bitFA: A Novel Data Structure for Fast and Update-friendly Regular Expression Matching*

Zhe Fu
Department of Automation
Research Institute of Information Technology
Tsinghua University, China
fu-z13@mails.tsinghua.edu.cn

Shijie Zhou
Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, USA
shijiezh@usc.edu

Jun Li
Research Institute of Information Technology
Tsinghua National Lab for Information Science and Technology
Tsinghua University, China
junl@tsinghua.edu.cn

ABSTRACT

This paper proposes bitFA, a novel data structure optimized for fast and update-friendly regular expression matching. bitFA leverages fast bit manipulation, instruction-level parallelism and bitmap compression techniques to achieve 5x to 25x acceleration compared to existing NFA or DFA based regular expression matching methods.

CCS CONCEPTS

• **Networks → Deep packet inspection**; **Firewalls**;

KEYWORDS

Regular Expression; Finite Automaton; Bit Manipulation

1 INTRODUCTION

Due to the rich expressiveness and powerful flexibility, regular expressions become more and more popular and play an important role in today's network security systems. Nondeterministic Finite Automaton (NFA) and Deterministic Finite Automaton (DFA) are the two most common methods to perform regular expression matching. NFA has a compact data structure, but the nondeterministic state transitions of NFA make it hard to guarantee the worst-case performance. In contrast, DFA requires only one state transition look-up per input character, so the performance is deterministic, and DFA is becoming the preferred method for network security systems. However, there are two main drawbacks of DFA which limit its further applications in practice:

First, the high performance of DFA is at the cost of huge space consumption, *i.e.*, the well-known state explosion problem, where the size of DFA often grows exponentially with the increase of the size of regular expressions. Second, with the development of SDN and NFV, the enforcement of security policies becomes more and more dynamic. The lengthy compiling time of DFA makes it hard to meet the requirements of frequent update of regular expression rulesets in real-world use.

	a	b	c	d	e
⓪	11000	10000	10000	10000	10000
①	01000	00100	00000	00000	00000
②	00100	00100	00110	00100	00100
③	00000	00000	00000	00001	00000
④	00000	00000	00000	00000	00000

Figure 1: a raw example of bitFA compiled from "ab.*cd" (without further optimization)

In this paper, we propose bitFA to overcome these shortcomings. bitFA implements fast regular expression matching as well as fast FA compilation, and it is suitable for the scenarios with frequently updated rules. bitFA takes advantage of fast bit manipulation, instruction-level parallelism, and bitmap compression. Evaluations demonstrate that bitFA achieves about 12x acceleration in average compared to NFA and DFA based methods when taking both preprocessing and matching time into account.

2 DESIGN

The main drawback of NFA could be ascribed to the multiple active states and uncertain memory accesses per input character. Similar to DFA, our design, bitFA, is built based on NFA; however, bitFA is distinctly different from DFA.

2.1 Compiling bitFA

After an NFA is built from a set of regular expressions, bitFA further encodes this NFA into multiple bit-vectors. For an NFA with N states and the alphabet Σ, a bitFA usually has $N \times |\Sigma|$ bit-vectors, and every bit-vector is a N-bit vector where each bit indicates whether the corresponding state will be active or not. More precisely, for bitFA(i, j), which stands for the bit-vector from the ith of the N states and jth of the alphabet, if the kth bit is 1, it means that if state i of an NFA is active and the input character is the jth of the alphabet, state k will be active. Figure 1 presents an intuitive view the bitFA compiled from the regular expression "ab.*cd". In this case, the size of the alphabet is 5, and the NFA has 5 states, so there are $5 \times 5 = 25$ bit-vectors and each is a 5-bit vector. Taking bitFA(2, 2)

*This work was supported by the National Key Research and Development Program of China (2016YFB1000102) and the National Key Technology R&D Program of China (2015BAK34B00).

SIGCOMM Posters and Demos '17, August 22–24, 2017, Los Angeles, CA, USA
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5057-0/17/08…$15.00
https://doi.org/10.1145/3123878.3132011

Table 1: Experimental results of three methods on different rulesets

rulesets (# of REs)			snort1 (24)	snort2 (34)	bro (217)	cisco (733)	dotstar (300)	range (300)
NFA	time (s)	preprocesing	0.108	0.124	0.266	59.207	2.957	1.367
		matching	11.294	11.656	85.955	292.971	107.291	57.526
	memory usage (KB)		162.520	250.882	605.470	4035.996	3165.721	3314.318
DFA	time (s)	preprocesing	16.449	24.527	35.934	\	\	\
		matching	0.047	0.048	0.086	\	\	\
	memory usage (KB)		8535.040	9988.096	6689.792	\	\	\
bitFA	time (s)	preprocesing	0.168	0.216	0.493	60.719	4.131	2.593
		matching	1.222	1.196	3.042	1.287	2.219	1.406
	memory usage (KB)		1203.646	1843.654	4413.894	30069.300	23212.102	24041.348

as an example, if state 2 is active and the input character is c, then state 2 and 3 will be active.

The procedure of bitFA compiling only needs a complete traversal of the previously-built NFA, so it is much faster and more concise than that of DFA.

2.2 State Transition in bitFA

The state transition in bitFA could be abstracted as a multiplication of a bit-vector and a bit-matrix. Additionally, a N-bit vector cur_bv is used to represent the currently active states. In the beginning, only state 0 is active, so the first bit of cur_bv is always set to 1 initially. After reading the input character j, the jth column of the bitFA is picked out, which is a $N \times N$ bit-matrix. We use formula (1) to iteratively compute state transitions in bitFA:

$$cur\_bv_{(1\times N)} = cur\_bv_{(1\times N)} \cdot bitFA(:, j)_{(N\times N)} \tag{1}$$

In the example in Fig. 1, suppose the active states are 0 and 2, *i.e.* $cur\_bv = [1\ 0\ 1\ 0\ 0]$. If the input character is c, the active states after state transitions are calculated as:

$$\begin{bmatrix}1 & 0 & 1 & 0 & 0\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & 1 & 0\\ 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0\end{bmatrix} = \begin{bmatrix}1 & 0 & 1 & 1 & 0\end{bmatrix}$$

The result means currently the state 0, 2 and 3 are active after reading c. To decide whether a state is accepted or not, the cur_bv is intersected by a N-bit accepted-states vector which is set before the matching procedure. If the intersection result is not zero, then a match happens.

2.3 Bit Manipulation Optimization

The multiplication of a bit-vector and a bit-matrix can be simplified to a bitwise OR operation of multiple bit-vectors from the bit-matrix, and meanwhile, the decision of accepted states is a bitwise AND operation. Recent popular processors already have several built-in fast instructions for bitwise operations, including the POPCNT instruction, which is to compute the number of ones in a word, and the LZCNT, which is to count the number of leading zero bits. These instructions have a throughput as high as one operation per CPU cycle, which significantly relieves the inherent problem of NFA. Besides, owing to its sparsity of data structure, the raw bitFA compiled from an NFA can be efficiently compressed to reduce the memory consumption. Several bitmap compression techniques (such as Roaring Bitmaps [5], WAH [7], *etc.*) could enable the bitwise OR and AND operation between compressed bit-vectors without decompression. All these bit manipulation optimizations improve the performance of bitFA and meanwhile reduces both of the computation and memory overhead.

3 IMPLEMENTATION AND EVALUATION

Experiments are performed on a workstation with Intel Core i7-4790 CPU (BMI1 supported). Regular Expression Processor [1] is used as the baseline of NFA and DFA based methods. Four rulesets picked from Snort, Bro and Cisco and two synthetic rulesets (dotstar and range in Table 1) are tested, while a traffic PCAP file (about 6.2 MB) dumped from the campus network is treated as the input data for different matching engines.

From Table 1 we can see that the preprocessing procedure of bitFA is almost as fast as that of NFA, while DFA requires much more time to build its data structure. Matching engines with less preprocessing time are more applicable to frequently changing ruleset. For large rulesets, the DFA states grow rapidly to more than 1 million, leading to compiling failures for DFA based methods. On the other hand, bitFA achieves far higher matching speed than NFA. Although DFA's matching procedure is the best among these three methods, the non-ignorable preprocessing time makes DFA based methods unsuitable for the situations where regular expressions are frequently updated or the ruleset is very large. We also measure the space cost of these three methods and come to a conclusion that the memory consumption of bitFA lies between that of NFA and DFA. Other DFA based methods (such as D^2FA [4], HybridFA [2], regular expression partition and grouping [3][6], *etc.*) focus on reducing the memory consumption of DFA, which requires more preprocessing time.

4 CONCLUSIONS AND FUTURE WORK

To accelerate regular expression matching especially for frequently-updated rules, this work proposes bitFA, a novel data structure which makes full use of bit-level optimizations. bitFA is demonstrated to be a promising design since it achieves 5x to 25x acceleration compared to existing algorithms. Our future work includes more efficient state encoding and bit manipulation to obtain better matching performance of bitFA.

REFERENCES

[1] Michela Becchi. *Regular Expression Processor.* http://regex.wustl.edu.

[2] Michela Becchi and Patrick Crowley. 2007. A hybrid finite automaton for practical deep packet inspection. In *Proceedings of the 2007 ACM CoNEXT conference*. ACM, 1–12.

[3] Zhe Fu, Kai Wang, Liangwei Cai, and Jun Li. 2014. Intelligent grouping algorithms for regular expressions in deep inspection. In *Proceedings of the 23rd International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 1–8.

[4] Sailesh Kumar, Sarang Dharmapurikar, Fang Yu, Patrick Crowley, and Jonathan Turner. 2006. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. In *ACM SIGCOMM Computer Communication Review*, Vol. 36. ACM, 339–350.

[5] Daniel Lemire, Gregory Ssi-Yan-Kai, and Owen Kaser. 2016. Consistently faster and smaller compressed bitmaps with roaring. *Software: Practice and Experience* 46, 11 (2016), 1547–1569.

[6] Kai Wang, Zhe Fu, Xiaohe Hu, and Jun Li. 2014. Practical regular expression matching free of scalability and performance barriers. *Computer Communications* 54 (2014), 97–119.

[7] Kesheng Wu, Ekow J Otoo, and Arie Shoshani. 2006. Optimizing bitmap indices with efficient compression. *ACM Transactions on Database Systems (TODS)* 31, 1 (2006), 1–38.