This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2009 proceedings

Compact DFA Structure for Multiple Regular Expressions Matching

Wei Lin[1, 2], Yi Tang[1], Bin Liu[1]
[1] Department of Computer Science and Technology
Tsinghua University
Beijing, China
lin-w04@mails.tsinghua.edu.cn

Derek Pao[2]
[2] Department of Electronic Engineering
City University of Hong Kong
Hong Kong, China

XiaoFei Wang[3]
[3] School of Electronic Engineering
Dublin City University
Dublin, Ireland

Abstract—**New applications such as real-time deep packet inspection require high-speed regular expression (regex) matcher, and the number of regexes in pattern store is increasing to several thousands, which requires a memory efficient solution. In this paper, a kind of hardware based compact DFA structure for multiple regexes matching called CPDFA is presented. According to statistics of regexes in Snort and L7-filter rules, transitions from each state to its next states are not evenly distributed. The summation of transitions from each state to its top three most popular next states takes about 90% of all the transitions. Therefore, CPDFA employs an indirect index table to represent transitions to top three most popular next states more efficiently. The remaining transitions which take about 10% of all the transitions are stored in direct transition table or K parallel SRAMs according to the number of remaining transitions from the same state is more than K or not. Simulation shows that CPDFA structure can save about 90% of memory storage comparing with the original DFA structure. By using pipelined architecture in FPGA, CPDFA can advance one character in one memory access cycle.**

Keywords-regular expression matching; deterministic finite automata; indirect index; parallel SRAMs; deep packet inspection.

I. INTRODUCTION

Nowadays, security has become one of the most serious concerns on the Internet. Network Intrusion Detection Systems (NIDSs) have been adopted by enterprises to defend against all kinds of exploits towards them. Regular expression is widely used as the standard language to represent attack patterns for signature-based NIDSs, because of its expressiveness and matching efficiency [1].

It is well-known that multiple regexes can be combined together into a DFA, which is used to provide constant time for checking each byte of packet's payload to find the matched regex. However, (i) new applications such as real-time deep packet inspection require high-speed regular expression matcher. (ii) The number of regex is increasing to several thousands. The famous open-source NIDS tool, Snort [2], has more than 7000 signatures. The memory size of naïve DFA-based approach is unacceptable, which is exponential in the worst case, e.g., the Snort HTTP signature set will result in a DFA larger than 15GB [3]. Various approaches [3-8] have been proposed to solve this issue.

In this paper, we propose a method called CPDFA. The contributions of CPDFA are:

1. It observes that transitions from a state to its next states are not evenly distributed. About 90% transitions will go to the top three most popular next states. Therefore it can save a lot of storage if transitions to the top three most popular next states can be represented more efficiently. CPDFA uses an indirect index scheme to organize these transitions.

2. The remaining transitions are further optimized to store more compactly and access more efficiently. K parallel SRAMs with different size are used to hold a majority of the remaining transitions. Only a few transitions need to be stored in direct transition table.

3. CPDFA uses pipelined architecture implemented in FPGA to process one input character in one memory access cycle.

The evaluation shows that CPDFA structure can save about 90% memory storage comparing with the original DFA implementation. When implemented by FPGA with clock frequency 500 MHz, the throughput of CPDFA can achieve 4Gbps. The throughput can be further improved when high performance FPGA or ASIC is used.

This paper is organized as follows. The related works is presented in section II. The detailed CPDFA structure and implementation are illustrated in section III. The experimental evaluation is provided in section IV. Section V is the conclusion.

II. RELATED WORKS

Regular expression matching gains prevalent adoption for packet payload scanning. It is becoming the pattern matching standard language for the power and flexibility of expression compared with traditional string matching. In the Linux Application Protocol Classifier L7-filter [9], all protocols are identified by regular expression. Similarly, the Snort [2] and Bro [10] intrusion detection systems also use regular expression in their rule set and the proportion is increasing.

To match a group of regexes, both NFA and DFA can be used. NFA uses a small amount of memory but has to track multiple states simultaneously. Sidhu et al. [11] and Sourdis et

This work is supported by NSFC (60625201,60873250), the Cultivation Fund of the Key Scientific and Technical Innovation Project, MoE, China (705003), the Specialized Research Fund for the Doctoral Program of Higher Education of China (20060003058), 863 high-tech project (2007AA01Z216,2007AA01Z468) and open project of state key Laboratory of Networking and Switching Technology (SKLNST-2008-1-05)

978-1-4244-3435-0/09/$25.00 ©2009 IEEE

al. [12] implemented NFA down to logic gates in FPGA. These approaches are difficult to update the rule set due to the requirement to reprogramming FPGA.

In high-speed networking applications, DFA is used widely to represent regular expression. It has only one state traversal per character so the executive time is predictable, and it maintains only one active state to reduce the complexity in "per flow" network links. However, its prohibitive memory requirement restricts the DFA's popular for complicated rules. And the combination of regular expression makes matters worse for DFA.

Yu et al. [4], proposed to combine regexes into multiple DFAs instead of one DFA to reduce memory. But network traffic has to match multiple DFAs and potentially would be slow. Therefore, their approaches would be infeasible for complex regex rule sets.

D^2FA [7] is the most famous method in this area, in that paper, the author proposed an algorithm to compress DFA through the introduction of default transition, which can save the memory storage with the cost of accessing the DFA multiple times per input character. Following D^2FA, CD^2FA [6] was proposed by using recursive content labels to reduce the diameter bounds of D^2FA to 2 with one 64-bit wide memory access. In [8], Becchi proposed limit the bound of the number of default paths in D^2FA, and improve the worst case of D^2FA.

In [13] Becchi proposed an algorithm to merge states, which reducing partial state information from states into labeled transitions. The author claimed the memory reduction can achieve one order of magnitude. Kumar et al. proposed historical finite automata, i.e., H-FA and H-cFA [5]. The basic idea is to use flag registers to create the conditional transitions so that in many cases the state explosion caused by Kleene closure can be eliminated. Moreover, in H-cFA, Kumar further argued that in the length restriction cases, such as "abc.{100}", a counter could be used to count the number of character so that the corresponding finite automata could have small memory requirement. But in their paper they did not prove the expressiveness of H-FA/H-cFA was indeed equivalent to the original DFA.

In [3], XFA was proposed to use finite automata plus sketch memory to solve the problem. The auxiliary memory (data domain in their terminology) was used to keep track of Kleene closure and length restrictions. However, their regexes to XFA complication still has limitations, XFA requires to manually generating EIDD (Efficient Implementable Data Domain) for each regex which could be very time-consuming and error prone if the operators are not experienced.

In [6], Hayes et al. proposed a scheme called DPICO. The author eliminated the most frequent next-state transitions using a default transition and implemented a matching engine through modified content addressable memory, interleaved memory bank and data packing. The DPICO achieves a good compression rate as the paper claimed. However, there are still some problems. First, the compression rate is not as high as the author claimed, some obliviously storage redundancy exists in the system. Second, DPICO cannot guarantee a consistent processing time. In most case, more than one memory access is required for each character.

In this paper, we propose a scheme called CPDFA, it can obtain compression rate of about 90% through introducing an indirect index table to represent most transitions with only two bits. For each input character, CPDFA can get the DFA lookup work done within one memory access cycle and the line-speed steady throughput can be guaranteed.

III. CPDFA ALGORITHM AND ARCHITECTURE

Assuming the number of character classes in the DFA in Figure 1 is 10. Each transition is marked with the corresponding transition condition. The next states of each state are arranged in descending order based on the number of transitions from the state to each next state. As shown in Figure 1, state B is the first most popular next state for state A, because there are 4 outgoing transitions from A to B. State C is the second most popular next state for state A, because there are 3 outgoing transitions from A to C. State D is the third most popular next state for state A, because there are 2 outgoing transitions from A to D, and so on.

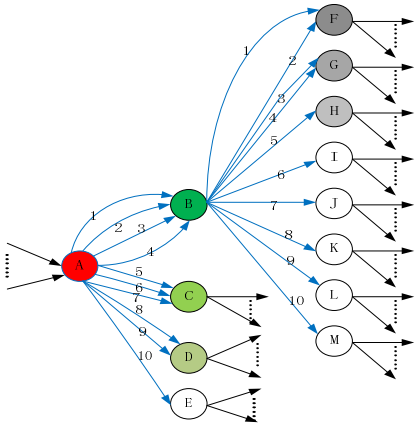

Figure 1 Selection of the three most significant next states for state A

According to the statistics of DFA built with L7-filter rules, the summation of transitions from each state to its top three most popular next states takes about 90% of all the transitions in DFA. Therefore, if these transitions can be represented more efficiently, the storage required by DFA will be significantly reduced.

As shown in Figure 1, it's possible that there are multiple transitions from each state to one of its top three most popular next states. These transitions have the same transition output information. Therefore, an indirect indexing scheme is used to represent transitions from each state to its top three most popular next states.

The input characters are encoded into several character classes when generating the original DFA. The top three most popular next states for each state are calculated based on the number of transitions from the state to each next state. Each state may have different top three most popular next states.

TABLE 1 INDIRECT INDEX TABLE FOR TOP THREE MOST POPULAR NEXT STATES OF EACH STATE

	1	2	3	4	5	6	7	8	9	10
A	00	00	00	00	01	01	01	10	10	11
B	00	00	01	01	10	11	11	11	11	11
C										
D										
E										

As shown in TABLE 1, an index table is used to store transitions from each state to its top three most popular next states. The row index is the state ID and the column index is the input character class. The table entry is represented using two bits. 00 means the transition is from the state to its first most popular next state, 01 means the transition is from the state to its second most popular next state, 10 means the transition is from the state to its third most popular next state, and 11 means the transition is from the state to one of other next states.

Because the three most significant next states for each state are different, a transition output table is used to record the three most significant next states for each state. As Shown in Table 2. Let the number of states be M and the size of a state ID be L bits. The transition output table can be organized as M rows by 3 x L bits wide. In this case, the current state ID can be used to index into the transition output table. The offset obtained from indirect index table can be used to choose one of the three next state IDs of the selected row. The index table and the transition output table can be accessed in pipeline manner. In this way, transitions from each state to its top three most popular next states can be stored and accessed efficiently.

TABLE 2 TOP THREE NEXT STATES FOR EACH STATE

State ID	Index 00	Index 01	Index 10
⋮	⋮	⋮	⋮
A	B	C	D
B	F	G	H
⋮	⋮	⋮	⋮

Excluding transitions from each state to its top three most popular next states, the remaining transitions of each state are stored and accessed in different way according to whether the number of the remaining transitions of each state is more than K or not.

As shown in Figure 2 if the number of remaining transitions of a state is not more than K (K=4), the states will be arranged in descending order based on the number of remaining transitions of the state, and the transitions of the same state will be stored at the same address in different SRAMs from left to right. The state ID is used as index to access the same address of multiple SRAMs. The remaining transitions of the state are readout and compared with the encoded input character class simultaneously by comparing logic to get the matched transition. For example, excluding transitions from state A to state {B, C, D}, there is only one remaining outgoing transitions from A to E for state A. This transition is stored in the first one SRAM of the K parallel SRAMs, which is indexed by state ID of A. If state A is the current state, transition from A to E is readout and compared with the encoded input character class.

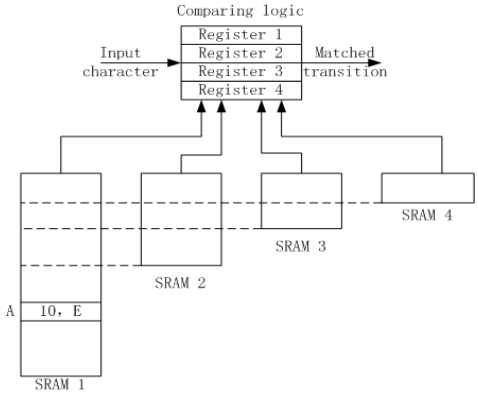

Figure 2 K parallel SRAMs to store remaining transitions not more than K for some states

As shown in TABLE 3, if the number of remaining transitions for a state is more than K, the transitions will be stored in a direct transition table (DTT). The row index is the state ID, and the column index is the input character class. The table entry is the next state information. For example, excluding the outgoing transitions from B to {F, G, H}, if the number of outgoing transitions for state B is more than K (K=4), one entry in DTT will be used to store these remaining outgoing transitions for state B. According to statistics of DFA built with L7-filter rules, the number of entries in DTT is less than 10, when K=8. Because transitions to the top three most popular next states can be stored in indirect index table which takes about 90% of all the transitions and the remaining transitions not more than K can be stored in K parallel SRAMs, only quite a few transitions are left to store in DTT.

TABLE 3 DTT FOR REMAINING TRANSITIONS MORE THAN K

State ID	1	2	3	4	5	6	7	8	9	10
B	F	F	G	G	H	I	J	K	L	M

TABLE 4 STORAGE ESTIMATION

Table name	Storage requirement
Indirect index table	2*M*L bits
Transition output table	48*M bits
K parallel SRAMs	8*N*K bits (in average)
DTT	16*(M-N)*L bits

As shown in TABLE 4, assuming the number of character classes is X and there are M states in the DFA, excluding the transitions to each top three next states, N states have remaining transitions not more than K. The width of state ID is two bytes. The reduction ratio is about 90% comparing with original DFA solution, when X=128 and N is nearly equal to M. The above assumption is near to reality when the size of DFA is large enough.

As shown in TABLE 5, state ID is used to guide which tables should be accessed during the looking up procedure. Each state has an entry in the indirect index table and one entry in the transition output table, so the range of state ID in the above two tables is allocated from 0 to (M-1). Each remaining transition of the N states has an entry in one of K parallel SRAMs and the remaining transitions of the same state are stored in the same address of different SRAMs, so the range of state ID in K parallel SRAMs is allocated from 0 to (N-1). The other part of remaining transitions belonging to (M-N) states are stored in DTT and transitions of the same state are stored in one entry in DTT, so the range of state ID in DTT is allocated from N to (M-1).

TABLE 5 STATE ID ASSIGNMENT

Table name	Range of State ID assignment
Indirect index table	0 to (M-1)
Transition output table	0 to (M-1)
K parallel SRAMs	0 to (N-1) (N<=M)
DTT	N to (M-1) (N<=M)

When looking up DTT, N is used as base address; the actual address in DTT can be calculated using state ID minus N.

In order to share the above four tables with multiple DFAs, each DFA will be allocated a section of address space in each table and each DFA will record its starting address in each table. The actual address can be got by adding the starting address.

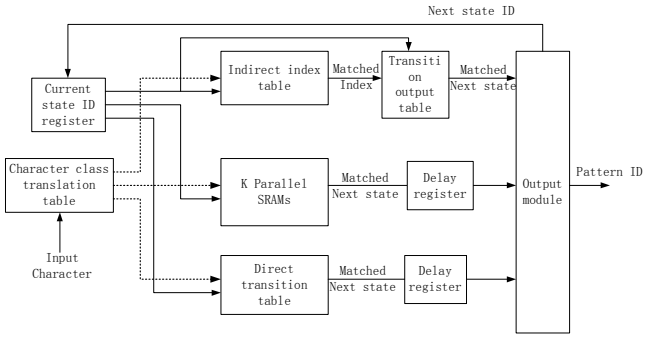

Figure 3 Architecture of the compact pattern matcher

The architecture of the compact pattern matcher is shown in Figure 3. The input character is sent to character class translation table to get the corresponding character class. The current state ID register stores the state ID to indicate which state in DFA is currently accessed. If the current state ID is less than N, both the indirect index table and K parallel SRAMs are accessed simultaneously. If the current state ID is equal or more than N, both the indirect index table and DTT are accessed simultaneously. The current state ID and the input character class are used as keys to lookup corresponding tables. The lookup result of each table is sent to the output module, and the output module will output the matched pattern ID or update the current state ID register with the new state ID. This system can be pipelined with delay registers added to synchronize the output phase of multi-tables, and the throughput of the system is one character per memory access cycle. Each table can be implemented using memory blocks in FPGA. The value of K can be reconfigured when more parallel SRAMs are required to hold larger DFA.

By using the indirect index to represent transitions to top three most popular next states, about 90% transitions can be represented more compactly and efficiently. The remaining transitions of each state are stored in K parallel SRAMs or DTT, which further reduced the storage requirement to represent DFA transitions. By using the pipeline architecture, each input character can be processed in one memory access cycle.

IV. PERFORMANCE EVALUATION

In order to evaluate the effectiveness of our CPDFA scheme, the statistical characteristic of DFAs built with Snort or L7-filter rules is analyzed in Table 6. Then we performed the following experiments on how to get optimization result using these rule sets.

We considered the regular expressions used both in Snort rules set and Linux layer-7 application protocol classifier. Based on an efficient algorithm to partition large set of regular expressions into multiple groups proposed by Yu et al. [4], we modified the grouping algorithm to make it work for our experiment environment which pursues the purpose of covering DFAs with different size. We choose the grouped set randomly. Then we selected some of them who have different number of states so as to make our CPDFA structure available in any situation. Furthermore, we applied Counting algorithm shown below to these regular expression groups to calculate all kinds of statistical characteristic data for analyzing.

TABLE 6 SELECTED DFA GROUPS OF DIFFERENT SIZE

DFA Groups	*Total # of states*	*Total # of transitions*	*Char class*	*R-trans[1] # <= K[2]*	*# of states with R-trans> K*	*The % of TOP-3[3]*
L1[4]	826	38822	47	1166	0	96%
L2	3281	131240	40	4614	0	95%
L3	10972	1250808	114	75320	2453	91%
L4	23198	2180612	94	177498	1463	90%
S5[5]	3580	246400	61	6845	2744	85%
S6	15051	1008417	67	68605	2	91%
S7	366686	23467904	64	309668	19317	86%

[1] R-trans: remaining transitions except transitions to top 3 most popular next states.
[2] K: the default value is set to 10.
[3] TOP-3: Transitions from each state to its top three most popular next states
[4] L: L7-filter rules group.
[5] S: Snort rules group.

As shown in TABLE 6, it presents statistical characteristic of rule groups with different size and source. It shows that, the number of TOP-3 transitions dominate (about 90%) of all transitions. Definitely it is efficient to use indirect index table to store these part of transitions. Except Top-3 transitions, we use K parallel SRAMs to store the remaining transitions of a state whose remaining fan-out number is not more than K. Finally, we maintain a DTT table for the left transitions of a state whose remaining fan-out number is more than K.

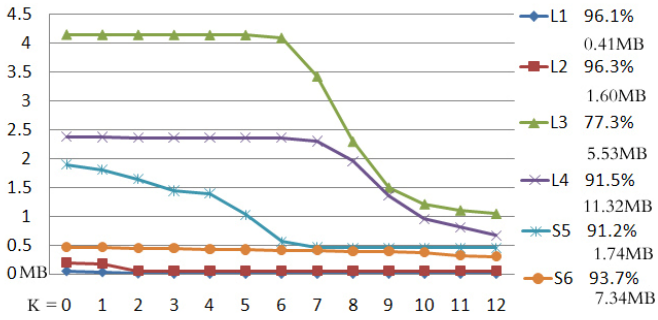

Figure 4 The DFA storage varies with different values of K

In Figure 4, we plot the tendency of DFA storage with different values of K. The value on the right side is the compression ratio and the original DFA cost for each group. It's obvious that the DFA storage drops rapidly at first and becomes steady gradually, which means it's efficient to employ K parallel SRAMs in this system and a proper value of K can be obtained to balance the hardware resource cost and the expected reduction ratio. It will consume too much DTT space if K has been set a very small value. But when K is increasing, such as S6 remains steady and L3 keeps on reducing slightly. Notice that, we set a moderate value 10 to K in our implementation, which can achieve about 90% reduction ratio in average.

TABLE 7 COMPARISON WITH OTHER METHODS

	Compression ratio (in average)	**# of states accessed per input character**
Original DFA	0	1
D²FA	90%	≥2(in average)
DPICO	65%	1(but may access one SRAM multiple times)
CPDFA	90%	1 (SRAMs are accessed in parallel)

We compare our CPDFA architecture with the original DFA method and other state of the art methods. The results are summarized in Table 7.

V. CONCLUSION

A kind of hardware based compact DFA structure for multiple regular expressions matching called CPDFA is presented in this paper. According to statistics, transitions from one state to its next states are not evenly distributed. The summation of transitions from each state to its top three most popular next states takes about 90% of all the transitions. The remaining transitions to other next states only take about 10% of all the transitions. Therefore, CPDFA use an indirect index table to represent transitions to top three most popular next states, where each transition can be represented by two bits. In this way, the majority of transitions can be stored compactly. In order to further improve the reduction ratio, a part of the remaining transitions, which are from the same state and the number is not more than K, are stored in K parallel SRAMs. The number of transitions needs to be stored in direct transitions table is much less than the original DFA method. The reduction ratio of CPDFA is about 90% in average. Using the Altera Stratix II FPGA with RAM blocks frequency 500MHz, the system throughput of CPDFA is about 4Gbps.

REFERENCES

[1] R. Sommer and V. Paxson, "Enhancing byte-level network intrusion detection signatures with context," in Proceedings of the ACM Conference on Computer and Communications Security New York, NY 10036-5701, United States: Association for Computing Machinery, 2003, pp. 262-271.

[2] Snort, "http://www.snort.org/," 2009.

[3] R. Smith, C. Estan, and S. Jha, "XFA: Faster signature matching with extended automata," in Proceedings - IEEE Symposium on Security and Privacy New York, NY 10016-5997, United States: Institute of Electrical and Electronics Engineers Inc., 2008, pp. 187-201.

[4] Yu Fang, Chen Zhifeng, Diao Yanlei, T. V. Lakshman, and R. H. Katz, "Fast and memory-efficient regular expression matching for deep packet inspection," in ANCS 2006 - Proceedings of the 2006 ACM/IEEE Symposium on Architectures for Networking and Communications Systems New York, NY 10036-5701, United States: Association for Computing Machinery, 2006, pp. 93-102.

[5] Sailesh Kumar, Balakrishnan Chandrasekaran, Jonathan Turner, and George Varghese, "Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia," in ANCS 2007 - Proceedings of the 2007 ACM/IEEE Symposium on Architectures for Networking and Communications Systems 2007.

[6] S. Kumar, J. Turner, and J. Williams, "Advanced algorithms for fast and scalable deep packet inspection," in ANCS 2006 - Proceedings of the 2006 ACM/IEEE Symposium on Architectures for Networking and Communications Systems New York, NY 10036-5701, United States: Association for Computing Machinery, 2006, pp. 81-92.

[7] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner, "Algorithms to accelerate multiple regular expressions matching for deep packet inspection," Computer Communication Review, vol. 36, no. 4, pp. 339-350, 2006.

[8] B. Michela and C. Patrick, "An improved algorithm to accelerate regular expression evaluation," in ANCS 2007 - Proceedings of the 2007 ACM/IEEE Symposium on Architectures for Networking and Communications Systems 2007.

[9] Application Layer Packet Classifier for Linux, "http://l7-filter.sourceforge.net/," 2009.

[10] Bro Intrusion Detection System, "http://bro-ids.org/," 2009.

[11] R. Sidhu and V. K. Prasanna, "Fast regular expression matching using FPGAs," in Proceedings - FCCM 2001, pp. 227-238.

[12] I. Sourdis and D. Pnevmatikatos, "Pre-decoded CAMs for efficient and high-speed NIDS pattern matching," in Proceedings - 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2004 Los Alamitos, CA 90720-1314, United States: IEEE Computer Society, 2004, pp. 258-267.

[13] M. Becchi and S. Cadambi, "Memory-efficient regular expression search using state merging," in Proceedings - IEEE INFOCOM Piscataway, NJ 08855-1331, United States: Institute of Electrical and Electronics Engineers Inc., 2007, pp. 1064-1072.