韓國컴퓨터情報學會 論文誌
第16卷 第5號, 2011. 05.

2011-16-5-1-2

심층 패킷검사를 위한 정규표현식 패턴매칭 하드웨어 구조

윤 상 균 *, 이 규 희 *

A Hardware Architecture of Regular Expression Pattern Matching for Deep Packet Inspection

SangKyun Yun *, KyuHee Lee *

요 약

최근의 네트워크 침입탐지 시스템들은 침입패턴을 나타내는 데 정규표현식을 사용하고 있으며 빠른 심층 패킷 검사를 위해서 하드웨어 기반의 패턴매칭이 필요하다. 하드웨어 기반 정규표현식 패턴매칭에 대한 많은 연구가 이루어졌으나 {10}과 같은 제한반복 연산자에 대한 구현은 제약이 있었다. 본 논문에서는 일반적인 정규표현식 서브패턴에 대한 제한반복을 더 낮은 하드웨어 복잡도로 구현할 수 있는 제한반복 블록 구조를 제시하였다. 제안된 제한반복 블록은 단일 문자, 고정길이 문자 뿐 만 아니라 일반적인 정규표현식 서브패턴의 제한반복 구현도 가능하다. 제안된 제한반복 블록 구조는 모든 제한반복을 펼치지 않고 구현할 수 있도록 하여 정규표현식 패턴매칭 하드웨어를 더 효율적으로 구현할 수 있도록 하였다.

▸ Keyword : 패킷 검사, 침입 탐지, 패턴 매칭 하드웨어, 정규표현식

Abstract

Network Intrusion Detection Systems use regular expression to represent malicious packets and hardware-based pattern matching is required for fast deep packet inspection. Although hardware architectures for implementing constraint repetition operators such as {10} were recently proposed, they have some limitation. In this paper, we propose hardware architecture supporting constraint repetitions of general regular expression sub-patterns with lower logic complexity. The subpatterns supported by the proposed contraint repetition architecture include general regular expression patterns as well as a single character and fixed length patterns. With the proposed building block, we can implement more efficiently regular expression pattern matching hardwares.

▸ Keyword : packet inspection, intrusion detection, pattern matching H/W, regular expression, NIDS

• 제1저자, 교신저자 : 윤상균
• 투고일 : 2011. 02. 18, 심사일 : 2011. 03. 08, 게재확정일 : 2011. 03. 15..
* 연세대학교 컴퓨터정보통신공학부(Dept. of Computer and Telecommunication Engineering, Yonsei University.)
※ 이 논문은 2008년 정부(교육과학기술부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(KRF-2008-521-D00446).

I. 서 론

최근에 네트워크를 통한 시스템에 대한 침입과 공격이 증가하고 있으며 이러한 악의적인 접근들을 탐지하기 위하여 네트워크 침입탐지 시스템 (Network Intrusion Detection System: NIDS)이 널리 사용된다. NIDS는 네트워크 패킷을 분석하여 침입 패턴을 포함한 패킷을 탐지한다. 패킷의 헤더 뿐 만 아니라 패킷 데이터까지 검사하는 것을 심층 패킷검사라고 한다. NIDS에서 시간이 가장 많이 소요되는 작업은 심층 패킷검사를 수행할 때에 네트워크를 통해 입력되는 모든 패킷들을 침입이 의심되는 패턴 규칙 집합의 문자열 패턴과 비교하는 다중 패턴 매칭 작업이다.

전통적인 NIDS들은 다중 패턴 매칭 알고리즘을 소프트웨어로 구현한 방법을 많이 사용하였다. 소프트웨어 기반 NIDS는 패턴 규칙을 쉽게 업데이트 할 수 있는 유연성을 제공하지만 고속 네트워크 환경에서는 패킷에 대한 패턴 매칭을 실시간으로 수행할 수 없게 되었다. 이를 해결하기 위하여 여러 연구자들에 의해서 하드웨어 기반 패턴 매칭 방법들이 제안되었으며 침입 패턴 규칙이 지속적으로 갱신되기 때문에 패턴매칭 하드웨어는 메모리 기반으로 구성되거나 재구성이 가능한 FPGA (Field Programmable Gate Arrays)를 사용하여 구성되었다.

Snort[1], Bro[2], Bleeding Edges[3]와 같은 NIDS들은 패턴 규칙을 나타내는 데에 “xyz”와 같은 정적 문자열 패턴 뿐 만 아니라 “ab*”나 “[^a]b+c”와 같은 정규표현식도 함께 사용하고 있다. 그러므로 패턴 매칭 하드웨어는 정규표현식 패턴에 대한 패턴 매칭을 지원하여야 한다. 여러 연구자들에 의해서 패턴 매칭 하드웨어의 구조에 대한 연구가 수행되었으며, 이들 중 초기의 연구들은[4-7] 고정 패턴 매칭만 다루었으며 나머지 연구에서는[8-14] 정규표현식 패턴 매칭을 다루었다.

정규표현식에 대한 패턴 매칭에서는 특별한 의미를 갖는 메타문자들을 사용하므로 고정 문자열 매칭과 차이가 있으며 하나의 정규표현식이 여러 개의 문자열을 나타낼 수 있다. 정규표현식은 패턴의 반복을 나타내기 위해서 *, + 연산자 뿐 만 아니라 지정된 범위의 횟수의 반복을 나타내는 {200}, {10,20}, {10,}와 같은 제한 반복 연산자를 제공한다. 침입 패턴 규칙을 나타내는 데 *와 +가 많이 사용되기는 하지만 최근에는 제한 반복 연산자의 사용 빈도가 증가하고 있다.

그렇지만 대부분의 이전 연구들에서는[8-11] *나 +와 같은 반복 연산자들은 대부분 구현을 하였으나 제한반복 연산자는 별도로 구현하지 않고 반복을 펼쳐서 구현하였다. 예를 들어 a{5}는 aaaaa와 같이 펼쳐서 구현한다. 이 때문에 하드웨어는 반복횟수에 비례하여 복잡해진다. 그러므로 정규표현식 패턴매칭 하드웨어를 효율적으로 구현하기 위해서는 제한반복 연산자를 펼치지 않고 구현하는 것이 필요하다.

Bispo 등은 정규표현식의 제한반복 연산자를 처음으로 반복을 펼치지 않고 구현하였다. 그런데 Bispo 등은 초기 연구[12,13]에서는 a{10}과 같은 단일문자에 대한 제한 반복만을 구현하였으며, 후속 연구에서[14] (ab){10}과 같은 고정길이 서브패턴에 대한 제한반복을 구현하였다. 그렇지만 (ab*c){10}과 같은 일반적인 정규표현식 패턴에 대한 제한반복은 구현하지 못하였으며 제한반복 블록의 하드웨어 복잡도도 여전히 반복횟수에 비례하였다. 그리고 Bispo 등은 제한반복의 구현에서 중첩 매칭의 처리에 대한 어려움을 언급하고 향후 과제로 제시한 바가 있다[14].

본 논문에서는 정규표현식 패턴매칭 하드웨어 설계에 대한 기존 연구의 제한점을 해결하기 위해서 일반적인 정규표현식 패턴에 대한 제한반복을 지원하며 더 낮은 하드웨어 복잡도를 갖는 제한반복 블록 구조를 제시하고, 중첩 매칭과 관련하여 발생하는 문제점에 대한 해결하는 방안 등을 제시한다. 이를 통하여 정규표현식 패턴매칭을 효율적이고 정확하게 수행할 수 있는 하드웨어를 구현할 수 있도록 한다.

이 논문의 구성은 다음과 같다. 2장에서 관련 연구를 소개하고, 3장에서 제한 반복 블록의 설계를 중심으로 한 정규표현식 패턴 매칭 하드웨어 구조를 제시한다. 4장에서 제시한 하드웨어 구조에 대한 분석과 평가를 수행하며, 5장에서 결론을 맺는다.

II. 관련 연구

1. 정규표현식

정규표현식은 다양한 문자열 패턴을 나타내기 위해서 널리 사용되고 있으며 문자들과 특별한 의미를 갖는 메타문자들로 구성된다. 표 1은 본 연구의 패턴매칭 하드웨어가 지원하는 정규표현식의 메타문자들과 그 의미를 보여준다. 예를 들어서 a*는 a가 0번 이상 반복됨을 뜻하며 a|bc는 패턴 a 또는 bc를 표시한다. a{3}은 a가 정확히(Exactly) 3번 반복됨을, a{3,}은 a가 3번 이상(AtLeast) 반복됨을, 그리고 a{2,4}는 a가 2번 이상 4번 이하(Between) 반복됨을 나타낸다. [abc]는 문자 a, b, c로 구성되는 문자 그룹을, [^ab]는 문

자 a, b를 제외한 모든 문자를 나타낸다. ^abc는 abc로 시작함을 나타내며, xyz$는 xyz로 끝남을 나타낸다.

표 1 제안구조가 지원하는 정규표현식의 메타문자
Table 1 Metacharacters in regular expressions supported by our approach

문자	의미	문자	의미
|	OR 연산자	.	임의의 문자
*	0번 이상 반복	*	메타문자의미제거
+	1번 이상 반복	[abc]	문자그룹
?	0 또는 1번	[a-f]	범위문자그룹
{n}	n번 반복	[^ab]	제외문자그룹
{n,}	n번 이상 반복	\xFF	16진수
{n,m}	n번이상 m번이하 반복	\012	8진수
^	처음	\d, \w, \s	특정 문자그룹
$	마지막	\n, \r,\ t	LF,CR,탭문자

2. 패턴매칭 하드웨어

정규표현식 패턴매칭을 수행하는 하드웨어 구현은 여러 연구에서 유한오토마타(Finite Automata: FA)를 기반으로 이루어졌다. 유한오토마타는 결정 유한오토마타(Deteministic FA: DFA)와 비결정 유한오토마타(Nondeterministic FA: NFA)로 구분된다. DFA는 하나의 상태만 활성화되기 때문에 소프트웨어 구현에 적합하지만 상태수가 많은 단점이 있다. 반면에 NFA는 상태 수가 적지만 주어진 시간에 여러 개의 활성화된 상태를 가질 수 있어서 소프트웨어 구현에 적합하지 않지만 하드웨어는 여러 상태가 동시에 활성화되는 것을 구현할 수 있기 때문에 하드웨어 구현에 적합하다.

Sidhu 등은[8] NFA를 이용하여 정규표현식 패턴매칭 하드웨어를 구현하였다. 단일문자, 스타(*), 결합, OR(|), 괄호 등에 대한 기본 블록을 제시하고 이 블록들을 사용하여 하드웨어로 맵핑하였으며 각 상태마다 플립플롭을 할당하여 여러 상태를 동시에 활성화시킬 수 있는 one-hot 인코딩을 사용하였다. Clark 등은[6] 입력 문자에 대한 디코더를 사용하여 입력 문자와 패턴의 문자에 대한 비교 결과를 제공하고 이 결과를 모든 패턴매칭 불록에서 공유하는 공유디코더 방법을 제시하였으며 개별적인 비교기를 사용하지 않기 때문에 하드웨어 자원 사용량이 감소하였다.

그림 1는 Sidhu 등이 제시한 기본 블록의 구조이다. 그림 1(a)는 문자 c에 대한 패턴매칭 블록의 내부 회로와 블록도이다. 여기서 m_c는 문자 c에 대한 공유디코더의 출력으로서 입력과 문자 c의 비교 결과를 나타낸다. 정규표현식 연산자 *, +, ?와 |는 모두 그림 1(c)-(f)와 같이 OR 게이트를 함께 사용하여 구현된다. 이 그림에서 R, R_1, R_2를 포함한 사각형은 해당 정규표현식의 패턴매칭을 구현한 블록을 나타낸다.

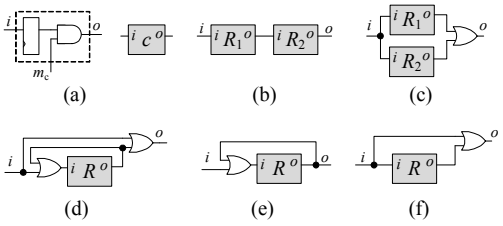


그림 1. 기본블록
Fig. 1 Basic Building Blocks
(a) 문자 c (b) R1R2 (c) R1 | R2 (d) R* (e) R+ (f) R?

Bispo 등은[12,13] 정규표현식의 제한반복 패턴매칭 하드웨어를 처음으로 제시하였다. Exactly와 Between 제한반복에 대한 블록은 Xilinx사의 FPGA에서만 제공되는 SRL16 16비트 시프트 레지스터와 카운터, 플립플롭을 사용하여 구현하였으며 AtLeast 제한반복 블록은 카운터와 플립플롭을 사용하여 구현하였다. 그림 2는 SRL16을 사용하여 구현한 Exactly 제한반복 블록에 대한 구조이다.

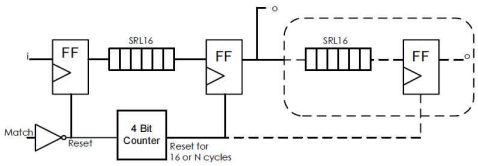


그림 2. a{N}에 대한 Exactly 제한반복 블록
Fig. 2 Exactly block for a{N}

그렇지만 Bispo 등이 제시한 구조는 단일문자에 대한 제한 반복만 구현할 수 있다. 그리고 특정 회사의 FPGA에서만 제공되는 시프트 레지스터를 사용하였기 때문에 다른 회사의 FPGA에는 적용할 수 없으며 반복횟수가 N일 때에 구현에 필요한 SRL16의 개수가 ⌈ N / 17⌉ 로 N에 비례하여 증가하는 단점이 있다.

Bispo 등은 후속 연구[14]에서 (ab){5,9}나 (ab|cd){5}와 같은 고정된 길이의 서브패턴에 대한 제한반복을 지원하는 제한반복 구조도 제시하여 그들의 이전 연구의 제한점을 개선하였다. 그렇지만 (ab*){10}와 (a|bcd){5,}와 같이 가변길이를 갖는 패턴에 대한 제한 반복은 여전히 구현할 수 없다.

Ho 등은 NFA 기반이 아닌 Bloom 필터를 기반으로 구현한 정규표현식 패턴매칭회로 PERG를 만들었고[15], 이를 확장하여 제한반복을 처리하도록 한 PERG-Rx를 만들었다[16].

Snort의 패턴 규칙에 가변길이 패턴에 대한 제한반복이 포함되어 있다. 그러므로 이러한 규칙에 대한 패턴매칭을 수행하는 하드웨어 설계를 위해서는 단일문자나 고정된 길이의

서브패턴이 아닌 일반적인 정규표현식 패턴의 제한 반복을 효율적으로 처리할 수 있는 정규표현식 패턴매칭 하드웨어 구조에 대한 연구가 필요하다.

III. 제안하는 정규표현식 패턴 매칭 하드웨어 구조

이 절에서는 정규표현식 패턴 매칭을 위한 하드웨어 구조의 설계 방법을 제시한다. 특히 기존 연구에서 불충분하게 구현되었던 제한반복 연산을 제약 없이 구현할 수 있게 하는 정규표현식 패턴매칭 하드웨어 구조를 제시한다. 제시되는 제한반복 블록은 일반적인 정규표현식에 대한 제한 반복을 구현할 수 있도록 하고, 필요한 하드웨어 자원의 크기가 반복횟수에 비례하지 않도록 하기 위해서 시프트 레지스터를 사용하지 않고 카운터 만 사용하여 설계한다. 그리고 제한반복 블록에서 중첩매칭의 수행에 문제가 없도록 하는 방안을 제시한다.

1. 기본 블록

정규표현식 패턴매칭 회로의 구현은 정규표현식의 각 요소에 대한 기본 블록들을 정의하여 정규표현식을 하드웨어로 맵핑하는 방식을 사용한다. 기본적인 연산자에 대해서는 II장의 그림 1과 같은 기존 연구에서 제안한 블록들을 그대로 사용한다. 예를 들어서 패턴 ab+c(d|e)에 대한 패턴매칭 회로는 그림 3과 같이 구현된다. 이 그림에 나타나지 않은, 문자 a, b, c, d, e에 대한 공유디코더의 출력이 각 기본 블록들에 연결된다.

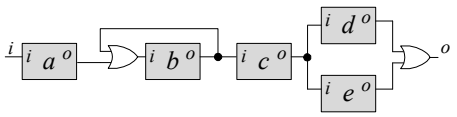


그림 3. 정규표현식 ab+c(d|e)에 대한 블록
Fig. 3 Block for a regular expression ab+c(d|e)

2. 제한반복 블록 구조

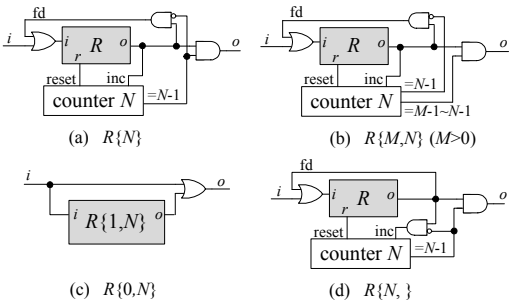
정규표현식에서는 패턴의 제한된 횟수의 반복을 나타내기 위해 다음과 같은 세 종류의 제한반복 연산자들을 사용한다.

- R{N}: (Exactly) R이 정확히 N번 반복함
- R{N, }: (AtLeast) R이 N번 이상 반복함
- R{N,M}: (Between) R이 N번 이상 M번 이하 반복함

여기서 R은 단일 문자 뿐 만 아니라 정규표현식의 패턴이 될 수도 있다. Snort 패턴 규칙에서 제한 반복 연산자들의 사용이 점점 증가하고 있다.

제한반복 블록 구조에 대한 기존 연구의 문제점을 해결하기 위하여 본 연구에서 제시하는 구조는 그림 4와 같이 서브패턴 R에 대한 패턴매칭 회로(앞으로 R블록이라고 부름)와 반복 횟수를 계수하는 동기식 카운터를 사용하여 구성된다.

카운터는 기본적으로 서브패턴 R의 패턴매칭이 성공할 때마다 증가하며, 실패하면 0으로 리셋된다. 그림 4에서 사각형으로 표시된 R블록 내의 i와 o는 R블록의 매칭 입출력을 나타내며, 출력 r은 R블록에서 반복매칭이 진행되지 않을 때에 카운터를 리셋시키기 위한 신호이다. R블록의 이 세 신호를 각각 Ri, Ro, Rr 로 나타낼 것이다. 여기에서 서브패턴 R은 일반적인 정규표현식을 제약 없이 사용할 수 있다.



(a) $R\{N\}$ (b) $R\{M,N\}$ $(M>0)$
(c) $R\{0,N\}$ (d) $R\{N,\}$

그림 4 제한반복 블록 구조
Fig. 4 Constraint Repetition Block Architecture
(a) Exactly 블록 (b),(c) Between 블록 (d) AtLeast 블록

2.1 Exactly 블록

정규표현식 R{N}에 대한 패턴매칭을 수행하는 Exactly 블록은 입력문자열이 서브패턴 R과 정확히 N번 반복하여 매칭될 때에 출력이 1이 되는 회로로서 0부터 N-1까지 계수하는 $\log_2 N$ 비트 카운터를 사용하여 그림 4(a)와 같이 구현된다.

서브패턴 R의 매칭이 성공하여 Ro가 1이 될 때마다 카운터를 증가시키고, 반복 매칭을 수행하도록 R블록의 매칭입력으로 피드백시킨다. 카운터가 N-1일 때에 Ro가 1이 되면 서브패턴 R이 N번 반복되어 매칭된 것이므로 매칭 출력을 1로 만들며, 반복매칭을 더 이상 수행하지 않도록 피드백을 차단한다.

그림 4(a)의 Exactly 블록은 카운터 증가신호 inc, 피드백 신호 fd와 매칭출력 R{N}o를 다음 식의 회로로 구현하여 이와 같은 동작을 수행하도록 한다.

inc = Ro ····· (1)

$$fd = (q < N-1) \wedge Ro = \sim(q = N-1) \wedge Ro \quad (2)$$

$$R\{N\}o = (q = N-1) \wedge Ro \quad (3)$$

이 식에서 q는 카운터 값을 나타내며 카운터는 q가 마지막 상태 N-1임을 알려주는 신호(그림에서 =N-1로 표시됨)를 제공한다. 서브패턴 R이 N번 매칭되면 카운터는 N-1에서 0으로 되돌아가지만 피드백이 되지 않으므로 서브패턴 R의 추가적인 반복에 대해서는 R블록에서 패턴매칭이 진행되지 않는다. 반복 횟수가 N번에 도달되기 전에 반복 매칭이 실패하면 R블록에서 패턴매칭이 진행되지 않기 때문에 카운터 리셋 신호가 발생하여 카운터는 0으로 초기화된다.

2.2 Between 블록

정규표현식 R{M,N}에 패턴매칭을 수행하는 Between 블록은 입력문자열이 서브패턴 R과 M번 이상 N번 이하로 반복하여 매칭될 때에 1이 되는 회로로서 그림 4(b)와 같이 구현된다. 이 회로는 다음과 같이 카운터 증가신호 inc와 피드백 신호 fd는 Exactly 블록과 같으며 매칭출력 신호만 다르다. 카운터가 M-1에서 N-1사이일 때에 Ro가 1이 되면 매칭출력을 1로 만든다.

$$inc = Ro \quad (4)$$

$$fd = (q < N-1) \wedge Ro = \sim(q = N-1) \wedge Ro \quad (5)$$

$$R\{M,N\}o = (M-1 \leq q \leq N-1) \wedge Ro \quad (6)$$

Between 블록의 카운터는 q가 마지막 상태 N-1임을 알려주는 신호와 q가 M-1이상 N-1이하임을 알려주는 신호(그림에서 =M-1~N-1로 표시됨)를 함께 제공한다.

그림 4(b)의 블록은 M이 1이상일 때에 적용가능하며 M이 0일 때에, 즉 R{0,N}에 대한 블록은 그림 4(c)와 같이 R{1,N}의 블록과 OR게이트를 사용하여 구현된다.

2.3 AtLeast 블록

정규표현식 R{N, }에 대한 패턴매칭을 수행하는 AtLeast 블록은 입력문자열이 서브패턴 R과 N번 이상 반복하여 매칭될 때에 출력이 1이 되는 회로로서, Exactly 블록과 마찬가지로 $\log_2 N$ 비트 카운터를 사용하여 그림 4(d)와 같이 구현된다.

AtLeast 블록은 서브패턴 R을 반복횟수의 제한없이 계속하여 매칭시킬 수 있어야 하므로 카운터 상태에 관계없이 서브패턴 R이 매칭될 때마다 R블록 매칭입력으로 피드백하도록 다음과 같은 피드백 신호를 갖는다.

$$fd = Ro \quad (7)$$

카운터는 반복횟수를 무한히 계수하는 것 대신에 N-1에 도달하면 더 이상 증가하지 않고 반복매칭이 실패하여 카운터가 리셋될 때까지 그대로 유지되도록 한다. 이렇게 하면 0부터 N-1까지 계수하는 카운터를 사용하여 AtLeast 블록을 구현할 수 있다. 이를 위한 카운터 증가 신호는 다음과 같다.

$$inc = (q < N-1) \wedge Ro = \sim(q = N-1) \wedge Ro \quad (8)$$

서브패턴 R과 매칭되는 반복횟수가 N번 이상인 경우에 카운터는 N-1을 유지하므로 AtLeast 블록의 매칭출력 신호는 다음과 같다. 이 신호는 Exactly 블록과 같다.

$$R\{N,\}o = (q = N-1) \wedge Ro \quad (9)$$

AtLeast 블록에서 카운터는 R블록에서 패턴매칭이 더 이상 진행되지 않을 때에 발생하는 카운터 리셋 신호에 의해서 0으로 초기화된다.

2.4 카운터 리셋 회로

R블록에서 패턴매칭이 진행되지 않으면 리셋 신호를 제공하여 카운터를 0으로 초기화하도록 한다. 그림 5는 서브패턴 abc에 대한 R블록 회로로서 리셋 신호 생성 회로가 포함되어 있다. 기본적으로 리셋신호는 그림 5(a)와 같이 R블록 내의 모든 매칭상태 A, B, C가 모두 0일 때에 1로 만든다. 모든 매칭상태가 0이면 R블록에서 반복매칭이 진행되지 않음을 의미한다.

그렇지만 b(abc){4}와 같이 제한반복 패턴 앞에 제한반복 서브패턴에 포함되는 문자열 패턴(여기서는 b)이 있는 경우에는 첫 번째 매칭상태 A는 반복 매칭이 아닌 R블록에서의 첫 번째 매칭을 포함할 수 있으므로 그림 5(b)와 같이 c 다음에 a가 입력되는 것을 확인하는 회로를 추가하여 A 대신에 이 회로의 출력 A2를 반복매칭 상태 확인용으로 사용한다.

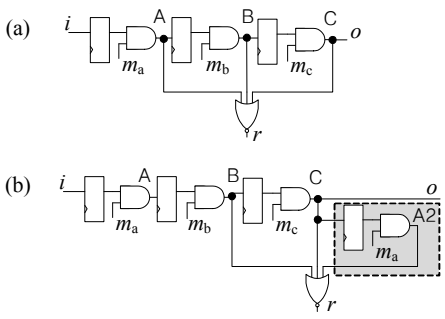


그림 5 리셋 신호를 포함한 R블록 회로
Fig. 5 R block circuit with reset signal

2.5. 반복종료 서브패턴에 대한 제한반복 블록

(abc+){5}와 같이 서브패턴이 *, +와 같은 반복연산자로 종료되는 제한반복 패턴에 대해서는 그림 4의 회로들은 R블록에서 한 번의 서브패턴 반복에 대해서 매칭출력 Ro가 여러 번 1이 될 수 있기 때문에 카운터가 여러 번 증가할 수 있는 문제점을 갖고 있다. 예를 들어서 문자열 입력 abcccabcc은 서브패턴 abc+가 두 번 반복한 것이지만 밑줄친 부분이 입력될 때에 매칭출력 Ro가 1이 되므로 카운터는 5번 증가하여 Exactly블록의 매칭출력을 1로 만들게 된다.

이러한 문제점을 해결하기 위하여 서브패턴이 R_1R_2* 또는 R_1R_2+인 경우에는 제한반복 패턴을 다음과 같은 방법으로 서브패턴이 반복 연산자로 끝나지 않도록 변환한 후에 그림 4의 회로를 사용하여 구현한다.

$(R_1R_2*)\{N\} = R_1(R_2*R_1)\{N-1\}R_2*$

$(R_1R_2*)\{N,\} = R_1(R_2*R_1)\{N-1,\}R_2*$

$(R_1R_2*)\{M,N\} = R_1(R_2*R_1)\{M-1,N-1\}R_2*$

3. 시작 제한반복의 블록 구조

패턴들은 입력 문자열의 임의의 위치에 존재할 수 있기 때문에 입력 문자열의 모든 위치에서 패턴 매칭을 시작할 수 있어야 한다. 어떤 입력 문자열은 주어진 패턴 집합에 대해서 여러 개의 패턴과 동시에 매칭이 진행될 수 있으며 이것을 중첩 매칭이라고 한다. 예를 들어서 패턴 집합 abc, bc, c에 대해서 입력 문자열 pqabc는 c가 입력이 될 때에 세 개의 패턴 abc, bc, c에 모두 중첩하여 매칭된다. NFA 기반의 정규표현식 패턴매칭 하드웨어 구현에서는 입력 문자열을 검사하는 동안에 시작 블록의 매칭입력을 1로 유지되도록 하여 입력의 모든 위치에서 패턴 매칭을 시작할 수 있게 한다[8]. 그렇지만 제한반복 블록의 구현에서는 이러한 중첩매칭의 구현이 어렵다는 것을 기존 연구[14]에서 언급한 바가 있다. 이 절에서는 이러한 문제에 대한 해결 방안을 제시한다.

3.1 시작 제한반복의 AtLeast 블록 구현

패턴이 제한반복으로 시작하는 경우에는 그림 4에서 제시한 Exactly 블록과 Between 블록은 중첩매칭과 관련하여 동작에 문제가 발생할 수 있다. 예를 들어서 패턴 a{3}bc에 대하여 a{3}를 Exactly 블록으로 구현하면, 입력 문자열 "aaaaabc"에 대하여 a가 반복될 때마다 카운터가 증가하여 a의 반복 횟수는 5가 되므로 패턴매칭이 실패한다. 그렇지만 실제로 이 패턴은 3번째 입력부터 시작하는 밑줄 친 부분의 입력과 매칭되므로 패턴매칭은 성공되어야 한다.

R{N}으로 시작하는 패턴에 대해서 입력 문자열이 서브패턴 R과 N번 이상 반복하여 매칭되면 R과 N번 반복 매칭된 것을 포함하므로 패턴의 시작에 있는 R{N}은 R{N,}으로 변경하여 AtLeast 블록으로 구현하면 위와 같은 문제를 해결할 수 있다. 이와 마찬가지로 패턴의 시작에 있는 R{M,N}도 Between 블록 대신에 AtLeast 블록 R{M, }으로 바꾸어서 구현해야 한다. 이처럼 패턴의 시작에 위치한 제한반복 블록은 제한반복 연산자 종류와 관계없이 AtLeast 블록을 사용하여 구현한다.

그리고 AtLeast 블록 R{N,}이 시작 블록인 경우에 패킷 데이터를 검사하는 동안 R블록의 입력이 1로 유지되므로 그림 4(d)의 AtLeast 블록은 피드백이 불필요하여 제거되어 그림 6(a)와 같이 간단해진다.

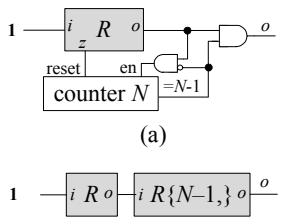


그림 6. 시작 제한반복 블록
Fig. 6 Beginning Constraint Repetition Block
(a) 시작 제한반복 블록 R{N,}의 구조 (R{N}, R{N,M} 포함)
(b) SP-중첩 패턴 R에 대한 시작 제한반복 블록 R{N,}의 구조

3.2 SP 중첩 서브패턴에 대한 시작 제한반복 구현

패턴의 접두사(suffix)와 접미사(prefix)가 일치하는 경우에 SP-중첩 패턴이라고 한다[14]. 예를 들어 abcab는 ab가 접두사와 접미사이므로 SP-중첩 패턴이다. R{N,}으로 시작하는 패턴에서 R이 SP-중첩 패턴이면 그림 6(a)의 블록은 중첩매칭으로 인하여 R의 반복 횟수를 잘못 계수하는 문제가 발생할 수 있다.

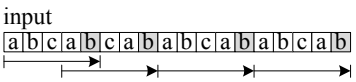


그림 7. SP중첩서브패턴에 대한 입력 문자열
Fig 7. an input string for SP-overlapped subpattern

예를 들어서 패턴 (abcab){3,}…을 생각해보자. 이 패턴에 대한 매칭 회로에 입력 문자열로 그림 7의 문자열이 들어올 때에 시작블록은 모든 위치에서 매칭이 시작되므로 AtLeast 블록은 회색 바탕의 b에 대하여 모두 반복을 계수하여 반복횟수가 4가 된다. 그렇지만 실제로는 그림 7에서 보듯이 abcab가 4번이 아니라 3번 반복 입력된 것이다. 첫 번째 반복은 첫 번째 회색 바탕의 b다음에 c가 입력되어 반복이 중단되므로 카운터가 리셋되어야 하지만 R블록은 SP 중첩 때문에 두 개의 매칭이 중첩하여 진행되어 매칭상태가 모두 0이 되지 않으므로 리셋 신호를 발생시키지 않는다.

이러한 문제를 해결하기 위해서 시작 제한반복 블록의 서브패턴 R이 SP-중첩 패턴인 경우에는 AtLeast 블록 R{N,}을 그림 6(b)와 같이 R과 R{N-1,}로 나누어 구현한다. 이 구현에서 AtLeast 블록 R{N-1,}은 시작 블록이 아니어서 패킷입력이 서브패턴 R과 매칭될 때에만 매칭입력이 1이 되기 때문에 SP-중첩 문제가 발생하지 않아서 R블록은 두 개의 매칭이 중첩하여 진행되지 않는다.

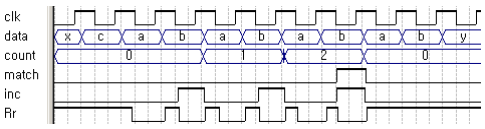
IV. 평 가

본 연구에서는 심층패킷 검사의 고속화를 위한 정규표현식 패턴매칭 회로의 설계 방법을 다루었고 특히 기존 연구에서 제대로 다루지 못한 제한반복 회로의 블록 구조를 제안하였다.

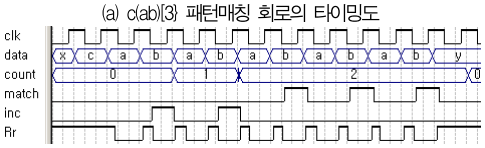
설계된 제한반복 블록들은 Verilog 언어로 설계하고 Altera의 Quartus 9.0을 사용하여 Cyclone II FPGA를 타겟 디바이스로 합성한 후 타이밍 시뮬레이션을 통하여 동작을 확인하였다. 타이밍 시뮬레이션을 통하여 본 논문에서 설계한 블록들이 제대로 동작하는 지를 확인하였다.

그림 8은 Exactly 블록, AtLeast 블록, Between 블록의 동작에 대한 타이밍도이다. 이 그림에서 data는 입력 문자열, count는 카운터 값, match는 최종 매칭출력, inc와 Rr은 R블록의 카운터 증가신호와 카운터 리셋신호를 나타낸다.

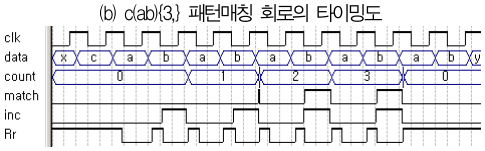
그림 8(a)는 Exactly 블록을 포함한 c(ab){3}에 대한 패턴매칭 회로에서 입력 문자열이 cababababy...일 때의 타이밍도이다. 3번째 반복(count값은 2)에서 match가 1이되며 4번째 반복에서는 match가 0이 됨을 확인할 수 있다. 그림 8(b)는 AtLeast 블록을 포함한 c(ab){3,}에 대한 패턴매칭 회로에서 입력 문자열이 cababababy...일 때의 타이밍도이다. 3번째 이상의 반복(count값은 2로 유지됨)에서 match가 계속하여 1이 됨을 확인할 수 있다. 그림 8(c)는 Between 블록을 포함한 c(ab){3,4}에 대한 패턴매칭 회로에서 cababababababy...가 입력 문자열일 때의 타이밍도이다. match는 3번째와 4번째 반복에서는 1이 되지만 5번째 반복에서는 0이 됨을 확인할 수 있다.



(a) c(ab){3} 패턴매칭 회로의 타이밍도



(b) c(ab){3,} 패턴매칭 회로의 타이밍도



(c) c(ab){3,4} 패턴매칭 회로의 타이밍도

그림 8. 제한반복 블록에 대한 타이밍도
Fig. 8. Timing diagram for constraint repetition blocks

그림 9는 SP중첩 서브패턴에 대한 시작 제한반복 패턴인 (abcab){3,}에 대한 패턴매칭 회로에 대한 타이밍도이다. 이 회로는 그림 6(b)와 같이 abcab에 대한 회로와 (abcab){2,}에 대한 회로를 연결하여 구현하였으며 R은 첫 번째 회로의 매칭출력이고 Rn_1은 두 번째 AtLeast 회로의 R블록의 매칭출력이다. 출력 R의 파형에서 타원으로 표시한 두 부분은 첫 번째 R블록이 주어진 입력에 대해서 중첩매칭이 됨을 보여준다. 그렇지만 출력 Rn_1의 파형은 AtLeast 회로의 R블록에서는 중첩매칭이 발생하지 않음을 보여준다. 그림 9에서 문자열 abcab의 반복 입력 횟수가 3번이상일 때에 match가 1이 됨을 확인할 수 있다. 이처럼 본 연구에서 제시한 여러 가지 제한반복 블록들이 정상적으로 동작함을 확인할 수 있다.

본 연구에서 제시한 제한반복 블록과 Bispo 등이 제안한 제한반복 블록의 구현 가능 범위와 소요되는 하드웨어 자원에 대한 비교를 통하여 본 제시된 구조에 대한 평가를 하고자 한다.

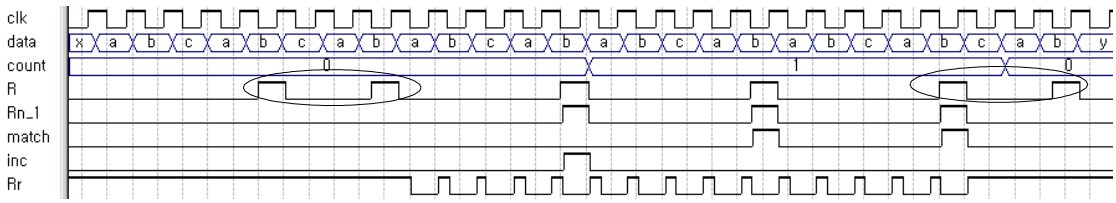


그림 9. SP중첩 서브패턴에 대한 시작 제한반복 블록의 타이밍도
Fig. 9. Timing diagram for beginning constraint repetition block with SP-overlapped subpattern

표 2는 본 연구에서 제안한 제한반복 블록과 Bispo 등이 제안한 제한반복 블록에서 구현 가능한 서브패턴의 형태에 대해서 비교한 것이다. Bispo 등이 제안한 두 가지 방법은 단일문자와 고정길이의 문자열에 대한 제한 반복을 구현할 수 있지만 (ab+){10}과 같은 가변길이 패턴에 대한 제한 반복은 구현할 수 없다. 가변길이 패턴에 대한 제한 반복은 반복횟수만큼 펼쳐서 구현해야하므로 매우 비효율적이다. 이에 비해서 본 연구에서 제안한 구조는 일반적인 정규표현식 서브패턴에 대한 제한반복을 구현할 수 있기 때문에 가변길이 패턴에 대한 제한반복을 효율적으로 구현할 수 있다.

표 2. 제한반복 블록의 구현 가능 범위 비교
Table 2. Comparison of functions implemented in each constraint repetition block

서브패턴의 형태	Bispo'06[12]	Bispo'08[14]	제안된 구조
단일문자	O	O	O
고정길이 문자열	X	O	O
일반적인정규표현식	X	X	O

그리고 본 연구에서는 제안된 기본적인 제한반복 블록 구조를 사용할 때에 문제가 발생하는 (ab+){10}과 같이 반복으로 끝나는 서브패턴에 대한 제한반복의 구현 방법과 시작 제한반복 블록의 서브패턴이 SP중첩일 때의 구현 방법을 함께 제시하여 구현에 제약이 없는 일반적인 정규표현식 서브패턴에 대한 제한반복 블록 설계 방법을 완성하였다.

표 3은 Bispo 등이 제안한 구조와 본 연구에서 제안한 구조의 제한 반복 블록에 대한 회로 복잡도를 플립플롭, 카운터와 SRL16/플립플롭 쌍의 수로 나타낸 것이다. FPGA의 기본 단위인 LE(logic element)는 LUT(lookup table)과 플립플롭의 쌍으로 구성되어 있으며, 회로는 플립플롭 개수 이상의 LE를 필요로 한다. 카운터는 해당 비트 수 만큼의 플립플롭을 사용한다. Xilinx FPGA는 LUT를 16비트 시프트 레지스터인 SRL16으로 사용할 수 있도록 하였으며 LE는 SRL16/플립플롭의 쌍으로 사용된 수 있다. 따라서 표 3의 플립플롭과 카운터 비트수를 합하면 구현에 필요한 최소 LE 개수가 된다.

표 3. 제한 반복 블록의 하드웨어 구조 비교
Table 3. Comparison of constraint repetition hardware architectures

제한반복연산	Bispo'06[12]	Bispo'08[14]	제안된 구조
Exactly블록 R{N}	- $\lceil N/17 \rceil$ SRL16/플립플롭 - 4비트 카운터	- $\lceil \Delta \times N/17 \rceil$ SRL16/플립플롭 - 4비트카운터 (Δ: 서브패턴 길이)	- $\log_2 N$ 비트 카운터 - Δ 플립플롭 (Δ: 서브패턴의 문자수)
AtLeast블록 R{N,}	- $\log_2 N$ 비트 카운터	모름	- $\log_2 N$ 비트 카운터 - Δ 플립플롭
Between블록 R{M,N}	- $\lceil N/17 \rceil$ SRL16/플립플롭, - $\log_2 M$ 비트 카운터, - $\log_2 (N-M)$비트 카운터	모름	- $\log_2 N$ 비트 카운터 - Δ 플립플롭

Bispo 등[12]이 제안한 단일문자의 제한반복만 지원하는 구조는 Xilinx FPGA에서만 지원되는 SRL16을 사용하였으며 Exactly 블록과 Between 블록은 SRL16/플립플롭 쌍의 수가 반복횟수 N이 17이 증가할 때마다 1개씩 증가하여 N에 비례하게 된다. 그리고 Exactly 블록에서는 SRL16의 리셋을 위해서 4비트 카운터를 사용하며 Between 블록은 두 개의 카운터를 사용한다.

고정길이 문자열에 대한 제한반복을 지원하는 구조에서 [14] Exactly 블록은 $\lceil N/17 \rceil$ 과 서브패턴인 고정길이 문자열의 길이 Δ를 곱한 만큼 SRL16/플립플롭의 쌍의 수가 증가된다. AtLeast 블록과 Between 블록에 대해서는 구조가 명확하게 설명되어 있지 않아서 데이터를 제시하지 않았다. 이처럼 앞의 두 방법 모두 플립플롭의 개수가 반복횟수에 비례한다.

이에 비해서 본 논문에서 제안한 방식은 $\log_2 N$ 비트 카운터와 몇 개의 게이트를 사용하여 제한반복 블록을 구현할 수

있으므로 N이 커지더라도 추가되는 하드웨어 자원의 양은 크게 증가하지 않는다. 그리고 R블록이라고 불렀던 서브패턴에 대한 패턴매칭 회로도 함께 구현해야 하는 데 단일문자 서브패턴에 대한 블록은 한 개의 플립플롭이 사용되며, 길이가 Δ인 고정 길이 서브패턴에 대한 블록은 Δ개의 플립플롭이 사용된다. 그리고 일반적인 정규표현식 서브패턴은 서브패턴에 포함된 문자수만큼의 플립플롭이 사용된다. 제안한 방식은 SRL16과 같은 특정회사의 FPGA에서만 제공되는 하드웨어 자원을 사용하지 않았기 때문에 제조사에 관계없이 FPGA를 선택할 수 있는 장점이 있다.

제한반복의 서브패턴이 반복으로 끝나는 경우와 시작 제한반복에서 서브패턴이 SP중첩일 때에는 2개의 R블록을 구현해야 하므로 서브패턴에 Δ개의 문자가 포함되어 있다면 2Δ개의 플립플롭이 사용된다.

이처럼 제안된 정규표현식 제한반복 블록은 구현 가능한 제한반복의 서브패턴 형태와 하드웨어 복잡도 측면에서 모두 기존 연구보다 우수하다.

Ⅵ. 결 론

본 연구에서는 NFA 기반의 정규표현식 패턴매칭 하드웨어 구조에서 기존 연구에서 완전하게 구현하지 못한 일반적인 정규표현식 서브패턴의 제한반복을 제약이 없이 구현할 수 있는 제한반복 블록 구조를 제시하였다.

기존 연구에서는 구현할 수 없는 서브패턴의 제한반복을 펼쳐서 비효율적으로 구현해야 했지만 본 연구에 제시된 제한반복 블록을 사용하면 이러한 구현을 훨씬 효율적으로 수행할 수 있다.

제안된 제한반복 블록 구조를 사용할 때 문제가 발생할 수 있는 반복으로 종료되는 서브패턴에 대한 제한반복과 SP중첩 서브패턴의 제한반복으로 시작하는 패턴에 대해서 제한반복 구현 방법을 함께 제시하여 정규표현식의 제한반복 연산자를 제약없이 구현할 수 있도록 하였다.

하드웨어 구조 측면에서 제안된 제한반복 블록구조는 카운터 기반으로 설계하여 특정 회사의 FPGA에서만 구현이 가능하였던 기존연구의 단점을 없애서 여러 회사의 FPGA에서 사용할 수 있는 일반적인 구조를 가지며 기존 연구보다 하드웨어 복잡도도 감소한다.

본 연구 결과를 사용하면 모든 제한반복 패턴매칭 하드웨어 설계를 서브패턴 형태에 관계없이 펼치지 않고 구현함으로써 제한반복 연산을 포함하는 패턴 집합에 대한 NFA 기반의 정규표현식 패턴매칭 하드웨어의 설계를 더 효율적으로 수행할 수 있다. 효율적인 정규표현식 패턴매칭 하드웨어의 설계는 하드웨어 기반 침입탐지 시스템의 고속화와 효율적인 설계에 기여를 할 수 있다. 향후 과제로 패턴매칭의 고속화를 위해서 정규표현식의 제한반복 패턴매칭을 한 번에 여러 바이트 단위로 수행할 수 있는 구조를 설계할 필요성이 있다.

참고문헌

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

[2] Bro Intrusion Detection System, http://www. bro-ids.org

[3] Bleeding Edges, http://bleedingedges.net/

[4] Christopher R. Clark and D. E. Schimmel, "Scalable pattern matching for high speed networks," in IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM'04), pp. 249-257, 2004.

[5] B. L. Hutchings, R. Franklin, et al. "Assisting network intrusion detection with reconfigurable hardware," in IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM'02), pp. 111-120, 2002.

[6] J. Moscola, J. Lockwood, R. Loui, and M. Pachos, "Implementation of a content-scanning module for an Internet firewall," in IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM'03), pp. 31-38, 2003.

[7] I. Sourdis and D. Pnevmatikatos, "Pre-decoded CAMs for Efficient and High-Speed NIDS Pattern Matching," in IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM'04), pp. 258-267, 2004.

[8] R. Sidhu and V.K. Prasanna, "Fast Regular Expression Matching using FPGAs," in IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM'01), pp. 227-238, 2001.

[9] C.H. Lin, C.T. Huang, et al. "Optimization of regular expression pattern matching circuits on FPGA," in Proc. of Conference on Design, Automation and Test in Europe (DATE'06), pp. 12-17, 2006.

[10] Z.K. Baker, H.J. Jung and V.K. Prasanna, "Regular expression software deceleration for intrusion detection systems," in Int'l Conf. on Field Programmable Logic and

Applications (FPL'06) pp. 418-425, 2006.

[11] B. C. Brodie, D. E. Taylor, and R. K. Cytron, "A scalable architecture for high-throughput regluar expression pattern matching," ACM SIGARCH Computer Architecture News, vol. 34, no. 2, pp. 191-202, 2006.

[12] J. Bispo, I. Sourdis, J. Cardoso, and S. Vassiliadis, "Regular Expression Matching for Reconfigurable Packet Inspection," in IEEE Int'l Conf. on Field Programmable Techn.ology (FPT'06), pp. 119-126, 2006.

[13] I. Sourdis, S. Vassiliadis, J. Bispo, and J. Cardoso, "Regular Expression Matching in Reconfigurable Hardware," Journal of Signal Processing Systems, vol. 51, pp. 99-121, 2008.

[14] J. Bispo, and J. Cardoso, "Synthesis of regular expressions for FPGAs," International Journal of Electronics, vol. 95, no. 7, pp. 685-704, 2008.

[15] J. Ho and G. Lemieux, "PERG: a scalable FPGA-based pattern-matching Engine with Consolidated Bloomier Filters," in IEEE Int'l Conf. on Field Programmable Technology(FPT'08), pp.73-80, 2008.

[16] J. Ho and G. Lemieux, "PERG-Rx: a hardware pattern-matching engine supporting limited regular expressions," in proc. ACM/SIGDA Int'l. Symp. on Field programmable gate arrays (FPGA'09), pp. 257-260, 2009.

저 자 소 개

윤 상 균

1984: 서울대학교 전자공학과 학사.
1986: KAIST전기및전자공학과 석사.
1995: KAIST전기및전자공학과 박사.
현 재: 연세대학교 컴퓨터정보통신공학부 교수
관심분야: 컴퓨터 및 네트워크 시스템, 임베디드시스템, NIDS
Email : skyun@yonsei.ac.kr

이 규 희

2007: 연세대학교 컴퓨터공학전공 학사.
2009: 연세대학교 전산학과 석사
현 재: 연세대학교 전산학과 박사과정
관심분야: 임베디드시스템, NIDS,
Email : powerpc@yonsei.ac.kr