A Flexible Wildcard-Pattern Matching Accelerator via Simultaneous Discrete Finite Automata

Hsiang-Jen Tsai, Chien-Chih Chen, Yin-Chi Peng, Ya-Han Tsao, Yen-Ning Chiang, Wei-Cheng Zhao, Meng-Fan Chang, *Senior Member, IEEE*, and Tien-Fu Chen, *Member, IEEE*

***Abstract*— Regular expression matching becomes indispensable elements of Internet of Things network security. However, traditional ternary content addressable memory (TCAM) search engine is unable to handle patterns with wildcards, as it precisely tracks only one active state with single transition. This paper proposes a promising simultaneous pattern matching methodology for wildcard patterns by two separated engines to represent discrete finite automata. A key preprocessing to encode possible postfix pattern by a unique key ensures that follow-up patterns can accurately traverse all possible matches with limited hardware resources. This approach is practical and scalable for achieving good performance and low space consumption in network security, and it can be applicable to any regular expressions even with multiwildcard patterns. The experimental results demonstrate that this scheme can efficiently and accurately recognize wildcard patterns by simultaneously tracking only two active states. By adopting SRAM TCAM in the proposed architecture, the energy consumption is reduced to around 39%, compared with the energy consumption using a computing system that contains a large memory lookup and comparison overhead.**

***Index Terms*— Deep packet inspection (DPI), discrete finite automata (discrete-FA), network security, simultaneous pattern matching, ternary content addressable memory (TCAM) based search engine, wildcard pattern matching.**

I. Introduction

With the recent rapid increase in the popularity of Internet of Things (IoT) in electronic systems, security of networks has become a critical challenge because numerous small networks merge into large networks [1]–[3]. At the same time, limited hardware resources such as communication channels/interfaces, bandwidth, storage, and energy cause various potential vulnerabilities [4]–[6]. An efficient design for deep packet inspection (DPI) is an indispensable element because the security problems will be a key factor in IoT development.

In the existing pattern matching methods of DPI, regular expression matching algorithms are widely used in networking devices [7], [8]. These algorithms are expressive, efficient, and flexible in detecting attacks because the payload of packets can be examined, regardless of whether any predefined regular expressions are matched by regular expression matching algorithms [9]–[11]. They perform intrusion detection, attack detection duties, and provide antivirus defense such as firewalls in layer-7 switches [12], [13]. Fig. 1 shows that a virus detection computing system, called memory lookup implementation, is implemented by the processor and memory in the firewall router to check the payload to confirm that the connection is secure. However, the disadvantage of this architecture is its sequential comparison operations and memory access.

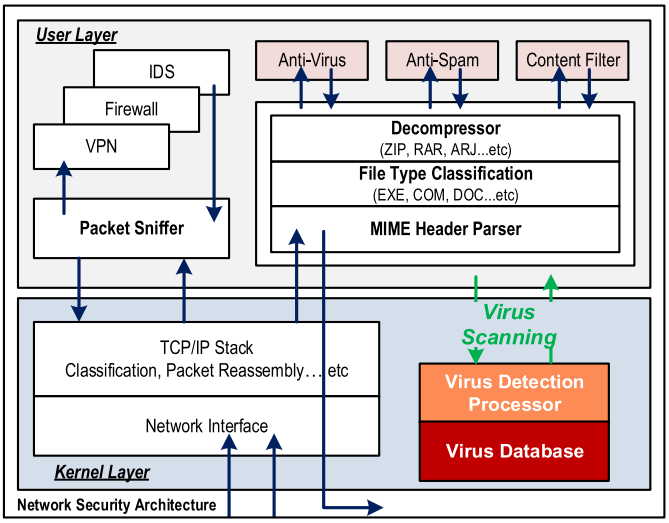


Fig. 1. Architecture of firewall router [12].

Recently, ternary content addressable memory (TCAM) based search engines have been used to implement regular expression matching algorithms to utilize their parallel comparison and "don't care (X)" search abilities to achieve high speeds [14], [15]. In this architecture, the state machine of regular expression can be efficiently implemented because the number of transitions in the state machine is equal to the number of TCAM entries. In addition, the patterns that spread across multiple packets in a flow can be monitored by TCAM-based search engines because they run a unique state machine for each flow [15], [16].

Unfortunately, TCAM-based search engines cannot be used to recognize wildcard patterns, which are restricted by

Manuscript received June 28, 2016; revised December 21, 2016 and February 14, 2017; accepted February 15, 2017. Date of publication March 7, 2017; date of current version November 22, 2017.

H.-J. Tsai, C.-C. Chen, Y.-C. Peng, Y.-H. Tsao, and T.-F. Chen are with the Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan (e-mail: hjtsai@cs.nctu.edu.tw; ccchen99@cs.nctu.edu.tw; pengyc@cs.nctu.edu.tw; yhtsao@cs.nctu.edu.tw; tfchen@cs.nctu.edu.tw).

Y.-N. Chiang, W.-C. Zhao, and M.-F. Chang are with the Department of Electrical Engineering, National Tsing Hua University, Hsinchu 300, Taiwan (e-mail: yenning0527@gmail.com; a29823479@gmail.com; mfchang@ee.nthu.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2017.2671408

precisely one active state for tracking and one state transition lookup. Detecting the match of unpredictable input strings in wildcard patterns is in fact the most critical challenge because the nondeterministic nature of wildcard patterns results in a large number of possible matches. Therefore, restrictions on the search operation in the architecture and unpredictable match strings are the major issues in this paper.

Based on the above observations, we propose an efficient separated TCAM search engine employing a clever simultaneous pattern matching methodology that uses discrete finite automata (discrete-FA) for wildcard-pattern matching problems. A key challenge of this design lies in having accurate traversal and traversing all possible matches in an arbitrary number of characters that appear from a wildcard. In this paper, we present algorithm/architecture codesign techniques that can be used in hardware implementation, to recognize whether input strings and wildcard patterns match with limited hardware resources. We address several design issues, including how to design the search engines using TCAM for wildcard pattern matching, how to construct the discrete-FA for the predefined patterns, and how to detect the match of input strings that have accurate traversal and traverse all match possibilities. In summary, the primary contributions of this paper are as follows.

1) We propose an efficient separated TCAM search engine architecture and a simultaneous pattern matching methodology utilizing TCAM features to detect all possible matches with limited hardware resources, which resolve the nondeterministic nature of wildcard patterns that cause a large amount of possible potential matches. This technique does not alter the data flow during search execution.
2) We construct discrete-FA by dividing a regular expression with wildcards to recognize wildcard patterns, thereby enabling simultaneous search. This technique resolves the problem of wildcard pattern matching with limited hardware resources and does not require significant architectural modification.
3) We reduce the nondeterministic number of active states from $O(N)$, where N is the number of wildcard patterns, to $O(1)$ in wildcard pattern matching by simultaneous tracking only two active states with two transitions.
4) We present a cluster encoding method to significantly reduce the key size and resolve ambiguity problems in the key assignment, thereby enabling TCAM capacity reduction for wildcard pattern matching in the proposed work.

II. Limitations of Hardware Search Engines

In this section, we first explore the negative effect of the number of active states and point out that nondeterministic finite automata (NFA) based methods that use multiple active states in a hardware search engine are unsuitable. Then, we describe the basic operations of traditional TCAM-based search engines and use the "don't care (X)" features of TCAM to compress data entries. We then construct state machines for wildcard patterns in traditional TCAM-based search engines and identify the unsuitability of these search engines in recognizing wildcard patterns. Finally, by implementing the traditional TCAM-based search engines for a practical case, we analyze the shortcomings of wildcard pattern matching across an input scenario.

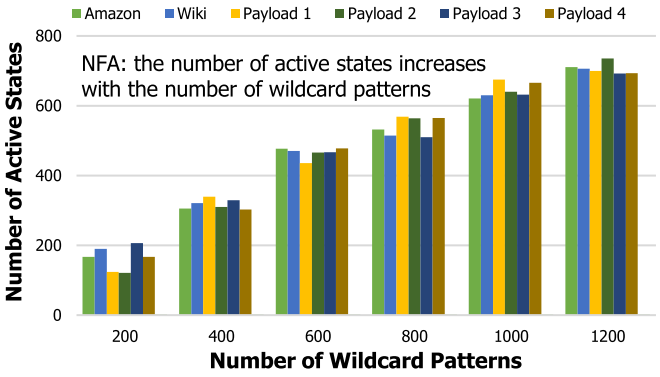


Fig. 2. Active state evaluations of NFA.

A. Why Limit the Number of Active States?

Regular expressions are typically implemented by two classic finite automata (FA): deterministic FA (DFA) and NFA. However, neither is ideal for implementation in the hardware search engine with limited resources for real-world pattern sets. Previous studies have shown that the NFA-based methods can recognize wildcard patterns in regular expression matching algorithms [17], [18]. In such methods, a drawback is that multiple states are active in parallel for tracking all possible potential matches. To make state transitions for processing input character, the number of active states implies the number of memory access required. Therefore, if wildcard patterns are recognized, then to have accurate traversal, multiple memory access with multiple active states must be used to detect two possible match situations. One possibility is that the following input string matches with the follow-up of the wildcard pattern that is recognized and another possibility is that the following string is matched with some other predefined patterns.

In our observations, as shown in Fig. 2, we find that the number of active states increases with the number of wildcard patterns, indicating that an unpredictable number of active states are required to detect all possible matches concurrently. This causes each input character requires more than one search in the wildcard pattern matching process and then results in $O(N)$ memory access and computation to examine each character in the payload, where N is the number of active states in the automata. The input character is difficult to detect immediately because of the number of active states, which considerably increases overhead in search operations with a large number of sequential memory access and computations. In addition, with the limited hardware resources, such as bandwidth, storage and the number of processors, the nondeterministic computation, and memory access overhead of NFA-based methods will slow matching speed at run time. Therefore, NFA-based methods are not suitable to implement in the embedded systems because of the multiple active states.

In contrast, although DFA only requires a single memory access to process each character in the payload, it is impossible

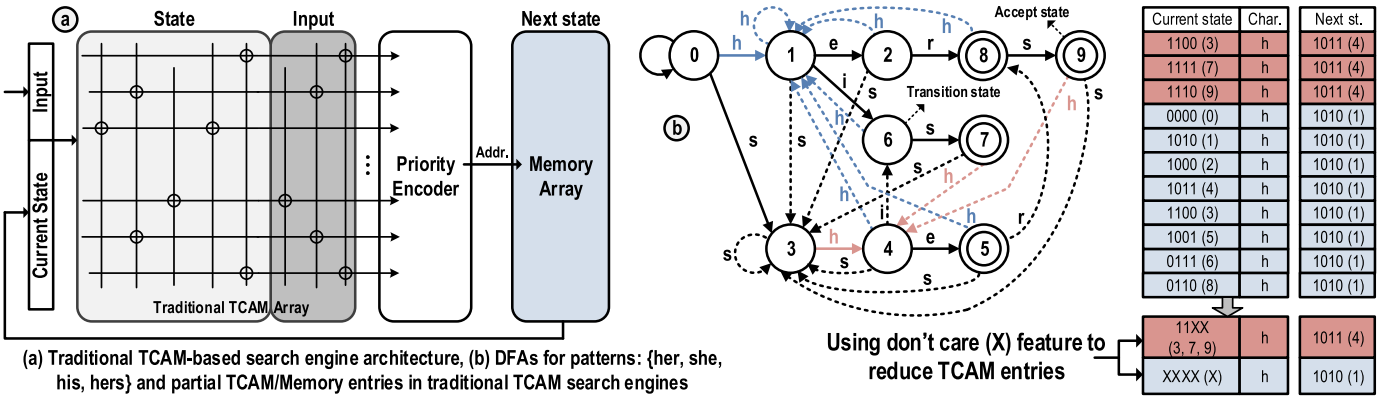


Fig. 3. Traditional TCAM-based search engines. (a) Traditional TCAM-based search engine architecture. (b) DFAs for patterns: {her, she, his, hers} and partial TCAM/Memory entries in traditional TCAM search engines.

to construct the DFA consisting of all possible potential situations for the wildcard pattern matching. This is because the number of states and transitions in the DFA are exponentially larger than that in the NFA [19], [20].

The construction of a hybrid FA (hybrid-FA) has been proposed to reduce memory storage by combining the benefits of DFA and NFA [21], [22]. However, unlimited activations of these approaches for traversing all possible matches also turns out to be a major constraint. Given an input character, these prior approaches usually require many concurrent comparison operations and memory access, which leads to performance degradation. More importantly, these approaches have focused on how to reduce memory requirements and mitigate the impact of state explosion. In contrast, the main idea of our work is to utilize the parallel search ability of TCAM to avoid sequential memory access and, thus, limit the number of active states because it is impossible to have unlimited comparison units for multiple active states.

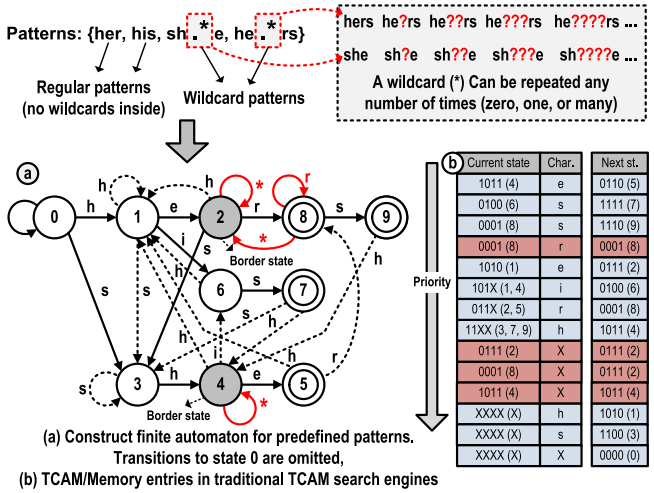


Fig. 4. Wildcard patterns in traditional TCAM-based search engines. (a) Construct finite automation for predefined patterns. Transitions to state 0 are omitted. (b) TCAM/memory entries in traditional TCAM search engines.

B. Traditional TCAM-Based Search Engines

Fig. 3(a) shows that the TCAM-based search engine architecture consists of three major components: a TCAM array, a priority encoder, and a memory array [16]. The data of each TCAM entry consist of the current state and an input character, which is used to search for state transition. In a search operation, the search data are stored in the input register and the current state register is initialized to state 0. If the search data matches the state and input character in the TCAM array, the index of the matching entry is given as the output by the priority encoder, and the next state information is obtained from the memory array.

Multiple entries can be concurrently matched because using the "don't care (X)" feature of TCAM can efficiently reduce data entries as illustrated in Fig. 3(b). This is because the match is verified regardless of the search data by the "don't care (X)" feature. The priority encoder outputs the index of the first matched (high priority) entry and encodes this index into binary format for retrieving the corresponding next state from the memory array. The input pointer is advanced to the next input character and the current state register is set to the initial state unless there is a matched entry in the TCAM. If no data need to be detected in the packets, the process will be terminated.

C. Restrictions of Single Active State

In regular expression matching, regular expressions, which consist of wildcard patterns and regular patterns (no wildcards inside), can be used to construct state machines to recognize input strings [23], [24]. Fig. 4 illustrates the state machine of predefined patterns and the corresponding TCAM/memory entries for traditional TCAM-based search engines. In traditional TCAM-based search engines, one active state for tracking and one state transition lookup in search operations are sufficient to recognize regular patterns [25], [26]. However, in wildcard pattern matching, infinite possible match strings are generated because of a wildcard that can repeat any number of arbitrary characters (zero, one, or many) as shown in Fig. 4. Therefore, when the border state is reached in wildcard pattern matching, the following strings, which are composed of an arbitrary number of characters, need to be detected.

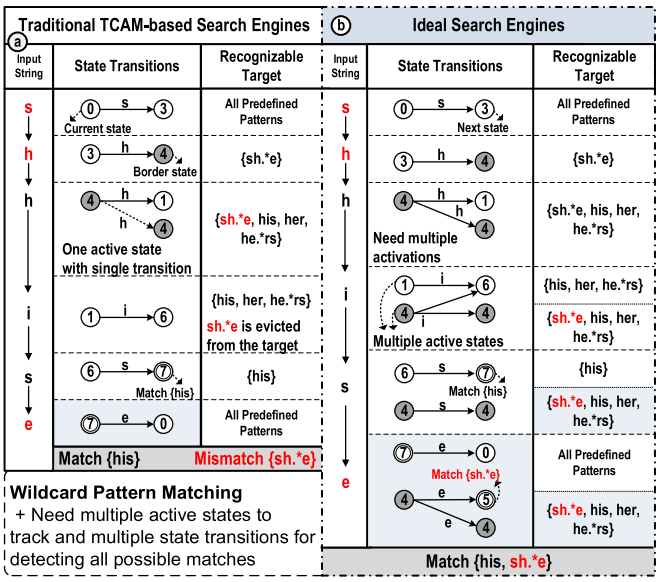


Fig. 5. Limitations of single activation in traditional TCAM-based search engines. (a) Traditional TCAM-based search engines. (b) Ideal search engines.

Fig. 5 illustrates the limitations of recognizing input strings in traditional TCAM-based search engines. An example of traversal in search engines with input string "shhise" is shown in Fig. 5(a). After processing two input characters "sh," the wildcard pattern {sh.*e} become a recognizable target in the matching process and border state "4" is reached. Then, when the next input strings are input to the state machine, the match is not only detected in the follow-up of the wildcard pattern which is recognized but also traversed in the other patterns.

For example, if the next input character is "h," extra patterns such as {her, his, he.*rs} need to be added in the matching process. This is because the following input strings are unpredictable, which may or may not be matched with other predefined patterns. In this case, the recognizable target will be changed from {sh.*e} to {his, her, her.*rs}, where the priority of state "1" is higher than state "4" in the TCAM-based search engines, as shown in Fig. 5(a). The wildcard pattern {sh.*e} will be eliminated from the recognizable target because one active state tracks only one possible situation in the matching process. Hence, even though the following input string is "e," the wildcard pattern {sh.*e} cannot be matched because this pattern is not the recognizable target in the matching process. We cannot traverse the match in the existing matching process, which is limited by the number of active states for tracking and state transition lookup in traditional TCAM-based search engines.

Fig. 5(b) shows that multiple active states are required to track and multiple state transition lookups are needed for accuracy in wildcard pattern matching. With multiple activations, the wildcard pattern {sh.*e} is not eliminated from the recognizable target and the patterns {sh.*e, his, her, her.*rs} will be recognized concurrently in the matching process. Therefore, the patterns in the input strings can be recognized in ideal search engines that have multiple activations. We therefore propose a novel architecture, separated TCAM search engine architecture, to resolve the constraint of single activation for traversing all possible matches.

III. Efficient Separated TCAM Search Engines

In this section, we first describe a new wildcard-pattern matching architecture, separated TCAM search engine architecture, which combines two traditional TCAM search engines and a simultaneous matching engine. It has accurate traversal and detects all possible matches to achieve fast and scalable regular expression matching with wildcard patterns. The proposed architecture utilizes the unique parallel and wildcard matching capabilities of TCAM to recognize input strings, unlike the existing solutions that use memory lookup methodology to determine the match by sequential memory access. Next, we describe a simultaneous patter matching scheme that employs discrete-FA to recognize wildcard patterns in the proposed architecture. The prefix and suffix state machines in the discrete-FA can be searched independently for detecting all possible matches of wildcard patterns. Finally, a cluster encode method is proposed to ensure that follow-up patterns can accurately traverse all possible matches by a unique key without ambiguity.

A. Hardware Architecture

In the proposed architecture, simultaneous pattern matching methodology is proposed to recognize wildcard patterns in regular expressions to resolve architecture restrictions in traditional TCAM-based search engines. The proposed separated TCAM search engine consists of three major components, which includes two small isolated TCAM search engines and a simultaneous matching engine, as shown in Fig. 6. We define a state flag, a key, and a segment ID, which serve as the simultaneous pattern matching methodology of each state in each search engine. In our observations, we find that using two active states with two transitions were sufficient to recognize wildcard patterns in the simultaneous pattern matching methodology. Therefore, two isolated search engines are used to implement different state machines that are constructed by the prefix segment and suffix segment of patterns. This means that the state machine of the prefix segment and suffix segment can be searched concurrently. This is plotted in Fig. 6(a) and (b).

1) Simultaneous Matching Engine: The proposed simultaneous matching engine consists of three major components as shown in Fig. 6(c). The first component is a search comparator, which is used to determine the actions of the key according to the state of the flag, where the states include abandon, search, write, or out-match. The second component is a key matching engine that is used to store the key and compare the stored key with the search key in a TCAM array. For wildcard pattern matching, the corresponding key of the accept state (stored key) will be stored for the next operation, which is the match case of the prefix state machine. On the other hand, the corresponding key of the accept state in the match case of the suffix state machine (search key) is used to search the stored key in the simultaneous matching engine. In Fig. 6(c), if the search result is matched, the wildcard pattern will be detected. The key feature of this approach is

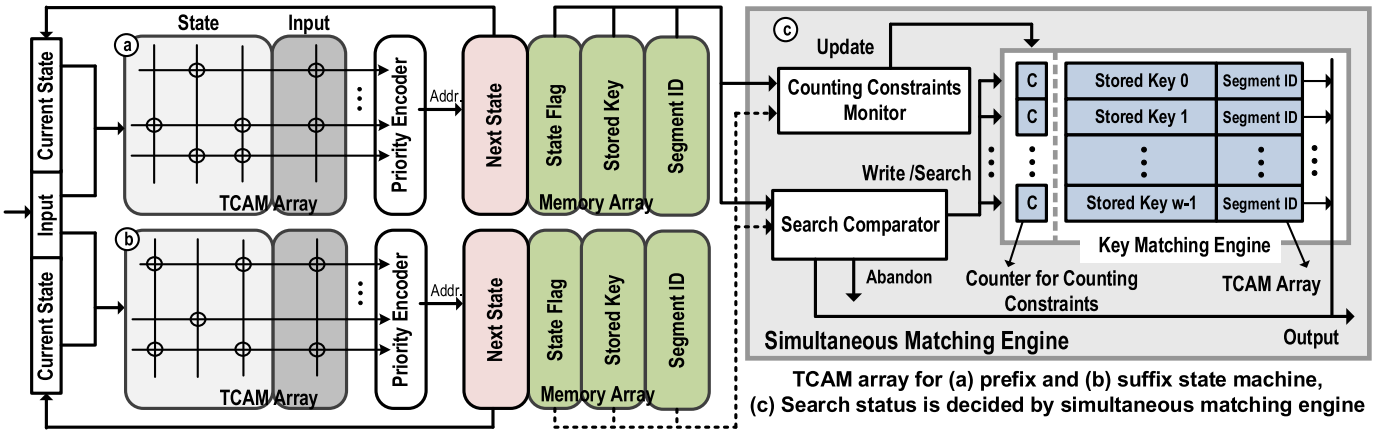


Fig. 6. Separated TCAM search engines. TCAM array for (a) prefix and (b) suffix state machine. (c) Search status is decided by simultaneous matching engine.

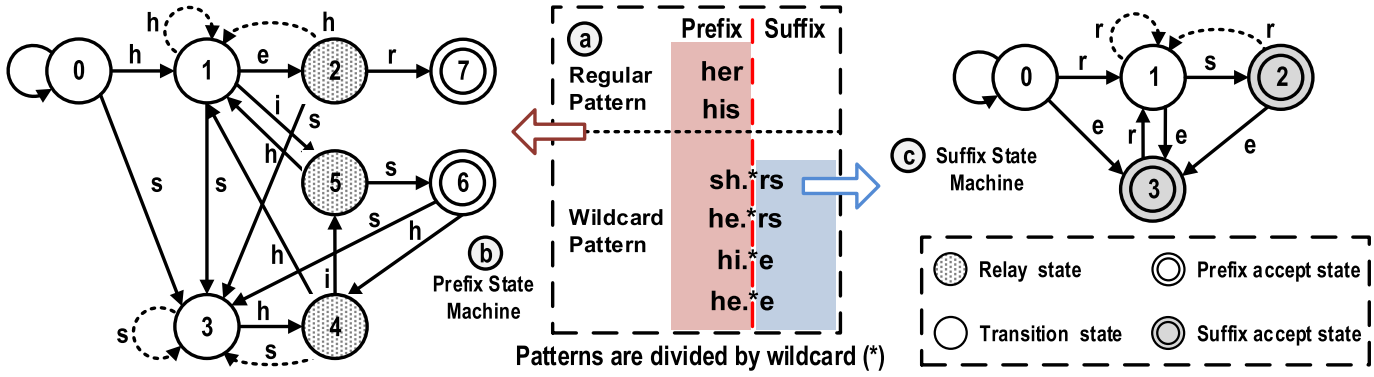


Fig. 7. Construct discrete-FA for predefined patterns. (a) Patterns are divided by wildcard (∗). (b) Prefix state machine. (c) Suffix state machine.

that it uses another small TCAM to process the search results of the prefix state machine and the suffix state machine to resolve the wildcard-pattern matching problem. The operation of wildcard pattern matching will be discussed in more detail in Section III-C.

The third component is a counting constraints monitor to update the counter of the stored key according to the state flag. As mentioned earlier, the wildcard of the wildcard pattern can be replaced by any number of arbitrary characters in the input string. Thus, we define a counter in the key matching engine to serve as the counting constraints of each stored key, which limits the lifetime of wildcard patterns that is recognized. The counter of the stored key which is used to tolerate the characters because of a wildcard during the process of recognizing the pattern determines the eviction operation for the stored key. When the counter of the stored key does not equal zero, the counter is decremented for the counting constraints. If the counter of the stored key equals zero, the engine makes space for the incoming stored key by evicting this key. The segment ID is used to ensure that each segment of a multiwildcard pattern is sequentially matched in the input strings. The operation of multiwildcard pattern matching will be discussed in more detail in Section IV.

B. Single Wildcard Patterns in Discrete-FA

One challenge in recognizing wildcard patterns in TCAM-based search engines is to identify precisely only one active state with single transition in each search operation that negatively affects detecting all possible matches. We know that using more active states with more state transitions can traverse a greater number of possible matches. It is impossible to have variable active states to track because of the limitations of hardware implementation. Therefore, a hardware search engine is required that can detect all possible situations with fixed active states to search for wildcard pattern matching. As mentioned before, all possible matches of a wildcard pattern are generated by a wildcard. However, the fundamental match condition of wildcard pattern matching is satisfied only if the prefix and suffix segments of the wildcard patterns are sequentially matched in input strings, irrespective of the number of characters inserted between the prefix segment and the suffix segment. On the other hand, we also need to address the possibility of matches with other predefined patterns in an arbitrary number of characters that are inserted between the prefix segment and the suffix segment during the process of recognizing the wildcard pattern.

Based on the above observations, we find that an architecture consisting of two active states with two state transitions in each search operation is sufficient for accurate traversal and for detecting two possible ways of match in the regular expression matching. For constructing state machines in discrete-FA, the predefined patterns, which consist of wildcard patterns and regular patterns, can be divided into two segments (prefix and suffix) with a wildcard at the beginning of constructing state machines, as shown in Fig. 7(a). To increase the complexity

of pattern matching, we added two extra wildcard patterns {hi.*rs, he.*e} in the predefined patterns. After the partitioning of wildcard patterns, a wildcard pattern {sh.*e} can be transformed to two regular patterns {sh, e}. In the original regular patterns {he, his}, all the elements of patterns belong to the prefix segment because there is no wildcard inside the patterns. On the other hand, the forward elements which are partitioned by wildcard {sh, he} are the subpatterns of the prefix segment. In contrast, the remaining elements of wildcard patterns {e, rs} belong to the suffix segment in the discrete-FA. Next, two state machines for the prefix segment and the suffix segment were constructed independently for the proposed architecture, as shown in Fig. 7(b) and (c).

In the discrete-FA, for matching each input character, the prefix state machine is used to detect the possible matches of the original regular patterns and the prefix segment of wildcard patterns. On the other hand, the suffix state machine is used to detect the matches of regular patterns obtained from the suffix segment of wildcard patterns. Therefore, the match can be traversed in two possible ways in this matching process. However, the search results of the prefix state machine and the suffix state machine need to be processed in advance because one of them is not the match in the wildcard patterns. The process of the key matching will be discussed in more detail in the following section.

C. Simultaneous Pattern Matching

To distinguish between the search status of the search operation in discrete-FA, we defined several new state status such as transition state, relay state, prefix accept state, and suffix accept state, which serve as the search operation in pattern recognition. During the state machine construct, each state is marked to a different status, which corresponds to the nature of the discrete-FA, as illustrated in Fig. 7(b) and (c). In the prefix state machine, states "6" and "7" are marked to the prefix accept state, which is the pattern matching state in the regular pattern {her, his}. The relay state, such as states "2," "4," and "5," is very important in the discrete-FA, which is used to link the prefix segment and the suffix segment in the wildcard pattern matching process. When the relay state is reached in the matching process, corresponding actions are activated for wildcard pattern matching. In this phase, the corresponding key (stored key) of the wildcard pattern will be wrote to the key matching engine for the next phase matching process.

Next, in the suffix state machine, states "2" and "3" are marked to the suffix accept state, which is the pattern matching state in the wildcard pattern. If the suffix accept state is reached, the corresponding key (search key) will be used to search the stored key in the key matching engine. When the search result is matched, the wildcard pattern will be recognized. Furthermore, the remaining states of the prefix state machine and the suffix state machine are marked as the transition state, which is only used for state transition. The corresponding actions of state status and TCAM/memory entries of the discrete-FA in simultaneous pattern matching are illustrated in Figs. 8 and 9. The key assignment of patterns will be discussed in more detail in Section III-D.

Patterns: { her, his, sh.*rs, he.*rs, hi.*e, sh.*e }

Current state	Char.	Next st.	State flag	Key	Segment ID
110 (5)	s	101 (6)	01	000 000 000	1
001 (2)	r	111 (7)	01	000 000 000	1
010 (1)	e	001 (2)	11	**001 010 000**	1
10X (3, 6)	h	011 (4)	11	**001 001 00X**	1
01X (1, 4)	i	110 (5)	00	**001 100 000**	1
XXX (X)	h	010 (1)	00	000 000 000	1
XXX (X)	s	100 (3)	00	000 000 000	1
XXX (X)	x	000 (0)	00	000 000 000	1

(a) TCAM/Memory entries for prefix state machine

Current state	Char.	Next st.	State flag	Key	Segment ID
01 (1)	s	10 (2)	10	**001 0XX 000**	1
XX (X)	r	01 (1)	00	000 000 000	1
XX (X)	e	11 (3)	10	**001 X0X 00X**	1
XX (X)	x	00 (0)	00	000 000 000	1

(b) TCAM/Memory entries for suffix state machine

Fig. 8. Data entries for prefix and suffix state machine. (a) TCAM/memory entries for prefix state machine (b) TCAM/memory entries for suffix state machine.

Action	State flag	State status	Action describe
1	00	Transition state	Abandon
2	01	Prefix accept state	Output match
3	10	Suffix accept state	Search corresponding key in key matching engines + Match → output match + Mismatch → output mismatch
4	11	Relay state	Search corresponding key in key matching engine + Match → update counter of the stored key, if need + Mismatch → key is wrote into key matching engine

Fig. 9. Corresponding actions of state status in discrete-FA.

Fig. 10 shows the state transitions for recognizing input strings. After processing each input character, the corresponding actions of the state will be activated to determine the following operations in the simultaneous matching engine. For example, after processing the two input characters "sh," the relay state "4" is reached and the corresponding action is activated to write the key in the key matching engine for the next phase matching process. After processing the input string "rhe," action 3 is activated to search the corresponding key with the stored key in the key matching engine. If there is a match at the output, a stored key exists in this case. On the other hand, action 4 is activated concurrently, in which the corresponding key is written to the key matching engine because of the character "e." After processing an input character "r," the pattern {her} is recognized by action 2. After processing the input string "rs," action 3 is activated to search the corresponding key with the stored key in the key matching engine and then the pattern {he.*rs} is recognized. According to this process, regular patterns and wildcard patterns can be recognized by this methodology because it not only could detect a match in the recognized pattern but also could

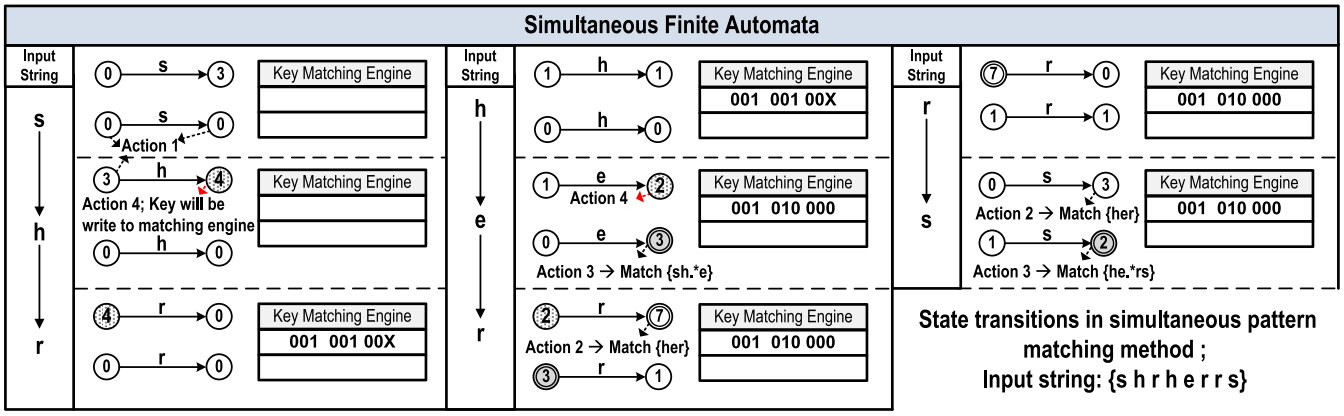


Fig. 10. Simultaneous pattern matching for input strings in the separated TCAM search engine architecture.

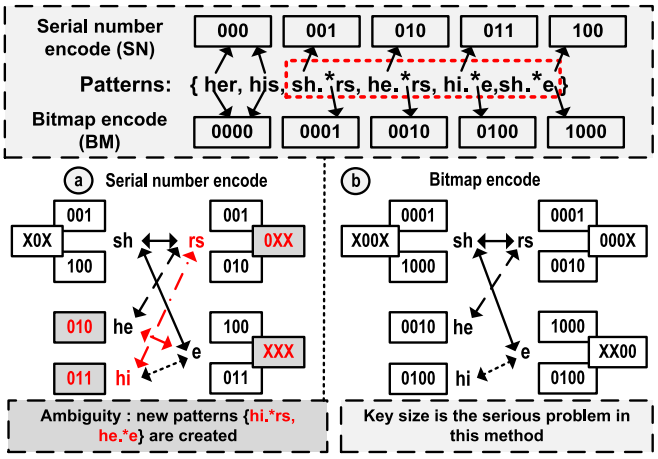


Fig. 11. Issues of key assignment. (a) SN. (b) BM.

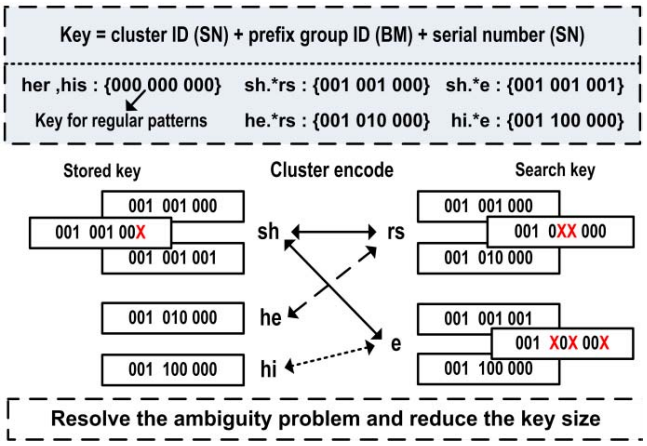


Fig. 12. Scheme of cluster encode.

traverse other possible matches in other patterns. Therefore, our proposed method utilizes two TCAM engines to represent prefix and suffix state machines, which reduces the number of active states to $O(1)$ for processing each character.

D. Resolve Ambiguity in Key Assignment

Another challenge in implementing simultaneous pattern matching methodology is to identify the fixed number of stored keys and search keys in each pattern in the memory array that negatively impact on key assignment. In the ClamAV antivirus signature, there are two special situations: the same prefix pattern combined with different suffix patterns and different prefix patterns combined with the same suffix pattern. Figs. 11 and 12 illustrate the issues of key assignment by using two conventional encode methods. In serial number encode (SN), although the key size is very short (1000 wildcard patterns only need 10 bits to present), an ambiguity problem will arise because the keys are merged by using the "don't care (X)" feature of TCAM. This results in a recognition error in pattern matching. New patterns "hi.*rs, he.*e" are created for pattern matching in addition to the predefined patterns. In contrast, although the ambiguity problem can be avoided in the bitmap encode (BM), the key size is another serious problem. If there are 1000 wildcard patterns in predefined patterns, each key requires 1000 bits to present in this encoding method.

To avoid ambiguity problems and key size problems, we propose a new key assignment method, cluster encode, to encode possible postfix pattern by a unique key ensures that follow-up patterns can accurately traverse all possible matches with limited hardware resources. In the proposed encoding method, the key is partitioned into three parts: a cluster ID, a prefix group ID, and a serial number, as shown in Fig. 12. The cluster ID and serial number use the serial number encoding to reduce the key size and the prefix group ID uses the bitmap encoding to avoid ambiguity problems because of the merging process of the keys. The key assignment flow and a corresponding example of the proposed encoding method are illustrated in Figs. 13 and 14, respectively.

The regular patterns will not be assigned any key in the proposed encoding method, which does not require the processing of the match result in advance in the simultaneous matching method, as shown in Figs. 13(a)–(h) and 14(a)–(h). On the other hand, the proposed method clusters wildcard patterns from the prefix segment and the suffix segment by accommodating the same prefix pattern combined with different suffix patterns and different prefix patterns combined

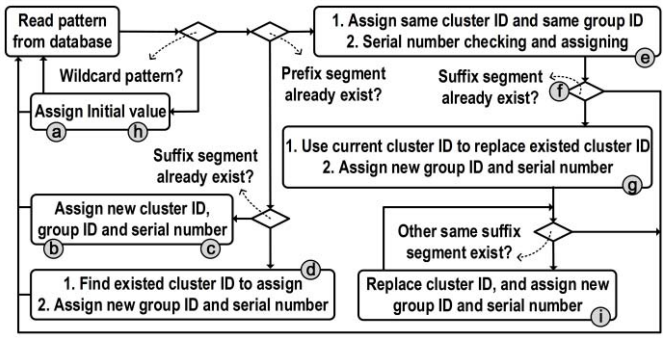


Fig. 13. Key assignment flow of the cluster encode.

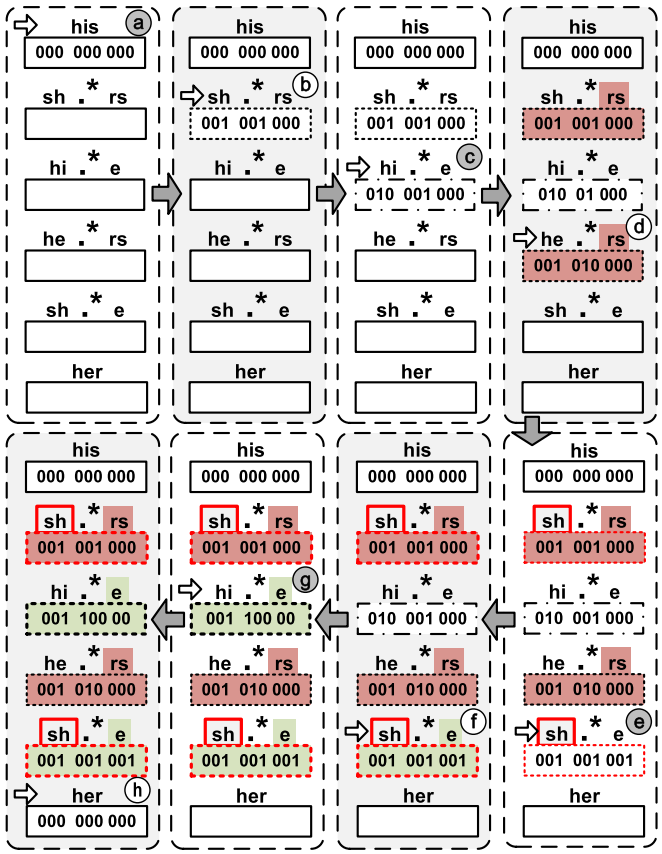
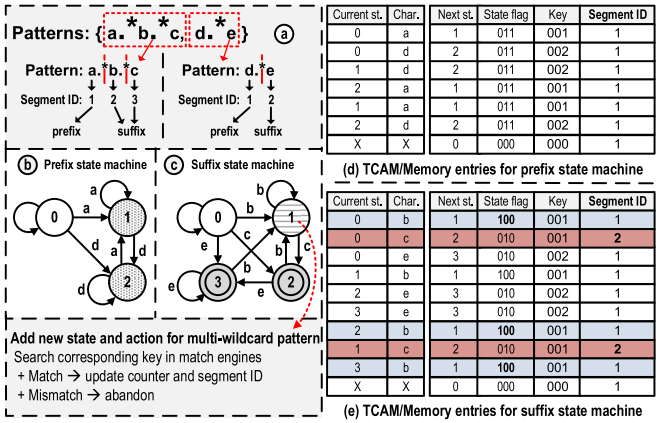


Fig. 14. Key assignment by the proposed flow.

with the same suffix pattern. If there is no relationship between the wildcard patterns, the different cluster ID are assigned to each wildcard pattern {sh.*rs, hi.*e} as illustrated in Figs. 13(b) and (c) and 14(b) and (c). When the suffix segment of the new pattern {he.***rs**} is the same as the previous pattern's suffix segment {sh.***rs**}, the pattern has the same cluster ID as plotted in Figs. 13(d) and 14(d).

However, when the new pattern {sh.*e} is read for key assignment, all the keys of wildcard patterns that are assigned before need to be checked for accuracy in pattern matching. First, the wildcard patterns {**sh**.*e, **sh**.*rs} have the same cluster ID because of the same prefix segment "**sh.**" This means that this pattern need to be contained in the same cluster by the proposed flow as plotted in Figs. 13(e) and 14(e). Next, the key of the previous pattern {hi.***e**} is reassigned, in which the suffix segment "**e**" is the same as the prefix segment of the current pattern {sh.***e**}, as shown in Figs. 13(f) and 14(f). This signifies that those patterns need to be contained in the same cluster and the key of the pattern {**hi.***e} needs to be reassigned by the proposed flow as illustrated in Figs. 13(g) and 14(g). In addition, patterns with previously assigned keys are checked to see whether the suffix segment is the same as the current pattern's suffix segment. If that is the case, we need to reassign the cluster ID, prefix group ID, and serial number of previous patterns, as plotted in Fig. 13(i). Although the prefix group ID requires more bits to present with bitmap encoding to avoid ambiguity problems, the total key size is no longer a serious problem. This is because the related wildcard patterns with the same prefix segment or suffix segment are collected in a cluster.



Current st.	Char.	Next st.	State flag	Key	Segment ID
0	a	1	011	001	1
0	d	2	011	002	1
1	d	2	011	002	1
2	a	1	011	001	1
1	a	1	011	001	1
2	d	2	011	002	1
X	X	0	000	000	1

Current st.	Char.	Next st.	State flag	Key	Segment ID
0	b	1	100	001	1
0	c	2	010	001	2
0	e	3	010	002	1
1	b	1	100	001	1
2	e	3	010	002	1
3	e	3	010	002	1
2	b	1	100	001	1
1	c	2	010	001	2
3	b	1	100	001	1
X	X	0	000	000	1

Fig. 15. (a) Construct discrete-FA for multiwildcard patterns. (b) Prefix state machine. (c) Suffix state machine. (d) TCAM/Memory entries for prefix state machine. (e) TCAM/Memory entries for suffix state machine.

IV. Multiwildcard Pattern

To recognize multiwildcard patterns, the proposed architecture is needed to modify with a minor extension. The key design difference is that several corresponding actions are complemented for matching follow-up segments of the wildcard pattern. This is done to recognize the segment of the wildcard pattern in the simultaneous pattern matching methodology and to ensure that each segment of the multiwildcard pattern is sequentially matched in the input strings.

In the proposed design, the prefix and suffix segments that are used to recognize the input strings are matched with predefined patterns and with the follow-up of the recognized wildcard patterns, respectively. Hence, all elements of the wildcard pattern, except the elements before the first wildcard, can be placed in the suffix segment to recognize, as shown in Fig. 15(a). For example, a multiwildcard pattern {a.*b.*c} can be transformed to three regular patterns {a, b, c}. The first element "a" belongs to the prefix DFA and the other elements "b" and "c" belong to the suffix DFA. The new field, segment ID, is used for identifying which segments are recognized by "b" or "c" in the matched sequence. Next, the two state machines for the prefix and suffix segments are constructed independently for the proposed architecture, as shown in Fig. 15(b) and (c).

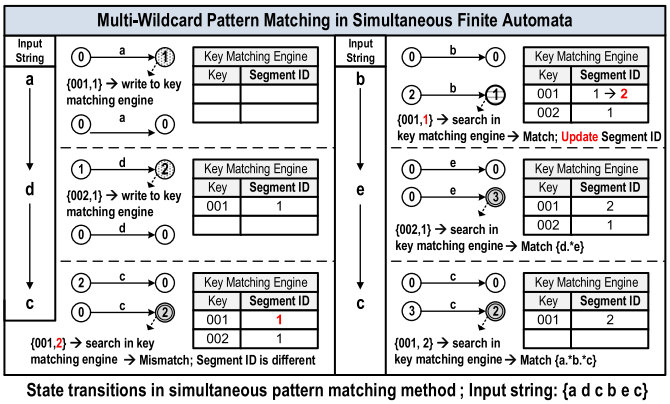


Fig. 16. Example for multiwildcard matching.

Fig. 16 shows an example of the state transitions for recognizing input strings in multiwildcard pattern matching. The example shows traversal in search engines with the input string "adcbec." After processing two input characters "ad" in the prefix segment, the corresponding key and segment ID {001, 1} and {002, 1} are written to the key matching engine, respectively. Next, when the following string matches "c" in the suffix segment, the corresponding data {001, 2} is used to search in the key matching engine. There is a mismatch in the key matching engine because the search data {001, 2} and stored data {001, 1} are different. When the subsequent string matches "b" in the suffix segment, the corresponding data {001, 1} is used to search in the key matching engine. If there is a match, the segment ID in the key matching engine will be modified to {001, 2} for the next search. After processing an input character "e," the pattern {d.*e} is recognized by the proposed design. Finally, when the subsequent input string is "c," the wildcard pattern {a.*b.*c} is recognized because the search data {001, 2} and stored data {001, 2} are the same. Thus, the multiwildcard pattern is recognized by adding a new segment ID in the proposed technique.

V. Experimental Results

We endeavored the following efforts to evaluate our design: first, we developed several simulators, which construct the transition tables for DFA, NFA, and discrete-FA, separated search engine architecture for simultaneous pattern matching, and provided a sophisticated key assignment method for pattern matching. Second, we implemented a synthetic workload generator to generate several workloads, which is based on BigDataBench [27]. BigDataBench includes several big data workloads with varying data inputs, which not only covers broad application scenarios but also includes diverse data sets. Third, we extracted patterns from a ClamAV antivirus database, and then inserted to the workloads for evaluating the proposed design, including 1518 wildcard patterns and 28 862 regular patterns [28].

During the inspection, the synthetic workloads are generated to be examined by different FAs to evaluate their performance and energy consumption. The detailed information of the workloads is presented in Table I. Since the proposed design is for wildcard pattern matching, we do not compare our design with other types of methodologies that are orthogonal and complementary to our design, such as state compression [25] and transition compression [29], [30]. In this paper, we utilize these algorithms to reduce storage requirement. The space complexity of the proposed design will be discussed in more detail in the following section.

TABLE I
Information of Workloads

	Workload					
	Amazon	Wiki	Payload 1	Payload 2	Payload 3	Payload 4
Strings	3333	336	1311	2732	1095	4195

TABLE II
Comparison of DFA, NFA, and Discrete-FA

		DFA	NFA	**Discrete-FA**
Regular Expression	Regular Pattern	✓	✓	✓
	Wildcard Pattern	N/A	✓	✓
Implement	**TCAM (Parallel)**	✓	N/A	✓
	Memory Lookup (Sequential)	✓	✓	N/A
High Matching Speed		✓	N/A	✓
Low Space Complexity		N/A	✓	✓
Number of Active States		*O(1)*	*O(N) N:#(wildcard patterns)*	***O(1)***
Key Matching Engine		N/A		✓
Capacity of Key Matching Engine				***O(N) * data width N:#(wildcard patterns)***

A. Discrete-FA Versus DFA and NFA

Table II lists the important features of three FA: DFA, NFA, and discrete-FA. We describe several observations as follows.

1) Implementation: The use of memory lookup in these methodologies results in slow matching speed in the matching process because of sequential memory access. In addition, the matching speed is reduced in advance in the NFA methods because of multiple active states, which increases the number of memory access. In contrast, parallel comparison and "don't care (X)" search abilities of TCAM are used to achieve high speed in the proposed design. More importantly, there are only two active states, which can be operated simultaneously by the proposed architecture.

2) Active State Effect: In terms of the matching process, the number of concurrently active states results in $O(N)$ memory access to process each character in the input strings. In the NFA-based methods, the number of active states is variable because of the nondeterministic features of the automata,

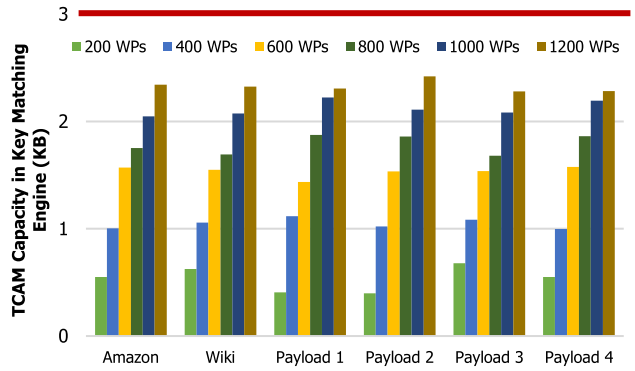


Fig. 17. TCAM capacity evaluation with variable wildcard patterns.

which depend on the number of wildcard patterns. As the number of active states increases, the memory access and comparison operations also increase in the matching process because each input character requires more than one search for detecting all possible matches. In contrast, in the proposed method, we use two isolated state machines to represent discrete-FA, which limits the complexity of the number of active states for processing each character to $O(1)$. Therefore, each input character can be examined immediately in the proposed matching process.

3) Space Complexity: Using the "don't care (X)" feature to minimize space consumption is a well-studied problem in TCAM search engines such as covered state encoding and CompactDFA [25], [29]. In this paper, we compress the data entries of discrete-FA by utilizing the covered state encoding, which has a superior performance to reduce TCAM capacity requirement. Although the proposed design uses an extra TCAM array in the key matching engine to process the search results for wildcard pattern matching, the space consumption of the design that uses other approaches will linearly increase with the number of patterns.

Considering these observations, we note that discrete-FA inherits all the advantages of DFA and NFA, which include parallel search ability, high matching speed, low space complexity, and limited active states for wildcard pattern matching. However, it has a disadvantage that a key matching engine is required to process the search results of automata.

B. Evaluation of Key Matching Engine

This paper makes our proposed method a viable alternative for pattern matching because of small TCAM requirements in a key matching engine. To evaluate the TCAM requirement of the key matching engine, we inserted wildcard patterns in the workloads, which were extracted from the ClamAV antivirus database, and then replaced the wildcard with an arbitrary number of characters to estimate and analyze the TCAM capacity requirement in the proposed design. Fig. 17 shows that we only needed a 3-kbyte TCAM array to serve the wildcard pattern matching in our design with variable wildcard patterns in the workloads. This is because the key length is dramatically reduced by our proposed design, as listed in Table III. In the worst case, all stored keys of wildcard patterns are stored in key matching engine for pattern matching, we only need a 7-kbyte TCAM to serve the wildcard pattern matching in our design. As a result, the proposed design needs a small overhead to efficiently and accurately recognize wildcard patterns. The size of the hardware depends on the number of stored keys and different lifetimes of stored keys for pattern matching across all traces and all simulations in applications.

TABLE III
COMPARISON OF KEY ASSIGNMENT METHODS

Encode method	Number of wildcard patterns (ClamAV)	Key length
Bitmap	1518	1518
Proposed	1518	**27**

TABLE IV
PARAMETERS OF S-TCAM AND nvTCAM

		S-TCAM [29]	nvTCAM [32]
Compare	Latency	1x	0.89x
	Power	1x	1.65x
Leakage		1x	0.09x

TABLE V
PARAMETERS OF SRAM AND RRAM

		SRAM [29]	RRAM [32]
Latency	Read	1.03x	1.55x
	Write	1x	1.94x
Power	Read	1x	1.02x
	Write	1x	1.81x
Leakage		1x	0.11x

C. Performance and Energy Consumption

For IoT applications in embedded environments, regular expression by traditional memory lookup implementation will result in nondeterministic computation overhead in NFA-based methods. All memory/TCAM module parameters for this experiment, as presented in Tables IV and V, such as read/search latency and energy were adopted from previous studies [31], [32], which include SRAM-based TCAM (S-TCAM), SRAM, and RRAM. The simulation that includes TCAM arrays, priority encoders, and memory arrays is done by HSPICE.

1) Performance Evaluation: One challenge in implementing NFA-based methods by memory lookup implementation may be to identify at runtime the number of the memory access and sequential comparisons that negatively impact system execution time. For each input character, these methods usually require many concurrent comparison operations and memory access. When the number of the memory access is increased, it will stall comparison operations and then increase system execution time for pattern matching. The TCAM-based implementations can search each TCAM macro concurrently to avoid sequential accessing data for comparison and to have deterministic computations to reduce system

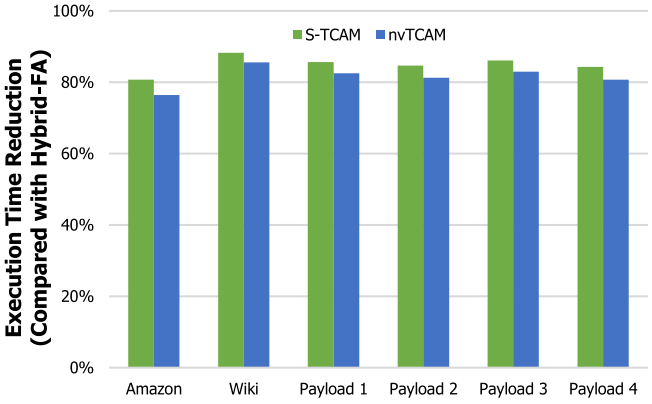


Fig. 18. Execution time reduction of TCAM search engines by using discrete-FA.

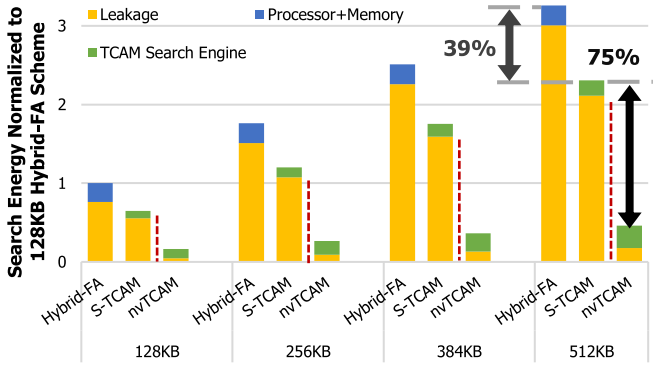


Fig. 19. Energy comparison of using discrete-FA.

execution time. In terms of search operations, the results of execution time from memory lookup implementation indicate serious overhead compared with the TCAM-based architecture. Fig. 18 shows that in comparison with hybrid-FA [21], using the proposed solution can reduce 76%–88% execution time depending on workload behaviors. The dramatically decreasing execution time of the proposed design is caused by the reducing the mass of sequential memory access and comparison operations in the matching process due to the parallel search abilities of TCAM. The improvement of execution time in nonvolatile TCAM (nvTCAM) search engines is less than S-TCAM search engine because the access latency of nonvolatile memory is longer than SRAM, as shown in Table V. Although using nvTCAM has 4% overhead in the system execution compared with the S-TCAM search engine, the total energy is reduced because of nonvolatility. The energy consumption of TCAM search engines will be discussed in more detail in Section VI.

2) Search Energy Improvement: The patterns were extracted from the ClamAV antivirus database in system evaluation for various applications such as IoT, wireless sensor networks, and wearable devices because not all signatures were suitable in embedded systems. Therefore, we need different capacities to store them for different amounts of signatures in various applications. In addition, three implementations, memory lookup, S-TCAM and nvTCAM, were used to implement for energy exploration. Fig. 19 shows that the energy consumption can be broken down into TCAM search engine energy, processor energy, memory energy, and leakage. Compared with hybrid-FA (using SRAM), the S-TCAM can save the total energy up to around 39%. On the other hand, in the dynamic energy consumption, S-TCAM can save around 23% compared with hybrid-FA. This is due to the reduction of the number of memory access to retrieve data for comparison because the comparison of TCAM macros occurs in parallel.

VI. Discussion

A. Using Nonvolatile TCAM in Network Security

Nonvolatile memory (RRAM) and nvTCAM have been designed to achieve a small area and fast/low-power wake-up operations. Their high-density property and competitive search time with near-zero leakage energy are more suitable to implement in embedded systems compared to S-TCAM. As presented in Tables IV and V, the parameters of RRAM and nvTCAM (RCSD-4T2R) are implemented with Industrial Technology Research Institute's 90-nm process with the back end of line RRAM by HSPICE [32]. Fig. 18 shows that using nvTCAM can reduce execution times by 66% on average. The improvement of using nvTCAM is affected by the long write/read latency of nonvolatile memory. Although using nvTCAM has 6% overhead compared with S-TCAM, the overall energy is reduced because of nonvolatility in the total energy consumption.

In the energy evaluation, even though the TCAM array energy consumption of nvTCAM was higher than that of the S-TCAM, the bulk of the energy consumption in the nvTCAM array was determined by the number of search operations, due to the high costs of the search operation. Using nvTCAM in the proposed architecture can result in a reduction in total energy usage of around 75%, compared with S-TCAM, as shown in Fig. 19. This is because nvTCAMs, which have been designed to achieve low standby power, can reduce leakage consumption by 91%.

B. Applicability and Future Work

With significant advancements in technology scaling, the proposed design can be used not only in network security but also in broader applications such as wireless sensor networks, biometrics, and vehicle license plate recognition. In the applications, regular expressions are a key function to fast analyze unstructured textual data. Our work fully utilizes the features of TCAM and does not need a specific architecture modification for supporting regular expression matching.

In the future work, saving the energy consumption and storage space by improving the energy consumption and performance by eliminating priority encoders will be considered. In the TCAM-based search engines, the priority encoders are needed to obtain the first match (highest priority) entry in each search operation. This implies that data should be stored in an order with sorting their lengths. The restrictions on ordering result in increased energy consumption and decreased performance for data updates and search operations. For discarding priority encoders, previous works used the length information of the matched data to decide the longest prefix match data. In the future, we will explore the effectiveness of our proposed

TABLE VI
COMPARISON OF REGULAR EXPRESSION MATCHING METHODOLOGY WITH THE OTHERS

	Regular Expression Grouping [37-39]	Transition Compression [29, 30, 40-43]	Hybrid Construction [17, 18, 20-22]	TCAM-based search engines [14-16,44-46]	Proposed
Matching Speed	Moderate; multiple DFAs run concurrently	Limited to the additional memory lookups	Moderate, due to the non-deterministic in NFA part	Fast due to parallel comparison ability	
Space Consumption	Moderate; avoids state inflation in the DFA of each group	Dependent on the regular expression sets	Moderate; limited to the DFA part	Small; Compressed transitions by "don't care (X)" feature	
Scalability	Restricted by the scale and complexity of regular expressions	Restricted by the complexity of regular expressions	Restricted by the scale of regular expressions	Scalable, the space and speed are comparable with NFA and DFA, respectively.	
Practicality	Cannot recognize wildcard patterns	Cannot recognize wildcard patterns	Preprocessing complexity and multiple active states need to be considered	***Cannot recognize wildcard patterns***	***Supports wildcard patterns***
Main Idea	Dividing regular expressions into several groups for constructing independent DFA	Eliminating redundant transitions and states of automata	Constructing the hybrid finite automata by utilizing the features of DFA and NFA	Utilizing parallel comparison and "don't care (X)" search ability to improve speed and space	

separated TCAM search engine integrated to those designs for reducing energy consumption in update and search operations for an energy-efficient TCAM-based search engine design.

VII. PRIOR MATCHING METHODOLOGY

In the past, several studies have explored the memory problem of DFA or the speed problem of NFA by employing the characteristics of memory or algorithm [33]–[36]. These studies can be categorized into four directions based on their implementation platforms: regular expression grouping, transition compression, hybrid construction, and TCAM-based search engines. Table VI summarizes the brief comparison of the previous works.

A. Regular Expression Grouping

Yu *et al.* [37] first used the grouping method to divide the given set of regular expressions into the fewest groups through greedy heuristics. This can efficiently improve matching speed and reduce memory requirement because DFA of each group is run independent of the regular expression. Rohrer *et al.* [38] found the optimal distribution of regular expressions as an energy minimization problem and aimed to optimize storage efficiency and performance by distributing regular expressions to a limited number of scan engines. Majumder *et al.* [39] proposed an agglomerative clustering technique to optimize the tradeoffs among overall DFA size and processing cost by DFA state caching and regular expression grouping. However, the utility of regular expression is not suitable in practice, which depends on the scale and complexity of the given regular expression set. More importantly, the DFA state explosion problem, which is caused by a single regular expression, cannot be handled.

B. Transition Compression

To reduce the memory consumption of DFA, several studies have explored elimination of redundant states and transitions of automata by compression methods [40]–[43]. Bremler-Barr *et al.* [29] presented an efficient encoding scheme, CompactDFA, to compress the DFA entries and eliminate all cross transitions in the TCAM-based implementation by constructing the common suffix tree. Yun *et al.* [25] proposed covered state encoding that eliminates all the failure transitions of Aho-Corasick NFA in a TCAM-based solution. Kumar *et al.* [30] proposed a technique for parsing regular expressions using delayed input DFA, which can reduce memory requirements by using default transitions. The compression ratios of these solutions are not very stable because they rely on the internal structure of the automaton tree in patterns.

C. Hybrid Construction

Becchi *et al.* [21] proposed a hybrid-FA solution, hybrid-FA, to improve matching speed and prevent state explosion, through single head-DFA and tail-NFAs. Becchi *et al.* [22] explored the recognition ability of Perl-compatible regular expressions, which include character repetitions and back-references by extended-hybrid-FA. Liu *et al.* [18] presented an efficient matching algorithm, called dual FA, to use linear FA (LFA) to represent the NFA states and an extended DFA (EDFA) to represent the remainder. The performance of the matching process in the EDFA part is good because of the DFA features. In contrast, the performance of the matching process in the LFA part is greatly reduced because of the nondeterministic feature of NFA. Yang *et al.* [20] proposed semideterministic FA (SFA) that offer an effective tradeoff between the computation complexity of NFA and the space complexity of DFA for regular expression matching. The SFA-based solution clusters all NFA states into the fewest subsets by using "state grouping" heuristics. However, the algorithm is quite complex for judging state grouping and it is infeasible to implement large-scale regular expression matching in practice.

D. TCAM-Based Search Engines

In the early studies, the TCAM-based search engines were designed for plain string matching, which are based on the

"don't care (X)" search ability [44]–[46]. Yu *et al.* [44] proposed a multibyte multiple-string matching algorithm with limited support for wildcards in the TCAM-based search engines. Weinsberg *et al.* [46] presented a TCAM-based solution, rotating TCAM algorithm, to achieve speedup of matching multiple patterns in a single operation. Recently, TCAM-based search engines have been proposed as a promising approach for regular expression matching because of fast and scalable multipattern matching. These TCAM-based search engines run a unique state machine instance for each flow, which can be used to monitor patterns spread across multiple packets.

Several studies have explored regular expression matching issues for regular expression matching by employing the characteristics of TCAM. Alicherry *et al.* [16] presented a classical TCAM-based search engine architecture with a state-encoding scheme for the pattern matching algorithm in small TCAMs, which is based on the Aho–Corasick algorithm. Meiners *et al.* [14] used the optimization methods that include transition sharing, table consolidation, and variable striding to reduce TCAM space consumption and improve regular expression matching speeds for a regular expression matching algorithm. Peng *et al.* [15] evaluated wildcard pattern matching by TCAM-based search engines using a chain-based DFA deflation method, which exploits the structural connection between NFA and DFA. *A priori* assumption of these designs is that a DFA can be built for a given wildcard pattern of regular expression. Unfortunately, for wildcard patterns in practice, building a DFA with a reasonable number of states is impossible because in a wildcard, any number of arbitrary characters can be repeated.

Previous studies of regular expression matching in the TCAM-based search engines have focused on how to reduce memory requirements and improve the speed of search operation. Different from the existing matching algorithms, our work fully exploits the features of discrete-FA and TCAM to build a foundation for both the recognition ability of wildcard patterns and linear scalability by partition of regular expression.

VIII. Conclusion

This paper proposes an efficient separated search engine based on the detailed analysis of wildcard pattern properties. It indicates that two active states with two transitions are sufficient for wildcard-pattern matching problems. A simple architecture modification based on simultaneous matching methodology is proposed as an alternative. This exhibits accurate traversal and traverses all possible matches by two separated engines to represent discrete-FA. We used the cluster encoding method to resolve problems of ambiguity and key size for the simultaneous pattern matching method. In our experimental results, we only needed a small sized TCAM array in the key matching engine for various applications in the proposed design. By adopting TCAM-based solutions, S-TCAM and nvTCAM, the search energy reduction is approximately 39% and 84%, respectively, because memory access and comparison operations are relatively reduced in the TCAM array. We also demonstrated that the proposed design is an attractive option for wildcard pattern matching in embedded systems.

References

[1] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future Internet: The Internet of Things architecture, possible applications and key challenges," in *Proc. ICCSEE*, Dec. 2012, pp. 257–260.

[2] K. Zhao and L. Ge, "A survey on the Internet of Things security," in *Proc. CIS*, Dec. 2013, pp. 663–667.

[3] A. Riahi, Y. Challal, E. Natalizio, Z. Chtourou, and A. Bouabdallah, "A systemic approach for IoT security," in *Proc. DCOSS*, May 2013, pp. 351–355.

[4] H. Ning and H. Liu, "Cyber-physical-social based security architecture for future Internet of Things," in *Proc. AIT*, 2012, pp. 1–7.

[5] Z.-K. Zhang, M. C. Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen, and S. Shieh, "IoT security: Ongoing challenges and research opportunities," in *Proc. SOCA*, Nov. 2014, pp. 230–234.

[6] S. S. Basu, S. Tripathy, and A. R. Chowdhury, "Design challenges and security issues in the Internet of Things," in *Proc. TENSYMP*, May 2015, pp. 90–93.

[7] A. V. Aho and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," *Commun. ACM*, vol. 18, no. 6, pp. 333–340, Jun. 1975.

[8] J. Van Lunteren, C. Hagleitner, T. Heil, G. Biran, U. Shvadron, and K. Atasu, "Designing a programmable wire-speed regular-expression matching accelerator," in *Proc. MICRO*, Dec. 2012, pp. 461–472.

[9] K. Atasu, "Resource-efficient regular expression matching architecture for text analytics," in *Proc. ASAP*, Jun. 2014, pp. 1–8.

[10] K. Wang, Z. Fu, X. Hu, and J. Li, "Practical regular expression matching free of scalability and performance barriers," *Comput. Commun.*, vol. 65, pp. 97–119, Dec. 2016.

[11] N. L. Or, X. Wang, and D. Pao, "MEMORY-based hardware architectures to detect ClamAV virus signatures with restricted regular expression features," *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1225–1238, Apr. 2016.

[12] C.-J. Cheng, C.-C. Wang, W.-C. Ku, T.-F. Chen, and J.-S. Wang, "A scalable high-performance virus detection processor against a large pattern set for embedded network security," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 5, pp. 841–854, May 2012.

[13] P. Piyachon and Y. Luo, "Design of high performance pattern matching engine through compact deterministic finite automata," in *Proc. DAC*, Jun. 2008, pp. 852–857.

[14] C. R. Meiners, J. Patel, E. Norige, A. X. Liu, and E. Torng, "Fast regular expression matching using small TCAM," *IEEE/ACM Trans. Netw.*, vol. 22, no. 1, pp. 94–109, Feb. 2014.

[15] K. Peng, S. Tang, M. Chen, and Q. Dong, "Chain-based DFA deflation for fast and scalable regular expression matching using TCAM," in *Proc. ANCS*, Oct. 2011, pp. 24–35.

[16] M. Alicherry, M. Muthuprasanna, and V. Kumar, "High speed pattern matching for network IDS/IPS," in *Proc. ICNP*, Nov. 2006, pp. 187–196.

[17] Y. Sun, H. Liu, V. C. Valgenti, and M. S. Kim, "Hybrid regular expression matching for deep packet inspection on multi-core architecture," in *Proc. ICCCN*, Aug. 2010, pp. 1–7.

[18] C. Liu and J. Wu, "Fast deep packet inspection with a dual finite automata," *IEEE Trans. Comput.*, vol. 62, no. 2, pp. 310–321, Feb. 2013.

[19] B. C. Brodie, D. E. Taylor, and R. K. Cytron, "A scalable architecture for high-throughput regular-expression pattern matching," in *Proc. ISCA*, Jul. 2006, pp. 191–202.

[20] Y.-H. E. Yang and V. K. Prasanna, "Space-time tradeoff in regular expression matching with semi-deterministic finite automata," in *Proc. INFOCOM*, Apr. 2011, pp. 1853–1861.

[21] M. Becchi and P. Crowley, "A hybrid finite automaton for practical deep packet inspection," in *Proc. CoNEXT*, Dec. 2007, Art. no. 1.

[22] M. Becchi and P. Crowley, "Extending finite automata to efficiently match Perl-compatible regular expressions," in *Proc. CoNEXT*, Dec. 2008, Art. no. 25.

[23] A. N. Arslan, B. George, and K. Stor, "New algorithms for pattern matching with wildcards and length constraints," *Discrete Math., Algorithms Appl.*, vol. 7, no. 3, p. 1550032, 2015.

[24] A. N. Arslan, D. He, Y. He, and X. Wu, "Pattern matching with wildcards and length constraints using maximum network flow," *J. Discrete Algorithms*, vol. 35, pp. 9–16, Nov. 2015.

[25] S. Yun, "An efficient TCAM-based implementation of multipattern matching using covered state encoding," *IEEE Trans. Comput.*, vol. 61, no. 2, pp. 213–221, Feb. 2012.
[26] K. Huang, L. Ding, G. Xie, D. Zhang, A. X. Liu, and K. Salamatian, "Scalable TCAM-based regular expression matching with compressed finite automata," in *Proc. ANCS*, Oct. 2013, pp. 83–94.
[27] L. Wang *et al.*, "BigDataBench: A big data benchmark suite from Internet services," in *Proc. HPCA*, Feb. 2014, pp. 488–499.
[28] *ClamAV Anti-Virus System*. accessed on Oct. 30, 2015. [Online]. Available: http://www.clamav.net
[29] A. Bremler-Barr, D. Hay, and Y. Koral, "CompactDFA: Scalable pattern matching using longest prefix match solutions," *IEEE/ACM Trans. Netw.*, vol. 22, no. 2, pp. 415–428, Apr. 2014.
[30] S. Kumar, J. Turner, and J. Williams, "Advanced algorithms for fast and scalable deep packet inspection," in *Proc. ANCS*, Dec. 2006, pp. 81–92.
[31] H.-J. Tsai *et al.*, "Energy-efficient non-volatile TCAM search engine design using priority-decision in memory technology for DPI," in *Proc. DAC*, Jun. 2015, Art. no. 100.
[32] L.-Y. Huang *et al.*, "ReRAM-based 4T2R nonvolatile TCAM with 7x NVM-stress reduction, and 4x improvement in speed-wordlength-capacity for normally-off instant-on filter-based search engines used in big-data processing," in *Proc. VLSIC*, Jun. 2014, pp. 1–2.
[33] T. Song, W. Zhang, D. Wang, and Y. Xue, "A memory efficient multiple pattern matching architecture for network security," in *Proc. INFOCOM*, Apr. 2008, pp. 673–681.
[34] Y. Sun, V. C. Valgenti, and M. S. Kim, "NFA-based pattern matching for deep packet inspection," in *Proc. ICCCN*, Jul. 2011, pp. 1–6.
[35] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner, "Algorithms to accelerate multiple regular expressions matching for deep packet inspection," in *Proc. SIGCOMM*, Sep. 2006, pp. 339–350.
[36] M. Bando, N. S. Artan, and H. J. Chao, "Scalable lookahead regular expression detection system for deep packet inspection," *IEEE/ACM Trans. Netw.*, vol. 20, no. 3, pp. 699–714, Jun. 2012.
[37] F. Yu, Z. Chen, Y. Diao, T. V. Lakshman, and R. H. Katz, "Fast and memory-efficient regular expression matching for deep packet inspection," in *Proc. ANCS*, Dec. 2006, pp. 93–102.
[38] J. Rohrer, K. Atasu, J. van Lunteren, and C. Hagleitner, "Memory-efficient distribution of regular expressions for fast deep packet inspection," in *Proc. CODES+ISSS*, Oct. 2009, pp. 147–154.
[39] A. Majumder, R. Rastogi, and S. Vanama, "Scalable regular expression matching on data streams," in *Proc. SIGMOD*, Jun. 2008, pp. 161–172.
[40] D. Ficara, S. Giordano, G. Procissi, G. Vitucci, G. Antichi, and A. D. Pietro, "An improved DFA for fast regular expression matching," in *Proc. SIGCOMM*, Oct. 2008, pp. 29–40.
[41] T. Liu, Y. Yang, Y. Liu, Y. Sun, and L. Guo, "An efficient regular expressions compression algorithm from a new perspective," in *Proc. INFOCOM*, Apr. 2011, pp. 2129–2137.
[42] J. van Lunteren and A. Guanella, "Hardware-accelerated regular expression matching at multiple tens of Gb/s," in *Proc. INFOCOM*, Mar. 2012, pp. 1737–1745.
[43] J. Patel *et al.*, "Bypassing space explosion in regular expression matching for network intrusion detection and prevention systems," in *Proc. NDSS*, 2012, pp. 1–15.
[44] F. Yu, R. H. Katz, and T. V. Lakshman, "Gigabit rate packet pattern-matching using TCAM," in *Proc. ICNP*, Oct. 2004, pp. 174–183.
[45] C.-H. Lin and S.-C. Chang, "Efficient pattern matching algorithm for memory architecture," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 1, pp. 33–41, Jan. 2011.
[46] Y. Weinsberg, S. Tzur-David, D. Dolev, and T. Anker, "High performance string matching algorithm for a network intrusion prevention system (NIPS)," in *Proc. HPSR*, Jun. 2006, pp. 147–154.

Hsiang-Jen Tsai received the M.S. degree in computer science and engineering from National Chung Hsing University, Taichung, Taiwan, in 2009, and the Ph.D. degree in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2017.

His current research interests include multicore system-on-chip design, embedded system design, low-power methodology, nonvolatile memory design, and computer architecture.

Chien-Chih Chen received the M.S. degree in computer science and information engineering from National Chung Cheng University, Chia-Yi, Taiwan, in 2009, and the Ph.D. degree in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2016.

His current research interests include homogeneous/heterogeneous multicore architecture design, performance evaluation, and physical implementation/verification.

Yin-Chi Peng received the M.S. degree in computer science and information engineering from National Chung Cheng University, Chia-Yi, Taiwan, in 2011, and the Ph.D. degree in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2017.

His current research interests include multicore system-on-chip design, embedded system design, and computer architecture.

Ya-Han Tsao received the B.S. degree in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2014, where she is currently pursuing the M.S. degree in computer science.

Her current research interests include computer architecture and embedded system.

Yen-Ning Chiang received the B.S. degree in engineering and system science from National Tsing Hua University, Hsinchu, Taiwan, in 2015, where she is currently pursuing the M.S. degree with the Institute of Electronics Engineering.

Her current research interests include circuit design of emerging nonvolatile memory.

Wei-Cheng Zhao received the B.S. degree in electrical engineering from National Tsing Hua University, Hsinchu, Taiwan, in 2015, where he is currently pursuing the M.S. degree in electronic engineering.

His current research interests include SRAM and ternary content addressable memory circuit design.

Meng-Fan Chang (M'05–SM'14) received the M.S. degree from Pennsylvania State University, State College, PA, USA, and the Ph.D. degree from National Chiao Tung University, Hsinchu, Taiwan.

He was with industry over 10 years. From 1996 to 1997, he designed memory compilers at Mentor Graphics, Bedminster, NJ, USA. From 1997 to 2001, he designed embedded SRAMs and managed the Memory-IP Validation Program at the Design Service Division, TSMC, Hsinchu, Taiwan. From 2001 to 2006, he was the Director at IPLib, Hsinchu, Taiwan, where he developed embedded SRAM and ROM compilers, flash macros, and flat-cell ROM products. His current research interests include circuit designs for volatile and nonvolatile memory, ultralow-voltage systems, and memristor logics.

Dr. Chang was a recient of the Ta-You Wu Memorial Award of the National Science Council, Taiwan, in 2011. He is the corresponding author of several ISSCC and Symposium on very large scale integration papers. He served on the Program/Organization Committee for the IEEE MTDT from 2007 to 2009. He has been serving on the Program Committee for the IEEE A-SSCC since 2011. He has also been serving as the Associate Executive Director for Taiwan's five-year National Program of Intelligent Electronics since 2011.

Tien-Fu Chen (S'90–M'93) received the B.S. degree in computer science from National Taiwan University, Taipei, Taiwan, in 1983, and the M.S. and Ph.D. degrees in computer science and engineering from the University of Washington, Seattle, WA, USA, in 1991 and 1993, respectively.

He was a System Software Engineer with Wang Computer Ltd., Taipei, Taiwan, Taiwan, for three years. He is currently a Professor with the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan. He has authored several widely-cited papers on dynamic hardware prefetching algorithms and designs. He has made contributions to processor design and system-on-chip (SoC) design methodology. His current research interests include multithreading/multicore media processors, on-chip networks, low-power architecture techniques, related software support tools, SoC design environment, computer architectures, SoC design, design automation, and embedded software systems.