2014년 6월 전자공학회 논문지 제 51권 제 6 호
Journal of The Institute of Electronics and Information Engineers Vol. 51, NO. 6, June 2014 http://dx.doi.org/10.5573/ieie.2014.51.6.110

논문 2014-51-6-13

네트워크 보안을 위한 서픽스 트리 기반 고속 패턴 매칭 알고리즘

(High Performance Pattern Matching algorithm with Suffix Tree Structure for Network Security)

오 두 환*, 노 원 우**

(Doohwan Oh and Won Woo Ro©)

요 약

패턴 매칭 알고리즘은 컴퓨터 네트워크, 유비쿼터스 네트워크, 그리고 센서 네트워크 등을 위한 보안 프로그램에 주로 사용 된다. IT 기술의 발전과 함께 정보의 디지털화가 가속화되면서 네트워크를 통해 전달되는 데이터양이 급증하고 있다. 이에 따라 패턴 매칭 연산의 복잡도도 폭발적으로 증가하고 있다. 따라서 더 많은 패턴을 보다 빠르게 검색할 수 있는 고성능 알고리즘의 개발이 끊임없이 요구되고 있다. 본 논문은 서픽스 트리 기반 패턴 매칭 알고리즘을 새롭게 제안하여 대용량 패턴 매칭 연산의 성능을 높였다. 서픽스 트리는 사전에 정의된 복수 패턴들의 서픽스를 기반으로 생성된다. 이 트리에 쉬프트 노드 개념을 추가하여 기존 패턴 매칭 연산들 중 불필요한 연산의 수행 횟수를 줄였다. 결과적으로 제안하는 구조를 통해 기존 알고리즘 대비 24% 이상의 성능 향상을 이루었다.

Abstract

Pattern matching algorithms are widely used in computer security systems such as computer networks, ubiquitous networks, sensor networks, and so on. However, the advances in information technology causes grow on the amount of data and increase on the computation complexity of pattern matching processes. Therefore, there is a strong demand for a novel high performance pattern matching algorithms. In light of this fact, this paper newly proposes a suffix tree based pattern matching algorithm. The suffix tree is constructed based on the suffix values of all patterns. Then, the shift nodes which informs how many characters can be skipped without matching operations are added to the suffix tree in order to boost matching performance. The proposed algorithm reduces memory usage on the suffix tree and the amount of matching operations by the shift nodes. From the performance evaluation, our algorithm achieved 24% performance gain compared with the traditional algorithm named as Wu-Manber.

Keywords : Pattern Matching, Wu-Manber, Virus Scanning, Network Intrusion Detection System

Ⅰ. 서 론

다중 패턴 매칭 알고리즘은 복수의 패턴들이 입력 데이터 스트림에 포함되어 있는지 그 여부를 검사하는데 사용된다[1~3]. 컴퓨터 보안 분야에 주로 활용되며 스팸 메일, 바이러스, 네트워크 침입 등의 악성 패턴을 검출하는데 사용된다[4~6]. 최근 들어 IT기술의 발전과 함께 데이터양이 폭발적으로 증가하게 되었다. 게다가 네트

* 학생회원, ** 정회원, 연세대학교 전기전자공학과
(Department of Electrical and Electronic Engineering, Yonsei University)
© Corresponding Author(E-mail: wro@yonsei.ac.kr)
※ 본 논문은 산업통상자원부 산업융합원천기술개발사업으로 지원된 연구결과입니다[10041971, 상황대응형 분산트랜스코딩 기술을 이용한 저전력 고성능 멀티미디어 콘텐츠관리기술 개발].
접수일자: 2014년03월20일, 수정일자: 2014년05월01일
수정완료: 2014년05월29일

워크 속도가 기가바이트 이상으로 빨라지면서 네트워크 보안 시스템의 처리 속도역시 그 이상으로 높아져 가고 있다[7]. 검사해야할 데이터양의 증가와 네트워크 속도의 향상으로 인해 고성능의 패턴 매칭 알고리즘의 필요성이 높아져 가고 있다.

패턴 매칭에 사용되는 패턴의 수 역시 급속도로 증가하고 있다[8~10]. 악의를 품은 크래커들은 지속적으로 새로운 악성 코드들을 생성하여 임의의 컴퓨팅 시스템에 침입을 시도하고 있다. McAfee Inc.가 발표한 보고서[11]에 따르면 2013년 3분기에 2천만 개의 새로운 악성 패턴들이 만들어졌으며 이는 2분기 대비 14% 증가한 수치임을 보고하였다. ClamAV[12]는 대표적인 안티바이러스 오픈소스 프로젝트이다. 다중 패턴 매칭 알고리즘인 Aho-Corasick 알고리즘[13]과 Wu-Manber 알고리즘[14]을 혼용하여 악성 패턴이 입력 데이터 스트림에 포함되어 있는지 검사한다. 불행하게도 이 악성 패턴의 수 역시 급속도로 증가하고 있다. 2008년 발표된 ClamAV 0.91.2버전에 정의된 바이러스 패턴의 수는 230,000개 이었으나, 2014년 발표된 0.98.1 버전에서는 4백만 개 이상으로 늘어났다. 따라서 대규모의 패턴들을 동시에 검출할 수 있는 고성능 패턴 매칭 알고리즘 개발이 필요하다.

대표적인 다중 패턴 매칭 알고리즘으로는 오토매톤 기반의 Aho-Corasick (AC) 알고리즘이 있다. 패턴 매칭 연산이 시작되기 전에 주어진 패턴으로 유한 오토마톤을 구성한다. 매칭이 시작되면 타깃 텍스트의 글자를 하나씩 오토마톤으로 입력하여 트랙킹하는 방식으로 패턴을 검출한다. 오토마톤은 두 종류가 존재한다. 하나는 Deterministic Finite Automaton (DFA)이고 다른 하나는 Non-deterministic Finite Automaton (NFA) 이다. 이 두 방식 모두 오토매톤 저장을 위해 큰 메모리 공간을 요구한다. 오토마톤의 states 수는 패턴의 개수와 길이에 비례해 증가하며 또한 각 state는 전이(transition) 가능한 모든 states에 대한 정보를 포함해야하기 때문에 많은 수의 패턴을 저장하는데 어려움이 있다.

다른 패턴 매칭 방식으로는 발견적 해결 방법을 사용하는 Wu-Manber (WM) 알고리즘이 있다. WM은 많은 수의 패턴을 동시에 검색하기 위해 패턴들을 테이블로 간략화 한다. 이를 통해 큰 사이즈의 데이터를 로딩할 때 발생하는 메모리 액세스 지연 현상을 최소화하였다. WM이 사용하는 데이터 구조로는 쉬프트 테이블, 해시 테이블, 프리픽스 테이블이 있다. 쉬프트 테이블은 불필요한 연산을 제거하는 기능을 수행 한다. 해시 테이블은 서픽스가 같은 패턴들을 그룹 하여 관리한다. 프리픽스 테이블은 프리픽스 기준으로 패턴을 분류하여 관리한다. 이 중 WM의 가장 큰 강점은 쉬프트 테이블이다. 불필요한 연산을 수행하지 않음으로써 전체 처리 시간을 단축할 수 있기 때문이다. 하지만 쉬프트 테이블은 패턴의 개수가 일정 수 이상으로 많아지게 되면 메모리사용량이 급격하게 증가하는 문제가 발생한다. 그 이유는 블록이라는 변수에 있다. 블록의 크기는 쉬프트 테이블의 익덱스 개수와 연관이 있다. 인덱스 개수는 블록 내부 글자 수에 지수 승으로 증가하기 때문이다. 블록 내부 글자 수 B는 다음수식에 따라 정해진다.

$$B = \log_{|\Sigma|} 2M, \quad (1)$$

Σ는 패턴에 사용된 글자의 종류 수, M은 가장 짧은 패턴의 길이 곱하기 패턴 수이다. 즉, 블록의 크기는 패턴의 길이와 개수에 비례하여 증가하기 때문에 대규모의 패턴을 매칭하기 위해서는 거대한 테이블을 구성해야 한다.

본 논문에서는 오토마톤을 기반으로 하는 새로운 패턴 매칭 알고리즘인 SSST (Shift State on Suffix Tree) 알고리즘을 제안한다. 메모리 문제를 해결하기 위해 패턴의 서픽스만으로 오토매톤 구성하여 크기의 문제를 해결하였다. 또한 WM의 장점인 쉬프트 테이블의 원리를 적용하기 위해 shift state를 추가하였다. 이를 위해 AC가 사용하는 포워드 방식 대신 백워드 방식으로 오토마톤을 구성하였다. 즉, 패턴의 서픽스를 역순으로 나열하여 오토마톤을 구성하였다.

본 논문의 구성은 다음과 같다. 먼저, Ⅱ장에서 제안하는 알고리즘이 패턴 매칭 수행 이전에 오토마톤을 구성하는 방법에 대하여 서술한다. Ⅲ장에서는 오토마톤을 트랙킹하면서 패턴을 찾는 방법에 대하여 설명한다. Ⅳ장에서는 성능 평가 결과에 대하여 논한다. 마지막으로 Ⅳ장에서는 결론에 대해 기술한다.

Ⅱ. 본 론

1. 서픽스 트리 구성

SSST는 입력 텍스트 D = $\{d_0 d_1 ... d_{r-1}\}$가 패턴 세트 P

= {p_1,p_2, ...,p_n}중 하나 또는 그 이상의 패턴을 포함하는지 검사하기 위해 서픽스 트리를 사용한다. (d_0는 첫번째 글자를 의미함, p_1는 첫 번째 패턴을 의미함.) 서픽스 트리는 검사가 시작되기 전에 패턴 세트 P를 기반으로 구성된다. P에 포함된 패턴의 수가 많거나 각 패턴의 길이가 길어지면 트리의 사이즈가 기하급수적으로 커지게 되므로 이를 방지하기 위해 SSST는 P중 가장 짧은 패턴의 길이를 찾고 모든 패턴을 이 길이로 맞춘다. SSST는 길이가 동일해진 패턴을 서픽스 트리로 구성한다. 즉, 가장 짧은 패턴의 길이를 q라 정의하면 서픽스 트리 구성에 고려되는 패턴들은 pi′ = {$c_0c_1...c_{q-1}$} 가 된다 (1 <= i <= n, i는 정수, c^i_0은 i번째 패턴의 첫 번째 글자를 의미). 우리는 $p^{i'}$를 각 패턴의 서픽스라 정의

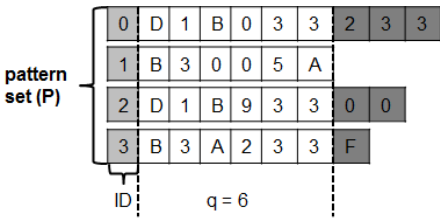


그림 1. 샘플 패턴 세트 (P)
Fig. 1. Sample Pattern Set (P).

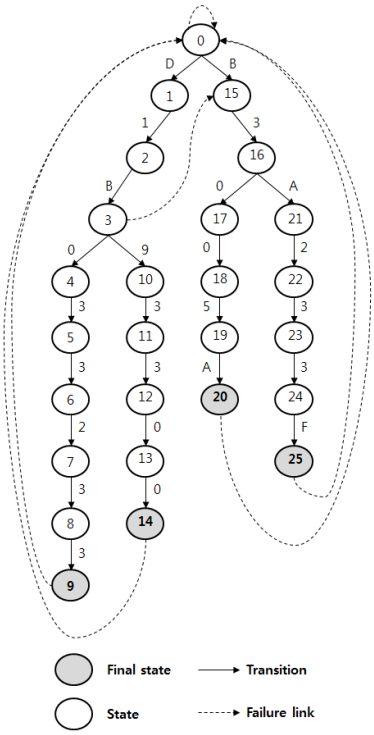


그림 2. 패턴 세트 P의 NFA구조
Fig. 2. NFA structure for the pattern set P.

하고 이 집합을 P′라 한다.

그림 1에 보인 패턴 세트 P에 대한 NFA의 포워딩 방식을 그림 2에 보였다. 포워딩 방식은 각 패턴의 첫 번째 글자부터 순서대로 트리 노드를 구성한다. 글자와 글자가 이어지는 부분은 transition으로 연결한다. 패턴의 마지막 글자에 대한 transition은 final state로 연결된다. 모든 states는 transition의 없는 입력에 대해 failure link를 가지며, state 0로 이동 한다. (그림에는 final state만 failure link가 연결되어 있지만 모든 state가 전부 failure link로 state 0에 연결된다.) 단, 3번 state의 failure link는 15번 state에 연결된다.

서픽스 트리는 패턴 세트 P의 변형인 P′를 기반으로 그림 3과 같이 구성된다. (ID 3의 패턴 “B3A233F”중 앞 q개 글자인 “B3A233”이 $P^{3'}$가 된다.) 이때 AC가 사용하는 포워딩 방식 대신 백워드 방식으로 트리를 구성한다. 이를 통해 WM의 쉬프트 테이블 개념을 서픽스 트리에 도입할 수 있게 된다. 서픽스 트리는 패턴의 역순으로 구성되기 때문에 q번째 글자부터 state를 구성한다. 포워딩 방식과 마찬가지로 q개의 글자 입력 마지막에는 final state가 존재하며 모든 state는 failure link로 0 state로 연결된다. 단, 2. 13, 그리고 19번 state는 failure link로 1 state로 연결되고, 12번 state는 15번 state로 연결된다.

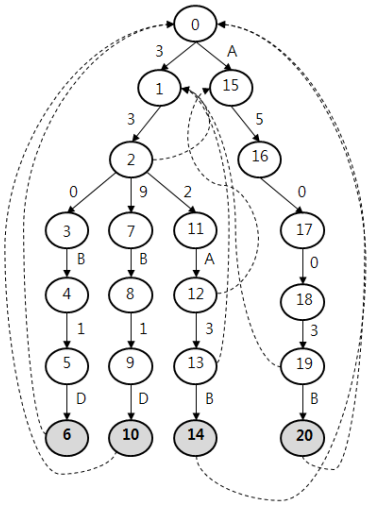


그림 3. 패턴 세트 P′ 의 서픽스 트리 구조
Fig. 3. Suffix tree for the pattern set P′.

2. Shift state 추가

Boyer-Moore 알고리즘[15]은 단일 패턴 매칭 알고리

즘으로서 bad character shift table을 사용해 성능 향상을 꾀하였다. 이 table은 현재 검사하고 있는 텍스트 위치에서 얼마나 많은 수의 글자를 검사하지 않고 스킵할 수 있는지 알려준다. WM은 이 table을 다중 패턴 매칭에 사용하기 위해 쉬트프 테이블로 확장하였다.

쉬프트 테이블의 인덱스는 B개의 글자로 나올 수 있는 모든 조합으로 구성된다. (WM은 B의 크기를 수식 (1)에 따라 보통 2 또는 3의 수치로 정한다.) 각 인덱스는 쉬프트 값을 하나씩 갖는다. 쉬프트 값은 해당 인덱스의 B개 글자 조합이 P′의 패턴에 포함되어 있을 경우 q-l의 값을 갖는다. l은 B개 글자 조합이 패턴 내 l번째 글자에 위치함을 의미한다. 만약 B개 글자 조합이 어떤 패턴에도 포함되어 있지 않다면 그 인덱스의 쉬프트 값은 q-B+1이 된다. 그림 4에 패턴 세트 P′에 대한 쉬프트 테이블의 일부를 보였다.

본 논문에서 제안하는 SSST알고리즘은 쉬프트 테이블을 shift states로 확장하여 서픽스 트리에 추가하였다. Shift state 방식은 WM의 쉬프트 테이블과는 달리 복수의 인덱스 크기 (블록의 크기, B), 즉, 블록의 크기를 1부터 B범위의 모두에 대하여 고려한다. 쉬프트 값을 결정하는 핵심 요소는 가장 짧은 패턴의 길이 q와 블록 크기 B 이다. q 수치가 클수록 평균 쉬프트 값은 높아진다. 반면에 B 수치는 작을수록 평균 쉬프트 값이 커진다. 그렇다고 너무 작은 값을 가지면 B개의 글자 조합이 패턴에 존재할 확률이 증가하여 0 쉬프트 값이 많아지게 되는 문제가 있다. 따라서 B의 크기는 최소 수식 (1)의 조건을 만족해야 한다.

서픽트 트리는 쉬프트 테이블과는 달리 구조적 특성상 수식 (1)로 유도된 B보다 작은 수치에 대한 쉬프트 값들도 고려할 수 있다. 그 이유는 첫 번째 글자 입력이 패턴에 존재한다고 해도 그 다음 글자 입력이 존재하지 않으면 현재까지 입력된 글자 조합을 통해 쉬프트 값을 계산하여 불필요한 연산을 스킵할 수 있기 때문이다.

서픽스 트리에 shift states를 추가하는 방법은 다음과 같다. Shift state는 하나의 쉬프트 값을 갖는다. 총 shift states의 개수는 나올 수 있는 모든 쉬프트 값들의

SHIFT table

Index (*s*)	D1	1B	B0	03	33	B3	30	00	05	5A	...	*
shift value	4	3	2	1	0	4	3	2	1	0	...	5

그림 4. 패턴 세트 P′의 쉬프트 테이블
Fig. 4. Shift table for the pattern set P′.

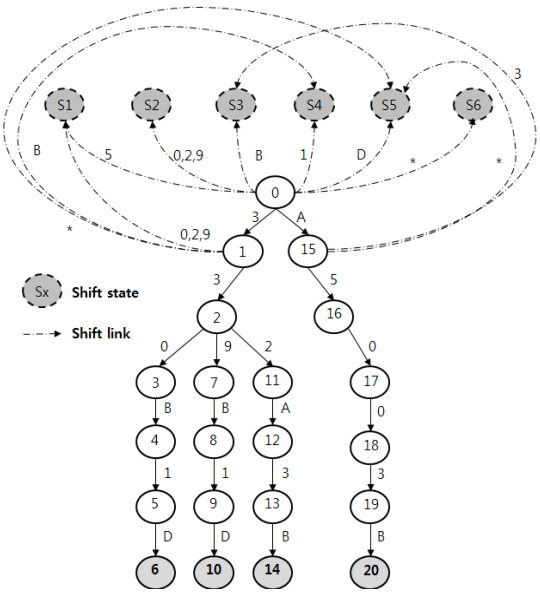


그림 5. Shift states를 포함하는 서픽스 트리
Fig. 5. Suffix tree with shift states.

수와 동일하다. 즉, 1부터 q-B+1까지의 shift states가 존재한다. 단, 블록 크기는 하나로 고정되지 않고 1에서 B까지 가변적이므로 총 q개의 shift states가 트리에 존재한다. 일반 states에서 shift states로의 transition을 shift link라 정의하며, 이 link를 생성하기 위해 다음의 과정을 거친다. 우선, 블록 크기가 1 일 경우에 대한 쉬프트 테이블을 생성하고 테이블의 인덱스를 서픽스 트리에 역순으로 입력한다. 이 입력이 failure link에 해당하면 failure link를 shift link로 대체하고 해당 인덱스의 쉬프트 값과 일치하는 shift state로 연결한다. 모든 인덱스에 대한 shift link연결이 완료되면, 블록 크기를 1 증가시키고 위 과정을 반복한다. 블록 크기 B까지 shift link연결이 완료되면 서픽스 트리에 shift states 추가가 완료된다.

그림 5는 서픽스 트리 구조(그림 3)에 shift states를 추가한 것이다. 0번 state에 연결된 shift links는 B가 1일 때 구성된 쉬프트 테이블을 기반으로 그려진다. State 1과 15는 B가 2일 경우 쉬프트 테이블(그림 4)을 기반으로 그려진다. 이 쉬프트 테이블에서 인덱스 “00,” “20,” 그리고 “90”의 쉬프트 값은 ‘1’이다. 따라서 State 1에서 입력이 ‘0,’ ‘2,’ 또는 ‘9’일 경우 shift state ‘S1’로 연결된다. (‘S1’은 쉬프트값 ‘1’을 의미함.) 나머지 shift links도 이와 같은 방법으로 연결된다.

3. 서픽스 트리 기반 패턴 매칭

Shift states를 포함하는 서픽스 트리가 구성되면 SSST는 입력받는 데이터 스트림(텍스트)에서 패턴 매칭을 시작한다. 패턴 매칭은 검색 윈도우(search window)가 텍스트를 따라 오른쪽으로 이동하면서 진행된다. 검색 윈도우의 크기는 q이며, q번째 글자부터 서픽스 트리에 입력되어 q-1, q-2번째를 거쳐 1번째 글자까지 역순으로 트랙킹한다. 트랙킹 중 shift state로 이동하면 검색 윈도우는 해당 쉬프트 값만큼 오른쪽으로 이동한 후 다시 q번째 글자부터 서픽스 트리에 입력하여 트랙킹하는 동작을 반복한다. 트랙킹 중간에 final state를 만나면 검색 윈도우의 글자조합을 가지는 패턴이 존재함을 의미한다. 따라서 현재 검색 윈도우의 q번째 다음 글자인 q+1번째 글자부터 순차적으로 해당 패턴의 q+1번째 글자와 한 글자씩 매칭을 진행한다. 패턴의 마지막 글자까지 매칭이 이루어지며 해당 패턴이 입력 텍스트에 존재함을 의미한다. 예로, P가 바이러스 패턴의 집합이라면 현재 텍스트에 바이러스가 존재함을 의미한다. 따라서 시스템은 즉각적으로 이를 관리자에게 보고한다. 해당 위치의 검색 윈도우에 대한 조치가 마무리되며 검색 윈도우는 한 글자 오른쪽으로 이동되고 다시 트랙킹 작업을 반복한다. 검색 윈도우가 입력 텍스트의 끝 글자에 도달하면 전체 패턴 매칭 작업은 완료된다.

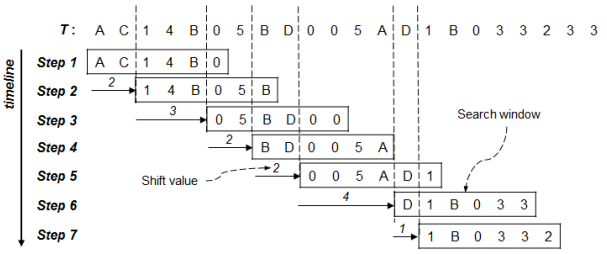


그림 6. SSST 패턴 매칭 프로세스
Fig. 6. Pattern matching process of SSST.

Ⅲ. 실 험

본 장에서는 SSST의 성능을 이전의 알고리즘들과 비교 평가한다. 평가는 실제 컴퓨팅 시스템에서 ClamAV 바이러스 패턴이 리눅스 바이너리 파일에 존재하는지 검색하는데 걸린 시간을 측정하여 진행하였다. 평가 시스템의 CPU는 1.4 MHz로 동작하는 AMD 사의 A10-5800K 프로세스이고, RAM은 32 GB 용량을 가진다. 모든 알고리즘은 C 언어로 구현되었으며, 최적화 옵션 03와 함께 gcc 4.7 버전으로 컴파일 되어 Ubuntu 12.04 운영체제에서 실행되었다.

평가에 사용한 샘플 패턴은 ClamAV 0.97.6 버전의 패턴 세트 중 15,000개를 추출하여 구성하였다. 샘플 패턴의 가장 짧은 길이는 16 글자이고, 가장 긴 패턴은 1022 로 평균 206의 길이를 가진다. 입력 파일로는 Ubuntu OS의 /usr/bin 폴더의 바이너리들을 사용하였다. 용량은 24.7 MB에 달한다.

1. 서픽스 트리의 크기 비교

본 절에서는 동일한 패턴 세트에 대한 SSST의 서픽스 트리와 AC가 사용하는 NFA 크기를 비교하였다. 결과는 그림 7에 보였다. NFA의 경우 각 패턴의 전체 길이에 대한 오토마톤을 구성하기 때문에 SSST에 비해 훨씬 많은 states로 구성된 것을 확인할 수 있다. 패턴 개수가 10,000개 일 경우 NFA는 804,333개의 states로 구성된다. 반면에 SSST는 shift states를 포함해 160,017개로 구성된다. 이 차이는 패턴의 개수가 늘어날수록 커진다. 15,000개 패턴에 대해 NFA는 1,397,297 states로 구성된 반면 SSST는 240,017개로 구성된다. NFA의 크기는 SSST대비 약 5.8배 크다. States의 개수가 늘어나는 것은 각 state의 transition 정보의 증가를 의미함으로 실제 메모리 사용량의 차이는 더 크게 벌어지게 된다.

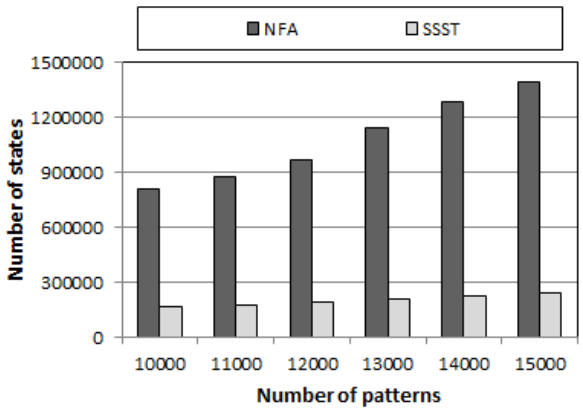


그림 7. NFA와 서픽스 트리의 states개수 비교
Fig. 7. Comparison of number of states on NFA and Suffix tree.

2. 패턴 매칭 성능 비교

SSST의 패턴 매칭 성능을 평가하기 위해 WM, AC, 그리고 BH 알고리즘[16]과의 패턴 매칭 성능을 비교하였다. BH 알고리즘은 Lin 연구 그룹에서 제안하였으며, 평균 쉬프트 수치를 높이기 위해 백워드 해싱 알고리즘을 적용하였다. 이 알고리즘은 검색 윈도우 내에 블록을 두 개를 위치하여 쉬프트 값을 계산한다. 두 블록의 사이즈는 WM과 마찬가지로 B로 고정된다.

결과는 그림 8에 보였다. 패턴의 개수가 상대적으로 적은 10,000개 패턴에 대한 매칭 성능은 AC가 1.173초, BH가 1.285초, WM이 1.335초, 그리고 SSST가 1.603초를 나타냈다. SSST가 다른 알고리즘들에 비해 가장 느린 성능을 보였다. 하지만 패턴의 개수가 증가할수록 SSST외의 나머지 3개 알고리즘들의 성능 저하가 뚜렷하게 나타났다. SSST는 13,000개 패턴을 매칭할 때 WM과 BH알고리즘의 성능을 추월하였으며, 14,000개 패턴을 매칭할 때는 AC의 성능도 추월해 가장 빠른 매칭 속도를 보였다. 결국 15,000개 패턴을 매칭할 때 AC는 2.012초, BH는 2.079초, WM은 2.223초, 그리고 SSST는 1.786초를 보였다. 이 수치는 AC대비 12%, WM대비 24% 높은 성능을 의미한다. SSST가 다른 알고리즘보다 높은 성능을 보이는 것은 패턴 개수에 비례해 증가하는 서픽스 트리의 크기보다 AC의 NFA의 크기 증가가 더 컸지 때문이다. 또한, shift states는 WM과 BH의 고정된 B 수치 대신 모든 1부터 B까지 가변적 수치에 대하여 쉬프트 값을 계산하기 때문에 더 많은 글자를 매칭 연산 없이 스킵할 수 있기 때문이다.

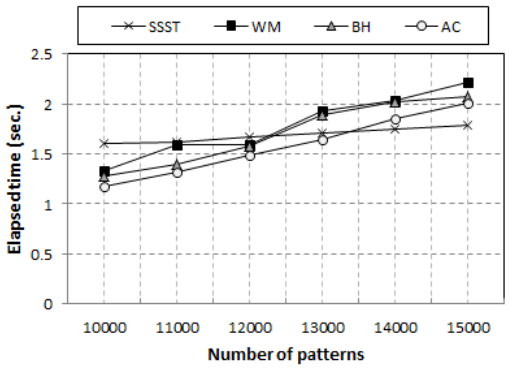


그림 8. 패턴 매칭 성능 비교
Fig. 8. Comparison of processing time on pattern matching operations.

Ⅳ. 결 론

본 논문에서는 서픽스 트리 구조를 기반으로 하는 고성능 패턴 매칭 알고리즘인 SSST 알고리즘을 제안하고 있다. 이 알고리즘은 패턴 매칭 프로세스의 메모리 사용량을 줄이기 위해 패턴들을 동일한 크기로 재구성한 후 이를 역순으로 구성한 서픽스 트리를 사용하였다. 또한 이 서픽스 트리에 shift states를 추가하여 불필요한 연산이 실행되는 것을 방지함으로써 패턴 매칭 프로세스의 실행 시간을 단축하였다.

실제 네트워크 보안에 널리 사용되고 있는 ClamAV 모델에 SSST 알고리즘을 적용하였을 경우 기존 시스템 대비 최대 24%의 성능 향상을 보이는 것을 확인하였다. 이를 통해 보안 시스템의 단위시간당 데이터 처리량을 높일 수 있을 것으로 기대한다.

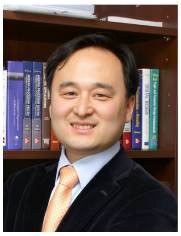
REFERENCES

[1] G. Navarro and M. Raffinot, “Flexible pattern matching in strings: practical on-line search algorithms for texts and biological sequences,” *Cambridge University Press*, 2002.
[2] C. Charras and T. Lecroq, “Handbook of Exact String Matching Algorithms.” *King’s College Publications*, 2004.
[3] B. Phoophakdee and M. J. Zaki, “Genome-scale disk-based suffix tree indexing,” in Proc. of ACM SIGMOD International Conference on Management of data (SIGMOD’07). New York, NY, USA: ACM, June 2007, pp. 833.844.
[4] Z. K. Baker and V. K. Prasanna, “A methodology for synthesis of efficient intrusion detection systems on FPGAs,” in Proc. of IEEE Symposium on Field-Programmable Custom Computing Machines. Washington, DC, USA: IEEE Computer Society, April 2004, pp. 135.144.
[5] P.-C. Lin, Y.-D. Lin, T.-H. Lee, and Y.-C. Lai, “Using string matching for deep packet inspection,” *Computer*, vol. 41, no. 4, pp. 23.28, 2008.
[6] N. B. Guinde and S. G. Ziavras, “Efficient hardware support for pattern matching in network intrusion detection,” *Computers & Security*, vol. 29, no. 7, pp. 756.769, 2010.
[7] Young Choi, Eun-Kyung Hong, Tae-Wan Kim, Seung-Tae Paek, Il-Hoon Choi, and

Hyeong-Cheol Oh, “A Traffic Pattern Matching Hardware for a Contents Security System,” *The Institute of Electronics Engineers of Korea - Computer and Information*, vol. 46, no. 1, pp.88.95, 2009.
[8] Z. Zhou, Y. Xue, J. Liu, W. Zhang, and J. Li, “MDH: a high speed multi-phase dynamic hash string matching algorithm for large-scale pattern set,” in Proc. of the 9th international conference on Information and communications security, ser. ICICS’07. Berlin, Heidelberg: Springer-Verlag, 2007, pp.201.215.
[9] J. Ostell, “Databases of discovery,” *Queue*, vol. 3, no. 3, pp. 40.48, 2005.
[10] Dae-Sung Kim and HyunJin Kim, “A heuristic for Mapping Patterns on Parallel String Matching Hardware in Deep Packet Inspection,” *IEEK Summer Conference*, vol. 35, no. 1, pp.522.523, 2012.
[11] McAfee, “McAfee Labs Threats Report: Third Quarter 2013,” Tech. Rep., November 2013.
[12] T. Kojm. (2012) Clam AntiVirus User Manual. [Online]. Available: http://www.clamav.net/doc/latest/clamdoc.pdf
[13] A. V. Aho and M. J. Corasick, “Efficient string matching: an aid to bibliographic search,” *Commun. ACM*, vol. 18, pp.333.340, June 1975.
[14] S. Wu and U. Manber, “A fast algorithm for multi-pattern searching,” Department of Computer Science, University of Arizona, Tech. Rep. TR-94-17, 1994.
[15] R. S. Boyer and J. S. Moore, “A fast string searching algorithm,” *Commun. ACM*, vol. 20, pp. 762.772, October 1977. [Online]. Available: http://doi.acm.org/10.1145/359842.359859
[16] P.-C. Lin, Y.-D. Lin, and Y.-C. Lai, “A hybrid algorithm of backward hashing and automaton tracking for virus scanning,” *Computers, IEEE Transactions on*, vol. 60, no. 4, pp. 594.601, 2011.

저 자 소 개

오 두 환(학생회원)
2007년 경희대학교 전자공학과 학사 졸업.
2010년 연세대학교 전기전자 공학과 석사 졸업.
2010년～현재 연세대학교 전기 전자공학과 박사과정
<주관심분야 : 컴퓨터 시스템, 네트워크 보안, 병렬 프로세싱>



노 원 우(정회원)
1996년 연세대학교 전기공학과 학사 졸업.
1999년 University of Southern California 석사 졸업.
2004년 University of Southern California 박사 졸업.
2003년～2004년 Apple Computer Inc. 인턴 연구원.
2004년～2007년 California State University 전기 및 컴퓨터공학과 조교수.
2006년～2007년 ARM Inc. 소프트웨어 엔지니어.
2007년～현재 연세대학교 전기전자공학부 부교수.
<주관심분야 : 고성능 마이크로프로세서 디자인, 컴파일러 최적화, 임베디드 시스템 디자인 등>