논문 15-40-11-09 The Journal of Korean Institute of Communications and Information Sciences '15-11 Vol.40 No.11
http://dx.doi.org/10.7840/kics.2015.40.11.2169

네트워크 침입 탐지 시스템에서 다중 엔트리 동시 비교기를 이용한 고속패턴 매칭기의 설계 및 구현

전 명 재◆, 황 선 영°

Design and Implementation of High-Speed Pattern Matcher Using Multi-Entry Simultaneous Comparator in Network Intrusion Detection System

Myung-Jae Jeon◆, Sun-Young Hwang°

요 약

본 논문은 네트워크 침입 탐지 시스템에서 CAM 및 해시 구조 기반 알고리듬의 비용 한계를 극복하기 위해 RAM을 이용한다. RAM을 이용한 기존 알고리듬의 다중 엔트리 처리 시 실시간 처리속도 지연 문제를 보완한 새로운 패턴 매칭기를 제안한다. 제안된 패턴 매칭기는 Merge FSM 알고리듬을 적용하여 스테이트의 수를 줄이고, RAM을 사용하기 위해 스테이트 블록과 엔트리 블록을 포함한다. 입력된 문자열과 비교할 엔트리문자열이 여러 개 존재할 때 엔트리 블록에서 입력된 문자열과 엔트리 문자열들을 동시에 비교한다. 제안된 패턴 매칭기는 Snort 2.9 규칙을 이용하여 검증하였다. 실험결과 기존 탐색 방법과 비교하여 메모리 접근 빈도가 15.8% 감소하였고, 전체 메모리 크기는 2.6% 증가하였으며, 처리속도는 47.1% 증가하였다.

Key Words : NIDS, Pattern Matching, RAM, FPGA

ABSTRACT

This paper proposes a new pattern matching module to overcome the increased runtime of previous algorithm using RAM, which was designed to overcome cost limitation of hash-based algorithm using CAM (Content Addressable Memory). By adopting Merge FSM algorithm to reduce the number of state, the proposed module contains state block and entry block to use in RAM. In the proposed module, one input string is compared with multiple entry strings simultaneously using entry block. The effectiveness of the proposed pattern matching unit is verified by executing Snort 2.9 rule set. Experimental results show that the number of memory reads has decreased by 15.8%, throughput has increased by 47.1%, while memory usage has increased by 2.6%, when compared to previous methods.

Ⅰ. 서 론

최근 스마트폰 및 웨어러블 디바이스의 증가로 인한 네트워크 트래픽이 기하급수적으로 증가하고 있으며, 네트워크와 개인생활이 밀접해 짐에 따라 개인정보를 노리는 네트워크 침입사례가 빈번히 발생하고 있다. 네트워크 보안 정책을 위반하는 행위들을 실시간으로 감시하기 위해 네트워크 침입 탐지 시스템

◆ First Author : Sogang University Department of Electronic Engineering, jsungki@sogang.ac.kr, 학생회원
° Corresponding Author : Sogang University Department of Electronic Engineering, hwang@sogang.ac.kr, 종신회원
논문번호 : KICS2015-10-336, Received October 16, 2015; Revised November 10, 2015; Accepted November 10, 2015

(Network Intrusion Detection System)이 활발히 연구 중에 있다.

네트워크 침입 시도나 공격은 패킷의 패턴이나 서브 패턴의 조합으로 이루어져 있으며, 네트워크 침입 탐지 시스템은 실시간 침입시도나 공격을 감시하기 위한 패턴 매칭 기법을 이용한다. 패턴 매칭 기법은 네트워크 침입 탐지 시스템의 성능의 중요한 척도이다[1]. 현재 네트워크 침입 탐지 시스템은 대부분 소프트웨어를 기반으로 상용화 및 연구[2,3]가 진행되고 있으며, 기존의 잘 알려진 Snort[4]와 OSSEC[5] 같은 알고리듬 기반의 소프트웨어 네트워크 침입 탐지 시스템은 10Gbps의 트래픽을 실시간으로 감시할 수 없다. 따라서 패턴 매칭 시 소모되는 시간을 줄이기 위해 하드웨어 네트워크 침입 탐지 시스템 연구가 진행되었다. 하드웨어 기반 네트워크 침입 탐지 시스템은 패턴 정보를 로직으로 구현하는 방법[6-8]과 메모리 아키텍처를 이용한 방법[9-11]으로 나눠진다. 패턴 정보를 로직으로 구현하는 패턴 매칭기는 빠르게 악의적인 패턴을 구별할 수 있지만 새로운 패턴 정보 업데이트 시 로직을 재구성해야 한다는 단점이 있다. 반면 메모리 아키텍처를 이용한 패턴 매칭기는 새로운 패턴 정보 업데이트 시 제약이 없어 활발히 연구 중에 있다.

악의적인 패턴이 나날이 증가함에 따라 요구되는 메모리의 크기도 점차 증가하고 있는 추세이며, 최근 대부분 연구에서는 악의적인 패턴을 메모리에 효율적으로 저장하는 알고리듬에 초점을 맞추고 있다. 메모리를 효율적으로 사용하기 위해서 다양한 알고리듬이 소개되었으나 동일한 스테이트를 합병하여 스테이트의 수를 최소화하는 Merge FSM 알고리듬[12]이 잘 알려져 있다. CAM 기반의 Merge FSM 패턴 매칭기는 색인 문자열을 CAM 내의 모든 엔트리 tag들과 동시 비교하기 위한 구조가 복잡하고 전력소모가 매우 크며, 큰 라우팅 테이블이나 IPv6로의 확장에도 적합하지 않다는 단점을 가지고 있다. CAM 기반 하드웨어 매칭기의 단점을 보완하기 위해 RAM 기반 패턴 매칭기가 연구되었다[13,14].

RAM 기반의 패턴 매칭기는 CAM 기반의 패턴 매칭기에 비해 구조가 단순하고, 비용이 저렴하며, 큰 라우팅 테이블이나 IPv6로 확장에 용이하다는 장점을 가지고 있으나 처리 속도가 느리다는 단점[14,15]을 가지고 있다. 이 단점을 보완하기 위해 다양한 자료 탐색 기법들이 연구되었다. 대표적인 탐색기법인 Search method based on the out-degree indexing [14]은 스테이트 천이시킬 수 있는 엔트리들이 다수 존재할 경우 테이블로 구성하고, 엔트리와 입력 문자열을 하나씩 비교한다. 본 논문에서 Search method based on the out degree indexing은 주 비교대상으로 출현빈도가 높아 Out-degree indexing으로 축약하여 표현한다. Out-degree indexing 방법은 엔트리 개수가 증가함에 따라 메모리를 읽는 횟수도 선형적으로 증가하여 병목구간을 형성한다.

본 논문에서는 다중 엔트리들을 가진 스테이트에서 입력 문자열을 하나씩 비교하지 않고, 입력 문자열과 다중 엔트리들을 동시에 비교할 수 있는 탐색 기법을 제안한다. 제안된 탐색 기법을 적용한 패턴 매칭기는 다중 엔트리를 가질 경우 기존 탐색기법을 적용한 패턴 매칭기에 비해 빠르게 엔트리를 비교하여 다중 엔트리로 인한 오버헤드를 감소시킨다.

본 논문의 2장에서는 관련 연구를 통해 기존 패턴 매칭기에 대해 분석하고, 3장에서는 제안한 탐색기법에 대해 설명한다. 4장에서는 구현된 패턴 매칭기의 성능을 분석한다. 5장에서는 결론 및 추후 과제를 제시한다.

Ⅱ. 관련 연구

본 절에서는 네트워크 침입 탐지 시스템에서 하드웨어 패턴 매칭기 관련 기존 연구를 소개 및 분석하고, RAM을 이용하기 위한 기존의 탐색기법을 소개한다.

2.1 CAM 기반 하드웨어 패턴 매칭기

Aho Corasick[16]은 잘 알려진 문자열 패턴 매칭 알고리듬으로 패턴을 문자열 트리로 구성하여 패턴 매칭을 수행한다. 이 알고리듬은 메모리 접근 빈도가 높아 처리속도가 느리고, 유한오토마타로 구현할 경우 스테이트 수가 많아 고용량 메모리를 요구하는 단점을 가지고 있다[17].

Aho Corasick 알고리듬을 보완하기 위해 다양한 알고리듬들이 제안되었다. JACK(Jump Ahead Aho Corasick Alogorithm)[17]은 Aho Corasick 알고리듬의 처리속도 증진을 위해 다중 문자열을 한 번에 비교하도록 하였고, 하나의 스테이트에 다중 문자열이 맵핑되므로 메모리 크기가 줄어들었다. PFAC(Parallel Failure less Aho Corasick Algorithm)[18,19]은 GPU(Graphical Processing Unit)를 기반으로 다중 쓰레드에 입력된 문자열을 각각 할당하여, 다중 쓰레드에서 동시에 다중 문자열을 비교한다. 각 쓰레드마다 스테이트 천이도를 가져야 하므로 고용량 메모리를 요구하며 처리속도는 여타 알고리듬에 비해 매우 빨

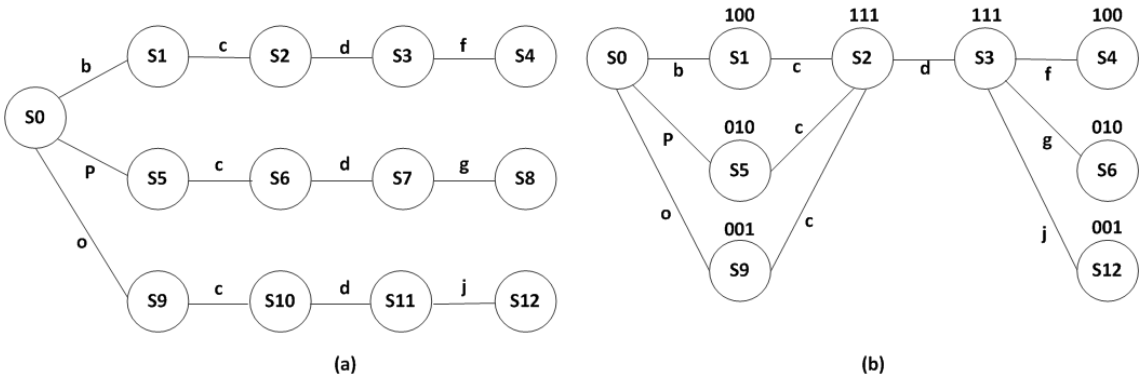

그림 1. 알고리듬 별 상태 천이도. (a) Aho Corasick, (b) Merge FSM.
Fig. 1. State diagram of each algorithm. (a) Aho Corasick, (b) M erge FSM.

라진다. Merge FSM 알고리듬[12]은 Aho Corasick 알고리듬에 의해 생성된 트리에서 동일한 스테이트를 가지는 노드들을 합병하여 전체 스테이트 수를 줄이는 알고리듬이다. 그림 1은 Aho Corasick 알고리듬을 적용한 패턴 트리 정보와 Merge FSM 알고리듬을 적용한 패턴 트리 정보를 나타낸다. 패턴 트리의 노드는 스테이트를 의미한다. 스테이트 S2와 스테이트 S6, 스테이트 S10은 동일하므로 하나의 스테이트로 병합할 수 있다. Aho Corasick 알고리듬에 “bcdg“ 문자열이 입력되면 스테이트 S3에서 스테이트 S0로 천이하여 정상적인 패턴으로 인식한다. Merge FSM 알고리듬을 적용 시 “bcdg“ 문자열이 입력되면 스테이트 S3에서 스테이트 S6으로 천이하여 악의적인 패턴으로 오인식하며 이를 방지하기 위하여 path 벡터를 이용한다. 이전 스테이트의 path 벡터와 현 스테이트 path 벡터를 AND 연산을 통해 연산 결과가 이전 path 벡터와 동일하지 않으면 정상패턴으로 인식하여 오인식을 방지한다. 스테이트 합병에 의해 전체적인 스테이트 수가 줄어 메모리 용량이 감소하고 전체적인 천이 횟수가 감소하여 처리속도가 향상된다. 타 알고리듬과 비교하여 Merge FSM 알고리듬이 처리속도 및 메모리 효율에 있어 가장 합리적인 알고리듬으로 알려져 있다.

2.2 RAM 기반 하드웨어 패턴 매칭기

TCAM(Ternary Content Addressable Memory)을 이용한 하드웨어 패턴 매칭기[20,21]는 TCAM의 가격과 회로의 복잡성, IPv6의 도입 시 거대한 라우팅 테이블을 적용하기 어렵다는 단점[14]을 가지고 있다. TCAM의 대체재로 RAM을 이용한 패턴 매칭기가 연구되고 있고, TCAM 보다 처리속도가 늦다는 단점을 가진 RAM의 처리속도 향상을 위해 자료 탐색 기법이 연구되고 있다. Out-degree index 탐색 기법을 적용한 패턴 매칭기[14]는 Merge FSM 알고리듬을 이용하여 패턴 정보를 트리구조로 구성하고 스테이트의 수를 최소화하였다. 스테이트와 천이 가능한 엔트리 수 정보를 가진 탐색 테이블과 천이 가능한 엔트리 문자열과 다음 스테이트 주소를 가진 스테이트 메모리 테이블을 제안하였다. 그림 2는 그림 1의 예시를 Out-degree index 탐색 기법을 적용해 나타낸 것이다. 그림 1에서 스테이트 S0와 S3은 3개의 엔트리 문자열을 갖는다. 만약 입력된 문자열이 “ocdj”라면 초기 스테이트 S0을 통해 스테이트 메모리 주소 0부터 검색을 시작한다. 스테이트 메모리의 엔트리 문자열 “b”와 입력 문

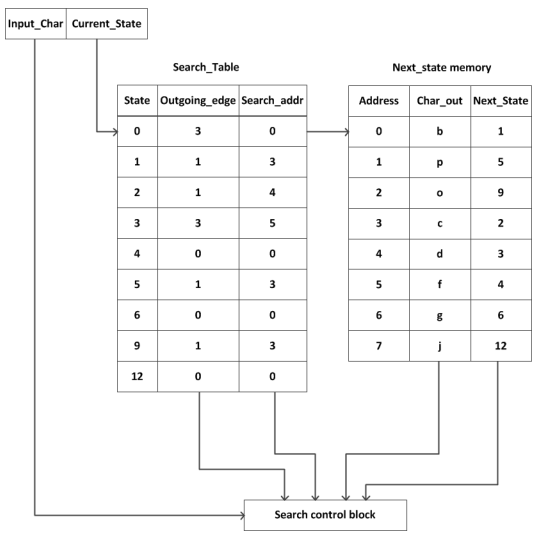

그림 2. Out-degree index 탐색 기법을 적용한 패턴 매칭기 구조.
Fig. 2. Architecture of Out-degree index search method.

자열 "o"를 비교하여 동일하지 않으므로, 어드레스 주소를 증가시켜 다음 스테이트 메모리의 엔트리 문자열 "p"와 비교한다. 엔트리 문자열 "p"와 입력 문자열 "o"를 비교하여 동일하지 않으므로, 어드레스 주소를 증가시켜 다음 스테이트 메모리의 엔트리 문자열 "o"와 비교한다. 입력 문자열과 엔트리 문자열이 동일할 경우 스테이트 메모리의 다음 스테이트 주소로 천이한다. 엔트리의 개수가 n개일 경우 최대 n번의 메모리를 읽어야 하므로 패턴 매칭기의 병목구간을 형성한다.

Ⅲ. 제안된 패턴 탐색 기법

본 장에서는 RAM 기반의 패턴 매칭기를 설계하기 위한 다중 엔트리 동시 탐색 기법을 제안한다. 제안된 탐색 기법은 두 종류의 메모리 블록을 제시하고, 가장 효율적인 메모리 블록 크기를 결정한다. 또한 제안된 패턴 탐색 기법을 적용한 전체적인 패턴 매칭기 구조를 보이고 그 동작을 설명한다.

3.1 메모리 블록

기존의 C. Jasmine이 제안한 Out-degree index 탐색 기법과 달리 현재 스테이트가 다중 엔트리를 갖는 스테이트일 경우 엔트리 블록을 삽입하여 엔트리 개수만큼 증가하던 메모리 접근 빈도를 낮춤으로써, 다중 엔트리의 경우 발생하는 오버헤드를 감소시킨다. 스테이트가 단일 엔트리를 가질 경우 그림 3(a)와 같이 엔트리 문자열, 매칭 결과, path 벡터, 다음 스테이트 주소로 구성 된 스테이트 블록을 사용하고, 스테이트가 다중 엔트리를 가질 경우 기존의 스테이트 블록에 그림 3(b)와 같이 엔트리 문자열들을 가진 엔트리 블록이 추가된다. 엔트리 블록은 남아 있는 엔트리 블록의 개수를 나타내는 엔트리블록 번호(Entry No.)와 엔트리 문자열(Entry Char[1-7])들로 구성된다. 엔트리블록 번호 값과 엔트리 문자열 비교 값을 이용해서 다음 스테이트 메모리의 주소를 결정하게 된다.

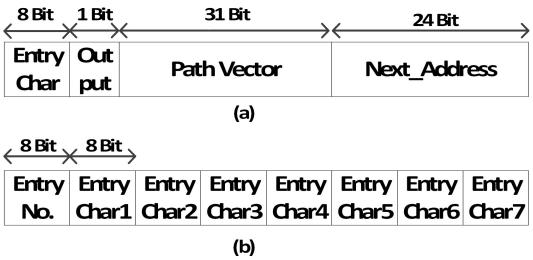

그림 3. 제안된 메모리 블록. (a) 스테이트 블록, (b) 엔트리 블록.
Fig. 3. Proposed memory block. (a) State block, (b) Entry block.

3.2 메모리 블록의 크기 결정

제안된 스테이트 블록과 엔트리 블록을 이용한 탐색 기법은 엔트리 블록의 크기에 따라 성능이 좌우된다. 엔트리 블록의 크기 증가는 동시에 비교할 수 있는 엔트리 개수가 증가하여, 엔트리 블록 크기 이상의 엔트리를 가질 경우 삽입되는 엔트리 블록의 개수를 줄일 수 있다. 비교해야 할 엔트리의 개수보다 엔트리 블록의 크기가 클 경우 사용하지 않는 메모리가 증가하므로 엔트리 블록의 크기가 클수록 메모리 효율성이 저하된다.

그림 4는 Snort 2.9의 규칙을 분석하여 엔트리 크기에 따른 메모리 접근 빈도와 메모리의 크기를 나타낸 것이다. 그림 4(a)는 메모리 접근 빈도가 엔트리 크기에 따라서 감소하는 양상을 나타냄을 알 수 있다. 그림 4(b)는 엔트리 크기가 증가함에 따라 메모리의 크기 또한 증가함을 알 수 있다. 그림 4(a)에서 엔트리 블록의 크기가 4 바이트일 때와 8 바이트일 때 메모리 접근빈도는 큰 차이를 보이며, 8 바이트일 때와 16 바

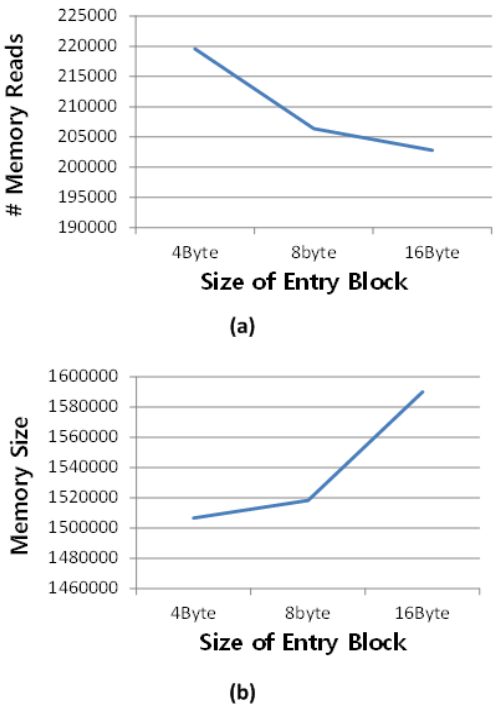

그림 4. 엔트리 블록 크기 별 메모리 접근 빈도와 메모리의 크기. (a) 메모리 접근 빈도, (b) 메모리의 크기.
Fig. 4. Number of memory reads and size of memory vs. Size of Entry Block. (a) Number of memory reads, (b) Size of memory.

이트일 때 메모리의 접근빈도는 큰 차이를 보이지 않는다. 반면 메모리의 크기는 4 바이트와 8바이트 사이에 큰 차이는 없으나, 8 바이트와 16 바이트 사이에는 큰 차이를 보인다. 메모리의 접근 빈도가 적을수록 처리 속도가 빠르며, 메모리의 크기가 작을수록 효율적인 메모리 사용이 가능하므로 엔트리 블록의 크기를 8 바이트로 고정한다.

3.3 제안된 패턴 매칭기 구조

그림 5는 전반적인 제안된 패턴 매칭기의 구조를 나타낸다. 제안된 패턴 매칭기는 컨트롤 회로에 의해 RAM에서 받은 정보를 엔트리 레지스터 또는 스테이트 레지스터에 저장한다. 다중 엔트리를 가지는 스테이트일 경우 스테이트 레지스터의 path 벡터의 모든 비트가 1이다. Path 벡터의 모든 비트가 1일 때 현재 메모리 주소를 8 바이트만큼 증가시키고, 컨트롤 회로에 신호를 보내 8 바이트만큼 증가된 주소로부터 얻어지는 정보를 엔트리 레지스터에 저장한다. 입력 문자열을 엔트리 문자열들과 비교하여 동일한 문자열이 존재할 경우 디코더를 통하여 다음 스테이트의 주소를 출력하며, 모든 엔트리 문자열과 입력 문자열이 불일치 할 경우 초기 스테이트 블록의 주소를 출력한다. 컨트롤 회로에 신호를 보내 다음 주소로부터 얻어지는 정보는 스테이트 레지스터에 저장한다.

그림 6은 각 레지스터별 콤비네이션 회로를 나타낸다. 그림 6(a)는 스테이트 레지스터 콤비네이션 회로이다. 스테이트 레지스터와 연결된 회로는 입력 문자열과 레지스터의 문자열이 동일할 경우 이전 스테이트의 path 벡터와 현재 스테이트의 path 벡터를 AND 연산하고 연산 결과가 모든 비트가 0을 갖지 않는다면 다음 스테이트 주소를 출력한다. 입력 문자열과 레지스터의 문자열이 동일하지 않거나 현재 스테이트와 이전 스테이트의 path 벡터를 AND 연산한 결과 모든

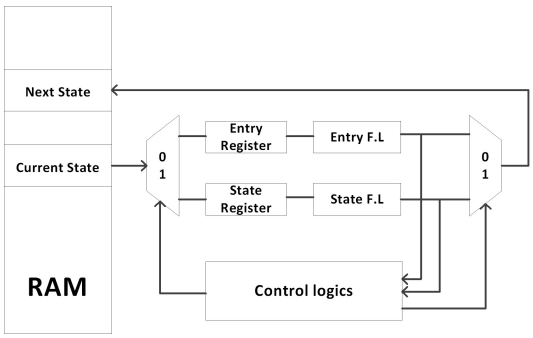

그림 5. 제안된 패턴 매칭기 구조.
Fig. 5. Proposed pattern matching architecture.

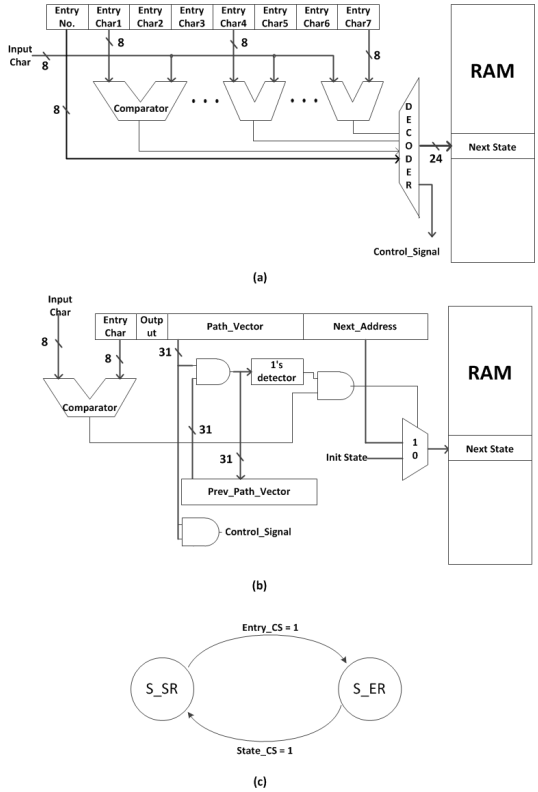

그림 6. 사용된 레지스터들의 회로 구조. (a) 스테이트 레지스터, (b) 엔트리 레지스터, (c) 컨트롤 로직.
Fig 6. Structures of registers used in the circuits. (a) State register, (b) Entry register, (c) Control logics.

비트가 0을 가질 경우 초기 스테이트로 천이한다. 그림 6(b)는 엔트리 레지스터 콤비네이션 회로이다. 엔트리 레지스터의 엔트리블록 번호는 현 스테이트에서 엔트리가 7개 이상일 경우 RAM상 엔트리 블록이 연속적으로 존재하고 이를 구별한다. 엔트리 레지스터에 존재하는 엔트리 문자열과 입력 문자열을 동시에 비교하여 동일한 문자열이 있을 경우 디코더를 이용하여 다음 스테이트 블록 주소를 출력한다. 엔트리 문자열들과 입력 문자간 동일한 문자열이 없을 경우 다음 엔트리 블록의 주소를 출력하며, 다음 엔트리 블록이 존재 하지 않을 경우 초기 스테이트의 주소를 출력한다. 그림 6(c)는 컨트롤 로직 회로이다. 컨트롤 로직 회로는 2개의 스테이트로 구성되어 있다. 스테이트 레지스터 스테이트에서 스테이트 콤비네이션 회로로부터 컨트롤 신호가 입력되면 엔트리 레지스터 스테이트로 천이하며, 엔트리 레지스터 스테이트에서 엔트리 콤비네이션 회로로부터 컨트롤 신호가 입력되면 스테이트 레지스터로 천이한다. 스테이트 레지스터 스테이트일 경우‘1’의 출력을 내고 엔트리 레지스터 스테이

트일 경우 '0'의 출력을 낸다. 각 멀티플렉서에서 컨트롤 로직의 출력을 이용하여 RAM으로부터 얻어지는 정보를 각 레지스터에 저장하고 각 레지스터의 콤비네이션 회로의 출력을 콤비네이션 회로와 RAM사이의 디멀티플렉서를 이용해 RAM에 선택적으로 전달한다.

그림 7은 위 그림 1의 패턴정보를 제안된 탐색기법을 적용한 예이다. 입력된 문자열이 "pcdg"일 경우 초기 스테이트 S0는 다중 엔트리를 가지고 있으므로 메모리 주소를 증가시켜 엔트리 블록을 엔트리 레지스터에 로드하고, 엔트리 문자열을 비교하여 현 메모리 주소로부터 오프셋을 더한 스테이트 S5의 주소를 출력한다. 스테이트 S5는 단일 엔트리를 갖는 스테이트로 스테이트 블록을 스테이트 레지스터에 저장하고 문자열을 비교하여 스테이트 S2 주소를 출력한다. 위의 방법을 동일하게 반복하여 입력된 문자열이 "pcdg"일 경우 S0 State, S0 Entry, S5 State, S2 State, S3 State, S3 Entry, S6 State 순으로 메모리를 총 7회 읽어온다. 입력된 문자열이 "ocdj"일 때 또한 S0 State, S0 Entry, S9 State, S2 State, S3 State, S3 Entry, S12 State 순으로 메모리를 총 7회 읽어온다. 반면 Out-degree Index 탐색 기법을 이용할 경우 "pcdg"가 입력 될 경우 7회 "ocdj"가 입력 될 경우 8회의 메모리를 읽어온다.

그림 8은 다중엔트리를 갖는 스테이트의 추가되는 메모리 접근 빈도를 나타낸다. 기존의 Out-degree index 탐색 기법을 적용하였을 경우 엔트리가 증가함에 따라 메모리 접근 빈도가 엔트리 수만큼 증가하나 제안한 탐색 기법은 엔트리가 증가함에 따라 메모리 접근 빈도의 증가폭은 크게 감소한다. 엔트리의 개수가 2개일 경우 기존 탐색기법과 동일한 처리속도를 나타내며, 엔트리 개수가 2개 이상일 경우 엔트리 개수가 늘어남에 따라 제안한 탐색기법의 처리속도가 기존

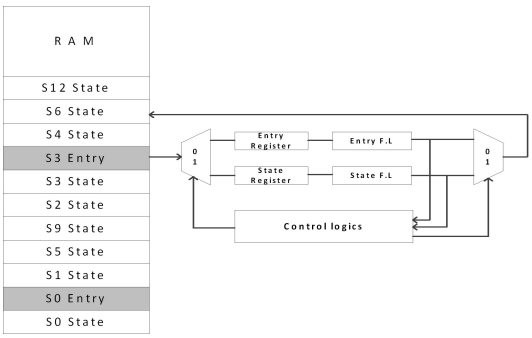

그림 7. 제안된 메모리 탐색기법의 예시.
Fig. 7. Example of proposed method.

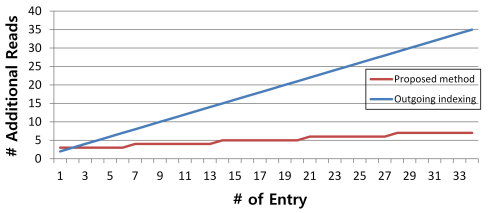

그림 8. 엔트리 개수에 따른 추가되는 메모리 접근 빈도 비교
Fig. 8. Comparison number of additional memory reads by number of entry.

탐색기법에 비해 월등히 좋아지는 것을 알 수 있다.

Ⅳ. 실험 결과

제안된 탐색 기법을 적용한 패턴 매칭기는 VHDL 언어로 구현하였으며, 구현된 매칭기는 Modelsim 시뮬레이터로 기능을 검증하였다. Snort2.9 규칙을 패턴 매칭기에 적용하여 성능을 실험하였다. 처리율을 측정하기 위해 와이어샤크[22]를 이용하여 패킷을 수집하였고, 수집한 패킷에 대하여 패턴 매칭을 수행하였다.

4.1 Snort 2.9 rule set 분석

Snort 2.9 rule set을 Merge FSM 알고리듬을 적용할 경우 스테이트 수를 기존 AC 알고리듬에 비해 27%가량 감소시킬 수 있다. 전체 총 스테이트 중 엔트리를 2개 이상 보유하고 있는 스테이트는 총 스테이트의 6.2%이다. 악의적인 패턴들 중 동일한 서브패턴으로 구성된 패턴들이 많고 동일한 서브패턴을 가질 경우 동일한 서브패턴으로 구성된 패턴들은 하나의 서브패턴으로 병합 가능하며 엔트리의 수를 증가시키지 않는다.

그림 9는 Snort 2.9 rule set을 Merge FSM 알고리

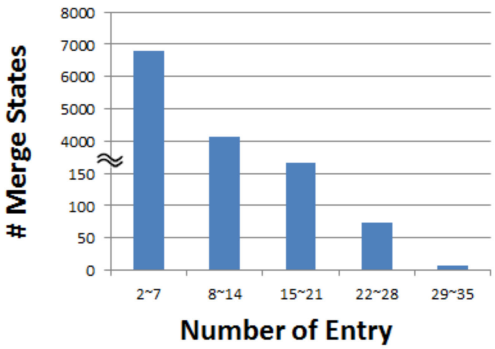

그림 9. 엔트리 개수에 따른 병합 스테이트의 수
Fig. 9. Number of merge states by number of entry.

듬을 적용하여 병합된 스테이트의 엔트리 수에 따른 분포를 나타낸 것이다. 최대 엔트리는 32개를 가지고 있으며, 엔트리 2개부터 7개 사이를 가지는 스테이트가 전체 엔트리의 가진 스테이트의 60.9%를 차지하고 엔트리 8개부터 14개 사이를 가지는 스테이트가 36.9%차지하여 엔트리 2개부터 14개 사이를 가지는 스테이트가 병합된 스테이트 전체의 약 97%이상을 차지한다. 가장 많은 엔트리를 가진 스테이트는 초기 스테이트로 기존 탐색방법을 이용했을 경우 초기 스테이트에서 최악의 경우 32번의 메모리 접근 후에 천이 된다. 제안한 탐색방법은 초기 스테이트에서 최악의 경우 5번의 메모리 접근 후에 천이 된다.

4.2 성능 비교

표 1은 Out-degree index 탐색 방법과 제안한 탐색 방법 간 메모리 접근 빈도와 메모리 크기, 처리율을 비교하였다. 처리율을 비교하기 위해 와이어 샤크를 통해 수집한 패킷을 이용하였고, 250Mhz 512k*72 Sync SRAM을 이용하여 패턴 매칭기를 구성하였다. 기존의 Out-degree index 방법에 비해 제안된 방법은 워스트 케이스 메모리 접근 빈도가 15.8% 감소하였고 전체 메모리는 2.6% 증가하였으며, 처리율은 2.8 Gbps로 기존 방법에 비해 47.1% 신속하게 패턴 매칭을 수행 할 수 있었다. 일상생활의 패킷은 악의적인 패턴이 존재하지 않아 패턴 매칭 도중 초기 스테이트로 천이가 자주 발생하였으며, 초기 스테이트가 가장 많은 다중 엔트리를 가지고 있으므로 제안한 탐색 방법의 다중 엔트리 동시 비교 장점이 극대화 되어 워스트 케이스 메모리 접근 빈도의 감소율보다 실제 처리율의 증가율이 크게 나타났다. 실험을 통하여 Out-degree index 방법보다 제안된 방법을 적용한 패턴 매칭기가 고속으로 입력 패킷을 검색 할 수 있음을 보였다.

표 1. Snort 2.9 규칙을 이용한 성능 비교 결과
Table 1. Comparison results of Snort 2.9 rule performance

	Out-degree index[14]	Proposed method	Comparison (%)
# transitions	181,194	181,194	0%
# states	179,063	179,063	0%
# memory read	245,169	206,370	15.8%
# memory size (Bytes)	1,479,146	1,518,274	-2.6%
Throughput	1.7 Gbps	2.5 Gbps	47.1%

V. 결론 및 추후 과제

본 논문에서는 기존에 잘 알려진 Merge FSM 알고리듬을 적용해 스테이트의 수를 최소화하고, 패턴 매칭기를 RAM과 간단한 회로로 구성하여 고속으로 패턴 매칭을 할 수 있도록 구현하였다. 현재 사용되는 패턴 매칭기는 대부분 소프트웨어 기반으로 구현되어 있으나 속도가 느리고, 하드웨어로 구현된 패턴 매칭기는 속도가 빠르다는 장점이 있으나 CAM의 메모리 구조의 복잡성과 고비용 문제로 상용화되기 어렵다. 제안한 패턴 매칭기는 패턴 정보를 RAM에 저장하므로 업데이트가 간단하고 저비용으로 구현 가능하다. 패턴 저장을 위해 요구되는 RAM 용량은 Snort 2.9 규칙을 적용 하였을 때 1.5MB로 기존의 RAM 기반 패턴 매칭기에 비해 3.9% 증가하나 다중 엔트리 동시 비교를 통해 메모리를 접근 빈도를 15.8% 줄여 처리 속도를 47.1% 향상시켰다. 실험 결과 제안된 패턴 매칭기는 구조가 간단하고 하드웨어 리소스의 사용량이 작아 네트워크 침입 탐지 시스템에서 패턴 매칭을 위하여 효율적으로 사용될 수 있을 것으로 기대된다. 추후 CAM 기반의 패턴 매칭기에 비해 RAM 기반 패턴 매칭기의 처리속도가 느리다는 단점을 보완하기 위해서 여러 개의 RAM 기반 패턴 매칭기를 병렬적으로 수행하고 실제 패킷의 데이터 출현 빈도에 관한 통계를 통한 출현 빈도가 높은 문자열을 초기에 엔트리 비교할 수 있도록 하드웨어를 설계하여 전체 처리속도를 향상 시키는 추가 연구가 필요하다.

References

[1] M. Fisk and G. Varghese, *An analysis of fast string matching applied to content-based forwarding and intrusion detection*, Technical Report, CS2001-0670, University of California, San Diego, 2002.

[2] J. Choi, J. Park, and M. Kim, “Processing speed improvement of HTTP traffic classification based on hierarchical structure of signature,” *J. KICS*, vol. 39, no. 4, pp. 191-199, Apr. 2014.

[3] K. Shim and S. Yoon, “Automatic generation of snort content rule for network traffic analysis,” *J. KICS*, vol. 40, no. 4, pp. 666-

677, Apr. 2015.

[4] Retrieved Sept. 3, 2015, from http://www.snort.org

[5] T. Jack, "Intrusion detection using open source tools," *Informatica Economica J.*, vol. 12, no. 2, pp. 75-79, 2008.

[6] Z. Baker and V. Prasanna, "High-throughput linked-pattern matching for intrusion detection systems," in *Proc. Symp. ANCS*, pp. 193-202, Princeton, NJ, Oct. 2005.

[7] C. Clark and D. Schimmel, "Scalable pattern matching for high speed networks," in *Proc. 12th Ann. IEEE Symp. FCCM,* pp. 249-257, Napa, CA, Apr. 2004.

[8] B. Hutchings, R. Franklin, and D. Carver, "Assisting network intrusion detection with reconfigurable hardware," in *Proc. 10th Annu. IEEE Symp. FCCM*, pp. 111-120, Napa, CA, Apr. 2002.

[9] M. Alicheery, M. Muthuprasanna, and V. Kumar, "High speed pattern matching for network IDS/IPS," in *Proc. IEEE ICNP*, pp. 187-196, Santa Barbara, CA, Nov. 2006.

[10] Y. H. Cho and W. H. Mangione-Smith, "Fast reconfiguring deep packet filter for 1+gigabit network," in *Proc. 13th Ann. IEEE Symp. FCCM*, pp. 215-224, Napa, CA, Apr. 2005.

[11] Y. H. Cho and W. H. Mangione-Smith, "A pattern matching co-processor for network security," in *Proc. 42nd IEEE/ACM Des. Autom. Conf.,* pp. 234-239, Anaheim, CA, Jun. 2005.

[12] C. Lin, "Efficient pattern matching algorithm for memory architecture," *IEEE Trans. VLSI Syst.*, vol. 19, no. 1, pp. 1-9, Jan. 2011.

[13] Y. Yoon and S. Hwang, "Design and implementation of high-speed pattern matcher in network intrusion detection system," *J. KICS*, vol. 33, no. 11, pp. 1020-1029, Nov. 2008.

[14] C. Jasmine and T. Latha, "Finite automata in pattern matching for hardware based NIDS applications - A tutorial and survey," *Progress in Sci. Eng. Res. J.*, vol. 2, pp. 351-360. Apr. 2014.

[15] K. Pagiamtzis and A. Sheikholeslami, "Content addressable memory(CAM) circuits and architectures - A tutorial and survey," *IEEE J. Solid-state Cir.*, vol. 41, no. 3, Mar. 2006.

[16] A. Aho and M. Corasick, "Efficient string matching: An aid to bibliographic search," *Commun. ACM*, vol. 18, pp. 333-340, Jun. 1975.

[17] S. Dharmapurikar and J. Lockwood, "Fast and scalable pattern matching for content filtering," in *Proc. Symp. Architecture for Netw. Commun. Syst.*, pp. 183-192, Oct. 2005.

[18] C. Lin, "Accelerating string matching using multi-threaded algorithm on GPU," in *Proc. 2010 IEEE Global Telecommun. Conf.*, pp. 1-5, Miami, FL, Dec. 2010.

[19] C. Lin, "Memory-efficient pattern matching architectures using perfect hashing on graphic processing units" in *Proc. IEEE INFOCOM*, pp. 1978-1986, Orlando, FL, Mar. 2012.

[20] I. Sourdis and D. Pnevmatikatos, "Pre-decoded CAMs for efficient and high-speed NIDS pattern matching," in *Proc. 12th Annu. IEEE Symp. Field Programmable Custom Comput. Machines*, pp. 258-217, Napa, CA, Apr. 2004.

[21] F. Yu, R. Katz, and T. Lakshman, "Gigabit rate packet pattern matching using TCAM," in *Proc. 12th IEEE Int. Conf. Netw. Protocols*, pp. 174-173, Berlin, Germany, Oct. 2004.

[22] L. Ulf, S. Richard, and E. Warnicke, *Wireshark User's Guide*(2014), Retrieved Nov., 6, 2015, from http://www.wireshark.org

전 명 재 (Myung-Jae Jeon)

2014년 2월 : 서강대학교 전자공학과 졸업
2014년 3월~현재 : 서강대학교 전자공학과 CAD&ES 연구실 석사과정
<관심분야> MDL based Design Methodology, NIDS

황 선 영 (Sun-Young Hwang)

1976년 2월 : 서울대학교 전자공학과 학사
1978년 2월 : 한국과학원 전기 및 전자공학과 공학석사
1986년 10월 : 미국 Stanford 대학 전자공학 박사
1976년~1981년 : 삼성반도체(주)연구원, 팀장
1986년~1989년 : Stanford 대학 Center for Integrated System 연구소 책임연구원 및 Fairchild Semiconductor Palo Alto Research Center 기술자문
1989년~1992년 : 삼성전자(주) 반도체 기술 자문
1989년 3월~현재 : 서강대학교 전자공학과 교수
<관심분야> SoC 설계 및 framework 구성, CAD시스템, Com. Architecture 및 DSP System Design 등