JCSE Journal of Computing Science and Engineering

Regular Paper
Journal of Computing Science and Engineering,
Vol. 8, No. 2, June 2014, pp. 78-86

A Regular Expression Matching Algorithm Based on High-Efficient Finite Automaton

Jianhua Wang*, Lianglun Cheng, and Jun Liu
Faculty of Automation, Guangdong University of Technology
123chihua@163.com, llcheng@gdut.edu.cn, liujun7700@163.com

Abstract
Aiming to solve the problems of high memory access and big storage space and long matching time in the regular expression matching of extended finite automaton (XFA), a new regular expression matching algorithm based on high-efficient finite automaton is presented in this paper. The basic idea of the new algorithm is that some extra judging instruments are added at the starting state in order to reduce any unnecessary transition paths as well as to eliminate any unnecessary state transitions. Consequently, the problems of high memory access consumption and big storage space and long matching time during the regular expression matching process of XFA can be efficiently improved. The simulation results convey that our proposed scheme can lower approximately 40% memory access, save about 45% storage space consumption, and reduce about 12% matching time during the same regular expression matching process compared with XFA, but without degrading the matching quality.

Category: Smart and intelligent computing

Keywords: Regular expression; Matching algorithm; High-efficient; Deterministic finite automaton

I. INTRODUCTION

Deep packet inspection is a detection technology regarding packet content and plays an increasingly important role in modern network intrusion detection systems (NIDSes) [1]. Due to its better expressiveness and flexibility, regular expression matching has been widely used in NIDSes. However, regular expression matching engenders much convenience to NIDSes, but also simultaneously brings significantly high computation and storage complexity, thereby prohibiting its wide usage in NIDSes applications. Thereby, how to design a regular expression matching algorithm that can achieve both time and space efficiency is of great challenge.

Deterministic finite automaton (DFA) and nondeterministic finite automaton (NFA), as two main finite automata, are typically used to implement regular expression matching. Although DFA has fast matching speed and deterministic matching performance, it causes a memory explosion problem. While NFA requires less memory, it suffers from slow matching speed and nondeterministic matching performance [2]. In a high-speed or resource-restricted environment, because both DFA and NFA automata require excessive memory, time or per-flow state, neither of them is suitable to implement a regular expression matching operation in a high-speed or resource-restricted setting. Recently, many research works have focused on either improving the matching speed or reducing the memory access spent in a regular expression matching.

In accelerating the matching speed of a regular expres-

Open Access http://dx.doi.org/10.5626/JCSE.2014.8.2.78 http://jcse.kiise.org

Received 24 April 2013; **Revised** 30 October 2013; **Revised** 25 January 2014; **Accepted** 10 March 2014
*Corresponding Author

 pISSN: 1976-4677 eISSN: 2093-8020

sion aspect, Kumar et al. [3] proposed a D2FA regular expression matching algorithm in order to accelerate the matching speed. Becchi and Cadambi [4] suggested a DFA regular expression matching algorithm based on state fusion, making use of a migration edge marking method in order to integrate a multiple non-equivalent state for guaranteeing DFA performance. In Brodie method, Brodie et al. [5] presented a new way that can increase the throughput of regular expression matching by expanding the alphabet set; however, this method could result in an exponential growth of memory requirement in the worst case. Moreover, although sampling techniques are introduced in order to accelerate regular expression matching [6], it does not suit all kinds of regular expression. Kumar et al. [7] applied default transitions in order to improve average performance. In eliminating the storage space explosion of the DFA aspect, Yu et al. [8] proposed state compression techniques based on mDFA in order to deflate state explosion. Kumar et al. [9] used heuristics in order to eliminate the insomnia problem existing in the regular expression matching algorithm. In the method of Wang et al. [10], a chain-based DFA deflation method for fast and scalable regular expression matching by using ternary content addressable memory (TCAM) is proposed to effectively deflate the DFA size. In order to prevent state explosion, Smith et al. [11] introduced a partial NFA to DFA conversion. In [12], a new art work extended finite automaton (XFA) based on auxiliary memory was introduced in order to reduce the DFA state explosion as well as to achieve a great reduction rate. However, XFA is not suitable for real-time applications in networks due to its significant startup overhead, high memory access and big storage space in regular expression matching.

This paper aims to solve the problem of high memory access and big storage space and long matching time during the regular expression matching process of XFA by proposing a new regular expression matching algorithm based on high-efficient finite automaton (HFA). We use some extra judging instruments at the starting state in order to reduce any unnecessary transition paths and eliminate any unnecessary state transitions. Consequently, the high memory access consumption and big storage space consumption and long matching time in the regular matching process of XFA can be efficiently improved. The simulation results convey that our proposed scheme can lower 40% memory access, save 45% storage space consumption, and reduce 12% matching time in the same regular expression matching process compared with XFA, but without degrading the matching quality.

The rest of this paper is organized as follows. In Section II, the state space explosion of DFA is introduced. Then the proposed HFA method is presented in Section III. The simulation result and analysis of the proposed scheme compared with XFA methods is presented in Section IV. In Section V, we provide some conclusions.

II. STATE-SPACE EXPLOSION OF DFA

State-space explosion is a main problem existing in combined DFA, which seriously influences its wide application of DFA. In this section, we mainly illustrate some related knowledge with regard to the DFA, including DFA definition, the reasons for state-space explosion of DFA, a general method to solve state-space explosion problem—XFA, and some problems existing in XFA.

A. DFA Definition

DFA can be described as $DFA = (Q, \Sigma, \delta, q_0, F)$, where Q represents states; Σ stands for input symbols; δ represents transition function; q_0 stands for start state, F represents accepting states and $F \subseteq Q$. For each state $q \in Q$, we define $paths(q)$ as the set of paths from q_0 to q; further, $paths(q)$ may be infinite in the emergence of cycles. Fig. 1 is the DFA for the regular expression of (.*ab.*cd). Fig. 2 is the DFA for the regular expression of (.*ef.*gh). In the regular expression of (.*ab.*cd), where $Q = \{S, K, P, Q, R\}$, $\Sigma = \{a, b, c, d\}$, $q_0 = S$, $\delta(S, a) = K$ and $\delta(K, b) = P$, and so on, $F = \{R\}$. In the regular expression of (.*ef.*gh), where $Q = \{N, M, J, V, C\}$, $\Sigma = \{e, f, j, h\}$, $q_0 = N$, $\delta(N, e) = M$ and $\delta(M, f) = J$, and so on, $F = \{C\}$.

B. Reasons for State-Space Explosion of DFA

In a DFA, a path is ambiguous when there is an ambiguous state in this path in which a state is ambiguous, given any finite sequence and state. Although the set of paths to this state exist in the same path suffix, not all can be expressed solely by the finite sequence. That is to say, an ambiguous state may usually be obtained by many distinct sequences. During the combining process of automata, the interaction between states in both ambiguous and unambiguous paths above will lead to a state-space explosion of DFA. The specific generating process

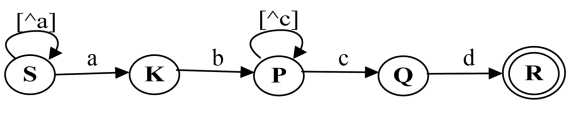

Fig. 1. The deterministic finite automaton (DFA) for the regular expression of (.*ab.*cd).

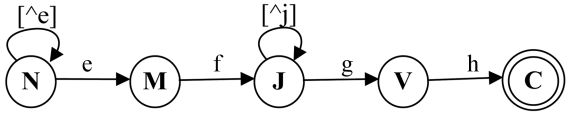

Fig. 2. The deterministic finite automaton (DFA) for the regular expression of (.*ef.*gh).

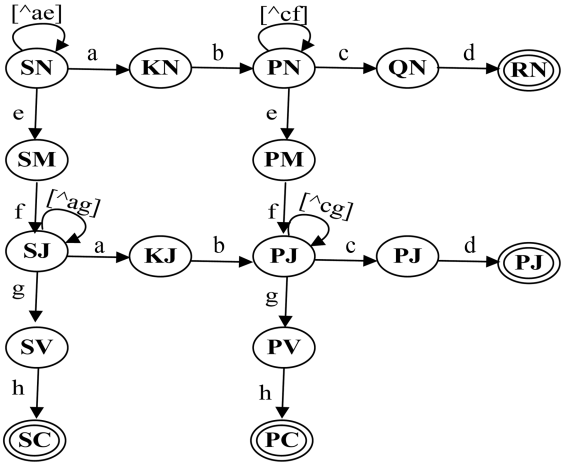

Fig. 3. The combined deterministic finite automaton (DFA) for the regular expression of (.***ab*.***cd*) and (.***ef*.***gh*).

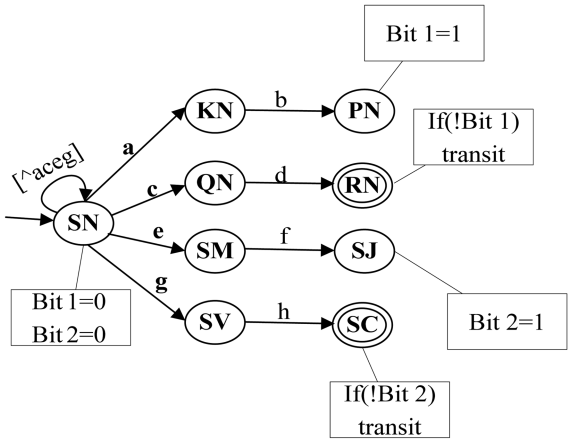

Fig. 4. Combined extended finite automaton (XFA) for the regular expression of (.***ab*.***cd*) and (.***ef*.***gh*).

is that the unambiguous states existing in the prefix of a path in one automaton can be replicated when it is combined with ambiguous states in a path in another automaton during the process of combination. The combined automata need to track the progress in matching both the unambiguous path and the ambiguous path independently [13], which will result in a state-space explosion of DFA. The amount of replication observed depends on the following two aspects: how extreme and pervasive the ambiguity is in the two important automata and how much interaction occurs between them. That is to say, automaton with limited levels of ambiguity can introduce comparatively small amounts of replication, whereas a path of infinite length can cause an entire automaton to be copied and therefore directly lead to exponential replication. Fig. 3 is the combined DFA for the regular expression of (.***ab*.***cd*) and (.***ef*.***gh*) [11].

C. XFA Method

In order to solve the problem of state-space explosion above, an XFA is proposed for this problem [12]. The XFA uses some auxiliary variables to record some matched paths, which can save lots of state-space as well as improve the matching speed. Fig. 4 is the XFA for the combined regular expression of (.***ab*.***cd*) and (.***ef*.***gh*).

From Fig. 4, we can clearly see that XFA uses an auxiliary variable (Bit 1) in order to record the matched path (ab) in the regular expression of (.***ab*.***cd*); further, it uses another auxiliary variable (Bit 2) in order to record the matched results (ef) during the regular expression process of (.***ef*.***gh*). Hence, XFA can use only 2 bits auxiliary variables and 9 states in order to fully express the combined regular expression of (.***ab*.***cd*) and (.***ef*.***gh*), whereas DFA has 16 states in Fig. 3, realizing the eliminating state space explosion of DFA.

D. Problems Existing in XFA

Although XFA can eliminate a DFA state explosion by adding auxiliary variables to record some completed matching results, at the same time, it also adds a great amount of redundancy migration side, which not only increases the number of storage space, but also leads to many unnecessary state transitions. Moreover, it adds memory accesses and matching time in the regular expression matching of XFA, therefore limiting its practical application. Fig. 5 is the XFA for the regular expression of (.***abc*.***def*). Fig. 6 is the XFA for the regular expression of (.***abcde*.***fghnm*).

From Figs. 5 and 6, we can see clearly that XFA has lots of redundancy migration sides, such as $d(Q \rightarrow S)$, $d(R \rightarrow S)$, $a(T \rightarrow Q)$ in Fig. 5 and $t(1 \rightarrow 6)$, $t(2 \rightarrow 6)$, $t(7 \rightarrow 1)$ in Fig. 6. These redundancy migration sides above not only increase storage space requirements, but also lead to many unnecessary state transitions, such as

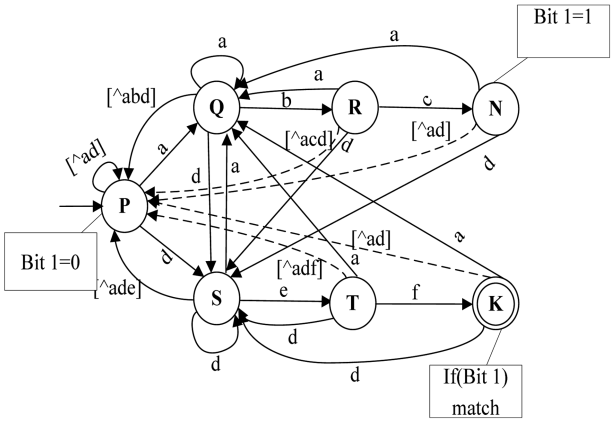

Fig. 5. Extended finite automaton (XFA) for the regular expression of (.***abc*.***def*).

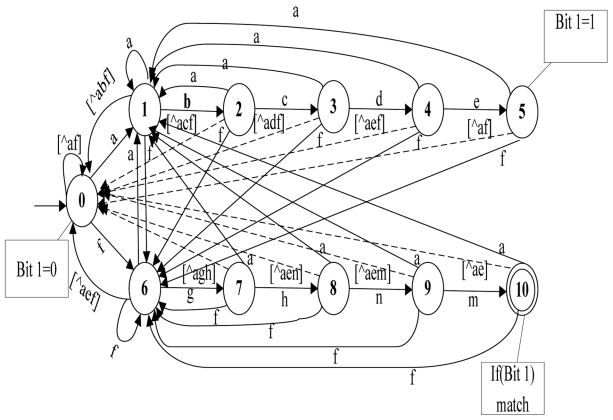

Fig. 6. Extended finite automaton (XFA) for the regular expression of (.**abcde*.**fghnm*).

state transitions $Q \xrightarrow{d} S$, $R \xrightarrow{d} S$, $T \xrightarrow{a} Q$ in Fig. 5 and $1 \xrightarrow{t} 6$, $2 \xrightarrow{t} 6$, $7 \xrightarrow{t} 1$ in Fig. 6. Consequently, it results in additional high memory accesses, big storage space and long matching time in the regular expression matching of XFA, thereby limiting its application.

III. PROPOSED SCHEME

A. Basic Idea of HFA Method

In order to solve the problems existing in the regular expression matching of XFA above, we propose a new regular expression matching algorithm in this paper, which is based on HFA. The basic idea of HFA is that, we add some extra judging instruments at the starting state in order to reduce the storage space of the redundancy migration side as well as to eliminate many unnecessary state transitions. The specific realizing process for HFA is that it makes full use of the current state, judging instruments, auxiliary variable value and the input character for determining which next state to shift and which auxiliary variable to update during the regular expression matching process of HFA. As a result, the problems of high memory access and big storage space and long matching time existing in the regular expression matching process of XFA can be efficiently improved, which greatly improves the overall matching performance of XFA.

B. Realization Process of HFA Algorithm

The working process of our proposed HFA method is comprised of some parts. Fig. 7 is the working flow composition of the HFA algorithm.

From Fig. 7, we can see that our proposed HFA algorithm is primarily made up of four parts: initialization, HFA_building, HFA_matching, and read result. The function of initialization mainly makes some necessary prepa-

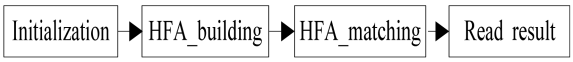

Fig. 7. Working flow composition of high-efficient finite automaton (HFA) algorithm.

```
Procedure HFA_building{
  Input : XFA xfa
  Output : HFA hfa
  (1) initialization;
  (2) read each side s at starting state of XFA←Read_stateside(xfa,s)
  (3) add judging instruments at branching side ←Add_judgingIn_struments (xfa,s)
  (4) delete migration side ←Delete_migration side (xfa,s)
  (5) output(hfa)
  }
```

Fig. 8. Part pseudocode of procedure HFA_building. HFA: high-efficient finite automaton, XFA: extended finite automaton.

```
Algorithm HFA_match{
  Input : regular expression, data stream
  Output : matching expression
  (1) initialization
  (2)select matching path←Search_path(hfa,state,variable,input))
  (3)if (select condition==success)then
  (4)go to (7)
  (5)else
  (6)go to (2)
  (7)search matching path←Search_path(hfa,state,input))
  (8)if (reaches predetermined path)then
  (9)record auxiliary variable
  (10)else
  (11)go to (2)
  }
```

Fig. 9. The part pseudocode of procedure HFA _matching. HFA: high-efficient finite automaton, XFA: extended finite automaton.

rations before the program begins to run. HFA_building primarily builds up a HFA from an XFA. It mainly includes the addition of some extra judging instruments at the starting state and also reduces redundancy migration side operations. Fig. 8 is the part pseudocode of procedure HFA_building( ).

Read result operation mainly reads the matching result from the HFA. The function of HFA_matching mainly takes the matching operations according to the regular expression and input data stream. The main realization process of HFA_matching includes the following two operations.

Operation 1: Select which branching path to execute according to the current staring state, judging instruments with auxiliary variable value and input character. If some paths are met, jump Operation 1 to go on; otherwise, jump Operation 2 to go on.

Operation 2: Determine the matching path according

to the current state and the input character. When the matching process reaches our predetermined path, the algorithm will set an auxiliary variable value as 1 to record it and take the corresponding process operate; then, jump Operation 2 to go on; otherwise, jump Operation 1 to go on.

Fig. 9 is the part pseudocode of procedure HFA_matching( ).

C. Case Study for HFA Algorithm

Take the regular expression (.*abc.*def), for example, to illustrate the whole realization process for our proposed HFA method above.

Step 1: Build HFA from XFA according to the regular expression of (.*abc.*def), which is shown in Fig. 10;

Step 2: Set the initial value of auxiliary variable (Bit1) as 0 at the starting state;

Step 3: Judge whether to execute the $P \xrightarrow{a} Q$ path according to the current staring state (P), judging instruments for auxiliary variable value (if(!Bit 1)) and input character (a);

Step 4: Decide which next state to transit through the current state (Q) and the input character (a or b). If the input character is b, transit to R state; if a, transit to Q state; if neither a or b, transit to P state. When the matching process reaches the $R \xrightarrow{c} N$ path, the algorithm will set the auxiliary variable (Bit = 1) in order to record it. When the matching process finishes given the regular expression, our algorithm executes the corresponding matching operation and updates the corresponding auxiliary variables.

Step 5: Go on to the matching operation above until the end of the matching work;

Step 6: Read the matching results from the matching process of HFA.

Fig. 10 is the HFA matching process for the regular expression of (.*abc.*def). Fig. 11 is the HFA matching process for the regular expression of (.*abcde.*fghnm).

From Figs. 10 and 11, we can clearly observe that our proposed HFA method includes 7 states and only 19 migration edges in the regular expression of (.*abc.*def); moreover, it includes 11 states and only 31 migration edges in the regular expression of (.*abcde.*fghnm), respectively. Compared with the XFA method in Figs. 5 and 6, our proposed HFA method not only can save 7 migration edges in the regular expression of (.*abc.*def) and 11 migration edges in the regular expression of (.*abcde.*fghnm), respectively, but can also reduce many unnecessary state transitions in both regular expressions above. The main reason for this effect is that some extra judging instruments are added at the starting state in our

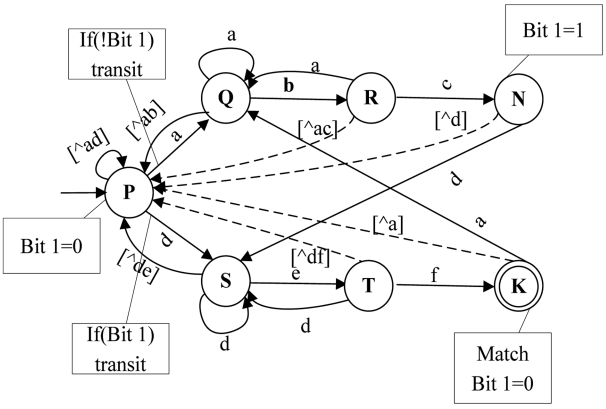

Fig. 10. High-efficient finite automaton (HFA) for the regular expression of (.*abc.*def).

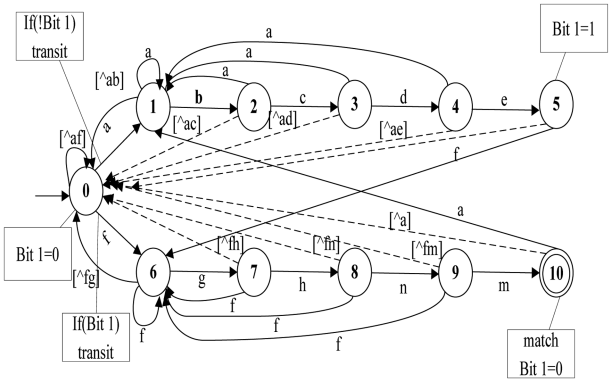

Fig. 11. High-efficient finite automaton (HFA) for the regular expression of (.*abcde.*fghnm).

HFA method in order to reduce any unnecessary transition size as well as eliminate any unnecessary state transitions, which can lower memory access, save storage space and reduce matching time. As a result, the problems of high memory access and big storage space and long matching speed in the regular expression matching of XFA can be efficiently improved.

IV. SIMULATION RESULTS AND ANALYSIS

In order to verify the effectiveness of our proposed method above, we conduct some experiments. Our designed experiments mainly include four parts: simulation environment, lower memory access, save storage space, and reduce matching time.

A. Simulation Environment

The simulation environment is conducted on Microsoft

Windows 7 operating systems, AMD-A63420M 4-core CPU processor, 2 G memory, 500 G hard disk, and VC++ 6.0 software tool. The testing regular rule set used in our experiment comes from Snort, whose snapshot was released on January 1, 2009. We extract all FTP rule sets and part HTTP rule sets from Snort as our testing object and grab the FTP data packet and HTTP data packet to match the data. Snort2Bro tool is used to change the Snort rule format into a Bro rule format as well as produce 82 FTP rules and 443 HTTP rules.

In our experiment, we compare our proposed HFA algorithm with the DFA algorithm and XFA algorithm, and evaluate three general important indicators: memory access, storage space, and matching time. As to the DFA algorithm, we produce a single DFA by making use of each regular expression. The combined DFA is obtained by merging the DFA algorithm. For the XFA algorithm, we generate a single XFA by adding some configuration information in the rules, such as notes, variable mapping, etc., and then use the merging XFA algorithm to produce a combined XFA. Our proposed HFA algorithm is produced based on XFA, but with the addition of some extra judging instruments at the starting state in order to reduce any unnecessary transition paths and eliminate any unnecessary state transitions. The detailed realization on the proposed scheme is shown in the proposed scheme part of this paper above. Table 1 is the main tool and parameters of our experiment, including the rule set name, the testing number of the rule set, compared indicator, compared methods and so on.

In order to reduce data randomness, we obtain our experimental data based on the average value of 5 testings.

B. Lower Memory Access

In our HFA method, reducing the number of unnecessary state transitions by adding some extra judging instruments at the starting state can save a great amount of memory access in the same matching process compared with XFA. Fig. 12 is the memory access consumption comparison result for three different matching methods.

Table 1. Experimental tools and parameters

Parameter	Value
Testing data source	Snort
Rule set in Snort	FTP, HTTP
Compared indicator	Memory access, Storage space, Matching time
Compared methods	DFA, XFA, HFA
Conversion tool	Snort2Bro
FTP rules number	82
HTTP rules number	443

DFA: deterministic finite automaton, XFA: extended finite automaton, HFA: high-efficient finite automaton.

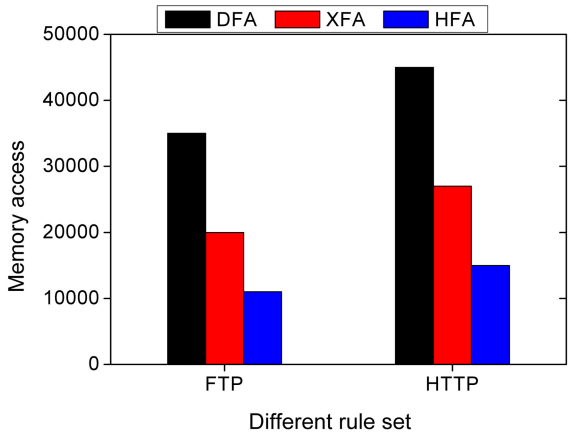

Fig. 12. Memory access consumption comparison for three methods. DFA: deterministic finite automaton, XFA: extended finite automaton, HFA: high-efficient finite automaton.

From Fig. 12, we can observe that our proposed algorithm has a good memory access saving performance in three different matching algorithms above, reaching approximately 65% in memory access saving compared with that of DFA and reducing about 40% in memory access saving compared with that of XFA. The main reason for it is that our proposed method uses some judging instruments at the starting state in order to narrow and determine the next execution area. Hence, many unnecessary state transitions can be eliminated, which can save a great amount of memory access.

C. Save Storage Space

In this subsection, we evaluate the storage space performance of our proposed HFA scheme compared with the DFA and XFA methods from four indicators: state number, migration edge number, instruction number, and the total storage space. Table 2 is the storage space consumptions conditions for three methods in two different rule sets.

From Table 2, we can observe that our proposed HFA algorithm has superior results in three matching algorithms above in two different rule sets. DFA takes up the biggest storage space consumption, reaching about 4052 kB and 526324 kB in two different rule sets, respectively. XFA then followed, getting to approximately 402 kB and 6794 kB. However, our proposed scheme only takes up 223 kB and 3685 kB, which saves 93% in storage space compared with that of DFA and 45% in storage space compared with that of XFA. The main reason for this effect lies in the use of judging instruments at the starting state in our proposed method, which can reduce a great

Table 2. Storage space comparison in matching for three methods

Storage space parameter	State number	Migration edge number	Instruction number	Total storage space (kB)
FTP				
DFA	5024	1355324	Null	4052
XFA	465	10442	3314	402
HFA	465	5961	4064	223
HTTP				
DFA	>675476	>171255362	Null	>526324
XFA	6750	1687554	700	6794
HFA	6750	1017585	1012	3685

DFA: deterministic finite automaton, XFA: extended finite automaton, HFA: high-efficient finite automaton.

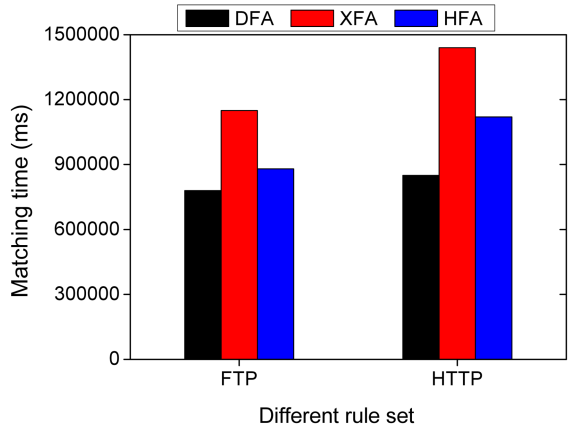

Fig. 13. Matching time comparison for three methods. DFA: deterministic finite automaton, XFA: extended finite automaton, HFA: high-efficient finite automaton.

amount of storage spaces of unnecessary state transitions and migration edges. Note that the Null in Table 2 implies no instruction number.

D. Reduce Matching Time

In this subsection, we evaluate the average matching time for three matching algorithms above in the FTP and HTTP rule sets. The specific experimental results are shown in Fig. 13.

From Fig. 13, we can see clearly that, compared to the XFA method, our proposed HFA algorithm shows a big improvement in matching time and portrays about 12% matching time saving. The main reason for this effect lies in the fact that our proposed method reduces redundancy migration edges and eliminates unnecessary state transitions by adding the judging instruments at the starting state, which can reduce many unnecessary matching processes. From Fig. 13, we can also observe that our proposed HFA is still lower in matching time compared to that of DFA. This result is due to the deterministic matching state in each regular expression matching process of DFA.

V. CONCLUSION

In this paper, a high-efficient regular expression matching algorithm based on HFA is proposed to improve the performance in the regular expression matching of XFA. In our scheme, we exactly narrow down the execute area size for the next step as well as eliminate any unnecessary state transitions by adding some judging instruments at the starting state, which can reduce a great amount of unnecessary memory access, storage space, and matching time in the regular expression matching of XFA. The simulation results convey that our proposed HFA scheme can slower 40% memory access, save about 45% storage space, and reduce about 12% matching time in the regular expression matching process compared with XFA, but without degrading the matching quality.

AKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their constructive opinions in improving this paper. The work was supported by the Joint Funds of the National Natural Science Foundation of China and Guangdong Natural Science Foundation (No. U2012A002D01); The Strategic Emerging Industries Special of Guangdong Province (No. 2012A09100013-2012BAF11B04-5150); Foundation for Distinguished Young Talents in Higher Education of Guangdong (No. LYM11057); and the Doctoral Project for Natural Science Foundation of Guangdong Province (No. S2012040006666).

REFERENCES

1. K. Heyse, K. Bruneel, and D. Stroobandt, “Proving correct-

ness of regular expression matchers with constrained repetition," *Electronics Letters*, vol. 49, no. 1, pp. 41-42, 2013.
2. X. Wang, Y. Xu, O. Ormond, B. Liu, and X. Wang, "StriFA: stride finite automata for high-speed regular expression matching in network intrusion detection systems," *IEEE Systems Journal*, vol. 7, no. 3, pp. 374-384, 2013.
3. S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner, "Algorithms to accelerate multiple regular expressions matching for deep packet inspection," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4, pp. 339-350, 2006.
4. M. Becchi and S Cadambi, "Memory-efficient regular expression search using state merging," in *Proceedings of the 26th IEEE International Conference on Computer Communications*, Anchorage, AK, 2007, pp. 1064-1072.
5. B. C. Brodie, D. E. Taylor, and R. K. Cytron, "A scalable architecture for high-throughput regular-expression pattern matching," in *Proceeding of the 33rd International Symposium on Computer Architecture*, Boston, MA, 2006, pp. 192-202.
6. D. Ficara, G. Antichi, A. Di Pietro, S. Giordano, G. Procissi, and F. Vitucci, "Sampling techniques to accelerate pattern matching in network intrusion detection systems," in *Proceeding of the IEEE International Conference on Communications*, Cape Town, 2010, pp. 1-5.
7. S. Kumar, J. Turner, and J. Williams, "Advanced algorithms for fast and scalable deep packet inspection," in *Proceedings of the ACM/IEEE Symposium on Architecture for Networking and Communications Systems*, San Jose, CA, 2006, pp. 81-92.
8. 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 *Proceeding of the ACM/IEEE Symposium on Architecture for Networking and Communications Systems*, San Jose, CA, 2006, pp. 93-102.
9. S. Kumar, B. Chandrasekaran, J. Turner, and G. Varghese, "Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia," in *Proceedings of the ACM/IEEE Symposium on Architecture for Networking and Communications Systems*, Orlando, FL, 2007, pp. 155-164.
10. H. Wang, S. Pu, G. Knezek, and J. C. Liu, "Min-max: a counter-based algorithm for regular expression matching," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 92-103, 2013.
11. R. Smith, C. Estan, S. Jha, and S. Kong, "Deflating the big bang: fast and scalable deep packet inspection with extended finite automata," in *Proceedings of the ACM SIGCOMM 2008 Conference on Applications, Technologies, Architectures, and Protocol for Computer Communications*, Seattle, WA, 2008, pp. 207-218.
12. R. Smith, C. Estan, and S. Jha, "XFA: faster signature matching with extended automata," in *Proceeding of the IEEE Symposium on Security and Privacy*, Oakland, CA, 2008, pp. 187-201.
13. K. Peng, Q. Dong, and M. Chen, "TCAM-based DFA deflation: a novel approach to fast and scalable regular expression matching," in *Proceedings of the IEEE 19th International Workshop on Quality of Service*, San Jose, CA, 2011, pp. 1-3.

Jianhua Wang

Jianhua Wang was born on February 6, 1982 in Guangdong, China. He received his B.S. degree in Electronic Information Science and Technology from Shaoguan University, Guangdong, China, in 2006. Currently, he is pursuing a Ph.D. degree in Control Science and Engineering at Guangdong University of Technology. His research interests include 3G wireless video transmission, cyber-physical systems, IoT and wireless sensor networks.

Lianglun Cheng

Lianglun Cheng was born on August 22, 1964 in Hubei. He received his M.S. and Ph.D. degrees from Huazhong University of Science and Technology, Hubei, China, in 1992, and from Chinese Academy of Sciences Jilin, China, in 1999, respectively. He is a professor and doctoral supervisor of Guangdong University of Technology. His research interests include RFID and WSN, IoT and CPS, production equipment and automation of the production process, embedded system, complex system modeling and its optimization control, software of automation and information, etc.

Jun Liu

Jun Liu was born on October 11, 1986 in Hubei, China. He received his M.S. degree in Control Science and Engineering from Guangdong University of Technology, Guangdong, China, in 2012. Currently, he is pursuing a Ph.D. degree in Control Science and Engineering at Guangdong University of Technology. His research interests include cyber-physical systems and wireless sensor networks.