

PEDS: A Parallel Error Detection Scheme for TCAM Devices

Anat Bremler-Barr^{*}, David Hay[†], Danny Hendler[‡] and Ron M. Roth[§]

^{*}The Interdisciplinary Center, Herzliya, Israel. Email: bremler@idc.ac.il

[†]Politecnico di Torino, Turin, Italy. Email: hay@tlc.polito.it.

[‡]Ben-Gurion University, Be'er Sheva, Israel. Email: hendlerd@cs.bgu.ac.il

[§]Technion - Israel Institute of Technology, Haifa, Israel. Email: ronny@cs.technion.ac.il

Abstract—Ternary content-addressable memory (TCAM) devices are increasingly used for performing high-speed packet classification. A TCAM consists of an associative memory that compares a search key in parallel against all entries. TCAMs may suffer from error events that cause ternary cells to change their value to any symbol in the ternary alphabet “0”, “1”, “*”. Due to their parallel access feature, standard error detection schemes are not directly applicable to TCAMs; an additional difficulty is posed by the special semantic of the “*” symbol.

This paper introduces PEDS, a novel parallel error detection scheme that locates the erroneous entries in a TCAM device. PEDS is based on applying an error-detection code to each TCAM entry, and utilizing the parallel capabilities of the TCAM, by simultaneously checking the correctness of multiple TCAM entries. A key feature of PEDS is that the number of TCAM lookup operations required to locate all errors depends on the number of symbols per entry rather than the (orders-of-magnitude larger) number of TCAM entries. For large TCAM devices, a specific instance of PEDS requires only 200 lookups for 100-symbol entries, while a naive approach may need hundreds of thousands lookups. PEDS allows flexible and dynamic selection of trade-off points between robustness, space complexity, and number of lookups.

I. INTRODUCTION

Ternary content-addressable memory (TCAM) devices are increasingly used for performing high-speed *packet classification*, which is an essential component of many networking applications such as routing, monitoring and security. For packet classification, routers use a *classification database* that consists of *rules* (sometimes called *filters*). Each such a rule specifies a certain pattern, which is based on packet header fields, such as the source/destination addresses, source/destination port numbers and the protocol type. Furthermore, each rule is associated with an action to apply to the packets that matched the pattern rule. Packet classification is often a performance bottleneck in the network infrastructure since it should be done at the routers' line rate. It is therefore important to design packet classification solutions that scale to millions of key search operations per second. TCAM enables parallel matching of a key against all entries and thus provides high throughput that is unparalleled by software-based (or SRAM-based) solutions.

A TCAM is an associative memory hardware device that consists of a table of fixed-width *TCAM entries*. Each entry consists of W symbols taking on the ternary alphabet

{“0”, “1”, “*”}, where {“0”, “1”} are *proper bit* values and “*” stands for “*don't-care*.” The entry width of contemporary TCAM devices is configurable to a width of 72, 144, 288 or 576 symbols. For classification applications, TCAMs are configured to have width of 144 symbols, which leaves a few dozens of unused symbols (usually 36 symbols), called *extra bits* [1]. For notational consistency, we henceforth use the term *extra symbols* instead of extra bits. Note that the size (i.e. the number of TCAM entries) of contemporary TCAMs is orders-of-magnitude larger than their width. Current TCAMs can store more than 128K ternary entries that are 144 bits wide in a single device.

The input to the TCAM is a ternary word of length W called a *search key*. Search key u matches entry v , if the proper bits of u agree with those of v . The basic function of a TCAM is to simultaneously compare a search key with all TCAM entries. The index returned by the lookup operation is computed by a TCAM module called *match-line (ML) encoder*. If there is a single matching TCAM entry, its index is output by the ML encoder. If several entries match the search key, most TCAMs return the index of the *highest priority* entry, i.e., the entry with the smallest index; in this case, the specific type of ML encoder is called a *priority encoder*.

A. The Problem

Memory chips suffer from error events (often called *soft errors*), typically caused by low-energy alpha particles, neutron hits, or cosmic rays [2]. In a TCAM error event, a TCAM symbol can change its value to any symbol in {“0”, “1”, “*”}.

TCAMs are highly susceptible to soft errors, since TCAM chips are denser than regular memory chips, due to the additional logic required to perform parallel searches. Furthermore, TCAM memory consists of an array of SRAM cells, which are known to be more susceptible to soft errors than DRAM cells.

The conventional techniques of coping with errors in regular (that is, non-associative) memory, such as SRAM or DRAM, cannot be applied to TCAM. The soft errors problem in regular RAM is typically handled by using an error-detection or error-correction code (ECC). An ECC check is applied to a memory word *upon access*, which implies that only a single ECC check circuit is required. This single circuit is sufficient

since, in non-associative memory, the input to the memory device is an address and the output is the value stored at that address. Therefore, checking memory words just before they are accessed will capture all errors detectable by the ECC.

This is no longer true for TCAM devices, however, due to their parallel access feature. Recall that the input for the TCAM is a search key and the output is the highest priority entry that matches the search key, where all the entries are checked in parallel. Thus, a TCAM introduces new types of errors: an error in a TCAM entry may result either in an entry rejecting a search key, even though it should have been matched with it (*false miss*), or result in an entry matching a search key, even though it should not have been matched with it (*false hit*). When more than one match is possible, a false miss in a high-priority entry may result in matching a lower-priority entry that should not have been matched. We call this kind of errors an *indirect false hit*. Thus, a false miss may cause an indirect false hit.

To better understand these types of error, consider the toy example depicted in Figure 1, which shows a TCAM database of width 4, consisting of 3 entries, before and after a soft error hits the third symbol of the first entry. When a lookup operation is applied to the error-hit TCAM with search key “1100”, there are no matches, whereas the correct output should have been 1 (the index of the first matching entry). This is an instance of a *false miss*. A conventional ECC mechanism cannot be used to detect such an error, since all entries are compared with the search key in parallel (and none is matched), hence the only way of finding that the output is erroneous is to apply an ECC check to *all entries*. Next, consider a lookup operation with search key “1111”. Here, the output of the TCAM is 1 instead of 2. This is an instance of a (direct) *false hit*. False hits are the only type of errors in which an ECC check of the matched entry can identify the error. Finally, consider a lookup operation with search key “1101”. In this case, the TCAM outputs 3 instead of 1. This is an instance of an *indirect false hit*, since the wrong entry is hit (matched) due to an error that causes a miss on a higher-priority entry. In this case, performing an ECC check on the matched entry will not identify the error. Note that false hit and false miss events also occur in *binary* CAM, where the bits are only “0” or “1” values.

Matching a search key with the wrong entry can have significant adverse effect on the classification process, since different entries have different actions associated with them. As an example, such an error may cause packets that should be dropped due to security reasons to be erroneously forwarded.

B. Our results

In this paper, we present *PEDS*, a novel parallel error detection scheme for TCAM devices. *PEDS* utilizes the parallel matching capability inherent to TCAM devices and requires only minor hardware changes, less than any prior art TCAM error detection schemes that we are aware of. The key idea of

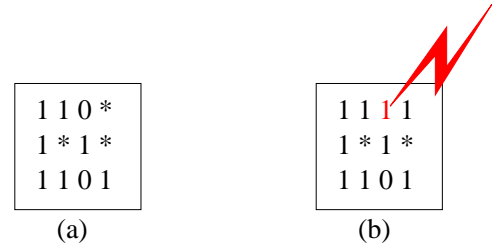


Fig. 1. TCAM memory with $W=4$, before (a) and after (b) an error event.

the scheme is that of applying a sequence of TCAM searches for a predetermined set of search keys, and analyzing their results. The number of lookups required by *PEDS* for checking all entries depends linearly on the entry width, rather than on the orders-of-magnitude larger number of TCAM entries. For example, a specific instance of *PEDS* requires only 200 lookups for 100-symbol entries, using a single extra symbol per entry, while the number of entries is usually more than 128K.

PEDS lookup operations do not have to be consecutive and can be performed lazily during idle TCAM cycles. As an example, in a TCAM that can perform 100 million searches per second (MSPS) (e.g., [3]) which is 99% loaded, *PEDS* needs only 0.2 milliseconds to detect all errors, using the scheme we describe in Section V.

For error correcting, an application can maintain a copy of the TCAM database in DRAM. When an erroneous TCAM entry is identified by *PEDS*, the application can correct it by copying the correct value from DRAM.

We evaluate the cost and performance of our scheme according to the following four performance metrics:

- 1) **Resilience:** the number of errors per TCAM entry that our scheme can detect.
- 2) **Space:** the number of check symbols per entry that are required.
- 3) **Time:** the number of TCAM lookup operations required to check all entries.
- 4) **Hardware changes:** the number (per entry) of additional logical gates and registers that are required for our scheme.

PEDS can use the extra symbols typically available in TCAM entries as check symbols for increased resilience and/or decreased time complexity. *PEDS* is unique in that it allows a dynamic selection of trade-off points between the above four criteria according to application requirements.

The rest of this paper is organized as follows: In Section II we provide an overview of related work. Section III describes the key ideas behind *PEDS* and its architecture. In Sections IV and V we describe two specific instances of *PEDS* that result in different performance trade-offs. Practical considerations and concluding remarks are given in Sections VI and VII, respectively. In the Appendix, we present a sketch of the proof

about the impossibility of creating error correcting code for TCAM, and the necessity of additional hardware (or further functionality) for error detection.

II. RELATED WORK

Currently, the state of the art solution in the Industry is to use an error detection code for each TCAM entry, and periodically read the entire one by one and check their correctness against the coding scheme chosen. While conceptually simple, this technique requires reading each entry from the device, implying that the TCAM is equipped with additional hardware in order to verify that each matching TCAM entry is error-free. Even for a weak detection code, namely adding a parity check symbol that protect only against one error, at least $W-1$ XOR gates and a register is needed. While our technique is capable to detect errors with only one XOR gates and one bit register to protect against multiple errors. The reduced hardware requirement of our solution has a direct influence on the die size, which is one of the major factors of the TCAM chip price. Moreover, the simple industrial solution requires a number of cycles linear in the number of TCAM *entries*. This is problematic as it may imply that multiple lookup operations return wrong values before an error is identified and fixed.

Prior art suggests several additional methods of coping with TCAM errors. All these methods focus on reducing the errors rate rather than on faster error detection/correction. Moreover, they require much more significant changes to TCAM hardware than PEDS. Roughly speaking, while prior art techniques are mostly chip-design solutions, PEDS is more of an algorithmic technique.

Noda et al. [4] describe a TCAM design that is based on DRAM cells instead of SRAM cells. DRAM-based TCAM devices can be more robust, since DRAM is less susceptible to soft errors than SRAM. Moreover, since SRAM memory is much faster and consumes less power than DRAM memory, it is not clear whether DRAM-based TCAM has the potential of becoming a viable alternative to SRAM-based TCAM. A different technique based on a similar idea was proposed by Noda et al. [5], which use ECC-enhanced embedded DRAM cells *in addition* to SRAM cells. DRAM cell values are continuously copied to the corresponding SRAM cells. This technique results, however, in a significant chip area increase.

Azizi and Najm [6] propose a new family of feedback-enhanced TCAM cells. Their designs exploit the fact that TCAM cells have an invalid state that is never used and make it instable, thus allowing other—legal—states to become more stable and, by that, decreasing the likelihood of soft errors. Their simulation results indicate that this technique can reduce error rates by up to 40% at the cost of a significant increase in area size.

We also mention the work of Pagiamtzis et al. [7], which applies to (binary) CAMs. They propose a design that uses coding guaranteeing that every two entries will differ in at least a prescribed number of positions (that number is 4 in

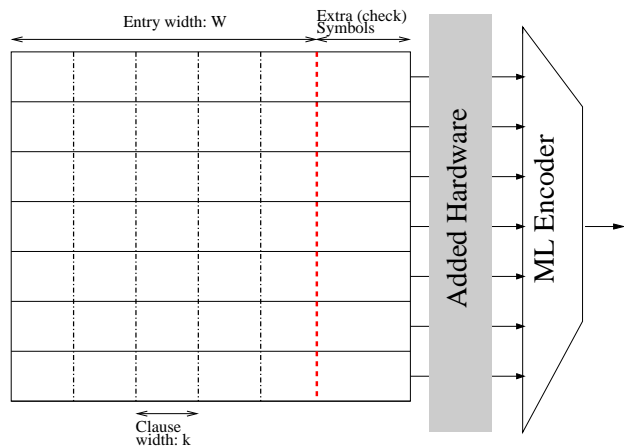


Fig. 2. The basic architecture for PEDS scheme. For ease of presentation the extra symbols are allocated at the end of each entry; throughout the paper we refer to the extra symbols as if they are interleaved within the entry such that each clause is followed by the extra symbols corresponding to it.

the example that they provide). In addition, they modify the match line sensing scheme so that it signals a match even when the search key and the entry disagree on one position. Their scheme requires adding 9 parity symbols per each 72-cell CAM entry and modifying the match-line sensing scheme. Their scheme does not correct the error-hit symbols back to their original state and an additional mechanism is required for that.

In contrast with all prior art, our technique allows fast detection of all erroneous entries and requires significantly less hardware changes to be applied TCAM hardware. Prior art algorithms use extra symbols for improving the space efficiency of packet classification and intrusion detection applications [8]–[12]. To the best of our knowledge, our work is the first that can use available extra symbols for TCAM error detection. This allows greater robustness and/or faster error detection without incurring additional hardware cost.

III. PARALLEL ERROR DETECTION SCHEME

In this section we explain the basic Parallel Error Detection Scheme (PEDS). We first explain in Section III-A the essence of our innovative idea using a toy example. Then, the general framework is presented in Section III-B. The following Sections IV and V describe specific implementations with specific trade-offs between the hardware complexity and the time complexity.

A. PEDS innovative basic idea

We start by demonstrating our scheme using a very simple example. Note that this example is used only for demonstration purposes since it employs a very large number of extra symbols and therefore cannot be implemented in practice.

Suppose that for each entry of width W , we add W extra symbols such that each original symbols is duplicated. Let the j -th pair of symbols denote the original j -th symbol and its

duplication. For example, the entry “0*10” will be coded as “00**1100”, and “**” is the second pair of symbols. In this case, if we assume that only one error can occur in each pair of symbols, the entry is correct if and only if the symbols in each pair are equal.

We will check the correctness of the entries, by iteratively for each j , checking the correctness of the j -th pair simultaneously for all entries. This is done by applying two search keys

$$(*^{2j}) 01 (*^{2W-2j-2}) \quad \text{and} \quad (*^{2j}) 10 (*^{2W-2j-2}),$$

for every $i \in \{1, \dots, W\}$.

If neither of the search keys match the entry, it is correct since the j -th pair is either 00 or 11. If both search keys match the entry, it is also correct since the j -th pair is **. On the other hand, if only one search key matches the entry it must be incorrect: if only $(*^{2j}) 01 (*^{2W-2j-2})$ matches the entry, the j -th pair is either 01, 0*, or *1; If only $(*^{2j}) 10 (*^{2W-2j-2})$ matches the entry, the j -th pair is either 10, 1*, or *0.

The resilience of the code is W errors with the restriction that no two errors can be made in the same pair. The code requires W extra symbols per entry, and the time to detect all errors is $2W$ searches. In addition, in order to distinguish between odd and even number of matches, only a mod-2 counter should be added at the end of each match-line.

B. PEDS architecture

This section describe the basic PEDS scheme, which is illustrated in Figure 2. Each entry of size W is partitioned into k -symbol clauses. For each such a clause, we compute check symbols according to a prescribed error-detection code over the ternary alphabet and store them in the extra symbols positions. In most of the examples that we present in this paper, the code requires only a single check symbol per clause, but generalization to other error-detection codes are also possible, as described in Section IV. To detect errors, we check each of the W/k clauses (in conjunction with its check symbols) separately—but *concurrently for all entries*—against a predetermined set of search keys. Our predetermined set of search keys has the the following property: An entry is error free if and only if the number of search keys that match it is in a predetermined set T . A hardware-based mechanism counts the number of matches per entry and determines whether it belongs to T . This mechanism is the only hardware change required for our proposed scheme and can be implemented between the cell array of the TCAM and its ML priority encoder, as depicted in Figure 2.

In Section IV, we fix T to the set of even integers, thus the entry is error free if and only if the number of keys that match the entry is even. This, in turn, requires adding a mod-2 counter between each match line and the ML priority encoder. With this hardware change, our scheme presents a trade-off between its resilience, the number of extra symbols it requires, and the time it takes to check all entries. We show that as the

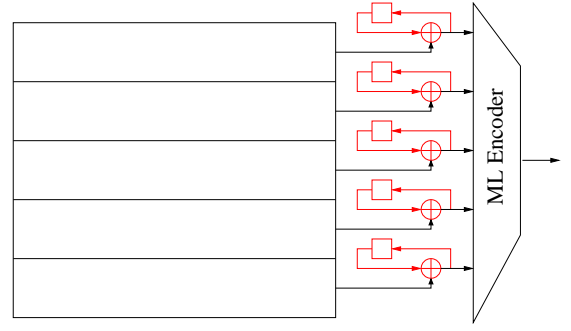


Fig. 3. The Hardware change required for implementing the fast detection scheme described in Section IV. \oplus denotes a XOR gate.

resilience increases, the time for checking all entries decreases but more extra symbols are needed.

Section V presents another scheme, where T is the set of the integer multiples of 3. This results in a faster error detection scheme albeit with higher hardware complexity: ternary mod-3 adders (rather than binary mod-2 counters) should be implemented at the end of each match-line.

IV. PEDS WITH MOD-2 COUNTERS

In this section, we present a scheme which locates the erroneous entries in a TCAM by performing error detection in parallel across all entries. Our scheme requires only a minor change of the hardware: adding a *mod-2 counter* (that is, a XOR gate and a single-bit register) at the end of each match line. This hardware change is shown in Figure 3. The simple example which was presented in Section III-A can be seen as a special case of the scheme we present here.

Our coding scheme is highly configurable and it introduces a trade-off between its error resilience, the number of check (redundancy) symbols it requires, and the time it takes to identify all the entries of the TCAM that are erroneous. The operation point on the curve that relates these parameters can be decided upon after the deployment of the TCAM device.

A second scheme, based on mod-3 counters, will be presented in Section V.

A. Encoding the contents of the TCAM

To facilitate the presentation and the analysis in the sequel, we will regard the three symbols “0”, “1”, and “*” as elements of the finite field of three elements, $\mathbb{F}(3)$. This field, which we denote hereafter by \mathbb{F} , consists of the three elements $+1$, -1 , and 0 , with addition and multiplication taken modulo 3. We will use the following mapping between TCAM symbols and elements of \mathbb{F} :

$$\begin{aligned} \text{“*”} &\longleftrightarrow 0 \\ \text{“0”} &\longleftrightarrow +1 \\ \text{“1”} &\longleftrightarrow -1. \end{aligned}$$

Under this mapping, each word of a given length ℓ over $\{“0”, “1”, “*”\}$ will be seen as a row vector in \mathbb{F}^ℓ . When no confusion arises, we will interchangeably use both symbol

sets—{“0”, “1”, “*”} and $\{+1, -1, 0\}$ —to denote the element set of \mathbb{F} .

Suppose that the designed raw width of the TCAM is W , namely, each TCAM entry is to be able to store W symbols. We refer to these symbols as the W information symbols. To allow for error detection within a TCAM entry, each entry will be extended into W' ($> W$) symbols by adding check symbols, in a manner that is described next.

We fix a parameter k which, for the sake of simplicity, is assumed to divide W , and sub-divide the W information symbols in an entry into non-overlapping clauses of length k . We also fix some linear $[n, k, d]$ code \mathcal{C} over \mathbb{F} . Namely, \mathcal{C} is a set of 3^k vectors (referred to as *codewords*) of length n over \mathbb{F} that form a linear space over \mathbb{F} , and the minimum (Hamming) distance between any two distinct codewords in \mathcal{C} is d : every two codewords in \mathcal{C} differ in at least d positions (and there are two codewords that differ in exactly d positions). An *encoder* for \mathcal{C} is any one-to-one mapping from \mathbb{F}^k into \mathcal{C} and, without real loss of generality of \mathcal{C} , we can assume that the encoding of a vector $\mathbf{u} \in \mathbb{F}^k$ is carried out by appending $n-k$ check symbols to \mathbf{u} , thereby forming the respective image codeword in \mathcal{C} . The value $n-k$ is called the *redundancy* of \mathcal{C} . Since the code \mathcal{C} has minimum distance d , we can detect any pattern of less than d errors that occur in codewords of \mathcal{C} , including changes to and from “*” (the latter being represented by the element 0 of \mathbb{F}).

Turning to our TCAM, in every entry, we encode each k -symbol clause into an n -symbol *block*, such that each block is a codeword of \mathcal{C} . We then concatenate the resulting W/k blocks to form an entry of the TCAM of length $W' = nW/k$. The simple example which was presented in Section III-A is a special case obtained when $k = 1$, $n = 2$, and \mathcal{C} is the code $\{(0\ 0), (+1\ +1), (-1\ -1)\}$ (clearly, this code has minimum distance $d = 2$).

B. Analysis

We next derive several results which will lead to the decoding strategy of locating the erroneous entries in the TCAM, assuming that these entries are encoded as we have just described at the end of Section IV-A.

We start by recalling well known concepts from coding theory. Let \mathcal{C} be the linear $[n, k, d]$ code over \mathbb{F} which is used in the encoding process. The linearity of \mathcal{C} implies that we can associate with \mathcal{C} a *parity-check matrix*, which is an $r \times n$ matrix H over \mathbb{F} with the property that \mathcal{C} forms its right kernel, namely:

$$\mathcal{C} = \{ \mathbf{v} \in \mathbb{F}^n : H\mathbf{v}^T = \mathbf{0} \} ;$$

here, $(\cdot)^T$ denotes transposition, the product $H\mathbf{v}^T$ is carried out over the field \mathbb{F} (namely, modulo 3), and $\mathbf{0}$ stands for the all-zero vector. The number of rows r must be at least the redundancy $n-k$, and equality can be attained when H is selected to have linearly independent rows over \mathbb{F} (there are many parity-check matrices for any given code).

Given such a matrix H with r rows $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_r \in \mathbb{F}^n$ and a vector $\mathbf{v} \in \mathbb{F}^n$, the *syndrome* \mathbf{s} is defined as the following column vector in \mathbb{F}^r :

$$\mathbf{s} = H\mathbf{v}^T .$$

Thus, $\mathbf{s} = \mathbf{0}$ if and only if $\mathbf{v} \in \mathcal{C}$. Now, since the minimum distance of \mathcal{C} is d , any pattern of less than d errors cannot change one codeword into another. This can be summarized as follows.

Lemma 1: Let the k -symbol clauses in the TCAM be encoded into n -symbol blocks such that each block is a codeword of a linear $[n, k, d]$ code \mathcal{C} over \mathbb{F} , and let $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_r$ be the rows of a parity-check matrix of \mathcal{C} . Suppose that each block is subject to less than d errors. Then the following two conditions are equivalent for every block \mathbf{v} in the TCAM:

- (i) \mathbf{v} is error-free.
- (ii) $\mathbf{h}_i\mathbf{v}^T = 0$ for every $i = 1, 2, \dots, r$.

We will use this fact in the sequel. We proceed by introducing several terms which, ultimately, will be used to define the set of search keys with which we will perform our decoding, namely, locating the erroneous entries in the TCAM.

For a row vector $\mathbf{u} = (u_1\ u_2\ \dots\ u_n)$ over \mathbb{F} , denote by $J(\mathbf{u})$ the support of \mathbf{u} : $J(\mathbf{u}) = \{j : u_j \neq 0\}$. Namely $J(\mathbf{u})$ contains the indexes of the coordinates which are not “*” in \mathbf{u} . Thus, two row vectors $\mathbf{u} = (u_1\ u_2\ \dots\ u_n)$ and $\mathbf{v} = (v_1\ v_2\ \dots\ v_n)$ in \mathbb{F}^n are said to *match* if they are equal on the intersection of their supports, namely,

$$u_j = v_j \quad \text{for every } j \in J(\mathbf{u}) \cap J(\mathbf{v}) .$$

Given a row vector \mathbf{h} in \mathbb{F}^n and an element $b \in \mathbb{F}$, denote by $\mathcal{S}(\mathbf{h}; b)$ the set

$$\mathcal{S}(\mathbf{h}; b) = \{ \mathbf{u} \in \mathbb{F}^n : J(\mathbf{u}) = J(\mathbf{h}) \quad \text{and} \quad \mathbf{h} \cdot \mathbf{u}^T = b \} .$$

Namely, $\mathcal{S}(\mathbf{h}; b)$ consists of all vectors in \mathbb{F}^n that have nonzero coordinates (that is, coordinates which are not “*”) precisely where \mathbf{h} has, and their scalar product with \mathbf{h} (in \mathbb{F}) equals b .

We associate with every vector $\mathbf{h} \in \mathbb{F}^n$ the following set $\mathcal{L}(\mathbf{h})$:

$$\mathcal{L}(\mathbf{h}) = \mathcal{S}(\mathbf{h}; +1) \cup \mathcal{S}(\mathbf{h}; -1) . \quad (1)$$

Equivalently, $\mathcal{L}(\mathbf{h})$ consists of all the n -prefixes of the vectors in $\mathcal{S}((\mathbf{h}\|+1); 0)$, where $(\cdot\|\cdot)$ denotes concatenation (note that these n -prefixes are all distinct).

The following lemma determines the size of $\mathcal{S}(\mathbf{h}; b)$ and, in particular, the parity of the size (whether it is even or odd). It is the parity of the sizes of the sets $\mathcal{S}(\mathbf{h}; b)$ which will play a primary role in the decoding process to be presented in Section IV-C. Recall that a Hamming weight of a vector over \mathbb{F} is the number of nonzero coordinates in that vector.

Lemma 2: Let \mathbf{h} be a nonzero row vector in \mathbb{F}^n with Hamming weight $w (> 0)$. Then:

- (a) $|\mathcal{S}(\mathbf{h}; 0)| = \frac{2}{3} (2^{w-1} + (-1)^w)$ (which is even).

(b) $|\mathcal{S}(\mathbf{h}; \pm 1)| = \frac{1}{3}(2^w - (-1)^w)$ (which is odd).

Proof: The proof is by induction on w , where the induction base ($w = 1$) is easy to verify. As for the induction step, suppose without real loss of generality that the last coordinate in \mathbf{h} is nonzero, and write $\mathbf{h} = (\mathbf{h}' \parallel \pm 1)$, where the Hamming weight of \mathbf{h}' is $w-1$ (> 0). We have

$$\mathcal{S}(\mathbf{h}; 0) = \bigcup_{b \in \{+1, -1\}} \left\{ (\mathbf{u} \parallel \pm b) : \mathbf{u} \in \mathcal{S}(\mathbf{h}'; -b) \right\}$$

(where the sign is determined by the last coordinate in \mathbf{h}). Therefore,

$$\begin{aligned} |\mathcal{S}(\mathbf{h}; 0)| &= |\mathcal{S}(\mathbf{h}'; +1)| + |\mathcal{S}(\mathbf{h}'; -1)| \\ &= 2 \cdot \frac{1}{3}(2^{w-1} - (-1)^{w-1}) \\ &= \frac{2}{3}(2^{w-1} + (-1)^w), \end{aligned}$$

where the second equality follows from the induction hypothesis.

As for $\mathcal{S}(\mathbf{h}; \pm 1)$, by symmetry we have $|\mathcal{S}(\mathbf{h}; +1)| = |\mathcal{S}(\mathbf{h}; -1)|$ and, so,

$$|\mathcal{S}(\mathbf{h}; 0)| + 2|\mathcal{S}(\mathbf{h}; \pm 1)| = \sum_{b \in \mathbb{F}} |\mathcal{S}(\mathbf{h}; b)| = 2^w.$$

Hence,

$$|\mathcal{S}(\mathbf{h}; \pm 1)| = 2^{w-1} - \frac{1}{2}|\mathcal{S}(\mathbf{h}; 0)| = \frac{1}{3}(2^w - (-1)^w),$$

as claimed. \blacksquare

We use Lemma 2 to prove the following theorem.

Theorem 3: Let \mathbf{h} and \mathbf{v} be row vectors in \mathbb{F}^n where \mathbf{h} is nonzero. The following two conditions are equivalent:

- (i) $\mathbf{h} \cdot \mathbf{v}^T = 0$.
- (ii) The number of vectors in $\mathcal{L}(\mathbf{h})$ that match \mathbf{v} is even.

Proof: Let w be the Hamming weight of \mathbf{v} , and write $b = \mathbf{h} \cdot \mathbf{v}^T$ ($\in \mathbb{F}$). By possibly permuting (simultaneously) the coordinates of \mathbf{h} and \mathbf{v} , we can assume that the nonzero coordinates of \mathbf{v} occupy its first w positions. Write

$$(\mathbf{h} \parallel +1) = (\mathbf{h}' \parallel \mathbf{h}'') \quad \text{and} \quad (\mathbf{v} \parallel 0) = (\mathbf{v}' \parallel \mathbf{v}''),$$

where \mathbf{h}' (respectively, \mathbf{v}') denotes the w -prefix of \mathbf{h} (respectively, \mathbf{v}); notice that $\mathbf{v}' \in \mathcal{S}(\mathbf{h}'; b)$. Now, the vectors in $\mathcal{L}(\mathbf{h})$ that match \mathbf{v} are the n -prefixes of the vectors in $\mathcal{S}((\mathbf{h} \parallel +1); 0)$ that match $(\mathbf{v} \parallel 0)$. The latter vectors, in turn, take the form $(\mathbf{v}' \parallel \mathbf{y})$, where \mathbf{y} ranges over all the vectors in $\mathcal{S}(\mathbf{h}''; -b)$. The result now follows from Lemma 2. \blacksquare

Combining Theorem 3 with Lemma 1 leads to the following theorem, which is the main result of this section.

Theorem 4: Let the k -symbol clauses in the TCAM be encoded into n -symbol blocks such that each block is a codeword of a linear $[n, k, d]$ code \mathcal{C} over \mathbb{F} , and let $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_r$ be

Algorithm 1 Algorithm for locating the erroneous entries

```

1: for  $j \leftarrow 1$  to  $W/k$  do
2:   for  $i \leftarrow 1$  to  $r$  do
3:     Reset all mod-2 counters
4:     for all  $\mathbf{a} \in \mathcal{L}(\mathbf{h}_i)$  do
5:       Apply search key  $\underbrace{*** \dots ***}_{\times n(j-1)} \mathbf{a} \underbrace{*** \dots ***}_{\times (W'-nj)}$ 
                                      $\triangleright$  where  $W' = nW/k$ 
6:       Apply a clock pulse to all mod-2 counters
7:     end for
8:     Flag as erroneous all TCAM entries whose match lines
       feed "1" to the priority encoder
9:   end for
10: end for

```

the rows of a parity-check matrix of \mathcal{C} . Suppose that each block is subject to less than d errors. Then the following two conditions are equivalent for every block \mathbf{v} in the TCAM:

- (i) \mathbf{v} is error-free.
- (ii) For every $i = 1, 2, \dots, r$, the number of vectors in $\mathcal{L}(\mathbf{h}_i)$ that match \mathbf{v} is even.

C. Decoding: locating the erroneous entries

Theorem 4 serves as the basis for our strategy in locating the erroneous entries in a TCAM that was encoded according to the method described in Section IV-A, under the assumption that each block in each entry is subject to less than d errors.

Specifically, let \mathcal{C} be the $[n, k, d]$ code used to encode the blocks in each entry of the TCAM, and let $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_r$ be the rows of a parity-check matrix of \mathcal{C} . Suppose first that $W = k$, namely, that the information symbols in each TCAM entry form one clause. To identify which entries (blocks) contain errors, we apply as search keys the elements of the following r sets:

$$\mathcal{L}(\mathbf{h}_1), \mathcal{L}(\mathbf{h}_2), \dots, \mathcal{L}(\mathbf{h}_r),$$

where $\mathcal{L}(\cdot)$ is defined by (1). Assuming that each block is subject to less than d errors, it follows from Theorem 4 that an erroneous block will be identified once we see at its match line an odd number of matches for at least one such set. The generalization to k which is a proper divisor of W is rather straightforward.

More concretely, the decoding (i.e., locating the erroneous entries) is carried out as follows. We first introduce the hardware change shown in Figure 3, where we insert a mod-2 counter at the end of each match line (but before the match line encoder). The decoding itself is performed using Algorithm 1.

In order to reduce the number of search keys applied in Algorithm 1, we should aim at finding parity-check matrices that have low density, namely, the Hamming weight of each row is small. Given any linear $[n, k, d]$ code \mathcal{C} over \mathbb{F} , it is always possible to find an $(n-k) \times n$ parity-check matrix H of \mathcal{C} in which every row has Hamming weight at most $k+1$, thereby leading to an upper bound of approximately $(n-k) \cdot 2^{k+2}/3$ on the number of search keys.

The next two examples present a couple of possible choices (among others) for the code \mathcal{C} .

Example 1: Scheme using a Hamming code over \mathbb{F} . The $[13, 10, 3]$ Hamming code over \mathbb{F} has a parity-check matrix

$$\begin{pmatrix} 0 & 0 & 0 & 0 & + & + & + & + & + & + & + & + & + \\ 0 & + & + & + & 0 & 0 & 0 & + & + & + & - & - & - \\ + & 0 & + & - & 0 & + & - & 0 & + & - & 0 & + & - \end{pmatrix}$$

(where “+” and “-” stand for +1 and -1, respectively). Namely, the parity-check matrix consists of all nonzero column vectors in \mathbb{F}^3 whose first nonzero coordinate is +1.

By removing the four columns that do not contain any zero coordinates, we get the matrix

$$H = \begin{pmatrix} 0 & 0 & 0 & 0 & + & + & + & + & + \\ 0 & + & + & + & 0 & 0 & 0 & + & - \\ + & 0 & + & - & 0 & + & - & 0 & 0 \end{pmatrix},$$

which is a parity-check matrix of a linear $[n=9, k=6, d=3]$ code over \mathbb{F} . Denoting by \mathbf{h}_1 , \mathbf{h}_2 , and \mathbf{h}_3 the rows of H , each \mathbf{h}_i has Hamming weight 5 and, so, by Lemma 2,

$$|\mathcal{L}(\mathbf{h}_i)| = 22.$$

Hence, the redundancy is 3 over $k=6$ information symbols, the number of detectable errors (per block of length 9) is 2, and the number of search keys is 66. ■

Example 2: Scheme using the parity code over \mathbb{F} . Let \mathcal{C} be taken as the set of all 3^k vectors in \mathbb{F}^{k+1} whose coordinates sum to zero (in \mathbb{F}). This code is called the *parity code* and it is a linear $[n=k+1, k, 2]$ code over \mathbb{F} . For this code, a parity-check matrix is given by one row which consists of the all-one vector $\mathbf{1}$. The redundancy is 1 over k information symbols, the number of detectable errors (per block of length $k+1$) is 1, and $|\mathcal{L}(\mathbf{1})| \sim 2^{k+2}/3$. ■

When the minimum distance d is fixed (e.g., $d = 2$ as in Example 2), the parameter k defines a trade-off between the following metrics:

- 1) **Resilience:** Since the number of detectable errors per block is fixed, a poorer error rate requires using a smaller k .
- 2) **Space:** The number of check symbols per entry is proportional to the number W/k of blocks (clauses) per entry and, therefore, it reduces as k increases.
- 3) **Time:** The number of search keys that are applied during the decoding increases (exponentially) with k .

Figure 4 demonstrates the trade-off between the last two metrics, assuming that the parity code is used (i.e., $d = 2$). Since contemporary TCAMs are usually configured so that 20–35% of the cells in each entry can be allocated as check symbols, we get that setting k to 3, 4, or 5 is the most practical choice.

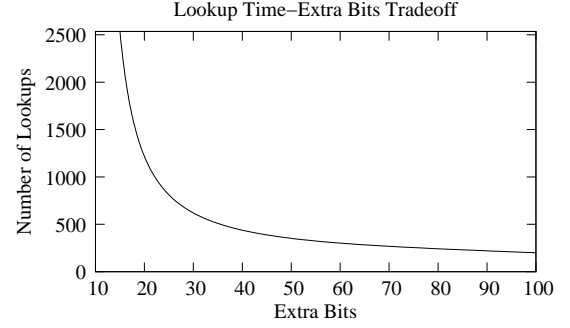


Fig. 4. Tradeoff between the number of search keys required and the number of check symbols when using the parity code, for $W = 100$ information symbols.

D. Pushing the mod-2 counters out

In this section, we consider the scenario where, in addition to a prior knowledge (or assumption) on the profile of errors *within each entry*, we can also assume an upper bound on the number of erroneous entries within each portion (of a prescribed size) of the TCAM. Under such circumstances, we can add an *external* circuit which is fed by the match lines and locates the erroneous entries, thereby eliminating the need to insert the mod-2 counters before the priority encoder. In fact, the circuit we describe here can sometimes serve also as the priority encoder itself.

Hereafter in this section, we assume that the TCAM consists of M entries, and at most t of them can be in error; the model of errors within each entry remains the same as in Sections IV-A–IV-C. (In practice, the TCAM can have many more entries than M , in which case we can divide the TCAM into disjoint portions, each consisting of M entries, and then implement the scheme presented here for each portion separately.)

In order to exhibit our technique, we refer to the inner loop of the algorithm in Algorithm 1, for some fixed values of the iterators j and i of the outer loops. We denote by \mathbb{B} the binary field $\text{GF}(2)$ (whose elements are 0 and 1 and its addition is XOR), and, for every $\mathbf{a} \in \mathcal{L}(\mathbf{h}_i)$, we let $\mathbf{y}_{\mathbf{a}}$ be the column vector in \mathbb{B}^M which is a snapshot of the match lines while the search key \mathbf{a} is applied to the TCAM; in other words, the l th coordinate in $\mathbf{y}_{\mathbf{a}}$ is 1 if and only if the value at the l th match line (before entering the mod-2 counter) is “1”.

Define now the following column vector $\mathbf{y} \in \mathbb{B}^M$:

$$\mathbf{y} = \sum_{\mathbf{a} \in \mathcal{L}(\mathbf{h}_i)} \mathbf{y}_{\mathbf{a}}, \quad (2)$$

where the sum is taken over \mathbb{B} (namely, it stands for a bitwise XOR). Clearly, \mathbf{y} is a snapshot of the output of the mod-2 counters after the last iteration of the inner loop in Algorithm 1; and, since we assume that no more than t entries are in error, it follows that the Hamming weight of \mathbf{y} is at most t .

Next, we define a “hash function” from \mathbb{B}^M to \mathbb{B}^{ρ} for some $\rho < M$ (and ρ will typically be much smaller than M), with

the property that this function is one-to-one as long as the pre-image has Hamming weight at most t . To define this function, we use again error-correcting codes. For the given parameters M and t , we select a linear $[M, K, 2t+1]$ code $\tilde{\mathcal{C}}$ over \mathbb{B} (the size of $\tilde{\mathcal{C}}$ is 2^K), and we let \tilde{H} be a $\rho \times M$ parity-check matrix of $\tilde{\mathcal{C}}$; such a matrix exists for ρ taken as the redundancy $M-K$ of $\tilde{\mathcal{C}}$. The hash function maps each column vector \mathbf{e} in \mathbb{B}^M into its syndrome $\tilde{H}\mathbf{e}$. Now, since the minimum distance of $\tilde{\mathcal{C}}$ is $2t+1$, all vectors in \mathbb{B}^M of Hamming weight t or less must have distinct syndromes: this is a fundamental property on which the decoding of linear codes is based [13, Chapter 1]. Furthermore, there are families of codes, such as BCH codes (and Hamming codes as a special case), for which the inverse mapping from syndromes to vectors of Hamming weight $\leq t$ can be computed efficiently [14, Chapter 7].

We conclude that we can recover (in fact, efficiently) the snapshot vector \mathbf{y} of the mod-2 counters, from its syndrome $\mathbf{s} = \tilde{H}\mathbf{y}$. On the other hand, by linearity, we get from (2) that \mathbf{s} can also be written as

$$\mathbf{s} = \sum_{\mathbf{a} \in \mathcal{L}(\mathbf{h}_i)} \tilde{H}\mathbf{y}_{\mathbf{a}}.$$

Namely, \mathbf{s} equals the bitwise XOR of the syndromes (i.e., the hashed values) of the match line snapshots $\mathbf{y}_{\mathbf{a}}$, where \mathbf{a} ranges over all the search keys in $\mathcal{L}(\mathbf{h}_i)$.

Thus, instead of inserting a mod-2 counter at each match line in the TCAM, we can keep one ρ -bit register \mathbf{s} , which is reset at the beginning of each iteration of the inner loop in Algorithm 1, and then, after each application of a search key $\mathbf{a} \in \mathcal{L}(\mathbf{h}_i)$ to the TCAM, the register \mathbf{s} is updated, using the hash value of the match line snapshot $\mathbf{y}_{\mathbf{a}}$, into

$$\mathbf{s} \leftarrow \mathbf{s} + \tilde{H}\mathbf{y}_{\mathbf{a}}$$

(with the sum standing for bitwise XOR). After the last iteration of the inner loop, we have in \mathbf{s} the syndrome of \mathbf{y} and, by applying a standard decoding algorithm for $\tilde{\mathcal{C}}$ to \mathbf{s} , we can recover \mathbf{y} from \mathbf{s} ; namely, we identify the erroneous TCAM entries. The hash values are computed during this process using a gate array that multiplies by the (fixed) $\rho \times M$ matrix \tilde{H} (typically, approximately half of the values in \tilde{H} will be zero).

For binary BCH codes, the value of ρ can be bounded from above by $t \lceil \log_2(M+1) \rceil$ (i.e., it is logarithmic in M). An interesting special case is that of binary Hamming codes, which correspond to BCH codes with $t = 1$ (i.e., minimum distance 3): here, the columns of \tilde{H} range over the first M nonzero column vectors in \mathbb{B}^ρ (according to the standard lexicographic ordering). In this case, the syndrome \mathbf{s} , if nonzero, points at the (unique) coordinate of \mathbf{y} which equals 1. An example for $t = 1$ and $M = 7$ is shown in Figure 5.

We remark that when each possible search key is expected to match at most t entries in the TCAM, then the circuit we have described here can also function as the TCAM priority encoder, without the need for additional hardware for the latter.

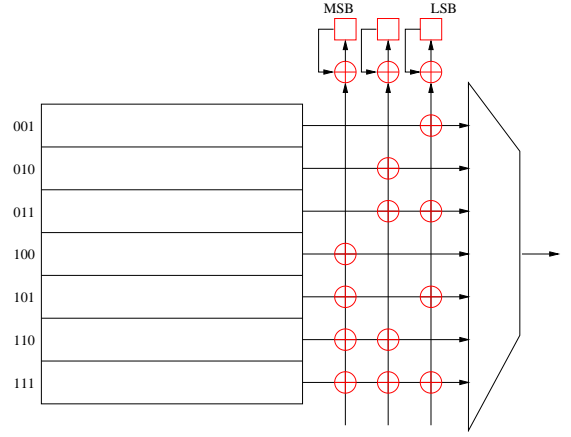


Fig. 5. Hardware change required for implementing the vertical approach for 7 entries ($m = 3$). \oplus denotes a XOR gate.

V. PEDS WITH MOD-3 COUNTERS

One obvious drawback of the scheme that was presented in Section IV is that the number of search keys that are applied in Algorithm 1 grows exponentially with k . This, in turn, limits the values of k that can be taken in practical realizations of the scheme, thereby posing restrictions on the error model within each entry: we need to prescribe an upper bound on the number of errors *within each n -symbol block*, as opposed to requiring an upper bound on the overall number of errors within a whole W' -symbol entry.

In this section, we present a parallel detection scheme in which this impediment is eliminated. In fact, the sub-division of entries into blocks will no longer be necessary, as the number of search keys will depend mildly on the raw width W of the TCAM and on the maximum number of errors that are expected in each entry; as such, the scheme we present here will be faster than the one described in Section IV. For this improvement, we will pay a (small) price though: now, we will insert mod-3 counters at each match line, instead of the mod-2 counters used earlier. A mod-3 counter is a device that is capable of storing an element of \mathbb{F} and, at each clock pulse, it gets at the input a “nonnegative” value of \mathbb{F} , which is either 0 or +1; the new contents of the counter is the sum modulo 3 of the input and the current contents.

A rather straightforward implementation of a mod-3 counter consists of a two-bit register and a *ternary semi-adder*, which adds two elements of \mathbb{F} , one of which is “nonnegative.” Such a semi-adder can be realized using eight NAND gates.

To build upon the notation that was used in Section IV, we will hereafter assume in this section that $k = W$ and $n = W'$ (namely, that a whole TCAM entry consists of one block). As we did in that section, we fix a linear $[n, k, d]$ code \mathcal{C} over \mathbb{F} , and let $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_r$ be the rows of an $r \times n$ parity-check matrix of \mathcal{C} . We will use the decoding algorithm depicted in Algorithm 1, with the following changes:

- The counters are replaced by mod-3 counters.

- The sets $\mathcal{L}(\mathbf{h}_i)$ will be replaced by multisets (i.e., sets with repetitions) $\mathcal{L}'(\mathbf{h}_i)$ to be defined below.

Notice that the choice $k = W$ makes the outer loop in Algorithm 1 redundant.

For $m = 1, 2, \dots, n$, let \mathbf{e}_m denote the unit vector in \mathbb{F}^n that has +1 in position m and is zero elsewhere (equivalently, \mathbf{e}_m is filled with “*”, except for position m , where it has “0”). Respectively, we define the multisets $\mathcal{U}_m(+1)$ and $\mathcal{U}_m(-1)$ by

$$\mathcal{U}_m(+1) = \{\mathbf{e}_m, -\mathbf{e}_m, -\mathbf{e}_m\}$$

and

$$\mathcal{U}_m(-1) = \{-\mathbf{e}_m, \mathbf{e}_m, \mathbf{e}_m\}.$$

Now, given a vector $\mathbf{h} = (h_1 \ h_2 \ \dots \ h_n) \in \mathbb{F}^n$, the multiset $\mathcal{L}'(\mathbf{h})$ is defined as the union

$$\mathcal{L}'(\mathbf{h}) = \bigcup_{m: h_m \neq 0} \mathcal{U}(h_m); \quad (3)$$

namely, if $h_m = \pm 1$, then $\pm \mathbf{e}_m$ is inserted once into $\mathcal{L}'(\mathbf{h})$ and $\mp \mathbf{e}_m$ is inserted twice. Notice that the size of $\mathcal{L}'(\mathbf{h})$ is three times the Hamming weight of \mathbf{h} (which is three times the number of proper bits—“0” and “1”—in \mathbf{h}).

The following theorem serves as a counterpart of Theorem 3 for the decoding procedure that we present in this section.

Theorem 5: Let \mathbf{h} and \mathbf{v} be row vectors in \mathbb{F}^n . The following two conditions are equivalent:

- $\mathbf{h} \cdot \mathbf{v}^T = 0$.
- The number of vectors in $\mathcal{L}'(\mathbf{h})$ that match \mathbf{v} is divisible by 3.

Proof: For $m = 1, 2, \dots, n$, denote by h_m and v_m the m th coordinate in \mathbf{h} and \mathbf{v} , respectively. In what follows, we characterize the contribution of each sub-multiset $\mathcal{U}(h_m)$ in the partition (3) of $\mathcal{L}'(\mathbf{h})$, to the number of matches in Condition (ii), for every m such that $h_m \neq 0$. It can be readily verified that the contribution of $\mathcal{U}(h_m)$ depends only on the value v_m (and not on other coordinates in \mathbf{v}), in the following manner:

- if $v_m = 0$ (namely v_m is “*”) then the contribution is $|\mathcal{U}(h_m)| = 3$;
- if $v_m \neq 0$ and $v_m = h_m$ then the contribution is 1;
- if $v_m \neq 0$ and $v_m \neq h_m$ then the contribution is 2.

In either case, the contribution is congruent modulo 3 to the product $h_m \cdot v_m$ in \mathbb{F} . This product, in turn, is the contribution of the m th coordinates in \mathbf{h} and \mathbf{v} to the scalar product $\mathbf{h} \cdot \mathbf{v}^T$, thereby leading to the desired result. ■

From Theorem 5 and Lemma 1 we get the following counterpart of Theorem 4.

Theorem 6: Let the W information symbols be encoded into a W' -symbol TCAM entry that forms a codeword of a linear $[n=W', k=W, d]$ code \mathcal{C} over \mathbb{F} , and let $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_r$ be the rows of a parity-check matrix of \mathcal{C} . Suppose that the TCAM entry is subject to less than d errors. Then the

following two conditions are equivalent for the contents \mathbf{v} of the entry:

- \mathbf{v} is error-free.
- For every $i = 1, 2, \dots, r$, the number of vectors in $\mathcal{L}'(\mathbf{h}_i)$ that match \mathbf{v} is divisible by 3.

Turning back to the Algorithm 1, if we use mod-3 counters and take the search keys from Algorithm 1 will locate all TCAM entries provided that each contains less than d errors. The total number of search keys that are now applied is

$$\sum_{i=1}^r |\mathcal{L}'(\mathbf{h}_i)| \leq 3rn = 3rW'.$$

For example, when \mathcal{C} is taken as a ternary BCH code, then r is at most $\lceil 2(d-1)/3 \rceil \cdot \lceil \log_3(W'+1) \rceil$ and, so, the overall number of search keys can be bounded from above by approximately $2dW' \log_3 W'$.

We remark that the duplication of elements in the multisets $\mathcal{U}_m(\pm 1)$ —and, hence, in $\mathcal{L}'(\mathbf{h})$ —can be avoided, thereby reducing the number of search keys to at most $2rn$. This is achieved by using a mod-3 counter that can also count backwards: the adder which is part of such a counter can be implemented using 15 NAND gates.

Finally, we mention that the technique that was presented in Section IV-D for eliminating the mod-2 counters, applies also here. The main difference is that now we will implement the hash function using a parity-check matrix of a code $\tilde{\mathcal{C}}$ over \mathbb{F} , rather than over \mathbb{B} .

VI. PRACTICAL CONSIDERATIONS

This section discusses some practical issues regarding the implementation of PEDS in contemporary real-life TCAMs.

One concern is that PEDS requires using regular TCAM lookups, and by that it might “waste” TCAM cycles. However, it is important to notice that the number of the lookups we perform is very small (that is, proportional to W and below 2000), which makes it negligible. Moreover, these lookups should not be performed one after the other, and can be done when the TCAM is idle. Specifically, each clause can be checked separately and even within each clause the lookups can be done lazily, since we save the intermediate result in the register in any case. For example, under 95% load, when $W = 100$ and $k = 3$, it will take less than 7,500 cycles—much faster than the naive approach—to complete checking the entire data, without interfering with the TCAM operations.

Another practical issue is how to forward outside the TCAM chip an indication of all erroneous entries. A simple but impractical way is to go over all the entries and check the value of registers in order to find the erroneous entries; the time complexity for this naive solution depends on the number of entries and thus is prohibitive. A better alternative is to take advantage of the ML priority encoder of the TCAM. Instead of feeding the encoder the values on the match-lines, one can feed the encoder the values of all registers and use the encoder

to find the first erroneous entries one by one: the priority encoder returns the entry number of the first entry with an error; then, the corresponding register is reset to zero, and the register values are fed again to the encoder to find the next error until no errors are found (i.e., the encoder returns no match). This process is repeated separately after the check of each clause. The hardware cost is an additional MUX for every entry that is placed before the priority encoder and allows to choose between the value of the match-line and the value of the register. The time cost is the number of errors in each clause, which is negligible in real-life situations.

VII. DISCUSSION

In this paper, we introduce *PEDS*, a parallel error detection scheme for TCAM devices. *PEDS* can use the extra symbols, available at each entry in IPv4 classification TCAM configuration, as check symbols and uses the parallel capabilities of the TCAM device in order to detect *all* erroneous entries.

PEDS is a highly-configurable scheme, which allows the selection of different trade-off points between resilience, the number of check symbols required, the number of lookup operations required to detect all erroneous entries, and the extent of required hardware changes. All properties except for the last can be configured *after the deployment* of the TCAM device according to the specific needs of the application.

Unlike prior art TCAM error detection schemes, our scheme does not change the core of TCAM, namely the structure of TCAM cells or the behavior of the TCAM search operation. Furthermore, the hardware changes required for *PEDS* is in the *peripheral circuitry* of the chip [15]. It can be viewed as a change confined to the TCAM encoder. For real-life TCAMs that are equipped with priority encoders, *PEDS* only requires the addition of a single one-bit register and a single feedback XOR gate for each match line. This translates into only eight additional transistors per match line. In the more rare case of TCAMs equipped with a *multiple-match encoder*, this encoder can be implemented according to the description provided in Section IV-D, thus eliminating the need for any additional hardware.

Acknowledgments:: The authors are deeply indebted to Cisco Systems for the grant that support this research, and would like to thank Yehuda Afek, David Belt, Michael Ben-nun, Shimon Listman and Eyal Oren of Cisco, and Martin Fabry of Netlogic Microsystems, for helpful discussions.

REFERENCES

- [1] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Comput. Surv.*, vol. 37, no. 3, pp. 238–275, 2005.
- [2] R. Mastipuram and E. C. Wee, "Soft errors' impact on system reliability," 2004, cypress Semiconductor.
- [3] M. Miller and B. Tezcan, "Investigating design criteria for searching databases," Feb. 2005. [Online]. Available: http://www.idt.com/content/solutions_InvestigatingDesignCriteriaForSearchingDatabases_wp.pdf
- [4] H. Noda *et al.*, "A cost-efficient high-performance dynamic TCAM with pipelined hierarchical searching and shift redundancy architecture," *IEEE J. Solid-State Circuits*, vol. 40, no. 1, pp. 245–253, 2005.

- [5] —, "A reliability-enhanced TCAM architecture with associated embedded PDRAM," in *IEICE Transactions on Electronics*, 2006, pp. 1612–1619.
- [6] N. Azizi and F. N. Najm, "A family of cells to reduce the soft-error-rate in ternary-CAM," in *Design automation conference*, 2006, pp. 779–784.
- [7] K. Pagiamtzis, N. Azizi, and F. N. Najm, "A soft-error tolerant content-addressable memory (CAM) using an error-correcting-match scheme," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, 2006.
- [8] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for advanced packet classification with ternary CAMs," in *ACM SIGCOMM*, 2005.
- [9] H. Liu, "Efficient mapping of range classifier into ternary-CAM," in *Hot Interconnects*, 2002.
- [10] D. Pao, P. Zhou, B. Liu, and X. Zhang, "Field domain segmentation approach to filter encoding for efficient packet classification with TCAM," in *Wireless and optical communications*, 2007.
- [11] J. van Lunteren and T. Engbersen, "Fast and scalable packet classification," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 4, pp. 560–571, May 2003.
- [12] Z. Zhang and M. Zhou, "A code-based multi-match packet classification with TCAM," in *Advances in Web and Network Technologies, and Information Management*, 2007, pp. 564–572.
- [13] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. North-Holland Mathematical Library, 1977.
- [14] R. E. Blahut, *Theory and Practice of Error Control Codes*. Reading, MA: Addison-Wesley, 1983.
- [15] M. Mohan, W. Fung, D. Wright, and M. Sachdev, "Design techniques and test methodology for low-power TCAMs," *IEEE Trans. VLSI Syst.*, vol. 14, no. 6, pp. 573–586, June 2006.

APPENDIX

While error detection schemes aim to determine whether some entry has errors or not, error correcting schemes should also correct, without any additional data, the erroneous entry itself. For example, in the context of data transmission over communication channels, having an error correcting scheme implies that there is no need to re-send the data between the sender and receiver even if error occurred during transmission. The analog scheme in TCAM devices will provide a code that, even in case of errors, will enable the device to get the same semantic results (that is, correct matches when applying search keys) as if no error occurred. This property is captured by the following definition:

Definition 1: A TCAM error-correcting code with resilience τ must ensure that for each word \mathbf{u} and search key s :

- (1) If s matches \mathbf{u} , s matches all words within Hamming distance τ of \mathbf{u} .
- (2) If s does not match \mathbf{u} , s does not match all words within Hamming distance τ of \mathbf{u} .

The following theorem, whose proof is omitted due to space constraints, shows that it is impossible to construct such a code even when $\tau = 1$. The gist of the proof follows from the fact that in order to match all words with Hamming distance 1 of a certain entry, the search key must be set to " $*** \dots **$ ". Thus, it matches all entries of the TCAM, contradicting the second requirement of Definition 1:

Theorem 7: No error-correcting code exists for TCAM devices, even if the number of errors per entry is at most 1.

We now turn to show it is necessary to add functionality to the TCAM device in order to devise an error-detection scheme.

Without any hardware change or additional functionality, the TCAM basically supports only a single operation SEARCH, which gets a search key as an input and returns the higher index of an entry which matches this search key. We next show that it is impossible to devise an error detection code using only this operation and, thus establishing that further functionality should be added to the TCAM. This usually implies an hardware change in the device is required.

Theorem 8: SEARCH operation (with a priority encoder) does not suffice to implement *error-detection scheme*.

Proof Sketch: Suppose there is an error-detection code that detects all errors by applying a search key on the TCAM database. Let $E : \mathbb{F}^W \mapsto \mathbb{F}^{W'}$ be the encoding function of the TCAM entries and $E' : \mathbb{F}^W \mapsto \mathbb{F}^{W'}$ be the encoding function of search key (note that in the general case E and E' need not be the same).

Let \mathbf{u} be an entry such that $E(\mathbf{u})$ contains at least “*” symbol, and denote by i the first occurrence of such a symbol. Assume on the way of contradiction, that any error in the i -th bit of entry \mathbf{u} can be detected using a single search key s' (of length W'). In order to detect such an error the i -th bit of s' must be in {“0”, “1”}, otherwise no comparison to the i -th bit $E(\mathbf{u})$ is made. If it “0” then a change from “*” to “1” in $E(\mathbf{u})$ will not be detected by s' . On the other hand, if it is “1”, a change from “*” to “0” in $E(\mathbf{u})$ will not be detected by s' .

This implies that in order to detect an error which turn a “*” symbol into a symbol in {“0”, “1”}, at least two search keys s'_1, s'_2 must be applied, and their result should be taken into account. Since applying s'_1 or s'_2 separately on $E(\mathbf{u})$ does not imply there is an error in $E(\mathbf{u})$, these keys may match other error-free entries as well. If the matched entries have higher priority than \mathbf{u} , it implies that the operations of s'_1, s'_2 with respect to \mathbf{u} are masked.

Hence, either the TCAM should be equipped with some sort of a *multi-match encoder* or should be designed to support additional operations (other than SEARCH). ■