A Soft-Error Tolerant TCAM for Multiple-Bit Flips Using Partial Don't-Care Keys

Infall Syafalni
Logic Research Co., Ltd.
infall@ieee.org

Tsutomu Sasao
Meiji University
sasao@cs.meiji.ac.jp

Xiaoqing Wen
Kyushu Institute of Technology
wen@cse.kyutech.ac.jp

***Abstract*—Ternary content addressable memories (TCAMs) are special memories which are used in high-speed network applications such as routers, firewalls, and network address translators. In high-reliability network applications such as aerospace and defense systems, soft-error tolerant TCAMs are indispensable to prevent data corruption or faults caused by radiation. This paper proposes a novel soft-error tolerant TCAM for multiple-bit-flip errors using partial don't-care keys (X-keys), called k-TX. k-TX corrects up to k-bit flip errors and significantly enhances the tolerance of the TCAM against soft errors, where k is the maximum number of bit flips in a TCAM word. k-TX consists of a TCAM, a preprocessed don't-care-bit index look-up memory (X look-up), and an ECC-SRAM. First, k-TX randomly selects a search key. After that, k-TX detects multiple-bit-flip errors by the generated X-keys using X look-up. If the keys match the different locations, then a soft error is detected and k-TX refreshes the TCAM words by using a backup ECC-SRAM. Experimental results demonstrate the advantages of k-TX. Moreover, the hardware overhead of k-TX is small due to the use of only a single TCAM. k-TX can be easily implemented and is useful for fault-tolerant packet classifiers.**

I. Introduction

A ternary content addressable memory (TCAM) is a special memory with three values *i.e.,* 0, 1, and $*$ (*don't-care*). It simultaneously compares the input vector with the entire list of registered vectors [1]. TCAM is a *de facto* standard in routers and devices for packet classification in high-speed network applications [2]. Fig. 1 shows a TCAM cell. The search bits (SL_1, SL_0) are compared with the stored bits (D_1, D_0). When there is a match, the match line (ML) sends a signal to the priority encoder to produce the match address.

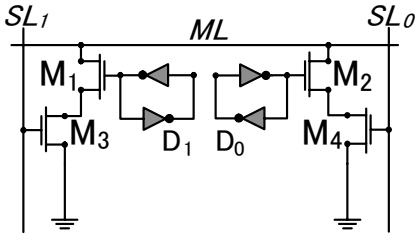


Fig. 1: NOR-type ternary cell [1]

TABLE I: TCAM Encoding

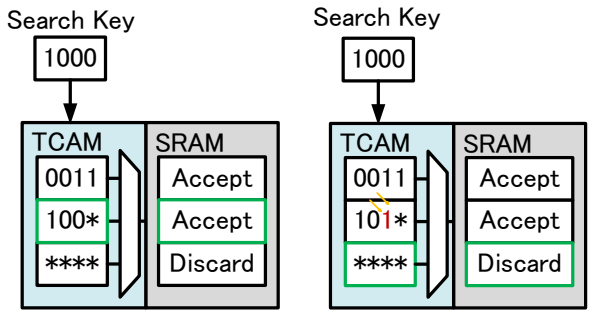
Value	D_1	D_0	SL_1	SL_0
0	0	1	0	1
1	1	0	1	0
*	1	1	0	0

A soft error is typically caused by radioactive atoms, alpha particles, cosmic rays or high-energy neutrons [3]. Several works investigated the effects of scaling down the size of transistors [4], [5]. A work in [5] shows that moving from the 130 nm process to the 22 nm process increases soft error rates up to 7 times. Furthermore, soft errors tend to cause more serious problems in low-power devices [3].

One of the troubling effects of soft errors in memories is that they hit memory cells and may change the values of some cells. Changes of the values of TCAM cells may lead to errors or data corruption. A soft error will not damage hardware; it only changes the data that is being processed, and it can result in faulty data [6]. TCAMs are more vulnerable to soft errors than RAMs since TCAMs are more complicated than RAMs. The bit storage of TCAMs uses SRAM cells which are susceptible to soft errors [7]. Furthermore, high-speed memories with smaller transistor sizes will make it more vulnerable to soft errors [8]. Thus, in applications that require high-reliability such as finance, aerospace, and defense networks, soft-error tolerant TCAMs are indispensable. Unfortunately, in TCAM, conventional ECC (Error Checking and Correction) techniques used in RAM are difficult to apply [9].

In previous works, hardware and software methods were developed to mitigate soft errors [6], [10]–[13]. In [12], modification of hardware by using XOR-based conditional keepers are used to overcome noises including soft errors. In [13], a system using bloom filters detects errors in TCAMs. Furthermore, in [6], a parallel system using two TCAMs detects and corrects TCAM words that are attacked by soft errors. However, previous hardware and software methods still suffer from severe drawbacks. Firstly, hardware methods are very costly to implement since they modify the circuits of TCAMs. Secondly, for software methods, researchers are still looking for more efficient way to tackle the soft-error problem in TCAMs. In [11], a TCAM with an optimized scrubbing interval is proposed against soft errors. However, this scheme does not consider that some keys are more popular (frequent) than others in the TCAM which can lead to more faults when the frequent keys hit the upset word caused by soft error. In [6], a TCAM checker is proposed that takes into consideration frequent keys through comparing the matched words caused by a soft error. However, this scheme uses two TCAMs, which doubles the hardware overhead and the power dissipation.

Table I shows the TCAM encoding. The stored bits are



(a) Correct match (b) False match

Fig. 2: A soft error in a TCAM

represented by D_1 and D_0. Note that the values of D_1 and D_0 are not necessarily complementary. The don't-care value $(*)$ is represented by $(D_1, D_0) = (1, 1)$. While the stored bits represent the value of the TCAM cell, the search lines represent a search key bit. Interestingly, when the search lines represent the don't-care value $(*)$, the search lines value are $(SL_1, SL_0) = (0, 0)$. If we refer to the transistor-level representation of a TCAM cell in Fig. 1, then this condition allows transistors M_3 and M_4 to be off, forcing a match of the bits regardless of the stored bits D_1 and D_0. This observation inspires us to propose our method in this paper.

Fig. 2 shows an example of a soft error. A **false match** is a match that would be a mismatch if no soft error occurred. In term of packet classification, a false match is referred to as a **misclassification**. Fig. 2(b) shows a misclassification which matches the third word of the TCAM, but should match the second word of the TCAM (Fig. 2(a)). The soft error occurs at the third bit of the second word. If we change the search key to 10*0, then the TCAM will match the correct word (*i.e.,* second word). We call this search key as a **partial don't-care key (X-key)** which will be explained later.

The **major contributions** of this paper are as follows: 1) A novel scheme, called k-TX, is proposed that can detect and correct multiple-bit-flip soft errors in a TCAM. 2) The k-TX requires no modifications to the TCAM. 3) The k-TX uses only one TCAM. 4) The soft-error tolerance of k-TX outperforms existing schemes.

The rest of the paper is organized as follows: Section II shows definitions and basic properties; Section III-A explains a soft-error tolerant TCAM for single-bit-flip errors using partial don't-care keys, TX; Section III-B describes the proposed scheme for multiple-bit-flip errors, k-TX; Section IV shows experimental results; and Section V concludes the paper.

II. Definitions and Basic Properties

A TCAM word, that represents a packet classifier rule, consists of several fields. In addition, each field is represented by a prefix sum-of-products.

A. Prefix Sum-of-Products

Definition 2.1. *$x_i^{a_i}$ denotes x_i when $a_i = 1$, and $\bar{x}_i$ when $a_i = 0$. x_i and $\bar{x}_i$ are **literals** of a variable x_i. The AND of literals is a **product**. The OR of products is a **sum-of-products expression** (SOP).*

Definition 2.2. *A **prefix SOP** (PreSOP) is an SOP consisting of products having the form $x^*_{n-1}x^*_{n-2}\cdots x^*_{m+1}x^*_m$, where x^*_i is x_i or $\bar{x}_i$ and $m \leq n-1$. [14]*

Example 2.1. *$f(x_1, x_0) = \bar{x}_1 x_0 \vee x_1$ is a PreSOP. And, $f(x_1, x_0) = x_0 \vee x_1$ is an SOP, for the same function, but is not a PreSOP.* ■

In high-speed network applications, PreSOPs are used instead of SOPs. This is because PreSOPs can be quickly generated from the binary decision trees of the functions.

B. Classification Functions

A classification function is defined as a mapping of fields specified by a set of rules. Each rule is a conjunction of fields that can be represented by PreSOPs.

Definition 2.3. *A **classification function** with k **fields** is a mapping $F : P_1 \times P_2 \times \cdots \times P_s \rightarrow \{0, 1, 2, \cdots, r\}$, where $P_i = \{0, 1, \cdots, 2^{t_i} - 1\}$ $(i = 1, 2, \cdots, s)$. F is specified by a set of r rules. A **rule** consists of s fields, and each field is specified by an interval of t_i bits.*

In the Internet, a packet classifier is specified by source and destination addresses, source and destination ports, and protocol types. The source and the destination ports are represented by intervals. Some works on the representation of classification functions can be found in [15], [16].

C. Soft Errors in TCAM Cells

Definition 2.4. *A **soft error** in a TCAM cell is a non-permanent error that changes cell values and can cause misclassification.*

Definition 2.5. *A **single-bit-flip** error or **multiple-bit-flip** errors are a change of a cell or several cells values of a TCAM word caused by a soft error, where the **fault model** is $0 \rightarrow \{1, *\}$, $1 \rightarrow \{0, *\}$, or $* \rightarrow \{0, 1\}$. The probability of bit-flip errors is $P_e = \frac{u}{w}$, where u is the number of bit-flip words and w is the number of words in a TCAM.*

III. TX: A TCAM using Partial Don't-Care Keys

This section describes our proposed method, TX: a TCAM using partial don't-care keys. T stands for TCAM and X stands for partial don't-care keys (X-keys).

A. TX: TCAM for Single-Bit-Flip Errors

1) Overview of TX: The TX is a scheme that enhances the tolerance of a TCAM against soft errors.The TX consists of a TCAM, an ECC-SRAM as a back-up, a multiplexer and a controller as shown in Fig. 3. In TX, no modification to the TCAM is required. The TX has two modes: normal mode and test mode. In normal mode, the TCAM looks up a search key in parallel. In test mode, the TX generates up to 2^d partial don't-care keys (X-keys) and detects soft errors, where d is the tolerance degree. It works sequentially as follows: First, the TX starts from the tolerance degree 1 and if any soft error

is detected, it will increase the degree until the desired degree. Don't-care bits are inserted in the X-keys and they are expected to match the words that contain soft errors. The TX applies X-keys to the TCAM and records all the returned indices. After that, if the indices are equal, then no soft error is detected in the operation; otherwise a soft error is detected.

Whenever a soft error is detected, the iteration count is checked. If the iteration is less than the desired degree, the TX will access refresh bits of TCAM words. If the value is 0, then the word has not refreshed yet and the TX will refresh the word with the current index by the back-up ECC-SRAM. We use this refresh bit to avoid refreshing a word more than once. This will significantly save power, since the refresh operation dissipates high power. If the word is refreshed, then we increase the degree and generate X-keys with a fewer inserted don't-cares. And, the operation is stopped when the indices are equal or the iteration count reaches the desired degree value. Fig. 5 illustrates the method to generate X-keys.

2) Tolerance Degree (d):

Definition 3.1. *Tolerance degree d is a parameter that controls the tolerance of the TCAM. The higher the degree, the more tolerant the TCAM is. The maximum degree is $\lceil \log_2 n \rceil$, where n is the number of bits in a TCAM word.*

The maximum tolerance degree shows the highest level of TCAM tolerance enhancement by the TX.

3) Partial Don't-Care Keys (X-keys):

Definition 3.2. *A partial don't-care key (X-key) is a search key with inserted don't-care bits. The key is used to find a soft error in TCAM words.*

Lemma 3.1. *Assume that don't-care bits are inserted consecutively in TX. Then, the number of don't-cares inserted into an X-key is at most $q = \lceil n/2^d \rceil$, where d is the tolerance degree and n is the number of bits in a word.*

The number of don't-cares inserted in an X-key depends on the required tolerance degree. The smallest number of don't-cares that can be inserted in X-keys is one.

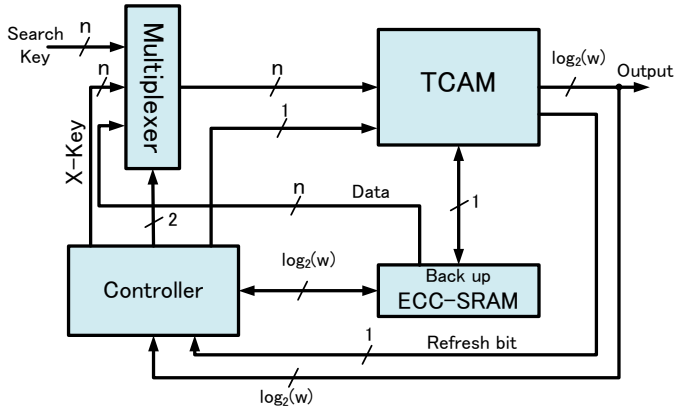


Fig. 3: TX: A soft-error tolerant TCAM using partial X-keys

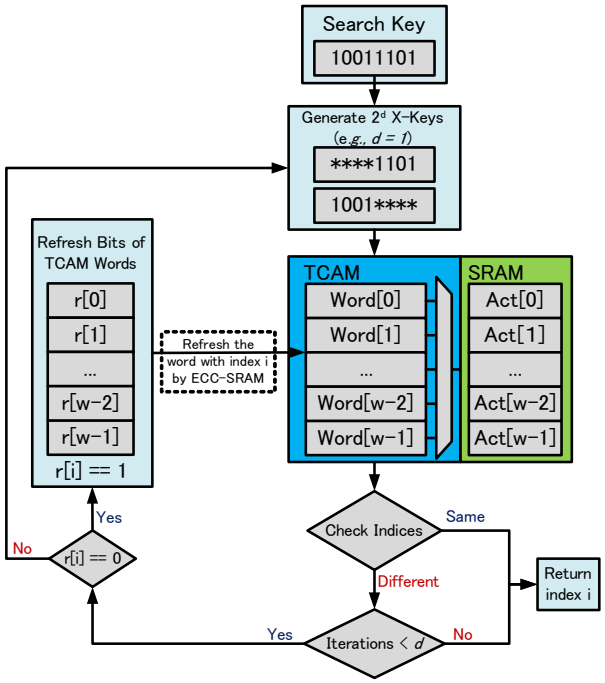


Fig. 4: Generation of partial don't-care keys for single-bit-flip errors

Lemma 3.2. *The number of partial don't-care keys (X-keys) is*

$$p = 2^d - \lfloor \frac{2^d q - n}{q} \rfloor,$$

where d is the tolerance degree, n is the number of bits in a word, and $q = \lceil n/2^d \rceil$.

Proof: If n is a multiple of 2^d, then the number of X-keys is $p = 2^d$. Otherwise, bit difference is $2^d q - n$, where q is the number of don't-cares in the X-keys. Since the number of bits in a word is n, we exclude the X-keys that cover the bits with index more than n. □

Algorithm 1 shows the procedure for generating X-keys. The number of don't-care bits in X-keys is q (Line 1). First, the search key is copied to the X-key (Line 5). We start from index $i = 0$. If $n - i$ is greater than q, then q consecutive don't-care bits are inserted in the X-key (Line 6 and 7). Otherwise, only the remaining $n - i$ consecutive don't-care bits are inserted in the X-key (Line 10).

4) Refresh Bits of TCAM Words:

Definition 3.3. *A refresh bit of a TCAM word has a value 1, when the TCAM word has been refreshed by the ECC-SRAM. Otherwise, it has a value 0. Every word in the TCAM has a refresh bit to indicate the status of refreshing of the word. The total number of refresh bits of TCAM words is w.*

Algorithm 2 shows the method for checking indices. If all the indices are equal, it will return 0, which shows the matched index is correct. Otherwise, it will return 1, which means a soft error is detected.

Algorithm 1 Generation of X-keys (*XKeyGen*())

```
/* Input: Search Key (searchKey) and degree (d). */
1: q ⇐ ⌈n/2^d⌉
2: i ⇐ 0
3: numKey ⇐ 0
4: while i < n do
5:    XKeys[numKey] ⇐ searchKey
6:    if n − i ≥ q then
7:       XKeys[numKey][i..i + q] ⇐ {∗}
8:       i ⇐ i + q
9:    else
10:      XKeys[numKey][i..n] ⇐ {∗}
11:      i ⇐ n
12:   end if
13:   numKey + +
14: end while
15: Return XKeys
```

5) Algorithm and Time Complexity of TX: A merit of using a TCAM is matching the search key simultaneously. However, in test mode, the TX uses a sequential TCAM look-up. Since the operation time of TX is significantly impacted by the TCAM look-up time, we briefly describe it.

Algorithm 2 Check of indices (*CheckIdx*())

```
/* Input: Indices (Idx) and number of keys (numKey). */
1: i ⇐ 0
2: while (Idx[i] == Idx[i + 1]) ∧ (i < numKey − 1) do
3:    i + +
4: end while
5: if i == numKey − 1 then
6:    Return 0 /* Correct match */
7: else
8:    Return 1 /* An error is detected */
9: end if
```

Lemma 3.3. *A sequential TCAM look-up runs in $\mathcal{O}(wn)$ time, where n is the number of bits in a word and w is the number of words in the TCAM.*

Proof: To check the TCAM sequentially, it requires n steps to check for each word and there are w words. □

Algorithm 5 shows the details of the TX scheme. TX uses a TCAM with wn bits and an ECC-SRAM for refresh operations. P_c shows the probability of the TX being used for the given search keys. If the randomized *Prob* is smaller than P_c, the TX runs; otherwise, a normal TCAM look-up runs. First, the indices are assumed to be different (*difIdx* $= 1$, Line 4). Next, it starts to generate X-keys with degree 1 (*iterations* $+ 1 = 1$, Line 7). Then, the X-keys are applied sequentially to the TCAM (Lines 8 to 10). If the indices are different (Line 11), then it checks the refresh bit of the TCAM word (r[*Idx*[i]]). If the refresh bit is 0, then the TX refreshes the TCAM word with index *Idx*[i] by that of the equal index of the ECC-SRAM and does another TCAM look-up by the corresponding X-key. After that, the TX

Algorithm 3 TX: Testing TCAM using partial don't-care keys for single-bit-flip errors

```
/* Input: A TCAM with w words and n bits each word, a
   search key (searchKey), an ECC-SRAM as a back-up of
   the TCAM for refresh operation, and the degree d. */
1: Prob ⇐ rand()
2: if P_c ≥ Prob then
3:    r ⇐ {0}
4:    difIdx ⇐ 1
5:    iterations ⇐ 0
6:    while (difIdx) ∧ (iterations < d) do
7:       XKeys ⇐ XKeyGen(searchKey, iterations + 1)
8:       for i = (0, · · · , numKey − 1) do
9:          Idx[i] ⇐ TCAM(XKeys[i])
10:      end for
11:      difIdx ⇐ CheckIdx(Idx, numKey)
12:      if difIdx then
13:         for i = (0, · · · , numKey − 1) do
14:            if r[Idx[i]] == 0 then
15:               Refresh the TCAM word with index Idx[i]
16:               r[Idx[i]] ⇐ 1
17:               Idx[i] ⇐ TCAM(XKeys[i])
18:            end if
19:         end for
20:         difIdx ⇐ CheckIdx(Idx, numKey)
21:      end if
22:      iterations + +
23:   end while
24: end if
25: Return TCAM(searchKey)
```

checks the indices again. If the indices are equal (*difIdx* $= 0$), then it moves out the while loop. Otherwise, the TX increases the degree and iterates the same routine.

Lemma 3.4. *The number of partial don't-care keys (X-keys) generated by the TX is at most $2^{d+1} - 2$.*

Proof: The TX iterates d times. It starts detecting and correcting the errors from degree 1 to d. Thus, by Lemma 3.2, the total number of X-keys generated by TX is at most $\sum_{i=1}^{d} 2^i$. To obtain an upper bound, we omit the right term of the equation in Lemma 3.2. Thus, the lemma is proved. □

Theorem 3.1. *The TX, which uses sequential TCAM look-up, requires $\mathcal{O}(wn^2)$ time, where n denotes the number of bits in a word and w denotes the number of words in the TCAM.*

Proof: Since $d = \lceil \log_2 n \rceil$ is the maximum tolerance degree, we have $2^d \geq n \geq 2^{d-1}$. From Lemma 3.4, the total number of X-keys generated by TX is $2^{d+1} - 2$. By substituting them, we have:

$$4n - 2 \geq 2^{d+1} - 2.$$

This indicates that to generate the X-keys, the TX requires $\mathcal{O}(n)$ steps. Moreover, each X-key must be applied to the

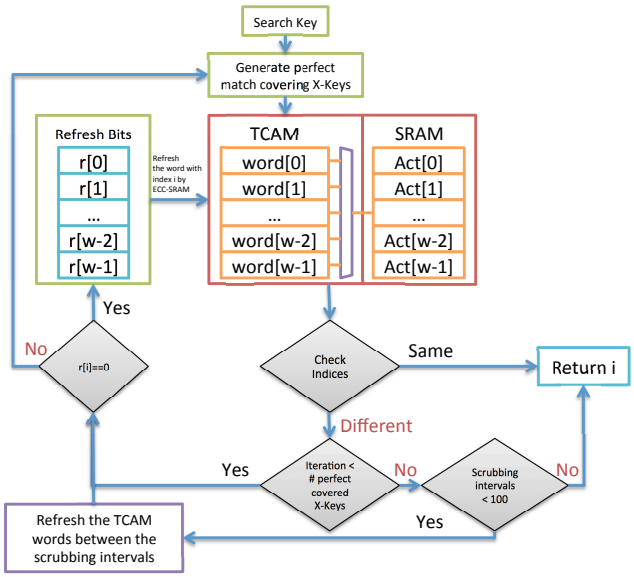


Fig. 5: Generation of partial don't-care keys for k-TX

TCAM. If we use sequential TCAM look-up, then by Lemma 3.3, the total time complexity of the TX is $\mathcal{O}(wn^2)$. □

B. k-TX: A TX for Multiple-Bit-Flip Errors

This section extends the work of TX in Section III-A. k-TX consists of a preprocessing X-Keys look-up table in the controller as shown in Fig. 6, a TCAM, and an ECC-SRAM for handling refresh operation.

1) Generating X-Keys Look-Up Table: Detection of multiple-bit-flip errors (k-flip) requires more than k fields in a word of the TCAM.

Lemma 3.5. *The number of X-Keys in look-up table for multiple-bit-flip errors is:*

$$p = \binom{s}{l} = \frac{s!}{l!(s-l)!},$$

where s is the number of fields in a TCAM word and l is the number of don't care groups in the X-Keys.

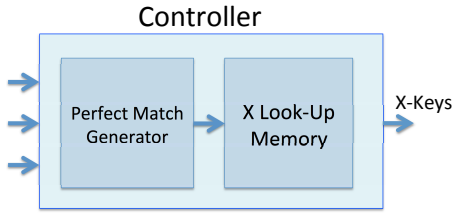


Fig. 6: Controller Illustration

Alg. 4 shows the algorithm to generate perfect match indices which are stored in the X look-up memory. The perfect match generator is a preprocess routine, and can be set based on the prediction of the maximum number of bit flips in a word in a TCAM.

2) Partitioning of TCAM Word: Consider the case, we use the case of $s = 5$ fields classification function: 32-bit source and destination addresses, 16-bit source and destination ports,

Algorithm 4 Perfect Match Generator (*PerfectXKeyGen*())

/* Input: Search Key (*searchKey*), n, s, and l, where n is the number of bit, s is the number of fields in a TCAM word, and l is the number of don't care groups in the X-Keys. */
1: Generate combinations of indices with the number of X-Keys $p = \binom{s}{l}$.
2: If $s \mod l \neq 0$, $g = s$, else $g = 1$.
3: Find perfect matches with the number of perfect covering of X-Keys is $u = \frac{p}{g \times l}$.
4: **while** $c \neq u$ **do**
5: **while** $CoveringVector[Idx]! = g$ **do**
6: Invoke the combinations of indices and find the perfect match. Every combination of indices can be used only once in order to compact the *XLookUp* memory.
7: **if** *Contradict* **then**
8: Backtrack by subtracting the previous addition operation in *CoveringVector*.
9: **else**
10: Write the perfect match pairs in the *XLookUp* memory.
11: **end if**
12: **end while**
13: **end while**
14: Return *XLookUp*

and 8-bit protocol, and the total number of bits in the TCAM is $n = 104$ bits. Fig. 7 shows the partitioning of the TCAM word.

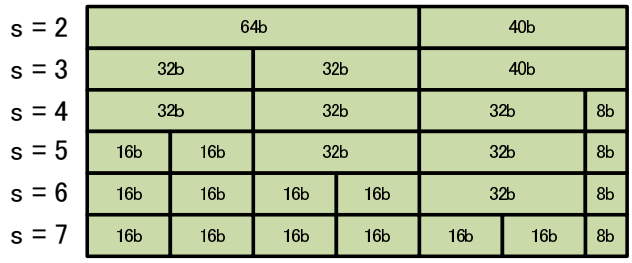


Fig. 7: Partitioning TCAM words

For example, if we have k-bit flips in a word, it requires $s > l \geq k$ to cover all the errors. Let $k = 4$, and we choose $s = 7$ and $l = 4$. Thus, we have $p = 35$ X-Keys to detect the errors. However, based on our observation, if the condition $l/s > 0.57$ holds, the X-Keys cannot detect the soft errors effectively, because the numbers of don't cares in the X-Keys are too large. In this case, the X-Keys will likely to match only the upper part of TCAM.

Fig. 8 shows the example of perfect match covering in the TCAM. In this case, for the best detection of soft errors, the index of an X-Key generated by combination can be only used once. Thus, when it contradicts, we have to reorder the X-Keys.

3) Scrubbing Interval: To reduce the time overhead, we can only use the small value of s and l. Thus, a short scrubbing interval is used in k-TX for improving the tolerance of k-TX against soft errors. First, k-TX compares the lowest and the

Algorithm 5 Multiple-bit-flip detection and correction using partial don't-care keys

/* Input: A TCAM with w words and n bits each word, a search key (*searchKey*), an ECC-SRAM as a back-up of the TCAM for refresh operation, the number of partition s, and the number of groups of don't-care l. */
1: Determine the prediction of the maximum number of multiple-bit-flip in a word of TCAM.
2: Preprocess $XLookUp \Leftarrow PerfectXKeyGen(n, s, l)$.
3: $Prob \Leftarrow rand()$
4: **if** $P_c \geq Prob$ **then**
5: $r \Leftarrow \{0\}$
6: $difIdx \Leftarrow 1$
7: $iterations \Leftarrow 0$
8: **while** $(difIdx) \wedge (iterations < u)$ **do**
9: $XKeys \Leftarrow XLookUp(searchKey, iterations + 1)$
10: **for** $i = (0, \cdots, numKey - 1)$ **do**
11: $Idx[i] \Leftarrow TCAM(XKeys[i])$
12: Store the lowest and the highest matched indices in temporary bits.
13: **end for**
14: Choose the smaller interval of lowest and highest matched indices with the temporary one.
15: $difIdx \Leftarrow CheckIdx(Idx, numKey)$
16: **if** $difIdx$ **then**
17: **for** $i = (0, \cdots, numKey - 1)$ **do**
18: **if** $r[Idx[i]] == 0$ **then**
19: Refresh the TCAM word with index $Idx[i]$
20: $r[Idx[i]] \Leftarrow 1$
21: $Idx[i] \Leftarrow TCAM(XKeys[i])$
22: **end if**
23: **end for**
24: $difIdx \Leftarrow CheckIdx(Idx, numKey)$
25: **end if**
26: $iterations++$
27: **end while**
28: **end if**
29: **if** $difIdx$ **then**
30: If the difference between the lowest and the highest matched indices is less than 100, refresh the scrubbing interval .
31: **end if**
32: Return $TCAM(searchKey)$

highest indices of matched X-Keys. If the difference between the lowest matched index and the highest matched index is less than the previous stored scrubbing interval, k-TX will store the scrubbing interval. Finally, if the X-Keys match different indices at the end of the routine, the TCAM will be refreshed based on the stored scrubbing interval.

Example 3.1. *Consider the TCAM in Fig. 9 and assume that the search key is 0111, where $n = 4$, $s = 4$ and $l = 2$. Perform detection and correction of a soft error using X-keys. First, X look-up memory is filled by perfect match pairs of indices. The number of generated X-keys is $p = \binom{s}{l} = 6$. The X-keys are 01**, **11, 0*1*, *1*1, *11* and 0**1. The X-keys are applied to the TCAM sequentially and the TCAM returns the matched indices. The 1st, 2nd, 3rd, 5th and 6th X-keys match the 1st TCAM word and only the 4th X-key matches the 2nd TCAM word. After that, Algorithm 2 checks the indices and returns 1 which means the indices have different values (a soft error is detected). Next, the refresh bits of TCAM words are checked with the corresponding indices (Line 19, Alg. 5. If the refresh bit is 0, then a refresh operation for the corresponding TCAM word is performed using the ECC-SRAM. In this case, the 1st and 2nd TCAM words are refreshed. Finally, the search key is applied to the TCAM and matches the 3rd TCAM word.* ■

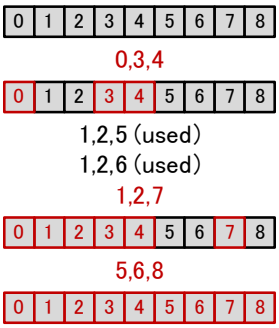


Fig. 8: Example of perfect matching in TCAM word

When the TCAM has entries with smaller *Hamming* distance, the TX may suspect that there is a soft error in the TCAM by the X-keys. However, the TX will check the refresh bits and refresh all the unrefreshed TCAM words with the corresponding X-key matched indices. Thus, the TCAM words are updated, regardless there is a soft error or not. In this case, it implies that the X-keys may matched the words where no soft error in there. And finally, the TX applies the original search key to the corrected TCAM.

Theorem 3.2. *The k-TX, which uses sequential TCAM look-up, requires $\mathcal{O}(pwn)$ time, where $p = \binom{s}{l}$, s is the number of fields in a TCAM word, l is the number of don't care groups in the X-keys, n is the number of bits in a word, and w is the number of words in the TCAM.*

Proof: The maximum number of iterations is $p = \binom{s}{l}$. Moreover, each X-key must be applied to the TCAM. If we use sequential TCAM look-up, then by Lemma 3.3, the total time complexity of the TX is $\mathcal{O}(pwn)$. □

Theorem 3.3. *Soft errors can be found by X-keys that covers all bits of the search key. If all the X-keys match the same word, then the matched word is correct.*

Proof: The correct match is represented by the intersection of all matched X-keys:

$$f = \bigwedge_{i=1}^{p} g_i,$$

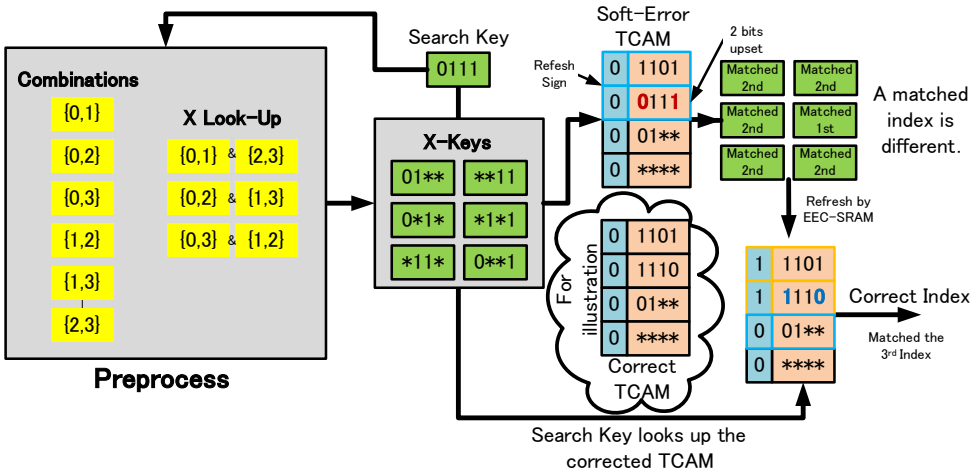
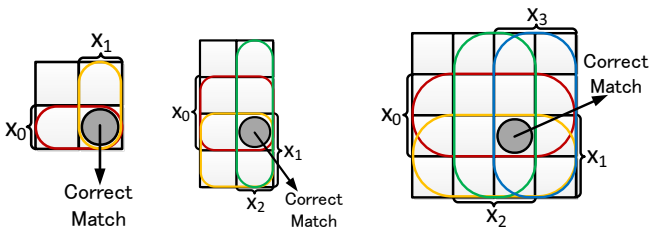


Fig. 9: Detection of Multi-Bit-Flip Errors ($k = 2$) using X-keys

where p is the number of X-keys, f is a correct match and g_i is an X-key with relations $f \subset g_i$ and $g_i \not\subset g_j, i \neq j$. □

Fig. 10 illustrates the covering of X-keys to a correct match for various numbers of bits n and various numbers of don't-care bits inserted in the X-keys q.



(a) $n = 2, q = 1$ (b) $n = 3, q = 2$ (c) $n = 4, q = 3$

Fig. 10: Illustration of X-key covering

IV. Experimental Results

For evaluation, we used packet classification benchmarks generated by ClassBench [17]. First, we generated about 1000 rules and 100000 search keys for each ACL1, ACL2, ACL3, ACL4 and ACL5 filter by ClassBench. Then, we used these benchmarks and evaluated by the computer program in OSX with i7 Intel machine and 8 GB memory. In this case, we compared our proposed scheme with two representative schemes from recent works: TCAM scrubbing (TS) [11] and TCAM checker (TC) [6] by implementing them in the same evaluation environment. Table II shows the numbers of rules and the numbers of search keys.

TABLE II: ACL filters and their search keys

Type	ACL1	ACL2	ACL3	ACL4	ACL5
# Rules	950	972	991	988	838
# Search Keys	95001	97288	99684	99741	83828

Table III compares misclassifications for TS, TC, TX, and k-TX, where the probability of bit flips in a word $P_e = 0.1$ and the probability of a scheme being used is $P_c = 0.1$. In this experiments, we use tolerance degree $d = 5$ for TX. As shown in Table III, we can see the significant tolerance improvement for our proposed method k-TX. We can see also, TX cannot handle multiple-bit-flip errors. Bold integers in the table show the minimum of numbers of misclassifications.

Table IV compares the total times for TS, TC, TX, and k-TX for all ACL filters in seconds. In TX, we can choose the degree d to solve single-bit-flip errors. In k-TX, we can choose the trade-off between time and performance by selecting s and l in our proposed method k-TX. k-TX requires more time, since it has combinatorial complexity and performs in a short scrubbing interval. As shown in Table III, in term of tolerance performance, k-TX with $s = 7$ and $l = 4$ is the best. However, if we consider the time complexity, we can choose k-TX with $s = 5$ and $l = 2$.

V. Conclusion

This paper proposes a novel soft-error tolerant TCAM for multiple-bit-flip errors using partial don't-care keys (X-keys), called k-TX. k-TX corrects up to k-bit flip errors and significantly enhances the tolerance of the TCAM against soft errors, where k is the maximum number of bit flips in a word of TCAM. k-TX consists of a TCAM, a preprocessed don't-care-bit index look-up memory (X look-up), and an ECC-SRAM. First, k-TX randomly selects a search key. After that, k-TX detects multiple-bit-flip errors using the generated X-keys by X look-up. If the keys match the different locations, then a soft error is detected and k-TX refreshes the TCAM words by using a backup ECC-SRAM. Experimental results demonstrate the advantages of k-TX. Moreover, the hardware overhead of k-TX is small due to the use of only a single TCAM. k-TX can be easily implemented and is useful for fault-tolerant packet classifiers.

In the future, we are going to optimize the scrubbing interval of k-TX and the refresh operation of the scrubbing interval. Thus, we can improve the time performance of k-TX.

Acknowledgments

This work was partly supported by JSPS Grant-in-Aid for Scientific Research (B) #25280016, JSPS Grant-in-Aid for

TABLE III: Comparison of misclassifications for multiple-bit-flip errors for ACL5 filter

k-flip	TS	TC	TX	k-TX								
			$(d=5)$	$(s=4)$ $(l=2)$	$(s=5)$ $(l=2)$	$(s=5)$ $(l=3)$	$(s=6)$ $(l=2)$	$(s=6)$ $(l=3)$	$(s=7)$ $(l=2)$	$(s=7)$ $(l=3)$	$(s=7)$ $(l=4)$	$(s=7)$ $(l=5)$
ACL1												
1	837	168	101	622	**35**	682	650	630	646	633	643	673
2	956	195	220	647	**20**	766	684	680	86	58	74	112
3	369	249	316	50	**38**	183	98	84	101	73	83	162
4	525	236	386	59	**67**	193	177	98	175	123	93	187
5	272	236	396	85	**36**	173	178	124	189	85	100	175
ACL2												
1	21	48	30	17	27	11	31	25	32	8	**5**	**5**
2	9	36	20	20	12	21	11	13	17	10	**3**	6
3	32	95	72	31	31	40	25	**19**	25	27	22	29
4	19	18	68	15	17	14	11	**6**	11	16	13	14
5	17	41	150	18	18	14	18	13	31	8	5	**1**
ACL3												
1	237	264	190	59	79	88	168	99	148	74	**45**	87
2	458	436	1342	83	128	105	287	137	264	82	**49**	261
3	453	561	1053	138	144	357	540	124	521	**92**	128	328
4	580	466	1024	128	158	377	566	198	522	324	**105**	430
5	542	513	968	128	189	204	529	264	439	170	**117**	259
ACL4												
1	588	506	235	69	99	378	130	92	120	91	**71**	429
2	600	674	900	107	114	118	231	111	152	**72**	94	193
3	1164	603	1288	**134**	**134**	951	278	192	248	811	157	942
4	1301	528	849	685	250	278	320	152	236	**105**	118	317
5	1189	851	1708	155	163	495	605	186	284	**107**	120	639
ACL5												
1	1167	375	258	79	180	437	131	89	151	109	**74**	468
2	941	622	729	110	271	404	312	164	278	187	**92**	569
3	4943	3314	1293	153	187	154	262	120	223	127	**96**	861
4	1488	720	1727	327	324	541	702	397	377	195	**97**	823
5	3186	686	3644	241	555	1057	742	364	674	424	**176**	1366

TABLE IV: Comparison of total detection and correction time for multiple-bit-flip errors in seconds

	TS	TC	TX	k-TX								
			$(d=5)$	$(s=4)$ $(l=2)$	$(s=5)$ $(l=2)$	$(s=5)$ $(l=3)$	$(s=6)$ $(l=2)$	$(s=6)$ $(l=3)$	$(s=7)$ $(l=2)$	$(s=7)$ $(l=3)$	$(s=7)$ $(l=4)$	$(s=7)$ $(l=5)$
ACL 1 to 5	13.156	28.291	45.245	26.471	30.879	34.293	32.493	39.402	40.995	68.041	81.296	56.506

Scientific Research on Innovative Areas #24650022, and JST-NSC Grant-in-Aid for Japan-Taiwan Joint Research on the Testing of Nano Devices.

REFERENCES

[1] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE JSSC*, vol. 41, No. 3, pp. 712-727, March 2006.

[2] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Computing Surveys*, vol. 37, no. 3, pp. 238-275, Sept. 2005.

[3] C. Slayman, "Soft errors–Past history and recent discoveries", *IEEE IIRWFR*, pp. 25-30, Oct. 2010.

[4] A. Dixit and A. Wood, "The impact of new technology on soft error rates", *IEEE International Reliability Physics Symposium*, pp. 5B.4.1-5B.4.7, April 2011.

[5] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo and T. Toba, "Impact of scaling on neutron-induced soft error in SRAMs from a 250 nm to a 22 nm design rule", *IEEE Transactions on Electron Devices*, vol. 57, no. 7, pp. 1527-1538, July 2010.

[6] M. Z. Shafiq, C. Meiners, Z. Qin, K. Shen, and A. X. Liu, "TCAM-Checker: A software approach to the error detection and correlation of TCAM-based networking system," *Journal of Network and System Management*, vol. 21, no. 3, pp. 335-352, Sept. 2013.

[7] K. Pagiamtzis, N. Azizi, and F. N. Najm, "A soft-error tolerant content-addressable memory (CAM) using an error-correcting-match scheme," *IEEE CICC*, pp. 301-304, Sept. 2006.

[8] W. Leung, F. Hsu, and M. E. Jones, "The ideal SoC memory: 1T-SRAM", *13th ASIC/SOC Conference*, pp 32-36, Sept. 2000.

[9] H. Noda, K. Dosaka, H. J. Mattausch, T. Koide, F. Morishita, and K. Arimoto, "A reliability-enhanced TCAM architecture with associated embedded DRAM and ECC," *IEICE Transactions on Electronics*, vol. E89-C, no. 11, Nov. 2006.

[10] I. Syafalni, T. Sasao, X. Wen, S. Holst, and K. Miyase, "Soft-error tolerant TCAMs for high-reliability packet classifications," *IEEE APCCAS*, pp. 471-474, Nov. 2014.

[11] S. Baeg, S. Wen, and R. Wong, "Minimizing soft errors in TCAM devices: A probabilistic approach to determining scrubbing intervals" *IEEE TCAS*, vol. 57, no. 4, pp. 814-822, April 2010.

[12] P.-T. Huang and W. Hwang, "A 65 nm 0.165 fJ/bit/search 256 x 144 TCAM macro design for IPv6 lookup tables", *IEEE Journal of Solid-State Circuits*, vol. 46, no. 2, pp. 507-519, Feb. 2011.

[13] S. Pontarelli, M. Ottavi, A. Evans, and S.-J. Wen, "Error detection in ternary CAMs using bloom filters", *DATE*, pp. 1474-1479, March 2013.

[14] I. Syafalni and T. Sasao, "On the numbers of products in prefix SOPs for interval functions," *IEICE Transactions on Information and System*, vol. E96-D, no. 55, pp. 1086-1094, May 2013.

[15] I. Syafalni and T. Sasao, "A TCAM generator for packet classification," *IEEE ICCD*, pp. 322-328, Oct. 2013.

[16] I. Syafalni and T. Sasao, "Head-tail expressions for interval functions," *IEICE Trans. Fund.*, vol. E97-A, no. 10, pp. 2043-2054, Oct. 2014.

[17] D. E. Taylor, J. S. Turner, "ClassBench: a packet classification benchmark," *IEEE/ACM TON*, vol. 3, no. 15, pp. 499-511, June 2007.