2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing

SRD-DFA: Achieving Sub-Rule Distinguishing with Extended DFA Structure

Gao Xia
Department of Computer Science and Technology, Tsinghua University, Beijing, China

Xiaofei Wang
School of Electronic Engineering, Dublin City University, Ireland

Bin Liu
Department of Computer Science and Technology, Tsinghua University, Beijing, China

***Abstract*—Deep Packet Inspection (DPI) relies highly on regular expression due to its power of description, generalization and flexibility. In DPI, the packet payload is compared against a large number of rules written in regular expression. To achieve high throughput, multiple regular expressions are combined and compiled into one DFA, which leads to two problems: a) State explosion of the DFA; b) Sub-rule distinguishing in the combined ruleset. While the first problem has been extensively studied in the recent years, we did not find any literature which formally discusses the second problem in detail. We formulate it and propose *Sub-Rule Distinguishable DFA (SRD-DFA)*, an extended DFA structure, and develop techniques to distinguish sub-rules from multiple regular expressions upon this structure. SRD-DFA can achieve the same throughput as minimized DFA, since it only incurs little extra memory consumption without extra run-time computation. Experimental results under the L7-filter ruleset and a subset of Snort ruleset demonstrate that our approach achieves 8 to 14 times higher throughput than the DFA without rule combination, while only introducing less than 8.4% overhead of state increase compared to the minimized DFA after rule combination. Furthermore, SRD-DFA can be easily used with advanced DFA compression algorithms to achieve much less memory consumption.**

I. Introduction

Due to its power of description, generalization and flexibility, regular expression has become a fundamental building block for various branches of computer science research areas, such as information retrieval (IR), compiler techniques and network security, etc. In practice, regular expressions can be implemented by either Nondeterministic Finite Automata (NFAs) or Deterministic Finite Automata (DFAs). NFAs require much less space consumption but take more time for pattern matching, while DFAs consume much larger space but lead to much less time for matching.

Generally speaking, the choice between NFAs and DFAs depends on the characteristics and requirements of application scenarios. There are two categories of them: a) Rate Insensitive Applications (RIAs), such as text processing/filtering/classification in IR field, which normally contains less than hundreds of rules but demands extending the expressive power of regular expressions with features such as *capture*, *backreference* and *lookahead*, at the expense of lowering the maximal processing rate; b) Rate Sensitive Applications (RSAs), such as network-based intrusion detection/prevention systems (NIDS/IPS), which could easily reach 10,000 signature matchings in typical systems like Snort [1], a widely used open-source NIDS. To meet the requirement of line processing rate, RSAs rarely support extended features as RIAs do.

From this taxonomy, regular expression engines for RIAs, such as PCRE [2], GNU POSIX regexp [3] and Boost Regex [4], are typically implemented with NFAs, supporting various extended features; while regular expression engines for RSAs should be implemented with DFAs to achieve high matching speed. However, due to historical reasons, Snort is built with PCRE; L7-filter (the Linux Application Protocol Classifier [5]) uses GNU POSIX regexp for regular expression matching. While these libraries are not well designed for supporting RSAs, considerable issues have been raised, such as system line rate degradation [6].

Nowadays, in order to match a great number of regular expressions efficiently, multiple regular expressions are generally combined and compiled into one DFA. However, the DFA combination method natively leads to two problems. One is DFA state explosion. For example, an attempt to combine 88 out of 1450 snort rules into one single DFA could cause memory consumption more than 15 GB [7]. In the recent years, a large number of approaches have been proposed to address the state explosion problem occurring when compiling complex regular expressions into a single DFA [6][7][8][9][10][11][12]. These approaches can achieve 1 to 2 orders of magnitude in memory size reduction, compared to traditional DFAs, without introducing significant extra computation overhead.

The other one is the sub-rule distinguishing problem. The combined and minimized DFA can not identify which one or multiple sub-rules are matched when the DFA arrives at some acceptable state (called the *"final"* state). The NFA alternatives are capable of distinguishing matched sub-rules, but are not time-efficient as discussed above. To achieve sub-rule distinguishing, applications typically execute each rule separately, which we call *"Per-rule Execution Model"*. Obviously, it is a time-inefficient approach to match a large number of regular expressions. In this paper, we propose *Sub-Rule Distinguishable DFA (SRD-DFA)*, an extended DFA structure, and a novel DFA construction and minimization algorithm to generate and minimize SRD-DFAs without losing information required to identify matched sub-rules. In SRD-DFAs, to maintain sub-rule information, we extend finite automaton structure, add extra but small storage for each final

978-0-7695-3929-4/09 $26.00 © 2009 IEEE
DOI 10.1109/DASC.2009.71

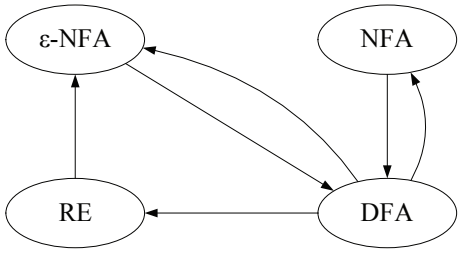


Fig. 1. The translation of four different notations for regular languages [14]

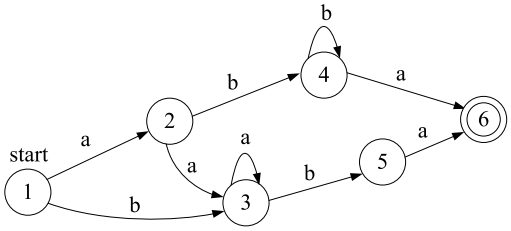


Fig. 2. Minimized but sub-rule undistinguishable DFA

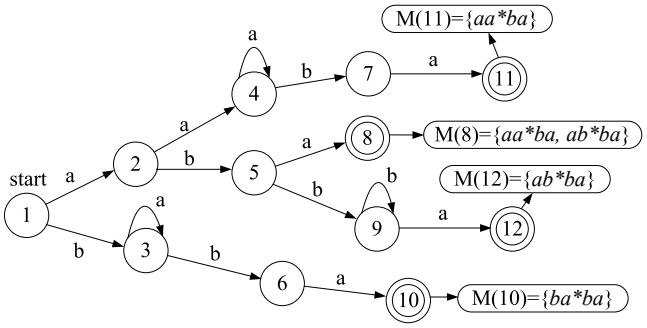


Fig. 3. Sub-rule distinguishable DFA

state, and keep the unique sets of sub-rules distinguishable in all steps of DFA construction and minimization process. SRD-DFA can achieve the same throughput as minimized DFA, since it only incurs little extra memory consumption without extra run-time computation.

Experimental results under the L7-filter ruleset and a subset of Snort ruleset show that our method introduces quite limited overhead with less than 8.4% state increase compared to the traditional DFA minimization algorithm, while achieving 8 to 14 times higher speed than per-rule execution implementation. The results show uniform throughput enhancement under both the L7-filter/Snort rulesets and normal/abnormal traffic, indicating the generality of our solution.

The remainder of the paper is organized as follows. After introducing background and related work in Section 2, we present the problem statement and methodology, and detail our algorithm in Section 3. Then we evaluate our methods through experiments under the L7-filter and Snort rulesets in Section 4. Finally, we conclude our work in Section 5.

II. Background and Related Work

Chomsky hierarchy, which was proposed by Noam Chomsky in 1956, divides formal languages and their grammars into four types: 1) type-0, recursively enumerable languages (unrestricted grammars); 2) type-1, context-sensitive languages (context-sensitive grammars); 3) type-2, context-free languages (context-free grammars); 4) type-3, regular languages (regular grammars) [13]. The automata which recognize the four type languages are turing machine, linear-bounded non-deterministic turing machine, non-deterministic pushdown automata, and finite state automata, respectively.

Regular expression is the notation of regular languages. Since its power of description, generalization and flexibility, regular expressions have been widely used in various aspects of computer science. Typically, regular expressions, which are not evaluated directly, are translated into NFAs or DFAs for matching. Their translation map is given in Fig. 1 [14]. A large number of algorithms have been proposed to construct finite automata from regular expressions. The work in [15] presents a taxonomy of finite automata construction algorithms. One of the construction algorithms was proposed by K. Thompson [16], which is also called "structural induction" in textbooks [14] and considered more readable than Thompson's original paper.

The finite automaton constructed from regular expression by Thompson's algorithm is a ε-NFA. ε-NFAs have almost the same properties with NFAs, except that ε-NFAs have ε-transitions while NFAs do not. In the rest of this paper, we also use the terminology "NFAs" to refer to ε-NFAs for simplicity. Since NFAs tend to be time-inefficient, rate-sensitive applications usually deploy DFA based equipments for regular expression matching. The most popular algorithm to convert NFAs into DFAs is "subset construction" [14].

The DFA generated by subset construction algorithm is usually not state-minimized. For DFAs which accept the same language, there is an equivalent state-minimized DFA. In the process of DFA minimization, states are divided into several equivalent groups; the states within each group are equivalent. Therefore, states in the same group can be merged together into one state so that the number of states is reduced. Many different DFA minimization algorithms have been proposed [17]. A well-known and easy-to-understand algorithm, which is introduced in almost all textbooks, is the table-filling algorithm [14], with a time complexity of $O(n^2)$. The most efficient minimization algorithm ever known is Hopcroft's algorithm, whose time complexity is reduced to $O(nlogn)$ [18]. In order to describe our algorithm more clearly, we will use table-filling algorithm as a basis in the following sections, but our idea can also be integrated with other algorithms including the Hopcroft's algorithm without incurring special overhead.

III. Methodology

A. Problem Statement

We first informally illustrate the problem of sub-rule distinguishing, and then formulate it. Given a set of three regular

expressions {*aa*ba, ab*ba, ba*ba*}, we need a combined DFA which matches the three regular expressions simultaneously. Traditionally, these regular expressions are combined as *aa*ba|ab*ba|ba*ba*. Then after NFA construction, DFA generation and minimization, the final built DFA is in Fig. 2. Obviously, it has only one final state, which is less than the number of sub-rules in the ruleset. So this DFA is sub-rule undistinguishable. It is necessary to construct a DFA such as the one in Fig. 3, which is sub-rule distinguishable. When final state 8 is reached, it indicates that sub-rules *aa*ba* and *ab*ba* are matched; when final state 10 is reached, *ba*ba* matched; and final state 11 for *aa*ba*, final state 12 for *ab*ba*. This example intuitively shows that traditionally combined and minimized DFAs are NOT sub-rule distinguishable. We need to find algorithms to build sub-rule distinguishable DFAs.

We formulate the problem of sub-rule distinguishing as follows: Let $\{RE_i\}(1\leqslant i\leqslant n)$ be a finite set of regular expressions, define $L_i = L(RE_i)$ to be the language described by RE_i, i.e. the set of strings which are accepted by RE_i. Define

$$L = \bigcup_{i=1}^{n} L_i,$$

the problem is whether there is a finite automaton A (either a NFA or a DFA), which accepts language $L(A) = L$, and for each input string $s\in L$, can indicate whether $s\in L_i$ or $s\notin L_i$ $(1\leqslant i\leqslant n)$.

B. Analysis and Solution

The traditional method to build a single DFA for multiple regular expressions is combining regular expressions with operator "OR" ("|") into one single rule and then compiling it into a NFA, which is then converted into a DFA and minimized. Transformations are performed upon the single "big" rule, not to distinguish sub-rules. Therefore, the final built DFA can only indicate whether the "big" rule is matched, but it cannot find out which one or multiple sub-rules are matched.

To solve this problem, we must distinguish each and every sub-rule in all steps of the DFA building process. Following this methodology, given R the finite set of regular expressions, we extend a finite automaton to 6-tuple $(Q, \Sigma, \delta, q_0, F, M)$, where:

- Q is a finite set of states;
- Σ is a finite set of input symbols;
- δ is a transition function: for a NFA, δ is a mapping $Q\times\Sigma\rightarrow 2^Q$, where 2^Q is the power set of Q; for a DFA, δ is a mapping $Q\times\Sigma\rightarrow Q$;
- q_0 is the only one start state;
- $F\subseteq Q$ is a set of final states;
- M is a mapping $F\rightarrow 2^R$, where 2^R is the power set of R.

We propose a method for SRD-DFA generation, which consists of three algorithms:

Algorithm 1 GenerateNFA()

1: For every sub-rule $RE_i(1\leqslant i\leqslant n)$, construct a NFA $N_i(Q_i, \Sigma, \delta_i, q_{0i}, F_i, M_i)(1\leqslant i\leqslant n)$ by Thompson's algorithm [16]; then for every $f\in F_i$, set $M_i(f) = \{i\}$, which means that RE_i is matched when one of the final states of N_i is reached;

2: Construct a "big" single NFA $N(Q, \Sigma, \delta, q_0, F, M)$: add a new start state q_0, and add n ε-transitions from q_0 leading to every start state q_{0i} of N_i; set

$$Q = \bigcup_{i=1}^{n} Q_i,\ F = \bigcup_{i=1}^{n} F_i,\ M = \bigcup_{i=1}^{n} M_i$$

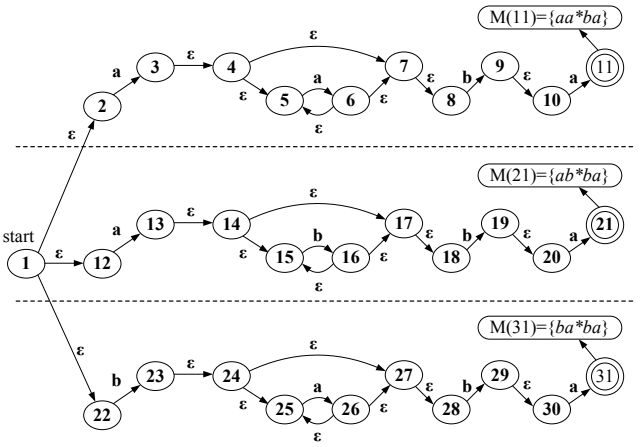


Fig. 4. Sub-rule distinguishable NFA

1) NFA Construction (Algorithm 1): Here we first construct NFAs and store rule IDs in the mapping *M* for every sub-rule, then combine them into a "big" NFA. In this manner, we can identify each sub-rule for each final state. Take the above ruleset {*aa*ba, ab*ba, ba*ba*} as an example, the constructed NFA is shown in Fig. 4. The upper part is a NFA for *aa*ba*, the middle part is for *ab*ba* and the lower part is for *ba*ba*. Note that the *M* elements are the mapping to store sub-rule information.

2) Converting NFA to DFA (Algorithm 2): In the process of this algorithm, each final state of the generated DFA D inherits sub-rule information from the final states in NFA N. Therefore, DFA D can report matched rules when it arrives at any final state. Applying algorithm 2 to the NFA in Fig. 4, we get a sub-rule distinguishable but non-minimized DFA, as in Fig. 5.

3) DFA Minimization (Algorithm 3): This algorithm is a modified Table-Filling Algorithm (TFA), which is a recursive process. The difference between the original TFA and modified TFA is the basis of recursion. The basis in the original TFA is that each of the final states and each of the non-final states are distinguishable. Besides that, the modified TFA claims that two final states are distinguishable if they belong to two

Algorithm 2 ConvertNFAtoDFA()

Construct $D(Q', \Sigma, \delta', q'_0, F', M')$, initialize $Q' = \emptyset$, $\delta' = \emptyset$, $F' = \emptyset$, $M' = \emptyset$. Also set $Q_N = \emptyset$, $Q_D = \emptyset$;
Compute q'_0: For NFA N, compute $\delta(q_0, \varepsilon) = \{q_k\} \subseteq Q$; let q'_0 be $\{q_k\}$, insert $\{q_k\}$ into Q_N, insert q'_0 into Q'; if $\{q_k\} \cap F \neq \emptyset$, set q'_0 as a final state, and $M'(q'_0) = \{\cup M(f) | \forall f \in \{q_k\} \cap F\}$;
while $Q_N \neq \emptyset$ **do**
  pop one element $\{q_k\}$ out of Q_N;
  if $\{q_k\}$ is not set as q'_0, insert it into Q_D;
  for every $\sigma \in \Sigma$ **do**
    compute $\{p_k\} = \cup\delta(q_k, \sigma)$;
    if $\{p_k\} \neq \emptyset$, and $\{p_k\} \notin Q_D$ **then**
      insert $\{p_k\}$ into Q_N;
    end if
  end for
end while
Let $j = 1$;
while $Q_D \neq \emptyset$ **do**
  pop one element $\{q_k\}$ out of Q_D, let q'_j be $\{q_k\}$,insert q'_j into Q';
  if $\{q_k\} \cap F \neq \emptyset$ **then**
    set q'_j as a final state;
    set $M'(q'_j) = \{\cup M(f) | \forall f \in \{q_k\} \cap F\}$;
  end if
  $j = j + 1$;
end while

Algorithm 3 MinimizeDFA()

Initialize a table of all unordered pairs of states of DFA $D(Q', \Sigma, \delta', q'_0, F', M')$ by leaving all entries unmarked;
for every pair (p, q) **do**
  if $p \in F'$ and $q \notin F'$, or $p, q \in F'$ but $M'(p) \neq M'(q)$ **then**
    mark (p, q) to be distinguishable (*"d"* for short);
  end if
end for
repeat
  for every unmarked pair (p, q) and every $\sigma \in \Sigma$ **do**
    if $(\delta(p, \sigma), \delta(q, \sigma))$ is marked *"d"* **then**
      mark (p, q) as *"d"*;
    end if
  end for
until no new entries are marked *"d"*;
For each state q, define $[q]$ as the set of states $\{p | (p, q)$ is not marked *"d"*$\}$;
Construct a new DFA $\widetilde{D}(\widetilde{Q}, \Sigma, \widetilde{\delta}, \widetilde{q_0}, \widetilde{F}, \widetilde{M})$, where

$$\begin{aligned} \widetilde{Q} &= \{[q] | q \in Q'\} \\ \widetilde{\delta}([q], \sigma) &= [\delta'(q, \sigma)] \ (\forall \sigma \in \Sigma) \\ \widetilde{q_0} &= [q'_0] \\ \widetilde{F} &= \{[q] | q \in F'\} \\ \widetilde{M}([q]) &= M'(q) \ (\forall q \in F') \end{aligned}$$

Output DFA $\widetilde{D}$.

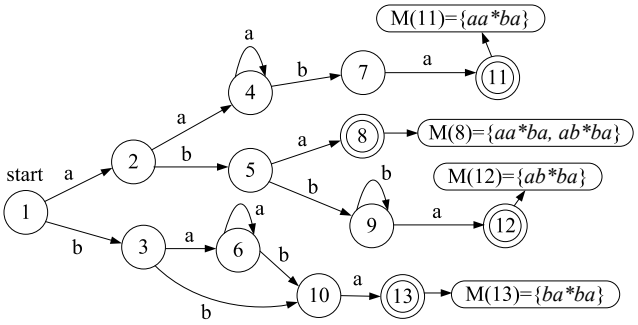
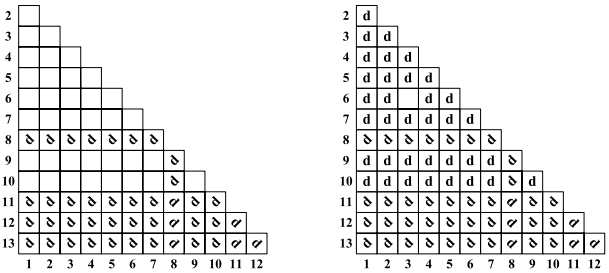


Fig. 5. Non-minimized DFA generated from NFA



(a) Initial status of the table in table-filling algorithm

(b) Final status of the table in table-filling algorithm

Fig. 6. Table-filling algorithm

different sets of sub-rules. Only final states which belong to the same set of sub-rules will be merged in this minimization process. Therefore, sub-rule information can be maintained correctly.

We minimize the DFA in Fig. 5 according to Algorithm 3. The initial table is in Fig. 6(a). Note that "d" marks slanting to the left are set by original TFA and "d" marks slanting to the right are set by modified TFA. After several recursive rounds, we get the final table in Fig. 6(b). It shows that only state 3 and 6 are non-distinguishable and can be merged. Therefore, we finally generate a sub-rule distinguishable DFA in Fig. 3.

The basic idea of SRD-DFA generation is to distinguish final states at the very beginning of NFA construction from a set of regular expressions, and keep them distinguishable when performing DFA minimization. In order to describe our algorithm more clearly, we use table-filling algorithm as a basis in this section, but our idea can also be integrated with other algorithms including the Hopcroft's algorithm without special overhead. Actually, we have adopted this idea to the Hopcroft's algorithm and obtained all of the experimental results in Section 4.

C. Feasibility of Further Compression

Since the information of sub-rules is stored in extra memory of final states, not changing the fundamental structure of DFAs, SRD-DFAs can be compressed by advanced DFA compression algorithms, such as [9][10][11][12], to achieve even less memory consumption. Edge compression algorithms

can be applied to SRD-DFAs without modification, because edges in SRD-DFAs are the same as those in traditional DFAs. State compression algorithms need to be modified a little bit not to merge final states which contain different sub-rule information.

IV. Evaluation

In this section, we evaluate our methods by comparing with the traditional non sub-rule distinguishable DFAs and the DFAs with per-rule matching. Our experiments are carried out with the recent regular expression rulesets from practical DPI applications over real-life network traffic to generate valid results.

A. Experimental Setup

We implement our generation algorithm for SRD-DFAs based on JFlex [19], an open source lexical analyzer generator. To maintain fairness, we also use it to generate the traditional DFAs with regular expression combination. The DFA execution engine in our experiment is written in C++, which loads DFA structures and receives incoming network traffic data in trace file format. The throughput of the matching is calculated by dividing the amount of layer 7 application data by average time of 10 runs. We carry out all the experiments on a desktop PC with Pentium IV 3.80 GHz CPU and 4 GB of memory.

We carefully select two sets of regular expressions for our experiments, both of which are from practical working systems. One is from L7-filter [5], with the ruleset updated on Dec. 18, 2008. Totally 107 regular expressions are selected, which are further divided into 5 groups according to the algorithm proposed in [6]. The other ruleset is from Snort [1], which is composed of 89 regular expressions organized into 3 groups.

The input network traffic data is also from real-life. One data set standing for normal traffic is collected at campus network of Tsinghua University, with an average TCP flow size of 11.74 Kbytes; the other data set representing the abnormal (intrusion) traffic is collected by MIT DARPA project [20], with an average TCP flow size of 46.01 Kbytes. With both data sets, we execute deep packet inspection over the layer 7 application data, after the IP defragmentation and TCP reassembly processes.

B. Results

Table I gives the total numbers of states in the minimized DFAs and our SRD-DFAs. Compared to the minimized DFAs, SRD-DFAs only increase the number of states by no more than 8.4%, for both the L7-filter and Snort rulesets. It demonstrates that we actually achieve the sub-rule distinguishing functionality with rather limited overhead in a ruleset-insensitive way.

Another slot of results given in Table II shows the maximal throughput with the per-rule execution implementation and our SRD-DFA based implementation. Compared with the baseline method of sub-rules distinguishing, our method obtains 8 to 14 times higher throughput, which is significant enough for real deployment considering its limited cost. Compared with traditional minimized DFA, SRD-DFA can achieve the same throughput since it only incurs little extra memory consumption without extra run-time computation. Again, our method shows uniform throughput enhancement under both the L7-filter/Snort rulesets and the normal/abnormal traffic, indicating the generality of our solution.

TABLE I
Comparison of Numbers of States

Rule set	Total # of States in Minimized DFAs	Total # of States in SRD-DFAs	Ratio of State Increase
L7-filter (5 groups)	90,415	97,670	8.02%
Snort (3 groups)	16,062	17,405	8.36%

TABLE II
Comparison of Throughput

Rule set	Implementation	Throughput (Mbps)	
		MIT Trace	Tsinghua Trace
L7-filter	Per-rule Execution Model	42.7	41.9
	SRD-DFAs (5 groups)	374.2	584.9
Snort	Per-rule Execution Model	45.7	45.4
	SRD-DFAs (3 groups)	414.4	382.0

V. Conclusion

In this paper, we formulate and present solutions to the problem of sub-rule distinguishing in the process of combining multiple regular expressions into one single DFA. We have proposed novel algorithms to generate such DFAs which could inherently indicate the matched one or multiple sub-rules at its arrival to any of the final states. Experimental results with practical rulesets have shown that our SRD-DFAs incur less than 8.4% state increase compared to the traditional state-minimized DFAs, but impressively achieve up to 14 times higher maximal throughput than the per-rule execution based implementation without combination. Furthermore, advanced DFA compression algorithms, such as [9][10][11][12], could also be applied to our solution to achieve even less memory consumption.

Acknowledgment

This work is supported by NSFC (60625201, 60873250), the Specialized Research Fund for the Doctoral Program of Higher Education of China (20060003058) and 863 high-tech project (2007AA01Z216, 2007AA01Z468).

References

[1] SNORT Network Intrusion Detection System. http://www.snort.org/
[2] PCRE - Perl Compatible Regular Expressions. http://www.pcre.org/
[3] GNU POSIX Regexp. http://www.gnu.org/software/libtool/manual/libc/Regular-Expressions.html

[4] Boost Regex. http://www.boost.org/doc/libs/1_38_0/libs/regex/doc/html/index.html
[5] J. Levandoski, E. Sommer, and M. Strait. Application Layer Packet Classifier for Linux. http://l7-filter.sourceforge.net/
[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 *ANCS*, 2006.
[7] R. Smith, C. Estan, and S. Jha, "XFA: Faster Signature Matching with Extended Automata," in *IEEE Symposium on Security and Privacy*, May 2008.
[8] R. Smith, C. Estan, S. Jha, and S. Kong, "Deflating the Big Bang: Fast and Scalable Deep Packet Inspection with Extended Finite Automata," in *SIGCOMM*, Aug. 2008.
[9] M. Becchi and P. Crowley, "An Improved Algorithm to Accelerate Regular Expression Evaluation," in *Proceedings of the 2007 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. Orlando, FL: ACM, Dec. 2007.
[10] M. Becchi and P. Crowley, "A Hybrid Finite Automaton for Practical Deep Packet Inspection," in *Proceedings of the International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. New York, NY: ACM, Dec. 2007.
[11] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner, "Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection," in *SIGCOMM*, 2006.
[12] S. Kumar, B. Chandrasekaran, J. Turner, and G. Varghese, "Curing Regular Expressions Matching Algorithms from Insomnia, Amnesia, and Acalculia," in *ANCS*, 2007.
[13] N. Chomsky, "Three Models for the Description of Language," *IRE Transactions in Information Theory 1956, 2:113-124*.
[14] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 2nd ed. Addison-Wesley, 2001.
[15] B. W. Watson, "A Taxonomy of Finite Automata Construction Algorithms," Eindhoven University of Technology, The Netherlands, Computing Science Note 93/43.
[16] K. Thompson, "Regular Expression Search Algorithm," *Comm. ACM 11(6), June 1968, pp. 419-422*.
[17] B. W. Watson, "A Taxonomy of Finite Automata Minimization Algorithms," Eindhoven University of Technology, The Netherlands, Computing Science Note 93/44.
[18] J. E. Hopcroft, "An $n\log n$ Algorithm for Minimizing the States in a Finite Automaton," *Z. Kohavi (ed.) The Theory of Machines and Computations, Academic Press, New York, pp. 189-196*.
[19] JFlex - The Fast Scanner Generator for Java. http://jflex.de/
[20] MIT DARPA Intrusion Detection Data Sets. http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/index.html