# A Survey on using String Matching Algorithms for Network Security

Sudheer Chelluboina

In this paper, we make a survey of String Matching Algorithms for network security. It gives the summary of String Matching algorithms by focusing on network security and its subjects that are mainly focused in the literature. In this context, this paper first provides the introduction where it highlights some important aspects in the context of Network Security. Second, it briefly describes some significant methods and finally in the third it makes the analysis of the described methods.

## Introduction:

The main problem of string matching algorithm is to find all occurrences of the pattern (String) S in the text T. From [1][2][3] we formalize the string matching problem as follows, let S (S[1..n]) be a string of length n, can be referred as pattern and T(T[1…..m]) be the longer string of length m (where m>n), can be refereed as text. String-matching is a subclass of pattern matching (searching a block of text to find the occurrence of a substring). Generally string-matching algorithms search the string using sliding window mechanism. That is the search of pattern will be done with the help of window with size equal n (which is the size of the pattern). First, the left ends of the window is aligned with the text, then compare the window characters with the text characters this is called an attempt. The window slides from left to right of the text till it finds the whole match or till it reaches the end of the text [1]. The three algorithms strategies only differ in the way in which the window is shifted. They are classified as prefix searching, suffix searching and factor

searching. Every string-matching algorithms have two stages: preprocessing stage and searching stage [8][9][10][11][13][15][16]. The other challenges of string-matching algorithms are memory usage and speed. Many researchers are working on string-matching algorithms to overcome these challenges. [1], provided nearly 35 string-matching algorithms with description and code. String matching is one of the important problems that have been investigated by many applications such as Text Processing, Virus Detection, Molecular Biology, Intrusion Detection, Web Searching, Genetics, etc. There is another type of string-matching technique called approximate string-matching algorithm, the problem of string-matching that allows errors. This is more relevant issue in many applications such as Information Retrieval and Computational Biology.

Now a days there is a rapid increase in using network applications through internet, to support these applications there is a need to increase the network security to detect the malicious behaviors such as hacks, worms and virus. Generally Intrusion Detection Systems (IDS) are of two types: misuse detectors and anomaly detectors. Misuse type of IDS uses the patterns of known or recorded attacks to match with packets content data and identify intrusion, in other words signatures should be created before matching. For anomaly type of IDS, the deviations from the created normal usage patterns are flagged as intrusions. To detect the malicious behaviors IDS uses string-matching techniques on signatures, packet payload and deep packet inspection. Signature based Intrusion detection systems have a problem with the speed of the string matching algorithm because each packet has to compare with hundreds of signatures, we summarized some fast string matching algorithm to solve the above problem [4].

[5], there are three approaches to detect the deep packet inspection; automata based, heuristic based and filtering based. Automata approach matches the patterns by state

transition of deterministic finite automata or non deterministic finite automata. It has linear execution time and also consumes more memory if the data structure is not compressed. Heuristic approach checks the block of characters in the window, it moves to the next position if the match is not found. Filtering approach searches text for necessary patterns and drops if the content does not contain the pattern. Heuristic and Filtering approaches are memory efficient but suffers in worst case.

[2], set-wise Boyer-Moore-Horspool is considered to be the first string matching algorithm in network intrusion detection system; it is based on Boyer Moore (it is explained later in this paper). [6], Aho-Corasick algorithm is used for matching the multiple strings simultaneously to reduce the running time. This algorithm is based on the automata; it takes one character at a time and searches for the patterns to match the input. ACMS [15] and CIAC [16] algorithms are based on Aho-Corasick algorithm, used to serve the specific requirement in network intrusion detection systems. MDH [11] can also be used for high speed string-matching and also can be used to cut down memory usage. Now the researchers are adopting some embedded systems to increase the speed. One of the memory devices is ternary content addressable memory (TCAM)[7] it performs searching at high speed by decreasing the accessing latency. That is it simultaneously compares the input string against all the entries in its memory.

From literature we believe that a string-matching algorithm over the packet content is an important issue to be reanalyzed to improve the efficiency and speed of string-matching algorithms. Advanced string-matching algorithms should be evolved for network intrusion detection systems to increase the network speed and traffic.

## Algorithms:

## Yuebin Bai and Hidetsune Kobayashi 's String Matching Algorithm

[8], presented a new string-matching algorithm based on Boyer-Moore-Horspool algorithm. Boyer-Moore-Horspool is designed based on the bad character search, and presented two searching procedures with simple BM as search for first character(SFC) and scan for lowest frequency character(SLFC) to support the BM algorithm(which searching procedure is used depending on the situation). [8], in preprocessing stage it will generate the array NEXT which is used to decide the movement of slide to proper position for next search. They claimed that the array NEXT is very important for algorithm to bring better performance for whole string matching algorithm. In the algorithm the position of next character of pattern string is assigned as the first reference point. To generate array NEXT we have to follow the rules,

- For one character in pattern string, assign the difference of length of pattern string to pointing position of pattern string to NEXT[c].
- For the characters that are not in the pattern string, assign the increment of length of the pattern string to NEXT[c].

They presented the concept of reference point which is used as a precondition to generate the array NEXT and also to perform the next shift in searching process. Search of pattern string in text string can be from left to right or right to left. The advantage of searching from right to left is that if the mismatch occurs it need not to start from the beginning position of the substrings to present location that is it starts from the following substrings position, with this procedure useless comparisons can be avoided. As the authors do not give any name to the algorithm, so for reference we call as AN (Array Next) string matching algorithm.

## Robust Quick String Matching Algorithm

[9], proposed a novel string matching algorithm RQS to achieve the efficiency in both normal and worst cases. It combines two heuristics, improved bad character heuristic and enhanced good suffix heuristic to improve the efficiency. In general bad character heuristic works in this way, if mismatch occurs then the window is shifted right so that the mismatching character is aligned with the pattern, and if the mismatching character is not in the pattern then the window is shifted left to the position of first character of pattern. For suffix heuristic, when mismatch occurs, and if suffix exists then the window is shifted to the next occurrence of suffix in pattern. If there is no suffix then it acts same as the above process. RQS improved bad character heuristic always uses the rightmost character of the current window as the bad character, with this assumption it provides the large shift value area. Normal good suffix heuristic will forgot the characters that are matched, if they are remembered it can reduce the comparisons. In the algorithm both bad character heuristic and good suffix heuristic is calculated at every checkpoint and goes with the heuristic which has high shift value. If it is a good suffix heuristic and if the matched characters are remembered we can avoid the comparisons for next check points by comparing only the remaining characters in the patterns.

## Contents Correction Signature Hashing String Matching Algorithm

[10], proposed a new string-matching algorithm, contents correction signature hashing (CCSH) for large scale pattern sets in network to detect malicious packets in network with high speed. To apply CCSH algorithm they configured hashing table for signatures. This algorithm has two stages byte windowing and rule matching. In byte windowing comparison will be done between the window and the inspected packet data

area. Author's executed window based on two byte reference value. A two byte reference value is taken from the signature based on random position selection scheme. Random position selection scheme was designed by considering the present network environments, in which there are various warms and also a variety of worms, are emerging every day. Most of them have strong similarity between them, which means the relationship between the signatures will be very high; this increases the possibility of having an identical hashing value based on fixed positioned two byte selection method. From that it is clear that each hashing value may have longer list of signatures. With two bytes, 65,536 hashing values can be generated, which can cover all the existing rules. The calculated hashing value through byte windowing is compared with the values in signature hash table. If the window size is increased, it will decrease the number of hash calculations and then it obviously increases the searching data size required for string matching. Once the byte windowing completed, rule matching will come up to find whether the packet assures the real rules or not. Before rule matching some detailed information was prepared, that information will be compared prior to the full matching between a signature and data area. Preparation of detail information will reduce the number of matching numbers and also the false detections. This process will continue till finding a matching value or by reaching to the end of the hash table. Proposed algorithm can be used for improving efficiency and also guarantees the scalability to the number of rules.

## Multi-Phase Dynamic Hash String Matching Algorithm

[11], proposed a novel high speed string-matching algorithm called Multi-Phase Dynamic Hash (MDH) for large scale pattern sets. New denial-of-service attacks, as sending the text of extremely high matches and jamming the pattern matching modules, so it is

necessary to study the conditions like heavy load cases when there is lot of matches in the text. [11], introduced the multi-phase hash to reduce the memory usage proposed the idea to speed up the searching procedure by dynamic-cut heuristic. This algorithm was the extension of Wu-Manber (WM) algorithm [12]. For better understanding first we describe WM algorithm and then the MDH. [12], used the basic idea of Boyer-Moore (BM) String-matching algorithm, but showed how to support the multiple patterns without sacrificing speed. In the preprocessing stage three tables are build, SHIFT table, HASH table and PREFIX table. BM looks at characters from text one by one, but in WM looks as block of size B. SHIFT table is similar to BM, but not exactly same. Instead of shifting just one character, WM shifts the last B characters. SHIFT table determines the size of characters to be shifted when the text is searched. The selection of B depends on the number of patterns. To avoid matching every pattern in the pattern list used the hashing technique to reduce the number of pattern comparisons. Used the same index number for HASH table as SHIFT table, to point the list of patterns whose B characters are hashed. While mapping the last B characters of all the patterns, they also mapped the first B' characters into PREFIX table to avoid the higher likelihood collisions (after finding the match it also compared with the respective PREFIX table index to check the exact match). HASH and PREFIX tables are used to verify the match. If pattern list increases, there will be high hash collision and will reduce the average shift value, so there will be compromise in the matching. By increasing the block size B can handle the high hash collision, but if B increases, SHIFT and HASH table will also grow, which increases the memory requirement. To overcome this problem, MDH [11] will be the solution. As we know that SHIFT and HASH table entries are one-to-one relation, but MDH uses two compressed HASH table and PMT (possible matching patterns) table with SHIFT table. First HASH table is same as the above HASH table and for second hash table,

MDH rehashes the SHIFT value and stores in the PMT table. In searching stage B size text window will slide from leftmost position of text to right. For each B size text window calculates the hash function and checks the related SHIFT table entry. If the SHIFT value in not zero move the window to right and so on. Otherwise hash this text window characters again using second hash function, now use the new hash value to identify the entry in PMT table. In the last step verify every possible matching pattern linked in this entry and then move the text window right to restart whole procedure again.

## Exclusion based String Matching Algorithm

General purpose string-matching in network intrusion detection systems (NIDSec) has the high cost of running for string matching computations. [13], designed a multiple string-matching technique, $E^2xB$ to solve some specific characteristics of NIDSec and also to increase the efficiency and capacity of NIDSec. Initially, proposed ExB [14] to find the quick negatives when the search string does not exist in the packet. The major improvements are removing overhead associated with initializing the occurrence map and faster preprocessing stage. The basic idea of ExB is to check whether the input (data packet received) contains the entire fixed size bit strings of the signature string, but not considering the bits are in sequence or not as existing algorithms. ExB declares signature does not match, if one bit-string of signature is not appeared in the packet. ExB creates occurrence bitmap, marking each fixed size bit-string that exists in the packet. Occurrence bitmap strings are matched against the bit-string of each signature. They defined the false matches or false positives as follows, suppose s is small string and I be the input string, if s contains at least one character that is not in I, then s is not a substring of I. The other case is where every character of s is in I, but s is not the substring of I which is a false positive. The way of checking the given

character c belongs in I or not will make the method effective; this check is done with the help of occurrence map. To reduce the false positives they are going with pair of characters, i.e., instead of recording the occurrence of single character in string I, they are recording the pair of consecutive characters in string I. In matching process also, instead of determining the single character is present in I, $E^2xB$ checks the pair of characters present in I.

## Aho-Corasick with Magic states String matching algorithm

[15], proposed a novel and effective string-matching algorithm named ACMS. It reduces the memory requirement without sacrificing speed by using the characteristics of magic states from deterministic finite state automata (DFA). From the name of algorithm we can say that the algorithm is the combination of AC algorithm and Magic States of DFA. Automata models has two stages, first stage searches for the next state in the state table by composing the current state and input character and in second stage it identifies if the next state is the output state or not, to get the respective matching pattern id. [15], thought that by using the tiny data structure the running cost will be reduced. In general first stage is represented as the state transition matrix, the element e(x,y) is the next state when the current state y receives the input character x. [6], showed how to speed up the AC matching machines by rearranging states, now ACMS just rearranges the state 0 to state t. This rearranging process is called renumbering and that has two steps, in the first step it will find the magic states and in the second step it will partition the transition matrix. If the state is receiving the same input character, so that state will have same next state and they are calling this state as magic state. The transition matrix is partitioned based on the threshold, first matrix will have the state values that are smaller than the threshold and second matrix is compressed by the process to generate the Bitmap Matrix B and State List Matrix S. The

size of second matrix and B are same and every state of second matrix has one state in S. The process is described as, it identifies all the elements in the second matrix, if the element is not a magic state then the corresponding location in B is set to 1 and the next state is inserted to S, otherwise the corresponding location in B is set to 0. The entire algorithm is clearly explained with example in [15]. By partitioning the transition matrix and the magic state based search can reduce the running time. In worst case, the state transitions are more than the threshold, that is if the input characters are supposed to find the next states from the second matrix, then mappings will increase, this will have an impact on the performance of algorithm.

## Character Index Aho-Corasick string matching algorithm

[16], proposed a memory efficient string-matching algorithm for network intrusion detection systems by examining the number of characters in a pattern set is less than the total number of characters. This algorithm is also based on DFA model. DFA search is used because its search speed is high when compared to NFA search. As routine string-matching this algorithm also has two stages. First is the preprocessing stage, here AC DFA is constructed which was explained in the previous algorithm. Then search the DFA to find the columns that are equal to the zero matrixes. Transpose the matrix to the AC DFA, that translates the state node into the character node and each character node has n next state entries corresponding to all the states (please refer figure 5 and 6 of [16], which shows the matrix transpose). By referring to the previously searched column numbers merge these zero rows to one row. Now a character indexed table with pointers to all the character nodes. Next is the searching stage, read one character to point out the CIAC DFA character node from corresponding character index table. Then read the current state to go to the next state

value by checking the character node. Now set the previously found state as the current state. Check the current state to decide the match is found, and continues the process till all the input characters completed.

## Analysis:

In our view RQS works well for worst case scenario (i.e., if the input string and search string are same and shift value is always equal to one), because it remembers the matched characters and avoids the comparison of next check point every time. Exclusion based string matching algorithm can be improved by increasing the pair of sequence bits. This algorithm works well for small packets and rule sets. A CCSH and MDH algorithm uses the hashing scheme for fast and scalable string matching algorithm. These algorithms give adverse results if many signatures have the same hashing value, for better performance hashing value should have evenly distributed signatures. In general for automata models single memory reference is needed for each input character but the time complexity is equal to the number of input characters, i.e., the usage of memory is more. To reduce the usage of memory ACMS [15] algorithm uses the magic states as discussed earlier. [15], they compared the memory usage and throughput at different thresholds and observed that there is a high throughput with more memory usage. We feel that the threshold plays a major role in memory usage and performance. That is if threshold is small, than the mappings to the transition matrix and magic states increases which will decrease the performance. As they are considering bitmap matrices, so the usage of memory will be less when compared to the traditional one. Other automata based model CIAC string matching algorithm; it indexes the transition matrix based on characters, so it obviously allows the fast access to the matrix, and because of data structure the space complexity also reduces.

All the above algorithms are compared with the base algorithms in the paper, but with different datasets and with different methodology, so it not proper to do the quantitative analysis. Every mentioned algorithm performed well in terms of space and speed with the base algorithms mentioned in the papers. Discussed analysis are summarized in the following table,

| Algorithm | Space | Speed | Algorithm complexity |
|---|---|---|---|
| AN (array NEXT) | Fair | Fast | Simple (but necessary to consider some rules) |
| RQS (Robust Quick string matching algorithm) | Good | Fast (in worst case also) | Simple |
| CCSH (Contents Correction Signature Hashing String Matching Algorithm) | Good (depends on the size pattern list) | Fast | Simple |
| MDH (Multi-Phase Dynamic Hash String Matching Algorithm) | Good (depends on the size pattern list) | Fast | Complex |
| $E^2xB$ (Exclusion based String Matching Algorithm) | Good for small sets and Poor for large pattern sets | Fast (for small packets) | Complex (need to take care of search procedure) |
| ACMS (Aho-Corasick with Magic states String matching algorithm) | Depends on the threshold | Fast | Complex |
| CIAC (Character Index Aho-Corasick string matching algorithm) | Good | Fast | Simple |

## References:


[1]. Charras, C. & Lecroq, T., Handbook of Exact String Matching Algorithms, King's College Publications, 2004.

[2]. M. Fisk and G. Varghese., Applying fast string matching to intrusion detection, Technical Report In preparation, successor to UCSD TR CS2001-0670, University of California, San Diego.

[3]. Liuling Dai, An aggressive algorithm for multiple string matching, Information Processing Letters, Elsevier North-Holland, Inc., 2009, 109, 553-559, http://portal.acm.org/citation.cfm?id=1519558.1519953 .

[4]. James Joshi, Saurabh Bagchi, Bruce S. Davie, Adrian Farrel, Bingrui Foo, Vijay K. Garg, Matthew W. Glause, Gaspar Modelo-Howard, Prashant Krishnamurthy, Pete Loshin, James D. McCabe, Lionel M. Ni, Larry L. Peterson, Rajiv Ramaswami, Kumar N. Sivarajan, Eugene H. Spafford, George Varghese, Yu-Sung Wu and Pei Zheng, Network Security: Know It All, Elsevier Inc, 2008.

[5]. Po-Ching Lin, Ying-Dar Lin, Yuan-Cheng Lai, Tsern-Huei Lee, Using String Matching for Deep Packet Inspection, IEEE Computer Society, vol.41, pages 23-28, April 2008 [doi>10.1109/MC.2008.138].

[6]. Alfred V. Aho and Margaret J. Corasick. 1975. Efficient string matching: an aid to bibliographic search. Commun. ACM 18, 6 (June 1975), 333-340. DOI=10.1145/360825.360855 http://doi.acm.org/10.1145/360825.360855 .

[7]. M. Gao, K. Zhang, and J. Lu, "Efficient packet matching for gigabit network intrusion detection using TCAMs," in Proc. of 20th International Conference on Advanced Information Networking and Applications (AINA'06), 2006, pp. 249–254.

[8]. Yuebin Bai; Kobayashi, H., "New string matching technology for network security," Advanced Information Networking and Applications, 2003. AINA 2003. 17th International Conference, 27-29 March 2003, pp198 -201.

[9]. Jianming Yu and Yibo Xue, Robust Quick String Matching Algorithm for Network Security, Journal of Computer Science & Network Security, pp180~184, Vol.6, No.7B, July 2006.

[10]. J.S.Wang, H.K.Kwak, Y.J.Jung, H.U.Kwon, C.G.Kim and K.S.Chung, A Fast and Scalable string matching algorithm using contents correction signature hashing for network IDS, IEICE Electronic Press, vol 5, no 22, 949-953, 2008.

[11]. Zongwei Zhou, Yibo Xue, Junda Liu, Wei Zhang and Jun Li, MDH: A High Speed Multi-phase Dynamic Hash String Matching Algorithm for Large-Scale Pattern Set, ICICS 4861, pp. 201-215, 2007.

[12]. S. Wu, U. Manber," A fast algorithm for multi-pattern searching," Tech. R. TR-94-17, Dept. of Comp. Science, Univ of Arizona, 1994.

[13]. Anagnostakis Antonatos Markatos, K. G. Anagnostakis, S. Antonatos, E. P. Markatos and M. Polychronakis, A Domain-Specific String Matching Algorithm for Intrusion Detection, In Proceedings of the 18th IFIP International Information Security Conference, 2003.

[14]. E. P. Markatos, S. Antonatos, M. Polychronakis, and K. G. Anagnostakis. ExB: Exclusion-based signature matching for intrusion detection. In Proceedings of the IASTED International Conference on Communications and Computer Networks (CCN), pages 146–152, November 2002.

[15]. Nen-Fu Huang; Yen-Ming Chu; Chen-Ying Hsieh; Chi-Hung Tsai; Yih-Jou Tzang; , "A Deterministic Cost-effective String Matching Algorithm for Network Intrusion Detection System," Communications, 2007. ICC '07. IEEE International Conference, vol., no., pp.1292-1297, 24-28 June 2007, doi: 10.1109/ICC.2007.218 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4288889&isnumber=4288671 .

[16]. Jianming, Y., Yibo, X., and Jun, L. 2006. Memory Efficient String Matching Algorithm for Network Intrusion Management System, In Proceedings of Global Telecommunications Conference, San Francisco, California, USA, pp. 1--5.