Probabilistic Model for Performance Analysis of a Heuristic with Multi-byte Suffix Matching

Yoon-Ho Choi[1]
[1] Kyonggi University,
Suwon, 443-760, Korea
[e-mail: ychoi@kyonggi.ac.kr]
*Corresponding author: Yoon-Ho Choi

Received December 13, 2012; revised March 15, 2013; accepted April 5, 2013; published April 30, 2013

Abstract

A heuristic with *multi*-byte suffix matching plays an important role in real pattern matching algorithms. By skipping many characters at a time in the process of comparing a given pattern with the text, the pattern matching algorithm based on a heuristic with *multi*-byte suffix matching shows a faster average search time than algorithms based on deterministic finite automata. Based on various experimental results and simulations, the previous works show that the pattern matching algorithms with multi-byte suffix matching performs well. However, there have been limited studies on the mathematical model for analyzing the performance in a standard manner. In this paper, we propose a new probabilistic model, which evaluates the performance of a heuristic with multi-byte suffix matching in an average-case search. When the theoretical analysis results and experimental results were compared, the proposed probabilistic model was found to be sufficient for evaluating the performance of a heuristic with suffix matching in the real pattern matching algorithms.

Keywords: Heuristic matching, suffix matching, probabilistic model, average-case search, pattern matching

http://dx.doi.org/10.3837/tiis.2013.04.006

1. Introduction

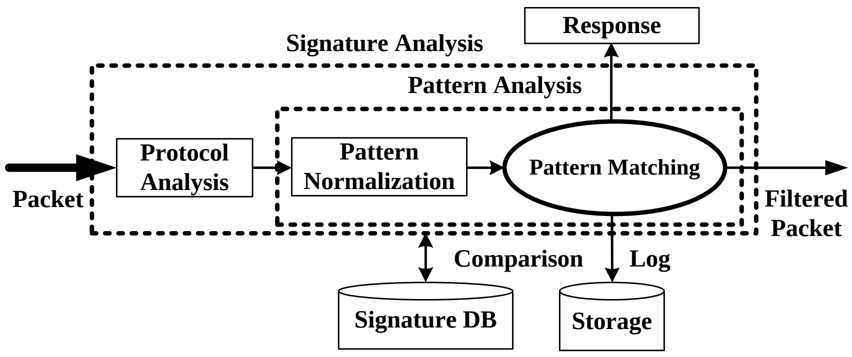

Fig. 1. Signature Analysis for Deep Packet Inspection

Many computer systems for deep packet inspection, especially intrusion detection systems, employ one or multiple pattern matching algorithms to search for signatures within the packet payload, which will be refered to as a text in this paper. As shown in **Fig. 1**, by comparing the contents in the text with the set of signatures using protocol analysis, pattern normalization and pattern matching algorithms, the signature analysis can interpret a certain series of packets, where the 'pattern' means a specific content in a signature. Given a packet, the protocol analysis module identifies and classifies the packet based on the protocol information and then, the normalization module decodes encoded payloads. After classifying packets that constitute an event, e.g., as normal or abnormal in an intrusion detection system, by using pattern matching algorithms, the signature analysis triggers a response to any abnormal packets and saves the corresponding event logs.

Given a pattern, a pattern matching algorithm finds all occurrences of the set of patterns within a text. Pattern matching algorithms can be classified into two types based on the number of matched patterns: single pattern matching algorithms and multi-pattern matching algorithms [1]-[16]. As an extension of the single pattrn matching algorithm, multi-pattern matching algorithms find all occurrences of the set of patterns within a text at a time. On the other hand, based on the data structure used for matching, signature matching algorithms can also be classified into four categories [17]: automaton-based, heuristics-based, hashing-based, and bit-parallelism-based.

Among the pattern matching algorithms, the pattern matching algorithm based on a heuristic with suffix matching, also called the shift-based algorithm, shows a faster average search time than the other algorithms [1]-[4]. This is because the shift-based algorithms can skip (shift) many characters at a time when comparing a given pattern with the text [5]-[14]. To realize such a heuristic with suffix matching, the skip-based algorithms use a bad-character shift table, each index of which consists of either a *single*-byte search unit or *multi*-byte search unit and determines how many characters in the text can be skipped when mismatching characters (called 'bad characters') are found in the text. Due to this heuristic, while scanning the text, the text is shifted and then, the bad characters are aligned in the rightmost position, where the bad characters appear in the text.

Compared to the skip-based algorithms with a *single*-byte search unit, the algorithms that use a *multi*-byte search unit can perform well even when thousands of the patterns are given. This is because the *multi*-byte search unit can limit the decrease in performance due to the large frequency of the last character in the thousands of patterns [14]. The skip-based algorithms with a *multi*-byte search unit can be further classified into those with a heuristic with multi-byte *prefix* matching and those with a heuristic with multi-byte *suffix* matching. While the FNP algorithm uses the so called shift distance table (SDT), which uses *prefix* sliding window (PSW) of window size w ($w \geq 2$), the Modified Wu-Manber (MWM) algorithm [13] uses a bad-character shift table, which uses *suffix* sliding window of window size B ($B \geq 2$). To provide the best average-case performance of the naive skip-based algorithm in deep packet inspection, the algorithm with a heuristic of *multi*-byte *suffix* matching has been widely used [16].

Previous studies that compared various experimental results and simulation results [11]-[16], have demonstrated that the skip-based algorithms with *multi*-byte suffix matching performas well. However, the performance of the algorithms can vary in the presence of different factors such as traffic characteristics, pattern characteristics, the length of the shortest pattern and so on. Also, it is difficult to collect and analyze full packet traces in depth because of privacy issues. Thus, a mathematical model is needed to examine the performance of various skip-based algorithms in a standard manner.

The main goals of this paper can be summarized as follows: (1) We propose a new mathematical model for evaluating the performance of a heuristic with *multi*-byte suffix matching. Since the performance of the skip-based pattern matching algorithms varies depending on the characteristics of a heuristic, the proposed model estimates the probabilistic performance of the skip-based pattern matching algorithms in an average-case search; (2) To the best of our knowledge, the proposed model is the first mathematical model that analyzes the performance of a heuristic with *multi*-byte suffix matching; (3) Based on a comparison between theoretical analysis results and experimental results under different lengths of the shortest pattern, the different numbers of signatures and the length of the text, we show that the proposed probabilistic model can be useful for estimating the performance of the real pattern matching algorithm based on a heuristic with *multi*-byte suffix matching.

The rest of the paper is organized as follows. In section 2, we provide an overview on related studies. In section 3, we describe the overall procedure of a heuristic with *multi*-byte suffix matching, and then describe the proposed probabilistic model for measuring the performance of the heuristic in section 4. After evaluating the proposed model in section 5, conclusions are provied in section 6.

2. Related Work

Research on performance measurements of the pattern matching algorithms can be classified into two types: one is based on experimental analysis and the other is based on theoretical analysis.

In experiments on different settings such as traffic characteristics, packet payloads, rulesets and processor architecture, S. Antonatos et. al. [16] measured the performance of a network intrusion detection system and found that the performance of the pattern matching algorithms were sensitive to traffic characteristics, processor architecture, packet content and ruleset content, and varies according to ruleset and packet size. The performance of many other pattern algorithms [1]-[16] has also been evaluated using experiments and simulations under various conditions. However, the algorithms were designed specifically for the particular text processing domain, such as an intrusion detection system and word processor and thus, evaluation of the algorithms has usually been constrained to the specific application domain. In a study on the performance measurement based on theoretical analysis, M.Fish et.al. in [11] proposed a probabilistic model that examined the average performance of a pattern matching algorithm, called the Set-wise Boyer-Moore-Horspool(SBMH) algorithm. The SBMH algorithm scans the text using a heuristic with *single*-byte suffix matching. The performance of the SBMH algorithm was measured in terms of the number of characters that must be examined per character of shift. It is worth noting that the number of characters that must be examined per character of shift can vary according to the size of search unit [5]. Thus, the model for a heuristic with *single*-byte suffix matching is not adequate to analyze the performance of the pattern matching algorithms based on a heuristic with *multi*-byte suffix matching. Since most pattern matching algorithms have been designed with *multi*-byte search unit to improve the search performance, it is essential to exactly estimate the performance of a heuristic with *multi*-byte suffix matching.

In a rough analysis for evaluating the performance of such pattern matching algorithms, Wu and Manber [15] showed that by assuming that both the text and the patterns were random strings with uniform distribution, the expected running time was less than linear in the size of the text. In this paper, we utilized the same assumptions as those of [15] and, proposed a good-enough probabilistic model that estimates the average performance of a heuristic with *multi*-byte suffix matching. Since it has been shown that when the size of search unit is two, the average performance is maximized from experiments, we developed a mathematical analysis model by focusing on *two*-byte search unit and demonstrated that we can model the general performance of a heuristic with *multi*-byte suffix matching.

3. Heuristic with Multi-byte Suffix Matching

The overall procedure for a heuristic with multi-byte suffix matching consists of two stages: (1) the preprocessing stage; (2) the scanning stage. In the preprocessing stage, the heuristic constructs the bad-character shift table by using the patterns in the rule sets. By using the given shift table, the heuristic scans the text to find the matching suffix in the scanning stage.

(1) Preprocessing procedure for constructing the bad-character shift table

A. From the set of patterns, save the length of the shortest pattern(*m*).

B. Find *m* characters starting from position 0.

C. Construct the bad-character shift table, each of whose indexes has the multi-byte search unit.

(2) Scanning procedure for finding patterns in the text by using the bad-character shift table

A. If the shift value of a multi-byte block in the text is zero in the bad-character shift table, find the pattern candidates which need exact matching.

i. Check the actual pattern candidates against the text directly.

ii. Move the current pivot by as much as one byte to the right direction in the text

B. Otherwise, move the current pivot by as much as the shift value in the bad-character shift table to the right direction in the text

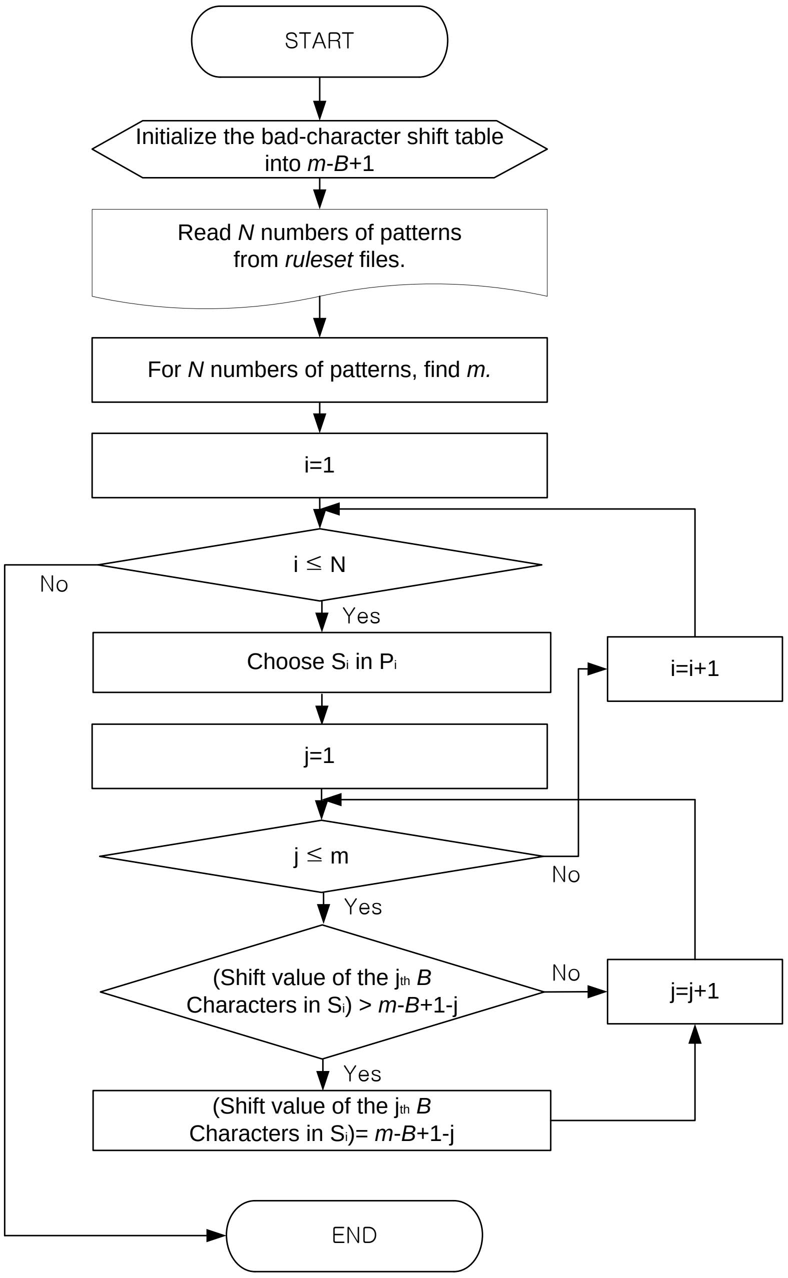
START
Initialize the bad-character shift table into m-B+1
Read N numbers of patterns from ruleset files.
For N numbers of patterns, find m.
i=1
i ≤ N
No
Yes
Choose Si in Pi
i=i+1
j=1
j ≤ m
No
Yes
(Shift value of the jth B Characters in Si) > m-B+1-j
No
j=j+1
Yes
(Shift value of the jth B Characters in Si)= m-B+1-j
END

Fig. 2. Flowchart Used for Constructing the Bad-Character Shift Table

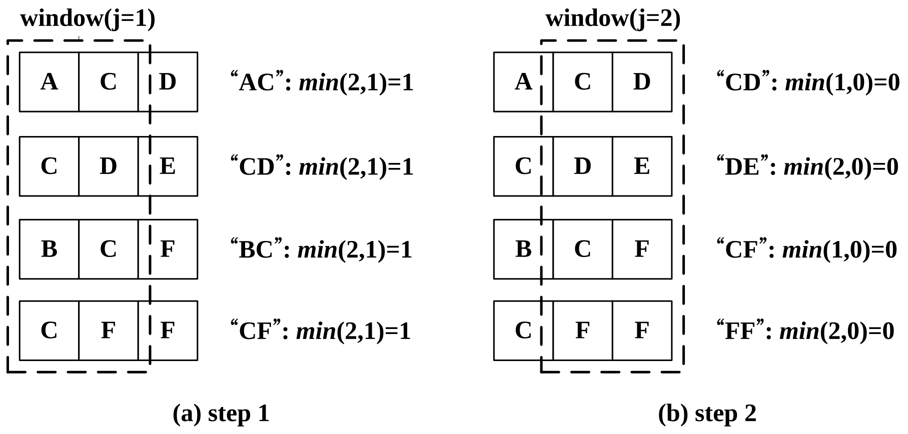

Fig. 3. Computation of Bad-Character Shift Values(m=3, B=2), where the 'dotted box' indicates a window of size B

Table 1. Bad-Character Shift Table(B=2, P={P_1, P_2, P_3, P_4})

Index	AA	...	AC	...	BC	...	CD	...
Value	2	2	1	2	1	2	0	2
Index	...	CF	...	DE	...	FF	...	ZZ
Value	2	0	2	0	2	0	2	2

To describe each stage in details, we defined the following parameters:

- N: Number of patterns
- P: Finite set of patterns P_i, where P={P_1, P_2, ..., P_N} and i∈{1, 2, ..., N}
- m: Length of the shortest pattern
- S: Finite set of the leftmost strings S_i of the length m in patterns P_i, where S={S_1, S_2, ..., S_N}
- ∑: Finite set of alphabets, |∑ |=a($a \geq 0$)
- B: Size of a search unit($B \geq 2$)
- t: Length of matching suffix
- T: Number of consecutive B-byte blocks generated from N patterns
- n: Length of the text

3.1 Preprocessing Procedure

By using the leftmost string of length m in each pattern, we construct the bad-character shift table. To skip mismatching characters of length m in the text, the initial value of the bad-character shift table is set into m-B+1. From the analysis of the given set of patterns P, the

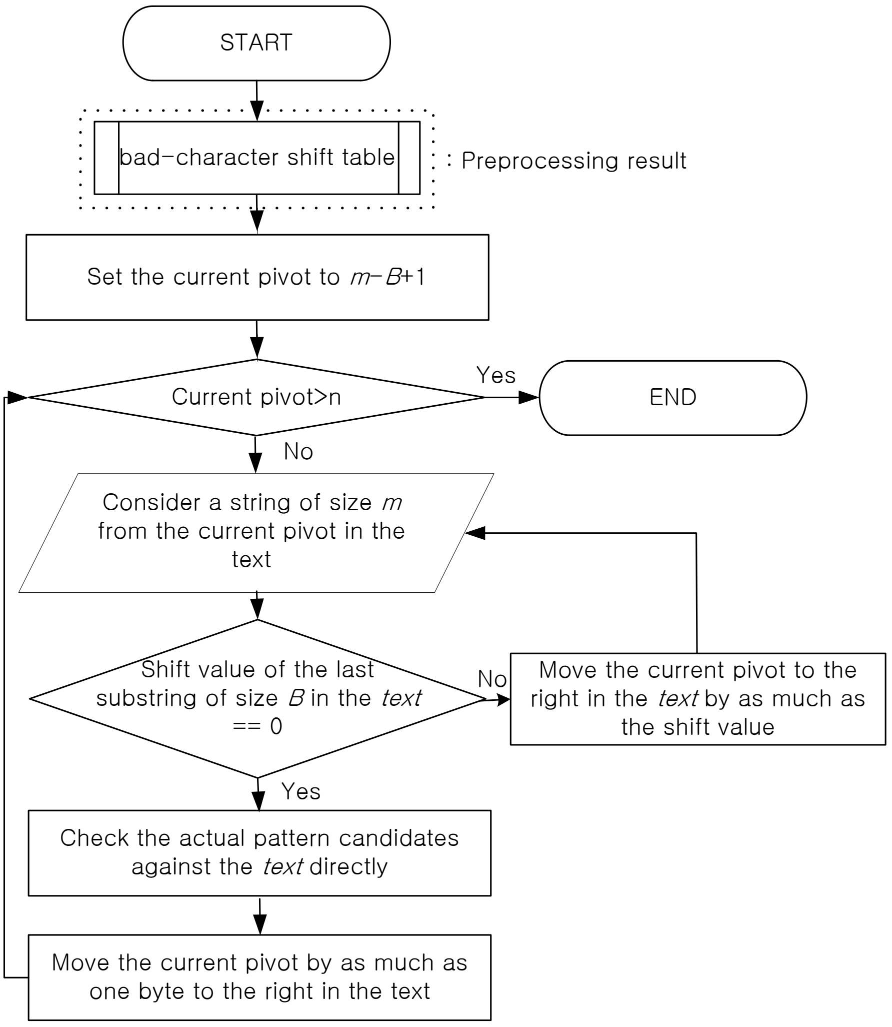
START
bad-character shift table
: Preprocessing result
Set the current pivot to m−B+1
Current pivot>n
Yes
END
No
Consider a string of size m from the current pivot in the text
Shift value of the last substring of size B in the text == 0
No
Move the current pivot to the right in the text by as much as the shift value
Yes
Check the actual pattern candidates against the text directly
Move the current pivot by as much as one byte to the right in the text

Fig. 4. Flowchart Used for Scanning the text by Using the Bad-Character Shift Table

value of each index in the shift table is renewed by setting its value to the minimum value by comparing to its current value. The details of the preprocessing operation for constructing the bad-character shift table are shown in **Fig. 2**.

For example, let us assume B=2, N=4 and P={P_1, P_2, P_3, P_4}, where P_1="ACDEH", P_2="CDE", P_3="BCFGBC" and P_4="CFFGB". For the given m=3, we find the leftmost strings of length three from every pattern: "ACD" from P_1, "CDE" from P_2, "BCF" from P_3, and "CFF" from P_4. First, if the string of B characters in the text does not exist in any of the patterns, we can shift by m-B+1 [13]. Thus, we set the initial value of the bad-character shift table. Second, from B characters in the leftmost string, as shown in **Fig. 3(a)**, we can update shift values for "AC", "CD", "BC", and "CF" into two to one. Also, as shown in **Fig. 3(b)**, by moving the window of size B to the right by as much as one byte, we can renew the shift values for "CD" and "CF" into one to zero, and for "DE" and "FF" into two to zero. Finally, we are able to generate the bad-character shift table that is shown in **Table. 1**.

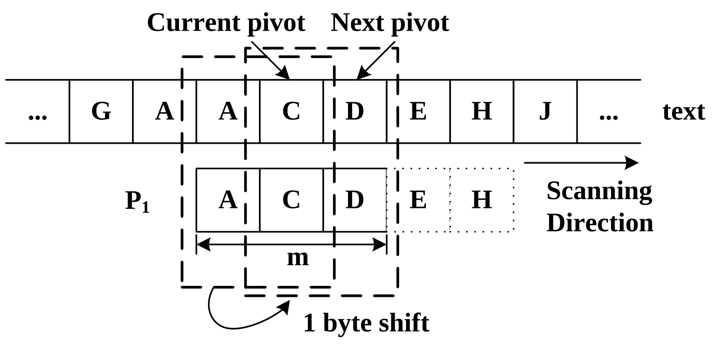

Fig. 5. Heuristic Matching with Two-byte Suffix Matching

3.2 Scanning Procedure

Using the given shift table, we scan the text while skipping the mismatching suffix. Also, if the matching suffix is found in the text, we examine the other characters in the pattern candidates. The details of the scanning operation for scanning the text by using the bad-character shift table are shown in **Fig. 4**.

For example, as shown in **Fig. 5**, "AC" in the text is found at P_1 and then, has the shift value one in the bad-character shift table. Thus, by skipping one-bad character, the *current pivot* located at 'C' in the text moves by as much as one byte to the *next pivot* position located at 'D' in the text. At the *next pivot* position, "CD" with the shift value zero is found from P_1, whose prefix('A') and the remaining characters("EH") also exist in the text. Thus, the algorithms can find the matched pattern P_1 in the text.

3.4 Complexity Analysis

The results of the complexity analysis of each procedure can be derived from [13, 14]. Given the text of length n and N numbers of patterns, the bad-character shift table is constructed in time $O(mN)$, because any substrings of length B are only considered once and this process requires a constant time on average to consider them. When scanning the text by using the bad-character shift table, the shift value of the B-byte substring at the current pivot can be *non-zero* or *zero*. If the shift value at the current pivot is *non-zero*, the heuristic clearly moves the current pivot to the right in the text by as much as the shift value. Since the maximum shift value of a heuristic of multi-byte suffix matching is m-1 for B=2, the total amount of time in the case of non-zero shifts by a heuristic with multi-byte suffix matching is $O(n/(m-1))$. On the other hand, when the shift value is zero, more characters to the left in the text need to be tested. That is, when we need to check the whole string of length m, it require time $O(m)$. It is worth noting that at most N strings lead to a shift value of i for $0 \le i \le m-B-1$ and the number of all possible strings of length B is at least $2mN$. Thus, the probability of one random string that leads to a shift value of i is $\le 1/(2m)$. Since the probability of a shift value of zero is also $\le 1/(2m)$, the expected total amount of time required for the case of zero shifts by a heuristic with multi-byte suffix matching is $O(n/m)$ without the help of any other heuristics such as the hash table [14] for filtering further bad-characters.

4. Probabilistic Model for A Heuristic With Multi-Byte Suffix Matching

In this section, we will describe the proposed mathematical model of a heuristic with *multi*-byte suffix matching. As the performance of the heuristic varies depending on the nature of the input texts and patterns, we examined the probabilistic performance in an average-case search. For this analysis, we assumed uniform distributions of characters in both the text and all the patterns, i.e., *Pr*[Any given character]=1/*a*. Following the common practice of the previous work [11], given the length of the text *n*, the performance can be measured in terms of the number of comparison that needs to be examined per character in the text:

$$PM = nE[t]. \tag{1}$$

Thus, we can measure the performance of a heuristic with *multi*-byte suffix matching into the number of comparison that needs to be examined per character of shift in the text:

$$PM = nE[t]/E[s]. \tag{2}$$

Since it is known that for *B*=2, the heuristic shows the best performance in the aspects of memory lookups [13], the analysis model for *B*=2 will be described in the following subsections.

4.1 Expected Number of Comparisons: E(t)

To find the matching suffix length *t*, the algorithm needs to examine 2*t* characters, since every character except for the first character and the (t+1)*th* character was examined twice. Given a string of length *m*, to compute the matching suffix length *t*, we considered the probability that *t* equals *x*, where $0 \le x \le m$. The probability that *t* equals zero was $1-T/a^2$ since the probability of at least consecutive two characters in the text matching a string of length *m* was T/a^2. Thus, the probability that *t* equals zero can be described as follows:

$$\Pr[t=0] = 1 - T/a^2. \tag{3}$$

Also, since the probability that *t* was less than *2* is zero since consecutive two characters were used for comparison, the probability can be given as follows:

$$\Pr[0 < t \le 1] = 0. \tag{4}$$

In general, the probability was given into $\Pr[0 < t \le B-1] = 0$.

The probability that *t* equals one was zero since consecutive two characters were used for comparison. That is, $\Pr[t=1]=0$.

To compute the probability that *t* equals *x*($2 \le x \le m$), we considered the number of consecutive *x*-byte blocks generated from *N* patterns. Given a string of length *m*, '$s_1, s_2, \ldots, s_{m-1}, s_m$', *m*-*x*+1 consecutive *x*-byte blocks were generated: '$s_1, s_2, \ldots, s_{x-1}, s_x$' to '$s_{m-x+1}, s_{m-x+2}, \ldots, s_{m-1}, s_m$'. Given *N* patterns, if *x* increases by as much as one, the number of consecutive *x*-byte blocks decreased by as much as *N*. Thus, the number of consecutive *x*-byte blocks equals *T*-*N*(*x*-2). Based on the fact that *Pr*[*t*≥*x*] was the probability of at least consecutive *x* characters in the text matching a string of length *m*, the following relationship can be derived:

$$\Pr[t \ge x] = \frac{T - N(x-2)}{a^x}, \tag{5}$$

and the probability that matching suffix length *t* equals *x*($2 \le x \le m-1$) was as follows: $\Pr[t=x] = \Pr[t \ge x] - \Pr[t \ge x+1]$. That is,

$$\Pr[t=x] = \frac{(a-1)(T-Nx) + N(2a-1)}{a^{x+1}}. \tag{6}$$

Also, since the probability that matching suffix length *t* equals *m* is, the following relationship can be derived:

$$\Pr[t=m]=\frac{T-N(m-2)}{a^m}, \tag{7}$$

the probability of matching suffix length *t* equals *x* (0≤*x*≤*m*) was as follows:

$$\Pr[t=x]=\begin{cases}1-T/a^2 & ,\text{x=0;}\\ 0 & ,\text{x=1;}\\ \dfrac{(a-1)(T-Nx)+n(2a-1)}{a^{x+1}} & ,2\le \text{x} \le \text{m-1;}\\ \dfrac{T-N(x-2)}{a^x} & ,\text{x=m.}\end{cases} \tag{8}$$

Thus, the expected value of *t* was computed from probabilities of each value of *t* from 0 to *m*:

$$\begin{aligned}E[t]&=\sum_{x=0}^{m} x\times \Pr[t=x]\\ &=\sum_{x=2}^{m-1} x\frac{(a-1)(T-Nx)+N(2a-1)}{a^{x+1}}+m\frac{T-N(m-2)}{a^m}.\end{aligned} \tag{9}$$

4.2 Expected Shift Value: E(s)

The probability that each of the last *s* two-byte blocks in the text mismatching a pattern was $(1-T/a^2)^s$ and the probability of each two-byte block in the text matching a pattern was T/a^2. Thus, the probability of the shift value *s*(2≤*s*≤ *m*-2) can be explained as follows:

$$(T/a^2)\left(1-T/a^2\right)^s. \tag{10}$$

In general, for the probability of the shift value *s*(*B*≤*s*≤ *m*-*B*), Eq.(10) can be written into $(T/a^B)\left(1-T/a^B\right)^s$.

Also, since the probability that the shift value *s* equals to *m*-1 was the only remaining case and occurs when all (*m*-1) two-byte blocks mismatch (: $\left(1-T/a^2\right)^{m-1}$), the probability of the shift value *s* was determined to be:

$$\Pr[s=l]=\begin{cases}0 & ,l\le 1;\\ (T/a^2)\left(1-T/a^2\right)^l & ,2\le l\le m-2;\\ \left(1-T/a^2\right)^l & ,l=m-1.\end{cases} \tag{11}$$

The expected shift value was computed from probabilities of each shift value from 0 to *m-1*:

$$\begin{aligned} E[s] &= \sum_{l=0}^{m-1} l \times \Pr[s=l] \\ &= \sum_{l=1}^{m-2} l \frac{T(a^2-T)^l}{a^{2(l+1)}} + (m-1)(1-\frac{T}{a^2})^{(m-1)}. \end{aligned} \tag{12}$$

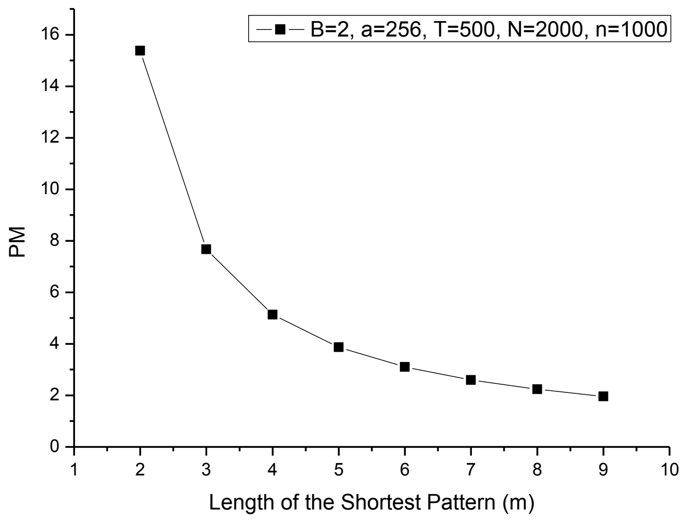

Fig. 6. Theoretical Analysis Results: Average Number of Comparison per Character of Shift in the text for various *m*

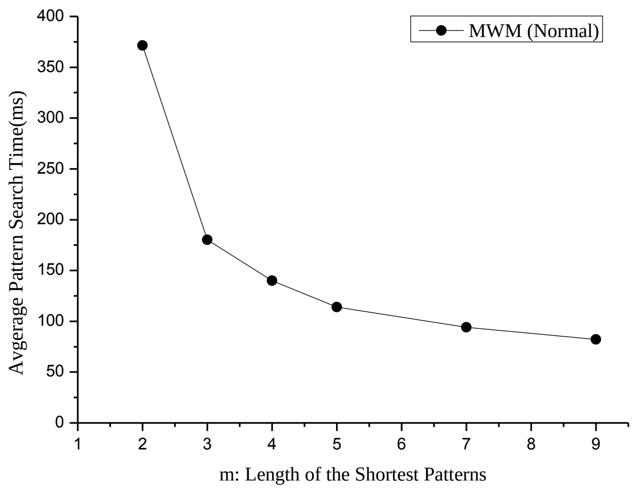

Fig. 7. Experimental Analysis Results: Average Pattern Search Time for various *m*

5. Evaluation

Based on the comparison between the theoretical and experimental analysis results, we show that the proposed model was sufficient to evaluate the performance characteristics of a heuristic with suffix matching in the skip-based pattern matching algorithms.

5.1 Experimental Environments

We conducted off-line experiments to compare the performance of the proposed

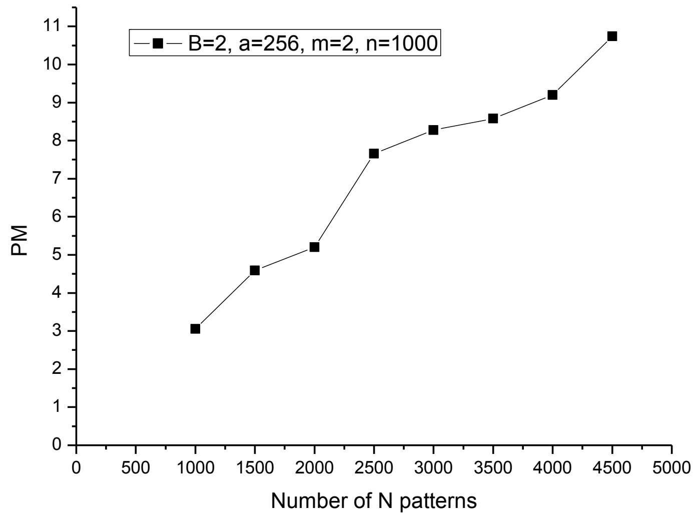

Fig. 8. Theoretical Analysis Results: Average Number of Comparison per Character of Shift in the text for various *N*

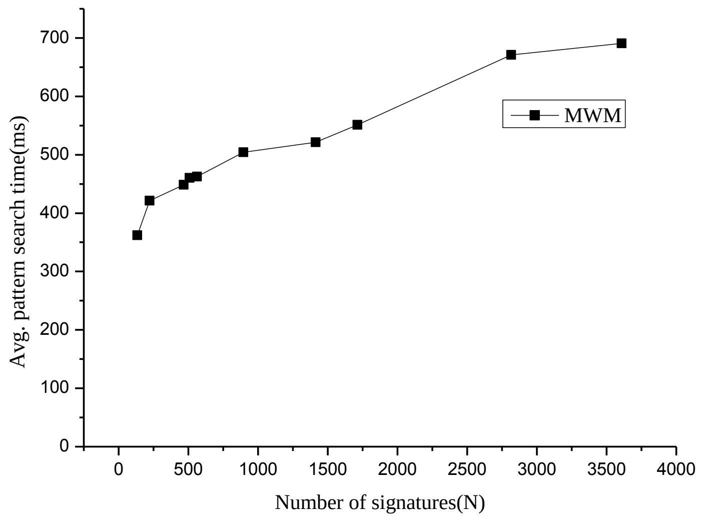

Fig. 9. Experimental Analysis Results: Average Pattern Search Time for various *N*

probabilistic model with that of a heuristic with multi-byte suffix matching on the real system, called the snort [18]. By using 100 synthetic patterns, we measured the search time of the MWM algorithm [13] for different values of *m* and a normal data set. Also, we measured the search time at the different values of *N* for a normal data set. It is worth noting that the performance of the skip-based pattern matching algorithms depended on many conditions such as the characteristics of traffic, patterns, system performance and so on. Since the proposed model does not measure the performance of the pattern search time of the skip-based pattern matching algorithms, but rather the effect of a heuristic with multi-byte suffix matching on the performance of the skip-based pattern matching algorithms, we measured

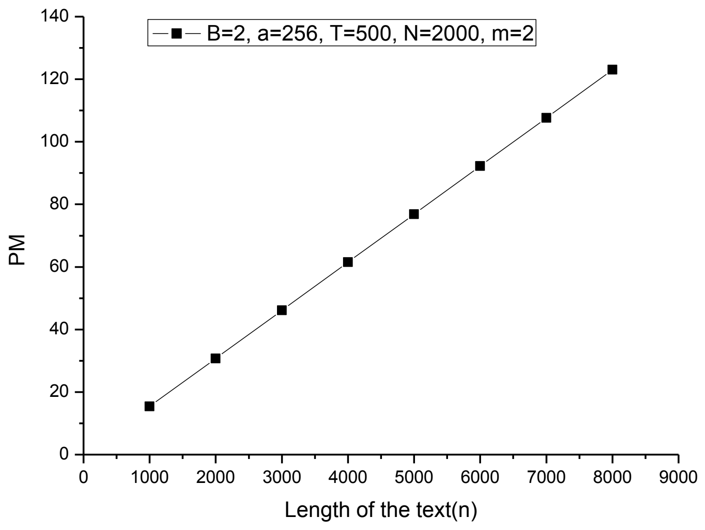

Fig. 10. Theoretical Analysis Results: Average Number of Comparison per Character of Shift in the text for various *n*

only the search time of the MWM algorithm that is a representative algorithm using a heuristic with multi-byte suffix matching. For the default ruleset of the snort, the normal data set generated 259 alerts out of 118,450 (TCP: 115,418, UDP: 2,389, ICMP: 84, OTHER: 559) packets (103,937KB). The search time was measured using a Linux machine whose CPU clock was 2.4GHz(L2 cache of 512KBytes), size of main memory is 512MBytes and kernel version was 2.6.6-8hl.

5.2 Effect of the Length of the Shortest Pattern

Since the performance of a heuristic with suffix matching mainly depends on *m* [13], we examined the proposed mathematical model by varying the values of *m*.

As shown in theoretical analysis results in **Fig. 6**, the performance of a heuristic with suffix matching exponentially decreased as the length of the shortest pattern increased. Specifically, for *m* less than five, the average number of comparison per character of shift in the text was relatively large compared to that for more than five. This indicates that the proposed model can correctly examine the performance, which mainly depended on the patterns in a rule set whose lengths were less than five [13,14].

In the experimental analysis, shown in **Fig. 7**, the average pattern search time exponentially decreased as the length of the shortest pattern increases. From a comparison between **Figs. 6 and 7**, we observe that the proposed mathematical model is sufficient to show the performance characteristics of a heuristic with multi-byte suffix matching in an average-case search. That is, from the theoretical and experimental analysis results, it was clearly demonstrated that when the length of the shortest patterns was less than five, the performance of the heuristic rapidly decreased. It was also found that when the length of the shortest patterns was larger than four, the performance slowly decreased.

5.3 Influence of the Number of Signatures

Since the performance of a heuristic with multi-byte suffix matching mainly depended on *N*, we examined the proposed mathematical model by varying the values of *m*. Specifically, since the value of *T* depends on *N*, we assumed that as *N* was increased from 1000 to 4500 at the increments of 500, *T* had the values of 100, 150, 170, 250, 270, 280, 300, 350, respectively.

Based on the theoretical analysis results shown in **Fig. 8** and experimental analysis results shown in **Fig. 9**, we found that the performance of a heuristic with suffix matching almost linearly increased as the number of patterns increased. In addition, the the performance largely increased between N=2000 and N=2500 when compared to the other intervals. This occured because *T* largely increased in the interval between N=2000 and N=2500. The difference between **Figs. 8 and 9** comes from the fact that the distribution of characters in the real patterns was not perfectly uniform. These results indicate that the proposed model can accurately estimate the performance characteristics of a heuristic with suffix matching in the skip-based pattern matching algorithms.

5.4 Influence of the Length of the Text

The effect of the length of the text(*n*) on the performance of a heuristic with suffix matching by the influence is shown in **Fig.10**. The number of comparison per character of shift was shown to linearly increase in proportional to *n*, which has been demonstrated previously.

6. Conclusions

In this paper, we propose a new probabilistic model for evaluating the performance of a heuristic with multi-byte suffix matching in an average-case search. Based on a comparison between the theoretical analysis results and experimental results, the proposed probabilistic model was shown to be useful for estimating the performance of the real pattern matching algorithm based on a heuristic with *multi*-byte suffix matching.

Acknowledgement

This work was supported by Kyonggi University Research Grant 2012-005.

References

[1] A. Aho and M. Corasick, "Efficient string matching: An aid to bibliographic search," *Communications of the ACM*, vol. 18, no. 6, pp. 333-343, 1975. Article (CrossRef Link).

[2] Nathan Tuck, Timothy Sherwood, Brad Calder, and George Varghese, "Deterministic Memory-Efficient String Matching Algorithms for Intrusion Detection," in *Proc. of the 23rd Conference of the IEEE Communications Society(IEEE INFOCOM 2004)*, 2004. Article (CrossRef Link).

[3] R. Smith, C. Estan and S. Jha, "XFA: Faster signature matching with extended automata," *IEEE Symposium on Security and Privacy (Oakland)*, May 2008. Article (CrossRef Link).
[4] N. Hua, H. Song and T.V. Lakshman, "Variable-Stride Multi-Pattern Matching For Scalable Deep Packet Inspection," in *Proc. of the 28th Conference on Computer Communications(INFOCOM 2009),* Apr. 2009. Article (CrossRef Link).
[5] R.S. Boyer and J.S. Moore, "A Fast String Searching Algorithm," *Communications of the ACM*, vol. 20(10), pp. 762-772, 1977. Article (CrossRef Link).
[6] A. Apostolico, R. Giancarlo, "The Boyer–Moore–Galil string searching strategies revisited," *SIAM Journal on Computing*, vol. 15, no. 1, pp. 98–105, 1986. Article (CrossRef Link).
[7] B. Commentz-Walter, "A string matching algorithm fast on the average," *in Proc. of the 6th International Colloquium on Automata, Languages, and Programming*, pp. 118–132, 1979. Article (CrossRef Link).
[8] B. Xu, X. Zhou, J. Li, "Recursive shift indexing: a fast multi-pattern string matching algorithm," *in Proc. of the 4th International Conference on Applied Cryptography and Network Security (ACNS)*, 2006. Article (CrossRef Link).
[9] Rong-Tai Liu, Nen-Fu Huang, Chih-Hao Chen, Chia-Nan Kao, "A fast string-matching algorithm for network processor-based intrusion detection system," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, no. 3 , pp. 614–633, 2004. Article (CrossRef Link).
[10] R. Horspool, "Practical fast searching in strings. Software Practice and Experience," vol. 10, no. 6, pp. 501-506, 1980. Article (CrossRef Link).
[11] M. Fish and G. Varghese, "Fast Content-Based Packet Handling for Intrusion Detection," *UCSD TR CS2001-0670*, 2001. Article (CrossRef Link).
[12] M.Fish and G.Varghese, "An analysis of fast string matching applied to content-based forwarding and intrusion detection," *UCSD technical report CS2001-0670*, 2002. Article (CrossRef Link).
[13] Giorgos Vasiliadis, Michalis Polychronakis, Spiros Antonatos, Evangelos P. Markatos, Sotiris Ioannidis, "Regular expression matching on graphics hardware for intrusion detection," in *Proc. of the 12th International Symposium On Recent Advances In Intrusion Detection (RAID)*, 2009. Article (CrossRef Link).
[14] Y.-H. Choi, M.-Y. Jung and S.-W. Seo, "A fast pattern matching algorithm with multi-byte search unit for high-speed network security," *Elsevier Computer Communications(ComCom)*, vol. 34, no. 14, pp. 1750-1763, Sep. 2011. Article (CrossRef Link).
[15] Wu, S. and Manber, U, "A fast algorithm for multi-pattern searching," *Department of Computer Science, University of Arizona. TR94-17*, 1994. Article (CrossRef Link).
[16] S. Antonatos, K.G. Anagnostakis, E.P. Markatos, and M. Polychronakis, "Performance Analysis of Content Matching Intrusion Detection Systems," in *Proc. of the IEEE/IPSJ Symposium on Applications and the Internet(SAINT 2004)*, pp. 26-30, Jan. 2004. Article (CrossRef Link).
[17] P.-C. Lin, Z.-X. Li, Y.-D. Lin, Y.-C. Lai and F.-C. Lin, "Profiling and accelerating string matching algorithms in three network content security applications," *Communications Surveys & Tutorials IEEE*, Volume: 8, Issue: 2, Page(s): 24–37, Feb., 2007. Article (CrossRef Link).
[18] Sourcefire, Inc., "Snort^{TM} Users Manual 2.8.4," *The Snort Project*, Apr. 2009. Article (CrossRef Link).

Yoon-Ho Choi is an assistant professor at department of convergence security in Kyonggi University, Suwon, Korea. He received the M.S. and Ph.D. degrees from school of electrical and computer engineering, Seoul National University, S. Korea, in Aug. 2004 and Aug. 2008, respectively, and the B.S. degree from school of electronics and electrical engineering, Kyungpook National University, S. Korea, in Aug. 2002. He was a postdoctoral scholar in Seoul National University from Sep. 2008 to Dec. 2008 and in Pennsylvania State University, University Park, PA, USA, from Jan. 2009 to Dec. 2009. He has served as TPC members in various international conferences and journals. His research interests include Deep Packet Inspection(DPI) for high-speed intrusion prevention, mobile computing security, vehicular network security for realizing secure computer and networks.