FPGA를 사용한 네트워크 침입탐지 시스템의 문자열 비교†

이장행* 황성호 박능수
건국대학교 컴퓨터공학과
{janghaeng, sungho, neungsoo}@ivlab1.konkuk.ac.kr

String matching for Network Intrusion Detection System using FPGA

Jang Haeng Lee* Sung Ho Hwang Neungsoo Park
Department of Computer Engineering, Konkuk University, Seoul

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 안에서 돌아다니는 패킷을 놓치지 않고 분석할 수 있어야 하며, 예측 불허의 공격 방법들에 대해서는 새로운 법칙을 적용하여 방어할 수 있어야 한다. 본 연구에서 NDIS에 snort를 이용한 소프트웨어적인 패턴매칭을 FPGA를 이용하여 하드웨어적 패턴매칭으로 구현하였으며, 새로운 법칙에 따라서 유연하게 적응할 수 있도록 패턴매칭을 정규 표현식(Regular Expression)으로 나타내어 FPGA에 재구성할 수 있도록 하였다.

1. 서론

컴퓨터분야에 네트워크의 출현 이후 데이터 통신 기술이 발달하면 할 수록 네트워크의 보안의 중요성은 점점 높아져 왔다. 네트워크를 통해 시스템에 침입 하려는 시도는 시간이 흐를수록 다양해지며 복잡해지고 있다. 이에 대한 하나의 해결책인 Network Intrusion Detection System(NIDS)는 컴퓨터 시스템으로 들어오는 네트워크 패킷들을 분석하여 침입 여부를 판단하는 시스템이다. 이 시스템의 성능은 전적으로 패턴매칭 기술에 좌우되는데, 비약적인 소프트웨어적 패턴매칭 알고리즘의 발전에도 불구하고 수없이 많은 패킷들의 내용까지 매칭을 시도하기에는 불가능하다. 최근의 경향을 보더라도 이미 Gbps급 Network Interface Card(NIC)가 보급되고 있는 현실에서 NIDS가 네트워크 속도를 따라잡지 못하여 잠재적으로 공격들을 막을 수 있는 법칙들을 비활성화 시켜야 하는 경우가 발생하기도 한다.

본 연구가 제안하는 것은 NIDS중 부하가 가장 많이 걸리는 패턴매칭 역할을 Field Programmable Gate Arrays(FPGA)에게 전가시키도록 하는 것이다. FPGA는 고정된 하드웨어(ASIC등)를 사용하는 것에 비해 재구성할 수 있는 특성을 바탕으로 자주 변화되는 패턴매칭에 최적화된 하드웨어적 알고리즘을 적용할 수 있고 하드웨어적 특성을 바탕으로 소프트웨어 기반의 NIDS에 비해 월등한 성능을 발휘할 수 있다.

본 연구는 NIDS와 관련되어 선행된 연구들의 구현 사례들을 살펴본 후 실제로 FPGA에서 NIDS의 핵심인 패턴매칭을 위한 회로 생성(circuit generation)에 대하여 초점을 맞춘다.

본 논문은 2 장에서는 관련연구에 관하여 기술하고 3 장에서 FPGA를 이용한 NIDS의 문자열 패턴매칭 구현에 대하여 설명하였다. 그리고 4장에서 결론을 맺었다.

2. 관련 연구

2.1. 정규 표현식 매칭 방식

NIDS에 Snort rule 언어를 정규 표현식(Regular Expression: RE)으로 표현되어 있다[1]. 이러한 정규 표현방식을 Nondeterministic Finite Automata(NFA)를 이용하여 대응하는 구성요소 회로로 표현하는 방식이 제시되었다[2]. 그림 1은 정규 표현식을 대응하는 패턴매칭 회로를 구성하는 방법을 보여주고 있다. 그러나 패턴매칭을 위한 회로를 구성하는데 문자 단위 비교기가 각 문자마다 배치되므로 불필요한 자원의 낭비가 발생한다.

† 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음.

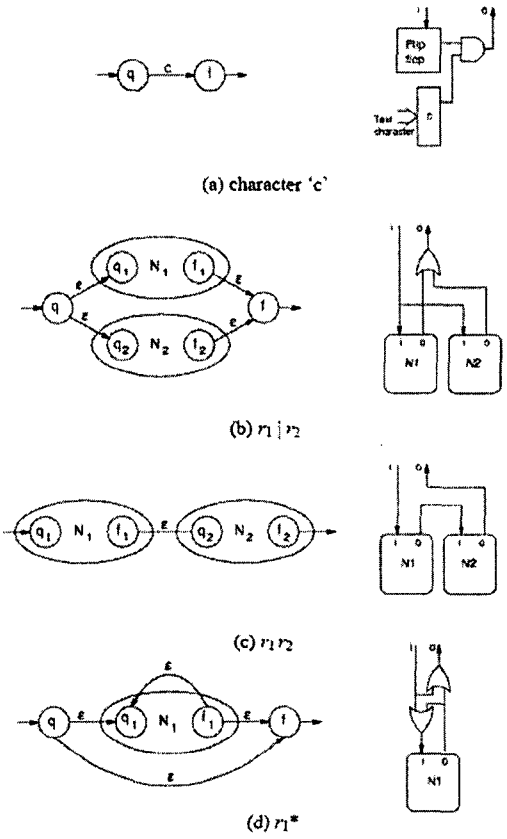


그림 1. 정규표현식에 대응하는 NFA와 회로[2]

2.2. 공유 디코더 방식

패턴들에 대한 회로들을 구성할 경우 비교가 시작되면 입력스트림 즉 NIDS에서의 패킷이 각 패턴을 구성한 회로들로 전달된다. 이때 비교는 최소 1바이트 단위밖에 할 수 없으므로 8Bit Bus를 통해 전달 될 것이다(분산비교 방식). 이러한 방식은 Routing Path의 낭비가 발생하게 되는데 한 바이트의 입력은 모든 패턴들의 회로들로 라우팅 될 필요가 없는 것이다. 이러한 불필요한 라우팅 경로의 낭비를 줄이기 위해 공유디코더(Shared Decoder) 방식이 제시되었다[3]. 그림 2는 공유디코더 방식이 어떻게 라우팅 경로를 줄이는지 보여준다. 디코더를 경유하면 암묵적으로 비교가 일어나며 해당되는 1bit line을 해당 Match Unit에 연결만 하면 된다. 최종 비교 결과는 분산비교방식과 같이 1Bit 의 M_{out} 으로 출력된다. 그렇지만 수시로 변화하는 다양한 침입 패턴을 빠르고 유연성 있게 회로로 재구성하는 방법을 제시하지는 않고 있다.

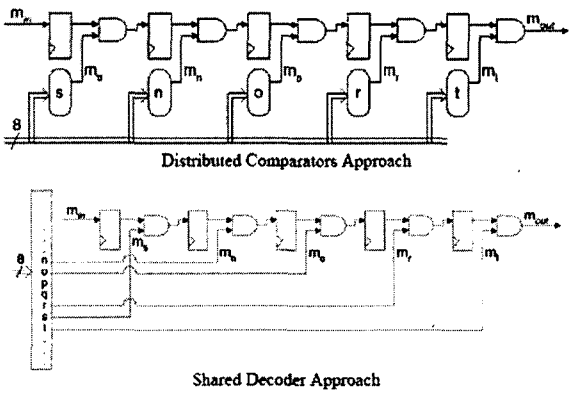


그림 2. 분산비교방식과 공유디코더방식[3]

3. FPGA 구현

3.1. 회로설계방법

매치 프로세서의 Datapath는 다음과 같다. 32bit 버스를 통해 입력스트림을 받고 1 clock cycle 당 1개의 문자비교를 위해 8bit로 직렬화 한다. 그 후 공유디코더를 사용하여 암묵적으로 비교를 수행하며 공유리코더로부터 출력되는 값은 각 패턴회로에 적절한 위치로 배정된다. 패턴 회로로부터 출력되는 비교결과를 레지스터에 저장해 두었다가 입력스트림의 종료와 함께 레지스터 값을 인코더를 통해 최종 결과로 출력하도록 한다.

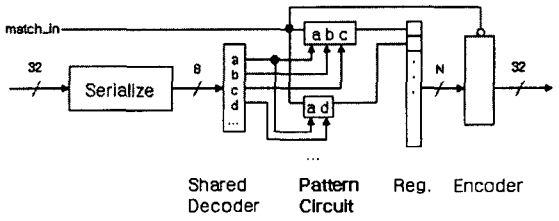


그림 3. datapath of matching unit

따라서 1 clock cycle 당 n개의 패턴에 대해 각자 1문자씩 비교를 수행하게 되며 모든 패턴의 검색은 단지 입력스트림의 바이트 수만큼의 clock cycle이면 완료된다.

3.2. 패턴회로 생성기

그림 3에서 나타내듯이 패턴회로 부분을 제외하고는 모두 고정적인 모듈이 된다. 패턴회로는 어떤 문자열 패턴들을 검색하느냐에 따라 디자인이 바뀌게 된다. 각각

의 패턴 안에 한 문자는 하나의 하드웨어 블록으로 할당하고 이들을 상호 연결하여 패턴회로를 구성한다. 이때 각각의 문자 블록은 VHSIC Hardware Description Language(VHDL)로 표현하여 하나의 모듈로 표현이 될 수 있다. 따라서 이 하드웨어 문자 블록을 컴포넌트화 하여 계층적인 하드웨어 디자인을 스크립트를 통해 생성(Generation)할 수 있도록 한다. 이러한 생성과정을 통하여 snort에 새로운 패턴이 추가되더라도 손쉽게 하드웨어를 재구성할 수 가 있다.

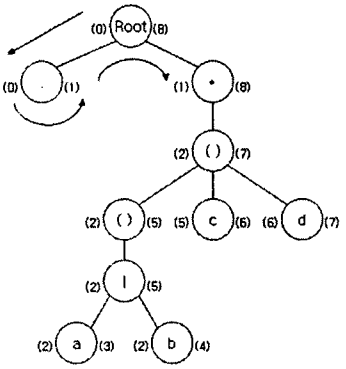
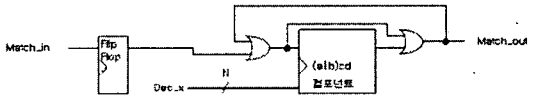
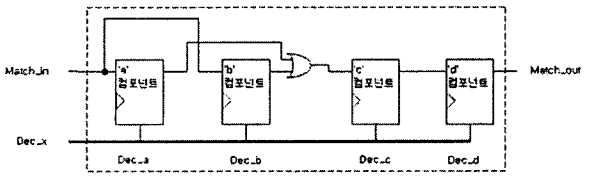


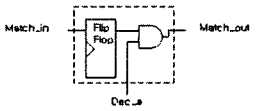
그림 4. 정규표현식 .((a|b)cd)* 에 대응하는 트리



(a) 정규표현식 .((a|b)cd)* 의 회로도



(b) 정규표현식 (a|b)cd 의 회로도



(c) 정규표현식 a 의 회로도

그림 5. 정규표현식 .((a|b)cd)* 의 회로도

정규표현식으로 표현되는 값에 대응하는 VHDL 생성을 위해 먼저 우선순위를 고려해야 한다. 정규표현식을 우선순위에 맞게 트리구조로 구성하고, 이때 각 노드의 왼쪽은 노드의 입력배선번호이며 오른쪽은 출력배선번호가 된다. 루트노드 왼쪽에서 왼쪽 아래방향으로 시작하여 리프노드에서 배선정보가 1 증가하여 오른쪽으로 올라간다. "*"과 "|"은 통과시에도 배선정보를 적절히 수정하며 배선할당이 완료되면 각 노드의 형식과 배선정보를 참고하여 VHDL 코드를 생성할 수 있다.

예제로서 그림 5는 정규표현식 .((a|b)cd)* 의 회로도를 나타내고 그림 6은 그림 5의 회로를 VHDL 의 Structural Modeling 기반으로 표현한다.

```
FF0 : flip_flop(CLK, match_in, wire(1));
wire(2) <= wire(1) or wire(7);
wire(8) <= wire(2) or match_out;
CH0 : char_match port map(CLK, wire(2), dec_a, wire(3));
CH1 : char_match port map(CLK, wire(2), dec_b, wire(4));
wire(5) <= wire(3) or wire(4);
CH2 : char_match port map(CLK, wire(5), dec_c, wire(6));
CH3 : char_match port map(CLK, wire(6), dec_c, wire(7));
```

그림 6. 생성된 .((a|b)cd)* 의 VHDL 구현 부분

4. 결론

현재 네트워크 침입을 위한 패턴들은 항상 새롭게 생겨나고 있으며 설정을 통해서도 패턴집합들은 수시로 바뀔 수 있다. 앞에서 구성한 내용에서 패턴들을 바꾸어 적용하려면 바뀐 하드웨어 디자인을 FPGA에 다시 적용시켜야 한다. 본 연구는 변경되는 침입 패턴에 따른 FPGA 디자인을 자동적으로 생성되는 기능을 구현하였다. 하지만 사용자 입장에서는 FPGA 디자인의 변경보다 응용프로그램에서의 패턴변경 적용이 더욱 유연하다. 하지만 시스템 메모리에 패턴들을 저장시키고 하는 매칭연산은 메모리와 하드웨어 사이의 제한된 버스 넓이로 인해 1 Clock Cycle에 모든 패턴들의 비교를 시도하기는 어렵다. 또한 FPGA 구성을 위해 자원 소모량을 최소화 시켜야 한다. 그러므로 패턴들이 하드웨어 디자인에 독립적으로 설정될 수 있으면서 높은 성능을 낼 수 있도록 하는 것이 앞으로의 과제이다.

5. 참고문헌

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

[2] R. Sidhu and V.K. Prasanna, "Fast Regular Expression Matching using FPGA," Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines, pp. Apr. 2001.

[3] Christopher R. Clark, "Design of Efficient FPGA Circuits for Matching Complex Patterns in Network Intrusion Detection Systems"