據所摘要得到的地址進行計數器數值的讀取與更新，藍色區間為將更新的值寫回記憶體中。

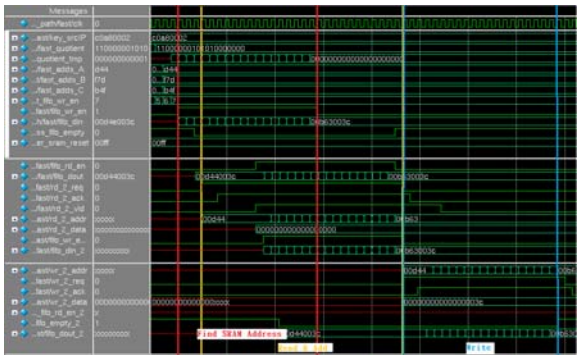

圖 8 Fast Sketch 模組進行封包更新之靜態記憶體讀、加、寫之動作波型。

4.3 異常流量偵測系統功能測試

此測試利用流量檔播送(Tcpreplay)的方式，將 Slammer 蠕蟲攻擊[16]播送至本系統所防護網域之主機，測試系統是否能正確偵測異常流量來源，並有效防禦其攻擊，測試中也利用流量錄製(Tcpdump)，可記錄防護前及防護後的網路流量變化。

此測試使用五部主機，分別實現下列功能：OpenFlow 控制器：運行 NOX Application 中的 Python Switch 模組(Pyswitch)及本系統實現的防禦模組(Defense)，以及 IPFIX 收集器(IPFIX Collector)模組；OpenFlow 交換器：搭配 NetFPGA 開發平台，運行 OpenFlow1.0.0 模組；網路分流器：搭配 NetFPGA 開發平台，運行 IPFIX 輸出器(IPFIX Exporter)模組；測試主機 1(Tcpreplay)：運行 Tcpreplay[17]進行流量記錄檔播送，作為外部攻擊端；測試主機 2(Victim)：運行 TCPdump[18]，作為內部受害者及流量記錄。此測試發送流量為 Slammer 蠕蟲案例[16]，案例發生於 2003 年，其特徵由來源端透過 UDP Port 1434 的漏洞，尋找網路中擁有此漏洞的主機，因此 Slammer 流量記錄檔僅擷取 UDP Port 1434 的封包，總長度為八小時。而

本論文擷取其中發生劇變的區段，總長度為 15 分鐘來進行測試，測試環境架構如圖 9 所示。

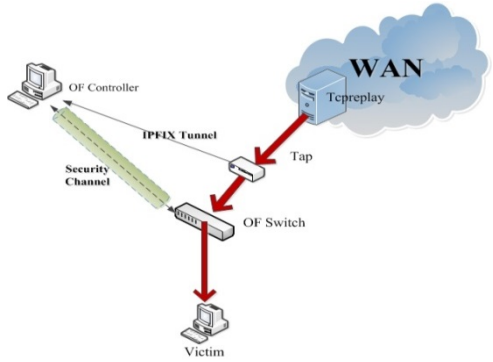

圖 9 防護測試環境示意圖

圖 10 及圖 11 分別為加入阻擋機制前/後，其播送後的流量記錄檔波形圖，由圖中可明顯看出，在異常來源開始進行攻擊時，會發送大量異常封包，造成網路頻寬的消耗，當加入本系統的防護機制後，可在觀測時間結束後，即時阻擋異常來源封包的傳送，回復原先正常傳送的網路環境。

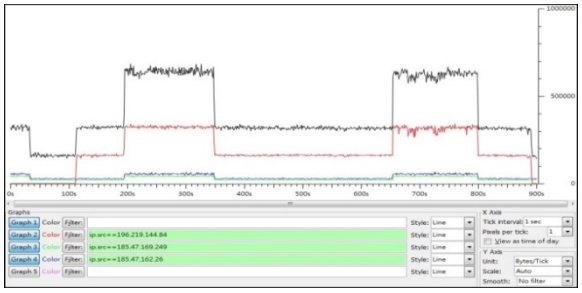

圖 10 防護前流量記錄檔波形圖

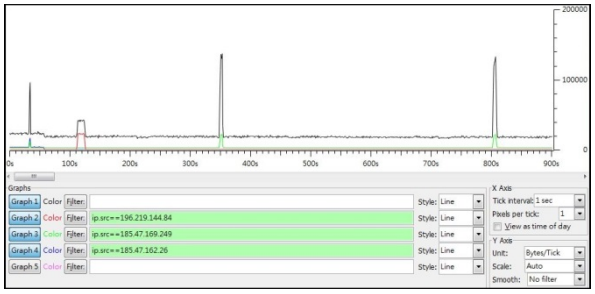

圖 11 防護後流量記錄檔波形圖

5. 結論

本論文實現具有速寫演算法回溯機制之網路流量變異偵測系統，結合 OpenFlow 架構，將偵測出異常流量的來源即時加以阻擋，以防止其繼續影響正常使用者的網路使用。

本論文實現的系統包含以下幾個特色：

- **以硬體實現 Fast Sketch 流量量測**：藉由觀測點的流量收集，送至分析中心回溯異常流量來源識別碼。
- **以軟體實現網路中心偵測**：將各個量測端的流量資料整合，並查詢異常的流量來源。加入布隆濾波器(Bloom Filter) ，於 Fast Sketch 回溯流量識別碼時，便可過濾不存在的流量識別碼，大幅降低誤判率。
- **即時查詢異常來源並加以阻擋**：將偵測出的異常來源在其發送的源頭網路設備加以阻擋，使其無法繼續影響正常使用者使用。

未來展望

目前系統實現以 Fast Sketch 演算法進行摘要，僅根據 SIP 分類，來偵測異常流量變化及回復異常來源，未來期許能進一步擴充系統效能及處理速度(10Gbps)，以更多封包標的(如 DIP、SPort、DPort)來實現量測系統。

本文承蒙行政院國科會個別型專題研究計畫補助（國科會編號：NSC 101-2221-E-033-072, 102-2221-E-033-031），特此致謝。

參考文獻

[1] 梁明章, 林孟璋, 謝欣叡, 陳俊傑, “TWAREN 骨幹防禦雲 Flow 路徑回溯分析與防禦機制之設計”, 2012 台灣網際網路研討會, Taiwan Academic Network, TANET2012, Oct.23~25, 2012

[2] 黃柏勳, 賴裕昆, “應用 OpenFlow 於網路流量變異偵測之系統實現”, 2012 台灣網際網路研討會, Taiwan Academic Network, TANET2012, Oct.23~25, 2012

[3] Liu. Yang, Chen. Wenji, Guan, Yong, “A fast sketch for aggregate queries over high-speed network traffic”, in INFOCOM, 2012 Proceedings IEEE, pp. 2741–2745.

[4] NetFPGA. http://netfpga.org/.

[5] OpenFlow Switch Consortium et al. OpenFlow Switch Specification Version 1.0.0. 2009.

[6] Cormode. Graham, Muthukrishnan. S, “An improved data stream summary: the count-min sketch and its applications”, Journal of Algorithms 55, 1 (2005), pp. 58–75.

[7] Lieven. Peter, Scheuermann, Björn, “High-speed per-flow traffic measurement with probabilistic multiplicity counting”, in INFOCOM, 2010 Proceedings IEEE (, 2010), pp. 1–9.

[8] Mikkel Thorup and Yin Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In Proceddings of the fifteenth annual ACM-SIAM synposium on Discrete algorithms, page 615 624, 2004.

[9] Claise. Benoit, “Specification of the IP flow information export (IPFIX) protocol for the exchange of IP traffic flow information”, (2008).

[10] Trammell. Brian, Boschi. Elisa, “An introduction to IP flow information export (IPFIX)”, Communications Magazine, IEEE 49, 4 (2011), pp. 89–95.

[11] libIPFIX. http://libipfix.sourceforge.net/

[12] Krishnamurthy. Balachander, Sen. Subhabrata, Zhang. Yin, Chen. Yan, “Sketch-based change detection: methods, evaluation, and applications”, in Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement (2003), pp. 234–247.

[13] "MAWI Working Group Traffic Archive". http://mawi.wide.ad.jp/mawi/

[14] Ixia. http://www.ixiacom.com/.

[15] Cisco SR2016 16-port 10/100/1000 Gigabit Switch – cisco systems. http://www.cisco.com/en/US/products/ps10013/index.html

[16] MS-SQL slammer traffic analysis. http://rbeverly.net/research/slammer/

[17] Tcpreplay. http://tcpreplay.synfin.net/

[18] Tcpdump. http://www.tcpdump.org/

[19] Bloom, Burton H. (1970), "Space/Time Trade-offs in Hash Coding with Allowable Errors", Communications of the ACM 13 (7): 422–426,doi:10.1145/362686.362692