情報保護學會論文誌
第18卷 第3號, 2008. 6

규칙 적용 성능을 개선하기 위한 다중 패턴매칭 기법*

이 재 국†, 김 형 식‡
충남대학교

A Multiple Pattern Matching Scheme to Improve Rule Application Performance*

Jae-Kook Lee†, Hyong-Shik Kim‡
Chungnam National University

요 약

인터넷 환경에서 내부 네트워크를 보호하기 위하여 침입탐지시스템이 광범위하게 사용되고 있다. 침입탐지시스템은 비정상 패킷의 특성을 분석하여 규칙을 생성하고 이 규칙들을 이용하여 패킷들을 필터링함으로써 내부 시스템들을 보호한다. 최근 공격 사례가 많아지고, 공격 형태가 구조화되면서 이를 탐지하는 규칙의 수도 지속적으로 증가하고 있다. 이에 따라 침입탐지시스템이 규칙을 적용하는 과정에서의 성능 하락 정도도 커지고 있다.

본 논문은 규칙을 적용하는 과정에서 상대적으로 오버헤드가 큰 문자열 검색 성능을 개선하고자 복수개의 부분패턴을 이용한 다중 패턴매칭 기법을 제안한다. 그리고 대표적인 고성능의 다중 패턴매칭 알고리즘인 Wu-Manber 알고리즘과 성능을 비교하고 그 결과를 보인다.

ABSTRACT

On the internet, the NIDS(Network Intrusion Detection System) has been widely deployed to protect the internal network. The NIDS builds a set of rules with analysis results on illegal packets and filters them using the rules, thus protecting the internal system. The number of rules is ever increasing as the attacks are becoming more widespread and well organized these days. As a result, the performance degradation has been found severe in the rule application for the NIDS.

In this paper, we propose a multiple pattern matching scheme to improve rule application performance. Then we compare our algorithm with Wu-Manber algorithm which is known to do high performance multi-pattern matching.

Keywords : multiple pattern matching, rule application, intrusion detection

접수일 : 2008년 1월 3일; 수정일 : :2008년 3월 23일;
채택일 : 2008년 4월 10일
* 본 연구는 지식경제부 및 정보통신연구진흥원의 대학IT연구센터 지원사업의 연구결과로 수행되었음 (IITA-2008-C1090-0801-0016)
† 주저자, empire@csal.cnu.kr
‡ 교신저자, hkim@cnu.kr

Ⅰ. 서 론

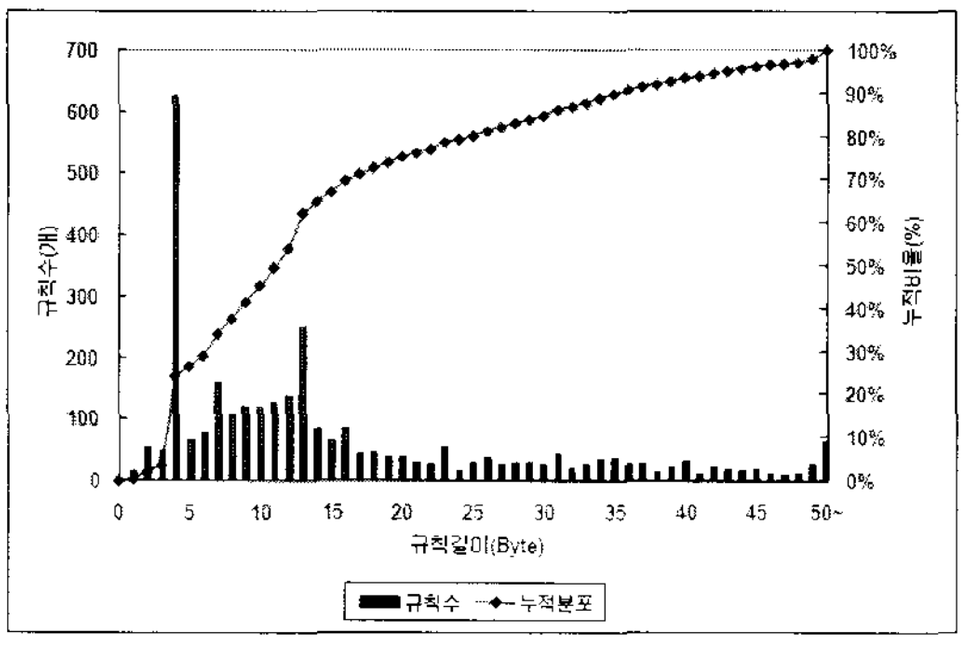
인터넷이 광범위하게 사용되면서 웜이나 트로이목마, 바이러스, DoS 등과 같은 다양한 형태의 침해가 나타나고 그 파급효과도 커지고 있다. 관리자들은 악의적인 공격으로부터 내부 네트워크를 보호하기 위하여 침입탐지

시스템(NIDS:Network Intrusion Detection Systems)을 사용한다. Snort[1]는 가장 대표적인 공개 소스 소프트웨어로 규칙을 활용하여 침입을 탐지하고 방지한다.

Snort와 같은 NIDS는 침입을 탐지하기 위하여 사용하는 알고리즘에 따라 차이가 있지만 패킷의 페이로드 부분에서 침입탐지 목적으로 컨텐트를 검색하기 위하여 전체 수행시간의 40~70%을 소비한다. 컨텐트 매칭은 주어진 패턴들과의 일치 여부를 검사하는 것으로 전체 수행 명령어의 60~85%가 패턴매칭에 이용된다[2].

악의적인 공격의 수가 증가하고 공격의 패턴이 다양화 되면서 Snort에서 사용되는 규칙들의 수가 계속적으로 증가했다. Snort 규칙집합은 패킷의 페이로드 부분의 컨텐트를 검사하기 위한 부분 때문에 2000년에서 2003년 전반기까지 2.5배 이상 증가하여 1533개에 이르고, 2005년 4월에 발표된 버전 2.3.3에는 컨텐트를 포함하는 규칙이 3033개로 2003년 6월과 비교해 2배 정도 증가하였다.

[그림 1]은 Snort 2.3.3 버전에 포함된 컨텐트의 길이별 분포를 나타낸 것이다. 컨텐트의 길이가 가장 짧은 것은 1바이트이고 가장 긴 컨텐트의 길이는 364바이트이다. 컨텐트를 갖고 있는 규칙 중에서 길이가 4바이트 이상인 규칙은 전체의 96.5% 이상을 차지한다. 이러한 특징은 다른 NIDS의 규칙집합에서도 나타날 것으로 예상된다. 따라서 침입탐지에 따르는 시간 비용을 줄이기 위해서는 컨텐트를 검색하는 패턴매칭 오버헤드를 개선해야 한다. Snort의 경우 패턴매칭을 위해서 초기에는 가장 보편화된 단일 패턴매칭 알고리즘인 Boyer-Moore 알고리즘[3]을 사용했지만 패턴의 증가에 따라 Aho-Corasick 알고리즘[4]이나 Wu-Manber 알고리즘[5]과 같은 다중 패턴매칭 알고리즘을 사용하여 속도를 향상시키고 있다.



〔그림 1〕 컨텐트의 길이별 분포

본 논문은 NIDS의 규칙수가 계속적으로 증가면서 상대적으로 많은 오버헤드를 발생시키는 패턴매칭 성능을 개선하기 위하여 복수개의 부분패턴을 이용하여 고정된 크기의 시프트를 보장하는 다중 패턴매칭 기법을 제안한다. 제안한 기법은 전처리 단계와 패턴매칭 단계로 구분된다. 전처리 단계에서 부분패턴을 이용하여 해싱테이블을 구성하고, 패턴매칭 단계에서 패킷의 페이로드 즉 문자열 부분에서 패턴이 존재하는지 전처리 단계에서 생성된 부분패턴을 이용하여 비교하고, 부분패턴이 존재하는 경우, 선택적으로 패턴 전체를 비교한다.

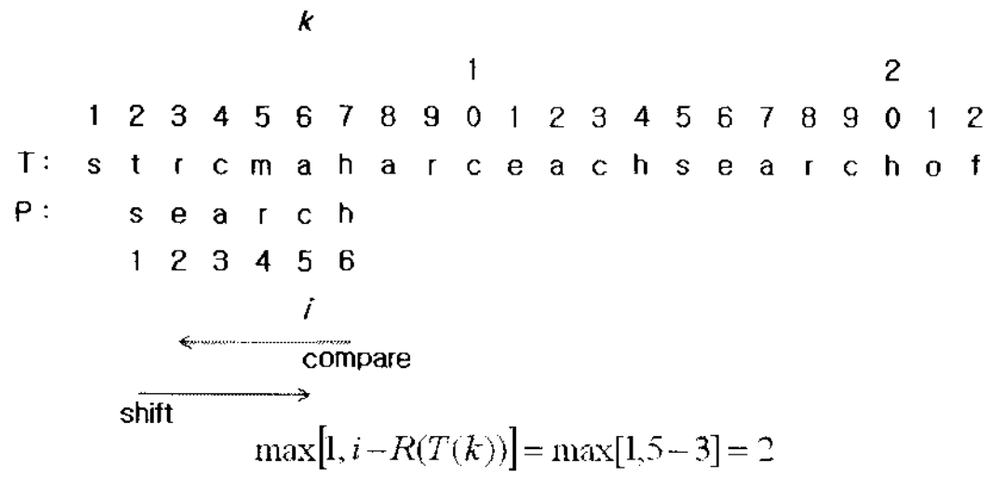
논문의 구성은 다음과 같다. 2절에서는 대표적인 패턴 매칭 알고리즘인 Boyer-Moor 알고리즘과 본 논문에서 비교의 대상이 될 Wu-Manber 알고리즘 및 이를 개선한 알고리즘에 대해서 설명한다. 3절에서는 제안하는 해싱 함수를 이용한 패턴 매칭의 전처리 알고리즘과 검색 알고리즘을 설계하고, C 프로그램을 이용하여 구현한다. 4절에서는 실제 Snort 패턴의 일부를 이용하여 패턴의 개수를 증가시키면서 Wu-Manber 알고리즘과 성능을 비교하고 분석한다. 끝으로 5절에서는 결론을 맺고 향후 개선해야 할 부분에 대하여 기술한다.

Ⅱ. 관련연구

2.1 Boyer-Moore 알고리즘

Boyer와 Moore에 의하여 제안된 이 알고리즘은 일반적인 응용에서 가장 효율적인 단일 패턴매칭 알고리즘으로 꼽힌다. Boyer-Moore 알고리즘은 찾고자 하는 패턴의 오른쪽에서부터 왼쪽으로 문자를 비교한다. 찾고자 하는 패턴에 매치가 되는 문자가 나오지 않는 경우에는 수식(1)을 이용하여 쉬프트 연산을 수행한다. 수식에서 i는 패턴에서 패킷의 페이로드 부분(이하 텍스트)과 일치하지 않는 문자가 나타나는 위치를 나타내고, k는 텍스트에서 패턴과 일치하지 않는 문자가 나타나는 위치를 나타낸다. $R(T(k))$는 텍스트에서 일치되지 않는 문자가 패턴에서 처음 나타나는 위치를 나타낸다.

$$shift = MAX[1, i - R(T(k))] \quad (1)$$



〔그림 2〕 Boyer-Moore 동작방식 예제

예를 들어, [그림 2]의 T와 같은 텍스트에서 P와 같은 패턴을 찾는다고 할 때, i는 패턴에서 일치하지 않는 문자가 나타나는 위치를 의미하므로 5가 된다. k는 T에서의 위치를 나타내므로 $T(k)$는 'a'가 되고 $R(T(k))$는 패턴에서 'a'가 나타나는 위치인 3이 된다. 수식에 의하여 2만큼 시프트하여 비교를 계속한다.

Boyer-Moore 알고리즘은 단일 검색 알고리즘으로 한 개의 패턴을 검색하는데 $O(mn)$(m은 패턴의 길이, n은 전체 텍스트의 길이)의 복잡도를 갖는다. 만약 p개의 패턴을 비교한다고 하면 $O(pmn)$의 복잡도가 된다.

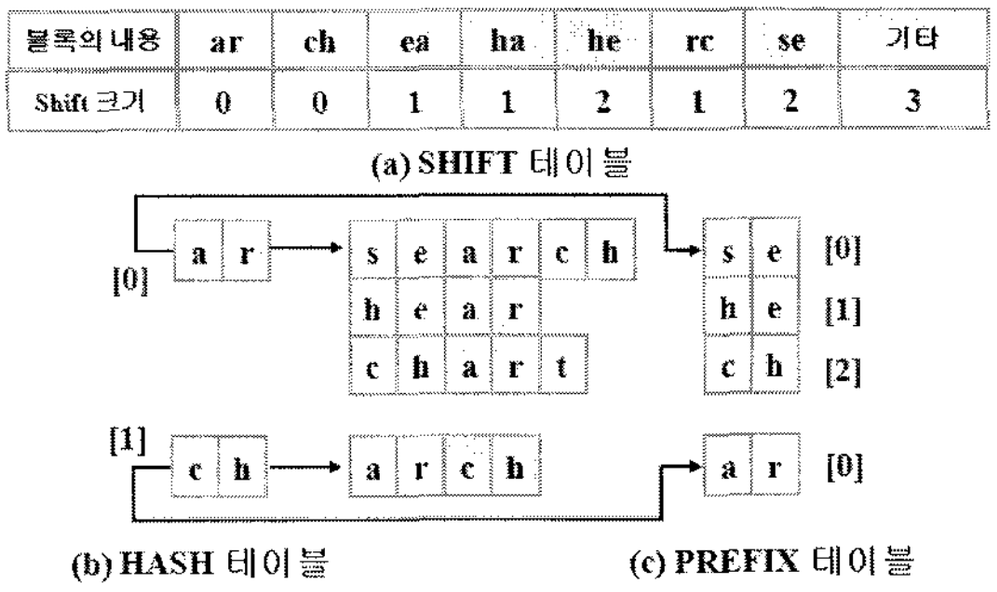
2.2 Wu-Manber 알고리즘

Wu-Manber 알고리즘은 단일 패턴매칭 알고리즘인 Boyer-Moore 알고리즘을 다중 패턴매칭이 가능하도록 개선한 것으로, 패턴매칭 성능을 개선하기 위하여 전처리 과정을 두고 SHIFT 테이블, HASH 테이블, PREFIX 테이블을 먼저 생성한다. SHIFT 테이블은 Boyer-Moore와 비슷한 불일치 문자 시프트(bad-character shift) 연산을 활용하여 생성한다. 그러나 Boyer-Moore 알고리즘과 같이 문자 하나씩을 비교하는 것이 아니라, 일정 크기의 블록 단위로 비교한다. 실제 패턴매칭 단계에서는 패턴을 검색하기 위해 HASH 테이블을 사용함으로써 매칭 시간을 줄인다. SHIFT 테이블의 시프트 값이 '0'인 경우엔 HASH 테이블과 PREFIX 테이블을 이용하여 매칭을 위한 후보를 결정하고 패턴매칭을 한다. 즉, 3단계를 거쳐서 최종 패턴매칭을 하게 된다. 예를 들어 {search, hear, arch, chart}의 패턴 집합이 있을 때 블록의 크기 B를 2로 하면 [그림 3]과 같이 SHIFT 테이블과, HASH 테이블, PREFIX 테이블이 생성된다.

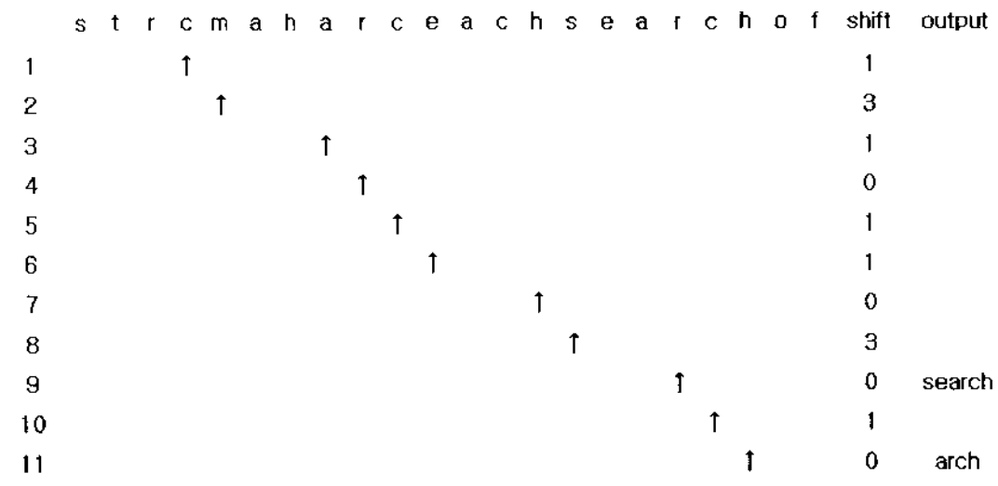
만약, 주어진 텍스트가 'strcmaharceachsearchof'라고 할 때, Wu-Manber 알고리즘을 이용하여 패턴매칭을 한다면, 검색 윈도우의 크기는 패턴의 최소길이인 4가 되고, 블록의 크기 B는 2 또는 3으로 할 수 있다. 본 절에서는 설명의 편의를 위해 블록의 크기를 2로 하였다.

[그림 4]는 Wu-Manber 알고리즘으로 [그림 3]과 같이 전처리 단계에서 생성된 패턴 테이블을 이용하여 실제 텍스트에서 패턴매칭하는 단계를 나타낸 것이다. 첫 번째로 SHIFT[rc]는 1이므로 다음으로 1만큼 시프트한다. 두 번째는 SHIFT[cm]이 3이므로 3만큼 시프트한다. 같은 방법으로 아홉 번째에는 SHIFT[ar]이 0을 가지므로 HASH 테이블과, PREFIX 테이블을 이용하여 'search'가 검색된다. 그리고 열 번째에서 SHIFT [rc]는 1이므로 1만큼 시프트하고 11번째에서 SHIFT [ch]이 0이되고 HASH 테이블과 PREFIX 테이블을 이용하여 'arch'를 찾는다.

Wu-Manber 알고리즘의 패턴매칭에 수반되는 오버헤드는 최선의 경우 $O(Bn/m)$(B는 블록의 크기, m은 패턴의 최소 길이, n은 전체 텍스트의 길이)의 복잡도를 갖는다. 이론상 패턴의 개수가 늘어난다 하더라도 시간 복잡도는 증가하지 않지만, 패턴의 최소길이에 영향



〔그림 3〕 Wu-Manber 알고리즘의 테이블



〔그림 4〕 Wu-Manber 패턴매칭

을 받는다. Wu-Manber 알고리즘은 평균적으로 가장 좋은 성능을 갖는 알고리즘의 하나이다[6].

2.3 QWM 알고리즘

QWM 알고리즘[7]은 Wu-Manber 알고리즘의 성능을 개선하기 위하여 시프트 크기를 증가시킨다. 검색단계에서 검색되는 패턴의 존재에 상관없이 1만큼 시프트되는 것을 개선하기 위하여 Quick Search 알고리즘[8]을 적용하였다. 즉 SHIFT 테이블에서 시프트 크기가 0이지만 패턴이 정확하게 일치하지 않을 경우 최대 패턴의 최소길이인 m만큼 시프트시킨다.

QWM 알고리즘에서는 Wu-Manber 알고리즘과 같은 방법으로 PREFIX 테이블과 HASH 테이블을 생성하지만, SHIFT 테이블은 Wu-Manber 알고리즘보다 1만큼 크게 시프트되도록 변경되는 대신에 패턴의 처음 B만큼의 문자열이 추가로 HEAD 테이블에 저장된다. [그림 5]는 {search, hear, arch, chart}를 패턴으로 할 경우 전처리 단계에서 새롭게 생성되는 HEAD 테이블과 변경되어야 하는 SHIFT 테이블을 나타낸다.

만약, 주어진 텍스트가 'strcmaharceachsearchof'라고 할 때, QWM 알고리즘을 이용하여 패턴매칭을 한다면, 검색 윈도우의 크기는 패턴의 최소길이인 4가 되고, 블록의 크기 B는 2가 된다.

[그림 6]은 QWM 알고리즘으로 전처리 단계에서 생성된 패턴 테이블을 이용하여 실제 텍스트에서 패턴매칭하는 단계를 나타낸 것이다. 첫 번째로 HEAD[st]는 0이고 SHIFT[cm]은 4이므로 4만큼 시프트한다. 두 번째에서도 HEAD[ma]가 0이고 SHIFT[ar]이 1이므로 1만큼 시프트한다. 같은 방법으로 여덟 번째에는 HEAD[se]가 1이므로 패턴이 정확하게 일치하는지 PREFIX 테이블과 HASH 테이블을 이용하여 검색하여 'search'를 검색한다. 다음으로 SHIFT[rc]가 2이므로 2만큼 시프트한다. 같은 방법으로 아홉 번째 단계에서 'arch'가 검색되어 Wu-Manber 알고리즘보다 빠르게 검색된다.

QWM 알고리즘은 찾고자 하는 패턴의 PREFIX 값이 일치하는 텍스트가 많고, 하나의 텍스트에 그 값이 여러 번 반복되어 나타나는 경우에 제한적으로 Wu-Manber 알고리즘보다 빠른 성능을 보인다. 그러나 일

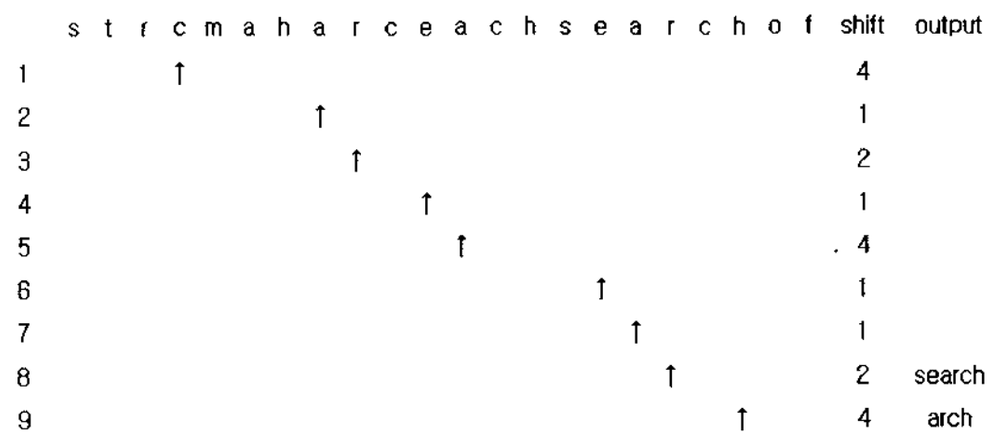
블록의 내용	ar	ch	ea	ha	he	rc	se	기타
Shift 크기	1	1	2	2	3	2	3	4

(a) SHIFT 테이블

패턴의 처음	se	he	ar	ch	기타
값	1	1	1	1	0

(b) HEAD 테이블

〔그림 5〕 QWM의 변경된 테이블



〔그림 6〕 QWM 패턴매칭

반적인 환경에서 QWM 알고리즘은 원래의 Wu-Manber 알고리즘과 비교하여 시프트 크기를 증가시키는 대가로 HEAD 테이블의 값을 검색하기 위한 시간이 소요되고 이에 따라 성능이 오히려 저하되기 때문에 NIDS에 적용하기에 적합하지 않다.

Ⅲ. 복수 부분패턴을 이용한 검색

Wu-Manber 알고리즘에서는 패턴매칭 속도를 향상시키기 위하여 휴리스틱을 이용한 시프트 연산과 테이블을 사용한다. Wu-Manber 알고리즘에서 SHIFT 테이블에서 가장 크게 시프트할 수 있는 거리는 $m-B+1$ (m:패턴의 최소길이, B:블록크기)이다. 그러나, 패턴의 개수가 많아지면서 SHIFT 테이블에서의 시프트 크기가 0 또는 1이 되는 경우가 상대적으로 많아지게 되고, 이에 따라 텍스트에서 패턴매칭을 하는데 평균 시프트 횟수나 패턴 비교횟수가 늘어나게 되어 성능이 저하되는 요인이 된다.

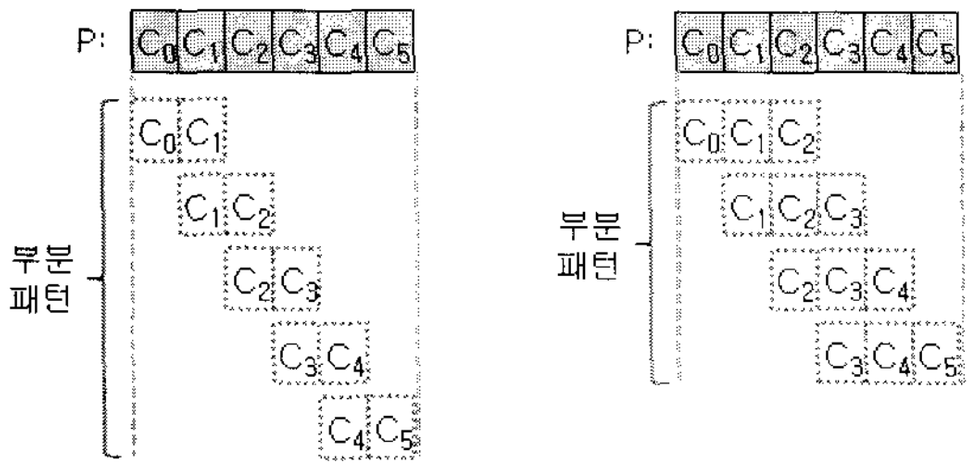
본 절에서는 시프트 크기를 증가시킴으로써 비교횟수를 줄이기 위하여 복수개의 비교 대상 블록을 고려하는 다중 패턴매칭 알고리즘을 제안한다. 본 절에서는 패턴을 분할하여 패턴매칭 목적의 블록으로 사용하는데 이를 부분패턴이라고 한다.

3.1 전처리 단계

본 논문에서는 패턴매칭에 따르는 비교하는 횟수를 줄이기 위하여 부분패턴의 개수를 증가시킴으로써 고정된 크기의 시프트를 보장할 수 있는 기법을 제안한다. 비교횟수를 줄이기 위하여 시프트를 $s(s \geq 2)$만큼 하기 위해서는 s개의 연속된 부분패턴이 필요하다. [그림 7]과 같이 길이가 6인 패턴 P를 $C_0C_1C_2C_3C_4C_5$로 정의 할 때, [그림 7(a)]와 같이 길이 2인 연속된 부분패턴은 C_0C_1, C_1C_2, C_2C_3, C_3C_4, C_4C_5 등의 5개가 존재한다. 텍스트 T에 패턴 P가 포함되어 있을 경우 시프트 크기를 5로 설정하더라도 C_0C_1, C_1C_2, C_2C_3, C_3C_4, C_4C_5 등 부분패턴중의 하나와 일치하는 것을 보장할 수 있다. 또한 [그림 7(b)]와 같이 길이 3인 연속된 부분패턴은 $C_0C_1C_2$, $C_1C_2C_3$, $C_2C_3C_4$, $C_3C_4C_5$ 등의 4개가 존재한다. [그림 7(a)]와 마찬가지로 텍스트 T에 패턴 P가 포함되어 있을 경우 [그림 7(b)]와 같이 시프트 크기를 4로 설정하더라도 $C_0C_1C_2$, $C_1C_2C_3$, $C_2C_3C_4$, $C_3C_4C_5$ 등의 부분패턴중에서 하나와 일치하는 것을 보장한다. 즉, 연속된 부분패턴의 수만큼 시프트하더라도 패턴매칭의 동작성을 항상 보장할 수 있다.

임의의 패턴 P_i의 길이를 $m_i(m_i \geq 2)$라 하고, 패턴 P_i에서 생성 가능한 부분패턴의 크기를 b, 부분패턴의 개수를 s라고 하면, 수식(2)와 같이 부분패턴의 크기와 부분패턴의 개수를 더한 값은 패턴의 길이에 1만큼 증가시킨 값보다 작거나 같은 범위에서 정할 수 있다. 즉, 임의의 패턴에 대하여 수식(2)가 성립하도록 b 또는 s를 최대화할 수 있다.

$$b+s \leq m_i+1 \tag{2}$$



(a) 2크기의 연속된 블록 (b) 3크기의 연속된 블록

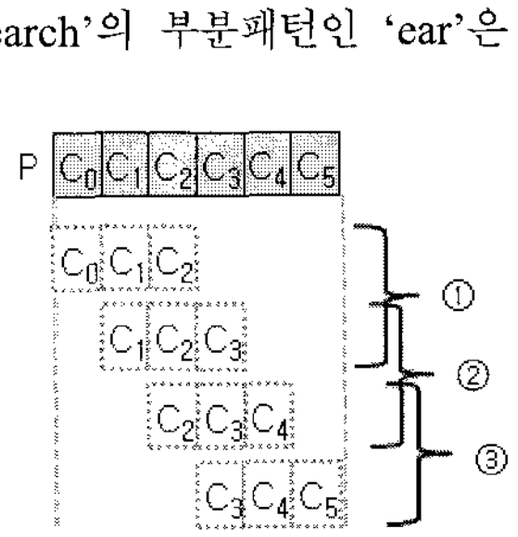
〔그림 7〕 연속된 부분패턴에 따른 검색

다중검색을 위해서 b를 고정시키려면 시프트 크기 s는 모든 패턴에 대하여 $s \leq m_i-b+1$을 만족시켜야 한다. 패턴의 길이 m_i에서 가장 작은 값을 $m_{\min}$이라고 할 때, $S=m_{\min}-b+1$로 정하면 S는 모든 패턴을 검색하기 위한 공통의 시프트 크기가 된다. 시프트 크기 S는 필요한 부분패턴의 수와 일치한다는 점에서 $m_i > m_{\min}$인 패턴의 경우 생성 가능한 부분패턴의 수인 m_i-b+1은 필요한 부분패턴의 수인 $m_{\min}-b+1$ 보다 크다. 이때, 패턴 P_i는 연속된 부분패턴에 대하여 선택의 문제가 발생한다.

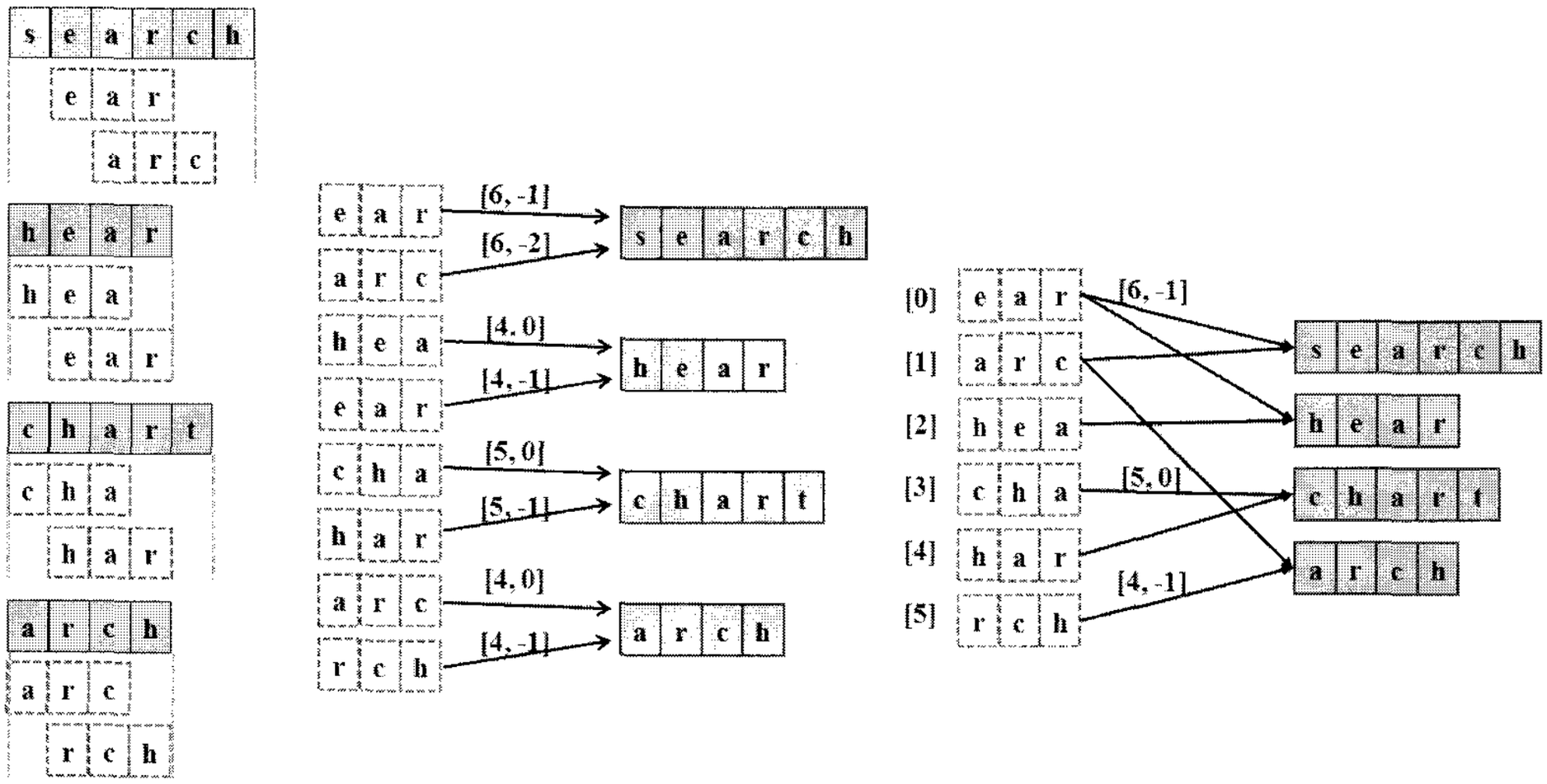
예를 들어, [그림 7]에서의 길이가 6인 패턴 P를 검색한다고 가정한다. 부분패턴의 크기 b를 $2 \leq b \leq 5$를 만족시키는 범위에서 3으로 결정하면, [그림 7(b)[에서 보인 것처럼 최대 4($s \leq m_i-(b-1)=6-(3-1)$)개의 연속된 부분패턴 $C_0C_1C_2$, $C_1C_2C_3$, $C_2C_3C_4$, $C_3C_4C_5$을 얻을 수 있다. 만약 $m_{\min}=4$인 패턴이 존재하면 필요한 부분패턴의 개수는 2($S=m_{\min}-(b-1)(=4-(3-1))$가 된다.

이때 연속된 2개의 부분패턴을 선택하는 방법은 [그림 8]의 ①, ②, ③ 등 세 개가 있다. 본 논문에서는 부분패턴의 연속을 선택할 때 ①과 ③처럼 양쪽 끝이 아닌 ②처럼 중앙 부분의 문자를 포함하는 것들을 취하기로 한다.

제안한 기법을 따르면 {search, hear, arch, chart}와 같이 4개의 패턴 집합이 있을 경우(최소길이 $m_{\min}=4$가 된다), 부분패턴의 크기 $b=3$으로 하면, [그림 9(a)]와 같이 각 패턴에서 2개씩의 연속된 부분패턴을 선택할 수 있다. 각 부분패턴들을 이용하여 패턴매칭을 하기 위하여 패턴의 길이 m_i 정보와 비교할 패턴에서 부분패턴의 위치정보인 d가 필요하다. 부분패턴이 일치하면 d만큼 이동한 후 m_i만큼 문자열을 비교함으로써 주어진 패턴의 존재 유무를 알 수 있다. 이에 따라 [그림 9(b)]와 같이 부분패턴에 [m_i, d] 정보를 추가시킨다. [그림 9(b)]에서 'search'의 부분패턴인 'ear'은 앞에 하나의



〔그림 8〕 연속된 부분패턴 선택



(a)부분패턴선택 (b)합병 전 (c)합병 후

〔그림 9〕 부분패턴의 선택과 합병

문자 's'가 있고 패턴의 길이가 6이므로 [6, -1]이 된다. 또한 'arc'는 앞에 두 개의 문자 'se'가 있고 패턴의 길이가 6이므로 [6, -2]가 된다. 이와 같이, 선택된 부분패턴의 검색정보를 추가하면 [그림 9(b)]와 같다. 이 때 중복된 부분패턴들을 병합하면 [그림 9(c)]와 같은 최종적인 형태를 얻을 수 있다. 이해의 편의를 위하여 [그림 9(c)]에는 검색정보를 일부 생략하였다.

3.2 패턴매칭 단계

패턴매칭 단계에서는 전처리 단계에서 생성된 부분패턴의 합병된 테이블을 이용한다. 패턴매칭을 위하여 텍스트의 처음부분부터 부분패턴의 크기 b 만큼의 문자를 선택하여 부분패턴들과 비교하고, S만큼 시프트하여 반복한다. 만약 부분패턴이 일치하면 부분패턴의 추가정보인 $[m_i, d]$를 이용하여 현재 텍스트의 오프세트에서 d만큼 이동한 후 m_i 길이만큼 패턴이 정확하게 매칭되는지 확인한다.

예를 들어, 텍스트에서 부분패턴의 크기 b 만큼씩을 검색하다가 'ear'을 발견하면 현재 텍스트의 오프세트에서 -1하여 6개의 문자를 비교하여 'search' 패턴의 검색 여부를 결정하며, 현재 텍스트의 오프세트에서 -1하여 4개의 문자를 비교하여 매칭이 되면 'hear' 패턴이 검색된 것으로 판단한다. 만약 부분패턴의 테이블에서 패턴이 존재하지 않는 경우 고정된 크기 S만큼 시프트할 수 있다.

패턴매칭 방법을 알고리즘으로 표현하면 다음과 같다.

```
(00) while( text <= textend ) {
(01)   subPattern = mksub(text, b);
       // 텍스트에서 블록의 크기 b만큼 선택
(02)   p = subPatternList[subPattern];
       // subPattern에 매칭되는 부분패턴의
       // 리스트가 존재하는지 확인
(03)   while( p != 0 ) {
       // 부분패턴이 존재하는 경우
(04)    matched = pattSearch(p->m_i, p->d);
        // 부분패턴의 추가정보를 이용하여
        // 텍스트의 현재 위치에서 d만큼
        // 오프세트를 이동하고 m_i 만큼 비교
(05)    if( matched ) {
        // 패턴매칭이 되는 경우
(06)     report the matching result;
         // 패턴매칭 결과를 보고
(07)     p = p->next;
         // 부분패턴 리스트의 다음으로 이동
(08)    }
```

(09) text += *S* // *S*만큼 시프트
(10) }
(11) }

(00) 텍스트의 끝까지 패턴매칭을 진행한다. (01) 텍스트에서 부분패턴의 길이 b 만큼의 문자를 선택하고 (02) 부분패턴의 합병된 테이블에서 매칭이되는 부분패턴의 리스트가 존재하는지 확인한다. (03) 만약 일치하는 부분패턴이 존재하게 되면, (04) 전처리 단계에서 생성된 추가적인 정보를 이용하여, 패턴매칭을 한다. (05) 만약 정확하게 패턴매칭이 되는 경우 (06) 결과를 보고한다. (07) 부분패턴 리스트의 다음으로 이동하여 다시 (03)에서 (06)의 과정을 반복한다. 만약 리스트에 더 이상 부분패턴이 존재하지 않으면, (09) 텍스트에서 고정된 크기 *S*만큼 시프트하여 (00)부터 반복한다.

패턴 집합 {search, hear, arch, chart}과 텍스트 'strcmaharceachsearchof'을 $m_{\min}=4$, $b=3$으로 하여 전처리를 하면 [그림 9(c)]와 같은 부분패턴의 합병된 테이블이 생성된다. [그림 10]은 제안된 알고리즘을 이용한 패턴매칭 단계를 보인다. 아홉 번째 단계에서 'search'와 'arch'가 검색된다.

	s	t	r	c	m	a	h	a	r	c	e	a	c	h	s	e	a	r	c	h	o	f	shift	output
1		↑																					2	
2			↑																				2	
3				↑																			2	
4					↑																		2	
5						↑																	2	
6							↑																2	
7								↑															2	
8									↑														2	
9										↑													2	search, arch

〔그림 10〕 패턴매칭 단계

3.3 부분패턴 비교 성능 개선

본 논문에서 제시된 방법은 패턴매칭 단계에서 텍스트의 일부가 부분패턴과 일치하는 경우에만 이 부분패턴을 포함하는 전체 패턴이 매칭되는지 검증하므로, 전처리 단계에서 부분패턴을 비교할 때는 정확성보다 빠른 처리율이 중요하다. 이런 목적에 따라 제안된 방법은 해싱함수를 이용하여 부분패턴의 테이블을 구성함으로써 전처리 단계에서의 부분패턴 처리 성능을 개선한다. 본 논문에서는 전처리 단계에서 필요한 부분패턴들을 정수 크기에 맞춰 16비트 크기로 해싱하였다.

Ⅳ. 시 험

본 절에서는 Snort의 패턴 집합과 텍스트 파일을 이용하여 제안된 방법의 규칙 적용 성능을 시험하고 Wu-Manber 알고리즘과 비교한다. Wu-Manber 알고리즘은 Agrep[9]에서 사용된 코드를 이용하였다. 성능을 측정하기 위한 패턴은 Snort 규칙 집합에서 'content' 옵션이 존재하는 규칙을 대상으로 'content' 옵션에 속하는 패턴의 길이가 4 이상인 것을 추출하였다. 2.1절에서 기술한 것과 같이 Snort 규칙 집합에서는 'content' 길이가 4 이상인 패턴이 전체의 95% 이상 대부분을 차지하고 있기 때문에 이를 대상으로 규칙 적용 성능을 개선할 수 있음을 보인다. 패턴매칭을 위해 사용될 텍스트는 구텐베르그 프로젝트의 e-Book[10]에서 다운로드가 가장 많은 상위 100개중에 6개 파일을 선택하여 생성한 약 10.8M 바이트(약 212,000 라인) 크기의 파일과 텍스트 파일로 된 약 4.0M 바이트(약 30,382 라인) 크기의 성경파일을 사용한다.

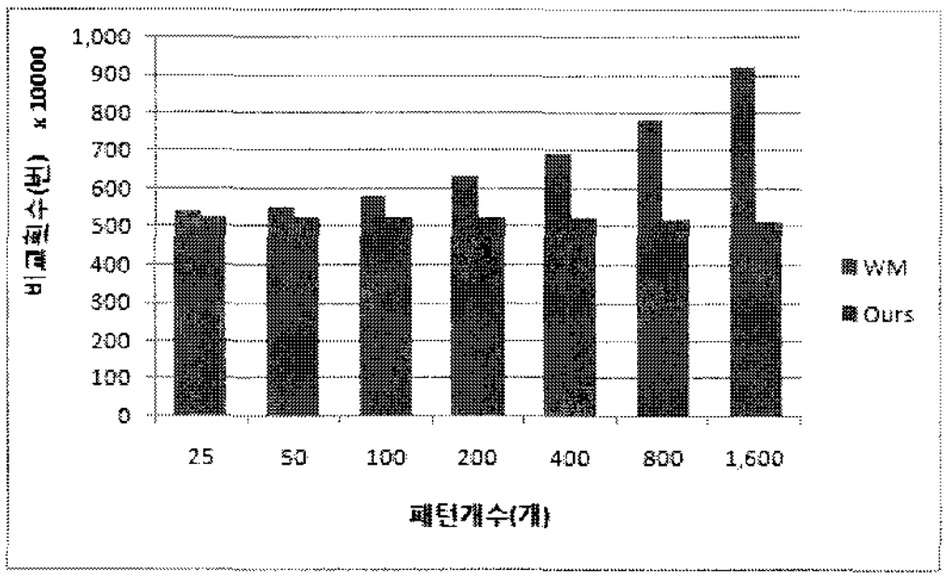
시험을 위한 시스템 환경은 다음과 같다.
CPU : Intel Core 2 Duo 6300, 1.86 GHz
메모리 : 1G Bytes
운영체제 : Linux (Ubuntu 5.10)
샘플파일(1) : 10.8M 바이트 e-Book 텍스트 파일
샘플파일(2) : 4.0M 바이트 성경 텍스트 파일

시험에서 부분패턴을 위한 해시 함수는 16비트 값을 생성한다.

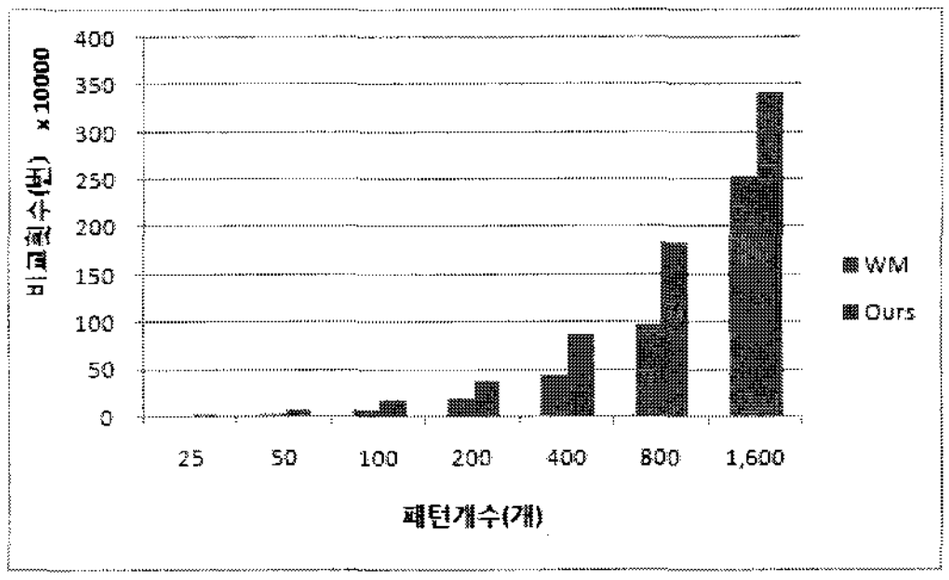
4.1 패턴개수 증가에 따른 성능분석

Wu-Manber 알고리즘과 제안한 알고리즘은 공통적으로 해싱함수를 적용하고 선택적으로 패턴 전체를 매칭하므로 해싱함수를 이용한 비교 횟수와 패턴 전체를 비교하는 횟수를 비교하여 성능을 분석한다.

[그림 11]은 Snort 규칙집합에서 추출한 패턴의 개수를 25, 50, 100, 200, 400, 800, 1600개로 증가시키면서

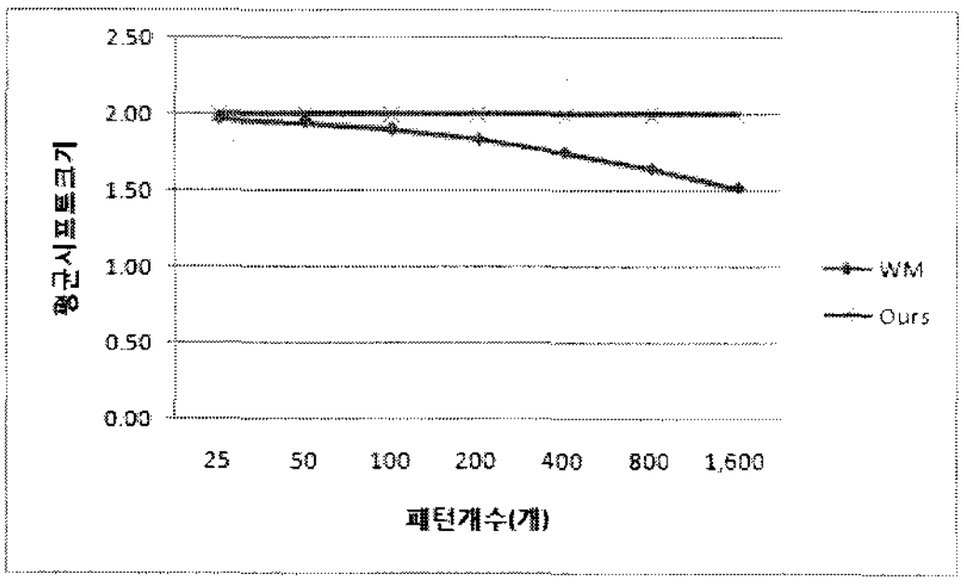


(a) 해싱함수를 이용한 비교 횟수



(b) 패턴 전체 비교 횟수

〔그림 11〕 패턴매칭에 소요되는 시간



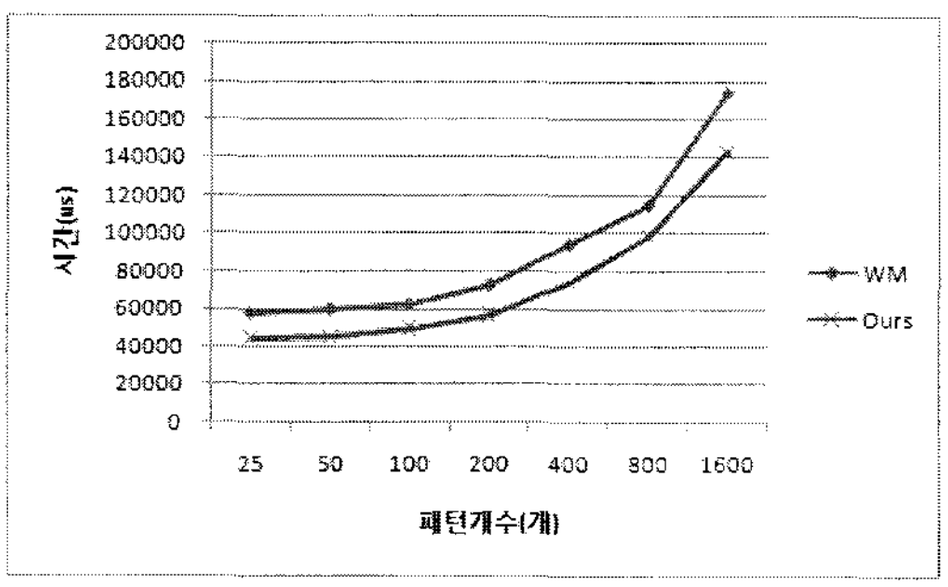
〔그림 12〕 패턴개수 증가에 따른 평균 시프트 크기

샘플파일(1)에 패턴매칭을 적용한 결과이다. Wu-Manber 알고리즘에서는 HASH 테이블과 PREFIX 테이블을 이용하기 위하여 해싱함수를 적용하며 본 논문에서 제안한 알고리즘에서는 부분패턴의 존재 유무를 결정하기 위하여 해싱함수를 적용한다.

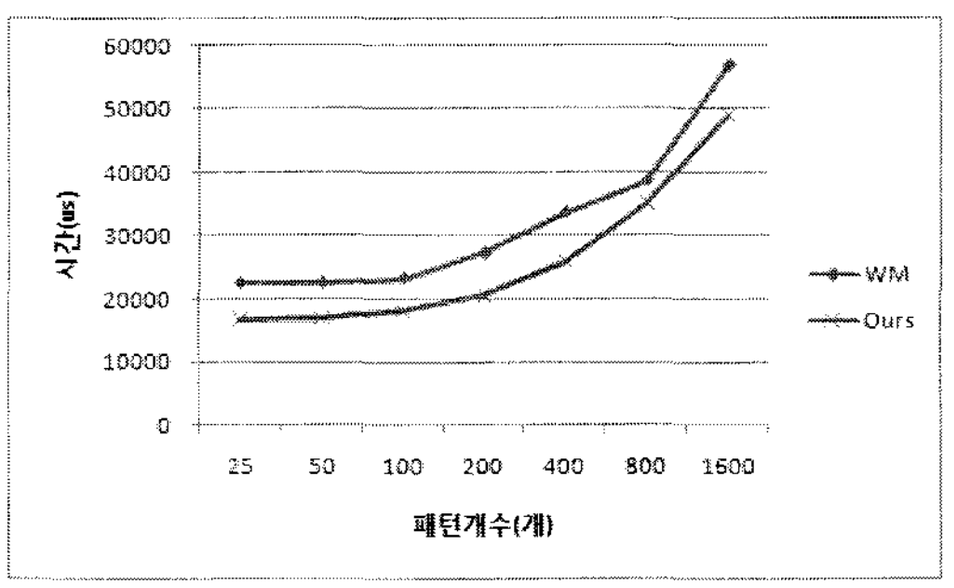
[그림 11(a)]에 나타난 것과 같이 제안한 알고리즘에 의한 부분패턴의 해싱함수를 이용한 비교 횟수는 패턴의 개수가 증가하더라도 크게 변화가 없는 반면에 Wu-Manber 알고리즘은 패턴의 개수가 증가하면서 비교 횟수가 크게 늘어난다.

[그림 11(b)]는 패턴의 개수를 늘리면서 패턴 P_i 전체를 비교하는 횟수를 보인다. Wu-Manber 알고리즘에 비해 제안한 알고리즘의 비교횟수가 많은 이유는 Wu-Manber 알고리즘의 경우 두 번의 해싱함수 적용을 거치면서 패턴 P_i 전체 비교에 따르는 횟수를 감소시켰기 때문이다. [그림 11]의 범례에 있는 WM는 Wu-Manber 알고리즘에 의한 것이고 Ours는 제안한 알고리즘에 의한 것이다.

[그림 12]는 Wu-Manber 알고리즘과 제안된 알고리즘을 샘플파일(1)에 적용할 때 블록과 부분패턴에서의 평균 시프트 크기를 나타낸다. Wu-Manber 알고리즘의 경우 패턴의 개수가 많아지면 평균 시프트 크기가 줄어든다. 이는 패턴의 개수가 증가하게 되면 SHIFT 테이블의 크기가 커지게 되고 블록의 종류도 다양해지기 때문이다. 그렇지만, 제안된 알고리즘의 경우 고정된 크기로 시프트하므로 패턴의 개수가 많아지더라도 영향을



(a) 샘플파일(1)



(b) 샘플파일(2)

〔그림 13〕 패턴매칭에 소요되는 시간

받지 않음을 볼 수 있다.

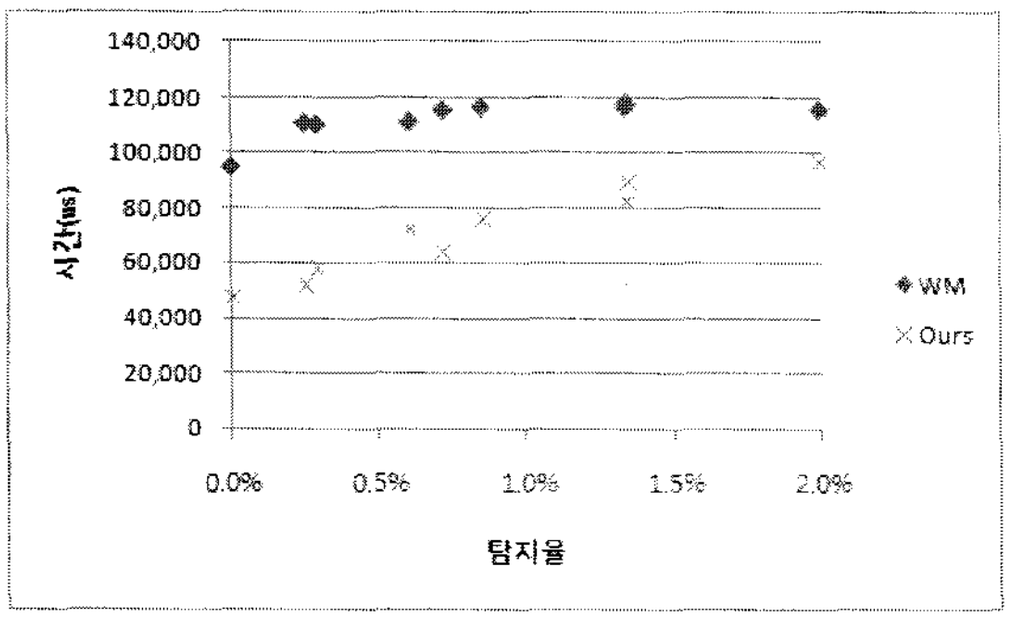
다음으로 패턴매칭에 따르는 수행시간을 비교하기 위하여 Wu-Manber 알고리즘과 제안된 기법을 구현하고 패턴 매칭에 소요되는 시간을 측정하였다. [그림 13]은 Snort 규칙집합에서 추출한 패턴의 개수를 증가시킬 때 두 개의 샘플파일에 대한 패턴매칭 시간을 그래프로 나타낸 것이다. 단 전체 텍스트 파일에서 패턴매칭이 일어날 확률은 5%가 넘지 않도록 패턴을 추출하였다.

[그림 13]의 결과와 같이 제안한 알고리즘은 전체적으로 Wu-Manber 알고리즘의 패턴매칭에 소요되는 시간 기준으로 샘플파일(1)의 경우 15~25% 정도, 샘플파일(2)의 경우 20~25% 정도 감소시킨다.

4.2 탐지율에 따른 성능분석

일반적으로 정상적인 인터넷 환경에선 비정상적인 패턴에 대한 탐지율이 매우 낮지만 비정상 트래픽이 유입되면 탐지율이 높아진다. 패턴매칭 알고리즘은 패턴매칭이 일어나는 빈도에 따라, 즉 탐지율에 따라 성능의 차이가 나타날 수 있다. 즉 특정 탐지율 영역에서만 효과적으로 동작할 수도 있다. 본 절에서는 탐지율 변화에 따르는 패턴매칭의 시간을 측정하여 탐지율의 변화가 성능에 미치는 영향을 분석한다.

[그림 14]는 패턴의 개수를 800개로 하고 탐지율 변화에 따라 Wu-Manber 알고리즘과 제안한 알고리즘의 패턴매칭에 소요되는 수행시간을 측정한 것이다. 그림에서와 같이 패턴의 탐지율이 높아지면 제안한 알고리즘의 수행시간이 다소 늘어나지만 0%부터 2% 구간의 전 영역에 걸쳐서, 즉 비정상 트래픽이 적은 경우와 상대적으로 많은 경우을 구분하지 않고 제안된 알고리즘이 Wu-Manber 알고리즘보다 수행시간을 단축할 수 있음을 확인할 수 있다.



〔그림 14〕 탐지율 변화에 따른 패턴매칭 시간

V. 결 론

웜 및 바이러스 등 악성 코드 및 해킹 공격이 많아지면서 네트워크의 안전성을 보장하기 위해 사용되는 NIDS와 같은 시스템에서 악성 코드나 행위를 탐지하기 위한 규칙들도 많아지게 되었다. 그리고 규칙들을 비교하는 알고리즘에서 70% 이상의 시간을 패턴매칭에 사용하고 있다.

본 논문에서는 패턴매칭 성능을 개선하기 위하여 복수개의 부분패턴을 사용하는 기법을 제안하였다. 각 패턴에 대하여 *S*개의 부분패턴을 생성하여 고정된 크기 *S*만큼의 시프트가 가능하도록 함으로써 패턴의 개수가 증가하더라도 성능감소 정도를 줄일 수 있었다.

시험을 통하여 Wu-Manber 알고리즘과 수행시간 및 평균 시프트 크기 등을 비교한 결과, 제안한 알고리즘이 Wu-Manber 알고리즘에 비해 패턴매칭을 수행하는 시간이 평균 20% 정도 단축된 것을 확인할 수 있었다.

참고문헌

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

[2] S. Antonatos, K. G. Anagnostakis, and E. P. Markatos, "Generating realistic workloads for network intrusion detection systems", *ACM Workshop on Software and Performance*, 2004.

[3] R. S. Boyer and J. S. Moore, "A Fast String Searching Algorithm," *Communications of the ACM*, 1977.

[4] A. V. Aho, and M. J. Corasick, "Efficient String Matching:An Aid to Bibliographic search", *Communications of the ACM*, 18 June 1975.

[5] Sun Wu and Udi Manber, "A Fast Algorithm for Multi-Pattern Searching", in Technical Report TR 94-17, University of Arizona at Tuscan, May. 1994.

[6] Natan Tuck, Timothy Sherwood, Brad Calder,

et al., "Deterministic Memory Efficient String Matching Algorithms for Intrusion Detection" *in Proceeding of IEEE Infocom*, Hong Kong, Match 2004.

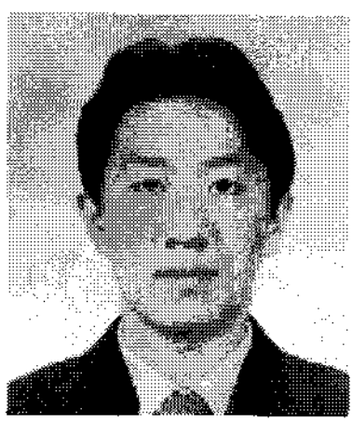
[7] Yang Dong Hong, Xu Ke, and Cui Yong, "An Improved Wu-Manber Multiple Patterns Matching Algorithm," in *Proceeding on Performance, Computing, and Communications Conference IPCCC 2006)*, April 2006.

[8] Sunday DM, "A Very Fast Substring Search Algorithm," in *Communica- tions of the ACM*, 1990.

[9] Udi Manber, AGREP, an approximate GREP, http://www.tgries.de/agrep/, 2005.

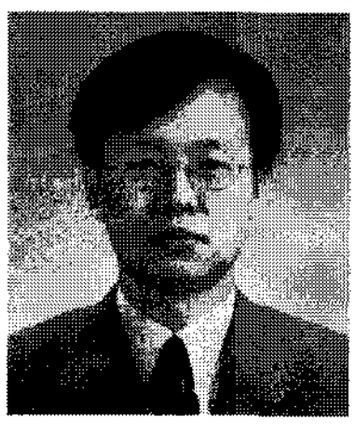
[10] 구텐베르그 프로젝트, http://www.guten- berg.org.

〈著者紹介〉



이 재 국 (Jae-Kook Lee) 학생회원

2002년 2월 : 충남대학교 컴퓨터과학과 졸업
2004년 2월 : 충남대학교 컴퓨터과학과 석사
2004년 3월~현재 : 충남대학교 컴퓨터공학과 박사과정
<관심분야> 인터넷보안



김 형 식 (Hyong-Shik Kim) 종신회원

1988년 2월 : 서울대학교 컴퓨터공학과 졸업
1990년 2월 : 서울대학교 컴퓨터공학과 석사
1997년 8월 : 서울대학교 컴퓨터공학과 박사
1997년 12월~1999년 8월 : 미국 UAH, Faculty Research Associate
1999년 9월~현재 : 충남대학교 전기정보통신공학부 컴퓨터전공 부교수
<관심분야> 인터넷보안, 컴퓨터시스템구조