NIDS를 위한 다중바이트 기반 정규표현식 패턴매칭 하드웨어 구조

종신회원 윤 상 균*, 준회원 이 규 희*

A Hardware Architecture of Multibyte-based Regular Expression Pattern Matching for NIDS

SangKyun Yun* *Lifelong Member*, KyuHee Lee* *Associate Member*

요 약

최근의 네트워크 침입탐지 시스템에서는 침입이 의심되는 패킷을 나타내는 데 정규표현식이 사용되고 있다. 고속 네트워크를 통해서 입력되는 패킷을 실시간으로 검사하기 위해서는 하드웨어 기반 패턴 매칭이 필수적이며 변화되는 패턴 규칙을 다루기 위해서는 FPGA와 같은 재구성 가능한 디바이스를 사용하는 것이 바람직하다. FPGA의 동작 속도 제한으로 바이트 단위의 패킷 검사로는 실시간 검사를 할 수 없는 경우에 이를 해결하기 위해서 여러 바이트 단위로 검사하는 것이 필요하다. 본 논문에서는 정규표현식 패턴 매칭을 n바이트 단위로 처리하는 하드웨어의 구조와 설계 방법을 제시하고 이에 대한 패턴 매칭 회로 생성기를 구현한다. Snort 규칙에 대해 FPGA로 합성된 하드웨어는 n=4일 때에 규칙에 따라서 2.62~3.4배의 처리 속도 향상을 보였다.

Key Words : Pattern matching hardware, Intrusion detection, NIDS, Regular expression

ABSTRACT

In recent network intrusion detection systems, regular expressions are used to represent malicious packets. In order to process incoming packets through high speed networks in real time, we should perform hardware-based pattern matching using the configurable device such as FPGAs. However, operating speed of FPGAs is slower than giga-bit speed network and so, multi-byte processing per clock cycle may be needed. In this paper, we propose a hardware architecture of multi-byte based regular expression pattern matching and implement the pattern matching circuit generator. The throughput improvements in four-byte based pattern matching circuit synthesized in FPGA for several Snort rules are 2.62~3.4 times.

Ⅰ. 서 론

네트워크의 사용이 급격히 증대되면서 네트워크를 통한 시스템에 대한 공격과 침입이 늘어나고 있다. 침입이 의심되는 패킷을 찾아내기 위하여 네트워크 침입탐지 시스템(Network Intrusion Detection System: NIDS)이 널리 사용된다. NIDS에서 시간이 가장 많이 소요되는 작업은 네트워크를 통해 입력되는 모든 패킷들을 침입이 의심되는 패턴 규칙 집합의 문자열과 비교하는 다중 패턴 매칭 작업이다.

※ 이 연구는 2008학년도 연세대학교 학술연구비의 지원에 의하여 이루어진 것임.
* 연세대학교 컴퓨터정보통신공학부 (skyun@yonsei.ac.kr)
논문번호: KICS2008-06-281, 접수일자 : 2008년 6월 21일, 최종논문접수일자 : 2008년 11월 12일

네트워크의 속도가 빨라지면서 다중 패턴 매칭 알고리즘을 소프트웨어로 구현한 소프트웨어 기반 방식으로는 패킷 검사를 실시간으로 수행할 수 없게 되었고 이를 해결하기 위해서 하드웨어 기반의 패턴 매칭이 제안되었다. 패턴 규칙은 지속적으로 갱신되기 때문에 메모리에 저장되거나 FPGA(Field Programmable Gate Array)와 같은 재구성이 가능한 하드웨어를 사용하여 구현된다.

Snort[1], Bro[2], Bleeding Edge[3]와 같은 최근의 NIDS에서는 패턴 규칙을 나타내는 데에 고정 문자열 패턴 뿐 만 아니라 정규표현식도 사용한다. 그러므로 패턴 매칭 하드웨어는 정규표현식 패턴 매칭을 지원해야 한다. 정규표현식 패턴 매칭을 지원하는 하드웨어 구현에 대한 연구가 여러 연구자들에 의해서 수행되었다[4-10]. 그런데 FPGA의 최대 클록 속도는 수백 MHz로서 수십 Gbps에 이르는 고속 네트워크 속도에 비해 낮기 때문에 한 클록에 한 바이트씩 검사를 하면 패킷을 실시간으로 처리하지 못할 수도 있게 된다. 이에 대한 해결책은 한 번에 여러 바이트를 검사하는 것이다.

고정 문자열에 대한 패턴 매칭을 다중 바이트 단위로 수행하는 하드웨어에 대한 연구는 이미 수행되었으나[11,12] 정규표현식을 다중 바이트 단위로 처리하는 하드웨어에 대한 연구는 최근에 수행되었다[10]. 이 연구에서는 *n*바이트 단위의 정규표현식 패턴 매칭 하드웨어에 대한 구현 방법을 소개하였지만 이 방법에 의한 정규표현식 패턴 매칭 회로 생성기는 일반적인 *n*바이트 단위에 대한 것이 아닌 1바이트와 2바이트 단위에 대한 것을 개별적으로 작성하였다. 그리고 자원 절약을 위하여 접두 고정 패턴 공유 방법을 함께 사용하였다.

본 논문에서는 *n*바이트 단위로 정규표현식 패턴 매칭을 수행하는 하드웨어 구조와 이에 대한 설계 방법을 체계화하여 제시하고 이 방법을 사용하여 일반화된 *n*바이트 단위 정규표현식 패턴 매칭 회로 생성기를 구현한다. 그리고 회로 생성 시에 접두 패턴 공유는 기존 연구에서 사용했던 고정 문자열 뿐 만 아니라 연산자를 포함한 정규표현식도 공유하도록 하여 자원을 더 절약할 수 있게 한다.

이 논문의 구성은 다음과 같다. Ⅱ장에서 관련연구를 소개하고 Ⅲ장에서 다중 바이트 기반 정규표현식 패턴 매칭 하드웨어 구조를 제시하며, Ⅳ장에서는 패턴 하드웨어 생성기 구현에 대해서 다루고 Ⅴ장에서 제시된 구조에 대한 분석, 평가를 하고 Ⅵ장에서 결론을 맺는다.

Ⅱ. 관련 연구

2.1 정규표현식

정규표현식은 문자열 패턴을 나타내기 위해서 널리 사용되고 있으며 문자와 메타문자들로 구성된다. 메타문자들 중에서 $, ^, ., [ ] 등은 특정 문자나 문자 그룹을 나타내기 위해서 사용되며 *, +, ? 등은 특정 패턴의 반복을 나타내기 위해서 사용된다. 그리고 | 는 여러 패턴을 나타내기 위한 OR 연산자이다. 예를 들어서 *a**는 *a*의 0번 이상 반복을, *a*+는 *a*의 1번 이상 반복을, *a*?는 *a*가 없거나 1번 나타남을 표시하고 *a* | *bc* 는 패턴 *a* 또는 *bc*를 표시한다.

Snort에서는 Perl 호환 정규표현식(PCRE)[13]을 사용하여 정규표현식을 나타낸다. 다음은 PCRE 형식의 정규표현식을 사용한 Snort 규칙의 예이다.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21
(  ... pcre: "/^PASS\s*\n/smi"; ...)
```

이 규칙에서 pcre: 다음에 있는 문자열이 정규표현식으로 나타낸 검색 패턴으로서 외부 네트워크에서 침입탐지 대상 네트워크에 있는 시스템의 21번 포트로 전송되는 tcp 프로토콜 패킷을 검사하여 PASS로 시작(^)하고 바로 뒤이어 임의의 개수(*)의 공백(\s)으로 채워진 줄이 발견되면 경고를 함을 뜻한다.

2.2 패턴 매칭 하드웨어

정규표현식은 여러 연구에서 이에 대한 유한오토마타(Finite Automata: FA)를 기반으로 하드웨어로 구현되었다. 유한오토마타는 비결정 유한오토마타(NFA)와 결정 유한오토마타(DFA)로 구분되는데 소프트웨어로 구현하기에는 DFA가 적합하지만 상태수가 많은 단점이 있다. NFA는 상태수가 적은 대신에 동시에 여러 상태가 활성화될 수 있어서 소프트웨어로 구현하기에 덜 적합하다. 그렇지만 하드웨어는 병렬성을 가지고 있기 때문에 NFA를 쉽게 구현할 수 있다.

Shidu 등은[4] 정규표현식의 NFA를 하드웨어로 구현하기 위한 문자, 스타(*), 결합, OR(|), 괄호에 대한 기본 블록을 제시하였으며 바이트 단위로 패턴 매칭을 수행한다. 각 패턴 매칭 블록에서 문자를 개별적으로 비교한다면 입력 문자를 모든 블록에 제공해야 하므로 복잡해진다. Clark 등은[11] 각 문자 또는 문자 그룹에 대한 비교 결과를 디코더에서 제

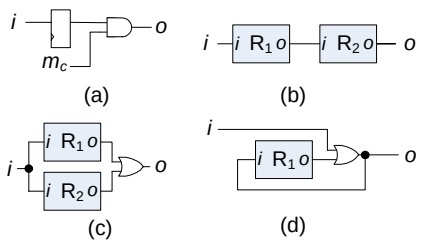

그림 1. 기본적인 블록 구현
(a) 단일문자 c (b) R_1R_2 (c) R_1 | R_2 (d) R_1*

공하고 디코더의 출력을 모든 블록이 공유하는 공유 디코더 방법을 제시하여 각 블록에서 비교기를 개별적으로 사용하지 않도록 하였으며 고정 문자열 매칭을 4바이트 단위로 수행하는 방법도 제시하였다. 그림 1은 Shidu 등이 제시한 기본 블록을 나타낸 것인데 원래의 그림에 있는 비교기 대신에 공유 디코더 출력을 사용하는 것으로 바꾸어 그렸다. 이 그림에서 m_c는 입력과 문자 c의 비교 결과를 의미하며 R_1과 R_2를 포함한 사각형은 해당 정규표현식의 패턴 매칭을 구현한 블록이다. Lin 등은[6,9] 여러 패턴들의 공통된 접두 및 접미 패턴 처리 하드웨어를 공유하는 방법을 제시하여 자원을 절약할 수 있게 하였다. 그렇지만 이 두 가지 방법은 정규표현식의 구현은 다루지 않았다.

Bispo 등은[8]은 시프트 레지스터와 카운터를 사용하여 문자의 반복이 제한된 범위에서 이루어지는 정규표현식을 바이트 단위로 구현하였다. Lee 등은[10] 정규표현식을 다중 바이트 단위로 처리하는 패턴 매칭 하드웨어 구조를 제시하였다. 그렇지만 이 구조에 대한 패턴 매칭 회로 생성기는 1바이트와 2바이트 단위에 대한 것만을 개별적으로 생성하였으며 일반적인 n바이트 단위의 정규표현식 패턴 매칭 회로는 구현하지 않았다.

본 논문에서는 정규표현식을 일반적인 n바이트 단위로 처리할 수 있는 하드웨어 구조와 이에 대한 체계적인 설계 방법을 제시하고 이 방법을 사용하여 일반적인 n바이트 단위의 정규표현식 패턴 매칭 회로 생성기를 구현한다. 정규표현식의 반복 메타문자는 *, +, ?를 지원하도록 하고 문자의 반복 뿐 만 아니라 정규표현식의 반복도 처리할 수 있게 한다.

Ⅲ. 정규표현식 패턴 매칭 하드웨어 구조

이 절에서는 패턴 매칭을 n바이트 단위로 수행하는 다중바이트 기반의 정규표현식 패턴매칭 하드웨어 구조를 소개한다. 편의상 4바이트 단위의 패턴 매칭 구조를 예로서 설명한다. 그렇지만 제시되는 구조는 n바이트 단위로 처리하는 구조로 일반화될 수 있다.

3.1 입력 문자열 비교와 상태 구현

정규표현식의 구현은 기본적으로 Shidu[4]가 제안한 방법과 같이 NFA를 기반으로 하며 각 상태마다 플립플롭을 할당하는 one-hot 인코딩을 사용하여 여러 개의 상태를 동시에 활성화 시킬 수 있도록 한다. 활성화된 상태의 플립플롭의 출력은 1이 된다.

4바이트 단위의 패턴 매칭을 위해서 입력 패킷은 매 클록마다 4바이트씩 다음으로 이동되며 NFA는 4바이트 입력에 따라서 상태가 전이되도록 구성된다. 각 상태에서의 4바이트 단위의 입력 비교는 각 상태에서 비교기를 개별적으로 사용할 필요가 없도록 Clark[11]가 사용한 공유 디코더 방법을 사용한다. 그림 2과 같이 4개의 문자 디코더를 사용하여 4바이트 입력에 대한 바이트 단위의 비교 결과를 제공하며 각 디코더의 결과에 대해서 AND 연산을 수행하여 4바이트 문자열의 비교 결과를 얻을 수 있다. 바이트 위치를 나타내기 위해서 각 문자 디코더의 출력들에 첨자 0, 1, 2, 3을 붙인다. 예를 들어서 q_1은 바이트 1이 q일 때에 출력이 1이 되는 두 번째 문자 디코더의 출력이다. $p_0q_1r_2s_3$이 1이면 현재의 4바이트 입력 문자열이 “pqrs”임을 나타낸다.

그림 3은 현재 입력이 “pqrs”일 때에 다음 상태로 전이되는 회로를 나타낸 것으로서 입력 비교회

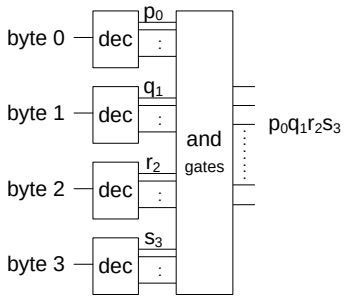

그림 2. 입력 비교 출력을 제공하는 공유디코더 회로

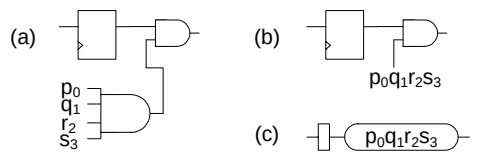

그림 3. 상태전이 회로 (a) 구현된 회로 (b), (c) 간편 표기

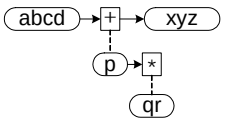

그림 4. 정규표현식 abcd(p(qr)*)+xyz의 분리

로는 (a)와 같이 문자 디코더 출력 p_0, q_1, r_2, s_3의 AND로 구현되지만, 공유 디코더 출력을 사용하게 되므로 편의상 (b)와 같이 식으로 표기하거나 (c)와 같이 현재 상태와 비교결과의 AND도 생략하고 타원 안에 비교문자열만 표기하여 나타낼 것이다.

정규표현식에서 [abc], [^xyz], \d등과 같이 표현되는 문자 클래스들은 공유 디코더 회로에서 함께 구현할 수 있다. 문자 클래스 입력에 대한 상태 전이 회로는 문자 비교 결과 대신에 문자 클래스 비교 결과를 사용하는 것을 제외하면 문자 입력에 대한 상태 전이 회로와 같다.

정규표현식의 하드웨어 구현은 다음과 같은 절차를 따른다.

1. 정규 표현식 패턴을 문자열 패턴인 고정패턴과 연산자가 적용되는 순수 정규표현식 패턴으로 분리하고 순수 정규표현식 패턴에서 피연산자 패턴도 마찬가지 방법으로 분리한다. 예를 들어서 정규표현식 abcd(p(qr)*)+xyz은 먼저 abcd, (p(qr)*)+, xyz로 분리되며 +의 피연산자 패턴은 p(qr)*는 p와 (qr)*로 분리되며 *의 피연산자 qr은 고정패턴이므로. 더 이상 분리할 필요가 없다. 그림 4는 이 정규표현식이 분리된 결과를 그림으로 나타낸 것이다.

2. 분리된 패턴에 대한 패턴 매칭 회로를 앞에서부터 구현한다. 순수 정규표현식 패턴은 피연산자 패턴을 먼저 구현한 후에 연산자를 구현한다. 구현된 각 패턴 매칭 회로를 순서대로 연결하여 정규표현식 하드웨어를 구현한다.

3.2 다중바이트 기반 고정패턴 매칭 회로

다중 바이트 기반 고정 패턴 매칭 회로는 기본적으로 Clark[1]가 제시한 방법을 따른다. 검색할 패턴들은 패킷의 임의의 위치에 포함될 수 있으므로 패턴이 패킷의 어느 위치에서 시작되어도 매칭을 할 수 있도록 패턴의 시작위치가 바이트 0부터 바이트 3까지의 4가지 패턴을 병렬로 비교해야 한다.

그림 5(a)는 문자열 "abcdef"에 대한 고정 패턴 매칭 회로이다. 고정 패턴 회로에서 비교 문자열에 바이트 0이 포함될 때마다 다음 사이클에서 비교되도록 앞에 플립플롭을 배치한다. 그림 5(b)는 이 회로에 대

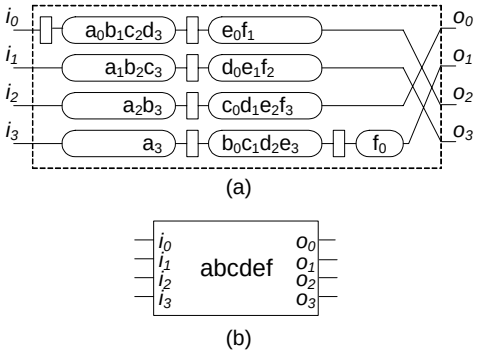

그림 5. 4바이트 단위 고정패턴 매칭 회로
(a) 내부 회로 (b) 블록도

한 블록도로서 사각형 안에 패턴을 기입한다.

4바이트 단위의 패턴 매칭 회로는 각각 4개의 입출력을 갖는다. 입력 $i_k(k=0,1,2,3)$는 패턴의 시작위치가 바이트 k인 입력을 나타내며 출력 o_k는 패턴의 마지막 위치 다음에 바이트 k가 연결되는 출력을 나타낸다. 여러 패턴 매칭 회로를 연결할 때에 출력 o_k는 다른 회로의 입력 i_k와 연결된다. 입력 i_0는 플립플롭을 거치지만 입력 i_1, i_2, i_3는 플립플롭을 거치지 않고 비교회로에 직접 연결된다. 출력 $o_k(k=1,2,3)$가 다른 패턴 매칭 회로의 플립플롭을 거치지 않는 입력 $i_k(k=1,2,3)$로 연결될 때 출력 o_k에서 제공되는 바이트 k-1까지의 비교 결과와 입력 i_k의 첫 단계에서 수행하는 바이트 k 이후의 비교 회로가 플립플롭을 거치지 않고 직렬로 연결되어 같은 클록 사이클에서 4바이트 전체에 대한 비교가 수행될 수 있다.

3.3 다중바이트 기반 정규표현식 패턴 매칭 회로

반복을 나타내는 연산자 *, +, ?와 OR 연산자 |에 대한 다중 바이트 기반 패턴 매칭 회로는 반복되는 피연산자 패턴이나 OR의 피연산자 패턴에 대한 패턴 매칭 회로를 먼저 구성한 후에 그림 6과 같이 피연산자 패턴 매칭 회로의 외부에 연산자에 대한 회로를 추가하여 구현한다.

그림 6에서 내부에 4가 표시된 OR 게이트는 4비트의 비트 단위 OR 연산을 수행하는 4개의 OR 게이트를 나타낸다. 반복 회로에서 패턴 R의 출력 o_k는 OR 게이트를 거쳐서 같은 회로의 입력 i_k로 피드백 되어 연결된다.

그림 7은 이와 같은 방식으로 구현된 정규표현식 r+에 대한 패턴 매칭 회로이다. 먼저 그림 7(a)와 같이 반복 문자인 r에 대한 패턴 매칭 회로를 구성

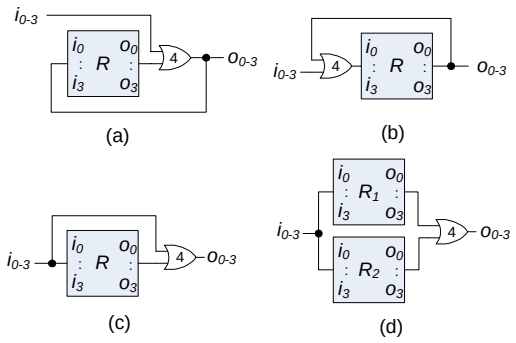

그림 6. 정규표현식 패턴 매칭 회로
(a) R* (b) R+ (c) R? (d) R_1 | R_2

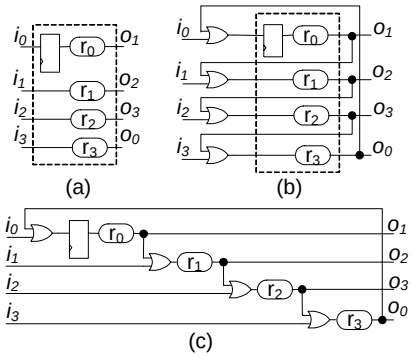

그림 7. 정규표현식 r+에 대한 패턴 매칭 회로
(a) 패턴 r에 대한 회로 (b), (c) 패턴 r+에 대한 회로

한 후에 그림 6(b)에서 제시한 방법을 사용하여 정규표현식 r+ 회로를 구성하면 그림 7(b)와 같은 회로가 만들어지며 그림 7(c)는 이 회로를 펼쳐서 그린 것이다. 이 회로에서 피드백을 통해서 r_0, r_1, r_2, r_3과의 비교회로가 플립플롭을 거치지 않고 직렬로 연결되어 한 클록 사이클 동안 4바이트가 동시에 비교될 수 있다.

3.4 완전한 정규 표현식 패턴 매칭 회로

분리하여 구현된 고정패턴 회로와 순수 정규표현식 회로들을 그림 8과 같이 패턴 순서대로 같은 첨자의 입출력끼리 연결하면 완전한 정규표현식 패턴 매칭 회로가 완성된다.

완전한 패턴에 대한 패턴 매칭 회로의 입력은 시작 상태 start와 연결한다. 검색할 패턴들은 패킷의 임의의 위치에 포함될 수 있으므로 패킷의 어느 위치도 패턴 매칭의 시작 지점이 될 수 있다. 따라서 현재 패킷 입력에 대해서 패턴의 첫 부분과의 매칭을 항상 수행하도록 패턴 매칭이 진행되는 동안 start는 1로 유지되어야 한다. 그리고 시작상태가 같은 시점에 패턴 매칭 회로의 첫 단계로 공급되도록 플립플롭을 거치지 않은 입력 i_1, i_2, i_3으로는 시작 상태 start가 플립플롭을 거쳐서 제공하도록 한다. 그리고 최종 출력 중 어느 하나가 1이 되어도 패턴 매칭이 성공한 것이므로 최종 출력들에 대해서 OR 연산을 수행하여 이 패턴에 대한 매칭 출력을 제공한다. 그림 8은 분해되어 설계된 패턴 매칭 회로를 결합하고 시작과 마지막 부분을 추가한 완전한 패턴 매칭회로를 보여준다.

그림 9는 정규표현식 abcder+xyz에 대한 패턴 매칭 회로이다. 이 회로는 이 정규표현식을 abcde, r+, xyz로 분해하여 설계한 후에 결합하여 구성한 것이다. 현재 입력에 대해서 패턴 매칭이 완성될 때에 매칭 출력 M_{out}이 1이 된다.

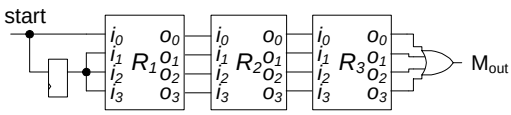

그림 8. 완전한 정규표현식 패턴 매칭 회로

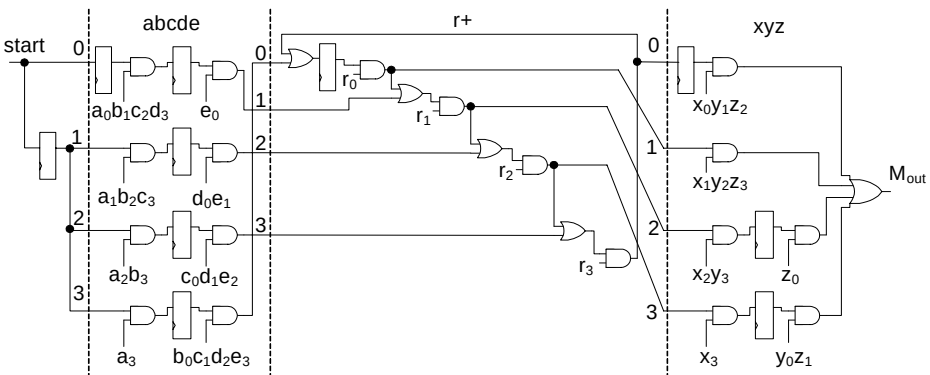

그림 9. 정규표현식 abcder+xyz에 대한 패턴매칭 회로

이 절에서 소개한 4바이트 기반 정규표현식 패턴 매칭 하드웨어는 4대신에 *n*으로 바꾸어서 *n*바이트 기반의 설계로 쉽게 일반화시킬 수 있다. *n*바이트 기반의 패턴 매칭 회로는 입력과 출력이 각각 *n*개 있으며 패턴 매칭 회로들 간의 연결 방법과 시작과 마지막 부분의 구성 방법은 4바이트 기반의 방식과 동일하다.

Ⅳ. 패턴 매칭 회로 생성기 구현

앞에서 제시한 방법을 사용하여 *n*바이트 기반의 정규표현식에 대한 패턴 매칭 회로를 자동적으로 생성해주는 회로 생성기를 Perl과 C언어를 사용하여 구현하였다. 입력은 Snort 규칙의 패턴을 대상으로 하였으며 출력은 입력 패턴에 대한 패턴 매칭 회로를 Verilog로 기술하여 나타내었다. 생성된 Verilog 파일은 FPGA 합성 도구를 사용하여 FPGA로 구현할 수 있다. 회로 생성기의 수행 순서는 다음과 같다.

1. 먼저 Snort 규칙 파일에서 content, uricontent, pcre와 함께 표시된 패턴을 추출한다. snort에서 content와 uricontent와 함께 표시된 고정 패턴은 pcre 형식이 아닌 snort의 표기 형식을 사용하므로 회로 생성기 입력의 일관성을 위하여 content와 uricontent 고정패턴은 모두 pcre 형식으로 변환한다. 예를 들어서 snort 표기 형식의 패턴 ab|00 01 02|pqr은 ab\x00\x01\x02pqr로 변환된다.

2. 추출된 pcre형식의 고정패턴 및 정규표현식 패턴은 기본 토큰 단위로 분리되어 정규표현식 파서로 전달된다. 예를 들어서 \x01과 같은 16진수는 문자 토큰으로, [abc], [^xyz], \d와 같은 표기는 문자군 토큰으로 전달된다. 토큰은 크게 연산자와 문자/문자군의 2가지로 구분된다. 문자군에 대한 토큰을 생성할 때에는 [abc], [a-c], [cba]와 같이 표기는 다르지만 내용이 같은 문자군을 모두 같은 토큰으로 전달하여 공유 접두 패턴을 찾을 때에 이들이 같은 문자군으로 취급되도록 하였다.

3. 정규표현식 파서는 정규표현식 패턴에 대한 토큰들을 입력받아서 파싱을 수행하여 각 입력 패턴에 대해서 그림 4와 같은 구조의 패턴 리스트를 만든다.

4. 공유 접두 패턴을 찾기 위하여 현재 입력된 패턴 리스트를 이전까지 입력된 패턴들로 구성된 패턴 트리와 비교한다. 서로 일치하는 부분은 공유하고, 패턴 리스트의 일치하지 않는 나머지 부분은

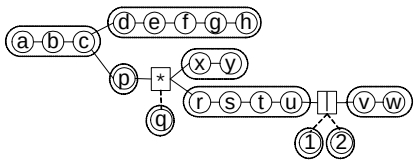

그림 10. 패턴 트리와 고정패턴 그룹핑

패턴 트리의 마지막 공유 노드 다음에 추가한다. 비교 과정에서 문자/문자군 만으로 구성된 고정 패턴 뿐 만 아니라 연산자를 포함한 정규표현식도 공유할 수 있도록 하였다.

5. 모든 입력 패턴들로 구성된 패턴 트리를 방문하면서 *n*바이트 기반 패턴 매칭 회로를 Verilog로 기술하여 생성한다. 패턴 트리는 루트부터 방문하여 자식이 없거나 2개 이상의 자식이나 연산자 자식을 갖는 노드를 만날 때까지의 문자열을 묶어서 고정 패턴의 *n*바이트 단위 패턴 매칭회로를 생성한다. 그 다음에 이 노드의 각 자식을 루트로 하는 서브트리를 같은 방식으로 방문하면서 패턴 매칭 회로를 생성한 후 부모와 자식에 대한 패턴 매칭 회로들의 입출력을 서로 연결한다. 연산자 노드에 대한 구현은 피연산자에 대한 패턴 매칭 회로를 먼저 생성한 후에 연산자를 구현한다.

그림 10은 abcdefgh, abcpq*xy, abcpq*rstu(1|2)vw와 같은 세 개의 패턴에 대한 패턴 트리이며 점선으로 묶은 문자열 단위로 *n*바이트 단위 고정 패턴 매칭 회로가 생성된다.

Ⅴ. 평 가

본 연구에서 한 클록 사이클 동안 패킷 데이터를 *n*바이트씩 검사를 할 수 있게 하는 일반적인 *n*바이트 기반 정규표현식 패턴 매칭 회로 설계 방법을 제시하고 패턴 매칭 회로 생성기를 구현하였다. 생성된 Verilog로 기술된 회로는 재구성이 가능한 FPGA로 합성되어 구현될 수 있다. FPGA에서 회로의 구현은 게이트가 아닌 LUT (lookup table)을 사용하여 이루어지며 LUT의 입력 개수에 제한이 있다. 그러므로 FPGA로 구현된 하드웨어 복잡도는 게이트 수와 차이가 있지만 제시된 하드웨어 구조의 하드웨어 복잡도를 추정하기 위해서 먼저 회로 구현에 필요한 게이트와 플립플롭 개수를 알아보고자 한다.

*n*바이트 기반의 패턴 매칭 회로는 비교 출력을 제공하는 공유디코더 회로와 비교 출력을 사용하여 패턴 매칭 상태 전이를 수행하는 상태 전이 회로로

구성되었다.

먼저 N문자 고정 패턴에 대한 상태 전이 회로를 구현하는 데 필요한 플립플롭 개수를 구해보자. 바이트 0부터 비교를 하는 회로는 문자열 사이에 $\lfloor \frac{N-1}{n} \rfloor$개의 플립플롭이 있고 맨 앞에 1개의 플립플롭이 있다. 바이트 k (k=1~n-1)부터 비교하는 회로는 문자열 사이에만 플립플롭이 있으며 플립플롭 개수는 $\lfloor \frac{N-1+k}{n} \rfloor$개이다. 따라서 N문자 고정 패턴에 대한 상태 전이 회로의 플립플롭 개수 F는 다음과 같다.

$$F = 1 + \sum_{k=0}^{n-1} \lfloor \frac{N-1+k}{n} \rfloor$$

$N-1 = nq+r$ (r=0~n-1)로 놓으면 이 식은 다음과 같이 변형된다.

$$F = 1 + \sum_{k=0}^{n-1} \lfloor \frac{nq+r+k}{n} \rfloor = 1 + nq + \sum_{k=0}^{n-1} \lfloor \frac{r+k}{n} \rfloor$$

$k \geq n-r$인 r개의 k에 대해서 $\lfloor \frac{r+k}{n} \rfloor = 1$이며 나머지 k에 대해서는 $\lfloor \frac{r+k}{n} \rfloor = 0$이므로 플립플롭 개수 F는 다음과 같이 N이 된다.

$$F = 1 + nq + r = N$$

이처럼 고정 패턴에 대한 n바이트 기반 상태 전이 회로의 플립플롭 개수는 n과 무관하며 패턴의 문자수 N과 같다.

정규 표현식 패턴에 대한 상태 전이 회로의 플립플롭 개수는 그림 6에서 보듯이 정규표현식 연산자를 구현할 때에는 플립플롭이 추가되지 않으므로 정규표현식에 있는 연산자를 제외한 문자의 개수와 같다. 예를 들어서 정규표현식 abc(d|ef)+g*r에 대한 상태 전이 회로는 8개의 플립플롭이 포함된다.

다음으로 N문자 고정 패턴에 대한 n바이트 기반 상태 전이 회로에서 사용되는 AND게이트 수를 구해보자. AND게이트는 첫 단계에서 기본적으로 n개가 필요하며, 플립플롭이 추가될 때마다 AND게이트도 함께 추가된다. N문자 고정 패턴에 대한 상태 전이 회로에 있는 N개의 플립플롭 중에서 N-1개의 플립플롭이 문자열 사이에 있으므로 N문자 고정 패턴에 대한 n바이트 기반 상태 전이 회로의 AND 게이트 수는 $N+n-1$개가 된다. AND게이트의 입력은 첫째 단계와 마지막 단계에서는 n개 이하 입력 AND 게이트들도 사용되지만 중간 단계에서는 플립플롭과 n개의 공유 디코더 출력을 입력으로 사용하여 n+1 입력 AND게이트가 사용된다.

정규표현식 패턴에 대한 상태 전이 회로에서의 AND게이트 수는 정규표현식이 m개의 회로로 분리되어 구현되는 경우에 각 회로의 문자수를 N_i라고 하고 문자수의 전체 합을 N이라고 하면 다음과 같이 나타낼 수 있다.

$$\sum_{i=1}^{m} (N_i + n - 1) = m(n-1) + \sum_{i=1}^{m} N_i = N + m(n-1)$$

이처럼 회로가 많이 분리될수록, 다시 말해서 고정 패턴 블록의 문자열 개수가 작을수록 AND게이트수가 증가한다. 예를 들어서 정규표현식 abc(d|ef)+g*r의 패턴 매칭 회로는 먼저 abc, d, ef, g, r의 5개 문자열로 분리하여 구현한 후 연산자를 구현하고 결합하므로 AND게이트 수는 8 + 5×n = 5n + 8이다.

n바이트 기반 패턴 매칭 회로에서 정규표현식의 연산자는 그림 6과 같이 n개의 2입력 OR게이트를 사용하여 구현된다. 따라서 정규표현식 패턴에 대한 상태 전이 회로의 OR게이트 수는 (연산자수)×n이 된다. 여기서 괄호는 연산자 수에 포함되지 않는다. 예를 들어서 정규표현식 abc(d|ef)+g*r은 3개의 연산자 |, +, * 를 포함하므로 이에 대한 n바이트 기반 상태 전이 회로는 3n개의 2입력 OR게이트를 포함한다.

n바이트 기반 패턴 매칭 회로의 공유 디코더는 n개의 문자 디코더로 구성되며 각 문자 디코더의 출력들이 각 바이트에 대한 입력 비교 출력으로 제공된다. 이처럼 공유 디코더는 n에 비례하여 복잡해진다.

패턴 매칭 회로의 최대 클록 속도는 상태 전이 회로에서 각 플립플롭 간의 최대 지연에 영향을 받는다. 상태전이 회로에서 공유 디코더 회로의 입력 비교 출력을 직접 사용하면 공유 디코더의 출력 지연시간이 플립플롭 간의 최대 지연에 포함되므로 최대 클록 속도는 낮아진다. 이를 해결하기 위해서 패턴 매칭 회로 생성기는 입력 비교 출력을 플립플롭을 거쳐서 다음 클록 사이클에서 사용하도록 만들어서 공유 디코더 회로에서의 지연이 최대 클록 속도에 영향을 미치지 않도록 하였다. 이 방법은 플립플롭 수가 증가하지만 FPGA로 구현을 할 때에 LE(logic element)에는 LUT 뿐만 아니라 플립플롭이 포함되어 있으므로 전체 LE 개수에는 거의 영향을 주지 않는다.

위와 같이 공유 디코더를 구현했을 때에 상태 플립플롭 간의 최대 지연은 그림 7(c)와 같이 n개의 1

바이트 입력 비교 회로가 직렬로 연결되는 경우이다. 이때에 n개의 AND게이트와 n개의 OR게이트를 거치게 되어 최대 $2n$게이트의 지연이 발생한다. 이론적으로는 최대 클록 속도가 $1/n$배가 되어 처리 속도에 대한 이점이 없다. 그렇지만 FPGA로 구현할 때에 FPGA의 동작속도 제한 때문에 최대 지연시간이 작은 회로는 이론적인 최대 클록 속도로 동작시킬 수 없는 경우가 많으며 게이트가 아닌 LUT을 사용하여 구현되므로 실제 지연시간은 이보다 작게 증가하여 단위 시간당 처리량은 증가한다.

실제적인 회로 복잡도와 처리 속도를 평가하기 위하여 표 1과 같은 특성을 갖는 snort 규칙을 입력으로 사용하여 패턴 매칭 회로 생성기로 만들어진 Verilog로 기술된 패턴 매칭 회로를 Altera사의 Quartus II 8.0 합성도구를 사용하여 Cyclone II FPGA로 합성하였다. 표 2는 각 snort 규칙의 합성 결과이다. 패턴 매칭 회로는 n=1, 2, 4에 대해서 각각 생성하였다. 표 1에서 구현 문자는 접두 패턴 공유에 따라서 실제로 구현되는 문자수를 나타낸다.

회로의 복잡도를 나타내는 LE 수는 n이 커짐에 따라서 증가하지만 n배 만큼 커지지는 않았다. n=4일 때에 LE 수는 1.62~2.47배가 증가되었다. 고정 패턴 블록 당 문자가 적은 ftp.rule이 다른 규칙보다 LE 수의 증가되는 비율이 컸다.

표 1. 합성에 사용된 규칙의 특성

rule	패턴	문자	구현 문자	연산자	고정패턴 블록수	문자수/ 고정패턴
ftp.rules	54	451	320	80	142	2.25
sql.rules	66	1142	896	11	103	8.70
web-iis.rules	104	1580	1254	14	152	8.25
web-php.rules	168	2334	1898	46	299	6.35

표 2. 합성 결과

rules	n	LE	freq	LE/문자	Gbps	LE비	속도비
ftp.rules	1	475	344.71	1.05	2.76	1.00	1.00
	2	713	292.83	1.58	4.69	1.50	1.70
	4	1175	225.89	2.61	7.23	2.47	2.62
sql.rules	1	1067	352.86	0.93	2.82	1.00	1.00
	2	1257	323.52	1.10	5.18	1.18	1.83
	4	1725	256.94	1.51	8.22	1.62	2.91
web-iis.rules	1	1530	346.62	0.97	1.88	1.00	1.00
	2	1738	318.27	1.10	3.21	1.14	1.84
	4	2557	237.02	1.51	5.30	1.67	2.74
web-php.rules	1	2309	254.26	0.99	1.77	1.00	1.00
	2	2760	245.58	1.18	3.24	1.20	1.93
	4	4047	215.98	1.73	5.64	1.75	3.40

최대 동작 속도(freq)는 snort 규칙에 따라서 차이가 있으며 n이 커짐에 따라서 감소하지만 n배만큼 감소하지는 않았으며 n=2일 때에 1.04~1.18배가, n=4일 때에 1.18~1.53배가 감소하였다. 단위 클록당 처리되는 데이터량인 처리속도(Gps)는 n=2일 때에 1.70~1.93배가, n=4일 때에 2.62~3.40배가 증가하였다. 그리고 n이 증가할 때 처리 속도의 증가 비율이 LE 수의 증가비율보다 더 크다.

Ⅵ. 결 론

본 연구에서 NIDS의 성능에 중요한 연산인 다중 패턴 매칭을 다중바이트 기반으로 처리하는 정규표현식 패턴 매칭 하드웨어 구조 및 설계 방법을 제시하였고 일반적인 n바이트 기반 패턴 매칭 회로 생성기를 개발하였다. 그리고 제시된 회로의 복잡도를 분석하고 snort 규칙에 대한 패턴 매칭 회로를 FPGA로 합성하여 구현 회로에 대한 성능과 복잡도를 평가를 하였다.

n바이트 기반 패턴 매칭 회로는 공유 디코더 회로의 자원은 n에 비례하여 증가하지만 상태 전이 회로에 대한 자원은 n이 증가할 때에 증가하지만 n에 정비례하지는 않는다. 그리고 구현 단위가 되는 고정 패턴 당 문자수도 게이트 수에 영향을 미친다. 상태전이 회로의 플립플롭 개수는 n과 무관하다. 그리고 한 클럭 사이클에 n바이트 씩 처리함으로써 처리 속도를 향상시킬 수 있다. n이 증가됨에 따라서 지연 시간 증가로 인해서 최대 클록 속도가 감소될 수 있지만 감소되는 정도는 n배보다 작으므로 처리 속도는 향상된다. FPGA 합성 결과의 처리 속도 증가 비율이 소요되는 LE 수의 증가 비율보다 더 크므로 제시된 구조의 회로는 성능 향상에 효과적이라고 판단된다.

참 고 문 헌

[1] Snort web site, http://www.snort.org

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

[3] Bleeding Edge Threats, http://bleedingthreats.net

[4] R. Sidhu and V.K. Prasanna, "Fast Regular Expression Matching using FPGAs", *IEEE Symp. on Field-Programmable Custom Computing Machines*, 2001, pp.227-238

[5] B. L. Hutchings, R. Franklin, and D. Carver, "Assisting network intrusion detection with

reconfigurable hardware", *IEEE Symp. on Field- Programmable Custom Computing Machines*, 2002, pp.111-120

[6] C.-H. Lin, C.-T. Huang, C.-P. Jiang, and S.-C. Chang, "Optimization of regular expression pattern matching circuits on FPGA", *Conf. on Design, Automation and Test in Europe* (DATE06), 2006, pp.12-17

[7] B.C. Brodie, D.E. Taylor, and R.K. Cytron, "A scalable architecture for high-throughput regular-expression pattern matching", *Comput. Architecture News*, Vol.34, No.2, 2006, pp.191-202.

[8] J. Bispo, I. Sourdis, J. Cardoso, and S. Vassiliadis, "Regular expression matching for reconfigurable packet inspection", *IEEE Conf. Field Programmable Tech.* (FPT06), 2006, pp.119-126.

[9] C.-H. Lin, C.-T. Huang, C.-P. Jiang, and S.-C. Chang, "Optimization of pattern matching circuits for regular expression on FPGA", *IEEE Trans. VLSI Systems*, Vol.15, No.12 Dec. 2007, pp.1303-1310.

[10] J. Lee, S.H. Hwang, N. Park, S.W. Lee, S. Jun, and Y.S. Kim, "A high performance NIDS using FPGA-based regular expression matching", *Symp. Applied Computing* (SAC2007), 2007, pp.1187-1191.

[11] C.R. Clark and D.E. Schimmel, "Scalable parallel pattern matching on high-speed networks", *IEEE Symp. on Field-Programmable Custom Computing Machines*, 2004. pp.249-257

[12] I. Sourdis and D. Pnevmatikatos, "Pre-decoded CAMs for Efficient and High-Speed NIDS Pattern Matching", *IEEE Symp. on Field- Programmable Custom Computing Machines*, 2004, pp.258-267

[13] PCRE-Perl Compatible Regular Expressions, http://www.pcre.org

윤 상 균 (SangKyun Yun) 종신회원

1984년 2월 서울대학교 전자공학과 졸업
1986년 2월 한국과학기술원 전기 및 전자공학과 석사
1995년 8월 한국과학기술원 전기 및 전자공학과 박사
1984년~1990년 현대전자 정보사업본부
1992년~2001년 서원대학교 전자계산학과 교수
1998년 1월~1999년 1월 University of Michigan, Ann Arbor, Visiting Scholar
2007년 9월~2008년 8월 University of Texas at Austin, Visiting Scholar
2001년 9월~현재 연세대학교 과학기술대학 컴퓨터정보통신공학부 교수
<관심분야>컴퓨터 및 네트워크 시스템, 임베디드 시스템, NIDS

이 규 희 (KyuHee Lee) 준회원

2007년 2월 연세대학교 컴퓨터정보통신공학부 컴퓨터공학전공 졸업
2007년 3월~현재 연세대학교 전산학과 석사과정
<관심분야> NIDS, 임베디드시스템