http://dx.doi.org/10.7840/kics.2014.39B.7.433

CAM과 비트 분리 문자열 매처를 이용한 DPI를 위한 2단의 문자열 매칭 엔진의 개발

김 현 진◆°, 최 강 일*

A Memory-Efficient Two-Stage String Matching Engine Using both Content-Addressable Memory and Bit-split String Matchers for Deep Packet Inspection

HyunJin Kim◆°, Kang-Il Choi*

요 약

본 논문은 DPI (deep packet insepction)를 위한 CAM (content-addressable memory)과 병렬의 비트 분리 (bit-split) 문자열 매처(matcher)를 이용한 2단의 문자열 매칭 엔진의 구조를 제안한다. 긴 타겟 패턴은 같은 길이의 서브 패턴으로 잘라지게 되고, 각 서브패턴은 1단의 CAM에 매핑된다. CAM으로부터의 매칭 인덱스의 시퀀스를 사용하여 2단에서 긴 패턴의 매칭 여부를 알 수 있다. CAM과 비트 분리 문자열 매처를 사용하여 이 기종의 메모리를 사용했을 경우에 메모리 요구량을 크게 줄일 수 있다.

Key Words : Aho-Corasick algorithm, Content-addressable memory, Deep packet inspection, Pattern mapping, String matching engine

ABSTRACT

This paper proposes an architecture of two-stage string matching engine with content-addressable memory(CAM) and parallel bit-split string matchers for deep packet inspection(DPI). Each long signature is divided into subpatterns with the same length, where subpatterns are mapped onto the CAM in the first stage. The long pattern is matched in the second stage using the sequence of the matching indexes from the CAM. By adopting CAM and bit-split string matchers, the memory requirements can be greatly reduced in the heterogeneous string matching environments.

Ⅰ. Introduction

In DPI, multiple signatures are matched in the string matching engine at high speed[1-4]. Due to the variety of signatures, pattern lengths or numbers of characters in patterns are different. In the native CAM-based string matching, because each signature was mapped onto an entry in CAM, the problem of various signature lengths caused memory wastes. In several approaches using ternary CAM (TCAM)[5,6],

※ 본 연구는 미래창조과학부 및 정보통신기술진흥센터의 정보통신·방송 연구개발 사업의 일환으로 수행하였음.[스마트 네트워킹 핵심 기술 개발]
◆° First Author and Corresponding Author : School of EEE, Dankook University, hyunjin2.kim@gmail.com, 정회원
* Smart Node Platform Lab., Electronics and Telecommunications Research Institute, forerunner@etri.re.kr
논문번호 : KICS2014-02-041, Received February 04, 2014; Reviewed July 15, 2014; Accepted July 15, 2014

TCAM requirements were over one mega bytes. In [7], TCAM requirements was proportional to the number of non-trivial state transitions, which was a great burden in minimizing TCAM size. In addition, separate general memory should be adopted to store the next state address for each TCAM index; therefore, long state transition time was required in the previous works[5-7].

In the string matching based on the Aho-Corasick algorithm[8], when all signatures were configured as one deterministic finite automata (DFA), the numbers of states and state transitions became large. The bit-split string matching[9] adopted parallel string matchers with split inputs in finite-state machine (FSM) tiles to reduce the numbers of states and state transitions. However, the problem of various signature lengths was not solved[9]. The string matching architecture in [10] had two-stage DFA-based string matching architecture with parallel bit-split string matchers in both stages. Even though the problem of various signature lengths can be solved somewhat, the memory requirements of the first stage increased. In addition, there were separate short pattern and remnant pattern matchers in [10], which made the implementation of the string matching engine complex.

This paper proposes a two-stage string matching engine using CAM and bit-split string matchers in the first and second stages, respectively. Each signature is divided into subpatterns with the same length. Subpatterns are mapped onto the CAM rows in the first stage, which reduces the required CAM size, compared to the traditional CAM-based string matching. In addition, the memory requirements of the first stage increase proportional only to the subpattern length. In the second stage based on the Aho-Corasick algorithm, long signatures can be recognized using the sequence of match vectors from the first stage. By applying a heuristic algorithm for the signature mapping in the second stage, the proposed architecture reduces the memory requirements of the bit-split string matchers.

Ⅱ. Proposed Architecture

Fig. 1 shows an example of the proposed parallel string matching architecture. Each long signature is divided into multiple subpatterns with the same length and a remnant pattern like [10]. However, in the proposed architecture, by adopting CAM in the first stage, short pattern and remnant pattern matchers are eliminated, which reduces the complexity of implementing the string matching engine. The first stage of the proposed string matching engine detects the matches with subpatterns. For example, a signature "aabbcdd" can be divided into subpatterns "aa," "bb," "cd" with two characters and a remnant pattern 'd.' The character '*x*' or *don't care* is added to the remnant pattern to make the remnant pattern length equal to the fixed subpattern length. Therefore, the signature "aabbcdd" is matched when subpatterns "aa," "bb," "cd," and "d*x*" are matched in order.

When each subpattern is matched in the first stage, the match vector for the subpattern is generated from the first stage. The match vector from the CAM is inputted to the second stage. With the match vectors from the first stage, the bit-split FSMs are constructed in the second stage. When the total number of subpatterns is u, the number of bits of a match vector is $\lceil log_2 u \rceil$.

Each match vector is split into multiple bit inputs for the bit-split string matching. In the example of Fig. 1, two bits are inputted to each memory-based FSM tile. The second stage has parallel

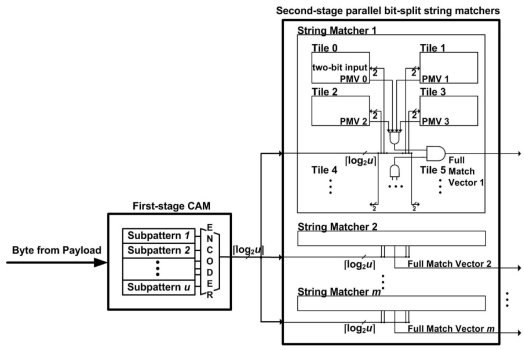

그림 1. 제안된 문자열 매칭 구조의 예
Fig. 1. An example of proposed string matching architecture.

homogeneous string matchers, where the maximum number of signatures to be mapped in a string matcher is limited. Because each signature consists of multiple characters, states should be buffered into registers to handle each character input. For example, when the length of subpatterns mapped onto the first stage is two, signatures “abab” and “ba” are assumed to be searched. The first signature can be matched when characters are inputted as a → b → a → b. In this case, the subpattern “ab” is matched twice in sequence. In addition, the second signature “ba” is matched because “ba” is composed of the last and first characters of a subpattern “ab” for the first signature.

Fig. 2 shows an FSM tile in the second stage. To deal with the signature matching at different time, l registers are inserted to buffer transitions between states in the FSM tile, where l is the length of subpatterns in the first stage. When the output state is reached, the partial match vector (PMV) for the original signature is outputted from each FSM tile. In this case, PMVs are buffered for the signature matching at different time in Fig. 2. With the bitwise AND operation of all PMVs from FSM tiles like [9], the full match vector (FMV) is obtained, where each bit in FMVs shows whether its related signature is matched or not.

Fig. 3 shows an example of state transitions in FSM tiles for the second stage. According to the input, state transitions can happen. In Fig. 3, the target signature is “aabbcdd.” This example assumes that four-bit match vector from the first stage. In this example, the matches with “aa,” “bb,” “cd,” and “dx” can generate the match vectors of “1000,” “1001,” “1010,” and “1011,” respectively. In

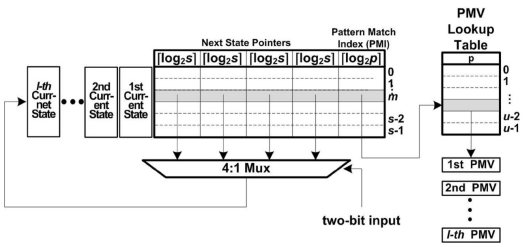

그림 2. 2단의 FSM 타일
Fig. 2. FSM tile in the second stage.

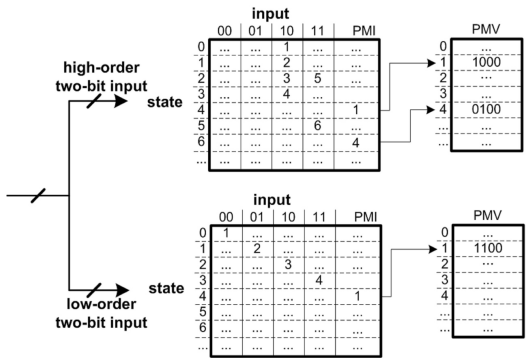

그림 3. FSM 타일의 상태 천이의 예
Fig. 3. An example of state transitions in FSM tiles.

addition, the matches with “aa,” “bb,” “ef,” and “kx” can generate the match vectors of “1000,” “1001,” “1110,” and “1111,” respectively. When match vectors “1000,” “1001,” “1010,” and “1011” are inputted into the second stage in order, the signature “aabbcdd” is matched. Each group of two bits is inputted into each FSM tile. The binary number on the upper side of left tables means input value. Each row indicates the state, where next state pointers are stored according to the input values. According to the high-order two-bit input, the output states for signatures “aabbcdd” and “aabbefk” can be four and six, respectively. The PMIs related to the output states contain pointers towards their PMVs. When the first and second bits in PMVs indicate the matches with “aabbcdd” and “aabbefk,” the values can be true or one in the first and fourth rows in the upper PMV table. On the other hand, the sequences of the low-order two-bit inputs for the two signature matches are the same. Therefore, both the first and second bits in the lower PMV table can be true or one simultaneously. In order to know which signature is matched, the PMV values from the upper and lower PMV tables are bitwise ANDed.

Ⅲ. Signature Mapping onto the Proposed String Matching Engine

In order to map signatures, signatures are divided and unique subpatterns with the same length are

obtained. A sequence of match vectors from the first stage for each long signature can be the input for the signature matching in the second stage. The set of the match vector sequences is partitioned into multiple subsets for homogeneous string matchers in the second stage. In this case, the maximum numbers of matched signatures and states in each FSM tile are considered. For the signature mapping of the second stage, the heuristic algorithm in [11] is applied to reduce the required number of string matchers in the second stage. In this algorithm, after obtaining the initial set with lexicographical sorting for each string matcher, the algorithm searches for other match vectors to be mapped simultaneously onto the string matcher of the second stage greedily. As shown in Fig. 4, the maximum number of signatures in a string matcher is adopted. After constructing DFAs, if the resource limitation is not met, the number of mapped signature in a string matcher decreases. The signature mapping is repeated until there are no unmapped match vectors for the second stage.

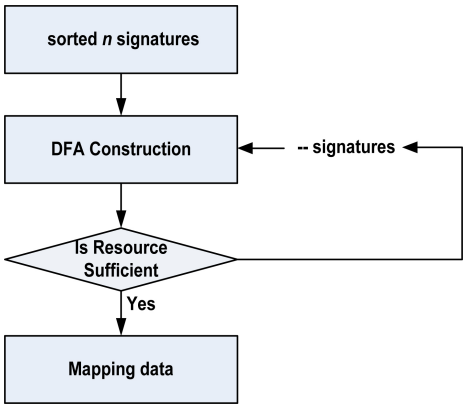

그림 4. 한 문자열 매처의 시그너처 매핑 과정
Fig. 4. Signature mapping in a string matcher.

Ⅳ. Experimental results

4.1 Experimental data

The evaluations were performed using a set that consists of 7784 unique signatures from Snort v2.8[12]. Because this architecture was composed of CAM and memory-based FSMs, the memory requirements of the first and second stages were individually analyzed. For the first stage, evaluations were performed by varying CAM width from 3 to 8. Table 1 summarizes the memory requirements according to the CAM width of the first stage. Because the maximum length in the signatures set was 122, if target signatures were not divided into subpatterns, 950 Kbytes were required to map total rule sets using only CAM. Therefore, by dividing signatures into multiple subpatterns, the required CAM size can be greatly reduced; for example, when CAM width was 4, CAM size was only 62 Kbytes. When CAM width changed from 3 to 4, the number of unique signatures increased greatly; on the other hand, when CAM width changed from 6 to 7, the number of unique signatures decreased slightly. However, CAM size increased with CAM width in all cases. Considering the TCAM sizes in [5-7], which were over one mega bytes, the proposed string matching required small TCAM in the first stage.

For the second stage, the maximum number of states in an FSM tile of the string matcher, S was set as 64 or 128, considering the maximum signature lengths of the Snort rule set. Experiments were processed with varied numbers of bits in a PMV, P from 16, 32, 48, and 64. Considering the previous studies in [9] and [10], the number of input bits in an FSM tile was set as two. In addition, the number of FSM tiles in each string matcher was fixed as 8. In order to show the effectiveness of signature mapping in the second stage, the previous signature mapping approaches with the original order and lexicographical sorting in [9] were evaluated, which were denoted as *origin* and *lexical*, respectively. In [10], each input bit for an FSM tile input was selected from both the most significant bits (MSBs)

표 1. 1단에서의 CAM 너비에 따른 실험 요약
Table 1. Experimental summary according to CAM width in the first stage.

stage	width	3	4	5	6	7	8
first	#subpatterns	12.5K	15.6K	17.7K	17.1K	16.9K	16.0K
	#CAM(Kbyts)	37.8	62.4	88.4	102.3	118.3	128.2
second	#matchers	611	427	365	302	297	259
	#Mem(Kbytes)	1760	1230	1051	870	855	746

and least significant bits (LSBs) of the byte input alternately. This input bit grouping method was applied to the signature mapping approaches with the original order and lexicographical sorting, which were denoted as $origin_g$ and $lexical_g$, respectively.

In the experiments, the memory requirements for the second stage can be minimized when S and P were 64 and 32, respectively. In Table 1, the required numbers of string matchers and its memory requirements are shown by sweeping CAM width when S and P are 64 and 32. By increasing the CAM width, the memory requirements of the second stage were reduced, which means that the number of unique subpatterns decreased with CAM width. In this case, when CAM width was 8, the number of required string matchers and the memory requirements were 259 and 746 Kbytes.

4.2 Comparisons and discussion

Table 2 shows memory requirements in terms of the required number of string matchers in the second stage. As the CAM width increased, the number of match vectors decreased; therefore, the reduced number of string matchers can decrease. In addition, the ratio of reduced memory requirements decreased. This can be mainly due to the decrease of the signature lengths in the second stage. The required numbers of string matchers in the second stage were reduced by 27.9%-15.1%, 26.0%-12.3%, 19.0%-10.8%, and 10.5%-2.7%, compared to the cases of *origin*, $origin_g$, *lexical*, and $lexical_g$, respectively. Therefore, it was concluded that the number of string matchers in the second stage was decreased by applying the heuristic algorithm in [11] without additional hardware. In order to know the performance of the proposed string matching engine, when CAM width was 8, the main blocks were implemented using an Xilinx field programmable gate array (FPGA)[13]. The maximum clock speed of the second stage was 370 MHz. When multiple CAM blocks with 2048 entries were used, the maximum clock speed in the first stage was 81 MHz. In this case, the throughput can be 648 Mbps. On the other hand, when the number of entries was 128 in each CAM block, the maximum clock speed increased up to 102 MHz. Therefore, as the number of TCAM entries increased, it was concluded that the maximum clock speed decreased in the FPGA implementation. In addition, considering the speed of TCAM slower than that of the general memory, the speed was limited in the first stage when the proposed string matching engine was implemented using FPGA.

On the other hand, the stand-alone commercial TCAM in [14] was considered to analyze the performance of the proposed string matching engine. In [14], the maximum clock speed was 360 MHz. Considering the maximum clock speed in the second stage implemented in the FPGA above, the speed of the second stage was sufficient to meet the performance of the first stage. According to the clock speed of TCAM, the maximum throughput can be 2.88 Gbps. Compared to the multi-cycle loops (2 or 3 clock cycles) in the previous TCAM-based string matching methods[5-7], the proposed string matching engine generated the match vector at each clock cycle in the first stage. In addition, the second stage obtained FMVs at each cycle in a pipelined manner. Therefore, the performance of the proposed string matching engine can be enhanced twice or three times. In Table 3, the pros and cons are summarized by comparing the proposed and

표 2. 2단에서의 요구되는 문자열 매처의 개수
Table 2. Memory requirements in terms of required number of string matchers in the second stage.

#CAM width	3	4	5	6	7	8
origin	779	541	467	362	360	298
origing	770	525	458	355	355	291
lexical	727	504	420	338	343	287
lexicalg	675	454	376	316	313	266
proposed	611	427	365	302	297	259

표 3. 대표 방법의 장단점 비교
Table 3. Summary of pros and cons.

method	pros	cons
TCAM	simple	high power&price
bit-split	general SRAM, high performance	many small separate memory blocks
proposed	small TCAM size, high performance (~x3)	both SRAM and TCAM

previous works.

Ⅴ. Conclusion

The proposed string matching adopts small CAM and bit-split string matchers in two stages, so that the problem of various signature lengths can be solved with reduced memory requirements and complexity. Considering the experimental results above, the proposed two-stage string matching can reduce CAM size in the first stage and the required number of string matchers in second stage. Therefore, it is expected that the proposed two-stage string matching architecture is useful for reducing the storage cost in the DPI.

References

[1] P.-C. Lin, Y.-D. Lin, T.-H. Lee, and Y.-C. Lai, "Using string matching for deep packet inspection," *IEEE Computer*, vol. 41, no. 4, pp. 23-28, 2008.

[2] K. Kim, S. Kang, I. Song, and T. Kwon, "TCAM partitioning for high - performance packet classification," *J. KICS*, vol. 31, No. 2B, pp. 91-97, 2006.

[3] T. AbuHmed, A. Mohaisen, and D. H. Nyang, "A survey on deep packet inspection for intrusion detection systems," *Mag. Korea Telecommun. Soc.*, vol. 24, No. 11, pp. 25-36, 2007

[4] Y.-C. Yoon and S.-Y. Hwang, "Design and implementation of high-speed pattern matcher in network intrusion detection system," *J. KICS*, vol. 33, no. 11B, pp. 1020-1029, 2008.

[5] F. Yu, R. H. Katz, and T. V. Lakshman, "Gigabit rate packet pattern-matching using TCAM," in *Proc. Int. Conf. Network Protocols (ICNP 2004)*, pp. 174-183, Oct. 2004.

[6] J.-S. Sung S.-M. Kang, Y. Lee, and T.-G. Kwon, "A multi-gigabit rate deep packet inspection algorithm using TCAM," in *Proc. IEEE GLOBECOM*, pp. 453-457, 2004.

[7] S. Yun, "An efficient TCAM-based implementation of multipattern matching using covered state encoding," *IEEE Trans. Computers*, vol. 61, no. 2, pp. 213-221, Feb. 2012.

[8] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Commun. ACM*, vol. 18, issue 6, pp. 652-654, 1975.

[9] 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, 2006.

[10] H. Kim, H.-S. Hong, and S. Kang, "A memory-efficient bit-split parallel string matching using pattern dividing for intrusion detection systems," *IEEE Trans. Parallel and Distributed Syst.*, vol. 22, no. 11, pp. 1004-1006, 2011.

[11] H. Kim, H. Hong, D. Baek, and S. Kang, "A pattern partitioning algorithm for memory-efficient parallel string matching in deep packet inspection," *IEICE Trans. Commun.*, vol. E93-B, no. 6, pp. 1612-1614, 2010.

[12] Snort, "Intrusion detection system," http://www.snort.org.

[13] Xilinx: Virtex-4 VLX FPGA, http://www.xilinx.com.

[14] Renesas: TCAM, http://www.renesas.com/ products/memory/TCAM/index.jsp.

김 현 진 (HyunJin Kim)

1997년 2월 : 연세대학교 전기공학과 졸업

2010년 2월 : 연세대학교 전기전자공학과 박사

2011년 9월~현재 : 단국대학교 전자전기공학과 조교수

<관심분야> 디지털 회로 설계 및 알고리즘 구현, 문자열 매칭, 가상화 시스템

최 강 일 (Kang-Il Choi)

1992년 : KAIST 전자계산학과 학사

1994년 : 서강대학교 전자계산학과 석사

2011년 9월~현재 : 한국전자통신연구원 스마트노드플랫폼 연구실 선임연구원

<관심분야> 패킷전달망 기술, 스마트 네트워트 기술, 네트워크 가상화 기술