An Improved Wu-Manber Multiple Patterns Matching Algorithm

Weifeng Zhang
Science and Technology on Communication Information Security Control Laboratory
Jiangnan Electronic Communication Institute
Jiaxing Zhejiang, China
zwf.zhang@gmail.com

***Abstract*—Protocol identification is an important step in information countermeasures. As a key technique in protocol identification, pattern matching algorithms aim to find the feature bit patterns of target protocol in bit stream. Traditional pattern matching algorithms are not accomplished in matching bit patterns. This paper proposed an multiple patterns matching algorithm based on the classical Wu-Manber algorithm. The proposed algorithm can improve the insufficient of the WM when dealing bit stream. The effectiveness of our improved algorithm is verified by a series of experiments.**

Keywords—Pattern matching; WM algorithm; Protocol identification.

I. INTRODUCTION

In the field of wireless communication protocol identification, pattern matching algorithms are the key step, which aim to find the features of protocols. Each wireless communication protocol has its unique bit streams. Finding the location of these unique bits provides important evidence for identifying protocols. The pattern matching algorithm is defined as finding the specific pattern P in given text T. Here T is assumed $T=t_1t_2\ldots t_n$, and $P=\{P_1P_2\ldots P_m\}$, and m<=n. When P is found in T, a match occurs. Pattern matching involves searching all locations of P in text T. Similarly, multiple patterns matching is to find all positions of more than one pattern from the given text [11][12].

AC (Aho-Corasick) [1] and WM (Wu-Manber) [2] algorithms are two most widely used multiple patterns matching algorithms. As an extension of KMP [3], AC is applied to multiple patterns [11]. A special automaton named Aho-Corasick automaton which is generated by the patterns is adopted in AC. For an "n" size text, the time complexity of pattern matching of AC is *O(n)* and has nothing to do with the number and length of patterns [12]. Considering the time of pre-treatment of patterns, the time complexity of AC can be *O(n+M)*, where M means the total length of patterns.

Compared with AC, WM is faster, simpler, and easier to realize. It was proposed by Sun Wu and Udi Manber in 1994. WM algorithm adopts the idea of the hash technology and high efficiency filtration. WM algorithm is one of the most quickly multiple patterns matching algorithms. And several algorithms based on WM have been proposed in the past decades to improve the performance of the original WM algorithm. An improved WM algorithm is proposed and used in Snort [4]. It obtains a good effect by using the SHIFT table and prefix hash table, and addding the function of prefix filter to the original WM algorithm. The author of [5] adopted the idea of QS algorithm [6] to enhance the shift distance. In [7], HASH table is modified, and the link list of suffix patterns is added to avoid the missing of matching, so that the subsequent patterns need not to be matched any more, when a pattern in the link list is matched. Literature [8] presents an improved WM algorithm, which divided patterns into different sets according to their length. It designed a specific matching method and data structure for each set and searching was performed simultaneously for different sets. This algorithm solved the problem of low efficiency caused by short patterns. Also an modified WM algorithm DHSWM [9] was proposed to resolve the problem of the less efficiency with the constant increase of number of patterns [11].

All the above pattern matching algorithms can be used to realize the wireless communication protocol identification. However, the existing algorithms are mainly based on the English characters, and not suit for matching bit streams. This paper designed an improved WM algorithm for wireless communication protocol identification. The low efficiency caused by short patterns and small size of text alphabet is considered.

The rest of this paper is organized as follows. Section 2 gives an introduction of original WM algorithm. Section 3 introduces our proposed algorithm based on WM. Section 4 is the experimental results and performance analysis. The conclusion is given in the last section.

II. INTRODUCTION TO WM ALGORITHM

WM algorithm is based on two core mechanisms. One is the filter mechanism based on hash technology, the other is the block character shift mechanism based on the bad character shift technology from BM algorithm [10]. The process of WM algorithm includes two stages, the preprocessing and the pattern searching [12].

1) Preprocessing

During the preprocessing stage, WM algorithm creates three tables, SHIFT, HASH and PREFIX. SHIFT is used to

decide the shifting distance when mismatch happens. HASH and PREFIX are used to decide which specific pattern needs to be matched when the SHIFT Table has a successful match. Before we give the detail of table establishment, we suppose text "$T=t_1t_2...tn$" and pattern set "$P=\{P_1, P_2, ..., P_l\}$", match window size is m which equals the shortest length of all patterns, block character size is B which generally is 2 or 3.

SHIFT table: SHIFT table builds an index for all the possible permutations of B characters. The SHIFT value is the distance between the most right B characters and the tail of all patterns. It decides the moving distance of string. There are two cases: first, the B-length character block X does not appear in any of the pattern, the match window moves m-B+1 characters position. Second, X appears in patterns, the algorithm matches the rightmost position X that appears in the pattern. Assuming X in appears in patterns, and the rightmost position is q, then we store m-q in SHIFT[i], where i is the hash value of X. This is expressed by the formula:

$$\mathrm{SHIFT}[i]=\begin{cases} m-B+1, X\notin\Sigma \\ min\{m-q \mid X_k=P_i[q-B+k], 1\le k\le B, P_i\in P\}, X\in\Sigma \end{cases} \quad (1)$$

where Σ is the set composed of B characters from all patterns.

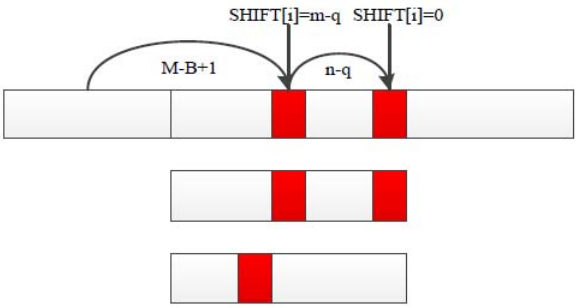

Fig.1 Matching position jumping of WM algorithm

HASH table: Get m characters in the front of patterns and calculate the hash value of the suffix of the m characters. Then link all the patterns with the same hash value together and store the entry of the list in the HASH table [12].

PREFIX table: In addition to mapping the last B characters of all patterns, the first B' characters of all patterns are also mapped into the PREFIX table. When we find a SHIFT value equals 0, then we go to the HASH table to determine if there is a match. Thus, the establishment of PREFIX table is similar to the HASH table. Instead of suffix, the prefix is used to hash calculation.

2）Pattern Search

The process of pattern search in WM is as follows:

Step1. Calculate a hash value h based on the current B characters in the text.

Step2. Check the value of SHIFT[h]: if it is > 0, shift the text and go back to step 1; otherwise, go to step 3.

Step3. Compute the hash value of the prefix of the text, denoted by *text_prefix*.

Step4. For each p (HASH[h] $\leqq p \leqq$ HASH[h+1]) whether PREFIX[p] = *text_prefix*. When they are equal, check the actual pattern against the text directly.

III. THE IMPROVED WM ALGORITHM

This paper proposed an improved WM algorithm to match bit pattern such as "*0001101101110101*" in a bit stream. The following improvements were made in our algorithm.

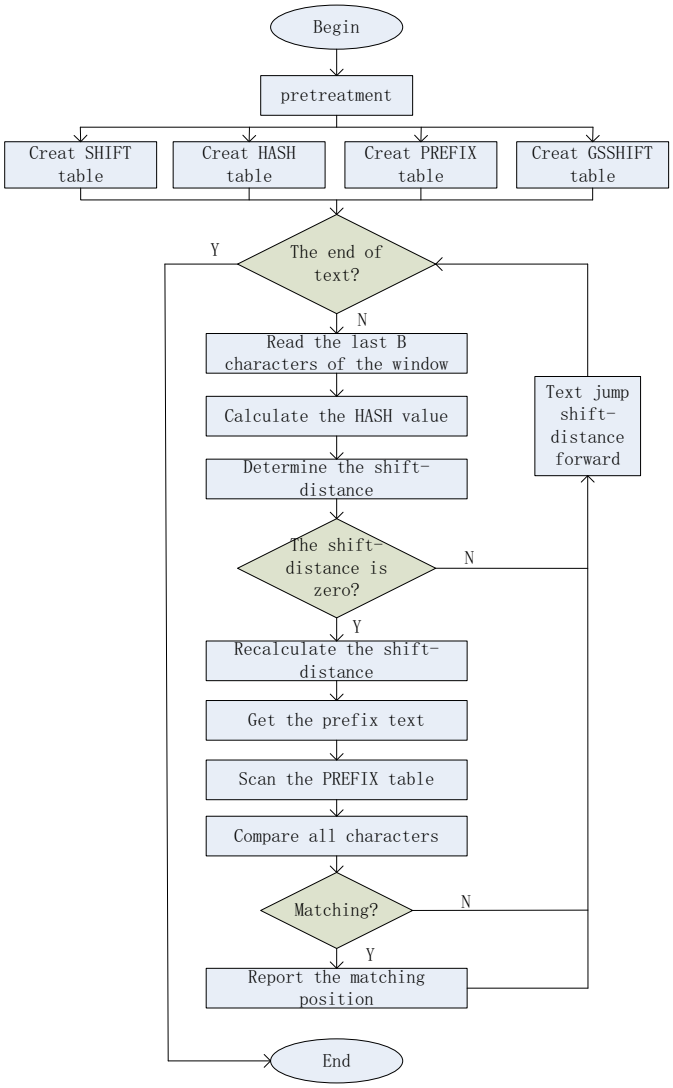

Fig 2. The flowchart of our improved WM algorithm

First, we reformat the text character. Because the characters in bit stream can only be "0" or "1", this small size alphabet decreases the shift distance when matching. By creating a mapping table, the bit string can be mapped into a bigger size alphabet. One simple example is that the bit string can be transformed to be hex string, such as "0011" mapped as "3", and "1111" mapped as "F". Thus the size of alphabet will become 16. In order to match the reformatted patterns in text, the patterns should also be expanded before matching. In the above example, bit string "001" should be expanded and reformatted as a set of patterns {"0001", "1001", "0010", "0011"}. Although this preprocessing method will increase the size of pattern set, it brings several insignificant advantages. The first is that the alphabet size extends. The second is that short patterns can be reformatted as long patterns. Third, this method makes all patterns have nearly the same length. The

above three advantages can improve the performance of the pattern match algorithm significantly.

Second, we created a GSSHIFT table in the preprocessing stage, to determine the shift distance when the corresponding SHIFT table value is zero. In the original algorithm, this shift distance always equals one when the SHIFT value is zero, and information in the patterns is lost. The value in GSSHIFT table can be calculated using the following formula.

$$\mathrm{GSSHIFT}[\mathrm{hash}(X)] = \min\{length(P_j) - B - pos\_in\_pat(X, P_j), P_j \in P\} \quad (2)$$

The *pos_in_pat*(*X*, P_j) in the above expression are as follows:

$$pos\_in\_pat(X, P_j) = \max\{pos \mid P_j[pos+i] = X[i], pos < length(P_j) - B, 0 \le i < B\} \quad (3)$$

where *X* is the last *B* characters of a pattern. The size of GSSHIFT equals the number of zero in SHIFT.

```
while (text < textend)
{
    hashVal = hash(text);
    shiftDistance = SHIFT[hashVal];
    if (shiftDistance == 0)
    {
        shiftDistance=GSSHIFT[hashVal];
        text_prefix = prefixBlock(text);
        p=HASH[hashVal];
        p_end = HASH[hashVal+1];
        while (p++<p_end)
        {
            if (text_prefix != PREFIX[p]) continue;
            else
                check the text against the pattern directly
        }
    }
    Text += shift_distance;
}
```

Fig 3. Pseudo code of pattern search of the improved WM algorithm

Here we give the flow of the improved WM algorithm in this paper. The improved algorithm has two stages as well as the original algorithm. First is the preprocessing stage which needs to establish four tables, HASH, SHIFT, PREFIX, and GSSHIFT. The establishment of HASH, SHIFT, PREFIX is the same as the original WM algorithm, and the GSSHIFT is created by using the formula (2) and (3). The bit stream text and patterns are also reformatted in this stage. Supposed that the original alphabet, that only having two symbols "0" and "1", is mapped into a alphabet whose size is 2^K. Then the *n*-length bit stream text will reformatted into a *n/k* length text and pattern P_j will reformatted as a set of patterns:

$$\{P_{j1}, P_{j2}, ..., P_{js}\}, s = 2^{r \times K - length(P_j) + 1} \quad (4)$$

where *r* is the length of reformatted pattern. Second is the pattern matching stage composed of four steps which is also used in the original WM algorithm, but some change is made. Figure 2 shows the flow chart of our algorithm and figure 3 gives the pseudo code of the pattern search stage.

IV. Experimental Results and Performance Analysis

In this chapter, several experiments are conducted to compare the performance of the original WM algorithm and our proposed algorithm. The testing machine is DELL 790, Intel core i5 3.1GHz, 4GB RAM, Windows XP, C++ on Visual Studio 2010. The testing data is a bit stream file which is demodulated from a received GPS signal and the file size is 8794KB.

First, the effect of the size of the minimum pattern is tested and compared. We randomly selected 100 patterns from bit stream which is demodulated from the wireless signal. All the patterns in each loop have the same length. In the whole test, the length of patterns ranges from 4 to 64 and the alphabet size in our algorithm is set to be 64. Results are shown in figure 4. It is obvious that long patterns have worse performance than short patterns. And our algorithm achieved better performance than the original WM algorithm, especially when the patterns are short. It is because the short patterns are reformatted as long patterns in our algorithm.

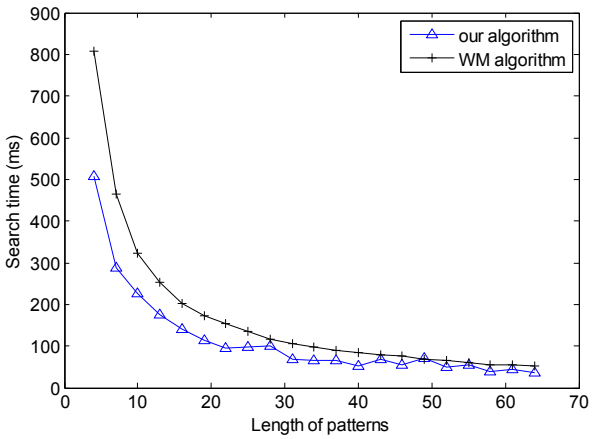

Fig 4. The effect of pattern length on the running time

Second, the effect of the number of patterns is tested and compared. Patterns are randomly selected and number of patterns ranges from 100 to 1000. The alphabet size in our algorithm is also set to be 64. Figure 5 shows the result.

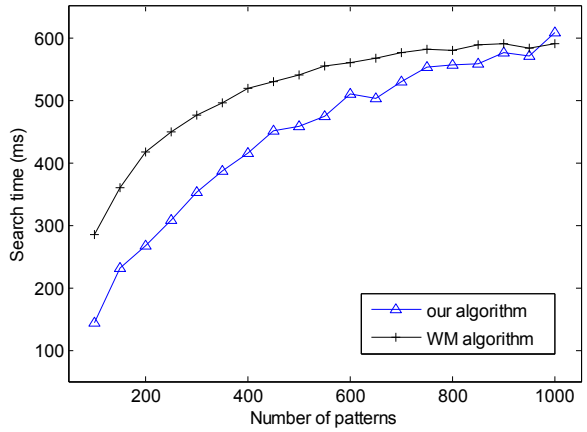

Fig 5. The effect of number of patterns on the running time

Figure 5 shows that the search time increases with the number of patterns increasing. The performance of our algorithm is better than the WM algorithm when the number of patterns is small. But our algorithm has no advantage when the number of patterns increases. The reason is that the patterns need to be reformatted in our algorithm and this processing brings more patterns.

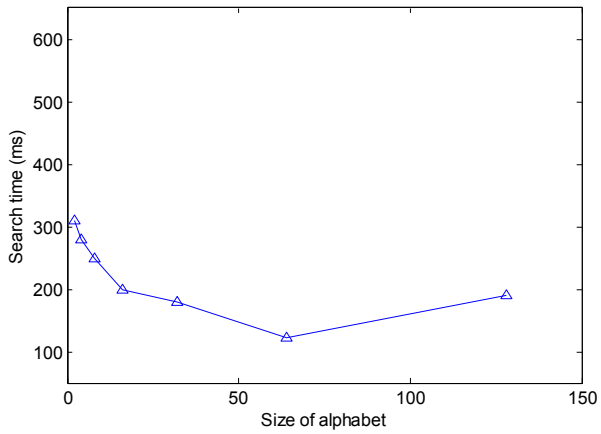

Fig 6. The effect of alphabet size on the running time

Finally, we test the effect of alphabet size. We select 100 patterns and the alphabet size ranges from 2 to 128. The selected patterns and text are reformatted and search time is recorded, with different alphabet size. Figure 6 shows that our algorithm gets best performance when the alphabet size is set to be 64 in our experiments.

V. CONCLUSION

This paper studied the WM algorithm and analyzed the performance of this algorithm. An improved WM algorithm is proposed in order to deal with wireless communication protocol bit stream. By reformatting bit stream and patterns, bit streams are reformatted as characters. It debases the effect on performance caused by small size of alphabet and short patterns. In addition, a new table GSSHIFT is adopted in our algorithm to increase the shift distance. Experiment shows that our algorithm has high efficiency when dealing bit streams.

References

[1] V. Ahoa and J. Corasickm (1975) Efficient string matching: an aid to bibliographic search, Communication of the ACM, vol. 18, no. 6, pp. 333-340.

[2] S. Wu and U. Manber (1994) A fast algorithm for multi-pattern searching, TR 94-1 7. Tucson, AZ: Department of Computer Science, University of Arizona.

[3] D. E. Knuth, J. H. Morris and V. R. Pratt (1977) Fast Pattern Matching in Strings, SIAM Journal on Computer, vol. 6, no. 2, pp. 323-350.

[4] Sourcefire Inc.. SOURCEfire SNORT 2.0 Hi-performance Multi-rule Inspection Engine, 2004.

[5] W. H. Ma, Y. M. Liu, F. Ye and X. D. Yang (2007) An improved Wu-Manber multiple patterns matching algorithm, Applied Science and Technology, Haerbin, vol.34, no. 10, pp. 32-34.

[6] D. M. Sunday (1990) A very fast substring search algorithm, Communications of the ACM, vol. 33, no. 8, pp. 132-142.

[7] X. S. Sun, Q. Wang, Y. Guan and X. L. Wang (2006) An improved Wu-Manber multiple-pattern matching algorithm and its application, Journal of Chinese Information Processing, Beijing, vol. 20, no. 2, pp. 47-52.

[8] B. Zhang, X. Chen and Z. Wu (2009) High concurrence Wu-Manber multiple patterns matching algorithm, Proceedings of the 2009 International Symposium on Information Processing, Huangshan, PR China.

[9] L. Wei-guo and H. Yong-gang (2011) DHSWM: An improved multipattern matching algorithm based on WM algorithm", Journal of Central South University (Science and Technology), vol. 42, no. 12, pp. 3765-3771.

[10] R. S. Boyer and J. S. Moore (1977), "A fast string searching algorithm," Communications of the ACM, vol. 20, no. 10, pp. 762-772.

[11] www.sersc.org

[12] www.academypublisher.com