RESEARCH ARTICLE OPEN ACCESS

A Survey of String Matching Algorithms

Koloud Al-Khamaiseh*, Shadi ALShagarin**
*(Department of Communication and Electronics and Computer Engineering, Tafila Technical University, 66110, Tafila, Jordan)
** (Computer and Information Technology Center, Tafila Technical University, 66110, Tafila, Jordan)

ABSTRACT
The concept of string matching algorithms are playing an important role of string algorithms in finding a place where one or several strings (patterns) are found in a large body of text (e.g., data streaming, a sentence, a paragraph, a book, etc.). Its application covers a wide range, including intrusion detection Systems (IDS) in computer networks, applications in bioinformatics, detecting plagiarism, information security, pattern recognition, document matching and text mining. In this paper we present a short survey for well-known and recent updated and hybrid string matching algorithms. These algorithms can be divided into two major categories, known as exact string matching and approximate string matching. The string matching classification criteria was selected to highlight important features of matching strategies, in order to identify challenges and vulnerabilities.

***Keywords* -** String matching, Intrusion Detection Systems (IDS), exact string matching, approximate string matching.

I. INTRODUCTION

A string searching algorithm aligns the pattern with the beginning of the text and keeps shifting the pattern forward until a match or the end of the text is reached. Let Σ be an alphabet (finite set). The Σ may be a usual human alphabet (for example, the letters A through Z in English). Other applications may use binary alphabet (Σ = {0, 1}) or DNA alphabet (Σ = {A, C, G, T}) in bioinformatics [1].

The general approaches for string matching algorithms work as follows. They scan the text using a window of the text whose size is generally equal to m. For each window of the text they check the occurrence of the pattern (this specific work is called an attempt) by comparing the characters of the window with the characters of the pattern, or by applying transitions on some kind of automaton, or by using some kind of filtering method. After achieving a match of the pattern or after a mismatch they shift the window to the right by a finite number of positions. This mechanism is usually called the sliding window mechanism. Then they repeat the sliding window mechanism until the right end of the window goes to the right end of the text [2].

The variety of known string matching algorithms creates the impression that the problem space is large, and hard to explore and address, and it is difficult to understand their similarities and differences.

The problem of string matching is well researched. This paper proposes a survey of string matching algorithms. In order to structure the string matching field and give a clear view of the problems and solution space. You will see that along with classification, we provide example of existing mechanisms. We do not pretend that this survey is detailed, since many levels could be divided into several deeper classes.

Also, new mechanisms are likely to appear, thus will add new levels to our work.

Our main objective was to select several important features of string matching mechanisms that might help researchers improve better solutions. It is important not to confuse the reader with a too extensive detailed classification. This work will be further extended by other researchers. We also do not pretend that classes divide string matching algorithms in an exclusive manner, i.e. that an instance of a particular string matching algorithm must be classified into a single class based on a given criterion. It is possible for algorithm to be comprised of several mechanisms, each of them belonging to a different class.

This paper does not propose any specific string matching algorithm. Even though we point out vulnerabilities in certain classes of string matching algorithms, our purpose is not to criticize, but to describe and attract attention to the existing problems so that they might be solved.

Following this introduction, Section 2 proposes the string matching algorithms survey. Section 3 provides an overview of related work. Section 4 discusses how to use the survey, and Section 5 concludes the paper.

II. SURVAY OF STRING MATCHING ALGORITHMS

In order to devise a survey of string matching algorithm, we observe the means used to answer two types of search models: (a) is a word (depends on the language); (b) is any sequence starting in an index-point. In order to these models, the answer models are: Exact match and approximate match respectively.

In the remainder of this section we review the recent updated and hybrid algorithms.

2.1 Exact String Matching

The exact string matching algorithms deal with finding all not part occurrences of pattern P in text T. We classify exact string matching approaches based on different character comparison methods. We differentiate between classical, deterministic finite automata, bit-parallelism and hashing string matching algorithms.

2.1.1 Classical Method

Classical string searching algorithms are based on character comparisons.

Brute-Force Algorithm: This algorithm could be considered the simplest string matching algorithm, since it performs character comparisons between the scanned text substring and the complete pattern from left to right. In the case of a mismatch or a complete match it shifts exactly one position to the right. It requires no preprocessing phase and no extra space [3].

Knuth-Morris (KMP) Algorithm 1977: This algorithm searches for occurrences of a pattern P within a main text X from left to right by employing the observation that when a mismatch occurs, what is the most we can shift the pattern so as to avoid redundant comparisons, thus benefiting from previously matched characters. This algorithm provides the advantage that the pointer in the text is never decremented [4].

Boyer-Moore (BM) Algorithm 1977: Is considered the basic and the best algorithm for single pattern matching algorithms and is used by Snort. BM algorithm matches pattern suffix from right to left and it maintains two heuristics in the case of mismatch. The first, called bad character heuristic in which the search pattern is shifted to align the mismatched character with the rightmost position where the mismatched character placed in the search pattern. The second, called good suffix heuristic, in which the mismatch occurs in the middle of the search string. Therefore the search pattern is shifted to the next occurrence of the suffix in the string [5].

The Boyer-Moore-Horspool (BMH) Algorithm 1980: It is based on the bad character search, and presented two searching procedures with simple BM [5] as search for the first character and scan for the lowest frequency character [6].

Apostolico-Giancarlo Algorithm 1986: In this approach all the suffixes of the pattern found in the text are remembered and then the shifts computed accordingly at the end of each attempt [7].

The Quick Search (QS) Algorithm 1990: This algorithm is a simplification of the Boyer Moore algorithm [5], its uses only the bad character shift [8]. Very fast in practice for short patterns and large alphabets [9].

The Boyer-Moore-Smith (BMS) Algorithm 1991: This algorithm benefits from taking the maximum shift value between the computed shifts with the text character just next the rightmost text character and the shift using the rightmost text character [10].

Colussi Algorithm 1991: This algorithm is an improvement of the Knuth Morris Pratt algorithm [4], where the set of pattern positions are divided into two disjoint subsets. The positions in the first set are scanned from left to right and when no mismatch occurs the positions of the second subset are scanned from right to left [11].

Raita Algorithm 1992: It is a tuned form from Boyer-Moore-Horspool algorithm [6]. Here, the search strategy start by comparing first the rightmost character of the window against its counterpart in the pattern, and after a match, by further comparing the leftmost character of the window and the leftmost character of the pattern. After that, the remaining characters are compared from right to left until a complete match or a mismatch occurs [12].

5The Turbo-BM (TBM) Algorithm 1994: This algorithm based on remembering the substring of the text that matched a suffix of the pattern during the last character comparisons [13].

Berry-Ravindran Algorithm 1999: Is an improvement of the Quick-Search algorithm [8], which based on the bad character rule that can be obtained by making use of a fast loop (or character unrolling cycle) [14].

2.1.2 Deterministic Finite Automaton (DFA) Method

Deterministic Finite Automaton (DFA) is a data structure that stores all the suffixes or prefixes of a string, enabling fast string matching. This method based on converting the general automaton into a deterministic one and reduces the states and the memory requirements. It has a linear execution time and also consumes more memory if the data structure is not compressed [15].

Automaton Matcher Algorithm 1974: Is the first linear algorithm based on deterministic automata, it scans the text character by character, from left to right, performing transitions on the automaton [16].

The Reverse Factor (RF) Algorithm 1994: This algorithm performs character comparisons from right to left using the smallest suffix automaton of the reverse pattern. The preprocessing phase requires linear time and space in the length of the pattern [13].

2.1.3 Bit Parallelism Method

Bit parallelism uses the essential parallelism of the bit manipulations inside computer words to perform many operations in parallel.

Aho-Corasick (AC) Algorithm 1975: Is an extension for Knuth-Morris-Pratt algorithm [4], by introducing automata. AC scans the characters one by one without any shift. At the beginning stage, AC [17] builds a Trie based state machine using the patterns to be matched. The Trie starts with empty root node (non-matching state). Each character to be matched in the patterns adds a state to the Trie starting at the root and going to the end of the pattern. Failure links points from each node to the longest prefix that leads to a partial match in the Trie. The state machine is traversed until a matching state is reached. Fig 1 shows a Trie constructed from the following strings {chart, ear, arch}. The dashed lines show the failure links, however all states failure links to the idle state are not shown. This gives a clear picture of Trie complexity for a small set of patterns. AC is a linear-time algorithm which makes it optimal for the worst case. However, AC preprocessing time and complexity increases almost exponentially with the number of characters. In addition to that, the state machine needs to be rebuilt every time anew pattern is added to the signature data base [17] [18].

Commentz-Walter Algorithm 1979: This algorithm combines the Boyer-Moore [5] technique with the Aho-Corasick algorithm [17]. In the preprocessing stage the algorithm constructs a state machine from the patterns to be matched. While in searching stage it based on the idea of Boyer-Moore algorithm [5]. The length of matching window is the minimum pattern length. And start scanning the characters of the pattern from right to left. In case of a mismatch or complete pattern match it uses a precomputed shift table to shift the window to the right [19].

Shift-Or (SO) algorithm 1992: This based on a bitwise technique. It represent the state of the search as a number, where each search step costs a small number of arithmetic and logical operations. Its efficient if the pattern length is no longer than the memory word size of the machine [20].
Backward Nondeterministic DAWG Matching (BNDM) Algorithm 1998: This algorithm uses a nondeterministic suffix automaton that is simulated using parallelism and encoding. Specifically, it works by shifting a window of length *m* over the text, for each window alignment, it searches for the pattern by scanning the current window backwards and updating the automaton configuration accordingly [21].

Backward-Oracle-Matching (BOM) Algorithm 1999: Is one of the most efficient algorithms especially for long patterns. This algorithm moves a window of size *m* on the text. For each new position of the window, it searches for the pattern by scanning the current window backwards to get secure shift [22].

2.1.4 Hashing Method

Hashing provides a simple method to avoid a quadratic number of character comparisons in most practical situations.

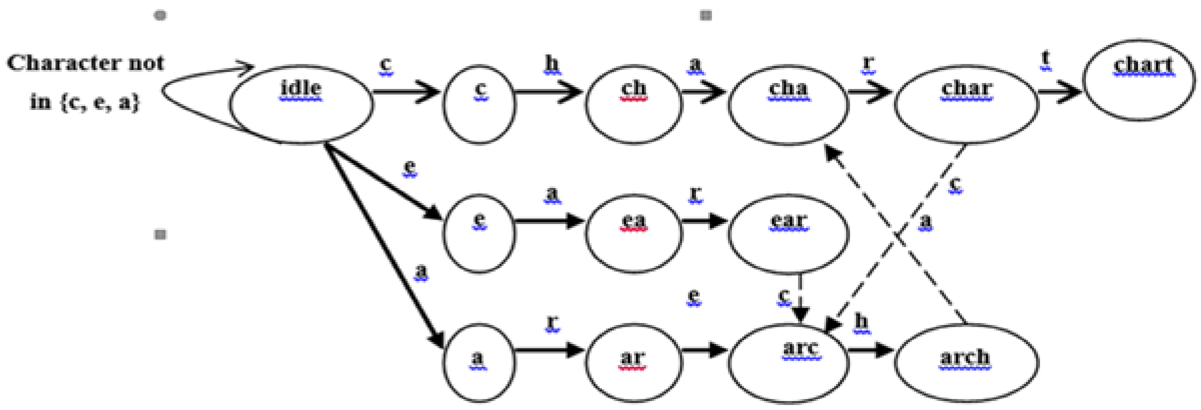

Figure 1: AC state machine for {chart, ear, arch}

Karp-Rabin (KR) Algorithm 1987: This algorithm computes the hashing function for each *m*-character substring in the text and check if it is equal to the hashing function of the pattern [23] [24].

Wu-Manber (WM) Algorithm 1994: This algorithm based on Boyer-Moore algorithm [5]. It uses the bad-character shift, and considers the characters from the text in blocks of size B instead of one by one; this will expands the effect of Bad-character shift. Also it uses hash table to index the patterns in the actual matching phase. The performance of the Wu-manber is dependent on the minimum length of the patterns. In preprocessing stage three tables (SHIFT, a HASH, and a PREFIX) are built.
An example is shown in Fig 2. The scanning phase traverses the text for the occurrences of any or all patterns by computing the hash value for the current block from the text. Then checks the SHIFT table value corresponding to this hash value, if it greater than zero, it shifts the text and computes the hash value for the new block. On the other hand, the value of the SHIFT table equals zero, the HASH and PREFIX tables are checked for matching the actual pattern against the text directly [25].

2.2 Approximate String Matching

The approximate string matching approach is a generalization of the exact string matching approach that involves finding substrings of a text string close to a given pattern string.

More specifically, the approximate string matching approach can be formally stated as follows: Let a given alphabet Σ, and a short pattern string P of length m, a large text string X of length n with m ‹‹ n, an integer k ≥ 0 and a distance function d. Approximate string matching approach consists of finding all the substrings S of T such that *d*(P, S≥ k) [26]. In general, in string matching applications the most interesting operations are: (a) substation of one character with another single character, (b) deleting one character from the given string, and (c) inserting a single character into the given string [27]. For distance functions; there are several functions implementing this process, we will consider only two very well-known functions, which are: the Hamming distance function, and Levenshtein distance function [28] [29].

Firstly, Hamming distance [28] is the number of positions with mismatching characters between two strings of equal length. So its perform substitution only. We call the approximate string matching algorithm with d Hamming distance string matching with k mismatches. Secondly, Levenshtein distance [29] is the minimum number of character insertions, deletions and substitutions that required transforming of one string to the other. We call the approximate string matching algorithm with d Levenshtein distance string matching with k differences (or k errors).

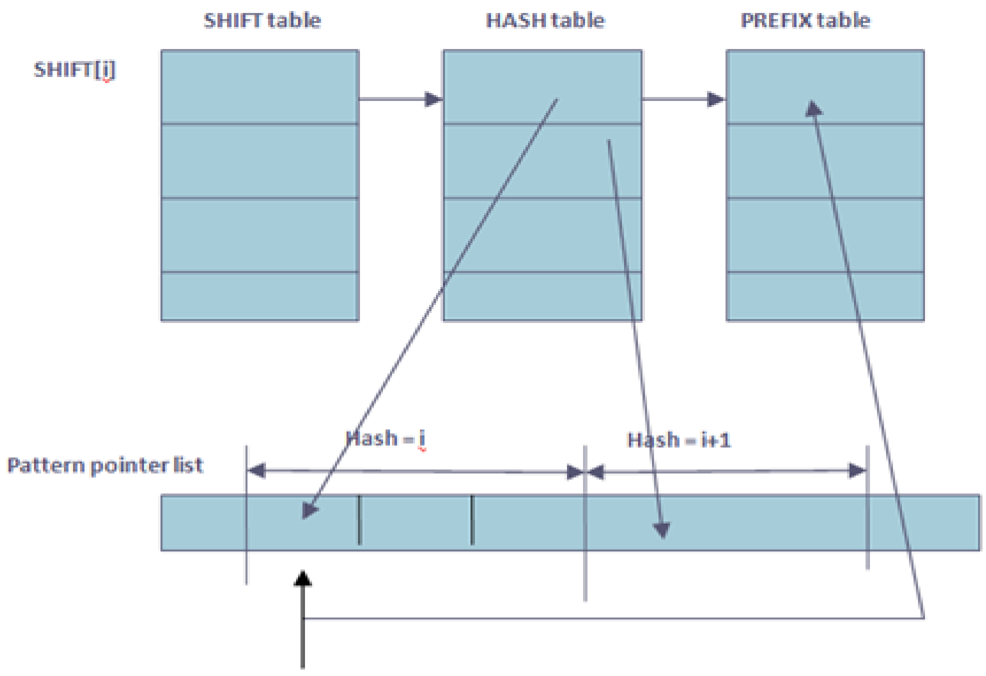

Figure 2: SHIFT, HASH, and PREFIX tables

The reasons for introducing approximate string matching are: low quality of text, heterogeneousness of databases, spelling errors in the pattern or text, searching for foreign names and searching with uncertainty [9]. We classify approximate string matching approaches based on different methods employed in searching phase; we differentiate between classical/dynamic programming, deterministic finite automata, bit-parallelism, counting and filtering string matching algorithms.

2.2.1 Classical/dynamic programming Method

Classical method as we mentioned earlier in exact string matching based on character comparisons. Dynamic programming approach also is a classical solution that computes the distance between strings [30].

Brute-Force algorithm (BF) Algorithm: This algorithm could be considered the simplest string matching algorithm, since it performs character comparisons between the scanned text substring and the complete pattern from left to right. In the case of a mismatch or a complete match it shifts exactly one position to the right. It requires no preprocessing phase and no extra space to count the number of mismatches found. If more than k has been found, shifts exactly one position to the right. At the end of the pattern we report an approximate occurrence [31].
Sellers Algorithm 1980: It is based on dynamic programming. It is try to find all approximate occurrences of P in the X [32].

Diagonal Transition Algorithm 1985: This algorithm based on computing in constant time the positions where the values along the diagonals are incremented [33].

Landau–Vishkin (LV) Algorithm 1986, 1989: This algorithm is similar to the Knuth–Morris–Pratt algorithm [4], where an array is derived from preprocessing the patterns. The text string is examined from left to right, and known information is exploited to reduce the number of character comparisons required [34] [35].

Chang–Lampe (CL) Algorithm 1992: It is a variation form of the dynamic programming array. It is based on a "column partition" approach, by exploiting a different property of the dynamic programming matrix. The algorithm again considers the fact that, along each column, the numbers are normally increasing [36].

2.2.2 Counting Method

Counting method based on arithmetic operations, thus it uses counters for every position of the text.

Baeza–Yates–Perleberg algorithm (BYP) Algorithm 1996: This algorithm is very practical and simple solution to the string searching with k mismatches problem and its performance is independent of k [37].

2.2.3 Deterministic Finite Automata Method

This approach model the search with a nondeterministic automaton (NFA).

Ukkonen (CUTOFF) Algorithm 1985: This algorithm proposed the idea of deterministic finite automaton (DFA). Its try to improve sellers algorithm [32], by considering the advantage of the geometric properties of the dynamic programming array i.e. values in neighbor cells differ at most by one. This is done by computing part of the dynamic programming array. But this algorithm has a large number of automaton states. So, we need large time and space requirements which may limit the applicability of this algorithm [38].

Wu–Manber–Myers Algorithm 1996: This algorithm try to solve Ukkonen Algorithm [38] space requirements by using a Four Russians technique [39].

Kurtz and Navarro Algorithm 1996, 1997: This is another solution to the space requirements problem by building the automaton in lazy form, i.e. build only the states and transitions actually reached in the processing of the text in Hamming approach. The automaton starts as just one initial state and the states and transitions are built as needed. By doing this, all those transitions that Ukkonen [38] considered and that were not necessary, were not built in fact [40] [41].

2.2.4 Bit-Parallelism Method

This approach is a general way to simulate simple nondeterministic finite automata (NFA) instead of converting them to deterministic one by performing many operations in parallel.

Shift-Or (SO) Algorithm 1992: The algorithm searches a pattern in a text (without errors) by parallelizing the operation of a nondeterministic finite automaton that looks for the pattern. It is treat mismatches by counting k differences using a counter of size $\log_2$, specifically, the bigger the number of bits needed to represent individual states, the smaller the length of patterns that are considered [20].

Tarhio–Ukkonen (TUD) Algorithm 1993: This algorithm performs filtering using Boyer–Moore–Horspool [6] techniques to filter the text. It generalizes both the right-to-left scanning of the pattern and the computation of shift distances to allow string matching with k-mismatches. It shifts the pattern to a position such that the rightmost $k + 1$ text characters in the previous alignment have at least one match. The shift distance is defined as the minimum one that satisfies the above condition [42].

Linear Expected Time (LET) Algorithm 1994: The algorithm work by traversing the text linearly, and at each time the longest pattern substring that matches the text is maintained. When the substring cannot be extended further, it starts again from the current text position. The algorithm uses a suffix tree on the pattern to determine in a linear pass the longest pattern substring that matches the text seen up to now [43].

Baeza-Yates (BYN) Algorithm 1996, 1999: This algorithm provide bit-parallel formula for diagonals parallelization using bits of the computer word, basing on backing the automaton along diagonals instead of rows or columns [27].
Myers (MYE) Algorithm 1998, 1999: This algorithm is based on bit parallel simulation of the dynamic programming array (matrix), by representing the differences along columns instead of the columns themselves [27].

2.2.5 Filtering Method

This method based on finding fast algorithms to drop a large number of characters from the text that cannot be matched and apply another matching algorithm for the remaining text, based on simple dynamic programming approach.

Baeza-Yates (BYPEP) Algorithm 1996: This algorithm combines the pattern partition approach with multiple string searching algorithms, by building an Aho–Corasick machine [17], to search for multiple patterns. Every match found, it extend the match, by checking if there are at most k differences, basing on the standard dynamic programming algorithm to check the edit distance between two strings [37].

COUNT Algorithm 1997: This algorithm performs filtering based on searching for substrings of the text whose distribution of characters differs from the distribution of characters in the pattern at most as much as it is possible under k differences [44].

2.3 Recent Updated and Hybrid String Matching Algorithms

In the last decade more than 50 new algorithms have been proposed for the string matching approach [2]. From literature we find that these algorithms either a kind of variations of the previous algorithms or a hybrid form that combines the features of these algorithms. We present these recent algorithms according to the main idea that leads to them.

2.3.1 Updated String Matching Algorithms

Navarro and Raffinot Algorithm 2000: This algorithm based on suffix automata. It is an adaptation to the exact string matching algorithm, BNDM [21], to allow errors. It is build a NFA to search the reversed pattern allowing errors, modify it to match any pattern suffix, and apply essentially the same BNDM algorithm using this automaton. A recent software program, called fnem nrgrepg, capable of fast, exact, and approximate searching of simple and complex patterns has been built with this method [45].

Yuebin Bai and Hidetsune Kobayashi's String Matching Algorithm 2003: This algorithm based on Boyer-Moore-Horspool algorithm [6]. Where in the preprocessing stage it generate an array NEXT which is used to decide the next position for next search, which is the first reference point. This means that it does not use the match heuristic. The pattern is compared from right to left with the text. After a complete match or in case of a mismatch, the pattern is shifted according to the pre-computed function [46].

The AKC Algorithm 2003: This algorithm is an updated form of Apostolico-Giancarlo algorithm [7]. At each search it scans the window characters from right to left and remembers every factors of the text that matches a suffix of the pattern during previous searches. Then, at the end of each search when the pattern is shifted, the AKC algorithm ensure that each text factor that previously matched a suffix of the pattern still match a factor of the pattern [47].

Simplified BNDM (SBNDM) Algorithm 2003: Additionally this algorithm is a variation of BNDM [21], it differs in the main loop where it skips the examining of longest prefixes. Which gives it lighter shift computation than BNDM [48].

Long BNDM (LBNDM) Algorithm 2003: This algorithm introduce a technique to handle long patterns with BNDM [21]. Where the pattern is partitioned in consecutive subpatterns. The leftmost subpattern is searched with the standard BNDM algorithm. Only when the match of the leftmost subpattern is found, the rest of an alignment is examined [48].

Shift-Vector Matching (SVM) 2003: This algorithm is kind of brute force approach, which maintains a bit-vector i.e. partial memory telling those positions where an occurrence of the pattern cannot end in order to transfer information from an alignment to sub-sequent alignments. The shifting based on this

bit-vector. The problem of computation of shift reduces to searching for the rightmost zero in a bit-vector [48].
Two-way Nondeterministic DAWG Matching (TNDM) Algorithm 2003: It is a two way variant of the BNDM [21] algorithm which uses a backward search and a forward search alternately. This algorithm based on Backward Nondeterministic DAWG Matching (BNDM) algorithm [21], benefiting from the nice feature of BNDM is that it simulates a nondeterministic automaton without explicitly constructing it. The main idea is that if the text character aligned with the end of the pattern is a mismatch, BNDM scans back-wards in the text if the conflicting character occurs elsewhere in the pattern. In such a situation TNDM will scan forward, i.e. it continues by examining text characters after the alignment [48]. An improvement to the TNBM algorithm is *Forward-Non-deterministic-DAWG-Matching (FNDM),* which observed that generally the forward scan for finding suffixes dominates over the BNDM backward scan. So it substitutes the backward BNDM check with a naive check of the occurrence, when a suffix is found [49].

Fast-Search Algorithms 2003: Are a family of algorithms that consists from three different variants of the Boyer-Moore [5] algorithm presented by Cantone and Faro [50]. The general base of these algorithms that at the end of each attempt the shift is computed with the bad character rule only if the first comparison of the attempt is a mismatch and the shift is computed using the good suffix rule otherwise. The first algorithm is *the Fast-Search (FS) algorithm that* compares the pattern with the current window characters from right to left at each attempt the pattern is compared with the current window characters from right to left. Then the shift is computed using the Horspool [6] bad-character rule if and only if a mismatch occurs during the first character comparison, otherwise the algorithm uses the good-suffix rule. The second algorithm from this family is the *Backward-Fast-Search (BFS) algorithm*. The algorithm benefits from combining the standard good-suffix rule with the bad-character rule to get the backward good suffix rule. Finally the *Forward-Fast-Search (FFS) algorithm* 2004, which preserve the same structure as the Fast-Search algorithm, but it uses a look-ahead character to determine larger shift advancements called forward good-suffix rule [51].

FAAST Algorithm 2005: It is a generalization to the Tarhio-Ukkonen algorithm [42], by requiring two or more matches when calculating shift distances, which makes the approximate string matching process significantly faster than the Tarhio-Ukkonen algorithm. Instead of requiring at least one match in the last $k + 1$ characters of the text in the previous alignment, the new algorithm requires at least x matches in the last k + x characters when calculating shift distances, where x is a small integer value (typically 2 or 3 in their experiments) [52].
The Wide Window (WW) Algorithm 2005: In this algorithm each search is divided into two steps. The first step consists in scanning the m rightmost characters of the window from left to right starting with the initial state until a full match or a lack of transition. And the second step consists in scanning the $m-1$ leftmost characters of the window from right to left. An improvement to the *WW* algorithm is *Bit Parallel Wide Window Algorithm (BPWW)* [53].

The Linear DAWG Matching (LDM) A*lgorithm* 2005: The searching in this algorithm as in WW algorithm [53], is also divided into two steps. The first step consists in scanning the m leftmost characters of the window from right to left starting with the initial state until a full match or a lack of transition. And the second step consists in scanning the m rightmost characters of the window from left to right [53].

Boyer-Moore-Horspool Algorithm Using Probabilities 2006: An updated form of the Horspool algorithm [6] by applying probabilities on the symbols within the pattern, where there are different probabilities for different symbols, the idea works by changing the order in which the symbols of the pattern are compared to the symbols of the current window of the text such that the probability of a mismatch is statistically maximized [54].

2Block Algorithm 2007: This algorithm is built on the original Boyer-Moore algorithm [5]. The two key ideas are to keep track of all the previously matched characters within the current window and not to move the searching position to the end of the pattern when a mismatch occurs. This approach has increased the average shift amounts and guarantees that any character of the text is read at most once [55].

Multi-Phase Dynamic Hash (MDH) String Matching Algorithm 2007: Is an extension to Wu-Manber algorithm [25]. The algorithm try to overcome the SHIFT and HASH tables growing i.e. increasing memory requirement in Wu-Manber algorithm by using two compressed HASH table and PMT (possible matching patterns) table with SHIFT table. The first HASH table is the same as Wu-Manber HASH table and for the second hash table, MDH rehashes the SHIFT value and stores in the PMT table. At each attempt the hash function for a block of text of size B is calculated and then checking the related SHIFT table entry. If the SHIFT value in not zero the block is moved to right and so on. Otherwise the hash function of this text block characters is calculated again using the second hash function, now identify the entry in PMT table by using the new hash value. In the last step a verification for every possible matching pattern linked in this entry and then moving the text window right to restart the whole procedure again [56].

Aho-Corasick with Magic states (ACMS) String matching algorithm 2007: Is an adaptation to Aho-Corasic algorithm [17], by reducing the memory requirement without sacrificing speed by benefiting from the characteristics of magic states in deterministic finite state automata. The algorithm rearrange the states, this is done into 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 and State List Matrix. The size of second matrix and Bitmap Matrix are the same and every state of second matrix has one state in State List Matrix. The search process works by identifying all the elements in the second matrix, if the element is not a magic state then the corresponding location in Bitmap Matrix is set to 1 and the next state is inserted to State List Matrix, otherwise the corresponding location in Bitmap Matrix is set to 0. The entire algorithm is clearly explained with example in [57].

Hashing Algorithms 2007: It is an adaptation of Wu-Manber algorithm [25] as multiple string matching to single string matching algorithm. The algorithm introduced *K* parameter of the algorithm which strongly affects the performance and the resulting complexity. More details function and calculations are presented by the author [58].

Two-Sliding-Windows (TSW) Algorithm 2008: Is a variation of the Berry-Ravindran algorithm [14]. The algorithm works by dividing the text into two equal parts and searches for matches by using two windows simultaneously. Where the first window scans the left part of the text from left to right, while the second window shifts from right to left scanning the right part of the text. This gives a parallel search, which is suitable for parallel processors structures. The TSW algorithm uses the Berry-Ravindran [14] bad character rule to calculate the shift value for better shift values [59].

Boyer-MooreHorspool with q-grams (BMHq) Algorithm 2008: It is a variation of Horspool algorithm [6], where at each alignment of the pattern, the algorithm reads and computes an integer i.e. *fingerprint* for a q-gram of characters. The scanning works by comparing the last q-gram of the pattern with the corresponding q-gram in the current window of the text, and then tests the equality of their fingerprints [60].

The Extended-Backward-Oracle-Matching Algorithm 2008] it is very fast and flexible variation of the Backward-Oracle- Matching algorithm [22]. It introduces tries two subsequent transitions for each iteration of the fast-loop with the aim to find with higher probability an undefined transition [61].

Fast pattern matching for intrusion detection using exclusion and inclusion filters (Exscind) Algorithm 2011: This algorithm try to reduce the number of times to perform pattern matching. It is introduces an exclusion-inclusion filter programmed only with signatures prefixes, using a specially modified Wu-Manber pattern matching algorithm. The exclusion-inclusion filter is a modified Bloom filter that produces a list of probable matching signatures for each suspect packet [62].

Function and Data Parallelization of Wu-Manber Pattern Matching for Intrusion Detection Systems 2012: This work introduces three parallel implementations of the Wu-Manber pattern matching algorithm [25]. The first implementation, the Shared Position (SP) algorithm, utilizes several scanning windows running in parallel and using a shared position variable. The second implementation, the Trace Distribution (TD) algorithm, divides the trace equally among the parallel threads. The third implementation (DSP) combines the first two algorithms [63].

2.3.2 Hybrid String Matching Algorithms

SSABS and TVSBS Algorithms 2004: These algorithms are a combination of the shifting method of the Quick-Search algorithm [8] and the testing method of the Raita algorithm [12]. This done by comparing the rightmost and leftmost characters first, and then continuing the comparison of the other characters from right to left until a complete match or a mismatch occurs. After each search, the shift of the window is computed by the Quick-Search [8] bad character rule for the next character to the window [64].

Robust Quick String Matching (RQS) Algorithm 2006: This algorithm combines two heuristics, where bad character heuristic and good suffix heuristic are enhanced to improve the efficiency. In general the bad character heuristic always uses the rightmost character of the current window as the bad character, so this provides the large shift value. For normal good suffix heuristic the characters that are matched will be forgotten, if they are remembered it can reduce the comparisons. Both the 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 [65].

Franek-Jennings-Smyth (FJS) Algorithm 2007: It is a hybrid algorithm that combines the linear worst case time complexity of Knuth-Morris-Pratt algorithm [4] and the sublinear average behavior of Quick-Search algorithm [8]. Each attempt of the search is divided into two phases. In the first phase, as with the Quick-Search approach, the FJS algorithm first compares the rightmost character of the pattern with its corresponding character in the text, if a mismatch occurs, a Quick-Search shift is used, when a match is found the FJS algorithm invokes the second step. Otherwise another Quick-Search shift occurs [2].
The second phase of the algorithm consists in a Knuth-Morris-Pratt pattern-matching starting from the leftmost character and, if no mismatch occurs, then whether or not a match is found, a Knuth-Morris-Pratt shift is performed followed by a return to the first step [66].

The Forward-Backward-Oracle-Matching Algorithm 2008: This algorithm mixes the ideas of the Extended-BOM algorithm [61] with those of the Quick-Search algorithm [8] by focusing on the character that follows the current window (the forward character) while computing the shift advancement [67]. For more improvement to this approach is the bit-parallel version of the Forward-BOM algorithm, which called Forward SBNDM Algorithm (FSBNDM) [2].

The Genomic Oriented Rapid (GRASPm) Algorithm 2009: Is an algorithm that combines the shifting method based on the Horspool [6] bad-character rule and the filtering method based on a hash function computed on 2-grams in the pattern [68].

Hybrid Multithreaded Pattern Matching Algorithm 2012: This algorithm based on two well-known multiple pattern matching algorithms Wu-Manber [25] and Aho-Corasick [17]. Where the algorithm benefits from wu-manber power in matching long patterns and Aho-Corasick for short patterns. It divide the patterns between the two algorithms to keep the workloads balanced for optimal performance. Additionally multiple threads are used to maximize the performance of the hybrid algorithm [69].

III. RELATED WORK

Several works are introduced to summarize and explore current techniques to cope with the problem of string matching.

Gonzalo Navarro 2001 [27] presented a tour to approximate string matching algorithm according to the pattern length and the time complexity for different string matching classes. This classification is more complete, since it considerers exact string matching algorithm basing on wider area of classification.

P.D. Michailidis and K.G. Margaritis 2001, 2002 [3] [31] proposed two surveys, one focused on on-line exact string matching algorithms, while the other considered on-line approximate string matching algorithm. Both of them also provide experimental results of each class, in order to explore its good and weakness aspects, and to make it easier for appropriate application deployment.

Christian Charras and Thierry Lecroq 2004 [15] presented a book that investigate exact string matching algorithm in details, including the main idea and application and the source code of the available exact string matching algorithms.

Simone Faro and Thierry Lecroq 2010, 2013 [70] [2] provided two strong surveys, the first one [70] gave a comprehensive experimental evaluation for exact string matching algorithms. The second one [2] reviewed the string matching algorithms which have been proposed in the last decade 2000-2010 and presented experimental results in order to bring order among the dozens of articles published in recent years.

Vidya SaiKrishna, Prof. Akhtar Rasool, and Nilay Khare 2012 [9] explored the various diversified fields where string matching has an eminent role to play and is found as a solution to many problems.

Kamal Alhendawi and Ahmad Baharudin 2013 [71] introduced a short survey for five of well-known string matching algorithms, including theoretical analysis, empirical testing of the execution time based on the change of two factors (text size and pattern size), then it measured the efficiency of each string matching algorithm in term of estimated execution time.

While Gulfishan Firdose Ahmed and Nilay Khare 2014 [72] presented a survey of several hardware based string matching algorithms such as Brute Force, KMP [4], and Aho-Corasicks [17] with their applications.

IV. HOW TO USE THE SURVEY

During the design of this survey, we try to select significant features of string matching algorithm. How can this survey be used?

A map of string matching research field. For beginner researchers, this survey provides a comprehensive overview for a quick introduction to the string matching field. While experienced researchers can extend this survey to structure and organize their knowledge in this field. This should lead to defining new directions for string matching research.

Exploring new string matching strategies. This survey explored a few strategies seen infrequently in the wild.

Common vocabulary. This survey offer a common vocabulary for string matching mechanisms.

Understanding string matching constrains. This survey highlights common performance constraints, so understanding these problems will attract research efforts on solving them.

V. CONCLUSION

String matching field contains numerousness mechanisms, which darks a global view of the string matching approach. This paper is an attempt to clear the ambiguity and structure the knowledge in this field. One benefit we foresee from this survey is that of keeping easier cooperation among researchers. Good surveys will facilitate communication and offer a common language for discussing solutions. They will also clarify how different mechanisms are likely to work in concert, and identify areas of remaining weaknesses that require additional work.

There is a pressing need for the research community to develop common metrics for string matching evaluation. Surveys will be helpful in shaping these tasks.

The proposed survey is not complete. Since new matching approaches will appear, that cannot be imagined. May they will highlight new features for classification. We hope this survey will offer a foundation for classifying string matching algorithms in intrusion detection systems. So as the field grows, the survey will also grow and be refined.

REFERENCES

[1] Y. Hong, X. Ke, and C.Yong, "*An improved Wu-Manber Multiple Patterns Matching Algorithm*", in *Performance, Computing, and Communications Conference, IPCCC 2006. 25th IEEE International*, 6 pp. – 680, 2006.

[2] S. Faro, T. Lecroq, "*The Exact Online String Matching Problem: a Review of the Most Recent Results*", *ACM Computing Surveys (CSUR) Surveys Homepage archive*, Volume 45 Issue 2, Article No. 13, February 2013.

[3] P. Michailidis and K. Margaritis, "*On-line String Matching Algorithms: Survey and Experimental Results*", *International Journal of Computer Mathematics*, volume 76, Issue 4, 2001.

[4] D. Knuth, J. Morris, and V. Pratt, "*Fast pattern matching in strings*", *SIAM Journal on Computing*, volume 6(1), 322–350. (1977).

[5] R. Boyer, J. Moore, "*A fast string searching algorithm*", *Communication of the ACM*, volume 20(10), 762–772, (1977).

[6] R. HORSPOOL, "*Practical fast searching in strings*", *Softw. Pract. Exp.*, Volume 10, 6,

[7] A. Apostolico, and R. Gianarlo, "*The Boyer-Moore-Galil string searching strategies revisited*", *SIAM J. Comput*., Volume 15, 1, 98–105, 1986.

[8] D. Sunday, "*A very fast substring search algorithm*", *Communications of the ACM*, Volume 33, No. 8, pp.132-142, 1990.

[9] V. SaiKrishna, A. Rasool, and N. Khare, "*String Matching and its Applications in Diversified Fields*", *International Journal of Computer Science Issues (IJCSI)*, Volume 9 Issue 1, p219-226, Jan2012.

[10] P. Smith, "*Experiments with a very fast substring search algorithm*", *Softw. Pract. Exp.*, Volume 21, No. 10, pp. 1065-1074, 1991.

[11] L. Colussi, "*Correctness and e efficiency of the pattern matching algorithms", Information and Computation*, Volume 95 Issue 2, Dec. 1991.

[12] T. Raita, "*Tunning the Boyer-Moore-Horspool string searching algorithm*", *Software- Practice and Experience*, Volume 22, No. 10, pp. 879-884, 1992.

[13] M. Crochemore, A. Czumaj, L. Gasieniec, S. Jarominek, T. Lecroq, W. Plandowski, and W. Rytter, "*Speeding Up Two String Matching Algorithms*", *Algorithmica*, Volume 12, No. 4-5, pp. 247-267, 1994.

[14] T. Berry, and S. Ravindran, "*A fast string matching algorithm and experimental results*", In*: Holub, J., Simánek, M. (eds.) Proceedings of the Prague Stringology Club Workshop 1999, Collaborative Report DC-99-05, Czech Technical University, Prague, Czech Republic*, pp. 16-26, 2001.

[15] C. Charras, and T. Lecroq, *Handbook of exact string matching algorithms*. King's College Publications, 2004.

[16] A. Aho, J. Hopcroft, and J. Ullman, *The design and analysis of computer algorithms,* Addison-Wesley, 1974.

[17] A. Aho and M. Corasick. "*Efficient string matching: An aid to bibliographic search*", *Communications of the ACM*, volume 18(6), pp. 333-340, 1975.

[18] M. Aldwairi, T. Conte, and P. Franzo, "*Configurable String Matching Hardware for Speeding up Intrusion Detection*", *ACM SIGARCH Computer Architecture News*, volume 33(1):99–107, 2005.

[19] B. Commentz-Walter, "*A string matching algorithm fast on the average*", in Proc. *6th International Colloquium on Automata, Languages, and Programming*, pp. 118-132, 1979.

[20] R. Baeza-Yates, and G. Gonnet, "*A new approach to text searching*", *Communications of the ACM*, Volume 35, No. 10, pp. 74-82, 1992.

[21] G. Navarro, M. Raffinot, "*A Bit-Parallel Approach to Suffix Automata: Fast Extended String Matching*", in Proc. of *the 9th Annual Symposium on Combinatorial Pattern Matching,* No. 1448, pp. 14-33, Springer-Verlag, Berlin, 1998.

[22] C. Allauzen, M. Crochmore, and M. Raffinot, "*Factor oracle: a new structure for pattern matching*", In *OFSEM '99 Proceedings of the 26th Conference on Current Trends in Theory and Practice of Informatics on Theory and Practice of Informatics*, pp. 295-310, 1999.

[23] C. harras, and T. Lecroq, "*Exact string matching algorithms*", *Technical Report*, 1997.

[24] T. Lecroq, and M. Christian, "*Exact String Matching Algorithms*," *Laboratoire d'Informatique de Rouen,* June 2004, Web page: http://www-igm.univ-mlv.fr/~lecroq/string/index.html, accessed 7.6.2014.

[25] S. Wu, and U. Manber, "*Fast algorithm for multi-pattern searching*", *Technical Report* TR94-17*, University of Arizona at Tuscon,* 1994.

[26] S.Wu, and U. Manber, "*Fast text searching allowing errors*", *Communications of the ACM*, volume 35(10), 83–91, 1992.

[27] G. Navarro, "*A Guided Tour to Approximate String Matching*", *ACM Computing Surveys*, Volume 33 Issue 1, Pages 31-88, March 2001 .

[28] D. Sankoff, S. Andmainville, "*Common Subsequences and Monotone Subsequences*", *Addison-Wesley*, Reading, MA, 363–365, 1983.

[29] V. Levenshtein, "*Binary codes capable of correcting spurious insertions and deletions of ones*", *Probl. Inf. Transmission*, volume 1, 8–17, 1965.

[30] R. Wagner, and M. Fischer, "*The string to string correction problem*", *Journal of theAssociation for Computing Machinery*, volume 21(1), 168–173.

[31] P. Michailidis, K. Margaritis, "*On-line approximate string searching algorithms: survey and experimental results*", *International Journal of Computer Mathematics, Intern. J. Computer Math.*, Volume 79(8), pp. 867–888, 2002.

[32] P. Sellers, "*The Theory and Computation of Evolutionary Distance: Pattern Recognition*", *Journal of Algorithms*, volume 1(4), 359–373, 1980.

[33] E. Ukkonen, "*Algorithms for approximate string matching*", *Information and Control*, volume 64, 100–118, 1985a. Preliminary version in Proceedings of *the International Conference Foundations of Computation Theory* (LNCS, vol. 158, 1983).

[34] G. Landau, and U. Vishkin, "*Efficient string matching with k mismatches*", *Theoretical Computer Science*, volume 43(2–3), 239–249, 1986.

[35] G. Landau, and U. Vishkin, "*Fast parallel and serial approximate string matching", Journal of Algorithms archive,* Volume 10 Issue 2, June 1989.

[36] W. Chang, and J. Lampe, "*Theoretical and Empirical Comparisons of approximate string matching algorithms*", In Proc. of *the 3rd Annual Symposium on Combinatorial Pattern Matching, No. 664 (Springer-Verlag, Berlin*), pp. 175–184, 1992.

[37] R. Baeza-Yates, and C. Perleberg, "*Fast and practical approximate string matching*", *Information Processing Letters*, volume 59(1), 21–27, 1996.

[38] E. Ukkonen, "*Finding approximate patterns in strings*", *Journal of Algorithms volume,* volume 4(1–3), 132–137, 1985.

[39] S. Wu, U. Manber, and G. Myers, "*A subquadratic algorithm for approximate limited expression matching*", *Algorithmica,* volume 15(1), 50–67, 1996.

[40] S. Kurtz, "*Approximate string searching under weighted edit distance*", In: Proc. *of the 3rd South American Workshop on String Processing (Carleton University Press),* pp. 156–170, 1996.

[41] G. Navarro, "*A partial deterministic automaton for approximate string matching*", In Proc. of *the 4th South American Workshop on String Processing (Carleton University Press),* pp. 112–124,1997.

[42] J. Tarhio, and E. Ukkonen, "*Approximate boyer-moore string matching, SIAM*", *Journal on Computing,* volume 22(2), 243–260, 1993.

[43] W. Chang, and E. Lawler, "*Sublinear approximate string matching and biological applications*", *Algorithmica,* volume 12, 4/5, 327–344, 1994.

[44] G. Navarro, "*Mutiple approximate string matching by counting*", In Proc. of *the 4th South American Workshop on String Processing (Carleton University Press),* pp. 125–139, (1997).

[45] G. NAVARRO, 2000b. Nrgrep: "*A fast and flexible pattern matching tool*", *Software—Practice & Experience,* Volume 31 Issue 13, November 10, 2001.

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

[47] M. Ahmed, M. Kaykobad, and R. Chowdhury, "*A new string matching algorithm*", *Int. J. Comput. Math.* Volume 80, 7, 825–834, 2003.

[48] H. Peltola, and J. Tarhio, "*Alternative algorithms for bit-parallel string matching*", In Proceedings of *the 10th International Symposium on String Processing and Information Retrieval SPIRE'03, M. A. Nascimento, E. S. de Moura, and A. L. Oliveira, Eds. Lecture Notes in Computer Science, vol. 2857. Springer-Verlag, Berlin, Manaus, Brazi*l, 80–94, 2003.

[49] J. Holub, and B. Urian, "*Fast variants of bit parallel approach to suffix automata*", In the *Second Haifa Annual International Stringology Research Workshop of the Israeli Science Foundation*, http: // www. cri. haifa. ac. il/ events/ 2005/ string/ presentations/ Holub. pdf ,accessed 8/07/2014.

[50] D. Cantone, S. Faro "*Fast-Search: a new efficient variant of the Boyer-Moore string matching algorithm*", In WEA 2003. *Lecture Notes in Computer Science,* vol. 2647. Springer-Verlag, Berlin, 247– 267.

[51] D. Cntone, and S. Faro, "*Searching for a substring with constant extra-space complexity*" In Proc. of *Third International Conference on Fun with algorithms, P. Ferragina and R. Grossi,* pp.118–131, 2004.

[52] Z. Liu, J. Borneman, and T. Jiang, "*A Fast Algorithm for Approximate String Matching on Gene Sequences*", *Combinatorial Pattern Matching Lecture Notes in Computer Science,* Volume 3537, pp 79-90, 2005.

[53] L. He, B. Fang, and J. Sui, "*The wide window stringmatching algorithm*", *Theor. Comput. Sci*., volume 332, 1- 3, 391–404, 2005.

[54] M. NEBEL,"*Fast string matching by using probabilities: on an optimal mismatch variant of Horspool's algorithm*", *Theor. Comput. Sci.* volume 359, 1, 329–343, 2006.

[55] M. Sustik, and J. Moore, "*Strin/g searching over small alphabets",* In *Technical Report TR-07-62. Department of Computer Sciences, University of Texas at Austin,* 2007.

[56] Z. Zhou, Y. Xue, J. Liu, W. Zhang and J. Li, "*MDH: A High Speed Multi-phase Dynamic Hash String Matching Algorithm for Large-Scale Pattern Set*", *ICICS 4861,* pp. 201-215*, 2007. Lecture Notes in Computer Science*, Volume 4861, pp 201-215, 2007.

[57] N. Huang; Y. Chu; C. 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*, pp.1292-1297, 24-28 June 2007, URL, accessed 09/7/2014: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4288889&isnumber=4288671

[58] T. Lecroq, "*Fast exact string matching algorithms*", *Inf. Process. Lett*. Volume 102, 6, 229–235, 2007.

[59] A., Hudaib, R. Al-khalid, D. Suleiman, M. Itriq, and A. Al-anani, "*A fast pattern matching algorithm with two sliding windows (TSW)*", *J. Comput. Sci*., volume 4, 5, 393–401, 2008.

[60] P. Kalsi, H. Peltola, and J. Tarhio, 2008. "*Comparison of exact string matching algorithms for biological sequences*", In Proceedings of *the Second International Conference on Bioinformatics Research and Development, BIRD'08*, 2008.

[61] S. Faro, and T. Lecroq, "*Efficient variants of the Backward-Oracle-Matching algorithm*", In Proceedings of *the Prague Stringology Conference* 2008, *J. Holub and J. ˇZ ˇd ´arek, Eds. Czech Technical University in Prague, Czech Republic*, 146–160.

[62] M. Aldwairi, and D. Alansari, "*Exscind: Fast pattern matching for intrusion detection using exclusion and inclusion*

filters", Next Generation Web Services Practices (NWeSP), 2011 7th International Conference on, pp. 24-30, 2011.

[63] M. Kharbutli, M. Aldwairi, and Abdullah Mughrabi, "*Function and Data Parallelization of Wu-Manber Pattern Matching for Intrusion Detection Systems*", *Network Protocols & Algorithms,* volume 4(3), 2012.

[64] S. Sheik, S. Aggarwal, A. Poddar, N. Balakrishanan, and K. Sekar, "*A fast pattern matching algorithm*", *J. Chem. Inf. Comput.* Volume 44, 1251–1256, 2004.

[65] M. NEBEL, "*Fast string matching by using probabilities: on an optimal mismatch variant of Horspool's algorithm*", *Theor. Comput. Sci.* volume 359, 1, 329–343, 2006.

[66] F. Franek, C. Jennings, and W. Smyth, "*A simple fast hybrid pattern-matching algorithm*", *J. Discret. Algorithms,* volume 5, 4, 682–695, 2007.

[67] S. Faro, and T. Lecroq, "*Efficient variants of the Backward-Oracle-Matching algorithm*", In Proceedings of *the Prague Stringology Conference* 2008*, J. Holub and J. ˇZ ˇd ´arek, Eds. Czech Technical University in Prague, Czech Republic*, 146–160.

[68] S. Deusdado, and P. Carvalho, "*GRASPm: an efficient algorithm for exact pattern-matching in genomic sequences*", *Int. J. Bioinformatics Res. Appl.* Volume 5, 4, 385–401, 2009.

[69] M. Aldwairi, and N. Ekailan, "*Hybrid Multithreaded Pattern Matching Algorithm for Intrusion Detections Systems*", *Journal of Information Assurance and Security*, Volume 6 (2011) pp. 512-521, 2011.

[70] S. Faro, and T. Lecroq, "*The exact string matching problem: a comprehensive experimental evaluation*", *Report arXiv*: 1012.2547, 2010.

[71] K. Hendawi, and A. Baharudin, "*String Matching Algoritms (SMAs): Survey & Empirical Analysis*", *Journal of Computer Sciences and Management,* Volume 2, Issue 5, 2013.

[72] G. Ahmed and N. Khare, "*Hardware based String Matching Algorithms: A Survey*", *International Journal of Computer Applications,* volume 88(11):16-19, February 2014.