ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 1, January 2017

A Survey on High Speed and Memory Efficient Regular Expression Pattern Matching

T. Krishna Kishore

Assistant Professor, Department of CSE, St. Ann's College of Engineering & Technology, Chirala, India

ABSTRACT: Deep packet inspection has become a key component in network intrusion detection systems (NIDSes), where every packet in the incoming data stream needs to be compared with patterns in an attack database, byte-by-byte, using either string matching or regular expression matching. Regular expression matching, despite its flexibility and efficiency in attack identification, brings significantly high computation and storage complex- ities to NIDSes, making line-rate packet processing a challenging task. In this paper, we present stride finite automata (StriFA), a novel finite automata family, to accelerate both string matching and regular expression matching. Different from conventional finite automata, which scan the entire traffic stream to locate malicious information, a StriFA only needs to scan a partial traffic stream to find suspicious information. The presented StriFA technique has been implemented in software and evaluated based on different traces. The simulation results show that the StriFA acceleration scheme offers an increased speed over traditional nondeterministic finite automaton/deterministic finite automaton, while at the same time reducing the memory requirement.

KEYWORDS: Deep packet inspection (DPI), deterministic finite automaton (DFA), network intrusion detection systems (NIDSes), nondeterministic finite automaton (NFA).

I. INTRODUCTION

Deep Packet Inspection (DPI) has widely been deployed in modern network intrusion detection systems (NIDSes) to detect attacks and viruses in Internet traffic based on patterns stored in a database. Examples include Snort [1], ClamAV [2], and security applications from Cisco Systems [3]. The format of these patterns is either strings or regular ex- pressions (regex). To support increasingly complex services, regexes have been used to replace strings in DPI because of their better expressiveness and flexibility. However, regex matching also brings significantly high computation and storage complexities to NIDSes, which prohibits its usage in applications that require high processing speed or have limited memory space. Designing a regex matching engine that achieves both time and space efficiency is a great challenge.

Deterministic finite automaton (DFA) and nondeterministic finite automaton (NFA) are two typical finite automata used to implement regex matching. DFA is fast and has deterministic matching performance, but suffers from the memory explosion problem. NFA, on the other hand, requires less memory, but suffers from slow and nondeterministic matching performance. Therefore, neither of them is suitable for implementing high- speed regex matching in environments where the fast memory (e.g., cache or on-chip memory) is limited.
Recently, many research works have focused on improving the speed and/or reducing the memory cost of regex matching [4]–[7]. These schemes can be roughly classified into two categories: 1) single-byte stride1 scanning, and (2) multibyte stride scanning. Traditional NFA and DFA along with some of their variations, including HybridFA [8], k-DFA [9], and XFA [10], [11], scan only one character at a time and belong to the first category. The main research focus of schemes in this category is to: 1) 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; or 2) reduce the memory consumption by reducing the state number or transition number of the automaton. Schemes in the second category scan multiple characters at a time, and therefore, naturally provide faster matching speed than those in the first category. However, most schemes in the second multibyte stride scanning category suffer from two problems.
1) Memory blow-up problem: due to the exponential growth of transition numbers when the stride increases.
2) Byte alignment problem: for multibyte scanning, every character of the input stream should have the chance to be examined as the first character; this requires duplicate automata to return the correct matching results.
In this paper, I undertake the problem of designing a variable-stride pattern matching engine that can achieve an ultrahigh matching speed with a relatively low memory usage. More specifically, we propose a stride finite automata (StriFA), which can process a variable number of characters at a time. Compared to other algorithms that also examine multiple 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. The proposed StriFA is a term, describing a family of automata and language that share the same concept. Stride deterministic finite automaton (StriDFA) and stride nondeterministic finite automaton (StriNFA) are two basic forms of implementation of StriFA.

II. RELATED WORK

Regex matching was originally studied as a topic in au- tomata theory and formal theory in the context of theoretical computer science [14]. To accelerate the regex matching in real-world systems, the problem has been intensively studied in practical scenarios in recent years. Vulnerability signatures have recently been proposed as an alternative to regex, but this still requires a high-speed regex matching subsystem [15].

Brodie et al. [16] increased the throughput of regex match- ing by expanding the alphabet set, resulting in an exponential increase in memory requirement in the worst case. A more re- cent method [17] introduced sampling techniques to accelerate regex matching, but not all types of regex are supported.

In addition to the aforementioned acceleration approaches, DFA-based compression methods also enhance the system performance because they result in smaller DFAs that can be put into the fast memory. Transition compression approaches obtain a high compression rate by reducing the number of transitions for each state. D2 FA [18] is acknowledged as the original work in this approach. This compresses DFA by applying default transitions at the cost of accessing the DFA multiple times per input character. Subsequent work, including [19] and [20], improves on the worst case and average performance.

State compression techniques were first utilized in [6], where patterns are selectively grouped to deflate state ex- plosion. Other work in [8] performed a partial NFA-to-DFA conversion to prevent state explosion. The state-of-the-art work XFA [10], [11] uses auxiliary memory to reduce the DFA state explosion and achieves a great reduction rate. However, XFA is not suitable for real-time applications on networks due to the significant startup overhead. Our proposed method does not conflict with the above work, since the fundamental DFAstructure is completely preserved.

III. STRIDFA FOR MULTISTRING MATCHING

In this section, we demonstrate the main concepts of StriFA using an example. For the sake of simplicity, the example used here only considers string matching, which is a special case of regex matching. In the next section, we will present a general solution covering both string matching and regex matching.

Suppose we have two patterns to match, “reference” (P1) and “replacement” (P2). The conventional scheme of pattern matching is to first convert the patterns to a DFA or NFA. In this example, we consider only DFA, which is shown in Fig. 1. The matching process 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. It is easy to see that the number of states to be visited during the processing is equal to the length of the input stream (in units of bytes), and this number determines the time required for finishing the matching process (each state visit requires a memory access, which is a major bottleneck in today’s computer systems).

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 1, January 2017

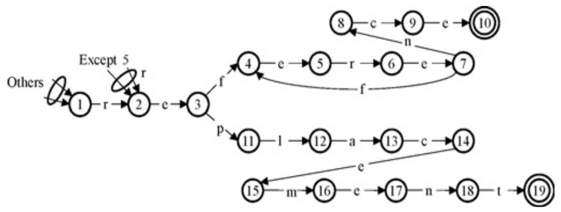

Fig.1.TraditionalDFAforpatterns“reference”and“replacement”(some transitions are partly ignored for simplicity).

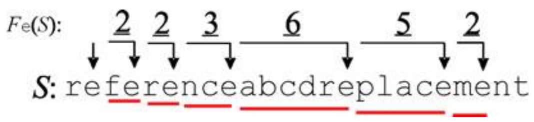

Fig. 2. Tag “e” and a sliding window used to convert an input stream into an SL stream with tag “e.”

Instead of comparing the input stream character by character with patterns in the rule set; we pick tag characters from the input stream and feed the fingerprint of these tag characters to the automaton for the matching examination. Since the fingerprint is normally much shorter than the original input stream, the number of state visits required by the matching process can be significantly reduced.

Here, we use distance (or the number of characters) between adjacent tags (denoted as stride lengths or step sizes) as the fingerprint. Stride lengths extracted from the rule set are compared with stride lengths extracted from the input strings for coarse grained matching.

For example, if we select “e” as the tag and consider “referenceabcdreplacement,” as shown in Fig. 2, then the corresponding stride length (SL) stream is Fe (S) = 22 3 6 5 2, where Fe (S) denotes
Fe (S) denotes the SL stream of the input stream.

IV. STRIDE FINITE AUTOMATON

The architecture of StriFA is shown in Fig. 4, which consists of three components: SL convertor, StriFA matching engine, and verification module.

1) The SL convertors convert the input byte stream into multiple SL streams according to different predeter- mined tags.

2) The core is a stride-based matching engine whose func- tion is to match the input against regex rules, similar to that of a traditional NFA or DFA.

3) Finally, the verification phase is used if a potential match is found by all StriFAs.

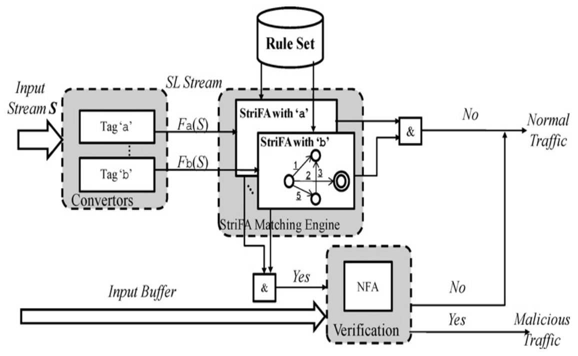

Fig. 4.Architecture of StriFA-based matching engine

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 1, January 2017

A.LIMITING THE SIZE OF ALPHABET SET BY SLIDING WINDOW

To solve the problem of the large alphabet set, a fixed size sliding window is adopted (a similar application can be found in [12-13]). The window works in the following way (see Fig. 5): if a tag is not found within a window distance (here window size w=5), then the last character of the window is marked as a fingerprint anchor (the fingerprint anchor is not a tag, but treated like a tag to get the SL from the previous tag), the window size w is sent to the StriFA and the character following the fingerprint anchor is set to be the beginning of the window. In this manner, any SL sent to a StriFA is limited in a finite alphabet set = {1,... , w}.

B.BUILDING STRINFA BY NFA- BASED METHOD

We propose a new StriNFA transforming approach, as de- scribed in the following steps (with corresponding block component diagrams in Fig. 6).

1) ***Step1:CompilingregextoCorrespondingNFA:***Themethod of compiling a regex to a corresponding NFA is conventional (intensively studied in [14]). Fig. 7(a) is the traditional NFA of regex .*abba.{2}caca. Instead of converting a traditional NFA to a traditional DFA, tag decision FA will be generated by the traditional NFA directly

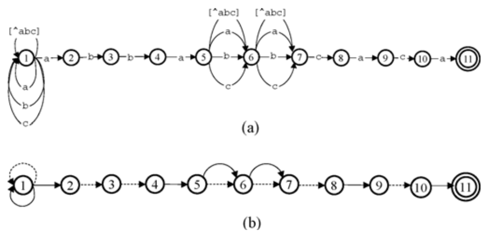

Fig.7.Traditional NFA and tag decision FA of regex .*abba.*{2}*caca. (a)TraditionalNFA.(b)TagdecisionFAusingtag“a.”

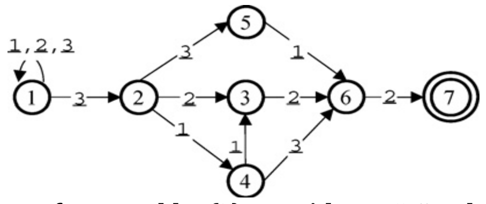

Fig.8.StriNFAofregex.*abba.*{2}*cacawithtag=“a”and*w*=3.

2) ***Step 2: Restructuring NFA to Tag Decision FA:*** In this step, each transition is drawn as a solid line if its label is the tag, or it is drawn as a dotted line otherwise. Then, all labels are removed from the transitions of the traditional DFA. The output structure is called tag decision FA, as shown in Fig. 7(b), after transformation from Fig. 7(a) using tag “a.”

3) ***Step 3:TransformingTag DecisionFAtoStriFA:***In this step, we generate a stride nondeterministic FA (StriNFA) based on the tag decision FA (a directed graph consisting of solid and dotted transitions). Nondeterministic means that some states can have more than one outgoing transition labeled with the same SL (integer). To explain the method, the following steps are processed recursively, starting from any state p in a tag decision FA.

1) Case 1: if a solid transition (pointing to state q) is reachable in l steps where $l \leq w$, add a transition from p to q with label l (stride length).

2) Case 2: otherwise, if there is a all-dotted-transition path of length w to state q, then add a transition from p to q with label w.

C.STRINFA TO STRIDFA

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 tradi- tional transformation algorithms is called structural induction in textbooks [21], which was proposed by Thompson [22].

As shown in Fig. 10, StriDFA can be constructed from StriNFA in Fig. 8 by the aforementioned construction method.

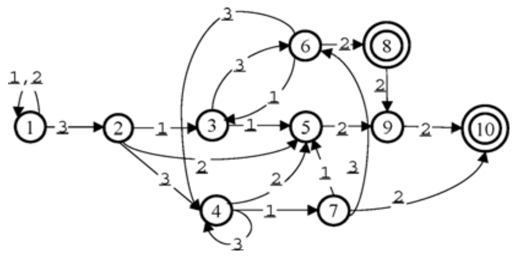

Fig. 10. Corresponding StriDFA (transitions back to states 1, 2, and 3 are partly ignored for simplicity).

D. CORRECTNESS PROOF

Generally, a false negative indicates that the intrusion de- tection system is unable to detect a genuine attack . The false negative in the StriFA detection system means that the traditional NIDS can find the intrusion, while the corresponding StriFA detection system is unable to detect the intrusion. In this subsection, we prove the correctness of traditional NFA/DFA and StriFA. The correctness here means that StriFA does not cause any false negatives: if the StriFA cannot be matched, then the original NFA/DFA cannot be matched either. Because if a statement is true, its contrapositive is also logically true. So, we only need to prove that if the original NFA/DFA can be matched, the corresponding StriFA can also be matched.

E. VERIFICATION MODULE

Since the SL stream is a highly compressed form of an input stream, part of the information is left out before being sent to StriFA. We have to perform an exact match in the verification module to confirm all the potential matches.

When the StriFA reports a possible match, the verification module is triggered to start the exact match. Instead of match- ing the whole buffer, only part of the input stream needs to be sent to the verification module. Considering the memory consumption of NFA is much less than the corresponding DFA, NFA can be used in the verification module.

If the verification module also reports a match, it means there is a real match from the input stream. Otherwise, the input stream is recognized as normal traffic.

V. ANALYSIS AND OPTIMIZATION

The design tradeoff of StriFA involves maximizing filter rate and minimizing the false alarm rate, while preserving other key performance indicators (i.e., throughput and memory usage) as best as possible. A low filter rate will trigger frequently use of the verification module, degrading the overall throughput. High false alarm rate leads to a waste of time in performing the accurate match.

A. TAG SELECTION

One of the problems for StriFA is how to choose an ap- propriate 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.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 1, January 2017

The character that has the most occurrences in a pattern can extract more SLs. Intuitively, more SLs could express more information for the original pattern. For example, considering pattern P1 = “reference,” Fe (P1)= 22 3 and Fr (P1)= 4. Fe (P1) has more SLs when using “e” as the tag, while Fr (P1) only has one SL with tag “r.”

Definition 1: If the occurrence of character x in pattern Pis > 2, then we say P can be covered by x.

The tag selection strategy is the following heuristic: at each selection, find a new tag that could cover the maximal number of patterns. With the greedy algorithm, one tag set is selected and used to generate the corresponding StriFAs.

B. STRIDE-NEIGHBOR FA

In order to reduce the false alarm rate, we propose a scheme called stride-neighbor FA. A neighbor symbol is defined as the character before the tag, or as the character before the last character of the window if a tag is not found within a window. For instance, in Fig. 11, the characters before the tags are f, r, and c, respectively, in P1 . Characters f, r, and c are used as neighbor symbols to construct a neighbor DFA. Similarly, neighbor symbols c and m are extracted from P2.

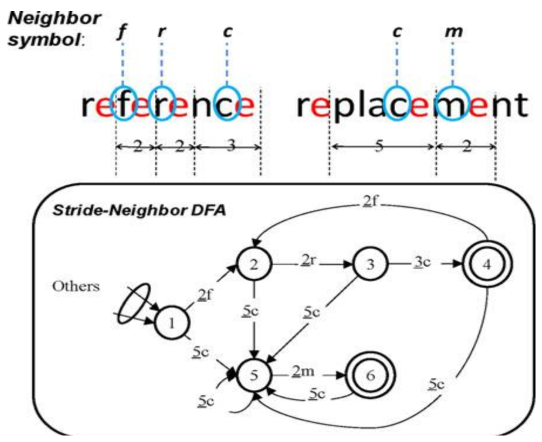

Fig. 11. Stride-neighbor DFA for P1 = “reference” and P2 = “replacement” with tag = “e” and w = 5.

The combination neighbor DFA of P1 and P2 is illustrated at the bottom of Fig. 11.The stride-neighbor DFA shown in Fig. 11 uses a combina- tion of a neighbor character together with the corresponding stride length to make StriDFA labels (e.g., 2l).The basic idea of stride-neighbor FA is to reduce the false alarm rate by starting with a preliminary step of exact matching for neighbor characters and then combining that with using the stride lengths. This is performed at a small memory consumption cost. Generally, the false alarm rate can be reduced by 94.1% when using the stride-neighbor FA.

TABLE I
Worst Case Comparisons of NFA, DFA, StriNFA, and StriDFA

	Oneregularexpression oflength*n*		***m*regularexpression compiledtogether**	
	Processing complexity	**Stora ge cos**	**Processing complexity**	**Storage cost**
NFA	$O(n^2)$	$O(n)$	$O(n^2m)$	$O(nm)$
DFA	$O(1)$	$O(^n)$	$O(1)$	$O(^{nm})$
StriNFA	$O((\underline{n})^2)$	$O(\underline{n})$	$O((\underline{n})^2m)$	$O(\underline{nm})$
StriDFA	$O(1)$	$O(w^n)$	$O(1)$	$O(w^{nm})$

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 1, January 2017

C. PERFORMANCE OF STRIFA

Table I shows that a single regular expression of length n can be expressed by an traditional NFA with O(n) states. When the traditional NFA is converted to traditional DFA, it may generate O(n) states. There is only one outgoing transition for each character from a DFA state, so the processing com- plexity of traditional DFA for each character is O(1), while it is O(n2) for traditional NFA when all n states are active at the same time. Considering StriDFA, there is also one outgoing transition for each input SL from each StriDFA state, so the processing complexity of StriDFA for each input SLis O(1). Suppose windows size = w, then the average stride length is w/2. The average state number is n/w/2, so the processing complexity of StriNFA is O((n)2) when all 2n/w states are,win the worst case, active at the same time.

D. CONVERSION COMPLEXITY OF STRIFA

Denote α as the average fanout in each state. From lines 26 to 36 in Algorithm 1, we can find the time complexity of the recursive procedure in a state is αw . Then, the average time complexity of converting NFA/DFA to StriNFA is nαw . In practice, α is less than 3 in Snort rules (α = 2.6 on average). So the complexity of converting to StriFA is acceptable. Furthermore, all the conversion from traditional NFA/DFA to the corresponding StriNFA/StriDFA can be done off-line. The rule set of the popular NIDS is updated once every one or two months, so we do not need to worry too much about the off-line complexity of StriNFA/StriDFA construction/updates.

VI. CONCLUSION

In this paper, we presented StriFA, a novel regular ex- pression 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. We provided the formal construction algorithm of StriFA that was able to transform an arbitrary set of regex to a StriFA. We also described the method to produce stride length stream so that false positive can be reduced to an acceptable level. Our results showed that our 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.

REFERENCES

[1] M. Roesch. (2001). Snort: The Lightweight Network Intrusion Detection System [Online]. Available: http://www.snort.org/
[2] ClamAntiVirus [Online]. Available: http://www.clamav.net/
[3] Cisco IOS IPS [Online]. Available: http://www.cisco.com
[4] H. Lu, K. Zheng, B. Liu, X. Zhang, and Y. Liu, "A memory-efficient parallel string matching architecture for high-speed intrusion detection," IEEE J. Sel. Areas Commun., vol. 24, no. 10, pp. 1793–1804, Oct. 2006.
[5] S. Dharmapurikar and J. W. Lockwood, "Fast and scalable pattern matching for network intrusion detection engines," IEEE J. Sel. Areas Commun., vol. 24, no. 10, pp. 1781–1792, Oct. 2006.
[6] F. Yu, Z. Chen, Y. Diao, T. V. Lakshman, and R. H. Katz, "Fast and memory-efficient regular expression matching for deep packet inspection," in Proc. ACM/IEEE Symp. ANCS, Dec. 2006, pp. 93–102.
[7] P. Hershey and C. Silio, "Surmounting data overflow problems in the collection of information for emerging high-speed network systems," IEEE Syst. J., vol. 4, no. 2, pp. 147–155, Jan. 2010.
[8] M. Becchi and P. Crowley, "A hybrid finite automaton for practical deep packet inspection," in Proc. ACM Int. CoNEXT, Dec. 2007, pp. 1.
[9] M. Becchiand P. Crowley, "Efficient regular expression evaluation: Theory to practice," in Proc. 4th ACM/IEEE Symp. Architectures Netw.Commun. Syst., Nov. 2008, pp. 50–59.
[10] R. Smith, C. Estan, S. Jha, and S. Kong, "Deflating the big bang: Fast and scalable deep packet inspection with extended finite automata," in Proc. ACM SIGCOMM, vol. 38, no. 4, pp. 207–218, Aug. 2008.
[11] R. Smith, C. Estan, and S. Jha, "XFA: Faster signature matching with extended automata," in Proc. IEEE Symp. Security Privacy, May. pp.187–201 .
[12] N. Hua, H. Song, and T. Lakshman, "Variable-stride multi-pattern matching for scalable deep packet inspection," in Proc. IEEE INFO- COM, pp. 415–423, Apr. 2009.
[13] L. Vespa, N. Weng, and R. Ramaswamy, "Ms-dfa: Multiple-stride pattern matching for scalable deep packet inspection," Comput. J., vol. 54, no. 2, p. 285, 2011.
[14] J. Hopcroft, R. Motwani, and J. Ullman, Introduction to Antomata Theory, Languages and Computation. Reading, MA, USA: Addison Wesley.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 1, January 2017

[15] Z. Li, G. Xia, H. Gao, Y. Tang, Y. Chen, B. Liu, J. Jiang, and Y. Lv, "Netshield: Matching with a large vulnerability signature ruleset for high performance network defense," in Proc. ACM SIGCOMM, Tech. Rep. NWU-EECS-08-07, 2010.
[16] B. C. Brodie, D. E. Taylor, and R. K. Cytron, "A scalable architec- ture for high-throughput regular-expression pattern matching," in Proc. ACM/IEEE ISCA, vol. 34, no. 2, pp. 191–202, Jun. 2006.
[17] D. Ficara, G. Antichi, A. Pietro, S. Giordano, G. Procissi, and F. Vi- tucci, "Sampling techniques to accelerate pattern matching in network intrusion detection systems," in Proc. IEEE ICC, pp. 1–5, May. 2010.
[18] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner, "Al- gorithms to accelerate multiple regular expressions matching for deep packet inspection," in Proc. ACM SIGCOMM, vol. 36, no. 4, pp.339–350, Sep. 2007.
[19] M. Becchi and P. Crowley, "An improved algorithm to accelerate regular expression evaluation," in Proc. ACM/IEEE Symp. ANCS, pp. 145–154, Dec. 2007.
[20] S. Kumar, J. Turner, and J. Williams, "Advanced algorithms for fast and scalable deep packet inspection," in Proc. ACM/IEEE Symp. ANCS, pp.81–92, Dec. 2006.
[21] J. Hopcroft, R. Motwani, and J. Ullman, Introduction to Automata Theory, Languages, and Computation, 2nd ed. Reading, MA, USA: Addison-Wesley, 2001.
[22] K. Thompson, "Regular expression search algorithm," Commun. ACM, vol. 11, no. 6, pp. 419–422, Jun. 1968.