A SURVEY ON RECENT DFA COMPRESSION TECHNIQUES FOR DEEP PACKET INSPECTION IN NETWORK INTRUSION DETECTION SYSTEM

[1]S.Prithi, [2]S.Sumathi

[1]Department of CSE, RVS College of Engineering and Technology, Coimbatore. (prithi.samuel@gmail.com)
[2]Department of EEE, PSG College of Technology, Coimbatore. (ssi@eee.psgtech.ac.in)

ABSTRACT:

The vast majority of the system security application in today's systems depends on Deep packet inspection. In recent years regular expression matching are used as an important operator which examines whether or not the packet's payload can be matched with a group of predefined regular expression. Regular expressions are parsed using the Deterministic Finite Automata representations. Conversely, to represent regular expression sets as DFA the system needs large amount of memory, an excessive amount of time, or an excessive amount of per flow state limiting their practical applications. There are several compression techniques available which provide memory efficient finite automata. This paper presents the analysis of these several compression techniques that are used to decrease the number of states such as HFA, History based NFA, DFA, Lazy DFA, NFA- OBDD, HFA, H-FA, H-cFA, XFA, D^2FA, CD^2FA, δFA, δ^2FA, Dual Finite Automata and DFA/EC. In this paper the Intelligent Optimization Grouping Algorithms (IOGA) are discussed to resolve the state blow up problem. As a result of using IOGA the system provides memory efficient automata by dispensing the regular expression sets into various groups and optimizing the DFAs.

Keywords: *Deep Packet Inspection, Deterministic Finite Automata, Intelligent Optimization Grouping Algorithms, Network Intrusion Detection System, Pattern Matching, Regular Expression*

1. INTRODUCTION

Today, a computer network has become an essential part of our day by day life. Internet has a fast growth from the most recent decade with increasing requirement of society on it. Internet provides a wide range of benefits to society however it is infected by many security attacks that disrupt the functionality of networking and computing infrastructure. To enhance the security of the network a large number of devices are introduced. Network Intrusion Detection Systems (NIDS) are amongst the foremost broadly used for this purpose [28]. Snort [22] and Bro [23] are two open source NIDS examples that have been broadly used to safeguard the network.

Network Intrusion Detection Systems use Deep Packet Inspection (DPI) for a variety of applications that enhances security like spam, monitoring and detecting viruses, malevolent traffic, unauthorized access and attacks. The main role of deep packet inspection is to permit Network Intrusion Detection System to effectively match the details of the network packets with respect to signature attacks and thereby be aware of malicious traffic. Formerly, string matching algorithms were used to match the signature attacks. There is an increasing obstacle in network attacks that has possessed the society of research to investigate a best string matching or signature representation. In spite of this a large research community suggests the regular expression as a dominant signature representation. Regular expression consists of a character sets that identify a search pattern. Regular expressions are grammars that denote the regular language. Regular expression matching is a traditional problem of computer science and technology. The authors in [37 - 39] have made productive developments to promote the research of regular expression in algorithms and theories. There are mainly two primary requirements that must be satisfied for any regular expression representations. They are time efficiency and space efficiency. Space efficiency specifies the size of the system representation and it must be less so that it guarantees that it fits inside the main memory of NIDS. Time efficiency specifies the amount of time that is required by the NIDS to process every byte of network traffic and it must be little so as to permit a large degree of traffic to match rapidly.

When compared with the simple string patterns regular expressions are considered to be very expressive and hence they are capable to represent an ample collection of payload signatures [25]. However to implement regular expressions need greater memory space and bandwidth. On the other hand the crucial task with these extremely fast regular expressions is to trim down the usage of memory and its bandwidth.

Regular expressions are usually evaluated by finite automaton which is a mathematical framework of a system that comprises of inputs and outputs. The system initially begins at the start state and it can be in any one of the finite states. Based on the previous input characters read the state of the system understands the systems behavior for the subsequent input string. The finite automata can be categorized into Non Deterministic Finite Automata (NFA) and Deterministic Finite Automata (DFA) depending on the prime technology and current resources. The foremost dissimilarity among NFA and DFA is that for each character that is read in packet payload NFA can have multiple state transitions while DFA can have only one state transition. Owing to this NFA has a time complexity of O(m) where m is the number of states while DFA requires a large amount of memory for the same packet payload.

A significant team of research work has been concentrated on compression strategies which aim towards decreasing the memory space that are required to represent DFAs. The set of regular expression when compiled to a single DFA frequently leads to state blowup problem with an enormous or even to impractical memory consumption. One way to alleviate this difficulty is to share out the collection of regular expression into many groups and to build independent DFAs for each group. Intelligent Optimization Grouping Algorithms (IOGA) [12] can be utilized effectively to overcome the issue of state blow up problem by obtaining the comprehensive deal among the number of groups and utilization of memory. One such way to solve the state blow – up problem and to provide an efficient finite automaton is to diminish the number of DFA states. In the rest of the paper the various existing compression techniques that are used to reduce the DFA states are analyzed and the ways through which Intelligent Optimization Grouping Algorithms can be efficiently used to solve the state explosion problem are discussed.

The remainder of the manuscript is structured as follows. Section II deliberates about the regular expression model and types of regular expressions. Section III discusses and evaluates the various state compression techniques that are used to reduce the DFA states and section IV discusses about grouping the regular expression using Intelligent Optimization Grouping Algorithms and section V delivers the concluding remarks.

2. THE REGULAR EXPRESSION MODEL

Pattern Matching is a technique of finding a string in a text based on a specific search pattern. The search pattern can be effectively described using regular expression. Thus regular expression matching plays a vital role in pattern matching. To better understand the regular expression matching in network intrusion detection system the different types of regular expression representation and its characteristics has to be studied. The most widely used NIDS open source tool is Snort rule set thus in this paper some of the important types of regular expression that are used frequently in Snort [22] rule sets are discussed. In the following section, the various types of regular expression and its characteristics are discussed in the way their complexities are mounted.

2.1 Exact-match strings.

Exact-match strings are the simplest patterns that are mostly found in the rule set. The size of patterns in an exact match string is fixed and it occurs in the input text exactly as it is appeared. The rule set which contains exact match strings exposes two vital properties. The first vital property is that DFA based solutions using Aho-Corasick algorithm [15] or the Boyer-Moore algorithm [35] can be efficiently utilized given that their size depends on the number of characters that are present in the pattern set. Secondly, optimization that depends on the hashing schemes [27, 34] can be used for a maximum length pattern size and it does not measure for arbitrarily long strings. When analyzing the properties the exact match string algorithm is not so expressive and if an assaulter appends padding in the regular expression then it can't identify malicious packets. However, the advantage of the exact match string regular expression is that it is easy to implement and can accomplish a high matching speed when compared with other types of regular expressions.

2.2 Character sets and simple wildcards.

Character sets and simple wildcard regular expression are basically found in two structures either

as [s_1-$s_j s_k s_l$] expressions, or as \s, \d, \a, \S, \D, \A. In the first structure the set incorporates all characters between s_1 and s_j, s_k and s_1 and in the second structure the set comprises of all space characters (\s), all digits (\d), all alphanumerical characters (\a), and their complements (\S, \D, \A). A wildcard is represented through a non-escaped dot and these sub-patterns represent a set of exact-match strings. As a rule, character sets and wildcards do not permit for immediate utilization of the Aho-Corasick algorithm [15] or the Boyer-Moore algorithm [35] and of hashing schemes [27, 34]. Regardless, in spending the time and cost for mounting the pattern set size it is better to perform a thorough enumeration of the exact match strings and to produce a less complicated case that do not disrupt the properties of exact match strings.

2.3 Simple Character Repetitions.

The next type of regular expression is the simple character repetition which looks like the ch^+ and ch^* structure, where ch is any character of the alphabet. It does not surpass the number of characters in the pattern set and maintains the same size of the DFA. However, it is impractical to permit in-depth details of exact string match to reduce the regular expression because there are an unlimited number of such strings. Therefore, hashing techniques such as [34] and [27] are not applicable. However, in a finite automaton hashing schemes are employed as a loop transition.

2.4 Character sets and wildcard repetitions.

In character sets and wildcard repetitions the various regular expression are compiled into a single DFA providing a memory blast in the size of DFA [5]. Thus it provides an additional complexity. Subsequently, hashing techniques cannot be applied to this problem and also a single DFA cannot be a possible solution. An obtainable solution is to group rules into multiple rules and form parallel DFAs [1]. This technique might reduce the consumption of memory but result in increased memory bandwidth. Precisely N number of DFAs depends on an N-fold growth in the memory bandwidth. NFA can be used as an alternative by exchanging off the utilization of memory with the requirements of memory bandwidth.

2.5 Counting Constraints.

Counting constraints implies the combination of simple character repetition and character sets and wildcards repetitions. The upper bound of counting constraints might or might not be constrained. As seen in the above implications a simple character repetitions having a constrained upper bound has a potential to do an in-depth enumeration of the exact match strings. As mentioned in [1,5] a single regular expression when converted into NFA and then to DFA can prompt to exponential state blow up. DFA techniques are impractical to design with counting constraints. Thus the counting constraints with bounded repetition are desirable to replace with unbounded repetition.

3. LITERATURE SURVEY

Deep packet inspection processes the complete packet payload and identifies a set of predefined patterns. In recent years, contemporary systems replace set of strings with regular expressions, because of their higher flexibility and expressive power. To make a pattern matching process fast and memory competent, many DFA compression techniques are carried out. In this section the merits and demerits of the various deterministic finite automata compression techniques and their performances are discussed.

3.1 Deterministic Finite Automata (DFA)

A DFA consists of five tuples ($Q, \Sigma, \delta, q_0, F$) where Q represents the set of finite states, Σ denotes the finite set of input alphabets, δ the transition function which takes a state and an input character as parameters and returns a state, q_0 denotes the start state and F represents the set of accepting states. In case of networking applications Σ contains 2^8 symbols from an extensive ASCII code. A primary characteristic of DFA is that only one state can be active at a time. It does not have multiple state transitions.

However it is infeasible to build a regular expression for the most repeatedly used rule set. Especially when the regular expression contains repeated wildcards it becomes difficult to build a DFA which contains a minimum number of states [24]. It takes only one main memory accesses per byte.

A hypothetical study was done and the worst case scenario [3] illustrated on the study shows that a single regular expression of size m is represented as a NFA with a complexity of O(m) states. The same expression when transformed into a DFA generates $O(\sum^m)$ states. In a DFA the processing complexity for every input character is O(1) however when all the m

states are active at the same time the complexity of NFA is $O(m^2)$.

Fang Yu et al, 2006 [1] proposed a DFA - based implementation called multiple DFAs (MDFA). It is an alternative DFA representing a set of regular expressions. The input string is compared against an MDFA by simulating every constituent DFA to determine whether there is a match or not. When compared with DFA, MDFAs are more compact because there is over a multiplicative raise in the number of states. Since all the elements of DFAs are matched against the input string the matching speed of MDFAs are slower than that of the DFAs. The regular expression matching speed of MDFA is about 50 to 700 instances higher than that of the NFA - based implementation [2] and they are mainly used in the Linux L7-filter [13], Bro [23] and Snort system [22]. On a DFA-based parser it achieves 12- 42 times speedup. The speed of pattern matching is almost at a gigabit rates for certain pattern sets.

Todd J. Green et al, 2004 [16] constructed lazy DFA in which the finite states, finite inputs and state transitions are equivalent to NFA at runtime, but they cannot be considered the same at compile time. In the lazy DFA the states and transitions form a subset of the standard DFA and they are much smaller than that of the standard DFA. The drawbacks of this technique are it leads to a high warm-up cost and large memory consumption.

3.2 Non Deterministic Finite Automata (NFA)

The working principle of NFA is same as DFA except that the transition function δ works by transiting to a new state from a state on an input alphabet. In a NFA multiple states can be simultaneously active at a time. The number of states in NFA that are essential to express a regular expression is equal to the number of alphabets that are required in the generation of regular expression. Therefore, Sidhu et al, 2004 [2] proposed a NFA based approach which improved the usage of memory. In NFA several states are active in parallel and it has multiple transitions thus it required multiple parallel operations in memory. At the same time all the states in NFA can be active which needs an excessive amount of memory bandwidth.

In [2], Sidhu et al, 2004 were the first to use the NFA to construct regular expressions for the given input string using FPGAs. To match a regular expression of size m, a serial machine requires $O(2^m)$ memory and requires the time complexity of O(1) per input character. However, the authors proposed a method that requires $O(m^2)$ space but process a character of text in O(1) time. Additionally, they presented a simple and fast algorithm that rapidly constructs the NFA for the given regular expression. To construct an NFA rapidly is crucial because the NFA structure depends upon the regular expression, which is known only at runtime.

Liu Yang et al, 2011 [17] developed a novel technique that employed Ordered Binary Decision Diagrams (OBDDs) in order to improve the time-efficiency of NFAs. An OBDD is represented using arbitrary Boolean formulae. In order to increase the competence of state - space exploration algorithms [18] model checkers used OBDDs. NFA-OBDDs were evaluated with three sets of regular expression. The first set comprises of 1503 regular expressions which were obtained from the Snort HTTP signature rule set [10]. The next set contains 2612 regular expressions and the third set contains 98 regular expressions, which were found from the Snort HTTP and FTP signature rule sets. NFA-OBDDs are between 570x–1645x faster when compared with NFAs and uses almost the same amount of memory as that of NFA. NFA-OBDDs improved the efficiency of time of NFAs without conceding their efficiency of memory.

3.3 Delayed Input DFA (D^2FA)

Sailesh Kumar et al, 2006 [29] constructed D^2FA by converting a DFA by means of incrementally substituting many state transitions with a single default transition. The D^2FA is represented by a directed graph, whose nodes are termed as states and whose edges are termed as transitions. Transitions perform a move to a new state based on the present state and the character that is read from a finite set of input alphabet Σ. Each state has not more than one unlabeled active transition known as default transition. There is one start state and for each and every state, a set of matching patterns is defined.

The authors conducted test on the regular expression obtained from Cisco Systems, Snort rule sets [22] , Bro NIDS rule sets [23], and in the Linux layer-7 filter [13] application protocol classifier. From these regular expression sets, DFAs were constructed with a small number of states and the set splitting techniques proposed by Yu et al, 2005 in [31] were applied. The regular expressions were divided into different sets so that every set created a small DFA. Then from the Cisco regular expressions 10 sets of rules were created and the footprints of total memory were reduced to 92 MB, with an

aggregate of 180138 states, and lesser than 64K states were obtained from every individual DFA. Then the Linux layer-7 expressions were split into three sets, and it obtained a total of 28889 states. Further the Snort set consisted of 22 complex expressions ware split further into four sets and the state was unpredictable. The regular expressions found from Bro rule set were simple and efficient therefore they compiled all of them into a single automaton.

This approach drastically decreased the number of distinct transitions among states. For a set of regular expressions drawn from current business and academic systems, a D^2FA representation reduced state transitions by more than 95%. For instance, using D^2FA, the space requirements used in deep packet inspection appliances of Cisco Systems were reduced to less than 2 MB. Unluckily the use of default transition decreased throughput as there was no use of input in default transition and memory has to be accessed to retrieve the next state.

3.4 Content Addressed Delayed Input DFA (CD^2FA)

S Kumar et al, 2006 [4] designed the Content Addressed Delayed Input DFA (CD^2FA), that matches the throughput of the conventional uncompressed DFAs. In a conventional uncompressed DFA implementation the numbers are represented as states and the characteristic information of the given state are found out using the number specified in the table entry. The main function of CD^2FA is that the state identifiers are replaced with content labels which specify the small portion of data that are stored in the table entry. The default transition that matches the present input characters are skipped using content labels. The table entry for the next state are found out using content label by means of hashing techniques.

A CD^2FA deals with the consecutive states of a D^2FA utilizing the content labels. This process provides the chosen information that is available in the state traversal approach and avoids unnecessary memory accesses. The number of main memory accesses required by CD^2FA is equal to those required by an uncompressed DFA. Because of the lower memory footprint and high cache hit rate the throughput of uncompressed DFAs is improved. With an unassuming 1 KB data cache, CD^2FA attains two times higher throughput than that of an uncompressed DFA and in the meantime only 10% of the memory is needed by table compressed DFA. Subsequently, the regular expressions are implemented by CD^2FA very economically and the throughput and scalability of the system is enhanced.

The effectiveness of a CD^2FA is evaluated experimentally on the regular expression sets from Cisco Systems, which contains more than 750 reasonably complex regular expressions in the Snort rule sets [22] and Bro NIDS rules sets [23], and in the Linux layer-7 [13] application protocol classifier. The authors created Cisco rules of ten sets with a total of 180138 states, and the number of states of each DFA is less than 64000 states. Then the Linux expressions were split into three sets with a total of 28889 states. Snort rules were divided into four sets which contains 22 regular expressions. Bro NIDS regular expressions were not divided because they are very simple. CD^2FA constructed from the Creation Reduction Optimization (CRO) algorithm [4] achieved a memory reduction of around 2.5 to 20 times higher. The memory utilization reductions of CD^2FA are 5 to 60 times higher than that of an uncompressed DFA

3.5 Hybrid Finite Automata (HFA)

M. Becchi and P. Crowley, 2007 [5] have introduced hybrid DFA-NFA state reduction solution. A hybrid DFA-NFA solution combines the strengths of NFA and DFA. When the automaton is constructed, NFA encoding is done on any node that contributes toward state blowup, while the rest of the states are converted into DFA nodes. The end result incorporates the memory utilization of NFA, and integrates the memory bandwidth requirements of a DFA. The size of the automaton is maintained by intruding the subset construction operation of NFA states that takes place when converting NFA to DFA and the growth causes state explosion. The critical states are easily determined by doing the above case.

The subset construction operation is intruded with an intermediate state that results in a hybrid automaton which contains DFA-like states, NFA-like states which are not expanded and the border state. The border states are considered to be a part of both a DFA and an NFA. Some of the useful properties of Hybrid FA are that the DFA - state is the start state; the NFA part of the automaton remains inactive till a border state is reached; and there is no backward activation of the DFA coming from the NFA.

The key factor is that the hybrid finite automaton is the first automaton that evaluates all the types of regular expression found in Snort NIDS rule set [22] and is implemented efficiently in real-world rapid systems. The hybrid finite automata uses default transitions [31, 36] and content addressing [4]

to encode the system and this leads to a variation in the storage requirements from 21KB up to 3MB. In reality, the default transition technique used in hybrid automaton eliminates approximately 98-99% of the DFA transitions, while the content addressing method implies the usage of state identifiers wide by 64 bit.

The main uniqueness of a hybrid finite automata are that it provides an unassuming memory storage requirement that is equivalent to a NFA solution, the memory bandwidth requirement of HFA in average case is also same as that of a single DFA solution, and in worst case it is linear containing dot-star condition and counting constraints.

To balance memory and throughput, a new method Deep Classification – DFA (DC-DFA) was proposed by Wei et al, 2013 [14]. DC-DFA is a compact representation that is based on hybrid finite automata which combines the advantages of NFA and DFA. It is supported mainly for large scale regular expression matching. GradeOne classification approach is used to reduce the memory usage of DC-DFA and uses deep classification approach to improve the throughput of DC-DFA. The experiments evaluated on DC-DFA shows that in case of very large state explosion, DC-DFA reduces DFA states by 75% and improves the utilization of memory more efficiently and maintains high system throughput.

3.6 History Based Finite Automata (H-FA)

When multiple partially matching signatures are present in the DFA, the system becomes inefficient and yields to the state blow-up problem. To overcome this scenario the authors S Kumar et al, 2007 [6] proposed an improved Finite State Machine. The approach builds a machine which retains a lot of information, and stores the data in a small and high-speed cache memory known as history buffer [6]. This type of system is named as History-based Finite Automaton (H-FA) which reduces space up to 95%.

Every transition is associated with a condition that depends upon the associated action and state of the history which decides whether to insert or delete the state from the history set, or both. H-FA is thus represented as a 6-tuple $H = (S, s_0, \Sigma, A, \delta, H)$, where S represents the finite set of states, s_0 denotes the start state, Σ specifies the input alphabet, A represents the set of accept states, the transition function δ, and H represents the history. The transition function δ functions by taking in an input alphabet, a state, and a history state as its arguments and returns a new state and a new history state.

$$\delta: S \times \Sigma \times H \rightarrow S \times H$$

The history buffer enhances the implementation of the H-FA and its automaton is similar to that of a DFA and contains set of states and transitions. For a single character there can be multiple transitions and leaves from a state but during execution only one of these transitions is taken, and that is resolved after investigating the details of the history buffer.

The performance was evaluated and experiments were conducted on the regular expressions used in the Cisco Systems. The rule sets from Cisco Systems contains over 750 reasonably complex regular expressions. The regular-expression signatures used in the open source Snort NIDS rule set [22] , Bro NIDS rule set [23], and in the Linux layer-7 [13] application protocol classifier were also considered. Linux layer-7 protocol classifier contains seventy rules and a Snort rule set contains more than 1500 regular-expressions. The Bro NIDS contains 648 regular-expressions and the results for the HTTP signatures were present.

The number of conditional transitions is very small and causes state blow-up. The outgoing transitions of a DFA are around 256 and in most of the H-FAs there are less than 500. Hence the number of transitions increases nearly by double and there is a decrease in the number of states and conversely there is a significant reduction in memory. The size of H-FA that is registered in history buffer depends upon the partial matches. But limitation of this approach is that it has a restricted number of transitions for each input character with a huge size of transition table and a slow inspection speed.

3.7 History Based Counting Finite Automata (H-cFA)

When there is a length restriction of l on a sub expression of a given regular expression, the number of states that is needed by the sub expression gets multiplied by l. S Kumar et al, 2007 [6] designed a machine called as H-cFA which can count such events thereby avoiding state explosion.

In H-cFA the length restriction is replaced with a closure and the closure is represented by a flag that is present in the history buffer. A counter is added for every flag in the history buffer. The flag is set by setting the counter to the length restriction rate by the conditional transitions while the flag is reset by resetting these transitions. Besides, the flag which is set are attached with the counter value 0 which denotes an additional condition. During the execution

of the machine, for every input character the value of every single positive counter is decremented.

This basic change is to a great degree compelling in reducing the number of states, particularly when long length restrictions strings are present. H-cFA is exceptionally effective in implementation of the Snort signatures because it contains many long length restriction strings. It is very effective in reducing the memory consumption. If there is no use in the counting capability of H-cFA there is a massive memory blowup in the composite automaton for Snort prefixes.

3.8 Extended Finite Automata (XFA)

Randy Smith et al, 2008 [7] designed a state based Extended Finite Automata (XFAs) which is augmented with a finite set of auxiliary variables in the standard DFA which is used to recollect different sorts of information that is relevant to the signature matching and to collect the explicit instructions that are attached to states in order to update these auxiliary variables. A state based extended finite automaton [7] is a 7-tuple $(S, V, \Sigma, \delta, U, (s_0, v_0), A)$, where

- S represents the finite set of states,
- Σ represents the set of input alphabets,
- $\delta : S \times \Sigma \rightarrow S$ is the transition function,
- V represents the finite set of variables,
- $U : S \times V \rightarrow V$ is the update function which describes how the data value is updated on states,
- (s_0, v_0) is the initial configuration which represents a start state s_0 and an initial variable value v_0,
- $A \rightarrow S \times V$ is the set of accepting configurations.

XFAs is a simplified version of standard DFAs which includes a finite set of possible variable values and are attached to states that operates with the variable during matching. Variable values along with a state are generalized to each of the initial states, transient state and accept state. In particular, individual XFAs are constructed and they are combined by means of standard techniques.

Randy Smith et al, 2008 [10] have also proposed an edge based XFA. This work gives an informal categorization to the state blow up problem and is focused on algorithms to build XFAs from regular expressions. Semantically, edge based XFAs are equal to state-based XFAs, but a lot of states are required for state-based XFAs. Conversely, state based XFAs provide an efficient result for matching, combination and optimization algorithms.

For the test set Snort signature set were used which were obtained in March 2007. Randy Smith et al, 2008 [10] collected at different time interval live traffic traces at the edge of the network, and each trace contained HTTP packets between 17,000 and 86,000. The performances were measured with the count of CPU cycles for each payload that are leveled to seconds per gigabyte (s/GB). The performance was evaluated by carrying the experiments on a standard Pentium 4 Linux workstation that runs at three GHz with three gigabyte of memory. The time complexity of Edge based XFAs is similar to DFAs and the space complexity is just like NFAs. When compared to DFA based system XFAs use 10 instances less memory and accomplish 20 instances higher matching speeds.

Michela Becchi et al, 2008 [19] proficiently handled counting constraints and back-references and proposed an advanced automation. This type of automaton covers all the patterns from the most expressive and popular Snort NIDS rule-set [22].

When the regular expressions are represented with counting constraints in DFA form there is a huge rise in memory space. When there is an increase in the number of repetitions it is infeasible to design DFA. To solve the issues in [19] the authors have introduced the idea of the counting automaton. The automaton designed with counting constraints aims to minimize the consumption of memory and bandwidth requirements. In particular, XFA size does not depend on the number of repetitions, the main memory access count that is required for each counter and does not depend on the number of active counter instances. The value of the induced alphabet becomes larger and secondly, there is an excessive increase in the size of the DFA. To solve this in [19] the authors have proposed Extended Hybrid FA which compiles several regular expressions into a single automaton.

The experimental results were evaluated on the Bro v0.9 rule set [23] and Snort [22] rule sets. First, they were able to compile a large number of complex regular expressions which contains simple regular expression with repeated character values, disjunctions of sub patterns, dot-star terms, and counting constraints and back references. Second, there was a decrease in the size of the NFA. Third, there was a reduction in the memory bandwidth in converted hybrid-FA representation and there was a need of an extra 156KB-16MB to hold the head-DFAs. The limited memory utilization makes a way to deploy the automata with static Random Access

Memory (SRAM) in an Application Specific Integrated Circuit (ASIC) implementation that allows an excess memory access rate of 500MHz.

A XFA has used number of automata alterations to eliminate restricted transitions which is limitation of HFA. XFA is confined to single supplementary state for each regular expression and it is unsuitable for tricky regular expressions.

3.9 Delta Finite Automata (δFA)

D Ficara et al, 2008 [8] proposed a compressed DFA called as Delta Finite Automata. The interpretations that were obtained from the above techniques are that most default transitions stay close to the start state and a state that is defined by its transition set represents the accepted rule and for a given input character most of the transitions are directed to the same state. Based on these interpretations the δFA was designed.

The last interpretations state that most states that are adjacent contribute a considerable portion of the same transitions and hence it is sufficient to store the difference between these adjacent states. Therefore a transition set of the current state is been preserved and stored in a table which represents the supplementary structure. The number of states and transitions used by the algorithm is reduced and the study shows that nearly all adjacent states share a few common transitions and it is sufficient to store only differences between them. Essential characteristic of the delta finite automata is that it required only a single state transition for each character, thus allowed a fast string matching.

In a δFA, an arbitrary number of transitions are obtainable and therefore each state does not have a stable size and consequently there is a necessity in state pointers, which are normally standard memory addresses. Char - State compression technique [8] based on input characters was proposed which exploited the relationship of few input characters with many states which reduced the number of bits required for each state pointer. This compression scheme has been included into the delta finite automata algorithm which provided a reduction in memory with an insignificant rise in the state lookup time.

Domenico Ficara et al, 2011 [20] proposed a compact representation that was an extension of the work [8] which deletes most of the neighboring states that share the common transitions and keeps only the different ones. Instead of specifying the transition set of a state concerning its direct parents, this requirement can be relaxed to obtain the adoption of 1-step ancestors which increases the chances of compression. The finest method to exploit the N^{th}-order dependence is to describe the state transitions among child and ancestors as impermanent. This, during the construction problem leads to NP-Complete problem. Therefore, to make it simpler a direct and negligent approach is chosen. The real rule-sets result shows that the there is no much difference between the simple approach and from the optimal construction. This technique shares the same property of many other existing approaches and they are orthogonal to the various discussed existing algorithms such as XFAs [7] and H-cFA [6] and allows for higher compression rates.

3.10 Second Order Delta Finite Automata (δ^2FA)

δ^2FA is an extended version of δFA. An as alternative of specifying the state transition set relating to its direct parents, there is an increased probability of compression with the acceptance of 2-step ancestor's. Before proceeding with the construction process of δ^2FA [9] δFA has to be constructed and that value should be used as input. The subsets of nodes are considered in which a transition for a given character are defined temporarily.

In a δ^2FA the table lookup is not similar to that of δFA. The main difference between δ^2FA and δFA is that there is an anxiety about the temporary transitions and the temporary transitions are not stored in the local transition set. Therefore, the lookup time complexity of δ^2FA is almost same as that of a δFA and memory consumption is better than δFA. δ^2FA takes advantage of the 2^{nd} order precedence among states and by implementing the concept of temporary transition it reduces the number of transitions. Only a single state transition per character is required by δ^2FA thus it allows for fast string matching and higher compression rates.

3.11 Dual Finite Automata (dual FA)

Cong Liu et al, 2013 [21] proposed a new approach called as dual finite automata (dual FA). The dual FA consists of an Extended Deterministic Finite Automaton (EDFA) and a Linear Finite Automaton (LFA). Dual FA consumes only a smaller memory when compared to DFA and the number of main memory access is very low when compared against the various discussed existing compressed DFAs. For instance it needs one or two main memory

access for every byte in the payload. This is because by using linear finite automata the dual FA efficiently controls unbounded repetitions of wildcards and character sets.

This technique mitigates the state blow up problem. First, the NFA states that are not dependent to a large number of other states and those states that cause state exploitation are identified. Then, these NFA states are implemented using linear automaton. Subsequently the rest of the NFA states are compiled into a single extended DFA, which reduces the NFA states. Finally, by considering the fact that these two mechanisms cannot work separately, an interaction mechanism is implemented. EDFA has an additional feature compared with DFA to support the interaction mechanism.

The experimental results evaluated on dual finite automata demonstrate that LFA is very efficient in dropping the number of states and transitions. The number of states is reduced for up to four orders of magnitude when compared with that of DFA and the number of transitions is reduced for two orders of magnitude in contrast with MDFA [1]. In dual FA there is only a rare increase in the number of main memory accesses, but in a MDFA there is a rapid increase in the number of main memory accesses as the number of DFAs increases.

Lastly one of the limitations of dual FA is that the number of LFA states cannot be large. When the dual finite automaton is implemented in personal computer, the effects in large number of LFA states considerably have lot of computational overhead. When large number of LFA states is existed in dual FA a larger per-flow state occurs, the storage size of the transition table becomes large and memory bandwidth also becomes large. The dual FA offers an effective solution among memory storage and memory bandwidth, and the implementation becomes very easy. When compared with DFA and MDFA the simulation results shows that in dual FA there is a drastic decrease in the storage demand and the memory bandwidth is almost close to that of DFA.

3.12 Deterministic Finite Automata with Extended Character Sets (DFA/EC)

Cong Liu et al, 2014 [11] have proposed a novel approach called Deterministic Finite Automata with Extended Character Sets(DFA/EC) which doubles the size of the character set [11,32] and considerably reduces the number of states. The DFA/EC can be efficiently implemented by dividing the design into two parts. The first part comprises of a compact DFA with a size m, which requires only one main memory access in its transition table for every byte in the packet payload. The second part consists of an efficient complementary program does not require any main memory access because it runs in the main memory without using the table lookup.

When compared with the above discussed existing compression techniques, the inspection speed of DFA/ EC is increased significantly by assigning the minimum value of one to the number of main memory accesses. The size of the inspection programs that are stored completely in the cache memory is kept small. Cong Liu et al, 2014 [11] conducted experimental results and the inspection program's speed was deliberated with C++ and JAVA implementations in a Unix machine with 16 Gigabyte of 1333 MHz DDR3 memory and with a 2.66 GHz Intel Core i5 CPU. In both C++ and JAVA implementations DFA/EC showed the fastest results, and when compared with DFA, DFA/EC were over ten times faster and were two times faster than MDFA in Java implementation. Thus DFA/EC is efficiently implemented on ASIC hardware or GPUs with less cache memory and more computation resources.

The memory bandwidth requirement of DFA/EC is much lesser than MDFAs and is very close to DFA. When considering the rule-sets exploit-19 and web-misc-28, DFA/EC can dramatically reduce the number of main memory accesses of DFA. When compared to DFA the number of states in a DFA/EC is about four orders of magnitude smaller and when compared to 2DFA it is around two orders of magnitude smaller than a 2DFA, and it is an order of magnitude smaller than a 4DFA, and is almost similar to that of an 8DFA. When compared to DFA the number of transitions of DFA/EC is almost four orders of magnitude smaller, when compared to 2DFA is it around two orders of magnitude smaller, with that of 4DFA it is 3 times smaller and 8DFA is comparable to DFA/EC.

The experiments are evaluated with the Snort rule-sets and the results shows that DFA/ECs are very compact and achieve high inspection speed. Particularly, in best case the DFA/ECs are more than four orders of magnitude lesser than DFAs. When compared with DFA, DFA/ECs require significantly lesser memory bandwidth. A DFA/EC is theoretically modest, implementation and upgrading is made easy due to faster construction speed.

4. DISCUSSIONS AND PROPOSED APPROACH

The various existing DFA compression techniques discussed in section 3 was analyzed to improve the memory consumption and to provide an efficient finite automata. The main reason that was analyzed from these existing DFA compression techniques for the number of states to get increased is due to the state explosion problem. The problem with exponential state explosion can be efficiently alleviated by grouping the regular expression. Grouping the regular expression falls into two cases. The first one is when the number of groups is known, the number of states in DFA can be minimized and second is when the maximum number of DFA states is known, the number of groups can be minimized [12]. Though total number of groups and total number of DFA states plays major criteria in minimizing the memory, only either of these two cases cannot be concentrated in minimization process. Minimizing the number of groups will lead to state explosion and minimizing only the number of states will end up with large number of subdivided group count. Therefore to analyze the performance of the regular expression grouping method both these cases should be equally focused in correspondent with the various practical demands.

Grouping the regular expression can be done by Intelligent Optimization Grouping Algorithms. To provide memory efficient deterministic finite automata, DFA compression techniques can be used along with the Intelligent Optimization Grouping Algorithms. Intelligent Optimization Grouping Algorithms such as Tabu Search (TS) [42], Simulated Annealing (SA) [41], Ant Colony Optimization (ACO) [12, 40], Swarm Intelligence (SI) [30] and Particle Swarm Optimization (PSO) [26, 30, 33] can be used effectively to solve the state blow-up problem by obtaining the overall most favorable distribution between the consumption of memory and number of groups. By recursively analyzing each feasible optimization outcome an exact optimum solution can be effortlessly acquired.

In Fig.1 the overall structure of the proposed approach which reduces DFA states by grouping the regular expression using the Intelligent Optimization Grouping Algorithms is shown. The payload files extracted from the rule sets such as Snort [22], Bro NIDS [23], and Linux L-7 filter [13] are used as input. The set of regular expressions are determined using packet payload files. Initially the parameters for the algorithms are assigned and the initial population is generated by randomly distributing the regular expression on the search space. The performance is evaluated according to the Intelligent Optimization Grouping Algorithms such as Simulated Annealing (SA) [41], Swarm Intelligence (SI) [30], Ant Colony Optimization (ACO) [12, 40], Particle Swarm Optimization (PSO) [26, 30, 33], Tabu Search [42], etc. Based on the performance the parameters such as search space, population, position, velocity etc are adjusted and the process is continued until the optimal groups are formed or until the maximum iteration is obtained.

Once the optimal solution is obtained and the regular expressions are grouped, the finite state automata is designed based on the discussed existing DFA compression technique and is integrated with the DPI search engine to identify the packets that hold the viruses, unauthorized access and attacks such as TCP connection attacks, fragmentation attacks and application attacks. Intelligent Optimization Grouping Algorithms applied on the discussed existing DFA compression techniques for deep packet inspection will provide memory efficient automata with an improved network intrusion detection throughput through the use of DPI techniques and improved malicious packet detection.

The experiments are evaluated in future on the proposed approach for the various performance metrics, and the expected outcome is compared with the various discussed existing DFA Compression techniques. Fig.2 shows the memory consumption, memory bandwidth, throughput and compression rate for the different DFA compression techniques. It depicts that the proposed approach will produce reduced memory consumption, better memory bandwidth, high throughput and better compression rate.

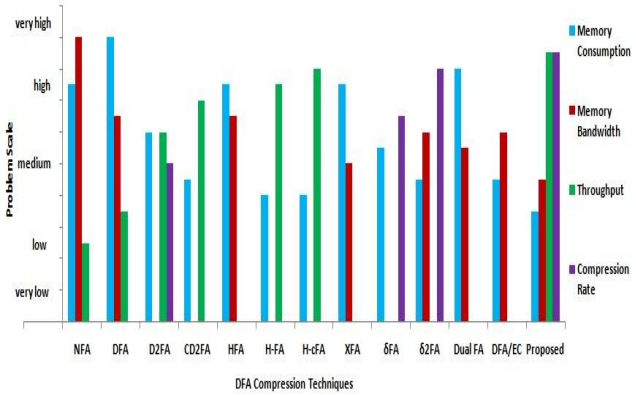

Fig. 2. Memory Consumption, Memory Bandwidth, Throughput and Compression Rate for different DFA compression techniques

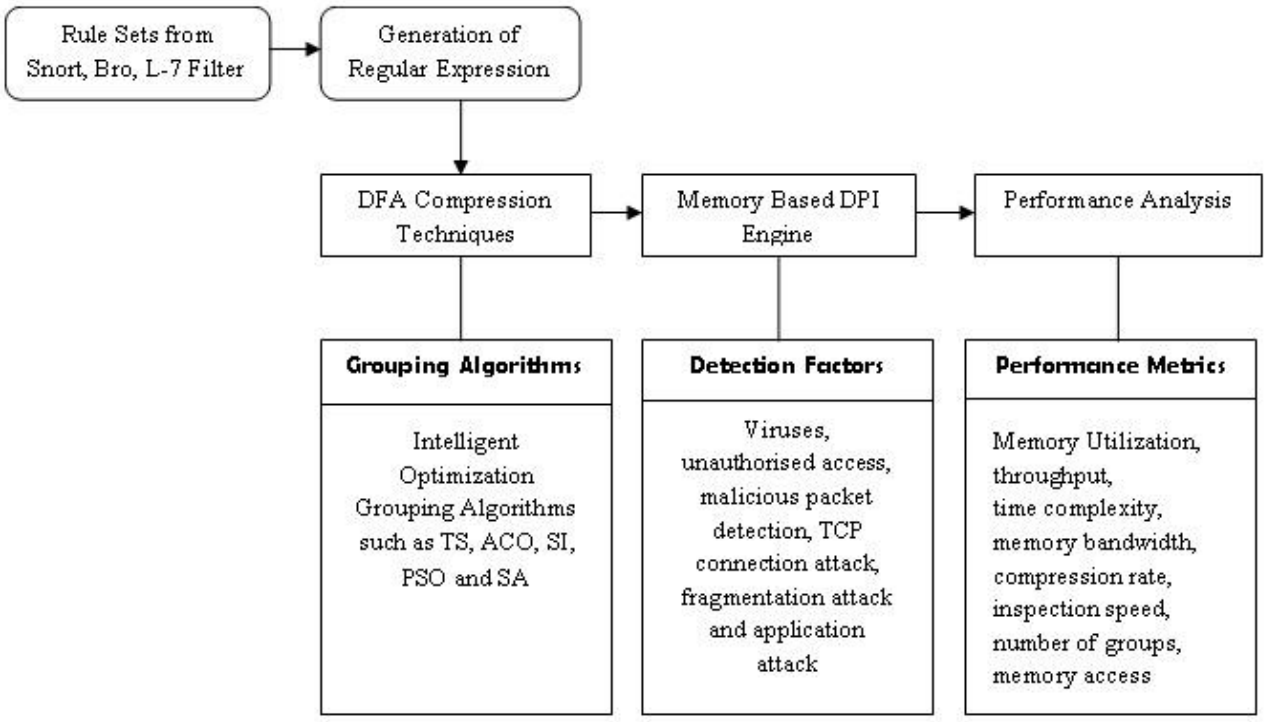

Fig. 1. Overall Structure of proposed approach

Fig.3 illustrates the performance measures of the main memory access time and the time complexity for the various DFA compression techniques and shows that the proposed approach will produce increased number of memory access time per input byte and will improve the time complexity.

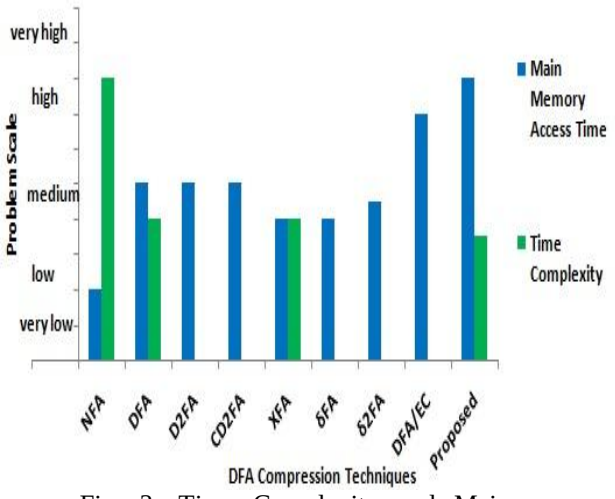

Fig. 3. Time Complexity and Main memory access time for different DFA compression techniques

Fig. 4 shows the inspection speed of intrusion detection and the regular expression matching speed for the various DFA compression techniques. It illustrates that the proposed approach will produce fast matching speed and high inspection speed of intrusion detection.

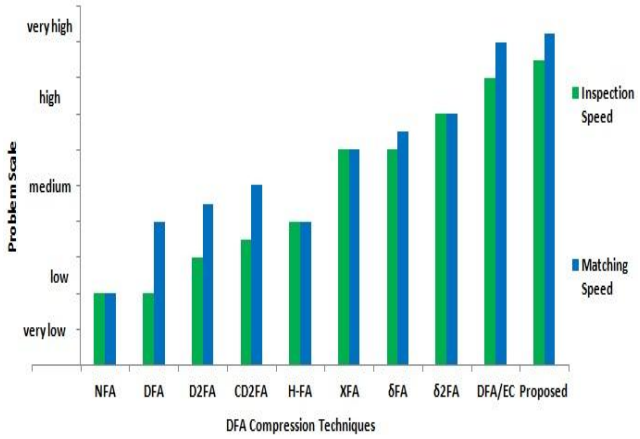

Fig. 4. Inspection Speed and Matching Speed for different DFA compression techniques

Thus the future experiments evaluated on the proposed approach using Intelligent Optimization Grouping Algorithms will provide an optimally efficient automaton when compared with the discussed existing DFA compression techniques and will also improve the throughput of network intrusion detection through the use of DPI techniques and will enhance the malicious packet detection.

5. CONCLUSION

In this paper, the different compression representations for Deterministic Finite Automata such as NFA, DFA, MDFA, Lazy DFA, NFA-OBDD, HFA, H-FA, H-cFA, XFA, D^2FA, CD^2FA, δFA, δ^2FA, Dual Finite Automata and DFA/EC are presented. MDFA increases the matching speed of the regular expression approximately to 50 to 700 times above the NFA-based implementation and achieves the speedup of up to 12-42 times over a DFA-based parser. For all practical applications the size of the lazy DFA remains little but the limitation is that it leads to a high warm-up cost and large memory consumption. NFA-OBDDs improve the time efficiency of NFA. The memory storage requirement of HFA is comparable to those of an NFA; its memory bandwidth is similar to that of a DFA, but the regular expressions that contains counting and dot-star conditions consumes high memory. H-FA reduces space close to 95% but has a vast size of transition table and a slow inspection speed. On the other hand H-cFA is extremely efficient in implementing long length restriction signature patterns. XFAs matching speed is around 20 times higher than a DFA and consumes 10 times lesser memory than DFA. The δFA substantially diminishes the number of transitions and number of states and needs only a single state transition for each character thus providing fast string matching. When compared to δFA, δ^2FA provides an effective improvement in memory utilization and lookup speed. D^2FA representation reduces the transitions between states by more than 95% and decreases the space requirements to less than 2 MB but the usage of default transitions decreases throughput. Memory reduction achieved by CD^2FA is between 2.5 to 20 times better when compared to a compressed DFA and 5 to 60 times higher when compared with uncompressed DFA. The number of main memory access of Dual FA is much quicker than the other existing techniques. When compared to other existing techniques DFA/EC tremendously increases the data packet inspection speed and provides only one main memory access. Each of the state compression techniques that were studied has certain strengths and limitations. Thus any one of these compact representation DFAs can be used along with Intelligent Optimization Grouping Algorithms to provide memory efficient deterministic finite automata that can be used for deep packet inspection.

REFERENCES

1. Yu, F., Chen, Z., Diao, Y., Lakshman, T.V., Katz, R.H.: *Fast and Memory-Efficient Regular Expression Matching for Deep Packet Inspection*. In: Proceedings of the ACM/IEEE Symposium Architecture for Networking and Communication Systems, 2001, pp. 93-102.

2. Sidhu,R., Prasanna,V.K.: *Fast Regular Expression Matching using FPGAs.* In: Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2001, pp. 227-238.

3. Hopcroft,J.E., Motwani,R. & Ullman,J.D.: *Introduction to Automata Theory, Languages, and Computation* 2nd Ed., Addison-Wesley series in computer science, Addison-Wesley-Longman, 2001.

4. Kumar,S., Turner,J., Williams,J.: *Advanced Algorithms for Fast and Scalable Deep Packet Inspection*. In: Proceedings of the ACM/IEEE Symposium Architecture for Networking and Communication Systems, 2006, pp. 81-92.

5. Becchi,M., Crowley,P.: *A Hybrid Finite Automaton for Practical Deep Packet Inspection.* In: Proceedings of the ACM Conference on Emerging Networking Experiments and Technologies, 2007

6. Kumar,S., Chandrasekaran,B., Turner,J., Varghese,G.: *Curing Regular Expressions Matching Algorithms from Insomnia, Amnesia and Acalculia.* In: Proceedings of the ACM/IEEE Symposium Architecture for Networking and Communication Systems, 2007, pp. 155-164.

7. Smith,R., Estan,C., Jha,S., Kong,S.: *Deflating the Big Bang: Fast and Scalable Deep Packet Inspection with Extended Finite Automata.* In: Proceedings of the ACM SIGCOMM 2008 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, 2008, pp.207-218.

8. Ficara,D., Giordano,S., Procissi,G., Vitucci,F., Antichi,G., Pietro,A.D.: *An Improved DFA for Fast Regular Expression Matching*. In: Proceedings of the ACM SIGCOMM Computer Communication Review, 2008, No. 38, Issue. 5, pp. 29-40.

9. Antichi,G., Di Pietro, A., Ficara, D., Giordano, S., Procissi, G., Vitucci,F.: *Second-Order Differential*

Encoding of Deterministic Finite Automata. In: Proceedings of the 28th IEEE Conference on Global Telecommunications, 2009, pp. 2838-2843.

10. Smith,R., Estan,C., Jha,S.: *Xfa: Faster Signature Matching with Extended Automata.* IEEE Symposium on Security and Privacy, 2008, pp. 187-201.

11. Cong Liu, Yan Pan, Ai Chen, Jie Wu.: *A DFA with Extended Character-Set for Fast Deep Packet Inspection.* IEEE Transactions on Computers, 2014, No.63, Issue. 8, pp. 1925-1937.

12. Zhe Fu, Kai Wang, Liangwei Cai, Jun Li.: *Intelligent Grouping Algorithms for Regular Expressions in Deep Inspection.* In: Proceedings of the International Conference on Computer Communications and Networks, Shanghai, China 2014.

13. Levandoski,J., Sommer,E., Strait,M.: *Application Layer Packet Classifier for Linux.* Available: http://l7-filter.sourceforge.net/

14. He,W., Guo,Y.F., Hu,H.C.: *Hybrid Finite Automata – Based Algorithm for Large Scale Regular Expression Matching.* Applied Mechanics and Materials, 2013, 263-266, pp.3108-3113.

15. Aho,A.V., Corasick,M.: *Efficient String Matching: An aid to Bibliographic Search.* Communications of the ACM, 1975, No.18, Issue.6, pp. 333-340.

16. Green,T.J., Gupta,A., Miklau,G., Onizuka,M., Suciu,D.: *Processing XML Streams with Deterministic Automata and Stream Indexes.* ACM Transactions on Database System, 2004, No.29, Issue. 4, pp. 752-788.

17. Yang, L., Karim,R., Ganapathy,V., Smith,R.: *Fast Memory-Efficient Regular Expression Matching with NFA-OBDDs.* Computer Networks: The International Journal of Computer and Telecommunications Networking, 2011, 55(15), pp. 3376-3393.

18. Burch,J.R., Clarke,E.M., McMillan,K.L., Dill,D.L., Hwang,J.: *Symbolic Model Checking: 10^{20} states and beyond.* In: Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science, 1990, pp. 428-439.

19. Becchi,M., Crowley,P.: *Extending Finite Automata to Efficiently Match Perl-Compatible Regular Expressions.* In: Proceedings of the ACM International Conference on Emerging Networking Experiments and Technologies, 2008.

20. Domenico Ficara, Andrea Di Pietro, Stefano Giordano, Senior Member, Gregorio Procissi, Fabio Vitucci, Gianni Antichi.: *Differential Encoding of DFAs for Fast Regular Expression Matching.* IEEE/ACM Transactions on Networking, 2011, No. 19, Issue. 3, pp. 683-694.

21. Cong Liu, Jie Wu.: *Fast Deep Packet Inspection with a Dual Finite Automata*. IEEE Transactions on Computers, 2013, No.62, Issue.2, pp. 310-321.

22. Roesch,M.: *Snort: Light Weight Intrusion Detection for Networks.* In: Proceedings of the 13th USENIX Conference on System Administration, 1999, pp. 229-238.

23. Paxson,V.: *Bro: A System for Detecting Network Intruders in Realtime.* Computer Networks: The International Journal of Computer and Telecommunications Networking, 1998, No. 31, pp. 2435-2463.

24. Hopcroft,J.: *An nlogn algorithm for Minimizing States in a Finite Automaton* Stanford University Stanford, CA, USA, 1971.

25. Sommer,R., Paxson,V.: *Enhancing Byte- Level Network Intrusion Detection Signatures with Context.* In: Proceedings of the 10th ACM Conference on Computer and Communications Security, 2003, pp. 262-271.

26. Jun Wang, Xu Hong, Rong-rong Ren, Tai-hang Li: *A Real-time Intrusion Detection System Based on PSO-SVM.* In: Proceedings of the International Workshop on Information Security and Application, 2009

27. Kumar,S., Turner,J.,S., Crowley,P., Mitzenmacher,M.: *HEXA: Compact Data Structures for Faster Packet Processing.* In: Proceedings of the International Conference on Network Protocols, 2007, pp 246-255.

28. Leau Yu Beng, Sureswaran Ramadass, Selvakumar.: *A Survey of Intrusion Alert Correlation*

and its Design Considerations. IETE Technical Review, 2014, No. 31, Issue. 3, pp.233-240.

29. Kumar,S., Dharmapurikar,S., Yu,F., Crowley,P., Turner,J.: *Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection.* In: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, 2006, No. 36, pp. 339-350.

30. Kolias,C., Kambourakis,G., Maragoudakis,M.: *Swarm Intelligence in Intrusion Detection: A Survey.* Computers and Society, 2011, No. 30, Issue. 8, pp 625-642.

31. Fang Yu, Zhifeng Chen, Yanlei Diao, Lakshman,T.,V., Randy H. Katz.: *Fast and Memory- Efficient Regular Expression Matching for Deep Packet Inspection.* In: Proceedings of the ACM/IEEE Symposium on Architecture for Networking and Communications Systems, 2006, pp. 93-102.

32. Liu,C., Chen,A., Wu,D., Wu,J.: *A DFA with Extended Character Set for Fast Deep Packet Inspection.* In: Proceedings of the International Conference on Parallel Processing, 2011, pp. 1-10.

33. Anantathanavit,M., Munlin,M.: *Using K-Means Radius Particle Swarm Optimization for the Travelling Salesman Problem.* IETE Technical Review, 2016, No. 33, Issue. 2, pp. 172-180.

34. Dharmapurikar,S., Lockwood,J.: *Fast and Scalable Pattern Matching for Content Filtering.* In: Proceedings of the ACM Symposium on Architecture for Networking and Communications Systems, 2005, pp.183-192.

35. Boyer,R.,S., Moore,J.,S.: *A Fast String Searching Algorithm.* Communications of the ACM, 1977, No. 20, Issue. 10, pp.762–772.

36. Becchi,M., Crowley,P.: *An Improved Algorithm to Accelerate Regular Expression Evaluation.* In: Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems, 2007, pp. 145-154.

37. Ken Thompson.: *Programming Techniques: Regular Expression Search Algorithm.* Communications of ACM, 1968, No.11, Issue. 6, pp. 419-422.

38. Eugene W. Myers.: *A Four Russians Algorithm for Regular Expression Pattern Matching.* Journal of the ACM, 1992, No.39, Issue. 2, pp. 430-448.

39. Gonzalo Navarro, Mathieu Raffinot.: 2001. *Compact DFA Representation for Fast Regular Expression Search.* In: Proceedings of the 5th International Workshop on Algorithm Engineering, 2001, pp 1-12.

40. Janakiriman,S., Vasudevan,V.: *ACO Based Distributed Intrusion Detection System.* In: Proceedings of the International Journal of Digital Content Technology and its Applications, No.3, Issue. 1, 2009, pp.66-72.

41. Elias D. Nino Ruiz, Henry Nieto Parra, Anangelica Isabel Chinchilla Camarg.: *Evolutionary Algorithm Based on Simulated Annealing for the Multi-Objective Optimization of Combinatorial Problems.* In: Proceedings of the International Journal of Combinatorial Optimization Problems and Informatics, 2013, 4(2), pp. 53-63

42. Wu Jian-guang, Tao Ran & Li Zhi-Yong.: *An improving Tabu Search Algorithm for Intrusion Detection.* In: Proceedings of the 3rd International Conference on Measuring Technology and Mechatronics Automation, 2011, No.1, pp. 435-439.