IEICE TRANS. COMMUN., VOL.E93–B, NO.6 JUNE 2010

LETTER

A Pattern Partitioning Algorithm for Memory-Efficient Parallel String Matching in Deep Packet Inspection

HyunJin KIM[†], **Hyejeong HONG**[†], **Dongmyoung BAEK**[††], *Nonmembers*, *and* **Sungho KANG**[†a)], *Member*

SUMMARY This paper proposes a pattern partitioning algorithm that maps multiple target patterns onto homogeneous memory-based string matchers. The proposed algorithm adopts the greedy search based on lexicographical sorting. By mapping as many target patterns as possible onto each string matcher, the memory requirements are greatly reduced.

key words: *computer network security, deep packet inspection, finite state machine, pattern matching, and network monitoring*

1. Introduction

In order to defend networking environments from numerous attacks, deep packet inspection (DPI) devices can be adopted to detect harmful packet payloads. The string matching engine is essential to identify target patterns. Especially, the deterministic-finite automaton (DFA)-based string matching engine has been preferred due to the regularity with a predetermined number of state transitions [1]. On the other hand, because the number of target patterns increases, DPI devices should provide scalability in order to update target patterns easily. For achieving both the regularity and updatability, the high performance string matching engine could be implemented with homogeneous memory-based string matchers for mapping multiple DFAs. With the homogeneous string matchers, target patterns are identified in parallel string matching. Based on the Aho-Corasick algorithm [2], several bit-split DFA-based string matching schemes were proposed in order to map compressed DFAs onto multiple finite state machine (FSM) tiles in each string matcher [3] and [4].

The target pattern lengths vary, so that the numbers of mapped target patterns onto homogeneous string matchers can be different. Therefore, the variety of target pattern lengths must be a serious problem in achieving both regularity and memory efficiency. The pattern partitioning decides which target patterns are to be mapped onto each string matcher. In the bit-split DFA-based string matching schemes in [3] and [4], several pattern partitioning approaches were proposed. A set of target patterns were partitioned sequentially into small groups to increase the number of common prefixes in each string matcher. The total lengths of target patterns in the sequentially partitioned sets for homogeneous string matchers can be different from each other; therefore, the large memory could be unused in the existing pattern partitioning approaches. A memory-efficient pattern partitioning algorithm is required in order to reduce the unused memory in homogeneous string matchers.

This paper proposes a pattern partitioning algorithm for the DFA-based parallel string matching engine with homogeneous string matchers. An initial set of mappable target patterns is obtained based on the lexicographical order, which increases the number of shared common prefixes in a string matcher. The proposed algorithm searches for other target patterns that can be mapped simultaneously with the initial set onto the string matcher greedily. The number of adopted string matchers is decreased by mapping as many target patterns as possible onto each homogeneous string matcher. By increasing the number of mapped target patterns onto each string matcher, the unused memory is greatly reduced, so that the problem of various target pattern lengths can be mitigated.

2. String Matching Engine Architecture

The proposed pattern partitioning can be applied to the general bit-split memory-based string matching engine with e homogeneous string matchers in Fig. 1. Each state has 2^n pointers for the next state based on an n-bit input. In Fig. 1(a), each FSM takes two input bits, so that the number of FSM tiles is four. In Fig. 1(b), each FSM tile contains s states and takes n bits of one character during each cycle. The address of a state indicates a partial match vector (PMV), where the number of bits in a PMV p refers to the maximum number of mapped target patterns onto a string matcher. In a PMV, the i-th bit represents whether the i-th

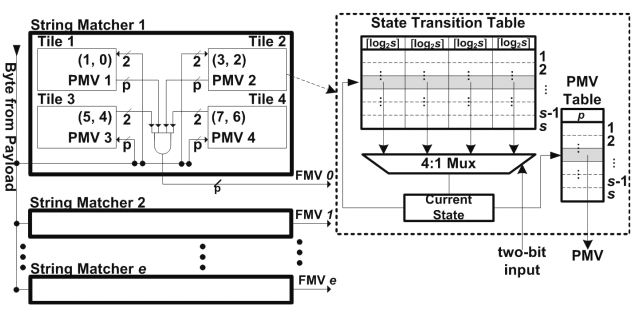

Fig. 1 Architecture of string matching engine and FSM tile.

Manuscript received December 3, 2009.
Manuscript revised February 10, 2010.

[†]The authors are with the Dept. of Electrical and Electronic Eng., Yonsei University, Seoul, Korea.
[††]The author is with Broadcasting and Telecommunication Convergence Research Laboratory, ETRI, Daejeon, Korea.
a) E-mail: shkang@yonsei.ac.kr

DOI: 10.1587/transcom.E93.B.1612

pattern is matched or not in the state. A matched pattern can be recognized with full match vectors (FMVs), which are obtained with the logical AND operation of PMVs from all FSM tiles. When a target pattern is the suffix of other target patterns, more than one bit in FMVs are true. For example, when a pattern "abc" is matched, a pattern "bc" is matched, too.

3. Proposed Pattern Partitioning Algorithm

In the proposed pattern partitioning, target patterns to be mapped are searched greedily based on the lexicographically sorted list of target patterns. By adopting the lexicographical sorting, the number of common prefixes in a string matcher can increase. Considering the number of states s in each FSM tile and the number of bits in a PMV p, the proposed pattern partitioning maps as many target patterns as possible onto each string matcher. The pseudocode of the pattern partitioning algorithm for a string matcher is described in Fig. 2, where a lexicographical sorted list of target patterns and several parameters of the string matcher architecture are given for the inputs.

As shown in the pseudocode, there are two main steps. In step 1, the initial DFAs, which are denoted as ***dfas***, are obtained by calling *Build_Initial_DFAs*. The lexicographical ordering could increase the number of common prefixes. In step 2, other patterns that can be mapped with the initial DFAs are greedily searched until resource constraints of s and p are violated. In step 2, the number of mapped target patterns is increased by consuming unused resource left after processing step 1. With the obtained DFAs ***dfas***, the Aho-Corasick algorithm is applied to add failing pointers. For the bit-split string matching, the PMVs of each FSM can be obtained by the construction in [3]. After the construction, FSM tile contents can be obtained.

When target patterns are mapped sequentially based on a determined order, if a target pattern cannot be mapped onto a string matcher, the target pattern is mapped onto a new empty string matcher. For example, let us assume that a string matcher has an FSM with eight input bits and eight states. In this case, a state in each FSM is reserved for the initial state. In addition, "add," "added," "caused," and "do" are assumed to be the target patterns that are lexicographically ordered. First, target patterns "add" and "added" can be the elements in the DFAs of the first string matcher, where the common prefixes for "add" can be shared between the two target patterns. The third target pattern "caused" cannot be the element in the initial DFAs because the number of states in an FSM is only eight. In addition, the subsequent pattern "do" cannot be mapped with the third target pattern "caused" onto a string matcher. Therefore, three string matchers are required for the pattern partitioning approaches that map target patterns sequentially based on a determined order. The proposed pattern partitioning merges the fourth pattern "do" into the initial DFAs that contain target patterns "add" and "added." The target pattern "caused" should be mapped onto a new empty string matcher. Therefore, only two string matchers are required by applying the proposed pattern partitioning.

```
FUNCTION Pattern_Partitioning (sorted patterns T, string matcher M)
    SET unmapped target patterns t to T;
    // step 1: sequential pattern mapping
    FOR π = p (M) to 1
        CALL Build_Initial_DFAs with front π target patterns of t
        RETURNING DFAs dfas;
        IF MAX(NUN_STATES(dfas)) > s (M) THEN CONTINUE;
        ELSE REMOVE π target patterns from t; BREAK;
        ENDIF
    ENDFOR
    // step 2: greedy search for mapping more patterns
    FOR φ=1 to Num (t)
        CALL Merge_DFAs with the φ-th target pattern of t and dfas
        RETURNING DFAs mdfas;
        IF MAX(NUM_STATES(mdfas)) ≥ s (M) CONTINUE;
        ELSE IF MAX(NUM_PATTERNS(mdfas)) ≥ p (M) CONTINUE;
        ELSE SET dfas to mdfas;
        ENDIF
    ENDFOR
ENDFUNCTION with RETURNING dfas
```

Fig. 2 Pseudocode of proposed pattern partitioning.

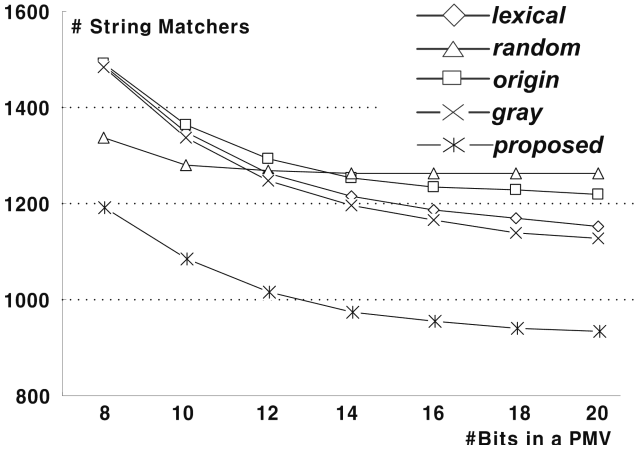

Fig. 3 Performance comparisons.

Until all target patterns are mapped onto multiple string matchers, the pattern partitioning for a string matcher is repeated. For a set of all target patterns, the contents of multiple string matchers are obtained. Since the number of mapped target patterns onto a string matcher is bounded, the proposed pattern partitioning shows the time complexity of $O(T)$, where T denotes the total number of target patterns. Considering the constant time complexity of the existing pattern partitioning approaches that map target patterns sequentially based on a determined order, the proposed pattern partitioning can reduce the number of adopted string matchers by sacrificing the time complexity.

4. Experimental Results

To evaluate the proposed algorithm, a set of total target patterns, which is denoted as *total*, was extracted from

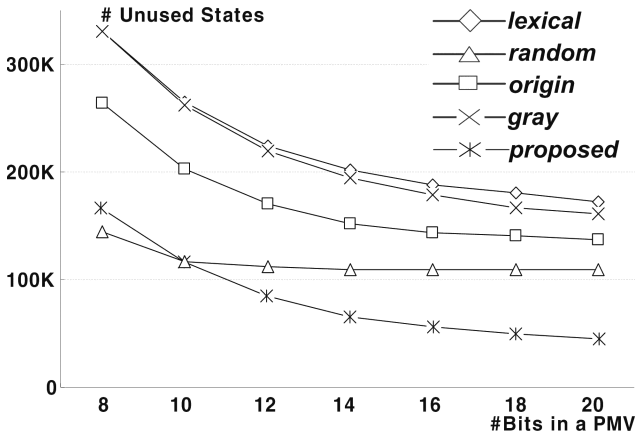

Fig. 4 Resource usage comparisons.

Snort v2.8 rules [5]. The set *total* contained 7784 unique target patterns, where the average length and the maximum length were 18.6 (char/pattern) and 122, respectively. For the apples-to-apples comparisons, several existing pattern partitioning approaches introduced in [3] were implemented; Along with the lexicographical order, two additional cases with random and original orders without sorting were adopted, which were denoted as *lexical*, *random*, and *origin*, respectively. In addition, the string matching scheme using gray-code based sorting in [4], which was denoted as *gray*, was evaluated. Based on the design analysis for the bit-split string matching in [3], each FSM tile took two input bits in our experiments. Figure 3 illustrates the performance comparisons in terms of the number of adopted string matchers. Considering the maximum target pattern length, the number of states in an FSM was set as 128; the number of bits in a PMV was varied from eight by considering the average target pattern length. The number of adopted string matchers was reduced on average by 19.4%, 21.8%, 20.6%, and 18.2%, compared with *lexical*, *random*, *origin*, and *gray*, respectively. In particular, even though the number of bits in a PMV increased, the number of adopted string matchers was saturated; there was a threshold point for minimizing memory requirements in terms of the number of bits in a PMV. Therefore, the proposed pattern partitioning algorithm minimized the memory requirements when the number of bits in a PMV was 12.

In Fig. 4, the resource usage was evaluated in terms of the number of unused states in all FSM tiles. The numbers of the unused states were reduced on average by 66.3%, 56.5%, 33.2%, and 65.3%, compared with *lexical*, *random*, *origin*, and *gray*, respectively. In particular, the numbers of unused states for *lexical* and *gray* were greater than those of other approaches, which means that the number of shared common prefixes in *lexical* and *gray* could be larger than those for *random* and *origin*. However, the decreasing number of adopted string matchers was not linearly proportional to the increasing number of shared common prefixes. This means that the decrease in the number of adopted string matchers of *lexical* and *gray* was limited mainly due to the variety of target pattern lengths and the pattern distributions for each string matcher. The proposed pattern partitioning reduced the number of unused states by increasing the number of mapped target patterns. Therefore, the problem of the variety of target pattern lengths could be mitigated.

5. Conclusions

The proposed memory-efficient pattern partitioning algorithm in this paper decreases the number of adopted string matchers by enhancing the resource usage efficiency. Without additional hardware overhead, the proposed algorithm reduces the number of adopted string matchers by 18.2–21.8%, compared with other existing approaches. Considering the performance enhancements, the proposed algorithm is useful for reducing storage cost without losing the regularity and scalability of the DFA-based parallel string matching engine.

Acknowledgements

This work was supported by the IT R&D program of MKE/IITA. [2009-S-043-01, Development of Scalable Micro Flow Processing Technology]

References

[1] P.-C. Lin, Y.-D. Lin, T.-H. Lee, and Y.-C. Lai, "Using string matching for deep packet inspection," Computer, vol.41, no.4, pp.23–28, 2008.
[2] A.V. Aho and M.J. Corasick, "Efficient string matching: An aid to bibliographic search," Commun. ACM, vol.18, no.6, pp.333–340, 1975.
[3] L. Tan, B. Brotherton, and T. Sherwood, "Bit-split string-matching engines for intrusion detection and prevention," ACM Trans. Archit. and Code Optimization, vol.3, no.1, pp.3–34, March 2006.
[4] H. Kim, H. Hong, H.-S. Kim, and S. Kang, "A memory-efficient parallel string matching for intrusion detection systems," IEEE Commun. Lett., vol.13, no.12, pp.1004–1006, Dec. 2009.
[5] [Online], Available: http://www.snort.org