PEDS: A Parallel Error Detection Scheme for TCAM Devices

Anat Bremler-Barr, *Member, IEEE,* David Hay, *Member, IEEE,* Danny Hendler and Ron M. Roth, *Fellow, IEEE*

***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-detecting 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 in a manner that is typically orders of magnitude smaller than the 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.

***Index Terms*—TCAM, Error Detection, Packet Classification, Error-Correcting Codes, Error-Detecting Codes**

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 *rule*s (sometimes called *filter*s). Each such rule specifies a certain pattern, which is based on packet header fields, such as the source/destination addresses, source/destination port numbers and 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 line rate of the routers. Furthermore, recent initiatives, such as *OpenFlow* [1], call for simplifying the data-paths of contemporary routers, by making them consist mainly of packet classification operations, through the use of flexible definitions of *flow*s [2]. It is therefore important to design packet classification solutions that scale to millions of key lookup 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 $\{\text{“}\texttt{0}\text{”}, \text{“}\texttt{1}\text{”}, \text{“}\texttt{*}\text{”}\}$, where $\{\text{“}\texttt{0}\text{”}, \text{“}\texttt{1}\text{”}\}$ are *proper bit* values and “$\texttt{*}$” 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* [3]. 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 $\mathbf{u}$ matches entry $\mathbf{v}$, if the proper bits of $\mathbf{u}$ agree with those of $\mathbf{v}$. 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 are matched by 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 [4]. In a TCAM error event, a TCAM symbol can change its value to any symbol in $\{\text{“}\texttt{0}\text{”}, \text{“}\texttt{1}\text{”}, \text{“}\texttt{*}\text{”}\}$.

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 lookups. Furthermore, TCAM memory consists of an array of SRAM cells, which

Patent pending. Supported by a research grant from Cisco Systems, Inc.

A. Bremler-Barr is with the Efi Arazi School of Computer Science, The Interdisciplinary Center, Herzliya, Israel (email: bremler@idc.ac.il).

D. Hay is with the Electrical Engineering Department, Columbia University, NY, USA (email: hdavid@ee.columbia.edu). This work was while he was a postdoc fellow in the Computer Science Department at Ben-Gurion University, Israel.

D. Hendler is with the Computer Science Department, Ben-Gurion University, Be'er Sheva, Israel (email: hendlerd@cs.bgu.ac.il).

R. M. Roth is with the Computer Science Department,Technion - Israel Institute of Technology, Haifa, Israel (email: ronny@cs.technion.ac.il).

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-detecting or error-correcting 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 is matched by 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.

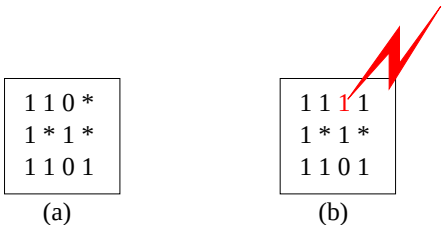

Fig. 1. TCAM memory with W=4, before (a) and after (b) an error event.

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 the scheme is that of applying a sequence of TCAM lookups 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 be to consecutive and can be performed lazily during idle TCAM cycles. As an example, in a TCAM that can perform 100 million searches (that is, lookups) per second (MSPS) (e.g., [5]) which is 99% loaded, PEDS needs only 0.2 milliseconds to detect all errors, using the scheme we describe in Section VI.

For error correction, 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 (flip flops) 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 V

and VI we describe two specific instances of PEDS that result in different performance trade-offs. Then, in Section VII, we show that some change in the hardware of the TCAM is unavoidable if one is to implement an error detection scheme with time complexity that does not grow with the number of TCAM entries. Finally, practical considerations and concluding remarks are given in Sections VIII and IX, respectively.

II. Related Work

State of the art TCAM devices contain additional per-entry state to store the extra symbols required for error detection. Dedicated TCAM circuitry periodically reads TCAM entries one by one to verify correctness. A disadvantage of this simple scheme is that the time to identify an error grows linearly in the number of TCAM *entries* (as opposed to our scheme, where this time grows linearly only in the *width* of TCAM entries). This is problematic since the elapsed time in between two checks of the same entry could be too long compared to the rate with which errors may occur. In other words, multiple lookup operations may return erroneous values before an error is detected and fixed, which may be unacceptable for some applications. Moreover, contemporary TCAM devices use at least $W-1$ XOR gates and one register per entry for storing error detection symbols. As we show, PEDS can make due with just a single XOR gate and a single register-bit per entry, which results in smaller die size. If extra symbols are available, they can be used by PEDS for improved resiliency.

Prior art suggests several additional methods of coping with TCAM errors. All these methods focus on reducing the error 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.* [6] 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.* [7], 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 [8] 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; this state is made unstable, 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.

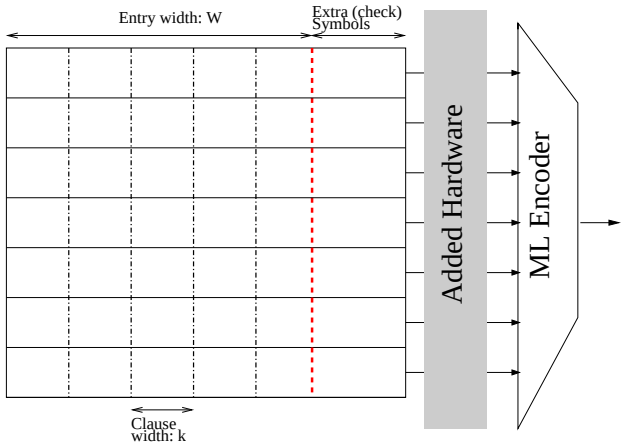

Fig. 2. High-level PEDS architecture. For presentation simplicity, clauses are shown to consist of consecutive symbols. As explained in Section III-B, symbols are actually interleaved across clauses.

We also mention the work of Pagiamtzis *et al.* [9], 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 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 [10]–[14]. 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). First, we explain, in Section III-A, the basic idea underlying PEDS, using a simple example. Then, the general framework is presented in Section III-B.

A. Basic Idea Underlying PEDS

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

$$\left(\texttt{*}^{2(j-1)}\right)\texttt{01}\left(\texttt{*}^{2W-2j}\right) \quad \text{and} \quad \left(\texttt{*}^{2(j-1)}\right)\texttt{10}\left(\texttt{*}^{2W-2j}\right),$$

for every $j \in \{1, 2, \ldots, W\}$.

If neither of the search keys matches a given entry, then this entry is correct since the j-th pair is either 00 or 11. The entry is correct also if it is matched by both search keys, since the j-th pair has to be **. On the other hand, if only one search key matches the entry then the entry is necessarily incorrect: if only $\left(\texttt{*}^{2(j-1)}\right)\texttt{01}\left(\texttt{*}^{2W-2j}\right)$ matches the entry, the j-th pair is either 01, 0*, or *1; and if only $\left(\texttt{*}^{2(j-1)}\right)\texttt{10}\left(\texttt{*}^{2W-2j}\right)$ 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 occurring in the same pair. The code requires W extra symbols per entry, and the time to detect all errors is $2W$ lookups. In addition, in order to distinguish between odd and even number of matches, it suffices to add just a modulo-2 counter at the end of each match-line.

Note that PEDS uses search keys that contain * symbols; this is supported by TCAM devices [15].

B. PEDS architecture

This section describes the high-level architecture of PEDS, which is illustrated in Figure 2. Each entry of size W is partitioned into k-symbol clauses such that the i-th clause contains all the symbols whose indices modulo k equal i. This strategy—of interleaving the symbols across clauses—addresses cases where the error events at adjacent symbols may be statistically dependent.

For each such clause, we compute check symbols according to a prescribed error-detecting code over the ternary alphabet and store them in the positions allocated for the extra symbols. In most of the examples that we present in this paper, the code requires only a single check symbol per clause, but generalizations to other error-detecting codes are also possible, as described in Section V. 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 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 V, we fix T to be the set of even integers, thus an entry is error-free if and only if the number of search keys that match it is even. This, in turn, requires adding just a modulo-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 resilience increases, the time for checking all entries decreases but more extra symbols are needed.

Section VI 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 modulo-3 adders (rather than binary modulo-2 counters) should be implemented at the end of each match-line.

IV. Preliminaries

In the sequel, we use the following basic concepts from the theory of error-correcting codes. For further details and thorough explanations, the reader is referred to [16]–[18].

Definition 1 (Linear Code): A *linear* $[n, k, d]$ *code* $\mathcal{C}$ over a finite field Ψ is a set of $|\Psi|^k$ vectors (referred to as *codewords*) of length n over Ψ that form a linear space of dimension k over Ψ, 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). The minimum distance d of a linear code also equals the smallest Hamming weight of (namely, the number of nonzero entries in) any nonzero codeword of $\mathcal{C}$ [16, p. 28]. A *coder*[1] for $\mathcal{C}$ is any one-to-one mapping from Ψ^k into $\mathcal{C}$ and, without real loss of generality of $\mathcal{C}$, we can assume that the coding of a vector $\mathbf{u} \in \Psi^k$ 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 $\mathcal{C}$. ∎

In the sequel, we use two finite fields, $\mathrm{GF}(2)$ and $\mathrm{GF}(3)$, which we denote hereafter by $\mathbb{B}$ and $\mathbb{F}$, respectively. The field $\mathbb{B}$ consists of two elements 0 and 1, where addition and multiplication are XOR (namely, addition modulo 2) and AND operations, respectively. The field $\mathbb{F}$ consists of the three elements $+1$, -1, and 0, with addition and multiplication taken modulo 3.

Perhaps the most commonly used linear code is the *parity code*. The parity code over a field Ψ is a linear $[k+1, k, 2]$ code which consists of $|\Psi|^k$ vectors in Ψ^{k+1} whose coordinates sum to zero (in Ψ). For each vector $\mathbf{u}$ in Ψ^k, the coder appends a single symbol which equals the additive inverse of the sum (in Ψ) of the k coordinates of $\mathbf{u}$. Thus, the redundancy of the code is 1. For the special case where $\Psi = \mathbb{B}$, the appended bit equals the sum modulo 2 (namely, XOR) of the coordinates of $\mathbf{u}$.

Definition 2 (Parity-check Matrix): Let $\mathcal{C}$ be a linear $[n, k, d]$ code over some field Ψ. The *parity-check matrix* H

[1]This term is called *encoder* in the theory of error-correcting codes; yet, we use the term coder to avoid confusion with the match-line (priority) encoder of the TCAM devices.

of code $\mathcal{C}$ is an $r \times n$ matrix over Ψ with the property that $\mathcal{C}$ forms its right kernel, namely:

$$\mathcal{C} = \left\{ \mathbf{v} \in \Psi^n \, : \, H\mathbf{v}^T = \mathbf{0} \right\} \, ;$$

here, $(\cdot)^T$ denotes transposition, the product $H\mathbf{v}^T$ is carried out over the field Ψ, and $\mathbf{0}$ stands for the all-zero vector. ∎

Notice that we can associate a parity-check matrix with any linear code $\mathcal{C}$. Its number of rows r must be at least the redundancy $n-k$ of $\mathcal{C}$, and equality can be attained when H is selected to have linearly independent rows over Ψ. In fact, there are many parity-check matrices for any given code.

For the parity code, whose redundancy is 1, a parity-check matrix is the all-one vector of length $k+1$. Alternatively, the $2 \times (k+1)$ all-one matrix is also a parity-check matrix for this code.

As another example, consider the following 3×9 matrix over the ternary field $\mathbb{F}$:

$$H = \begin{pmatrix} 0 & 0 & 0 & 0 & +1 & +1 & +1 & +1 & +1 \\ 0 & +1 & +1 & +1 & 0 & 0 & 0 & +1 & -1 \\ +1 & 0 & +1 & -1 & 0 & +1 & -1 & 0 & 0 \end{pmatrix} . \tag{1}$$

This is a parity-check matrix of a linear $[n{=}9, k{=}6, d{=}3]$ code over $\mathbb{F}$.

Definition 3 (Syndrome): Given an $r \times n$ parity-check matrix H of a linear $[n, k, d]$ code $\mathcal{C}$ over Ψ and a vector $\mathbf{v} \in \Psi^n$, the *syndrome* $\mathbf{s}$ of vector $\mathbf{v}$ is the following column vector in Ψ^r:

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

Thus, $\mathbf{s} = \mathbf{0}$ if and only if $\mathbf{v} \in \mathcal{C}$. ∎

For example, with respect to the parity-check matrix H in (1), the syndrome of $\mathbf{v}_1 = (-1, 0, 0, 0, 0, 0, 0, 0, +1)$ is $(+1, -1, -1)^T \neq \mathbf{0}$ and, therefore, $\mathbf{v}_1$ is not a codeword of the respective code. On the other hand, the syndrome of $\mathbf{v}_2 = (+1, +1, +1, +1, 0, +1, +1, +1)$ is $(0, 0, 0)^T = \mathbf{0}$, implying that $\mathbf{v}_2$ is a codeword.

Notice that with a linear $[n, k, d]$ code, we can always detect any pattern of less than d errors in a vector $\mathbf{v}$. Under this assumption on the number of errors, the syndrome of $\mathbf{v}$ will be zero if and only if no errors have occurred (see, for example, [16, p. 14]). We summarize this fact in the following lemma:

Lemma 1: Let $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_r$ be the rows of an $r \times n$ parity-check matrix of a linear $[n, k, d]$ code $\mathcal{C}$ over Ψ, and suppose that $\mathbf{v} \in \Psi^n$ is the result of less than d error occurring in a codeword of $\mathcal{C}$. The following two conditions are equivalent:

(i) $\mathbf{v}$ is error-free.

(ii) $\mathbf{h}_i \cdot \mathbf{v}^T = 0$ for every $i = 1, 2, \ldots, r$.[2] ∎

It is also worth mentioning that any two distinct vectors of Hamming weight at most $(d-1)/2$ must have distinct syndromes: this is a fundamental property on which the decoding of linear codes is based [16, p. 12].

[2] $\mathbf{h}_i \cdot \mathbf{v}^T$ is the scalar product of $\mathbf{h}_i$ and $\mathbf{v}^T$ computed over Ψ.

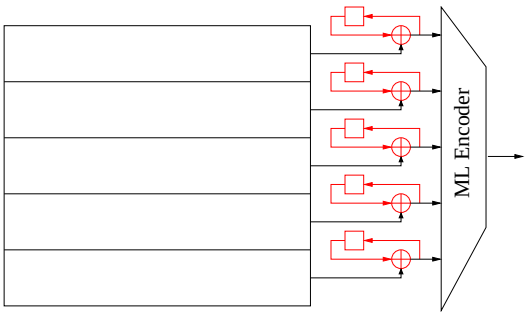

Fig. 3. Hardware change required for implementing the fast detection scheme described in Section V. Each $\oplus$ stands for a XOR gate and each square box is a single-bit delay flip-flop.

V. PEDS WITH MODULO-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 hardware change: adding a *modulo-2 counter* (that is, a XOR gate and a single-bit flip flop) 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. Coding 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 ternary field $\mathbb{F}$ (see Section IV). We will use the following mapping between TCAM symbols and elements of $\mathbb{F}$:

$$\begin{array}{rcl} \text{“}\texttt{*}\text{”} & \longleftrightarrow & 0 \\ \text{“}\texttt{0}\text{”} & \longleftrightarrow & +1 \\ \text{“}\texttt{1}\text{”} & \longleftrightarrow & -1 \, . \end{array}$$

Under this mapping, each word of a given length ℓ over $\{\text{“}\texttt{0}\text{”}, \text{“}\texttt{1}\text{”}, \text{“}\texttt{*}\text{”}\}$ will be is seen as a row vector in $\mathbb{F}^\ell$. When no confusion arises, we will interchangeably use both symbol sets—$\{\text{“}\texttt{0}\text{”}, \text{“}\texttt{1}\text{”}, \text{“}\texttt{*}\text{”}\}$ 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, 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 $\mathbb{F}$: 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 "$\star$" (the latter being represented by the element 0 of $\mathbb{F}$).

Now, in every TCAM entry, we code 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 coded as we have just described at the end of Section V-A. We start 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 $\mathbf{u} = (u_1\, u_2\, \ldots\, u_n)$ 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 "$\star$" in $\mathbf{u}$. Thus, two row vectors $\mathbf{u} = (u_1\, u_2\, \ldots\, u_n)$ and $\mathbf{v} = (v_1\, v_2\, \ldots\, v_n)$ in $\mathbb{F}^n$ are said to *match* if they are equal on the intersection of their supports, namely,

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

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

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

Namely, $\mathcal{S}(\mathbf{h}; b)$ consists of all vectors in $\mathbb{F}^n$ that have nonzero coordinates (that is, coordinates which are not "$\star$") 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 $\mathcal{L}(\mathbf{h})$:

$$\mathcal{L}(\mathbf{h}) = \mathcal{S}(\mathbf{h}; +1) \cup \mathcal{S}(\mathbf{h}; -1)\ . \tag{2}$$

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

(b) $|\mathcal{S}(\mathbf{h}; \pm 1)| = \dfrac{1}{3}\left(2^w - (-1)^w\right)$ (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 $\mathbf{h} = (\mathbf{h}'\|\pm 1)$, where the Hamming weight of $\mathbf{h}'$ is $w-1$ (> 0). We have

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

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

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

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,

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

as claimed. ∎

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

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

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 $\mathcal{S}((\mathbf{h}\|{+1}); 0)$ that match $(\mathbf{v}\|0)$. The latter vectors, in turn, take the form $(\mathbf{v}'\|\mathbf{y})$, where $\mathbf{y}$ ranges over all the vectors in $\mathcal{S}(\mathbf{h}''; -b)$. The result now follows from Lemma 2. ∎

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 coded 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 $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_r$ 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) For every $i = 1, 2, \ldots, r$, the number of vectors in $\mathcal{L}(\mathbf{h}_i)$ that match $\mathbf{v}$ is even.

Algorithm 1 Algorithm for locating the erroneous entries

```
for j ← 1 to W/k do
    for i ← 1 to r do
        Reset all modulo-2 counters
        for all a ∈ L(h_i) do
            Apply search key ***...** a ***...**
                             (× n(j−1))   (× (W′−nj))
                                          ▷ where W′ = nW/k
            Apply a clock pulse to all modulo-2 counters
        end for
        Flag as erroneous all TCAM entries whose match-lines
            feed "1" to the priority encoder
    end for
end for
```

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 coded according to the method described in Section V-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 code the blocks in each entry of the TCAM, and let $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_r$ 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:

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

where $\mathcal{L}(\cdot)$ is defined by (2). 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 modulo-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 the same as (1) and 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 the $[k+1, k, 2]$ parity 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 $|\mathcal{L}(\mathbf{1})| \sim 2^{k+2}/3$. Specifically, for $k = 3$,

$$\begin{aligned} \mathcal{L}(\mathbf{1}) \ = \ & \{\texttt{0000}, \texttt{1111}, \texttt{0111}, \texttt{1000}, \texttt{1011}, \\ & \texttt{0100}, \texttt{1101}, \texttt{0010}, \texttt{1110}, \texttt{0001}\}, \end{aligned}$$

where the symbol notation "0" and "1" is used instead of $+1$ and -1. ∎

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 (i.e., higher) 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 (corresponding to 34, 25 and 20 extra symbols, respectively) is the most practical choice.

D. Pushing the modulo-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

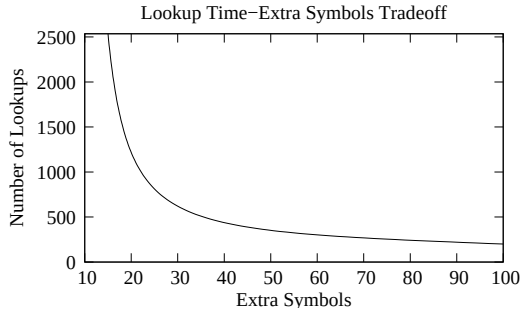

Fig. 4. Tradeoff between the number of search keys required and the number of check symbols when using the parity code, for $W = 100$ information symbols.

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 modulo-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 V-A – V-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. Recall that $\mathbb{B}$ denotes the binary field $\mathrm{GF}(2)$. For every $\mathbf{a} \in \mathcal{L}(\mathbf{h}_i)$, let $\mathbf{y_a}$ 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 ℓth coordinate in $\mathbf{y_a}$ is 1 if and only if the value at the ℓth match-line (before entering the modulo-2 counter) is "1".

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

$$\mathbf{y} = \sum_{\mathbf{a} \in \mathcal{L}(\mathbf{h}_i)} \mathbf{y_a} \ , \tag{3}$$

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 modulo-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 $\mathbb{B}^\rho$ 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 linear 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 $\tilde{\mathcal{C}}$; recall that 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. 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 [16, Chapters 5–6], [18, Chapter 7].

We conclude that we can recover (in fact, efficiently) the snapshot vector $\mathbf{y}$ of the modulo-2 counters, from its syndrome $\mathbf{s} = \tilde{H}\mathbf{y}$. On the other hand, by linearity, we get from (3) that $\mathbf{s}$ can also be written as

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

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

Thus, instead of inserting a modulo-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 $\mathbf{y_a}$, into

$$\mathbf{s} \leftarrow \mathbf{s} + \tilde{H}\mathbf{y_a}$$

(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 $\mathbb{B}^\rho$ (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.

VI. PEDS WITH MODULO-3 COUNTERS

One obvious drawback of the scheme that was presented in Section V 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

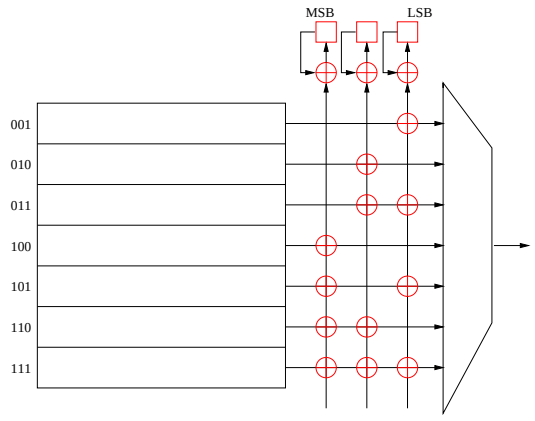

Fig. 5. The external circuit used to implement PEDS for seven entries and a single error. Each $\oplus$ stands for a XOR gate and each square box is a single-bit delay flip-flop.

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 V. For this improvement, we will pay a (small) price though: now, we will insert modulo-3 counters at each match-line, instead of the modulo-2 counters used earlier. A modulo-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 modulo-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 V, 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 $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_r$ 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 modulo-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 $m = 1, 2, \ldots, n$, 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 $\mathbf{h} = (h_1\ h_2\ \ldots\ h_n) \in \mathbb{F}^n$, the multiset $\mathcal{L}'(\mathbf{h})$ is defined as the union

$$\mathcal{L}'(\mathbf{h}) = \bigcup_{m\,:\,h_m \neq 0} \mathcal{U}_m(h_m)\ ; \tag{4}$$

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:

(i) $\mathbf{h} \cdot \mathbf{v}^T = 0$.

(ii) The number of vectors in $\mathcal{L}'(\mathbf{h})$ that match $\mathbf{v}$ is divisible by 3.

Proof: For $m = 1, 2, \ldots, n$, denote by h_m and v_m the mth coordinate in $\mathbf{h}$ and $\mathbf{v}$, respectively. In what follows, we characterize the contribution of each sub-multiset $\mathcal{U}_m(h_m)$ in the partition (4) 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 $\mathcal{U}_m(h_m)$ 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 $|\mathcal{U}_m(h_m)| = 3$;
- 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 mth 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 coded 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 $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_r$ 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:

(i) $\mathbf{v}$ is error-free.

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

If we use modulo-3 counters and take the search keys from the sets $\mathcal{L}'(\mathbf{h}_i)$, then 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 $\lceil 2(d{-}1)/3 \rceil \cdot \lceil \log_3(W'{+}1) \rceil$ (see [16, p. 174, Problem 5.26]); 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 modulo-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 V-D for

eliminating the modulo-2 counters, applies also here (using a parity-check matrix of some code over $\mathbb{F}$).

VII. The Necessity of Additional Hardware for Error Detection

Without any hardware change or additional functionality, and assuming that a priority encoder is used, the TCAM basically supports only the LOOKUP operation, which gets a search key as an input and returns the highest index of an entry (if any) that is matched by this search key.

More formally, denote by M the number of entries, let W be the number of symbol per entry before coding (namely, *raw symbols*), and let W' be the number of symbol after coding (namely, *coded symbols*). We assume that an *output-preserving coding scheme* is used; such a scheme comprises two coding functions: a function $E_{\text{TCAM}} : \left(\mathbb{F}^W\right)^M \rightarrow \left(\mathbb{F}^{W'}\right)^M$ for coding the raw database into the TCAM entries, and a function $E_{\text{KEYS}} : \mathbb{F}^W \rightarrow \mathbb{F}^{W'}$ for coding each search key before it is applied to the coded TCAM. These two functions satisfy the property that for every raw TCAM database $D \in \left(\mathbb{F}^W\right)^M$ and every search key $\mathbf{u} \in \mathbb{F}^W$, the output of applying $\mathbf{u}$ to D (that is, the highest priority entry index) is the same as the output of applying $E_{\text{KEYS}}(\mathbf{u})$ to $E_{\text{TCAM}}(D)$.

The next theorem presents a limitation on any output-preserving error detection scheme (for detecting even one single error in the whole TCAM), if no additional functionality is added to the TCAM.

Theorem 7: If the TCAM is equipped only with a LOOKUP operation (with a priority encoder) and uses an output-preserving coding scheme, then the number of applications of LOOKUP required to guarantee successful detection of a single error in the TCAM is at least $\min\{M, 2^{W-1}\}$.

Proof: Let $M' = \min\{M, 2^{W-1}\}$. Consider a raw contents of the TCAM where for every entry index $i \in \{0, 1, \ldots, M'-1\}$ there are at least two search keys which, when applied to the TCAM, produce the index i at the output of the priority encoder. Indeed, the condition that $M' \leq 2^{W-1}$ allows us to assume such a contents, say, by filling entry i with the $(W-1)$-bit binary representation of i, followed by "`*`". Since the coding scheme is output-preserving, it implies that each coded entry must be matched by at least two distinct (coded) search keys and therefore must contain at least one "`*`".

Suppose that there is an error detection scheme that operates by applying a subset $\mathcal{L} \subseteq \mathbb{F}^{W'}$ of (coded) search keys to the coded TCAM, and assume (by contradiction) that $|\mathcal{L}| \leq M'-1$. Consider the sequence of indices that is generated at the output of the priority encoder when the search keys in $\mathcal{L}$ are applied to an error-free coded TCAM. Since $|\mathcal{L}| \leq M'-1$, there must be at least one index, say i, which does not appear in that sequence. Denote by $\mathbf{v}$ the contents of the entry with index i in the coded TCAM.

Now consider the event where one "`*`" in that entry is changed into (say) "`0`", thereby producing a word $\mathbf{v}'$. We next show that this event will not be detected. Let $\mathbf{a}$ be an element of $\mathcal{L}$, and distinguish between the following two cases regarding the output of the priority encoder when $\mathbf{a}$ is applied to the error-free (coded) TCAM.

1) The priority encoder returns an index $j < i$ (namely the returned index corresponds to an entry with a priority that is higher than i). Since all entries with index j or less are not affected by the error, the same index j will be returned for w also in the erroneous TCAM.
2) The priority encoder returns an index $j > i$. This, in turn, implies that $\mathbf{a}$ does not match $\mathbf{v}$ and, thus, it does not match $\mathbf{v}'$. Since all entries with index larger than i are not affected by the error, the index j will be returned for $\mathbf{a}$ also in the erroneous TCAM.

Thus, in either case, the output of the priority encoder will be identical to the output obtained for an error-free coded TCAM, which means that the erroneous TCAM will be indistinguishable from the error-free one. ∎

Theorem 7 implies that further functionality must be added to the TCAM in any error detection scheme that uses a small number of lookups. This usually implies that a hardware change in the device is required. Indeed, PEDS, which uses a number of lookups that is typically orders of magnitude smaller than $\min\{M, 2^{W-1}\}$, requires adding counters at the end of each match-line.

Next we discuss the necessity of separating the error detection process and the regular operation of the TCAM device. Specifically, we prove the non-existence of *on-access error correcting schemes* which, even in the presence of a prescribed number of errors, should enable the device to get the same output (at the priority encoder) as if no error has occurred. This impossibility result is a direct consequence of the following simple proposition:

Proposition 8: Given any contents of the TCAM, among all search keys that match at least one TCAM entry, there is one and only one search key—the all-"`*`" word—for which the output of the priority encoder remains the same even when any one of the symbols in the TCAM is changed.

Proof: Obviously, when the all-"`*`" word is applied as a search key, the output of the priority encoder will always be the highest-priority index, regardless of the contents of the TCAM.

On the other hand, let $\mathbf{u}$ be a search key, other than the all-"`*`" word, which matches at least one entry in an error-free (and possible coded) TCAM, and let i be the index that is produced at the output of the priority encoder upon application of $\mathbf{u}$ to that TCAM. Without loss of generality, assume that it is the first position in $\mathbf{u}$ which is not "`*`" and, further, that this position contains "`0`". The first position in the entry that is indexed by i must therefore contain "`0`" too. Yet an error event which changes that "`0`" into "`1`" will result in a different output at the priority encoder, since then $\mathbf{u}$ will no longer match the entry at index i. ∎

Proposition 8 implies that, regardless of which mapping

is used to code data into the TCAM, for every search key $\mathbf{u} \neq$ "$**\ldots**$" that matches at least one of the entries of the TCAM, there is always an error event where one symbol in the TCAM is changed (at the highest-priority entry that is matched by $\mathbf{u}$) and, as a result, the priority encoder produces the wrong output. This, in turn, implies that, unless some hardware change is made, no on-access error correcting scheme exists, even if it is required to handle only a single error in the entire TCAM. (Recall that the on-access error correcting scheme that is proposed in [9] does also require a modification of the ML sensing scheme.)

VIII. Practical Considerations

This section discusses some practical issues concerning the implementation of PEDS in contemporary TCAM devices.

PEDS uses the regular TCAM lookup mechanism for error detection, hence cycles used by PEDS cannot be used for application lookups. Since PEDS lookup operations do not have to be consecutive, however, PEDS can use idle cycles and still check all entries very quickly. As an example, in a TCAM that can perform 100 million lookups per second (e.g., [5]) which is 99% loaded, PEDS needs only 0.2 milliseconds to detect all errors, using the scheme we describe in Section VI.

Once PEDS identifies erroneous entries, the TCAM device should output their indexes. The TCAM's ML priority encoder can be used to accomplish this efficiently. Once errors are detected, the encoder can be fed with the values of the per-entry flip flops used by PEDS to locate erroneous entries and output their indexes one by one. After the index of an erroneous entry is being output, the value of the flip flop at that entry is set to 0. This process is repeated until all erroneous entry indexes are reported. The cost of implementing this scheme in hardware is an additional multiplexer per entry which is placed before the priority encoder. The multiplexer allows to select which value is fed to the priority encoder: either the match-line value (in regular operation) or the value of the PEDS flip flop (when errors are reported).

IX. Discussion

In this paper, we introduce *PEDS*, a parallel error detection scheme for TCAM devices. PEDS adds extra symbols per entry at positions which are usually available in current applications (e.g., in IPv4 classification TCAM configuration), 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. Moreover, PEDS does not depend on the particular information contents that is coded into the TCAM entries, thus it can be combined with any TCAM optimization algorithms (e.g. rule compaction or range–rule representation) and non-conventional usage of the device (e.g., for flow classification or deep packet inspection).

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 lookup operation. Furthermore, the hardware changes required for PEDS is in the *peripheral circuitry* of the chip [19]. 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 flip flop and a single feedback XOR gate (totaling to 8 transistors) per match-line.

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

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[2] N. McKeown, "Software-defined networking," Infocom Keynote Talk, April 2009. [Online]. Available: http://tiny-tera.stanford.edu/~nickm/talks/infocom_brazil_2009_v1-1.pdf

[3] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Comput. Surv.*, vol. 37, no. 3, pp. 238–275, 2005.

[4] R. Mastipuram and E. C. Wee, "Soft errors' impact on system reliability," 2004, cypress Semiconductor.

[5] M. Miller and B. Tezcan, "Investigating design criteria for searching databases," Feb. 2005. [Online]. Available: http://www.idt.com/content/solutions_InvestigatingDesignCriteriaForSearchingDatabases_wp.pdf

[6] H. Noda, K. Inoue, M. Kuroiwa, F. Igaue, K. Yamamoto, H. J. Mattausch, T. Koide, A. Amo, A. Hachisuka, S. Soeda, I. Hayashi, F. Morishita, K. Dosaka, K. Arimoto, K. Fujishima, K. Anami, and T. Yoshihara, "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.

[7] H. Noda, K. Dosaka, H. Mattausch, T. Koide, F. Morishita, and K. Arimoto, "A reliability-enhanced TCAM architecture with associated embedded DRAM and ECC," vol. 89, no. 11, pp. 1612–1619, 2006.

[8] 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.

[9] 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.

[10] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for advanced packet classification with ternary CAMs," in *ACM SIGCOMM*, 2005.

[11] H. Liu, "Efficient mapping of range classifier into ternary-CAM," in *Hot Interconnects*, 2002.

[12] 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.

[13] J. van Lunteren and T. Engbersen, "Fast and scalable packet classification," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 4, pp. 560–571, 2003.

[14] 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.

[15] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, Mar. 2006.

[16] R. M. Roth, *Introduction to Coding Theory*. Cambridge University Press, 2006.

[17] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. North-Holland Mathematical Library, 1977.

[18] R. E. Blahut, *Theory and Practice of Error Control Codes*. Addison-Wesley, 1983.

[19] N. 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.

PLACE PHOTO HERE

Anat Bremler-Barr Anat Bremler-Barr holds B.Sc. degrees in mathematics and computer science (Magna Cum Laude), an LL.B degree in law, and M.Sc (Summa Cum Laude) and P.hD degree (with distinction) in computer science, all from Tel Aviv University. In 2001 Anat co-founded Riverhead Networks Inc., a company that provides systems to protect from Denial of Service attacks. She was the chief scientist of the company. The company was acquired by Cisco Systems in 2004. She joined the school of computer science at the Interdisciplinary Center, Herzliya in 2004. Anat research interests are in computer networks and distributed computing. Her current works are focused on Designing routers and protocols that support efficient and reliable communication.

PLACE PHOTO HERE

David Hay David Hay received his BA (summa cum laude) and PhD degree in computer science from the Technion - Israel Institute of Technology in 2001 and 2007, respectively. He is currently a post-doctoral research scientist at the department of electrical engineering, Columbia University, NY, USA. His main research interests are algorithmic aspects of high-performance switches and routers; in particular: QoS provisioning, competitive analysis and packet classification. Between 1999-2002, Dr. Hay was with IBM Haifa Research Labs. During summer 2006, he was interning at the Data Center Business Unit of Cisco Systems, San Jose. He was also a post-doc fellow at the department of computer science, Ben Gurion University of the Negev, Israel (2007-2008), and at the department of electrical engineering, Politecnico di Torino, Italy (2008-2009).

PLACE PHOTO HERE

Danny Hendler Danny Hendler received his B.Sc. degree in Mathematics and Computer Science in 1986, and his M.Sc. and Ph.D. degrees in Computer Science in 1993 and 2004, respectively, all from Tel-Aviv University.

Dr. Hendler returned to Academy in 2001 after working for 18 years in the Israeli high-tech industry. He is presently an assistant professor at the department of Computer Science at Ben-Gurion University of the Negev. Before joining Ben-Gurion University, he has been a post-doctoral research associate at the University of Toronto and at the Technion, Israel Institute of Technology. His main research interests are distributed and parallel computation and packet classification.

PLACE PHOTO HERE

Ron M. Roth Ron M. Roth received the B.Sc. degree in computer engineering, the M.Sc. in electrical engineering and the D.Sc. in computer science from Technion—Israel Institute of Technology, Haifa, Israel, in 1980, 1984 and 1988, respectively. Since 1988 he has been with the Computer Science Department at Technion, where he now holds the General Yaakov Dori Chair in Engineering. During the academic years 1989–91 he was a Visiting Scientist at IBM Research Division, Almaden Research Center, San Jose, California, and during 1996–97 and 2004–05 he was on sabbatical leave at Hewlett–Packard Laboratories, Palo Alto, California. He is the author of the book *Introduction to Coding Theory*, published by Cambridge University Press in 2006. Dr. Roth was an associate editor for coding theory in IEEE TRANSACTIONS ON INFORMATION THEORY from 1998 till 2001, and he is now serving as an associate editor in *SIAM Journal on Discrete Mathematics*. His research interests include coding theory, information theory, and their application to the theory of complexity.