SO TRÙNG MẪU DỰA TRÊN CUCKOO HASHING ỨNG DỤNG CHO NIDS

Trần Ngọc Thịnh

Trường Đại học Bách khoa, ĐHQG-HCM

(Bài nhận ngày 07 tháng 12 năm 2010, hoàn chỉnh sửa chữa ngày 20 tháng 04năm 2011)

TÓM TẮT: *Bài báo này mô tả một máy so trùng tên Cuckoo-based Pattern Matching (CPM) dựa trên một giải thuật hashing đã được phát triển gần đây gọi là Cuckoo Hashing. Chúng tôi hiện thực giải thuật này với những cải tiến song song hóa phù hợp cho việc so trùng nhiều mẫu có chiều dài khác nhau. CPM có khả năng cập nhật dữ liệu dễ dàng, nhanh chóng đồng thời chiếm ít tài nguyên phần cứng. Với khả năng xử lý song song lớn, speedup của CPM lên tới 128 lần khi so sánh với hiện thực Cuckoo nối tiếp. Khi so sánh với các hiện thực so trùng mẫu bằng phần cứng trước đây, CPM có hiệu suất vượt trội với việc tiêu hao phần cứng ít hơn 30%.*

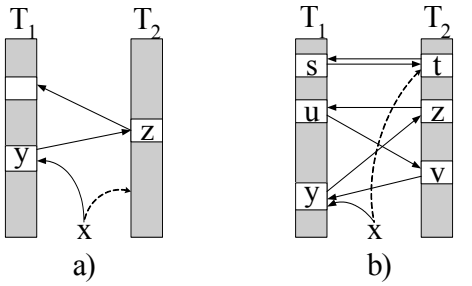
Từ khóa*: so trùng mẫu, Cuckoo hashing, FPGA.*

1. GIỚI THIỆU

Hiện nay, các hình thức xâm nhập bất hợp pháp là một trong những mối đe dọa lớn nhất của bảo mật mạng. Sự xâm nhập ngày càng gia tăng bởi sự có sẵn rất nhiều các công cụ phá hoại (hacking) với độ phức tạp, tinh vi gia tăng theo thời gian, dễ dàng làm suy yếu các công cụ bảo mật cơ bản. Thông thường, hệ thống mạng được bảo vệ bởi tường lửa, cung cấp các chức năng cơ bản của của việc giám sát và lọc ở mức phần đầu gói (packet header). Tuy nhiên, không phải tất cả xâm nhập hiểm độc đều bị ngăn chặn. Điều này đòi hỏi những giải pháp tốt hơn cho bảo mật mạng. Vì vậy, hệ thống phát hiện/ngăn chặn xâm nhập mạng (Network Intrusion Detection/Prevention Systems – NIDSs/NIPSs) [1] đi xa thêm một bước nữa là lọc ở mức sâu hơn gói dữ liệu mạng để tìm kiếm các nghi ngờ bên trong. NIDS khác tường lửa ở chỗ nó cho phép kiểm tra nội dung của gói dữ liệu (packet payload). Một cách tổng quát, phần lớn NIDS tìm kiếm dựa trên một tập quy luật (rules) có sẵn. Những qui luật này thường bao gồm thông tin về TCP/IP header và thường có những mẫu (patterns) của các loại hình tấn công như worm, Trojan, v.v… Hiện tại, phần lớn NIDS chạy dưới dạng phần mềm và chúng chỉ có thể đón bắt và xử lý gói dữ liệu ở tốc độ tối đa là vài trăm Mbps. Ví dụ, Snort [1] là một NIDS mã nguồn mở sử dụng một thư viện mở. Để phát hiện xâm nhập, Snort dựa vào một tập cơ sở dữ liệu chứa hàng ngàn quy luật, mỗi cái chứa các mẫu chữ ký tấn công.

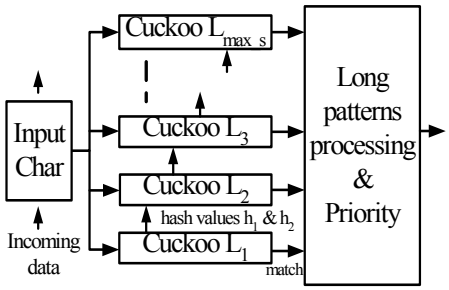
Đối với việc bảo vệ theo thời gian thực, NIDS phải chạy ở tốc độ đường dây mạng có thể lên tới hàng chục Gbps. Tuy nhiên, điều này rất khó đạt được trên các hệ thống general processor bởi vì hiệu suất của nó phụ thuộc vào việc so trùng với hàng ngàn mẫu ở tốc độ đường dây.

Để cải tiến hiệu suất của NIDS, người ta thường có xu hướng hiện thực các giải thuật so trùng mẫu trên phần cứng chẳng hạn như Application Specific Integrated Circuit (ASIC) hoặc Field Programmable Gate Arrays (FPGA). ASIC có thể đạt được các hiệu suất tính toán tối ưu nhưng khó thay đổi thiết kế theo thời gian. Ngược lại, các hệ thống tính toán trên phần cứng tái cấu hình (FPGA) có khả năng cung cấp hiệu suất của phần cứng và độ uyển chuyển để tái cấu hình lại phù hợp với chức năng cập nhật dữ liệu thường xuyên của các hệ thống NIDS.



Hình 1. Hàm băm Cuckoo Hashing [9], a) Khóa x được thêm vào thành công bằng cách di chuyển các khóa y và z, b) Khóa x không thể tìm được chỗ trống và quá trình băm lại (rehash) được yêu cầu.

Bài báo này trình bày một kiến trúc cho so trùng các mẫu có chiều dài khác nhau tên là CPM. Kiến trúc này ứng dụng một giải thuật đã được phát triển gần đây tên là Cuckoo Hashing [15]. Các mẫu có thể dễ dàng thêm vào hoặc xóa đi khỏi các bảng hash vì vậy CPM không giống như các hệ thống dựa trên FPGA trước đây, nó có thể cập nhật các mẫu cực kỳ nhanh chóng mà không cần tái cấu hình lại. CPM cũng đạt một hiệu suất tốt hơn hẳn khi so sánh với các hệ thống khác.



Hình 2. FPGA-based Cuckoo Hashing cho so trùng mẫu trong NIDS.

2. CÁC CÔNG TRÌNH NGHIÊN CỨU LIÊN QUAN

Để xử lý với tốc độ gigabit của network, nhiều công trình nghiên cứu xử lý mẫu dùng FPGA đã được đề nghị. Chúng có thể được chia làm các hướng tiếp cận chính sau đây.

Dịch và so sánh (shift-and-compare): [2-3] là phương pháp tiếp cận dễ dàng nhất và có tốc độ cao nhất. Các tác giả áp dụng các bộ so sánh song song và pipeline sâu trên những vị trí phân đoạn khác nhau của dữ liệu vào.Tuy nhiên hạn chế của phương pháp này là tiêu hao quá nhiều tài nguyên phần cứng.

Máy trạng thái (State machine:NFA/DFA): [4, 6] chuyển đổi các mẫu thành các biểu thức chính qui có thể hiện thực chạy song song trên máy trạng thái. Với lợi điểm có thể xử lý các mẫu có chiều dài rất lớn một cách dễ dàng và khá uyển chuyển, phương pháp này có dùng các khối block RAM để tiết kiệm tài nguyên logic cell của FPGA. Hạn chế là tốc độ xử lý thấp hơn các phương pháp khác.

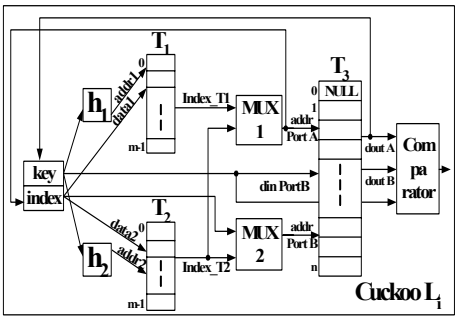
Băm (Hashing): [10-12] phương pháp tiếp cận này đã được sử dụng khá phổ biến trong các ứng dụng bảo mật khác, tuy nhiên trong hệ

thống NIDS, nó chỉ vừa được ứng dụng gần đây, và kết quả đạt được khá tốt. So với các phương pháp trên thì khả năng tiết kiệm phần cứng là vượt trội, với hiệu suất chỉ thua phương pháp shift-and-compare và tốt hơn phương pháp máy trạng thái.

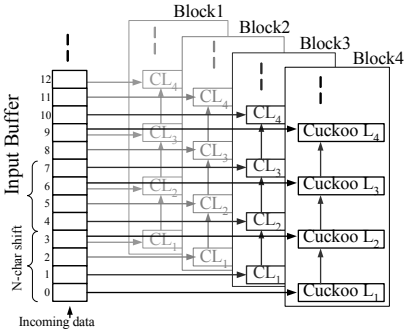
Hàm băm Cuckoo (Cuckoo Hashing) được đề nghị bởi Pagh và Rodler [15]. Giải thuật dùng 2 bảng T_1 và T_2 với kích thước $m = (1+\varepsilon)n$ cho mỗi bảng ($\varepsilon>0$) với n là số phần tử (khóa). CH dùng 2 hàm băm thông thường h_1 và h_2 để tính giá trị băm cho một khóa x ở địa chỉ $T_1[h_1(x)]$ hoặc $T_2[h_2(x)]$. Việc tìm kiếm hoặc xóa phần tử là hằng số, với thời gian tìm kiếm nhiều nhất là 2 lần truy xuất bộ nhớ độc lập.

Sự mới lạ của giải thuật nằm ở chỗ thủ tục thêm vào phần tử x. Nếu ô nhớ $T_1[h_1(x)]$ trống, thì x được đặt vào và quá trình thêm vào kết thúc. Nếu ngược lại, $T_1[h_1(x)]$ bị chiếm bởi một khóa y nào đó, $h_1(x) = h_1(y)$, thì x vẫn được đặt vào ô $T_1[h_1(x)]$ và y bị lấy ra khỏi ô. Tiếp theo, y được đặt vào ô $T_2[h_2(y)]$ của bảng thứ 2 với cách thức tương tự, nó có thể làm một khóa z khác với $h_2(y) = h_2(z)$ bị mất chỗ. Đến lượt z sẽ đặt ở ô $T_1[h_1(z)]$, và quá trình tiếp tục cho đến khi nào khóa hiện tại bị mất chỗ tìm được chỗ trống như hình 1.a. Quá trình này gọi là "Cuckoo process". Tuy nhiên, chúng ta có thể thấy trong hình 1.b, "Cuckoo process" có thể lặp vô tận. Vì vậy, số lần tương tác đổi chỗ được giới hạn bởi hằng số biên $M=3\log_{1+\varepsilon}m$. Trong trường hợp này, các khóa trong 2 bảng được tổ chức lại bằng các hàm băm mới h'_1 và h'_2, quá trình lặp lại đệ qui cho mỗi khóa.

3. KIẾN TRÚC HỆ THỐNG SO TRÙNG MẪU DỰA TRÊN CUCKOO HASHING DÙNG FPGA



Hình 3. Mô đun FPGA-based Cuckoo Hashing với khả năng tìm kiếm song song. Bảng T_1 và T_2 lưu trữ chỉ số của mẫu; T_3 lưu trữ mẫu thực.



Hình 4: Xử lý song song lớn với N ký tự (N = 4). Các mô đun Cuckoo được kết nối với bộ đệm nhập tại những địa chỉ được xác định trước.

Giải thuật CH được chúng tôi cải tiến và áp dụng cho phần so trùng mẫu tĩnh trong phần lọc nội dung của gói dữ liệu [7, 8]. Thông thường, tập mẫu pattern được tiền xử lý và xây dựng sẵn ở bên trong hệ thống. Chuỗi dữ liệu đi vào sẽ so sánh với tất cả các patterns. Vì vậy, Cuckoo hashing là lựa chọn thích hợp để đảm bảo thời gian tìm kiếm hằng số và đạt được tốc độ của mạng. Hơn nữa, khả năng cập

nhật nhanh chóng không ảnh hưởng hiệu suất tìm kiếm của Cuckoo Hashing cũng là một ưu thế. Hình 2 là mô hình tổng quát của hệ thống so trùng mẫu pattern dựa trên FPGA Cuckoo Hashing. Mỗi khối (module) tên *Cuckoo* L_i lưu trữ những pattern có chiều dài i ($1 \leq i \leq 16$) ký tự. Đối với những pattern dài hơn 16 ký tự, chúng tôi sẽ phân chia chúng thành những đoạn nhỏ hơn mà có thể thêm vào trong các khối Cuckoo của pattern ngắn. Sau đó, kỹ thuật danh sách liên kết được sử dụng để kết nối những đoạn này [7].

3.1. Mô đun Cuckoo Hashing dựa trên FPGA

Để gia tăng sự tận dụng bộ nhớ, chúng tôi xây dựng một khối hashing cho mỗi chiều dài của pattern và sử dụng lưu trữ gián tiếp. Các bảng hash nhỏ và thưa thớt chứa các chỉ số của mẫu là địa chỉ của một bảng lưu trữ mẫu thực cô đặc hơn. Cách tiếp cận của chúng tôi cũng thay đổi để quá trình tìm kiếm là song song. Việc thêm vào cũng được thay đổi để lựa chọn tốt hơn các khoảng trống trong hai bảng hash. Với những sự thay đổi này trong kiến trúc, hệ thống chúng tôi có thể tận dụng đầy đủ những lợi điểm của phần cứng.

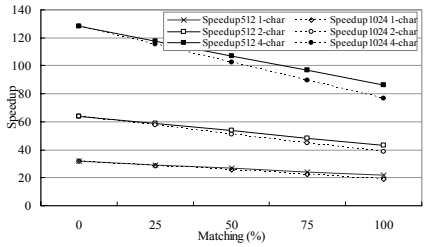
Kiến trúc của một khối CH dựa trên FPGA được trình bày trong hình 3 bao gồm 3 bảng. Hai bảng hash T_1 và T_2 là single-port SRAMs và một bảng lưu trữ mẫu T_3 là double-port SRAM cho việc xử lý song song. Các hàm hash có thể thay đổi nếu chúng được yêu cầu hash lại. Hai thanh ghi *key* và *index* là mẫu cần tìm kiếm và địa chỉ của mẫu trong T_3, tương ứng. Bên cạnh đó, hai multiplexers được sử dụng để chọn lựa địa chỉ của T_3. Cuối cùng, một bộ so sánh được dùng để so trùng chính xác *key* với hai mẫu ứng viên từ T_3.

3.2. Nguyên lý hoạt động

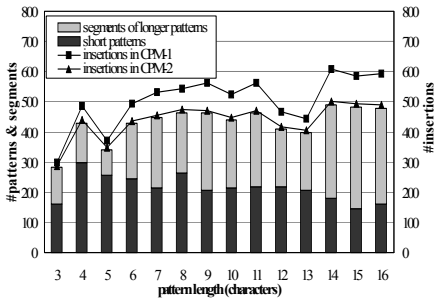
Bằng cách sử dụng kiến trúc pipeline nhiều pha và xử lý song song, quá trình tìm kiếm của máy CPM có thể xử lý một ký tự của mẫu x trong mỗi chu kỳ clock.

Đối với việc thêm vào phần tử x, như đã đề cập ở trên, chúng tôi xem xét cả hai bảng hash cùng một lúc để giảm thời gian thêm vào. Nếu một trong hai bảng hash có ô trống thì chỉ số của x được thêm vào T_1 hoặc T_2 và quá trình thêm vào kết thúc. Nếu cả hai bảng đều không trống thì chúng ta thêm chỉ số vào bảng có ít phần tử hơn. Phần tử bị lấy ra sẽ bắt đầu quá trình Cuckoo giống như phần lý thuyết trình bày ở trên.

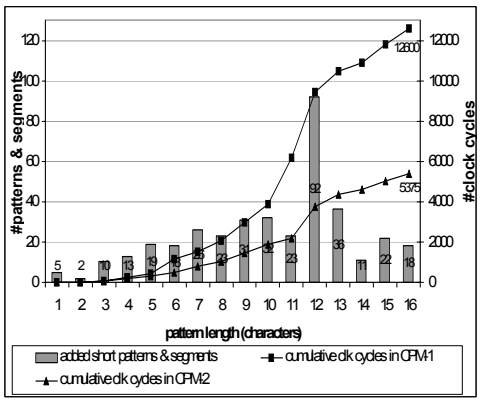
Đối với việc xóa phần tử (deletion), quá trình đơn giản như tìm kiếm. Khi sự tìm kiếm thành công, chúng ta xoá giá trị trong T_3 trước, sau đó xóa giá trị chỉ số trong bảng hash. Quá trình hoạt động chi tiết có thể tham khảo trong [7].



Hình 5. Speedup của các CPM so sánh với hiện thực nối tiếp của Cuckoo Hashing. *Matching (%)* là phần trăm của những mẫu nghi ngờ mà cần phải so trùng.



Hình 6. So sánh số lần thêm vào của 2 hệ thống CPM-1 và CPM-2 với số mẫu pattern



Hình 7. Thời gian thêm vào trung bình cho 381 chuỗi mới (patterns & segments)

Để gia tăng hiệu suất, máy CPM có thể dễ dàng mở rộng cho việc xử lý nhiều ký tự cùng một lúc. Hệ thống sẽ dịch N ký tự vào trong bộ đệm mỗi chu kỳ clock. Hệ thống có N block tương ứng với N ký tự cần xử lý. Trong mỗi block, mỗi mô đun Cuckoo nhận một ký tự từ địa chỉ đã xác nhận trước của bộ đệm nhập và giá trị hash từ mô đun trước đó. Hình 4 là một ví dụ của hệ thống xử lý 4 ký tự mỗi chu kỳ clock. Bởi vì độ phức tạp của địa chỉ cao hơn, chúng ta cần xác định địa chỉ của bộ đệm nhập mà mỗi mô đun Cuckoo được kết nối tới. Chúng ta giả sử rằng dữ liệu vào là chuỗi "$x_0x_1...x_k, x_{k+1}...$" và chúng ta dịch N ký tự mỗi chu kỳ clock. Trong mỗi block j ($1 \le j \le N$), nếu địa chỉ được kết nối tới mô đun trước đó là $A_{i-1,j}$ ($2 \le i \le L_{max\_s}$) và chứa ký tự x_k thì địa chỉ được nối tới mô đun kế tiếp, $A_{i,j}$, phải dịch N -1 từ $A_{i-1,j}$ và mô đun kế tiếp xử lý ký tự x_{k+1}. Cho ví dụ, dữ liệu vào là *"...abcd..."* và mỗi lần CPM dịch N = 4 ký tự; trong *Block 1*, *Cuckoo* L_1 tiêu thụ ký tự *'a'* ở địa chỉ $A_{i-1,j}$ thì *Cuckoo* L_2 tiêu thụ ký tự '*b'* ở địa chỉ $A_{i-1,j}+N-1$ một chu kỳ clock sau đó. Tổng quát, địa chỉ trong bộ đệm được kết nối với Cuckoo L_i của máy j là:

$$A_{i,j} = A_{i-1,j} + (N-1) = A_{i-2,j} + 2(N-1) = ... = A_{1,j} + (i-1)(N-1) \quad (1)$$

Từ phương trình (1), chúng ta có thể thấy rằng khi N = 1, địa chỉ của bất kỳ mô đun trùng với địa chỉ đầu tiên. Vì vậy, điều này đúng với thiết kế cho hệ thống xử lý một ký tự.

4. PHÂN TÍCH HIỆU SUẤT VÀ KẾT QUẢ THỰC NGHIỆM

Dựa trên kiến trúc của CPM, chúng ta có thể xác định được speedup của hệ thống, một thông số kỹ thuật đánh giá hiệu suất on-line. Chúng ta giả sử rằng mô hình Cuckoo nối tiếp có thể xử lý dữ liệu nhập mỗi chu kỳ clock và chiều dài của mẫu là nhỏ hơn 16 ký tự. Chúng ta xem xét hai trường hợp là speedup khi không xảy ra trùng (no-match) và xảy ra trùng (match). Khi hệ thống không xảy ra sự so trùng thì mô hình nối tiếp phải tìm kiếm trên cả hai bảng. Với 2 lần tìm kiếm và 16 mô đun Cuckoo chạy song song, speedup của trường hợp no-match của mỗi máy CPM là $S_{nomatch}$ = *16 x 2 = 32*.

Khi hệ thống xảy ra sự so trùng, mô hình nối tiếp đầu tiên tìm kiếm trong bảng T_1 . Nếu chưa xảy ra trùng thì lần tìm kiếm thứ hai là trong T_2. Xác suất của việc truy xuất một hoặc hai bảng tùy thuộc vào sự phân bố các mẫu trong T_1 và T_2. Thông thường, sự phân bố này trong T_1 khoảng 60%-75% phụ thuộc vào kích thước của các bảng hash. Speedup của trường hợp match của mỗi máy CPM là

$$S_{match} = 16 \times \frac{\%lookupT_1 + \%lookupT_2}{parallellookupT_1 \,\&\, T_2} < 32 \quad (2)$$

Cuối cùng, chúng ta có thể xác định speedup trung bình của hệ thống xử lý song song lớn so với mô hình nối tiếp như sau.

$$S_{avg} = N \times (\%S_{match} + \%S_{nomatch}) \quad (3)$$

Từ phương trình (3), speedup trung bình phụ thuộc vào phần trăm matching. Chúng ta minh họa rõ ràng hơn bằng hình 5. Với khả năng xử lý 4 ký tự, speedup của các máy CPM có thể lên tới *128X* khi so sánh với mô hình nối tiếp. Speedup giảm khi phần trăm matching tăng, bởi vì speedup của trường hợp trùng nhỏ hơn trường hợp không trùng như được thể hiện trong phương trình và hình. Kích thước của bảng hash cũng ảnh hưởng tới speedup. Speedup tăng khi kích thước bảng hash tăng từ 512 lên 1024. Tổng quát, speedup gia tăng khi hệ thống có thể xử lý nhiều ký tự trong mỗi chu kỳ clock.

Chúng tôi thu thập dữ liệu đầu tiên của Snort vào tháng 12 năm 2006. Có tất cả 4748 mẫu không trùng nhau, bao gồm 64873 ký tự. Trong đó, khoảng 65% có chiều dài nhỏ hơn hoặc bằng 16 ký tự. Đối với mẫu dài hơn, chúng ta phân thành các đoạn có chiều dài từ 3-16 ký tự. Chúng tôi định nghĩa *string* đại diện chung cho mẫu ngắn và các đoạn của mẫu dài. Tổng cộng, có 6,136 *strings.*

Thiết kế được hiện thực trên 2 mô hình là *CPM-1* và *CPM-2* với kích thước của bảng hash tương ứng là 512 và 1024. Những hệ thống này được dùng để đánh giá độ thỏa hiệp giữa khả năng dùng phần cứng và hiệu suất của sự thêm vào. Cả 2 hệ thống sử dụng các mô đun Cuckoo cải tiến song song như trên. Hình 6 trình bày số lần thêm vào cũng như số string trong mỗi chiều dài. Chúng ta có thể thấy số lần thêm vào của CPM-2 chỉ lớn hơn số string một chút. Thật không may mắn cho *CPM-1*, số lần thêm vào lớn hơn số string khoảng 16%.

Để minh họa khả năng cập nhật dữ liệu của hệ thống, chung tôi thu thập tập dữ liệu Snort mới hơn vào tháng 05 năm 2007 với 5026 mẫu và 68,266 ký tự. Khi so sánh với tháng 12 năm 2006, có 381 string được thêm vào bao gồm 3,476 ký tự và 15 mẫu xóa đi bao gồm 83 ký tự. Hình 7 trình bày số lần thêm vào trung bình tính theo chu kỳ clock cho việc thêm vào 381 string. Với 1024 ô nhớ của bảng hash, *CPM-2* chỉ mất 5,275 chu kỳ clock để thêm vào mà không xảy ra rehash. Trong khi đó, với *CPM-1*, rehash xảy ra ở các chiều dài 6, 11, 12 và 15 ký tự với phần trăm rất nhỏ 0.6%-2.9%. Tuy nhiên, số chu kỳ clock ở những chiều dài này tăng cao bởi vì những nhược điểm của rehashing. Vì vậy, hệ thống cần 12,600 chu kỳ clock, khoảng 2.5 lần khi so sánh với *CPM-2*.

Để xóa 15 mẫu, quá trình diễn ra nhanh chóng với khoảng 100 chu kỳ clock cho cả 2 hệ thống. Chúng ta giả sử rằng cả 2 chạy với tần số 200 Mhz, nhỏ hơn các kết quả tổng hợp trình bày tiếp sau đây. Với 5,375 và 12,700 chu kỳ clock, thời gian cập nhật chỉ khoảng 27 và 64 microseconds, tương ứng. Những kết quả này chỉ ra rằng cả hai hệ thống đều rất hiệu quả cho việc cập nhật các mẫu mới mà không cần quá trình tái cấu hình FPGA.

Bảng 1. So sánh của các hệ thống NIDS dựa trên FPGA dùng phương pháp Hashing

System	Dev.-XC (Xilinx)	Bits/ cycle	Freq. (MHz)	No. chars	No. LCs	Mem (kbits)	LCs/ char	Mem/ char (bits)	Through-Put (Gbps)	PEM
CPM-2	4VLX100	8	285	68,266	3,220	1,116	0.047	16.74	2.28	10.29
	4VLX100	16	282		6,120	2,070	0.090	31.05	4.51	10.92
	4VLX100	32	275		11,980	4,140	0.175	62.10	8.80	10.70
	2VP20	8	272		3,266	1,116	0.048	16.74	2.18	9.79
	2V6000	8	223		3,266	1,116	0.048	16.74	1.78	8.03
	2V6000	16	218		6,212	2,070	0.091	31.05	3.49	8.42
V-HashMem [13]	2VP30	8	306	33,613	2,084	702	0.060	21.39	2.49	8.60
HashMem [12]	2V1000	8	250	18,636	2,570	630	0.140	34.62	2.00	4.01
	2V3000	16	232		5,230	1,188	0.280	65.28	3.71	3.86
PH-Mem [11]	2V1000	8	263	20,911	6,272	288	0.300	14.10	2.11	4.71
	2V1500	16	260		10,224	306	0.490	14.98	4.16	6.44
ROM+Coproc[10]	4VLX15	8	260	32,384	8,480	276	0.260	8.73	2.08	5.90

Thiết kế của chúng tôi đã được phát triển trên Xilinx's ISE 8.1i. Chip hiện thực là các chip thuộc họ Virtex của hãng Xilinx. Tổng cộng, chúng tôi chỉ dùng 62 RAM blocks và 2,982 logic cells để lưu trữ 64,873 ký tự của toàn bộ tập mẫu trong XCV4LX25 FPGA chip. Throughput của một thiết kế có thể được tính bằng cách nhân tần số clock với độ rộng dữ liệu (8-bit) của những ký tự đến. Throughput của CPM có thể thay đổi từ 1.78-8.8 Gbps tùy thuộc vào kiểu của FPGA chips và số ký tự có thể xử lý mỗi chu kỳ clock.

Bảng 1 trình bày sự so sánh của các hệ thống dùng phương pháp hashing gần đây xây dựng trên FPGA. Hai thông số, Logic Cells per character (LCs/char) và SRAM bits per character (bits/char), được dùng để đánh giá sự hiệu quả của sử dụng FPGA. LCs/char xác định bằng cách chia tổng số logic cell đã sử dụng cho tổng số ký tự. bits/char là tỉ số của các khối block RAM tính bằng bit cho tổng số ký tự. Cả hai thông số này càng nhỏ càng tốt.

Với chỉ 0.043-0.047 LCs/char, CPMs là những máy hiệu quả nhất trong việc sử dụng logic cells of FPGA. Thêm vào đó, với 16.74-17.62 bits/char, việc sử dụng block RAM của CPMs cũng rất hiệu quả và có thể chấp nhận được khi so sánh với các hiện thực khác. Để có

thể đánh giá chính xác hiệu suất của các hệ thống, một thông số tên Performance Efficiency Metric (PEM) được sử dụng như là tỉ suất giữa throughput theo Gbps với số logic cell trên mỗi ký tự.

$$PEM = \frac{Throughput}{\frac{No.Logiccells + \frac{Membytes}{12}}{No.Characters}} \quad (4)$$

Giả sử rằng giá trị của 12 byte block RAM thì tương đương với một logic cell, phương trình trên tính cả 2 thông số diện tích phần cứng logic cell va block RAM để việc so sánh được công bằng giữa các hệ thống. Với PEM trong tầm 8.03-10.92, các máy CPM là tốt hơn các hệ thống dùng hashing ít nhất gần 30%.

5. KẾT LUẬN

Một máy so trùng mẫu dùng Cuckoo Hashing cho NIDS được trình bày. Theo như kết quả hiện thực, hiệu quả sử dụng phần cứng là tối ưu nhất khi so sánh với các công trình trước đây và throughput đạt được có thể lên tới 8.8 Gbps. Một đặc tính nổi trội của máy CPM này là khả năng cập nhật mẫu nhanh chóng mà không cần tái cấu hình lại FPGA. Mục tiêu công việc tiếp theo có thể là kết hợp với việc phân loại phần đầu của các quy luật (rule) để hình thành một hệ thống hoàn chỉnh cho xử lý các quy luật (rule) trong hệ thống NIDS.

CPM: CUCKOO-BASED PATTERN MATCHING APPLIED FOR NIDS

Tran Ngoc Thinh

University of Technology, VNU-HCM

ABSTRACT: *This paper describes the Cuckoo-based Pattern Matching (CPM) engine which based on a recently developed hashing algorithm called Cuckoo Hashing. We implemented the improved parallel Cuckoo Hashing suitable for hardware-based multi-pattern matching with arbitrary length. CPM is scalable with multi-character per clock cycle to sustain higher throughput rates with lower hardware resources. With the power of massively parallel processing, the speedup of CPM is up to 128X as compared with serial Cuckoo implementation. Compared to other hardware systems, CPM is far better in performance and save 30% of the area compared with the best system.*

Keywords*: pattern matching, Cuckoo hashing, FPGA*

TÀI LIỆU THAM KHẢO

[1]. "*SNORT official website*", http://www.snort.org (2009).

[2]. Y.H. Cho, S. Navab, and W.H. Mangione-Smith, *Specialized hardware for deep network packet filtering*,

Proceedings of the 12th International Conference on FPL, pp.452–461, (2002).

[3]. I. Sourdis and D.N. Pnevmatikatos, *Fast, large-scale string match for a 10gbps FPGA-based network intrusion detection system*, International Conference on FPL, pp.880–889, (2003).

[4]. J. Moscola, J. Lockwood, R.P. Loui, and M. Pachos, *Implementation of a content-scanning module for an internet firewall*, Proceedings of the 11th Annual IEEE Symposium on FCCM, pp.31–38, (2003).

[5]. G. Papadopoulos and D.N. Pnevmatikatos, *Hashing + memory = low cost, exact pattern matching*, International Conference on FPL, pp.39–44, (2005).

[6]. C.R. Clark and D.E. Schimmel, *Scalable pattern matching for high speed networks*, Proceedings of the 12th Annual IEEE Symposium on FCCM, pp.249–257, (2004).

[7]. T.N. Thinh, S. Kittitornkun, and S. Tomiyama, *Applying Cuckoo hashing for FPGA-based pattern matching in NIDS/NIPS*, IEEE International Conference on FPT, pp.121–128, (2007).

[8]. T.N. Thinh, S. Kittitornkun, and S. Tomiyama, *PAMELA: Pattern Matching Engine with Limited-time Update for NIDS/NIPS*, in IEICE Transactions of Information and Systems, Vol. E92-D, No.5, (2009).

[9]. R. Pagh and F.F. Rodler, *Cuckoo hashing*, Journal of Algorithms, vol.51, no.2, pp.122–144, (2004).

[10]. D. Pnevmatikatos and A. Arelakis, "Variable-length hashing for exact pattern matching," *International Conference on FPL*, pp. 1-6, (2006).

[11]. I. Sourdis, D. Pnevmatikatos, S. Wong and S. Vassiliadis, *A reconfigurable perfect-hashing scheme for packet inspection*, the 15th International Conference on FPL, pp. 644-647, (2005).