DOI: 10.3745/KIPSTC.2009.16C.5.583

FPGA를 이용한 효율적 정규표현매칭

이 장 행[†] · 이 성 원[††] · 박 능 수[†††]

요 약

Network Intrusion Detection System(NIDS)는 네트워크를 통해 들어오는 패킷들을 모니터링 하고 분석하여 내부 시스템에 유해한 내용을 담고 있는 패킷을 탐지 하는 시스템이다. 이 시스템은 네트워크의 패킷을 놓치지 않고 분석할 수 있어야 하며, 예측 불허의 공격 방법들에 대해서는 새로운 법칙을 적용하여 방어할 수 있어야 한다. 이에 대응하여, 소프트웨어적 처리에 비해 높은 비교 성능과 재구성이 가능한 유연성을 제공하는 FPGA는 좋은 해결책이다. 그럼에도 불구하고, 고속 네트워크의 등장과 축적되는 공격 패턴들의 증가는 제한된 속도와 공간을 가지고 있는 FPGA에게 부담이 된다. 본 연구는 추가적인 자원 사용을 최소화하고 성능의 극대화를 가져오는 방식으로 접두어 공유 병렬 패턴매치 기법을 제시하고 설계하였다. 실험을 통하여 입력 문자열을 8bit에서 16bit로 증가할 때 성능이 두 배 향상이 되면서 구현을 위해 사용되는 자원은 평균 1.07배 증가하는 것을 확인할 수 있다.

키워드 : 네트워크 침입탐지시스템, 정규표현식, FPGA, 문자열비교

Efficient Regular Expression Matching Using FPGA

Janghaeng Lee[†] · Seong-Won Lee[††] · Neungsoo Park[†††]

ABSTRACT

Network intrusion detection system (NIDS) monitors all incoming packets in the network and detects packets that are malicious to internal system. The NIDS should also have ability to update detection rules because new attack patterns are unpredictable. Incorporating FPGAs into the NIDS is one of the best solutions that can provide both high performance and high flexibility comparing with other approaches such as software solutions. In this paper we propose and design a novel approach, prefix sharing parallel pattern matcher, that can not only minimize additional resources but also maximize the processing performance. Experimental results showed that the throughput for 16-bit input is twice larger than for 8-bit input but the used LEs/Char in FPGA increases only 1.07 times.

Keywords : NIDS, Regular Expression, FPGA, String Matching

1. 서 론

컴퓨터분야에 네트워크의 출현 이후 데이터 통신 기술이 발달하면서 네트워크의 보안의 중요성은 점점 높아져 갔다. 특히 네트워크를 통한 시스템 침입 시도는 시간이 흐를수록 점점 다양해지고 복잡해져 가고 있다. 따라서 시스템의 보안을 위하여 컴퓨터 시스템으로 들어오는 네트워크 패킷들의 내용을 분석하여 침입 여부를 판단하는 네트워크 침입탐지 시스템(Network Intrusion Detection System: NIDS)이 제안되었다. NIDS의 성능은 전적으로 패턴매칭 기술에 좌우되는데, 이는 네트워크를 통하여 전달되는 수없이 많은 패킷들의 내용까지 매칭을 시도하기는 때문이다. 하지만 비약적인 소프트웨어적 패턴매칭 알고리즘의 발전에도 불구하고 네트워크의 속도가 급격히 발전에 가면서 소프트웨어만으로 실시간 처리하기는 어려워졌다. 최근의 경향을 보더라도 이미 Gbps급 네트워크 인터페이스 카드가 보급되고 있는 현실에서 NIDS가 네트워크 속도를 따라잡지 못하여 잠재적으로 공격들을 막을 수 있는 법칙들을 비활성화 시켜야 하는 경우가 발생하기도 한다.

이에 대한 해결책으로 NIDS중 부하가 가장 많이 걸리는 패턴매칭 역할을 하드웨어에 전가시키는 것이다[1-3]. 하드웨어의 방식에서 ASIC을 이용한 방법은 그 성능 면에서 뛰어나나 차후 새로운 공격방식에 대한 대응이 어렵다는 단점이 있다. 따라서 새로운 공격이 발견될 때 마다 이에 대응하는 탐지방식을 재구성을 할 필요가 있다. 본 연구에서는 FPGA(Field Programmable Gate Arrays)를 이용한 방법을 제시하고자 한다. FPGA는 고정된 하드웨어(ASIC등)를 사용하는 것에 비해 재구성 할 수 있는 특성을 바탕으로 자주 변화되는 패턴매칭에 최적화된 하드웨어적 알고리즘을 적용

※ 이 논문은 2006년도 건국대학교 학술진흥연구비 지원에 의한 논문임.
† 준 회 원 : 조지아공대 컴퓨터공학과 석사
†† 종신회원 : 광운대학교 컴퓨터공학과 교수
††† 종신회원 : 건국대학교 컴퓨터공학부 부교수(교신저자)
논문접수 : 2009년 7월 6일
수 정 일 : 1차 2009년 8월 5일
심사완료 : 2009년 8월 18일

할 수 있고 하드웨어적 특성을 바탕으로 소프트웨어 기반의 NIDS에 비해 월등한 성능을 발휘할 수 있다. 그러나 현재 네트워크의 성능은 Gbps에 도달하고 있고, FPGA의 내부 속도는 이보다 떨어져 있고, 또한 새로이 증가하는 공격 패턴들을 한정된 공간의 FPGA에 적용시켜야 하는 문제도 FPGA를 이용한 NIDS 구현의 쟁점 중에 하나이다.

본 논문에서는 SNORT[4] 규칙의 문법들을 지원할 수 있도록 정규표현식의 매칭을 하면서, 저비용으로 성능을 극대화 시킬 수 있도록 접두어 공유 병렬 패턴매칭(Prefix Sharing Parallel Pattern Matching) 회로를 구성하는 것이다. 제안된 방식은 제한된 내부 속도를 가지고 있는 FPGA의 단점을 극복하기 위해 하나의 클럭 사이클 동안 여러 개의 문자를 병렬로 비교하여 성능을 극대화 시키고 동시에 제한된 공간의 FPGA 리소스 사용량을 최소화 하도록 패턴들마다 중복되는 접두어(Prefix)를 공유하여 구현 비용을 최소화 할 수 있다. 본 논문에서 제안한 방식에 대하여 6종류의 SNORT 규칙에 대하여, 한 번의 클럭 동안 1개와 2개의 문자비교를 시도하는 회로들을 작성하고 이들 간의 성능과 리소스 사용량을 측정하도록 실험하였다. 실험 결과, 4.4Gbps의 스루풋(throughput)에도 평균 0.68 LEs/char의 비용 밖에 소모되지 않는 것을 확인 하였고, 1개의 문자 비교방식과 비교하여 2배의 스루풋을 내는 병렬회로에서의 비용은 약 1.07배인 것으로 확인하였다

본 논문은 2 장에서는 선행된 관련연구에 관하여 기술하고 3장에서 효율적이고 고성능의 패턴 비교를 수행하는 회로 설계에 대하여 설명하였다. 4장은 패턴회로의 생성 알고리즘에 대해 설명하고 5장은 성능평가에 대해 기술하였다. 그리고 6장에서 결론을 맺는다.

2. 관련연구

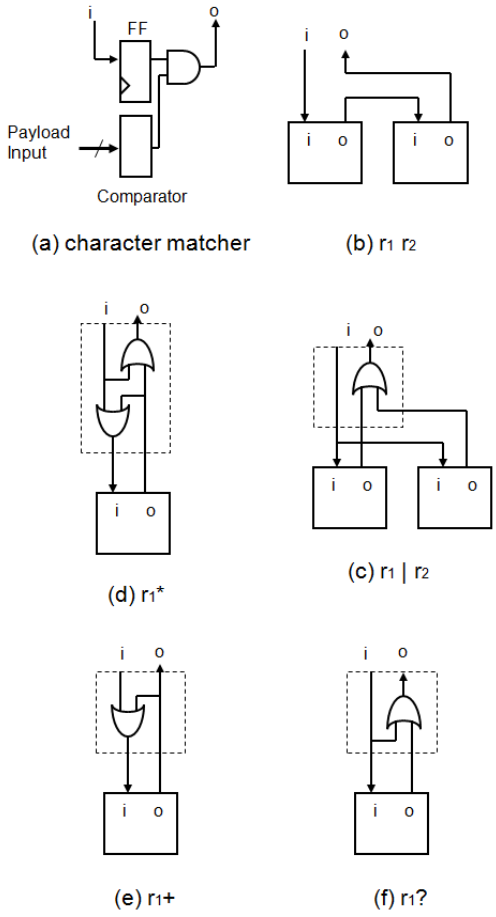
2.1 침입탐지를 위한 정규표현

NIDS의 SNORT 규칙은 두 가지 종류로 공격 패턴들을 정의한다. 첫 공격패턴은 콘텐트 필드를 사용하여 단순한 연속문자열로 정의하는 것으로서 전체 정의의 약87%를 차지한다. 다른 하나는 보다 복잡한 PCRE(Perl Compatible Regular Expression)로 정의된 필드를 사용하는 것이다. <표 1>은 SNORT 규칙의 예와 PCRE필드의 구성 예를 보여주고 있다. 이러한 정규표현식을 NFA(Non-deterministic Finite Automata)로 표현하고 이에 대응하는 회로로 표현할 수 있다[5].

NFA를 사용한 회로 설계는 한 클럭에 한 바이트 비교를 수행할 수 있도록 파이프라인 구조의 문자비교기들로 구성되어 있다. 문자비교기는 비교를 위한 조합회로와 비교결과 저장을 위한 1-bit의 저장장소로 이루어져 있다. 조합회로는 현재 들어오는 입력이 하드웨어적으로 프로그램된 문자코드와 같은지 비교하고, 저장장소는 이전 단계의 출력으로 현재까지 비교결과를 나타낸다. (그림 1)은 정규표현식을 대응하는 패턴매칭 회로의 예를 보여주고 있다. 이 방식은 정

〈표 1〉 SNORT 규칙의 예

alert tcp $EXTERNAL_NET any → $HOME_NET 21(msg:"FTP EXPLOIT STAT * dos attempt"; … pcre/^STAT\s+[^\n]*\x2a/smi"; …)	
Protocol	TCP
Source Addr	외부 네트워크
Source Port	Any
Destination Addr	내부 네트워크
Destination Port	21
Message	FTP EXPLOIT STAT * dos attempt
Condition	"STAT[공백문자1개이상]" 이후 엔터문자가 입력되기 전 헥사코드 0x2a값이 있는 조건
…	



(그림 1) 정규표현식에 대응하는 회로

규표현식에 대응하여 쉽게 문자 비교를 시도할 수 있게 되었다.

2.2 공유 디코더를 이용한 패턴매칭

패턴들에 대한 회로들을 구성할 경우, 비교는 최소 1 byte 단위로 할 것이므로 8bit 넓이의 입력은 각 패턴을 구성한 회로들의 비교기들로 전달된다. 문자열 입력과 프로그램 된 문자의 비교결과가 일치하는지 안 하는지의 결정을 내기 위해 각 비교기는1-bit의 정보만 출력 한다. 하지만

1-bit 정보를 위해 8 bit의 문자입력이 각 문자 매치기(match unit)로 전달되는 것은 많은 라우팅 패스를 발생시켜 비효율적이다. 따라서 각 매치기마다 8 bit의 비교기를 두는 대신 중앙에서 어떤 문자입력이 되는지를 판단하고 그 결과를 1 bit로 전달하도록 하여 훨씬 효율적으로 구성될 수 있다. 모든 비교를 하기 위해 중앙에 8 bit 디코더를 설치하고 디코더로부터 출력되는 256개의 결과 중 하나를 선택하여 문자매치기로 전달시키는 방식으로 공유 디코더 방식[6]이라 한다. 결과적으로 각 문자매치기의 비교기를 생략함으로써 보다 많은 리소스를 절약할 수 있다. 하지만 아직도 낭비되는 자원들은 여전히 존재 한다. SNORT 규칙에는 다양하고 많은 패턴들이 정의되어 있지만 이들 중 상당수의 접두어가 중첩된다. 중첩되는 문자열마다 회로를 각각 중복되게 구성하는 것은 자원의 낭비이다. 본 논문에서는 이러한 접두어를 최대한 공유할 수 있도록 회로를 구성하여 리소스 사용을 최소화하고자 한다.

2.3 병렬 비교기법

공유 디코더 방식은 자원을 많이 절약할 수 있고 FPGA의 크기에 대하여 효율적으로 구현이 가능해졌다. 하지만 FPGA는 내부 클럭 속도에 많은 제한을 가지고 있고 이는 스루풋을 제한하게 된다. 이런 제한을 극복하기 위해 한 클럭 사이클 동안 1개의 문자비교가 아닌 여러 문자를 동시에 비교하는 연구들이[6, 7] 진행되어 왔다. 병렬 비교기법은 문자열 비교에서 한 클록 동안에 두 개 이상의 문자를 동시에 비교하는 방식이므로 다수 문자 입력에서 문자열의 시작점이 다르기 때문에 이를 고려한 상태도를 작성하여 비교하여야만 한다. 기존의 방법은 복잡한 정규표현식이 아닌 단순 연속 문자열의 병렬 비교에 국한되어 있어 SNORT 규칙의 복잡한 정규표현식에 대한 지원이 필요하다. 또한 병렬 비교기법을 위한 병렬패턴을 새로이 생성하여 자원의 증가를 야기한다. 따라서 단순한 병렬패턴 매치의 경우 제한된 FPGA 자원을 이용할 경우 그 구성할 수 있는 패턴의 숫자를 줄어드는 단점이 발생한다. 본 논문에서는 복잡한 정규표현식을 병렬로 비교하면서 그 비용을 줄여 효율적으로 구현할 수 있는 방식을 제시하고자 한다.

3. 접두어 공유 병렬 패턴매칭기법

3.1 정규표현식 병렬 패턴매치기

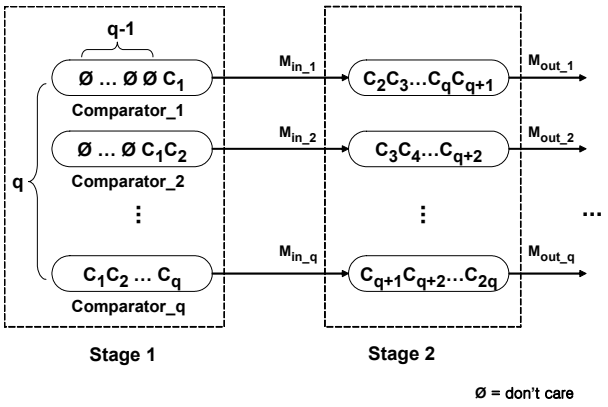
정규표현식을 비교하기 위한 NFA 기반의 패턴매치 방식은 파이프라인의 각 단계에서 문자 비교가 수행되고 그 비교 결과가 다음 단계로 전달되는 구조를 갖는다. 대응하는 병렬 비교 회로를 구성하기 위해, 첫 문자부터 연속된 문자열을 읽어 파이프라인 회로를 구성하되, 이 때 정규표현식의 괄호, |, +, ? 등이 나타나면 그때까지의 문자열 단위 파이프라인의 구성을 종료하고 다음 부분의 문자열 파이프라인을 구성하도록 한 후, 문법의 우선순위에 따라 각 문자열 파이프라인을 연결하도록 한다. 예를 들어, 정규표현식 ab(cdef|g)h* 은 ab, cdef, g, 그리고 h문자열들에 대해 각각 파이프라인으로 구성된 후 결합한다.

이러한 순차적 문자열 비교를 q개 문자로 구성된 한 문자열을 1 클럭 사이클 동안 병렬로 비교하는 경우로 바꾸어 생각해 보자. 이때 q개의 문자 중에서 하나의 문자열 패턴이 시작되는 곳은 문자열 패턴이 밀림 없이 매치되는 경우와 한 문자가 밀린 경우에서부터 q-1개의 문자가 밀리는 경우까지 총 q개의 경우를 고려해야 한다. 첫 번째 파이프라인 단계에서는 밀려서 들어오는 경우 선행되었던 문자들에 대한 정보가 없으므로 고려할 필요가 없다. 두 번째 파이프라인 단계에서 부터는 새롭게 비교하여야 할 q개 문자정보뿐만 아니라 이전 단계에서 밀렸던 q-1개의 문자들도 필요로 한다. 그러므로 각 파이프라인 단계에서는 2q-1개의 비교 문자정보가 필요하다. (그림 2)는 파이프라인 단계에서 구성된 문자비교기를 표현하고 있다. 여기서 Mout_N은 N번째 비교기의 비교 결과 값이 출력되는 신호를 의미하고, Min_N은 N번째 비교기로 입력되는 이전 단계의 비교 결과 신호를 의미한다.

(그림 2)와 같이 파이프라인을 구성하기 위해 연속문자열의 앞뒤로 적절히 don't care(Ø)를 추가시켜야 한다. 문자열의 첫 파이프라인에서 q-1개의 문자열 패턴이 밀리는 경우를 고려할 때 문자열의 접두부에는 q-1개의 don't care(Ø)를 붙여야 한다. 또한, 가장 마지막 파이프라인 역시 첫 글자에서 패턴이 종결된 경우 나머지 q-1개 문자열의 비교가 완료 되는 경우까지 고려하여 연속문자열의 접미부에 don't care(Ø)를 붙여야 한다. 추가로 덧붙여야 할 접미부의 don't care(Ø)문자 개수는 다음과 같다.

$$postfix\_length = q - (string\_length \bmod q)$$

한 예로, 정규표현식 ab(cdef|g)h*과 병렬도 q=2 경우를 가정하자. 수정된 부분 문자열(substring)을 집합으로 표현하면 {ØabØØ, ØcdefØØ, ØgØ, ØhØ}이다. 부분 문자열 수정이 완료되면 첫 번째 don't care(Ø)의 위치부터 q개의 문자를 바탕으로 파이프라인의 비교기 한 개를 정의하고 한 문자씩 포인터를 오른쪽으로 이동하며 이 과정을 반복한다.



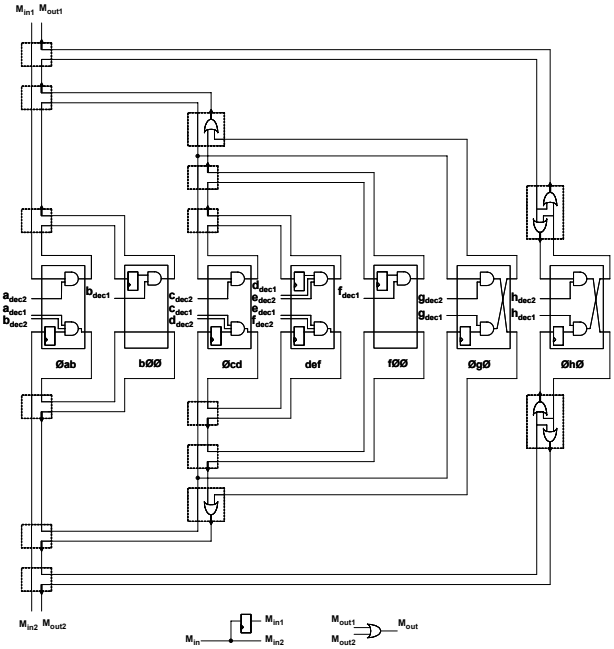
(그림 2) 파이프라인 병렬회로

이때 포인터가 q번 이동하면 하나의 파이프라인 단계 그룹이 완성된다.

연속문자열 단위로 파이프 라인을 구성하는 구조에서는, 비교가 종료된 연속문자열이더라 하더라도 다음 연속문자열의 첫 파이프라인에서 이어서 비교를 시도할 수 있다. 하지만, 다음문자열의 첫 파이프라인은 이전 문자열 마지막 파이프라인의 마지막 q-1개의 문자정보를 알 필요가 없다. 이미 이전문자열의 종결 파이프라인의 비교 결과 M_{out_N}이 다음문자열 첫 파이프라인의 M_{in_M}으로 입력되므로 M_{in_M}값과 현재의 문자입력 값으로 연속하여 비교를 시도할 수 있다. 여기서 유의해야 할 점이 두 가지가 있는데, 하나는 실제적으로 한 단계가 저장되려면 정확히 q개의 문자가 전부 비교가 완료 된 후여야 한다는 것이다. 각 비교기의 첫 번째 입력이 don't care(Ø)인 경우는 아직 q개의 비교가 완료되지 않은 것을 의미하므로 상태 저장(플립플롭) 없이 바로 다음 단계의 파이프라인으로 전달되어야 한다.

파이프라인 간 입출력 연결 시 추가로 유의할 점은 M_{out_N}에서의 N과 다음단계 파이프라인의 M_{in_M}에서의 M이 항상 일치하지 않는다는 것이다. 즉, 이전 파이프라인의 출력 선이 다음 파이프라인의 입력 선으로 나란히 매핑 되지 않을 수 있다. 출력 선들은 어떤 일정한 법칙을 가지고 어느 만큼씩 꼬이게 된다. 꼬이는 조건은, 그 파이프라인이 연속문자열의 종결 파이프라인인지 여부 이다. 종결 파이프라인은 두 번째 구성요소의 마지막 입력이 don't care(Ø)인 특징을 가지고 있다. 종결 파이프라인으로서 꼬임의 조건을 충족하면, 얼마만큼 전체 M_{out}을 이동시킬 것인지는 첫 번째 구성요소의 접미부 don't care(Ø)개수에 의해 결정된다. 만약, 첫 번째 구성요소의 접미부 don't care(Ø)개수가 N개이면 M_{out_1}은 다음단계 파이프 라인의 M_{in_N}과 매핑된다 (단, N=0 이면 M_{in_q}로 매핑된다). 나머지 M_{out}들은 첫번째 M_{out_1}이 이동된 거리에 따라 똑 같은 크기만큼 이동을 하게 된다.

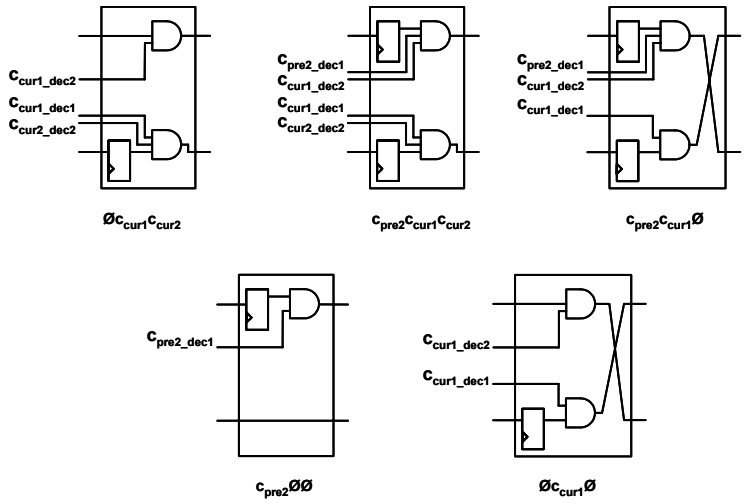
정리하면, q=2 인경우, 각 파이프라인에 구성될 수 있는 문자 비교기의 입력 문자집합은 {Ø$c_{cur1}c_{cur2}$, $c_{pre2}c_{cur1}c_{cur2}$, $c_{pre2}c_{cur1}$Ø, c_{pre2}ØØ, Øc_{cur1}Ø} 이다. 여기서 c는 공유 디코더로부터 비교되어 출력된 문자를 의미하고 cur는 현재 파이프라인, pre는 이전 파이프라인을 뜻하며, 숫자는 n번째 문자 입력을 뜻한다. (그림 3)은 q=2 일 때, 입력 문자에 대응 하는 비교기 회로를 보여주고 있으며 마지막 decN 은 공유 디코더 번호를 의미한다. (그림 4)는 정규표현식 ab(cdef|g)h* 가 최종적으로 회로로 구성된 모습을 보여주고 있다.



(그림4) 정규표현식 ab(cdef|g)h* 에 대응하는 회로

3.2 접두어 공유기법

Snort 규칙의 내용을 보면 검사할 패턴들 중 많은 수의 접두어가 중첩된다. (그림 5)는 oracle 규칙파일의 내용 중 일부로 "dbms_repcat_alter_m"의 문자열이 여러 패턴에서



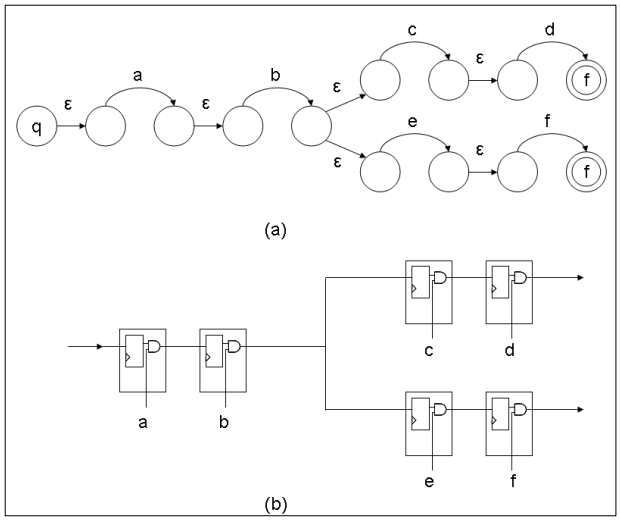
(그림 3) q=2일 때 입력 문자 비교기 회로

정의되어 있는 것을 보여주고 있다. 이렇게 중첩되는 문자들을 그대로 패턴마다 회로로 구성하는 것은 자원 낭비가 된다. NFA에 대응하는 회로를 기반으로, 중첩된 접두 문자열은 추가적인 조건 없이 상태의 공유가 가능하다. (그림 6)은 중첩된 접두 문자열을 공유하는 NFA와 회로도를 보여주고 있다.

```
alert tcp $EXTERNAL_NET any -> $SQL_SERVERS
$ORACLE_PORTS (msg:"ORACLE … content:
"dbms_repcat.alter_mview_propagation";…)

alert tcp $EXTERNAL_NET any -> $SQL_SERVERS
$ORACLE_PORTS (msg:"ORACLE … content:
"dbms_repcat.alter_master_repobject";…)
```

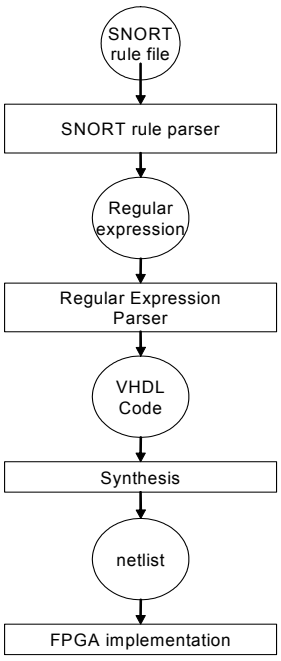
(그림 5) 패턴들의 접두 문자열 중첩 예제



(그림 6) 접두어를 공유하는 (a)NFA 와 (b) 회로도

3.3 패턴회로 자동생성기

SNORT 규칙들에 대한 회로를 생성하고 FPGA에 적용시키기 위해 먼저 SNORT 규칙 파일에서 content 필드, uricontent 및 pcre필드의 내용들을 추출하여 모두 정규표현식으로 출력되도록 SNORT 규칙 파서를 제작하였다. 그 후 정규표현식들을 바탕으로 VHDL 코드를 생성하는 정규표현식 파서를 제작하였다. VHDL코드 생성에 있어 가장 중요한 부분은 배선(routing)이다. 코드생성을 최대한 간략히 수행하기 위해 순차논리회로와 조합회로들은 가능한 한 블록으로 미리 구성하도록 하고 그 블로간 신호를 포트 매핑하는 구조 모델링 방식을 이용하여 VHDL 코드를 생성하였다. 본 연구에서는 컴파일러 제작 툴인 Lex 와 Yacc을 사용하여 SNORT 규칙 파서 및 정규표현식 파서를 제작하였다. 모든 회로의 생성과 FPGA에 구현의 단계까지 모든 과정은 (그림 7)과 같은 단계로 자동화하였다.



(그림 7) 회로 자동생성과정

4. 실험 및 결과

본 연구에서는 SNORT 규칙을 파싱하여 비교를 시도할 패턴들을 추출하고 접두어 공유를 기반으로 8bit(q=1)과 16bit(q=2) 입력의 경우에 대하여 2종류의 파서를 제작하였다. 따라서 같은 규칙을 기반으로 병렬처리를 하였을 때 얼마나 많은 자원을 더 사용하는 지와 FPGA의 최대 클럭 주파수가 얼만 큼 줄어드는지 관찰하였다. 성능 평가는 Pentium 4 3.4Ghz 의 Processor, 1GB RAM 의 사양을 가지고 있는 시스템에서 이루어 졌으며, ALTERA 사의 Quartus II 4.2를 합성 소프트웨어로, Cyclone EP1C12Q Device를 타겟 디바이스로 사용하였다. <표 2>는 실험에 사용된 SNORT 규칙의 종류 규칙별 패턴 개수, 그리고 문자 수 등의 정보를 정리하였다.

<표 3>은 실험결과로 LE(Logic Element)의 수는 q개의 공유 디코더와 인코더에 대한 LE도 포함되어 있다. 실험을 통하여 q=1 인 경우와 q=2 인 경우 같은 속도로 동작을 하여 스루풋은 정확히 2배의 성능 차이를 보이지만, 사용된 LE의 양은 약 1.07배에 불과하다. q에 따라 사용된 LE 개수

〈표 2〉 사용된 SNORT 규칙의 정보

Rule	# of Patterns	# of Characters	# of odd Sharings	# of ORs
ftp	151	677	8	0
*oracle	613	10,764	122	0
smtp	152	1,824	15	22
sql	61	1,140	4	2
web-iis	144	2,118	54	2
web-php	205	2,752	73	20
total	1,326	19,275	276	46

*Oracle 규칙은 pcre 옵션을 제외하고 구현됨.

〈표 3〉 실험 결과

Rule	8bit(q=1)		16bit(q=2)		증가율
	LEs	LEs/Char	LEs	LES/Char	
ftp	594	0.88	740	1.09	0.22
*oracle	5,008	0.47	5,405	0.50	0.04
smtp	1,530	0.84	1,690	0.93	0.09
sql	604	0.53	702	0.62	0.09
web-iis	1,885	0.89	1,141	1.01	0.12
web-php	2,123	0.77	2,395	0.87	0.10
total	11,744	0.61	13,073	0.68	0.07

가 변하는 것은 OR 개수와 공유되는 prefix개수에 영향을 받기 때문이다. 그 이유는, q=1일 때는 OR-gate가 필요 없지만 q=2 이상일 경우 패턴들 마다 마지막 파이프라인에서 출력되는 q개의 match 신호에 최종적으로 1개의 LE를 소모하는 OR-gate를 추가되기 때문이다. 그러나 접두어 공유로 인하여 oracle의 경우에서 보듯이 많은 공유가 발생할 경우에 그 비용의 증가가 상대적으로 적게 증가하는 것으로 분석된다.

5. 결 론

본 논문에서는 NIDS의 문자열 패턴매칭을 분석하고, FPGA를 사용하여 적은 리소스를 사용하고도 높은 스루풋을 내는 접두어 공유 병렬 패턴매칭 기법을 제시하였다. 본 논문이 제시한 방법은 한 클럭 사이클 동안 여러 개의 문자들을 입력 받아 동시에 비교를 수행하여 성능을 높이는 것이다. 또한, 최대한의 접두어를 공유하여 리소스 사용을 최소화 하였다. 실험을 통해 SNORT 규칙을 구현한 결과 패턴에 정의된 문자들의 개수보다 적은 평균 0.68 LEs/Char의 자원을 사용되는 것을 확인 할 수 있었으며, 병렬 비교를 통한 스루풋의 증가를 확인하였다. 또한 입력의 문자열을 8bit에서 16 bit로 병렬 비교를 증가하여도 1.07 배의 자원의 사용 증가가 발생하는 것을 확인하였다.

감사의 글

본 연구 내용의 일부는 The 22nd Annual ACM Symposium on Applied Computing (SAC '07) 학술대회에 발표되었습니다. 또한 이 논문의 완성에 조언을 하여주신 황성호, 전성익, 김영수님에게 감사드립니다.

참 고 문 헌

[1] Ioannis Sourdis and Dionisios Pnevmatikatos, Pre-decoded CAMs for Efficient and High-Speed NIDS Pattern Matching, In Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 258-267, April, 2004.

[2] James Moscola, John Lockwood, Ronald P. Loui, and Michael Pachos, Implementation of a Content-Scanning Module for an Internet Firewall, In Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machine, Apr., 2003.

[3] Y.Cho and W. Mangione-Smith, Deep Packet Filter with Dedicated Logic and Read Only Memories, In Proceedings of the 12th Annual IEEE Symposiumon Field-Programmable Custom Computing Machines, Apr., 2004.

[4] Martin Roesch and Chris Green, Snort User's Manual. http://www.snort.org/docs/writing_rules

[5] R. Sidhu and V.K. Prasanna, Fast Regular Expression Matching using FPGA, In Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01), Apr., 2001.

[6] Christopher R. Clark and David E. Schimmel, Design of Efficient FPGA Circuits for Matching Complex Patterns in Network Intrusion Detection Systems. In Proceedings of the 13th International Conference on Field Programmable Logic and Applications, June, 2003.

[7] Christopher R. Clark and David E. Schimmel, Scalable Pattern Matching for High Speed Networks, In Proceedings of the 12th Annual IEEE Symposiumon Field-Programmable Custom Computing Machines, Apr., .2004

이 장 행

e-mail : janghaeng@gatech.edu
2006년 건국대학교 컴퓨터공학과(학사)
2009년 Georgia Institute of Technology (석사)
관심분야 : Dynamic Compiler, Parallel Computing 등

이 성 원

e-mail : swlee@kw.ac.kr
1988년 서울대학교 제어계측공학과(공학사)
1990년 서울대학교 제어계측공학과(공학석사)
2003년 미국 University of Southern California 전기공학과(공학박사)
1990년~2004년 삼성전자 책임연구원
현 재 광운대학교 컴퓨터공학과 교수
관심분야 : 미디어프로세서, SOC설계, Power-Aware Computing, 영상 신호처리 등

박 능 수

e-mail : neungsoo@konkuk.ac.kr
1991년 연세대학교 전기공학과(학사)
1993년 연세대학교 전기공학과(석사)
2002년 미국 University of Southern California, 전기공학과(컴퓨터공학) (공학박사)
2002년~2003년 삼성전자 책임연구원
2003년~2007년 건국대학교 컴퓨터공학부 조교수
2007년~현 재 건국대학교 컴퓨터공학부 부교수
관심분야 : 컴퓨터구조, 임베디드 시스템, 병렬시스템, 멀티미디어 컴퓨팅, 컴퓨터보안 등