네트워크 침입 탐지 시스템에서 고속 패턴 매칭기의 설계 및 구현

준회원 윤 여 찬*, 정회원 황 선 영*

Design and Implementation of High-Speed Pattern Matcher in Network Intrusion Detection System

Yeo-Chan Yoon* *Associate Member,* Sun-Young Hwang* *Regular Member*

요 약

본 논문은 네트워크 침입 탐지 시스템에서 고속 패턴 매칭 알고리듬과 그 구조를 제안한다. 제안된 알고리듬은 실시간 입력 패킷에서 특정 패턴을 검사하며 정확한 문자열, 문자열 값의 범위, 그리고 문자열 값의 조합 등을 검색한다. 본 연구에서는 입력 패킷과 패턴은 동시에 겹치는 문자열들을 검색하기 위해 상태 전이 그래프로 모델링하였으며 상태 전이 그래프는 구현 복잡도를 줄이기 위해 입력 임플리컨트 단위로 분할하였다. 제안된 패턴 매칭 구조는 상태 전이 그래프와 입력된 문자열을 입력으로 사용한다. 제안된 패턴 매칭기는 VHDL 언어로 모델링하여 구현하였으며, 성능 분석을 통하여 제안된 기법의 적절성을 검증하였다.

Key Words : NIDS, Pattern Matching, STG, RAM, FPGA

ABSTRACT

This paper proposes an high speed pattern matching algorithm and its implementation. The pattern matcher is used to check patterns from realtime input packet. The proposed algorithm can find exact string, range of string values, and combination of string values from input packet at high speed. Given string and rule set are modelled as a state transition graph which can find overlapped strings simultaneously, and the state transition graph is partitioned according to input implicants to reduce implementation complexity. The pattern matcher scheme uses the transformed state transition graph and input packet as an input. The pattern matcher was modelled and implemented in VHDL language. Experimental results show the proprieties of the proposed approach.

Ⅰ. 서 론

최근 인터넷 사용자의 수가 증가하고, 인터넷의 용량이 증대됨에 따라 네트워크를 통한 해킹 시도가 급증하고 있으며 그로 인한 피해 역시 날로 심각해지고 있다. 그 결과 네트워크 보안의 중요성이 점차 부각되고 있으며, 네트워크를 통한 해킹 시도를 탐지하고 그에 대응하기 위한 네트워크 침입 탐지 시스템(Network Intrusion Detection System)에 대한 연구가 매우 활발히 진행되고 있다.

네트워크 침입 탐지 시스템은 악의적인 공격을 감지해내고, 인터넷 시스템을 보호하기 위해 사용된

※ 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 육성지원사업의 연구결과로 수행되었으며 IDEC에서 제공한 CAD tool을 이용해 simulation을 수행 하였습니다.

* 서강대학교 전자공학과 CAD & ES. 연구실 (ducks@eecad.sogang.ac.kr)

논문번호 : KICS2008-09-383, 접수일자 : 2008년 9월 2일, 최종논문접수일자 : 2008년 10월 14일

다. 네트워크 침입 시도나 공격은 패턴이나 서브패턴의 조합으로 표현될 수 있으므로, 패턴 매칭이 네트워크 침입 탐지 시스템에서의 가장 중요한 이슈이며, 사용하는 패턴 매칭 알고리듬에 따라 네트워크 침입 탐지 시스템의 성능이 결정된다[1]. 현재 네트워크 침입 탐지 시스템의 패턴 데이터베이스에는 수천개의 패턴들이 등록되어 있으며, 패턴의 수는 더욱 늘어날 전망이므로, 패턴 매칭에서의 병목 현상이 더욱 심해질 것으로 예상된다. 그러나 Snort나 OSSEC와 같은 침입 탐지 시스템들은 대부분 소프트웨어 기반의 패턴 매칭만을 지원하기 때문에 네트워크의 속도가 빨라지면 모든 패킷을 검사하지 못하는 현상이 발생하게 된다. 따라서 소프트웨어 기반 침입 탐지 시스템의 성능 향상을 위한 연구로 패턴/스트링 매칭 알고리듬의 개선, 다중 컴퓨터를 사용한 Load Balancing, 트래픽 센서 앞단에 Splitter를 사용하는 방법들이 연구되고 있다[2][3]. 패턴 매칭시 소모되는 시간을 줄이기 위해 하드웨어를 이용한 패턴 매칭 시스템에 대한 연구가 진행되었다. 하드웨어 기반 침입 탐지 시스템의 성능 향상을 위해 Brute force, Deterministic finite automata (DFA), Non-deterministic finite automate (NFA) 방법들이 연구되었다[4][5][6].

패턴 매칭에서의 성능 향상을 위해 하드웨어를 기반으로 한 침입 탐지 시스템도 많은 연구가 진행되었다. 하드웨어 기반 침입 탐지 시스템은 패턴 매칭기를 FPGA로 구현하므로 패킷을 고속으로 처리할 수 있는 장점을 가진 반면에 패턴이 업데이트될 때마다 FPGA에 새로 적용해야 하므로 업데이트 시간이 오래 걸린다는 단점이 있다.

본 논문에서는 입력 패킷에서 특정한 스트링을 효율적으로 검색하는 패턴 매칭 알고리듬을 제안하고 하드웨어로 구현한다. 제안된 패턴 매칭기는 입력된 패턴을 FSM으로 변환 후 RAM에 저장함으로써, 소프트웨어 기반 침입 탐지 시스템의 빠른 업데이트와 하드웨어 기반 침입 탐지 시스템의 빠른 패턴 매칭 속도 모두를 추구할 수 있다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 관련 연구를 통해 기존의 패턴 매칭기에 대해 분석하고, Ⅲ장에서는 제안된 알고리듬에 대해 설명한다. Ⅳ장에서는 제안된 패턴 매칭기를 설계하고, Ⅴ장에서는 구현된 패턴 매칭기의 성능을 분석한다. 마지막으로 Ⅵ장에서는 결론을 제시한다.

Ⅱ. 관련 연구

네트워크 침입 탐지 시스템에서 패턴 매칭을 위해 트리구조, 트라이(trie) 구조, 해시 구조, 해시와 트리를 조합한 구조, 상태천이 그래프(state transition graph)를 이용한 구조 등이 사용된다[7][8][9]. 네트워크 시스템에서 패턴 매칭을 하기 위해 필드 포지션(field position)이 정해진 필드 값의 추출, 필드 포지션에 상관없는 필드 값의 추출, 필드 값의 비교, 필드 값의 범위 비교, 필드의 일부 값 생략 후 비교, 다수 조건들이 조합된 비교 방법이 주로 사용된다. 하드웨어로 구현하는 패턴 매칭기의 경우 많은 설계 제약조건이 있어 주로 프로세서로 패턴 매칭을 수행하거나, 해시 구조, 해시와 트리를 조합한 구조, 멀티웨이 트리(multi-way tree) 구조들을 이용한 구조들이 발표되었다[10][11][12]. 또한 속도를 높이기 위하여 다수의 패턴 매칭기들을 직렬 또는 병렬로 다단 연결하여 수행할 수 있으며, 분류 자료들을 다수로 나눈 후 나누어진 분류 자료들을 이용하여 수행하는 구조들이 발표되었다[13].

2.1 트리 구조

트리 또는 트라이(trie)를 이용한 분류 및 검색 구조는 사전의 검색, 그래픽 에디터, 정지 영상 및 동영상의 압축 등에 자주 사용된다. 트라이 구조는 사전의 단어 검색에 주로 사용되며, 주어진 분류 자료를 포함하는 가장 근사한 엔트리를 찾을 수 있다는 장점이 있다. Quad-tree, Octree 등의 멀티 웨이 트리(multi-way tree)구조는 값의 비교(greater than, less than), 값의 범위 비교, 가장 근사한 분류자료의 선택 등의 기능을 수행할 수 있는 장점이 있으나, 복잡한 조합 규칙을 사용하는 분류는 수행하지 못하는 단점이 있다.

2.2 해시 구조

해시 구조는 구현이 간단하여 상용 소자에서 프로토콜 분류를 위한 복호화 과정(exact matching)에 자주 사용되는 구조이다. 해시 구조를 이용한 분류는 분류자료의 키 값을 계산하여 해시표(hash table)의 엔트리 주소에 대응하여 색인을 하는 구조로, 값의 비교(greater than, less than), 범위, 임의의 위치에서의 패턴 매칭, 최대 우도 매칭(most likely-hood matching) 등에 사용이 어려운 단점이 있다. 해시 구조는 다수의 분류자료가 하나의 키 값에 대응되는 경우가 있어, 분류자료들 사이의 충돌(hash collision)이

발생하는 경우가 있고, 분류자료와 해시표 엔트리가 동일한지를 비교하기 위하여 엔트리 내에 분류자료 또는 분류자료의 인식 정보를 포함해야 한다. 해시 충돌을 해결하는 방법으로 내부 해싱(internal hashing), 외부 해싱(external hashing), 해시와 트리 또는 해시와 트라이 구조 등이 사용되고 있다. 내부 해싱은 해시 충돌이 발생할 경우 다시 해시 함수를 이용하여 키 값을 계산함으로써 다른 해시표 주소를 할당받는 방법이다. 외부 해싱은 한 키 값에 해당하는 해시표 주소에 다수의 엔트리들을 저장하는 방법으로, 소프트웨어로 구현할 경우 주로 연결 리스트(linked list) 구조를 사용하고, 하드웨어로 구현할 경우 한 해시표 주소 당 다수의 고정된 수의 엔트리들이 저장되는 구조들을 사용한다.

상용 제품 중 램을 이용한 캐쉬 구조를 사용하는 구조 중 이와 유사한 구조들이 있으나, 한 주소에 작은 수의 엔트리들이 대응되므로(set-associative mapping 또는 direct mapping), 일반적인 fully-associative mapping을 사용하는 CAM(Content Addressable Memory)과는 다른 구조이다. Fully-associative mapping 을 사용하는 CAM은 메모리의 내용을 이용하여 색인을 해야 하므로, 색인 자료가 CAM 내의 모든 엔트리 tag들과 동시에 비교되어야 하는 부담이 있어 구조가 복잡하고 큰 용량을 제작하기 어렵다. 검색 속도를 높이기 위하여 해시표를 직렬 또는 병렬로 순차적으로 검색하는 다단 구조들이 발표되었다.

2.3 해시와 트리를 조합한 다단 검색 구조

검색 속도를 높이기 위하여 다양한 검색 구조들을 동시에 액세스하는 방법과 분류 자료들을 분할한 후 다수의 검색 구조들을 이용하여 순차적으로 검색하는 검색 구조들을 사용할 수 있다. 외부 해시 구조에서 다수의 해시표 엔트리들을 동시에 비교 후 삽입 및 검색을 하는 방법을 사용할 수 있다. 해시와 트리를 조합하여 분류를 하는 구조는 기존 해시 구조의 충돌현상을 해결하고, 면적을 보다 효율적으로 사용하기 위하여 제안되었다. 해시와 트리를 조합하여 분류를 하는 외부 해시 구조의 구현 예로 분류 자료를 다수의 부분으로 일부분을 이용하여 해싱한 후 나머지 분류 정보를 이용하여 트라이 구조로 색인을 하는 구조가 발표되었다.

Ⅲ. 제안된 패턴 매칭 알고리듬

본 논문에서 제안된 네트워크 침입 탐지 시스템

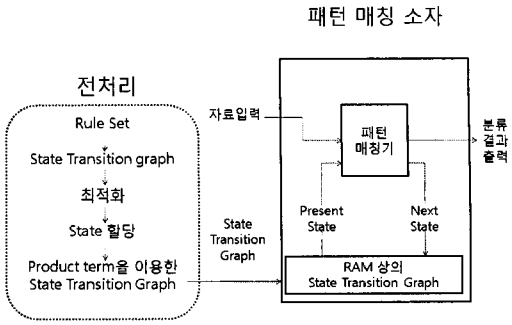


그림 1. 제안된 패턴 매칭기의 흐름도

을 위한 패턴 매칭기는 특정 패턴의 검색, 특정 패턴이 조합된 패턴을 찾는 기능이 가능하다. 그림 1은 제안된 패턴 매칭 과정의 흐름을 보인다. 검색해야할 패턴은 상태천이 그래프로 표현될 수 있으며 제안된 패턴 매칭기는 그림 1의 전처리 단계에서 생성된 상태천이 그래프를 입력받는다. 상태천이 그래프는 상태를 의미하는 노드와 노드들을 연결하는 방향성이 있는 에지로 구성되며, 현재 상태와 입력에 의해 출력과 next state가 결정된다. 만일 현재 입력된 패킷에 대한 검색을 마친 후에는 다음 입력을 입력 스트림 큐로부터 입력받는다. 생성된 상태천이 그래프는 검사과정을 거치며, 상태천이 그래프의 모든 노드는 모든 입력의 합집합이 전체 입력집합(항등함수)이 되어야 하고, 모든 입력들의 XOR 연산이 공집합이 되어야 하는 조건들을 만족해야 한다. 모든 입력의 합집합이 전체 입력집합(항등함수)이 아니면 상태천이 그래프에 지정되지 않은 입력이 있는 경우이므로 예상하지 못하는 동작을 하

Input function	Current state	Next state	Output
01000000	st0	st1	000000
01010000	st0	st4	000000
01010000	st0	st8	000000
0100----	st1	st2	000000
0101----	st2	st3	000000
0100----	st3	st0	Output1
0101----	st4	st5	000000
0100----	st5	st6	000000
0100----	st6	st7	000000
0100----	st7	st0	Output2
0101----	st8	st9	000000
0101----	st9	st10	000000
0100----	st10	st11	000000
0100----	st11	st0	Output3

그림 2. 상태천이 그래프를 표현하는 상태천이 표

```
scan( scan_start, input_stream[] )
{
    address ← initial state

    while (scan_start) {
        if ( (queue is not empty) &&
             (STG entry is last implicant)
             ‖ (request gen flag is TRUE) )
        {
            stream ← request to
                        input stream queue
        }
        request_flag ← FALSE
        last_implicant_flag ← FALSE

        if (stream input ∈ implicant)
        {
            address ← next state
            request gen flag ← TRUE
        }
        else if (STG entry is last implicant)
        {
            address ← initial state
            request gen flag ← TRUE
        }
        else
        {
            address++
            if (address == initial state)
                    request gen flag ← TRUE
            else        request gen flag ←
FALSE
        }
    }
}
```

그림 3. 제안된 패턴 매칭 알고리듬의 유사코드

는 경우가 존재하며, 모든 입력들의 XOR 연산이 공집합이 아닌 경우 한 입력에 대해 해당하는 상태천이가 여러 경우가 존재하는 경우이므로 유효하지 않은 상태천이 그래프이다.

그림 2는 특정 패턴을 검색하는 상태천이 그래프의 예로 "etri", "switch", "SoC"를 검색하는 상태천이 그래프를 보인다.

제안된 패턴 매칭 알고리듬은 입력 스트링을 이용하여 검색 상태를 운행(traversal)한다. 상태천이 표의 엔트리는 "Input function", "Current state", "Next state", "Output" 으로 구성된다.

현재 상태는 주소로 표현하였으며, 주소의 초기 값은 initial state 값을 가진다. 초기 상태의 상태천이 엔트리를 읽은 후 입력 패킷과 엔트리의 입력 조건 함수와 비교한다. 입력 조건 함수가 입력된 패킷을 포함하는 경우(implication) 엔트리의 출력값을 출력한 후 엔트리의 "Next state"로 천이하며 상태천이가 있을 때마다 다음 패킷을 다시 입력받아야 할지가 결정된다.

실제 입력된 패킷에는 다수의 패턴이 중복되어 존재할 수 있으며, 검색하는 다수의 패턴이 서로 중복되는 경우는 상태천이 그래프를 구성함으로써 모두 검색이 가능하고, 상태천이 그래프의 한 노드에서 다수의 패턴이 대응될 수 있으므로, 한 상태천이 표에서 다수의 결과가 출력될 수도 있다. 검색하는 두 패턴들이 중복되는 조합의 예를 그림 4에 보이며, 그림 5는 두 패턴의 중복되는 조합의 경우를 보인다.

그림 4와 그림 5의 검색하는 스트링들이 서로 겹치는 경우는 다음과 같이 동시에 검색이 가능하다. 검색 스트링의 시작이 동일한 한 스트링이 다른 스트링에 포함되는 경우에 두 스트링 "가나다" 와 "가나다라마"를 동시에 검색하는 예를 그림 6(a)에 보인다. 두 검색 스트링의 시작이 동일하지 않지만 한 스트링이 다른 스트링에 포함되는 경우에 두 스트링을 검색하는 예를 그림 6(b)에 보인다.

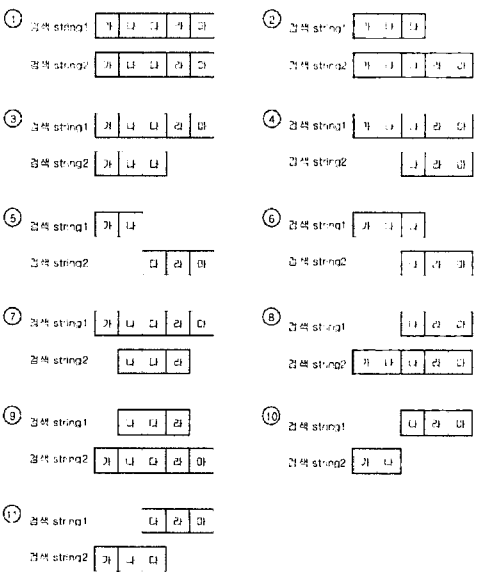
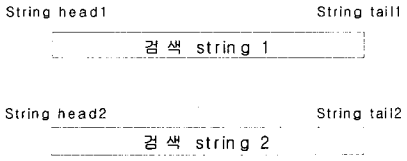


그림 4. 검색하는 스트링들이 중복되는 조합의 예



(a)

경우	Head1과 head2의 교	Tail1 과 tail2의 비교	중복 여부	설명
1	=	=	겹치는 경우	동일한 스트링
2	=	<	겹치는 경우	
3	=	>	겹치는 경우	
4	<	=	겹치는 경우	
5	<	<	겹치지 않는 경우	
6	<	<	겹치는 경우	
7	<	>	겹치는 경우	
8	>	=	겹치는 경우	
9	>	<	겹치는 경우	
10	>	>	겹치지 않는 경우	
11	>	>	겹치는 경우	

(b)

그림 5. 검색하는 스트링들이 중복되는 조합의 경우
(a) 입력되는 두 스트링, (b) 입력되는 두 스트링의 중복되는 조합의 경우

Ⅳ. 제안된 패턴 매칭 회로의 구조

제안된 패턴 매칭 알고리듬을 구현한 회로는 상태천이 표를 입력받아 입력된 패킷을 검색한다. 제안된 패턴 매칭기는 1 G/s 이상 고속의 네트워크 환경에 대해 고속으로 패턴 매칭이 가능하며, 구현 복잡도를 줄이기 위하여 분류 규칙, 검색 스트링 등의 분류 정보가 실시간으로 변경되지 않는다고 가정하였다. 상태천이 그래프를 소자에 구현할 경우 패턴 정보들을 현장에서 업데이트 가능해야 하므로 램에 저장하여 관리한다. 업데이트 된 패턴은 상태천이 그래프의 형태로 램에 저장되므로 패턴이 업데이트 될 때마다 FPGA를 새로 구성해야 하는 기존의 H/W 기반 패턴 매칭기에 비해 업데이트시 걸리는 시간이 적다. 패턴 매칭기 회로는 램에 저장된 상태천이 그래프의 엔트리들을 읽어 이용하여 검색한다. 고속 실시간 패턴 매칭기는 임의의 패턴을 램에 관리하면서 빠른 시간에 함수연산을 수행하기 위하여 많은 설계 제약조건들이 있다.

제안된 패턴 매칭기의 상태천이 그래프 실행 회로 구조를 그림 7(a)에 보였으며, 그림 7(b)에 한 상태천이 그래프 임플리컨트 엔트리를 수행하는 유한상태기계 회로를 보였다. 제안된 상태천이 구조는

검색 스트링1	검색 스트링2
가	가
나	나
다	다
	라
	마

->

입력	Current state	Next state	출력
가	state1	state2	
나	state2	state3	
다	state3	state4	"가나다" control 출력
라	state4	state5	
마	state5	state0	"가나다라마" control 출력

(a)

검색 스트링1	검색 스트링2
가	
나	나
다	다
	라
	마

->

입력	Current state	Next state	출력
가	state1	state2	
나	state1	state3	
나	state2	state3	
다	state3	state4	"가나다" control 출력
라	state4	state5	
마	state5	state0	"나다라마" control 출력

(b)

그림 6. 검색하는 스트링 들이 서로 겹치는 경우의 검색 예
(a) 시작이 동일한 한 스트링 이 다른 스트링 에 포함되는 경우, 두 스트링 "가나다" 와 "가나다라마"를 동시에 검색하는 예, (b) 두 검색 스트링 의 시작이 동일하지 않지만 한 스트링이 다른 스트링 에 포함되는 경우에 두 스트링 을 검색하는 예

천이에 다수의 클럭이 필요하며 한 상태의 입력과 입력 스트림들의 임플리컨트 수에 영향을 받는다. 한 패킷 입력에 대한 실행 시간(latency)은 가변적이므로, 패턴 매칭기에서 상태천이 표를 실행 도중 패킷 입력을 받을 것인지 말 것인가를 경우에 따라 결정하여 입력을 받는다. 패턴 매칭기의 전단에 입력된 패킷을 저장하는 큐가 있으며, 큐는 패턴 매칭기로부터 request 신호를 받으면 1 클럭 내에 패킷을 패턴 매칭기로 출력한다. 상태천이가 발생한 경우, current state의 마지막 임플리컨트를 읽은 경우, 그림 7(b) FSM의 ST1에서 입력을 기다리는 경우 패턴 매칭기는 request를 생성하여 큐로 전송한다.

한 상태천이 그래프의 엔트리에 해당하는 STG 임플리컨트 엔트리들은 순차적으로 실행되며, 한

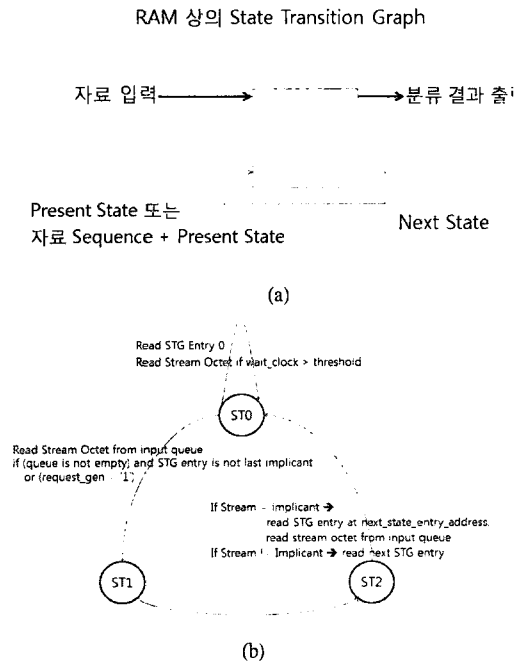


그림 7. 제안된 패턴 매칭기의 회로 구조
(a) 제안된 패턴 매칭기의 STG 실행 구조, (b) 한 STG 임플리컨트 엔트리를 수행하는 유한상태기계 회로

STG 엔트리 내의 입력 임플리컨트를 비교하는 과정 중에는 패킷 입력을 받지 않는다. 입력된 패킷에 대해 STG의 한 엔트리의 모든 STG 임플리컨트 엔트리를 비교한 후 해당되는 패턴이 없는 경우는 초기 상태로 천이하면서 다음 패킷을 입력 받는다. 그림 8에 패턴 매칭을 위하여 램 상에 구성된 STG 구조의 예를 보인다. STG의 모든 입력의 합집합이 전체 입력집합(항등함수)이 되어야 하지만, 지정되지 않는 입력 조건들에 대한 임플리컨트들을 지정하고 비교하는 것은 실행시간의 부담이 크므로, 필요한 입력 임플리컨트 조건들을 표현하고 그 외의 입력 조건들인 경우 초기 상태로 천이하도록 설계하였다.

그림 9는 RAM 상에 구성된 initial state로의 천이를 생략한 개선된 STG 구조 의 예를 보인다. 입력된 패킷이 STG 임플리컨트 엔트리에 해당되면 next state STG 엔트리의 주소로 천이하면서 다음 패킷을 읽는다. 모든 STG 임플리컨트 엔트리의 천이 과정 중 output flag가 유효하면 output control 신호를 출력한다.

제안된 패턴 매칭기의 상태천이 표는 출력 함수와 next state를 표현하기 위하여 입력변수(input variable)와 입력 시퀀스 변수(input sequence variable)들로 표현하며 패턴 매칭 소자에서 입력받는

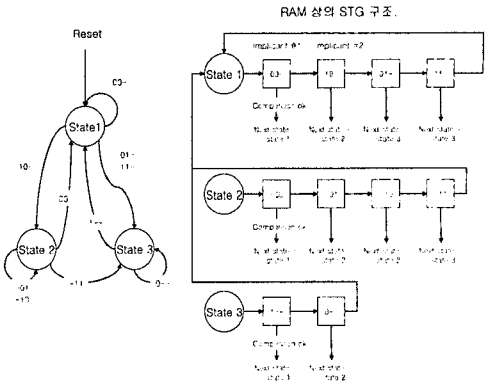


그림 8. 패턴 매칭을 위하여 RAM상에 구성된 STG 구조의 예

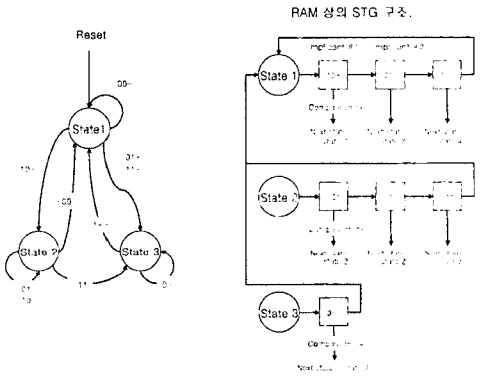


그림 9. RAM상에 구성된 STG 구조의 예 (initial state로의 천이 생략)

상태천이 표의 예를 그림 10에 보였다. 상태천이 그래프는 현재 상태와 입력(input function)에 의해 출력과 다음 상태가 결정된다. 패턴 매칭기에서 함수의 기능은 진리표, SOPs, POSs, 임플리컨트, BDD(Binary Decision Diagram) 등 다양한 방법으로 표현이 가능하다. 두 함수의 연산 과정은 효율적인 BDD의 경우 O(# BDD nodes)로 패턴 매칭기의 구현에 다수의 시간과 클럭이 필요하다 [14][15][16].

제안된 패턴 매칭기는 회로의 구현 복잡도를 줄이기 위하여 함수를 임플리컨트로 표현하고, 상태천이 표의 엔트리를 다시 함수의 임플리컨트 단위로 분류하였다. 임플리컨트는 입력된 패킷과 쉽게 연산이 가능하므로 회로의 구현 복잡도가 많이 개선되는 효과가 있다. 함수를 product term 또는 다수의 임플리컨트로 표현한 예를 그림 11에 보인다. 함수의 표현에 사용되는 변수의 값은 0, 1, 또는 don't-care 값을 가질 수 있으며, 다수의 변수의 product로 구성되는 임플리컨트는 변수들의 값과 don't-care 필터로 표현이

Input Product Term (value, filter)	Input Sequence (value)	Flag	Current state	Next state	Output
1(0100, 0101)	0000	0	st0	st1	000000
2(0101, 0011)	0000	0	st0	st4	000000
3(0101, 0011)	0000	1	st0	st8	000000
4(0100, 0101)	----	1	st1	st2	000000
5(0101, 0010)	----	1	st2	st3	000000
6(0100, 1001)	----	1	st3	st0	Output 1
7(0101, 0111)	----	1	st4	st5	000000
8(0100, 1001)	----	1	st5	st6	000000
9(0100, 0010)	----	1	st6	st7	000000
10(0100 , 0100)	----	1	st7	st0	Output 2
11(0101 , 0011)	----	1	st8	st9	000000
12(0101 , 0011)	----	1	st9	st10	000000
13(0100 , 1111)	----	1	st10	st11	000000
14(0100 , 0011)	----	1	st11	st0	Output 3

그림 10. 패턴 매칭 소자에서 입력받는 상태천이표의 예

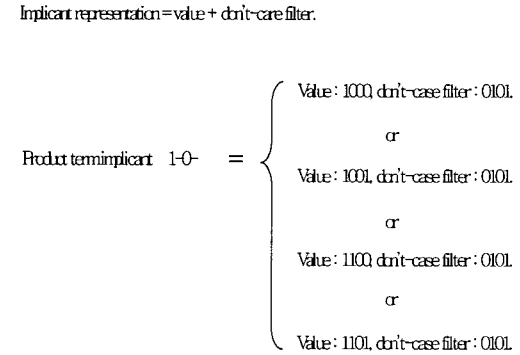


그림 11. 제안된 패턴 매칭 회로의 임플리컨트 표현

가능하다. 하나의 임플리컨트는 다수의 입력 큐브 또는 벡터들을 동시에 표현할 수 있다.

제안된 패턴 매칭기는 입력되는 스트림 값이 상태천이 표의 입력 임플리컨트 값을 비교하여 상태천이를 수행하며, 입력된 값이 입력 임플리컨트의 부분집합인지 여부에 대한 검사는 다음과 같이 계산할 수 있다. 입력된 패킷의 한 변수가 이 STG 임플리컨트의 한 변수의 부분 집합인지의 검사는 식 (1)과 표 1과 같이 계산이 가능하다. 식 (1)에서 변수 임플리케이션(variable implication) 값이 1인 경우 입력이 임플리컨트의 부분 집합인 것을 의미한다. 수식에서 + 는 비트 단위 OR연산, · 는 비트 단위 AND 연산을 뜻한다.

표 1. RAM에 저장된 STG 엔트리의 필드

STG entry field		설명
Entry valid flag		엔트리의 유효성('1'인 경우 유효한 엔트리)
입력	Implicant value	STG 입력 implicant value
	Implicant filter	STG 입력 implicant don't-care filter
	Flag(last implicant)	Input function의 last implicant 인지를 표현하는 flag (1 인 경우 유효한 input) 입력을 받을 것인지를 결정하는 control 정보
입력 sequence	Sequence value	검색 스트링의 입력 순서번호가 있는 경우, 입력의 순서번호를 표현하는 implicant value
	Sequence flag	검색 스트링 의 입력 순서번호가 있는 경우, 입력순서번호 의 don't-care filter
Current state		현재 state를 표현하는 큐브
Next state		다음 state를 표현하는 큐브
출력	Valid flag	입력이 STG 입력에 포함되는 경우 출력 신호의 유효정보 (1 인 경우 유효한 input)
	Control signals	입력이 STG 입력에 포함되는 경우 출력신호 또는 control signal set 의 link

$$\begin{aligned} Variable\ implication &= implicant\ value \cdot octet\ variable \\ &+ \overline{implicatn\ value} \cdot \overline{octet\ variable} \\ &+ implicant\ filter \end{aligned} \quad (1)$$

임플리컨트의 변수 수를 n, 식 1의 변수 v에 대한 임플리케이션을 $implication_v$으로 표현할 경우, 다수의 변수들을 포함하는 입력 벡터는 수식 2와 같이 동시에 병렬로 계산이 가능하다. 식 2에서 product term 임플리케이션 값이 1인 경우 입력 벡터가 STG 입력 임플리컨트의 부분 집합인 것을 의미한다.

$$\begin{aligned} &Product\ term\ implication \\ &= implication_{v1} \cdot implication_{v2} \cdot \cdots \cdot implication_{vn} \\ &= implicant\ value\ vector \cdot octet\ vector \\ &+ \overline{implicant\ value\ vector} \cdot \overline{octet\ vector} \\ &+ implicant\ filter\ vector \end{aligned} \quad (2)$$

표 2. 패턴 매칭기의 처리율 (상태천이 단위) 예

Link speed	Max. throughput	Min 자료 throughput	자료 stream 경로	Clock cycle
1 Gbits/s	3,000 K frame/s	333 ns (or 512 ns)	8 bits @ 125, 128 Mhz	8 ns, 8.19 ns
			16 bits @ 62.5, 64 Mhz	16.38 ns, 16 ns
			32 bits @ 31.25 Mhz	32.77 ns
2.5 Gbits/s	6,000 K frame/s	167 ns	20 bits @ 125, 128 Mhz	8 ns, 8.19 ns
			32 bits @ 100 Mhz	10.24 ns
			40 bits @ 62.5, 64 Mhz	16.38 ns, 16 ns
10 Gbits/s	26,000 K frame/s	10 ns	64 bits @ 180 Mhz	5.69 ns
			80 bits @ 125, 128 Mhz	8 ns, 8.19 ns
			128 @ 80 Mhz	12.8 ns

제안된 패턴 매칭기는 그림 11의 각 입력 임플리컨트 들로 다시 분류된 상태천이 표를 램에 관리하며, 표1에 각 RAM 엔트리의 필드를 보인다.

자료의 전송 용량에 따라 최소 길이 자료의 최대 처리율(throughput), 최소 길이자료의 프로세싱을 위한 처리율, 자료의 데이터 버스, 패턴 매칭기의 처리율(상태천이 단위)의 예를 표 2에 보인다. 제안된 패턴 매칭기는 임플리컨트를 입력받아 상태천이를 수행하는데 3 클럭이 필요하며, 각 상태의 임플리컨트의 수에 의해 영향을 받는다. 하나의 임플리컨트는 다수의 패턴을 표현할 수 있다. 자료 입력을 검색하기 위하여 상태천이의 평균 입력 임플리컨트 수가 5 이하인 경우, 128 bits @ 125Mhz 또는 64 bits @ 240Mhz로 사용하여 1Gbits/s의 입력 스트림을 검색할 수 있다. 한 상태천이 엔트리에서 5개의 입력 임플리컨트를 동시에 비교하는 경우, 24 bits @ 125 Mhz, 48 bits @ 62.5 Mhz, 또는 96 bits @ 31.25 Mhz로 1Gb/s의 입력 스트림을 검색할 수 있다. 상태천이 엔트리 각 상태의 평균 사용하는 임플리컨트 수의 입력을 동시에 비교하는 경우, 한 번의 상태천이에 3 클럭이 소요되므로 아래 표 2에서 제시된 패턴 매칭기의 처리율/3 으로 예측할 수 있다. 제안된 패턴 매칭기는 용량 확장을 위하여 다수의 검색 모듈을 직렬 또는 병렬로 연결하여 검색이 가능하다. 다수의 검색 모듈을 병렬로

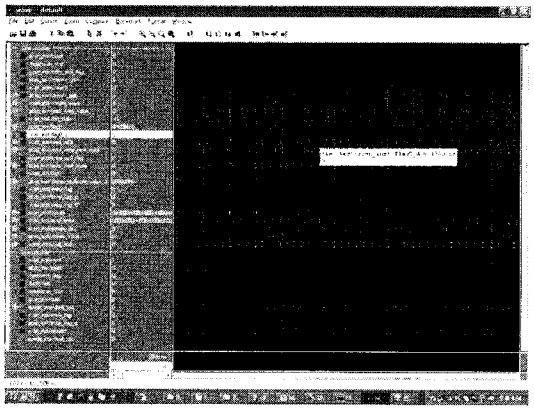


그림 12. 구현된 패턴 매칭기의 시뮬레이션 실험 결과

구성하는 경우, STG를 분할하여 동일한 입력 패킷에 입력받아 동시에 검색할 수 있다.

패턴 정보를 RAM에 저장하므로 검색해야할 패턴이 늘어날수록 RAM 용량이 늘어나게 되지만, STG 엔트리 수에 따른 RAM 용량 예측 결과, STG 엔트리 수가 8192개일 경우 최대 3MB의 RAM을 사용하므로 오버헤드가 크지 않다.

V. 실험 결과

제안한 패턴 매칭기는 VHDL 언어로 구현하였으며, 구현된 매칭기는 Modelsim 시뮬레이터로 기능을 검증하였다. 본문에서 사용한 그림 3과 그림 11의 예제 상태천이 표를 입력받아 패턴 매칭을 수행하는 실험 결과를 그림 12에 보였다. 패킷을 입력받아 패턴을 검색한 결과는 그림 12에서 high 값으로 전이된 flag_d신호로 확인할 수 있다. 표 3은 시뮬레이션 결과 패킷이 입력되었을 때 패턴 매칭을 위해 걸리는 클럭이 기존 해시 구조 패턴 매칭기와 비교하여 평균 90.6% 감소하였음을 보인다. 실험을 통하여 제안된 구조의 패턴 매칭기가 고속으로 입력 패킷을 검색할 수 있음을 보였다.

표 3. 해시 구조 패턴 매칭기와 제안된 검색기의 클럭 사이클 비교

입력 라인 수	해시 구조 패턴 매칭기	제안된 구조	감소율
128	4,226,382	396,499	90.6 %
256	8,452,735	792,959	90.6 %
512	16,905,561	1,584,070	90.6 %

Ⅵ. 결 론

본 논문에서는 패턴을 입력받아 입력 패킷과 비교하는 알고리듬을 제안하고, 고속으로 패턴 매칭을 할 수 있는 회로를 설계하고 구현하였다. 현재 상용 패턴 매칭기는 대부분 소프트웨어 기반으로 구현되어 있으며, 하드웨어로 구현된 패턴 매칭기는 속도가 빠르다는 장점이 있으나 패턴의 업데이트가 어렵다는 단점이 있다. 제안된 패턴 매칭기는 하드웨어로 구현되어 동작 속도가 빠르며 패턴 정보를 RAM에 저장하므로 업데이트가 빠르다는 장점이 있으며, 패턴 저장을 위해 요구되는 RAM 용량은 최대 3MB로 기존의 하드웨어 기반 패턴 매칭기에서의 요구량보다 적다. 구현 복잡도를 줄이기 위하여 비교할 패턴, 입력 패킷 등의 정보는 동작 중 변경되지 않는다고 가정하였으며, 상태천이 그래프를 임플리컨트로 다시 분할하여 수행하도록 설계하였다. 실험 결과를 통하여 제안된 패턴 매칭기가 고속으로 동작할 수 있음을 보였으며, 구조가 간단하고 하드웨어 리소스의 사용량이 작아 네트워크 침입 탐지 시스템에서 패턴 매칭을 위하여 효율적으로 사용될 수 있을 것으로 기대된다.

참 고 문 헌

[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] N. Desai, "Increasing Performance in High Speed NIDS: A look at Snort's Internals", Feb. 2002.

[3] I. Charitakis, K. Anagnostakis, and E. Markatos, "An Active Traffic Splitter Architecture for Intrusion Detection," in Proc. IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Orlando, Florida, pp.238-241, Oct. 2003.

[4] Y. Cho, W. Mangione-Smith, "Deep Packet Filter with Dedicated Logic and Read Only Memories," in Proc. IEEE Symposium on Field-Programmable Custom Computing Machines, pp.125-134, Apr. 2004.

[5] M. Aldwairi, T. Conte, and P. Franzon, "Configurable String Matching Hardware for Speeding up Intrusion Detection," ACM SIGARCH Computer Architecture News, Vol.33, No.1, pp.99-107, Jan. 2005.

[6] R. Sidhu and V. Prasanna, "Fast Regular Expression Matching Using FPGAs," in Proc. IEEE Symposium on Field-Programmable Custom Machines, Rohnert Park, CA, pp.227-238, May 2001.

[7] C. Hoffman and M. O'Donnell, "Pattern Matching in Trees," Journal of the ACM, Vol.29, No.1, pp.68-95, Jan. 1982.

[8] R. Karp and M. Rabin, "Efficient Randomized Pattern-Matching Algorithms," IBM Journal of Research and Development, Vol.31, No.2, pp.249-260, Mar. 1987.

[9] D. Pao, C. Liu, A. Wu, L. Yeung, and K. Chan, "Efficient Hardware Architecture for Fast IP Address Lookup," in Proc. IEEE Infocom, New York, NY, Vol.2, pp.555-561, Jun. 2002.

[10] S. Iyer, R. R. Kompella, and A. Shelat, "ClassiPI: An Architecture for Fast and Flexible Packet Classification," IEEE Network Magazine, pp.24-32, Apr. 2001.

[11] A. Feldmann and S. Muthukrishnan, "Tradeoffs for Packet Classification," AT&T Technical Report, 1999.

[12] A. Parakash and A. Aziz, "OC-3072 Packet Classification Using BDDs and Pipelined SRAMs," in Proc. Hot Interconnects, Stanford, CA, pp.15-20, Aug. 2001.

[13] J. Park and I. Jang, "Parallelisation of Trie-based Longest Prefix Matching for Fast IP Address Lookups," Electronics Letters, Vol.38, No.25. pp.1757-1759, Dec. 2002.

[14] S. Brown, R. Francis, J. Rose, and Z. Vranesic, Field-Programmable Gate Arrays, Kluwer Academic Publisher, 1992.

[15] P. Ashar, S. Devadas, and A. Newton, Sequential Logic Synthesis, Kluwer Academic Publisher, 1992.

[16] G. De Micheli, Synthesis and Optimization of Digital Circuits, McGraw-Hill, 1994.

윤 여 찬 (Yeo-Chan Yoon) 준회원

2007년 2월 서강대학교 전자공학과 졸업

2007년 3월~현재 서강대학교전자공학과 대학원 CAD & Embedded Systems 연구실 석사과정

<관심분야> 테스트 용이화 설계, NIDS

황 선 영 (Sun-Young Hwang) 정회원

1976년 2월 서울대학교 전자 공학과 졸업

1978년 2월 한국 과학원 전기 및 전자공학과 공학석사

1986년 10월 미국 Stanford 대학 전자공학 박사

1976년~1981년 삼성반도체 주식회사 연구원, 팀장

1986년~1989년 Stanford 대학 Center for Integrated System 연구소 책임 연구원Fairchild Semiconductor Palo Alto Research Center 기술 자문

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