Split-TCAM mechanism for storing patterns efficiently

Jae-Woo Sim, Chan-Ho Park, Chi-Hoon Shin, Young Woo Kim and Seongwoon Kim

1 Dept. of Computer Software & Engineering, University of Science and Technology (UST), 113 Gwahangno, Yuseong-gu, Daejeon, Korea
2 Server Platform Research Team, Electronics and Telecommunications Research Institute (ETRI), 161 Gajeong-dong, Yuseong-gu, Daejeon, Korea
a) plipper@ust.ac.kr

Abstract: TCAM-based payload inspection algorithms for detecting viruses and worms wasted much memory for storing redundant bytes, “don’t care”. The aim of our research was to eliminate the redundancy so as to store more rules in TCAM or reduce the area cost. To eliminate the redundancy, we devised Split-TCAM mechanism where a TCAM block is split into every byte. We simulated our Split-TCAM mechanism using Snort v2.8.5 [1]. Results indicated that the proposed mechanism could save the required size of memory to 37.4% and 27.7% on average when applying it to the R-TCAM and Jumping Window algorithm, respectively.

Keywords: TCAM, IDS, string matching, deep packet inspection

Classification: Science and engineering for electronics

References


1 Introduction

Network threats such as worms and viruses have been increasing rapidly and degrading the reliability of the Internet. To detect the malicious activities on...
network, the deep packet inspection (DPI), which tries to find out suspicious patterns in the payload field of packets, should be performed in the intrusion detection system (IDS).

The DPI mainly affects the performance of the IDS due to a large number and various lengths of rules to be detected. Thus, flexible and high performance algorithm for the DPI is needed not to degrade the line-speed while inspecting the packets. Recently, hardware based approaches [2, 3, 4, 5] using ternary content addressable memory (TCAM) have been proposed to guarantee real time inspection on multi-gigabit Ethernet.

Initially, F. Yu et al. proposed Sliding TCAM [2] algorithm which looks up the TCAM whenever the window is shifted by one byte at a time. To improve the inspection speed, algorithms such as Rotating TCAM (R-TCAM) [3] and Jumping Window [4, 5] have attempted to shift the window multi-bytes at a time. However, since the approaches must consider the every positions where patterns can appear, they waste significant amount of memory to make room for shifted patterns that contain redundant bytes, “don’t care,” when they store rules in a TCAM. Moreover, as a window size becomes wider, the degree of redundancy in the memory sharply increases.

In this paper, we propose “Split-TCAM” mechanism to eliminate the redundancy. The Split-TCAM is modified TCAM split into every byte for each pattern’s properties and sizes. By storing shifted-pattern of prefix and suffix except for “don’t care” bytes in each corresponding TCAM, the memory usage could be significantly reduced.

The rest of this paper is organized as follows. In section 2, we explain proposed Split-TCAM mechanism for storing patterns efficiently. In section 3, we present simulation results that show how the memory efficiency goes up. Finally, we conclude our work.

2 Split-TCAM mechanism

2.1 Architecture

TCAM-based pattern matching algorithms, R-TCAM and Jumping Window, includes redundant “don’t care” bytes when they store patterns. To avoid this redundancy in the memory, we proposed Split-TCAM mechanism and its architecture which stores the shifted-patterns and suffix in each corresponding TCAM, such as split TCAM for shifted-prefix (STP), split TCAM for suffix (STS) and split TCAM for prefix, infix, and suffix (STPIS) excluding “don’t care” bytes, as shown in Fig. 1 (a). We define that prefix is first \( w \) byte(s) of a pattern, infix is next \( m \times w \) byte(s), and suffix is the rest of the prefix and the infix, where \( m = \lceil \text{total length of the pattern} / w - 1 \rceil \).

Figure 1 (a) shows the architecture of Split-TCAM where, window size \( w \) is four bytes. The \( \text{STP-n} \) is TCAM whose width is \( n \) byte(s) for each shifted-patterns of prefix that is shifted by \( (w - n) \) byte(s), where \( n \) is shifted value of prefix, \( 1 \leq n \leq w - 1 \). The \( \text{STS-k} \) is TCAM whose width is \( k \) byte(s) for suffix, \( 1 \leq k \leq w - 1 \). And the \( \text{STPIS} \) is TCAM whose width is four bytes for prefix, infix, and suffix. The \( \text{Distributor} \) distributes the input string to
each corresponding TCAM and the Selector selects one of matching results on the basis of priority and outputs the result.

Detailed mechanism about store of patterns in TCAM and inspection of input data will be presented in the next section.

**Fig. 1.** (a) Split-TCAM architecture, where $w$ is 4 bytes. (b) When storing a pattern, “abcdef” by using R-TCAM algorithm. (c) When storing a pattern, “abcdef” by using Jumping Window algorithm. (d) Distribution and selection mechanism of Split-TCAM.
2.2 Storing patterns in TCAM

Algorithms, such as R-TCAM and Jumping Window store a lot of “don’t care” bytes in conventional TCAM. Assume that we store a pattern, “abcdef” in conventional TCAM using R-TCAM algorithm. When \( w \) is 4 bytes, the prefix of the pattern “abcd,” shifted-patterns of prefixes “?abc,” ??ab,” and “???a,” and a suffix of the pattern “cdef” are stored in the TCAM. In all, 20 bytes of the TCAM are needed to store one pattern whose size is 6 bytes. And 8 bytes of the 20 bytes are wasted due to the redundant bytes and repeated pattern “cd”.

On the other hand, as depicted in Fig. 1 (b), if we apply our Split-TCAM mechanism to R-TCAM, only 12 bytes are required to store the pattern, because “abcd,” “abc,” “ab,” “a,” and “ef,” where redundant bytes are eliminated, are stored in Split-TCAM. Each of the patterns is stored in the corresponding Split-TCAM. The pattern of “abcd” is stored in the STPIS, the “abc” in the STP-3 and so on.

Figure 1 (c) presents the case when a pattern, “abcdef” is stored in Split-TCAM by using Jumping Window algorithm. The pattern is stored in the same manner as the R-TCAM algorithm.

2.3 Inspection of input data

Before inspecting input strings, each patterns split for the string should be distributed to corresponding TCAM. Thus, the Distributor distributes the input strings to each pattern entries of Split-TCAM, such as the STS, the STP and the STPIS, for every \( w \) byte(s). For example, if \( w \) is 4 and the input string is “abcde. . .”, the Distributor distributes first 4 bytes, “abcd” to each pattern entry of Split-TCAM, i.e. “d,” “cd” and “bcd” to the STP, “a,” “ab” and “abc” to the STS, and “abcd” to the STPIS in Fig. 1 (d).

Match result from each TCAM is delivered to the Selector. The Selector selects a matching result on the basis of priority in order to deal with multi-match results. As shown in Fig. 1 (d), the STPIS has the highest priority and the STP-1 has the lowest priority. Since the priority policy is the same as previous one which is used in R-TCAM and Jumping Window, it leads to same result.

2.4 Additional benefit

By applying the Split-TCAM mechanism to TCAM-based algorithms, we also could solve rearranging problem that is occurred when inserting new pattern in conventional TCAM. Since conventional TCAM returns the highest matched row, i.e. the higher priority a pattern has, the upper row of TCAM the pattern should be stored in. Thus, as presented in Fig. 1 (e), the patterns with low priority such as “?abc” and “?efg” should be rearranged when inserting new pattern “ijkl,” which has higher priority, in conventional TCAM. However, in the Split-TCAM, it is not needed to rearrange the patterns because the patterns which have the same priority are stored in each corresponding TCAM which has same size (see Fig. 1 (e)).
3 Experimental results

We verified how much size of TCAM can be reduced compared to previous mechanism by simulation. We implemented a software version of the R-TCAM and Jumping Window in C language and stored patterns in conventional TCAM and Split-TCAM as shown in Fig. 2 (a). Snort v2.8.5 [1], which has 4,330 rules consisting of 6,246 patterns, was used for simulation.

Fig. 2. (a) The Simulation environment. (b) The distribution of patterns. (c) Window size vs. TCAM size.
Figure 2 (b) shows the distribution of the patterns.

Figure 2 (c) depicts the required memory size of Split-TCAM and conventional TCAM for R-TCAM and Jumping Window algorithm according to the window size. The results showed that the proposed Split-TCAM mechanism significantly reduces the required memory size for both algorithms. Although the required memory size is increased sharply as the window size grows wider in both cases of Split-TCAM and conventional TCAM, the amount of increment in memory size according to window size was relatively small in the case of Split-TCAM compared to conventional TCAM.

Table I indicated that the proposed mechanism could save the required size of memory to 37.4% and 27.7% on average respectively when applying it to the R-TCAM and Jumping Window algorithm. The mechanism also could save approximately 50% of memory when the window size is greater than 8 bytes for R-TCAM algorithm and 32 bytes for Jumping Window algorithm.

### Table I. The saving ratio of memory.

<table>
<thead>
<tr>
<th>Window Size</th>
<th>R-TCAM</th>
<th>Jumping Window</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Conventional TCAM (kB)</td>
<td>Split-TCAM (kB)</td>
</tr>
<tr>
<td>2</td>
<td>74</td>
<td>81</td>
</tr>
<tr>
<td>4</td>
<td>99</td>
<td>79</td>
</tr>
<tr>
<td>8</td>
<td>268</td>
<td>134</td>
</tr>
<tr>
<td>16</td>
<td>1009</td>
<td>460</td>
</tr>
<tr>
<td>32</td>
<td>4037</td>
<td>1924</td>
</tr>
<tr>
<td>64</td>
<td>15004</td>
<td>8033</td>
</tr>
<tr>
<td>128</td>
<td>62014</td>
<td>32575</td>
</tr>
<tr>
<td>Avg.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### 4 Conclusions

We suggested the Split-TCAM mechanism to avoid the redundancy in TCAM-based payload inspection algorithms such as R-TCAM, Jumping Window. By storing patterns without “don’t care” bytes in shifted patterns of prefix, we reduced the waste of memory. We applied our Split-TCAM mechanism to both R-TCAM and Jumping Window algorithm using Snort v2.8.5.

Results indicated that the proposed mechanism could save the required size of memory to 37.4% and 27.7% on average when applying it to the R-TCAM and Jumping Window algorithm, respectively. The mechanism also could save approximately 50% of memory when the window size is greater than 8 bytes for R-TCAM algorithm and 32 bytes for Jumping Window algorithm.

Additionally, we could solve rearranging problem which is occurred when inserting new pattern in conventional TCAM so as to update patterns fast.