A Survey on DPI Techniques for Regular Expression Detection in Network Intrusion Detection System

Mr. Girish M. Wandhare[1] Prof. S. N. Gujar[2] Dr. V. M. Thakare [3]
[1]Student [2]proffesor
[2]Department of Information Technology
[1,2]SKNCOE, Pune, Maharashtra, India [3]S.G.B. Amravati University, Amravati, Maharashtra, India

Abstract— Deep Packet Inspection (DPI) is becoming more widely used in virtually all applications or services like Intrusion Detection System (IDS), which operate with or within a network. DPI analyzes all data present in the packet as it passes an inspection to determine the application transported and protocol. Deep packet inspection typically uses regular expression matching as a core operator. Regular expressions (RegExes) are used to flexibly represent complex string patterns in many applications ranging from network intrusion detection and prevention systems (NIDPSs). Regular expressions represent complex string pattern as attack signatures in DPI. It examine whether a packet's payload matches any of a set of predefined regular expressions. There are various techniques developed in DPI for deep packet inspection for regular expression. We survey on these techniques for further improvement in regular expression detection in this paper. In the result we found that it is possible to reduce RegEx transaction memory required in network intrusion detection. We made this survey with possible use of DPI techniques in the wireless network.

Keywords: Deep Packet Inspection(DPI), Regular Expression(RegEx), Deterministic Finite Automata(DFA), LaFA, StriFA, CompactDFA, Tcam, DFA/EC, Snort, Bro

I. INTRODUCTION

In most of the applications REGULAR expressions (RegExes) are used to flexibly represent complex string patterns in many applications, such as network intrusion detection and prevention systems (NIDPSs), Compilers and DNA multiple sequence alignment [1]. Intrusion detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of possible incidents, which are imminent threats of violation of computer security policies and standard security practices. Intrusion detection and prevention systems (IDPS) are primarily focused on identifying possible incidents in packet, logging information about these incidents, attempting to stop and reporting them to security administrators [3].

Deep Packet Inspection (DPI) is a technology that enables the network owner to analyze internet traffic, throughout the network, in real-time and to differentiate them according to their payload [3]. Traditional packet inspection algorithms have been limited to comparing packets to a set of strings. Newer DPI systems, such as Snort [11], and Bro [10], use rule-sets consisting of regular expressions, these systems are more expressive, efficient, and compact in specifying attack signatures [4].

In Network intrusion detection system techniques such as Bro [10] and Snort [11] and Linux Application Level Packet Classifier (L7 filter) [5] use RegExes to represent attack signatures or packet classifiers. Regular Expressions (RegExes) have been widely used in network security applications; their inherent complexity often limits the total number of RegExes that can be detected using a single chip for a reasonable through- put [1]. Regular expressions represent complex string pattern as attack signatures in many applications. There is no current regular expression detection system is capable of supporting large RegEx set ; and even larger RegExp sets are expected in future with high speed demand [1].

The LaFA technology based on three observations, first RegExes consist of a variety of different components such as character classes or repetitions, Second is the order of components in a RegEx is preserved in the state machine detecting this RegEx and Third, most RegExes share similar components [1]. LaFA requires less amount of memory due to these three contributions: 1) providing specialized and optimized detection modules to increase resource utilization; 2) systematically reordering the RegEx detection sequence to reduce the number of concurrent operations; 3) sharing states among automata for different RegExes to reduce resource requirements.

The TCAM technology is the first hardware-based RE matching approach that uses ternary content addressable memory (TCAM), TCAM is has been widely deployed in modern networking devices for tasks such as packet classification. This technique introduces three novel techniques to reduce TCAM space and improve RE matching speed: Transition sharing, table consolidation, and variable striding. This three technique can achieve potential RE matching throughput of 10–19 Gb/s [2].

StriFA [6] technology presents the stride finite automata; it's a novel finite automata family, used to accelerate both string matching and regular expression matching. This technique implemented in software and evaluated based on different traces. The simulation results of StriFA shows that its acceleration scheme offers an increased speed over traditional nondeterministic finite automaton/deterministic finite automaton, the same time reducing the memory requirement. Main focus of this technique in this category is to: reduce the number of active states of the automaton during the matching phase which in turn reduces the number of memory accesses, resulting in an improved matching speed; reduce the memory consumption by reducing the state number or transition number of the automaton [5].

Compact DFA [6] proposed method is to compress DFAs by observing that the name used by common DFA encoding is meaningless. This degree of freedom and encode states in such a way that all transitions to a specific state are represented by a single prefix that defines a set of current states. Compact DFA technique applies to a large class of automata, which can be categorized by simple

properties. With a TCAM [2] the throughput of compact DFA reaches to 10 Gb/s with low power consumption. This technique uses Aho–Corasick (AC) algorithm, which uses a deterministic finite automaton (DFA) to represent the pattern set [6].

Extended Character set DFA [3] focused on reducing the memory storage requirement of DFAs, and it can be divided into the following categories: reducing the number of states, reducing the number of transitions, reducing the bits encoding the transitions, and reducing the character-set. Unfortunately, all of these approaches compress DFAs at the cost of increased main memory accesses. This technique focuses on implementations based on general-purpose processors that are cost-effective and flexible to update. This technique propose a novel solution, called deterministic finite automata with extended character-set (DFA/EC), which can significantly decrease the number of states through doubling the size of the character-set. Solution describe in this methodology requires only a single main memory access for each byte in the traffic payload [3].

We did this survey to understand the literature related to Intrusion detection using deep packet inspection techniques for RegExp matching, used to improve IDS technique in wireless networks. Implementation and comparatively evolution of existing technique with the new propose technique by considering different parameter such as bandwidth requirement, speed of intrusion detection e.t.c. The survey details describe in following section.

II. PAPER ORGRNIZATION

The rest of the paper organized as follows. An overview of DPI is given in Section 3. Here we describe Detail working of DPI and its levels. Section 4 describes the use of regular expression in DPI. Deterministic finite automata present in Section 5. Section 6 gives related DPI techniques present for RegEX detection. In this section we study each technique in detail. Limitations of this techniques and motivation we get from this survey describe in Section 7. Comparison between these techniques we show in Section 8. Section VII evaluates the performance of our architecture and presents simulation results. We conclude the paper in section 9 and discuss our future work

III. DEEP PACKET INSPECTION (DPI)

Deep Packet Inspection (DPI) is a technology that enables the network owner to analyses internet traffic, through the network, in real-time and to differentiate them according to their payload. Since, this has to be done on real time basis at the high speeds it cannot be implemented by processors or switches on software running. It has only become possible in the last few years through advances in computer engineering and in pattern matching algorithms [2].

Originally the Internet protocols required the network routers to scan only the header of an Internet Protocol (IP) packet. The packet header contains the origin and destination address and other information relevant to moving the packet across the network. The "payload" or content of the packet, which contains the text, images, files or applications transmitted by the user, was not considered to be a concern of the network operator. DPI allows network operators to scan the payload of IP packets as well as the header. Figure 1[8] shows the domain of packet inspection required in internet protocols and in DPI [8].

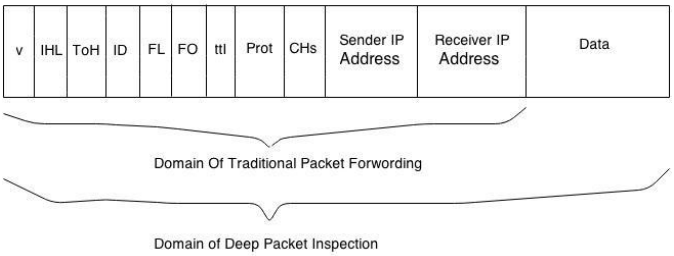

Fig. 1: Domain of Deep Packet Inspection [8]

DPI systems use expressions to define patterns of interest in network data streams. The equipment is programmed to make decisions about how to handle the packet or a stream of packets based on the recognition of a regular expression or pattern in the payload. This allows networks to classify and control traffic on the basis of the content, applications, and subscribers [8].

A. Levels of Packet Inspections

Many of the functions provided by DPI technology have been available before to limited extent depending on the level of packet analysis [1]. Packet inspection technologies that have been in use in networking environments can be classified in three classes. These three classes are 'shallow', 'medium', and 'deep' packet inspection. Figure 2 [8] provides a visual representation of the depth of inspection each of these technologies allows for.

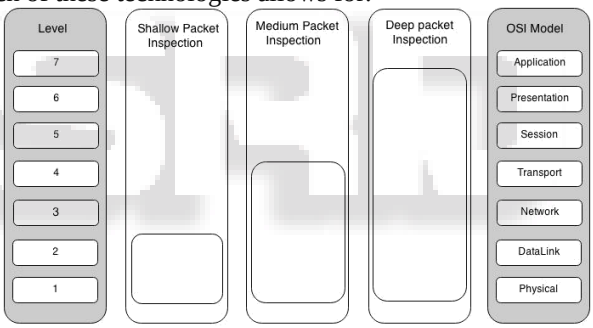

Fig. 2: Packet Inspection Depth [8].

1) Shallow Packet Inspection

Shallow packet inspection (SPI) examines the headers of the packets (which is the information placed at the beginning of a block of data, such as the sender and recipient's IP addresses), as opposed to the body or "payload" of the packet [8]. This kind of packet inspection allows the communications to remain 'virtually anonymous' since the content of the packets is not observed, and the information in the header is used only to route the packet.

2) Medium Packet Inspection

Medium Packet Inspection (MPI) is typically used to refer to 'application proxies', or devices that stand between end-users' computers and ISP/Internet gateways [8]. These proxies can examine packet header information against their loaded parse-list. When a packet enters the proxy is analyzed against a parse-list that system administrators can easily update. A parse-list allowed or disallowed specific packet-types based on their data format types and associated location on the Internet, rather than on their IP address alone.

3) Deep Packet Inspection

Deep Packet Inspection (DPI) technologies are intended to allow network operators precisely to identify the origin and content of each packet of data that passes through the networking hubs. Whereas MPI devices have very limited application awareness, DPI devices have the potential to look inside all traffic from a specific IP address, pick out the HTTP traffic, then drill even further down to capture traffic headed to and from a specific mail server, and can then reassemble e-mails as they are typed out by the user. DPI devices are designed to determine what programs generate packets, in real time, for hundreds of thousands of transactions each second.

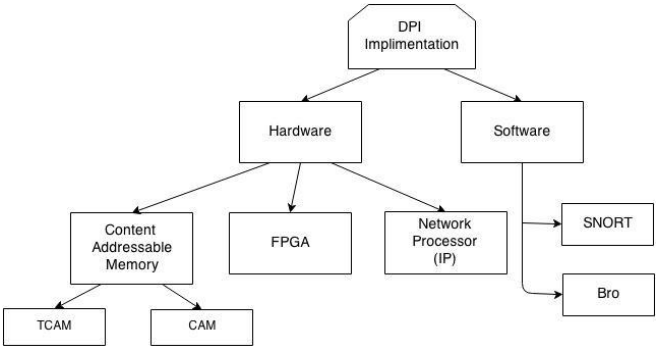

Fig. 3: DPI Implementation [7].

IV. REGULAR EXPRESSION

Today, deep packet inspection typically uses regular expression (RE) matching as a core operator. It examine whether a packet's payload matches any of a set of predefined regular expressions. REs are fundamentally more expressive, efficient, and flexible in specifying attack signatures. Prior RE matching algorithms are either software base or field-programmable gate array (FPGA) based [1].

For RegEx detection on an input, the FA starts at an initial state. Then, for each character in the input, the FA makes a transition to the next state, which is determined by the previous state and the current input character. If the resulting state is unique, the FA is called a *deterministic finite automaton* (DFA); otherwise, it is called a *nondeterministic finite automaton* (NFA) [10]. For the tradeoff between performance and resource usage, NFA and DFA represent two extreme cases. DFA has constant time complexity since it guarantees only one state transition per character by definition. The constant time complexity per character allows DFA to achieve high-speed RegEx detection, which makes DFA the preferred approach. This is especially so for software solutions since DFA can be serially executed quickly on commodity CPUs. NFA, on the other hand, requires massive parallelism, making it harder to implement in software. NFA allows multiple simultaneous state transitions, leading to a higher time complexity. The price paid for the high speed of DFA is its prohibitively large memory requirement.

RegExes consist of a variety of different *components* such as character classes or repetitions [1]. Due to this variety, it is hard to identify a method that is efficient for concurrently detection of all these different components of a RegEx. Most RegExes share similar components. In the traditional FA, a small state machine is used to detect a component in a RegEx. This state machine is duplicated since the similar component may appear multiple times in different RegExes. Furthermore, most of the time, RegExes sharing this component cannot appear at the same time in the input. As a result, the repetition of the same state machine for different RegExes introduces redundancy and limits the scalability of the RegEx detection system. Figure 3 shows example illustrating the transformation from a RegEx set R into the corresponding LaFA technique [1].

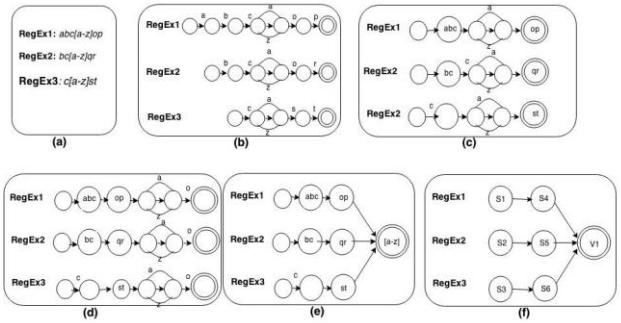

Fig. 4: Example illustrating the transformation from a RegEx set R into the corresponding LaFA. (a) RegEx set. (b) NFA corresponding to. (c) Separation of simple strings. (d) Reordering of the detection sequence. (e) Sharing of complex detection modules. (f) LaFA representation of the RegExes [1].

A. Why Regular Expression Detection

Currently, regular expressions are replacing explicit string patterns as the pattern matching language of choice in packet scanning applications. Their widespread use is due to their expressive power and flexibility for describing useful patterns. For example, in the Linux Application Protocol Classifier (L7-filter), all protocol identifiers are expressed as regular expressions. Similarly, the Snort intrusion detection system has evolved from no regular expressions in its ruleset in April 2003 to 1131 out of 4867 rules using regular expressions as of February 2006. Another intrusion detection system, Bro [10], also uses regular expressions as its pattern language [9].

As regular expressions gain widespread adoption for packet content scanning, it is imperative that regular expression matching over the packet payload keep up with the line-speed packet header processing. Unfortunately, this requirement cannot be met in many existing payload scanning implementations. For example, when all 70 protocol filters are enabled in the Linux L7-filter [1], we found that the system throughput drops to less than 10Mbps, which is well below current LAN speeds. Moreover, over 90% of the CPU time is spent in regular expression matching, leaving little time for other intrusion detection or monitoring functions. On the other hand, although many schemes for fast string matching have been developed recently in intrusion detection systems, they focus on explicit string patterns only and cannot be easily extended to fast regular expression matching [9].

B. How RegEx works in DPI

A regular expression describes a set of strings without enumerating them explicitly. Table 1 lists the common features of regular expression patterns used in packet payload scanning. For example, consider a regular expression from the Linux L7-filter for detecting Yahoo traffic: "^(*ymsg*|*ypns*|*yhoo*).?.?.?.?.?.?.?[*lwt*].**xc0\x80*". This pattern matches any packet payload that starts with

ymsg, *ypns*, or *yhoo*, followed by seven or fewer arbitrary characters, and then a letter *l*, *w* or *t*, and some arbitrary characters, and finally the ASCII letters *c0* and *80* in the hexadecimal form[9].

Syntax	Meaning	Example
^	Pattern to be matched at the start of the input	^AB means the input starts with AB. A pattern without '^', e.g., AB, can be matched anywhere in the input.
I	OR relationship	*A*|*B* denotes *A* or *B*
.	A single character wildcard	
?	A quantifier denoting one or less	*A*? denotes *A* or an empty string.
*****	A quantifier denoting zero or more	*A** means an arbitrary number of *A*s.
{}	Repeat	*A*{100} denotes 100 *A*s.
[]	A class of characters	[*lwt*] denotes a letter *l*, *w*, or *t*.
[^]	Anything but	[^*n*] denotes any character except *n*.

Table 1: Features of Regular Expressions [9].

V. Deterministic Finite Automata

The Deterministic Finite Automata (DFA) consists of a finite set of input symbols (which are denoted as P), a finite set of states, and a transition function to move from one state to the other denoted as @. In contrast of NFA, DFA has only one active state at any given time [4] [9].

The regular expression is required as a need for packet payload inspection to different protocols packets. It introduces a limited DPI system to deal with all packets structures. As the result of this limitation, state-of-art systems have been introduced to replace the string sets of intrusion signature with more expressiveness regular expression (regexp) systems. Therefore, there are several content inspection engine which have partially or fully migrated to regexps including those in Snort [11], Bro [10], and Cisco systems'. However, using the regexp to represent patterns includes converting this regexp to Deterministic Finite Automata (DFA). This DFA is represented in the DPI systems as table. This table represents the states and transitions of DFA as records which mean that the expansion of memory table of DFA of regexp depends on the size of DFA [9].

Experimentally, DFA of regexp that contains hundreds of pattern yields to tens of thousands of states which mean memory consumptions in hundreds of megabytes. As a solution of one of the common problems of HW based DPI solutions is the memory access because the memory accesses for the contents of the off chip memory are proportional with the number of bytes in the packet [9].

A. DFA Analysis for Individual Regular Expressions

Next, we study the complexity of DFA for typical patterns used in real-world packet payload scanning applications such as Linux L7-filter, Snort, and Bro. The study is based on the use of *exhaustive matching* and *one-pass search*. Table 4 summarizes the results [6] [9].

Explicit strings generate DFAs of length linear to the number of characters in the pattern [9].

- If a pattern starts with '^', it creates a DFA of polynomial complexity with respect to the pattern length *k* and the length restriction *j*.
- Our observation from the existing payload scanning rule sets is that the pattern length *k* is usually limited but the length restriction *j* can reach hundreds or even thousands.
- Therefore, Case 4 can result in a large DFA because it has a factor quadratic in *j*. • Patterns starting with ".*" and having length restrictions (Case 5) cause the creation of DFA of exponential size.

Pattern Feature	Example	# of States
1. Explicit string with *k*\ characters	*^ABCD* **ABCD*	*K+1*
2. Wildcards	*^AB*CD* **AB*CD*	*K+!*
3. Pattern with ^, a wildcard And a length restriction *j*	*^AB{j+}CD* *^AB{o,j}CD* *^AB{j}CD*	*O(k*j)*
4. Pattern with a^, a class of characters overlap with the prefix, and a length restriction *j*.	*^A+[A-Z]{j}D*	*O(k+j2)*
5. Pattern with a length restriction *j* where a wildcard or a class of characters overlaps with the prefix.	**AB{j}CD* **A[A-Z]{j+}D*	$O(k + 2^j)$

Table 2: Analysis of patterns with *k* characters [9].

VI. Related Work

A. LaFA [1]

LaFA is a novel detection method that resolves scalability issues of the current RegEx detection paradigm. LaFA is finite automata optimized for scalable RegEx detection. LaFA is used for representing a set of RegExes (R={r1;r2;r3;…}) which can also be called a *RegEx database*. An associated LaFA RegEx detection system can be queried with an input , such as a network packet. The system will return *a match* along with a list of matching RegExes if input includes one or more of the RegExes in . Otherwise, a *no match* is returned. LaFA facilitate the reduction of memory requirements and detection complexity.

1) LaFA Architecture

LaFA architecture shown in following fissure consist of two blocks, The detection block and The correlation block Detection Block is responsible for detecting the component in the input string. The simple string optimized for exact string matching module used to detect simple string.

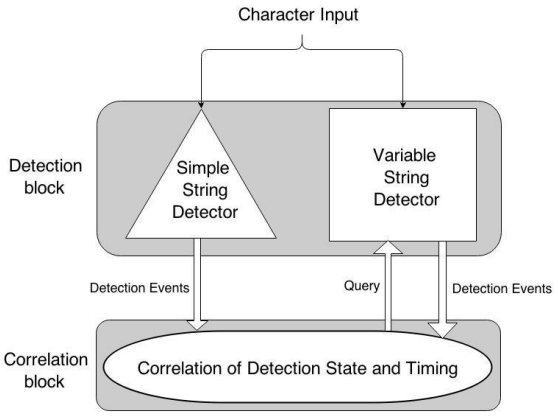

Fig. 5: LaFa Architecture [1].

The variable string detector consists of highly optimized variable string detection modules. The correlation block used to track status of RegEx. The correlation block inspects the sequence of components in the input string (i.e., order and timing). The detectors in the detection block communicate their findings to the correlation block by sending detection events. Each detection event consists of a unique ID for the detected component and its location in the input packet.

2) Evaluation of LaFA

Following Table 3 shows the number of maximum concurrent operations per simple-string detection for different RegEx sets.

RegEx Set	Number of Operations									Only One Operation	Max. Op.	
	1	2	3	4	5	6	7	8	9		LaFA	NFA
Snort24	44	4	0	0	0	0	0	0	0	91.67%	2	16
Snort31	1126	5	0	0	0	0	0	0	0	99.56%	2	38
Snort34	44	6	1	0	0	0	0	0	0	86.27%	3	11
SnortBD	215	14	5	4	3	0	0	0	0	89.22%	5	57
SnortPHP	34	0	0	0	0	0	0	0	0	100.00%	1	25
SnortWEBCL	531	8	3	1	0	1	0	0	0	97.61%	6	313
SnortSpy	644	52	11	5	4	1	3	3	0	88.34%	8	390
SnortComb	1445	102	19	12	8	1	5	3	0	90.04%	8	409
Bro	547	57	4	0	0	0	0	0	0	89.97%	3	263
LinuxL7	1447	15	10	1	24	0	0	0	0	74.62%	5	328

Table 3: Number of Operations Per Simple-String Detection [1].

Fig. 6 compares memory requirements with respect to RegEx set size between our scheme and four others: XFA, HybridFA, SplittingFA , and CD FA [1]. We can see that the memory requirement of LaFA is an order of magnitude smaller than other schemes.

The LaFA memory requirements trend among different sizes of rules sets is shown as a line in Fig. 6.

B. Small TCAM [2]

Small TCAM is the first ternary content addressable memory (TCAM)-based RE matching solution. This technology use a TCAM and its associated SRAM to encode the transitions of the DFA built from an RE set where one TCAM entry might encode multiple DFA transitions. There are three key reasons why TCAM-based RE matching works well given in this research: a small TCAM is capable of

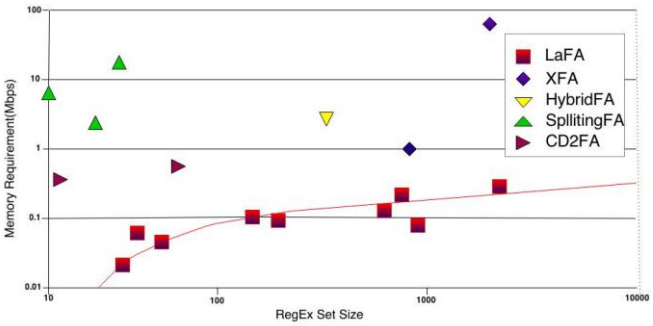

Fig. 6: Scalability comparisons with other schemes [1].

encoding a large DFA with carefully designed algorithms. TCAMs facilitate high-speed RE matching because TCAMs are essentially high-performance parallel lookup systems TCAMs are off-the-shelf chips that are widely deployed in modern networking devices; it should be easy to design networking devices that include our TCAM-based RE matching solution [2].

1) Working of TCAM

Transition sharing: The basic idea of transition sharing is to combine multiple transitions into a single TCAM entry. TCAM uses two transition shearing ideas, character bundling and shadow encoding. Character bundling exploits intrastate optimization opportunities and minimizes TCAM tables along the input character dimension. Shadow encoding exploits interstate optimization opportunities and minimizes TCAM tables along the source state dimension.

This technique present table consolidation where it combines multiple transition tables for different states into a single transition table. The combined table takes less TCAM space than the total TCAM space used by the original tables.

In Variable Striding technique explore ways to improve RE matching throughput by consuming multiple characters per TCAM lookup. Variable striding can achieve a significant RE matching throughput increase with a small and controllable space increase.

2) TCAM Methodology

TCAM-based RE matching solution on real-world RE sets focusing on two metrics: TCAM space and RE matching throughput. This technique requires the maximum TCAM size for the eight RE sets is only 0.50 Mb. With the transition sharing algorithm alone, the maximum TCAM size is only 1.43 Mb for the eight RE sets. This RE matching solution achieves high throughput with even 1-stride DFAs. the average throughput of TCAM is 4.60 Gb/s (ranging from 3.64 to 8.51 Gb/s).

Table 4 shows TCAM experimental results on the eight RE sets using 1-stride DFAs. It uses TS to denote our transition sharing algorithm including both character bundling and shadow encoding. It uses TC2 and TC4 to denote table consolidation algorithm where we consolidate at most two and four transition tables together, respectively. For each RE set, it measure the number states in its 1-stride DFA, the resulting TCAM space in megabits, the average number of TCAM table entries per state, and the projected RE matching throughput; the number of TCAM entries is the number of states times the average number of entries per state. The TS column shows our results when

TCAM apply TS alone to each RE set. The TS+TC2 and TS+TC4 columns show our results when we

apply both TS and TC under the consolidation limit of 2 and 4, respectively, to each RE set.

RE set #states	TS			TS+TC2			TS+TC4		
	Tcam Mbits	#rows Per set	Thru Gbps	Tcam Mbits	#rows Per set	Thru Gbps	Tcam Mbits	#rows Per set	Thru Gbps
Bro217 6533	0.31	1.40	3.64	0.21	0.94	4.37	0.17	0.78	4.35
C613 11308	0.63	1.61	3.11	0.52	1.35	3.64	0.45	1.17	3.64
C10 14868	0.61	1.20	3.11	0.31	0.61	3.64	0.16	0.32	4.35
C7 24750	1.00	1.18	3.11	0.53	0.62	3,64	0.29	0.34	3.64
C8 3108	0.13	1.20	5.44	0.07	0.62	5.44	0.03	0.33	8.51
Snort24 13886	0.55	1.16	3.64	0.30	0.64	3.64	0.18	0.38	4.35
Snort31 20068	1.43	2.07	2.72	0.81	1.17	2.72	0.50	0.72	3.64
Snort34 13825	0.56	1.18	3.11	0.30	0.62	3.64	0.17	0.36	4.35

Table 4: TCAM Size and Throughput For 1-Stride Dfas [2].

In this experiment, TCAM able to store a DFA with 25 K states in a 0.5-Mb TCAM chip; most DFAs require at most one TCAM entry per DFA state. With variable striding it shows a throughput of up to 18.6 Gb/s is possible.

C. StriFA[5]

StriFA technique has been implemented in software and evaluated based on different traces. It undertakes the problem of designing a variable-stride pattern matching engine that can achieve an ultrahigh matching speed with a relatively low memory usage. The technology proposes stride finite automata (StriFA), which can process a variable number of characters at a time. StriFA is designed to be immune to the memory blow-up and byte alignment problems, and therefore, requires much less memory than the previous schemes. Stride deterministic finite automaton (StriDFA) and stride nondeterministic finite automaton (StriNFA) are two basic forms of implementation of StriFA.

StriFA is a novel acceleration scheme for both RegEx matching and string matching. The main idea of the scheme is to convert the original input byte stream into a much shorter integer stream and then match the short integer stream with a variant of NFA/DFA, called StriNFA/StriDFA, to identify the potential matches in the original input byte stream. A formal algorithm is proposed to convert traditional NFA directly to StriNFA without the need for DFA. The memory explosion problem originally involved in the DFA construction can be avoided by this method. An improved version of StriFA that utilizes neighbor information to reduce its false alarm rate at the cost of a small increase in memory is proposed. The implementation of a general instance of StriFA achieves both space and time efficiency and can expediently migrate to existing platforms. Results show that approximately a ten-fold increase in speed is achievable, while the memory cost is smaller than that of traditional NFA/DFA.

1) StriFA Working

The matching process of RegEx is performed by sending the input stream to the automaton byte by byte. If the DFA reaches any of its accept states (the states with double circles), we say that a match is found. This technique reduces the number of states to be visited during the matching process. In this technology Instead of comparing the input stream character by character with patterns in the rule set; it pick tag characters from the input stream and feed the fingerprint of these tag characters to the automaton for the matching examination. It uses distance (or the number of characters) between adjacent tags (denoted as stride lengths or step sizes) as the fingerprint.

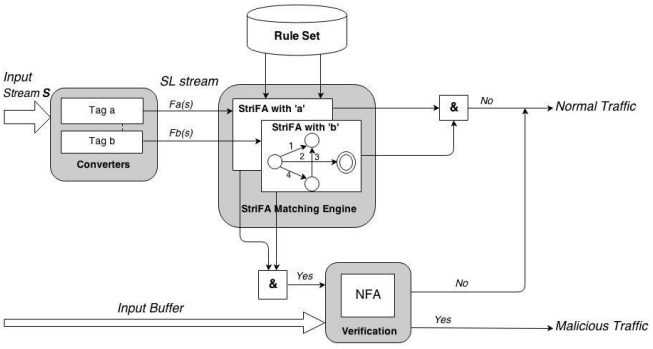

Fig. 7: Architecture of StriFA-based matching engine [5].

The architecture of StriFA is shown in Figure 7 [6], which consists of three components: SL convertor, StriFA matching engine, and verification module. The SL convertors convert the input byte stream into multiple SL streams according to different predetermined tags. The core is a stride-based matching engine whose function is to match the input against RegEx rules, similar to that of a traditional NFA or DFA. Finally, the verification phase is used if a potential match is found by all StriFAs.

StriNFA is a special kind of NFA, in which the labels are all integers. Transforming StriNFA to StriDFA is equivalent to the procedure of transforming from NFA to DFA. One of the traditional transformation algorithms is called structural induction used for this conversion.

2) 6.3.2 Evaluation

Messages on the Internet are transmitted in small units called packets. Each packet contains a header and a payload. This technique extracts the payload of each packet and feed it to StriFA-based matching engine. To evaluate the efficiency of the StriFA-based matching engine, an instance of it was implemented. The instance is implemented on a software platform. This experiment carried out on two desktop PCs, each having eight 3.8-GHz CPUs and 12-GB memory. The memory usage of StriNFA and StriDFA is much smaller than those of traditional NFA/DFA.

StriFA is a novel regular expression matching acceleration scheme for complex network intrusion detection systems. The main idea of StriFA is to convert the original byte stream into a much shorter integer stream and then match the integer stream with a variant of DFA, called StriFA. They provided the formal construction algorithm of StriFA that was able to transform an arbitrary set of RegEx to a StriFA..

The technology results showed that this architecture can achieve about 10-fold increase in speed, with a lower memory consumption compared to traditional NFA/DFA, while maintaining the same detection capabilities.

D. CompactDFA [6]

One of the main advantages of CompactDFA is that it fits into commercially available IP-lookup solutions. They may be used also for performing fast pattern matching. The output of the CompactDFA scheme is a set of compressed rules, such that there is only one rule per state. In the

compressed set of rules, a code of a state may match multiple rules. Algorithm of CompactDFA and gives the intuition behind each of its three stages: State Grouping, Common Suffix Tree Construction, and State and Rule Encoding.

1) Working

In this technique the DFA accepts the input string if one of the patterns exists in the input. All the incoming transitions to a specific state are labeled with the same symbol, this property is crucial to CompactDFA algorithm. There are no redundant states at the DFA. This implies that every state has a path of forward transitions to an accepting state. Any specific input leads to exactly one specific state; this is a by-product of the determinism of the automaton.

This technique suggests a method of reducing the number of transitions in DFA to the minimum possible one, the number of DFA states. CompactDFA does not depend on any specific hardware architecture or any statistic property of data. This technique eliminate cross transitions by using information from the next state label. It uses the table consolidation technique, which combines entries even if they lead to different states. It minimizes the total memory requirement, namely, the product of the number of rules and the total number of bits required to encode a rule. To increase the speed of DPI using CompactDFA, It performs a lookup on a sequence of characters of the input. In this each such stride will have a predetermined size of k , thus potentially boosting up the performance by a factor of k.

2) Evaluation

CompactDFA evaluates only for the pattern matching process. It uses two common pattern sets: Snort and ClamAV. This methodology can achieve fast pattern matching of 2 Gb/s with low power consumption.

This technique shows a reduction of the pattern matching problem to the IP-lookup problem. CompactDFA, gets as an input a DFA and produces compressed rule sets; each compressed rule defines the set of states that the rule applies to using a common prefix. It focuses on the usage of TCAM [2] for pattern matching, a hardware device that is commonly used for IP-lookup and packet classification and is deployed in many core routers. This technique can achieve fast pattern matching of 2 Gb/s with low power consumption. Due to its small memory and power requirements, this architecture can implement with several TCAM working in parallel. Each TCAM performs pattern matching on a different session, achieving a total throughput of 10 Gb/s and beyond.

E. A DFA with Extended Character-Set [4]

Implementation of this technique based on general-purpose processors that are cost-effective and flexible to update. It proposes a novel solution, called deterministic finite automata with extended character-set (DFA/EC), which can significantly decrease the number of states through doubling the size of the character-set. Solution describe in this methodology requires only a single main memory access for each byte in the traffic payload. It performs experiments with several Snort rule-sets. Results show that, compared to DFAs, DFA/ECs are very compact and are over four orders of magnitude smaller in the best cases.

1) Methodology

This technique introduces DFA/EC, a general DFA model that incorporates a part of the DFA state into the set of input characters. It provides an efficient implementation of the inspection program based on our DFA/EC model, which results in a compact transition table and a fast inspection speed. This methodology proves that DFA/EC is equivalent to DFA. It performs an extensive evaluation to compare DFA/ EC with related algorithms by using several Snort rule sets.

This technique proposes a novel solution, called deterministic finite automata with extended character-set (DFA/EC), which can significantly decrease the number of states through doubling the size of the character-set. DFA with extended character-set (DFA/EC) reduces the DFA size, DFA/EC select some of the most frequently active NFA states and incorporate them into the character-set (or the alphabet) of the DFA to form a slightly larger extended character-set.

In this methodology, It have additional constraints, which exclude some of the frequently active NFA states from the set of complementary states in order to enable a single-bit encoding method of the complementary states in the extended character set, and to facilitate an efficient DFA/EC implementation.

2) Working of DFA/EC

The advantages of a DFA/EC are summarized in the following: A DFA/EC requires only one main memory access for each byte in the packet payload, while significantly reducing storage in terms of table size. A DFA/EC is conceptually simple, easy to implement, and easy to update due to fast construction speed. A DFA/EC can be combined with other compression approaches to provide a better level of compression.

A DFA/EC maintains two states in runtime: one state for the main DFA, and an additional state for the complementary program. The current runtime state of the main DFA is represented by a DFA state label, and state of the complementary program is represented by a set that contains currently active complementary states. For each byte in the payload, the DFA/EC functions as follows. Firstly, the complementary program calculates the extra bit for the extended character by using the next byte and the current state of the complementary program and the next state of the main DFA and a label is looked-up by using the current state of the main DFA and the extended character, which is composed of the next byte and the extra bit. After that the complementary program calculates its next state by using its current state, the next byte, and the label on the main DFA transition. Essentially, the execution of DFA/EC is an interactive process between the main DFA and the complementary program. Interactions in DFA/EC can be implemented efficiently by using a single main memory access and several bit-wise instructions. In this implementation, the transition functions of the main DFA, is implemented by a transition table; and is implemented by the complementary program, which only contains several efficient instructions.

3) 6.5.3 Evaluation

For evaluation of this technique, several compilers are developed, which read files of rules and created the

corresponding inspection programs and the transition tables for DFA, MDFA, H-FA, and DFA/ EC. Rule-sets extracted from the Snort rules. Thirdly, the synthetic payload generator Developed. Inspection programs generated for the rule-sets, measure their storages, and load them with the synthetic payloads to measure their performances.

This algorithm compared with this widely adopted algorithm to show the efficiency of this method in terms of storage, memory bandwidth, and speed. To generate a payload stream for a rule-set, it travels the DFA of the whole rule-set. Measurement of the memory requirement for each inspection program in terms of the number of states, the number of transitions, and the bits needed to store the transitions.

The total minimum memory (storage) requirement of the transition tables in terms of bits and the number of bits is the product of the number of transitions and the number of bits needed to encode each transition measured. It gives result as, the memory bandwidth of DFA/EC can even be smaller than DFA in rule-sets exploit-19 and web-misc-28. Figure 10 show the number of main memory accesses per KB of payload.

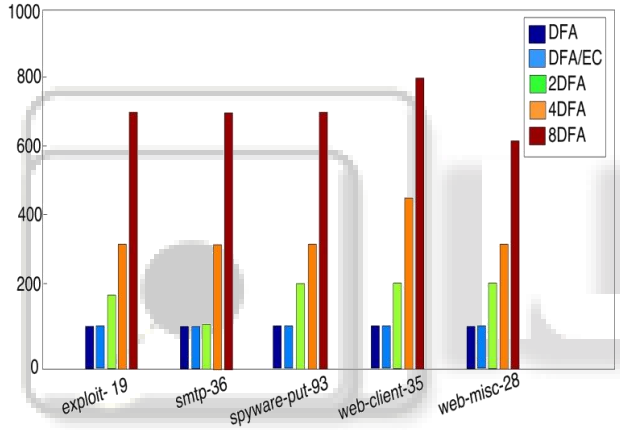

Fig. 8: Memory accesses (times/kb) with different rule-sets [4].

F. NBA Systems [3]

This is network behavior analysis (NBA) technique, which examines network traffic to identify threats that generate unusual traffic flows, such as distributed denial of service (DDoS) attacks, certain forms of malware (e.g., worms, backdoors), and policy violations. NBA systems are most often deployed to monitor flows on an organization's internal networks, also sometimes deployed where they can monitor flows between an organization's networks and external networks.

1) Components of NBA

NBA Solutions usually have sensors and consoles, with some products also offering management servers sometimes called analyzers. NBA sensors are usually available only as appliances used for network implementation. Some sensors are like network-based IDPS sensors in that they sniff packets to monitor network activity on network segments. Other NBA sensors do not directly monitor the networks, but instead rely on network flow information provided by routers and other networking devices. Network flow refers to a particular communication session occurring between hosts. There are various standards for flow data formats, including NetFlow and sFlow. Typical flow data particularly relevant to intrusion detection and prevention includes the following: Source and destination IP addresses Source and destination TCP or UDP ports, Number of packets and number of bytes transmitted in the session, and Timestamps for the start and end of the session.

2) Network Architecture

As with a network-based IDPS, a separate management network or the organization's standard networks can be used for NBA component communications. If sensors that collect flow data from other devices are used, the entire NBA solution can be logically separated from the standard networks. Figure-11 shows an example of NBA network architecture.

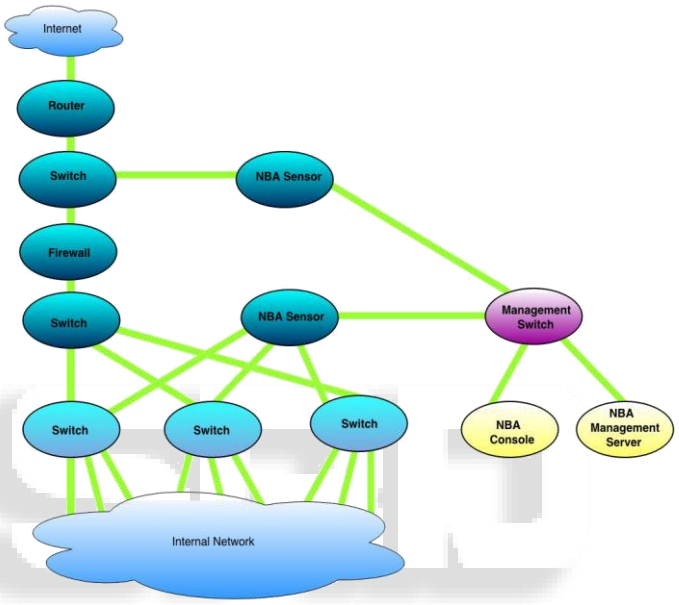

Fig. 9: Passive Network-Based IDPS Sensor Architecture Example [3].

3) Methodology

Choosing the appropriate network for the components, the administrators need to decide where the sensors should be located. NBA sensors can be deployed in passive mode only, using the same connection methods (e.g., network tap, switch spanning port) as network based IDPSs. NBA products provide a variety of security capabilities. Common security capabilities, divided into four categories, these are information gathering, logging, detection, and prevention. NBA technologies offer extensive information gathering capabilities, because knowledge of the characteristics of the organization's hosts is needed for most of the NBA product's detection techniques. NBA sensors can automatically create and maintain lists of hosts communicating on the organization's monitored networks. NBA technologies typically perform extensive logging of data related to detected events on the network. This information can be used to confirm the validity of alerts, to investigate incidents and to correlate events between the NBA solution and other logging sources. Some NBA sensors that directly monitor network traffic are able to log limited payload information from packets, such as authenticated user identifiers. This allows actions to be traced to specific user accounts. NBA technologies typically

have the capability to detect several types of malicious activity. Most products use primarily anomaly-based detection, with some stateful protocol analysis techniques, which is used to analyze network flows.

4) Experimental Results

Dos attacks typically involve significantly increased bandwidth usage or a much larger number of packets or connections to or from a particular host than usual user. Anomaly detection methods can determine if the observed activity is significantly different than the expected activity by monitoring these characteristics,. Scanning can be detected by atypical flow patterns at the application layer, transport layer, and network layer. Worms spreading among hosts can be detected in many ways. Some worms propagate quickly and use large amounts of bandwidth. Worms can be detected because they can cause hosts to communicate with each other that typically do not, and they can also cause hosts to use ports that they normally do not use.

VII. Limitations and Motivation

From the survey of above techniques we found that, the approaches describe in these techniques may require a large number of transitions for some cases, leading to an increase in the number of memory accesses per input byte. In addition, DFA construction is complex and requires significant resources [1]. There is very few network intrusion detection techniques discover in wireless networks. CompactDFA technique used in architecture requires several TCAM working in parallel, Due to its small memory and power requirements. NBA technologies have some significant limitations. They are delayed in detecting attacks because of their data sources, especially when they rely on flow data from routers and other network devices [3]. DFA/EC does not combine with the existing transition compression and character-set compression techniques, and perform experiments with more rule-sets [4]. One of the problems for StriFA is how to choose an appropriate tag. Since in both the rules and the incoming traffic, the occurrence probabilities of different characters vary from each other, it is a problem to choose an appropriate tag from the rule set [5].

From studying these limitations of intrusion detection techniques we motivate for implementation of approved intrusion detection technique with the use of wireless network.

VIII. Result and Comparison

Table 12 shows the comparison of above deep packet intrusion techniques with respect to the maximum throughput.

Intrusion Detection Techniques	Throughput
LaFA	34 Gb/s
CompactDFA	10 Gb/s
Small TCAM	18.6 Gb/s
StriDFA	26.5 Gb/s

Table 5: Comparison of deep packet intrusion techniques.

IX. Future work and Conclusion

On the basis of above survey, we conclude that network intrusion techniques discuss in this paper can be improved for better results and performance. In future we try to improve DPI techniques in wireless networks. We try to improve limitations on these techniques discuss in section 5 and try to reduce memory consumption required for regular expression matching. Furthermore we will implement or improve Dpi technique which will work on the wireless networks for RegExp detection.

References

[1] Masanori Bando, N. Sertac Artan, and H. Jonathan Chao., "*Scalable Lookahead Regular Expression Detection System for Deep Packet Inspection*", IEEE Transactions on Networking, Vol. 20, No. 3, June 2012.

[2] Chad R. Meiners, Jignesh Patel, Eric Norige, Alex X. Liu, and Eric Torng., "*Fast Regular Expression Matching Using Small TCAM*", IEEE/Acm Transactions On Networking, Vol. 22, No. 1, February 2014.

[3] Tiwari Nitin, Solanki Rajdeep Singh and Pandya Gajaraj Singh, "*Intrusion Detection and Prevention System (IDPS) Technology- Network Behavior Analysis System (NBAS)*", ISCA Journal of Engineering Sciences, Vol. 1(1), 51-56, July 2012.

[4] Cong Liu, Yan Pan, Ai Chen, and Jie Wu., "*A DFA with Extended Character-Set for Fast Deep Packet Inspection*", IEEE Transactions On Computers, Vol. 63, No. 8, August 2014.

[5] Xiaofei Wang, Yang Xu, Junchen Jiang, Olga Ormond, Bin Liu, and Xiaojun Wang, "*StriFA: Stride Finite Automata for High-Speed Regular Expression Matching in Network Intrusion Detection Systems*", IEEE Systems Journal, Vol. 7, No. 3, September 2013.

[6] Anat Bremler-Barr, DavidHay, and Yaron Koral, "CompactDFA: Scalable Pattern Matching Using Longest Prefix Match Solutions", IEEE/Acm Transactions On Networking, Vol. 22, No. 2, April 2014.

[7] Tamer AbuHmed, Abedelaziz Mohaisen, and DaeHun Nyang., "*A Survey on Deep Packet Inspection for Intrusion Detection Systems*", Information Security Research Laboratory, Inha University, Incheon 402-751, Korea, March 2008.

[8] Klaus Mochalski, and Hendrik Schulze,"*White paper on Deep Packet Inspection*", ITU-T study groups com13.

[9] Fang Yu, Zhifeng Chen, Yanlei Diao, T. V. Lakshman, and Randy H. Katz, "*Fast and Memory-Efficient Regular Expression Matching for Deep Packet Inspection*", ACM 580-0/06/0012, December 3–5, 2006.

[10] Bing Chen, Lee, J., and Wu, A.S., "*Active event correlation in Bro IDS to detect multi-stage attacks*", Fourth IEEE International Workshop on Information Assurance, 13-14 April 2006.

[11] Rafeeq Ur Rehman, "*Intrusion Detection Systems with Snor*t", ISBN 0-13-140733-3, Library of Congress Cataloging-in-Publication Data, Prentice Hall PTR Upper Saddle River, New Jersey 07458.

