\(L^+\text{-MWM}: \) A Fast Pattern Matching Algorithm for High-Speed Packet Filtering

Yoon-Ho Choi, Student Member, IEEE, Moon-Young Jung and Seung-Woo Seo, Member, IEEE
School of Electrical Engineering and Computer Sciences, Seoul National University, Seoul, Korea 151-744

Abstract—A signature-based network intrusion detection system (NIDS) identifies intrusions by comparing the data traffic with known signature patterns. In this process, matching of packet strings against signature patterns dominates the overall system performance. The MWM algorithm has been known as the fastest pattern matching algorithm when the patterns in a rule set rarely appear in packets. However, the matching time does not decrease if the length of the shortest pattern in a signature group is too short. In this paper, by extending the length of the shortest pattern, we minimize the pattern matching time of the algorithm which uses multi-byte unit. For example, when the length of the shortest pattern is less than 5, the proposed algorithm shows 38.87% enhancement in average.

I. INTRODUCTION

As computer and communication networks have commonly been accessible, a variety of security threats from internal and external sources have become prevalent. Those threats in networks may include many types of attacks such as denial of service, viruses and worms, unauthorized user access and so on. To take some appropriate measures against the attacks, currently available security solutions heavily rely on the intrusion detection capability of the system. Up to now, common approaches for the intrusion detection are based on either statistical anomaly detection or signature-based intrusion detection.

The matching of packet strings against collected signatures dominates the performance of the signature-based Network Intrusion Detection System (NIDS). For example, it is known that the string matching time accounts for 40% to 70% of the Snort running time [1]. The signature-based NIDS, that utilizes a fast pattern-matching technique, inspects all network traffic that crosses its network segment from internal and external sources to look for the known pattern, where a pattern means a specific content in a signature. Thus, a signature-based NIDS identifies possible attempts that may lead to break-ins into the network. For example, in one of the well-known open source based NIDS, i.e. the Snort [2], an alert is created if packets have a pattern contained in the rule sets after the normalization stages (such as IP defragmentation and TCP stream reassembly). Once a packet header rule is matched, the Snort finds the string within the packet payload by usually relying on one or multiple pattern matching algorithms such as in [3]-[7].

A. Overview of the pattern matching algorithms

The pattern matching algorithms used in a signature-based NIDS can be classified into a single pattern matching algorithm and a multi-pattern matching algorithm. Given a pattern, a single pattern matching algorithm finds all occurrences of the pattern within a text which is the content of each packet. On the other hand, a multi-pattern matching algorithm finds all occurrences of the given set of patterns within a text.

The most well-known single pattern matching algorithm is the Boyer-Moore algorithm [3]. It compares the characters of the pattern with those of the text from right to left by using two heuristics called as the bad-character shift (also called as the occurrence shift) and the good-suffix shift (also called as matching shift). Thanks to those heuristics, the Boyer-Moore algorithm runs in sub-linear order time and is much faster than any other single pattern matching algorithms such as the Knuth-Morris-Pratt algorithm [4].

However, the Boyer-Moore pattern matching algorithm has a problem that with thousands of different patterns, there is the possibility that most characters in the text match the last character of some patterns [3]. Therefore, if most characters in the text match the last character of some patterns in practice, there would be few shifts in the text for the patterns [5]. The Modified Wu-Manber (MWM) algorithm is designed in order to overcome this problem while keeping the speed of the Boyer-Moore algorithm as described in the next section. It is noted that there also have been a few other algorithms proposed for multi-pattern detection. The Setwise Boyer-Moore-Horspool algorithm (Setwise BMH) [6] constructs a trie that stores all patterns based on their suffixes after reversing all patterns. By using the trie, the algorithm compares the patterns with the text from right to left in the same way as the Boyer-Moore algorithm. However, except for the MWM algorithm, many algorithms for multi-pattern detection are based on a single-byte unit for search, i.e., compare the characters of the text with those of the patterns by one-byte unit. This implies that those algorithms still have the same problem as the Boyer-Moore algorithm.

B. Characteristics of the MWM algorithm

The MWM algorithm is known to be the fastest pattern matching algorithm and is also the default engine of the Snort when the search-set size exceeds 10 [12]. The authors in [2] show that when the Snort uses the MWM algorithm, the matching speed in a typical search becomes much faster than when using the Aho-Corasick (AC) [9] and other Boyer-Moore like algorithms [4]-[8].

The MWM algorithm uses the Boyer-Moore algorithm with a \(B(\geq2)\)-byte SHIFT table established by preprocessing.
all patterns, where the SHIFT table is used to determine how many characters in the text can be skipped when the text is scanned [5]. This search unit contributes to solving the problem mentioned above in other known multi-pattern and single pattern matching algorithms that use $B = 1$. However, the MWM algorithm still has a problem, i.e., in a signature group, if there are many patterns whose lengths are too short, it rarely enhances the speed of the Boyer-Moore algorithm. This is because the shift values of the SHIFT table are given dependent on the length of the shortest pattern, i.e. LSP, in a signature group, which is an important bottleneck in many pattern matching algorithms. For example, the shift value of each index of a SHIFT table is set to $\min(LSP - B + 1, LSP - B - q)$, where if $a \geq b$, $\min(a, b) = b$ otherwise, $\min(a, b) = a$. Moreover, LSP is known to dominate the matching time of the MWM algorithm in an average search case rather than in the worst case behavior [12].

C. Contribution and Organization of the paper

In this paper, we propose a new pattern matching algorithm called the $L^{+1}$-MWM algorithm. In practice, the $L^{+1}$-MWM algorithm minimizes the performance degradation due to the length of the shortest pattern in a signature group. By extending the length of the shortest pattern, we minimize the pattern matching time of the algorithm which uses the bad character shift table and $B(\geq2)$-byte unit for search. Also, we show that the optimal conditions that maximize the average shift value based on the $B$-byte unit for search.

The rest of the paper is organized as follows. In section II, we describe the main idea of the proposed algorithm by analyzing the rule sets of a real application system and the influence of LSP on the performance of the pattern matching algorithms. In section III, we describe the operation of the $L^{+1}$-MWM algorithm including how to construct the data structure. And then, we propose the optimal conditions that maximize the average shift value in the SHIFT table in section IV. Next, in section V, we evaluate its expected performance and show the experiment results. Finally, this paper is concluded in section VI.

II. EXTENSION OF THE LENGTH OF THE SHORTEST PATTERN

The following observations motivate the proposal of the new algorithm.

First, the analysis result of the Snort v2.0 rule sets shows that LSP of 32 rule sets among 45 rule sets which have at least one pattern is less than 5. More specifically, there are 11 rule sets of LSP 2, 5 of LSP 3 and 16 of LSP 4. This implies that the maximum shift value in the SHIFT table is mainly less than 3 when $B$ is 2, which is known as the optimal value of the MWM algorithm to get the best performance [5]. Additionally, this indicates that real matching time in the Snort depends on the patterns in a rule set whose lengths are less than 5, i.e., $2 \leq LSP \leq 4$

Second, by extending the length of the shortest patterns in a signature group, we investigate its influence on the pattern matching algorithms which use the bad character shift table. Let us consider the pattern matching algorithms which use one-byte unit for search such as the Boyer-Moore like algorithms. Assume that $m$ arbitrary imaginary alphabets are padded to the leftmost side of the shortest pattern. The shift value of the SHIFT table may be given as $\min(LSP + m, LSP + m - 1 - k_1)$ in stead of $\min(LSP, LSP - 1 - k_1')$, where using one-byte unit for search, $k_1'$ means the start position of the sliding window for strings of size $LSP$ given from the original patterns($0 \leq k_1' \leq LSP - 1$) and $k_1$ for strings of size $LSP + m$ given from the patterns where $m$ arbitrary imaginary alphabets are padded($0 \leq k_1 \leq LSP + m - 1$). However, the padded alphabets are not the real which are originated from the original patterns so that it can be given as any of all indices of the SHIFT table. Thus, the shift value of the SHIFT table is given as the same value as before we pad $m$ arbitrary imaginary alphabets, i.e., not $\min(LSP + m, LSP + m - 1 - k_1)$ but $\min(LSP, LSP - 1 - k_1')$.

Next, we consider the pattern matching algorithms which use $B(\geq2)$-byte unit for search such as the MWM algorithm. To the leftmost side of the shortest pattern, if we pad $m$ arbitrary imaginary alphabets less than $B$, at least one alphabet which composes substrings of size $B$ is originated from the original pattern. That is, the substrings of size $B$ which consist of the indices of the SHIFT table are composed of one or some padded alphabets and ones of the original pattern. It is noted that the $L^{+1}$-MWM algorithm shows good performance when the first(leftmost) character of patterns of size $LSP$ is not diverse, which means that specific characters frequently appear at the first position of patterns. That is because if many different characters are located at the first position of patterns of size $LSP$, we should set the initial shift value of each block as $LSP - B + 1$ to avoid the invalid shift in the text. In this case, we cannot increase the values of the SHIFT table while keeping the size of the SHIFT table the same as in the MWM algorithm. In practice from the analysis result of the Snort v2.0 rule sets, we see that the number of different characters located at the first character of the shortest pattern is less than 10. Thus, the shift value of most indices in the SHIFT table is given as $\min(LSP + m - B + 1, LSP + m - B - k_B)$ in stead of $\min(LSP - B + 1, LSP - B - k_B')$, where using $B(\geq2)$-byte unit for search, $k_B'$ means the start position of the sliding window for strings of size $LSP$ given from the original patterns($0 \leq k_B' \leq LSP - B$) and $k_B$ for strings of size $LSP + m$ given from the patterns where $m$ arbitrary imaginary alphabets are padded($0 \leq k_B \leq LSP + m - B$). Therefore, the average shift value of the SHIFT table increases and this analysis result motivates us to extend the length of the shortest pattern.

III. OPERATION OF THE $L^{+1}$-MWM ALGORITHM BASED ON $LSP$ EXTENSION

Based on the above analysis results, we explain the operation of the $L^{+1}$-MWM algorithm. The $L^{+1}$-MWM algorithm consists of the preprocessing stage and the scanning stage.
A. The preprocessing stage

In this stage, we construct the SHIFT, HASH and PREFIX tables. First, we compute \( LSP \) from each signature group and set \( mLSP = LSP + 1 \). Second, after padding one arbitrary imaginary alphabet to the leftmost side of the patterns of a size of \( LSP \), we consider the strings of a size of \( mLSP \) for the patterns in each pattern group. Note that the rightmost occurrence in all possible patterns of a substring of size \( B \) in the \( text \) determines the amount of shifting required to get to this substring in the \( text \). When we pad one arbitrary imaginary alphabet to the leftmost side of the patterns of a size of \( LSP \), the positions that a substring of size \( B \) appears in the patterns of size \( LSP \) are the same as when we consider the strings of size \( LSP \). Also, as we set the corresponding value for the imaginary substrings of size \( B \) in front of the patterns of size \( LSP \) as the minimum shift value (the initial shift value is given as \( LSP - B + 1 \)), we can avoid the invalid shifting. In case of the patterns whose size is greater than \( LSP \), strings of size \( mLSP \) are also the prefixes of the original patterns. Thus, although the rightmost occurrence of a substring of size \( B \) in the patterns changes, the new values of the SHIFT table become valid. Also, to increase the average value of the SHIFT table while keeping the size of the SHIFT table the same as in the MWM algorithm, we pad only one arbitrary imaginary alphabet when \( B \) is 2. An alphabet is an ASCII code which is used to construct patterns. Third, we construct the SHIFT table by moving the window of substring size of \( B \) from position \( mLSP - B \) to position 0 in a given string of size \( mLSP \).

The following code shows how to determine the value of indices in the SHIFT table.

1. **ALGORITHM III.1:**

   ```plaintext
   PreBadWordShiftTable(MWMSTRUCT)
   //m is 1 and B is 2 in the L^+1-MWM algorithm
   Input mLSP=LSP+1;

   //Initialize the default shift table
   FOR i=0 to SHIFTTABLESIZE-1 DO
     InitialShiftLenOfIndex ← mLSP-B+1;
   ENDFOR

   //Obtain Multi-Pattern B-byte Shift Table values
   FOR i=0 to NumberOfPatterns-1 DO
     IF LenOfPattern_i > LSP THEN
       FOR k=0 to InitialShiftLenOfIndex-1 DO
         shift ← mLSP-B-k;
       IF shift > 255 THEN shift ← 255;
       ENDIF
       index ← SubStringOf-SizeBStartingFromPosition_k;
       IF shift < ShiftValueOfIndex THEN
         ShiftValueOfIndex ← shift;
       ENDIF
       ENDFOR
     ELSE IF LenOfPattern_i == LSP THEN
       FOR j=0 to 255 DO
         shift ← (LSP-B+1);
         PadASCII ← j;
         index ← SubStringOf-SizeBWithPadASCIIAtPosition_k;
         IF shift < ShiftValueOfIndex THEN
           ShiftValueOfIndex ← shift;
         ENDIF
       ENDFOR
     ENDIF
   ENDFOR
   ```

B. The scanning stage

Let \( X=x_0 \ldots x_{B-1} \) be the characters in the \( text \).

- Case 1: \( X \) does not appear as a substring in \( P \), where \( P \) means a set of patterns \( \{p_i\} \) in a signature group consisting of an alphabet, \( a_j \).
- Case 2: \( X \) appears as a substring in \( P \).

For case 1, we can clearly shift \( mLSP - B + 1 \) characters in the \( text \), where \( mLSP - B + 1 \) is the initial shift value in the SHIFT table. For case 2, we find the rightmost occurrence of \( X \) in any of the patterns. If \( X \) appears at position \( k_{BR} \) of a pattern and \( k_{BR} \) is the largest start position in all possible patterns, we shift the \( text \) by \( mLSP - B - k_{BR} \) characters. And when the shift value is 0, we perform an approximate matching using the HASH and PREFIX tables. That is, using the HASH table, we find the pattern candidates whose last \( B \) characters in \( mLSP \) characters are the same as \( X \) in the \( text \) and using the PREFIX table, find the pattern candidates whose last \( B \) characters in \( mLSP \) characters are the same as \( X \) in the \( text \) and whose first \( B_p \) characters are the same as \( B_p \) characters found in the \( text \) by shifting \( mLSP - B \) or \( mLSP - B + 1 \) characters to the left. If \( B_p \) characters in the \( text \) are found in the PREFIX table by shifting \( mLSP - B \), this indicates that the pattern candidates whose lengths are greater than \( LSP \) exist and if \( B_p \) characters in the \( text \) are found by shifting \( LSP - B \), this indicates that the pattern candidates whose lengths equal \( LSP \) exist. Also, if \( B_p \) characters in the \( text \) are found in the PREFIX table by shifting \( mLSP - B \) and by shifting \( LSP - B \), this indicates that the pattern candidates whose lengths are greater than \( LSP \) and the pattern candidates whose lengths equal to \( LSP \) exist. We then execute exact
matching for the given pattern candidates. When we do exact matching, we check the pattern candidates against the text directly.

IV. VALIDATION OF THE $L^+1$-MWM ALGORITHM

Now, we prove the validity of the $L^+1$-MWM algorithm by showing the optimal conditions that maximize the average shift value of the SHIFT table. At first, we examine the influence of the standard unit for search, $B$.

Lemma 1: If we increase the value of $B$, the size of the SHIFT table increases exponentially and the shift value of the SHIFT table decreases.

Proof: Based on the value of $B$, we construct the SHIFT table with $2^{8+B}$ indices. Thus, the size of the SHIFT table exponentially increases dependent on the value of $B$. And the initial shift value of the SHIFT table is given as $LSP - B + 1$ and the shift value of each index is given as $min(LSP - B + 1, LSP - B - k_B')$. Here, $min(LSP - B + 1, LSP - B - k'_B)$ equals $LSP - B - k'_B$ and $0 \leq LSP - B - k'_B \leq LSP - B$ because $0 \leq k'_B \leq LSP - B$. Here, $k'_B$ is given from the patterns. Therefore, the shift value of the SHIFT table can decrease if the value of $B$ increases. Also, when $B$ is 2 or 3, the MWM algorithm gets the best performance as mentioned in [5]. From these, we know that $B = 2$ is the optimal value to get the best performance while keeping the size of the SHIFT table small.

We next investigate the influence of the $LSP$ for the shift value of the SHIFT table.

Lemma 2: The shift value of each index in the SHIFT table is less than $LSP + 2$.

Proof: Let’s assume that $LSP$ characters in all patterns are chosen. The initial shift value of the SHIFT table is given as $LSP - B + 1$ and the value of each index is given as $min(LSP - B + 1, LSP - B - k'_B)$. Given $B$, $LSP$ dominates the value of each index in the SHIFT table and cannot be greater than $LSP + 1$.

Finally, when we pad $m$ arbitrary imaginary alphabets in front of the patterns of a size of $LSP$, we investigate the optimal conditions that maximize the average shift value of the SHIFT table. Given $B$, we find the optimal value of $m$.

Lemma 3: The average shift value of the SHIFT table can be maximized when $B > m$ and $m$ equals 1.

Proof: Let us assume a string $b_0\ldots b_m-1 a_0\ldots a_{LSP-1}$ which is considered by padding an arbitrary imaginary string $b_0\ldots b_m-1$ to the leftmost side of a pattern $a_0\ldots a_{LSP-1}$, where $a_j$ and $b_j$ are the alphabets. If $B \leq m$, a substring of size $B$ in $b_0\ldots b_m-1$ can be given as the combination of any $B$ alphabets. However, the possible $2^{8+B}$ substrings do not exist in the rightmost side of the pattern $a_0\ldots a_{LSP-1}$. Thus, we should set its initial shift value in the SHIFT table to $LSP - B + 1$ and its shift value is given as $min(LSP - B + 1, LSP + m - B - k_B)$ when the substring exists in the pattern whose size is greater than $LSP$ or $min(LSP - B + 1, LSP - B - k'_B)$ when the substring exists in the pattern of size $LSP$.

If $B > m$, as the possible substrings to the leftmost side of a string $b_0\ldots b_m-1 a_0\ldots a_{LSP-1}$ includes at least one character of the pattern $a_0\ldots a_{LSP-1}$, their number can be limited to $2^{8+m}$. Thus, we can set the initial shift value of $2^{8+m}$ indices of the SHIFT table as $LSP - B + 1$ and the initial shift value of $2^{8+B} - 2^{8+m}$ indices of the SHIFT table as $LSP + m - B + 1$. And, the shift values of $2^{8+B} - 2^{8+m}$ indices are given as $min(LSP + m + B - 1, LSP + m - B - k_B)$ when the substring exists in the pattern whose size is greater than $LSP$ or $min(LSP + m - B + 1, LSP - B - k'_B)$ when the substring exists in the pattern of size $LSP$. This implies that the average shift value can increase almost as large as $\frac{m(2^{8+B} - 2^{8+m})}{2^{8+B}}$. From these and Lemma 1, we know that when $m$ equals 1, the average shift value of the SHIFT table can be maximized.

Consequently, we can prove the validity of the $L^+1$-MWM algorithm as in the following theorem.

Theorem 1: (Validity of the $L^+1$-MWM algorithm) The shift value of the SHIFT table mainly depends on the following three values: $B$, $LSP$ and $m$. When $B$ is 2 and $m$ equals 1, the average shift value of the SHIFT table is maximized under the influence of $LSP$. Thus, the $L^+1$-MWM algorithm has the optimal SHIFT table, which improves the pattern matching time of the MWM algorithm for a typical search.

Proof: From the above lemmas, it is clear that when $B$ is 2 and $m$ equals 1, the average shift value of the SHIFT table is maximized under the influence of $LSP$. Thus, we scan the text by using the optimal SHIFT table and the pattern matching time for a typical search can be reduced.

V. PERFORMANCE EVALUATION

A. Experiments

We conducted experiments on the Snort v2.0 to evaluate the performance of the $L^+1$-MWM algorithm.
1) Description of experimental environment: For off-line experiments, a Linux machine whose CPU clock is 2.4GHz(L2 cache of 512KBytes), size of main memory is 512MBytes and kernel version is 2.6.6-8hl is used. We measure the average pattern search time in a typical search using the normal data set which generates 259 alerts out of 118,450(TCP 115,418; UDP 2,389; ICMP 84; OTHER: 559) packets under the default 1,332 Snort rules. We also measure the performance in an attack dominant search using two data sets as the attack packet trace: a DDoS attack trace from MIT-DARPA 2000 data set [10] and full-packet traces from DEFCON “capture the flag” data set [11]. We iterate the experiment using the same traffic when the number of packets is less than 20,000 to measure the average pattern search time under the same condition, in which case the preprocessing time is not included. We compare the pattern search time of the \( L^{+1} \)-MWM algorithm with that of the MWM algorithm for the tables with the same size.

For on-line experiments, we conduct experiments with real on-line packets on the Seoul National University(SNU) campus network using a Linux machine of 2.6.11-1.1369_FC4smpl kernel composed of Intel Xeon 2.8 Ghz dual(L2 cache of 512KBytes) CPU, 3GBytes memory, and Intel PRO 1000/XF LAN card. The LAN card supports the maximum 1Gbps processing. The inbound and outbound packets are diverted by mirroring Tx/Rx ports of a Cisco Catalyst 6509 switch in a subnetwork with 16 class C subnet(24) IP addresses.

2) Experiments under various LSPs: By using 100 patterns for each case whose average size is typically made one byte larger than the minimal size, we compare the search time of the MWM algorithm with that of the \( L^{+1} \)-MWM algorithm under various LSPs. To compare the pattern search time of the two algorithms, we set the value of \( B \) as 2. From off-line experiments, it is seen that the pattern search time of the \( L^{+1} \)-MWM algorithm is smaller than that of the MWM algorithm by as much as 20% in average for normal traffic in Fig.1. Moreover, when \( LSP \) is less than 5, the \( L^{+1} \)-MWM algorithm shows smaller search time than the MWM algorithm by as much as 38.87% in average. And, the \( L^{+1} \)-MWM algorithm shows better performance than the MWM algorithm by as much as 12% in average for the MIT-DARPA 2000 data set and by as much as 8% in average for the DEFCON data set. The difference between two attack data sets relies on the attack traffic characteristics (such as the number of attack packets and the frequency of the substrings in the text that match the indices in the SHIFT table). From on-line experiments, as in Fig.2, it is also shown that the pattern search time of the \( L^{+1} \)-MWM algorithm is smaller than that of the MWM algorithm by as much as 12.48% in average. For the Snort running time, it is seen that the time of the \( L^{+1} \)-MWM algorithm is smaller than that of the MWM algorithm by as much as 11.32% in average corresponding to the influence of the pattern search time by as much as 18.08%. The difference between the influence ratio of the Snort running time and that of the pattern search time is because the preprocessing time in context of the Snort rather varies depending on the preprocessing time for various types of packet. These observations show that the \( L^{+1} \)-MWM algorithm reduces the degradation of the pattern search time due to the length of the shortest pattern and also significantly improves the Snort running time.

3) Experiments with various numbers of signatures: We measure the pattern search time by increasing the number of signatures from 74 to 1621 in a step-wise manner with size 128 in average. From off-line experiments, we can obtain 25% enhancement in average for normal traffic. And, we can also obtain 20% enhancement in average for the MIT-DARPA 2000 data set and 12% enhancement in average for the DEFCON data set. The difference among different data sets is because the DEFCON data set includes more patterns in signature groups compared to the MIT-DARPA 2000 data set and normal traffic includes less patterns in signature groups compared to other data sets. For on-line experiment, we can obtain 20.12% enhancement in average. For the Snort running time, we can see that the time of the \( L^{+1} \)-MWM algorithm is smaller than that of the MWM algorithm by as much as 16.33% in average. These observations show that the \( L^{+1} \)-MWM algorithm can significantly reduce the degradation of the pattern search time due to the number of many signatures.

VI. CONCLUSION

In this paper, we proposed a new pattern matching algorithm for fast intrusion detection. After we analyzed the parameters which made an effect on the shift value of the SHIFT table, we showed that we could get the optimal shift value of the SHIFT table by padding one arbitrary imaginary byte in front of the patterns of a size of LSP. And then by conducting the experiment in Snort v2.0, we demonstrated that the \( L^{+1} \)-MWM algorithm could provide a faster search and running time than the MWM algorithm under various LSP, different number of signatures and diverse traffic conditions.

REFERENCES