PC-TRIO: An Indexed TCAM Architecture for Packet Classifiers

Tania Banerjee-Mishra and Sartaj Sahni, University of Florida, Gainesville, FL, USA
{tmishra, sahni}@cise.ufl.edu
Gunasekaran Seetharaman, AFRL, Rome, NY, USA
Gunasekaran.Seetharaman@rl.af.mil

***Abstract*—We propose an indexed TCAM architecture, PC-TRIO, for packet classifiers. PC-TRIO uses wide SRAMs and index TCAMs. On our classifier datasets, PC-TRIO on an average reduced TCAM power by 96% and lookup time by 98%, compared to PC-DUOS+ [24] that does not use indexing or wide SRAMs. We extend PC-DUOS+ by augmenting it with wide SRAMs and index TCAMs using the same methodology as used in PC-TRIO, to obtain PC-DUOS+W. On ACL datasets, PC-DUOS+W reduced TCAM power by 86% and lookup time by 98%, compared to PC-DUOS+.**

I. Introduction

Packet classification is a key step in routers for various functions such as routing, creating firewalls, load balancing and differentiated services. Internet packets are classified into different flows based on packet header fields and using a table of rules in which each rule is of the form (F, A), where F is a filter and A is an action. When an incoming packet matches a rule in the classifier, its action determines how the packet is handled. For example, the packet could be forwarded to an appropriate output link, or it may be dropped. A d-dimensional filter F is a d- tuple $(F[1], F[2], \cdots, F[d])$, where $F[i]$ is a range specified for an attribute in the packet header, such as destination address, source address, port number, protocol type, TCP flag, etc. A packet matches filter F, if its attribute values fall in the ranges of $F[1], \cdots, F[d]$. Since it is possible for a packet to match more than one of the filters in a classifier thereby resulting in a tie, each rule has an associated cost or priority. When a packet matches two or more filters, the action of the matching rule with the lowest cost (highest priority) is applied on the packet. It is assumed that filters that match the same packet have different priorities.

TCAMs are used widely for packet classification. The popularity of TCAMs is mainly due to their high-speed table lookup mechanism in which all the TCAM entries are searched in parallel. Each bit of a TCAM may be set to one of the three states 0, 1, and '?' (don't care). A TCAM is used in conjunction with an SRAM. Given a rule (F, A), the filter F of a packet classifier rule is stored in a TCAM word and action A is stored in an associated SRAM word. All TCAM entries are searched in parallel and the first match is used to access the corresponding SRAM word to retrieve the action. So, when the packet classifier rules are stored in a TCAM in decreasing order of priority (increasing order of cost), we can determine the action corresponding to the matching rule of the highest priority, in one TCAM cycle. The main limitation of TCAMs is that these memories are power hungry. In fact at the same access rate, a TCAM may consume 30 times more power than an SRAM used for a software based classification [18]. The more the number of entries in the TCAM, the higher the power needed to perform a search. This problem is worsened for packet classifiers since typically a classifier rule includes port range fields that need multiple TCAM entries per rule for representation in the TCAM. This is called range expansion. Given that the source and destination port numbers are represented in 16 bits, the number of TCAM entries needed to represent a port range in the worst case is 30 corresponding to the range $[1, 2^{16} - 2]$. Thus, a filter having both source and destination port ranges set to $[1, 2^{16} - 2]$ undergoes a worst case expansion of $30 \times 30 = 900$ TCAM entries.

In this paper we evaluate a triple TCAM architecture, PC-TRIO for packet classifiers. In PC-TRIO, the TCAMs are augmented with indexing and wide SRAMs. The technique of indexing directly reduces the power consumption during lookup by selectively searching only a specific TCAM partition on the second stage of the lookup. In this architecture, port ranges are stored in wide SRAM words, rather than in the TCAM for most of the rules, and hence do not need multiple TCAM entries to represent them. The content of the wide SRAM word may be processed by a specialized and fast hardware. Finally, we present efficient incremental update algorithms. To the best of our knowledge, this is the first work that attempts to use an indexed TCAM architecture for packet classifiers.

Our paper is organized as follows. Section II presents background and related work in this area. Section III describes the PC-TRIO architecture and associated algorithms and Section IV presents experimental results. We conclude in Section V.

II. Background and Related Work

We describe the research on TCAM based packet classifiers in Section II-A, and describe existing indexed TCAM architectures for packet forwarding tables in Section II-B. We discuss the main problems in having an indexed TCAM architecture for packet classifiers in Section II-C and then in Section II-D show how to overcome these problems.

A. Packet Classifiers

The work on packet classifiers in TCAMs, targets three main problems: port range expansion, power consumption and updates. The first two problems are inter-related as reducing port range expansion also reduces the power consumption in a TCAM. Various approaches have been proposed in the literature to alleviate the range expansion problem. The schemes

This material is based upon work funded by AFRL, under AFRL Contract No. FA8750-10-1-0236.

in [1], [7], [6], [9], [13], [16] encode the ranges and store modified rules in the TCAM. As a packet arrives, an encoded search key is created from the packet header fields using the encoding algorithm and the TCAM is searched using the encoded search key. Spitznagel et al. [11] proposed enhancements to the TCAM hardware to include range comparison. With such an enhanced TCAM circuit, each rule occupies a single entry in the TCAM.

Compressing packet classifiers by removing redundancies is an effective strategy to reduce TCAM power consumption. The approaches in [4], [15], [10], [12], [14] present algorithms that transform an input classifier to an equivalent smaller classifier. These algorithms quite naturally contain port range expansions. While these approaches bring about significant reductions in classifier size, they are generally not suitable for incremental updates, since a rule to be deleted, for instance, may not be present in the transformed classifier.

Song and Turner [8] describe an algorithm for fast incremental filter updates. An explicit priority value (which we call block number in this paper) is calculated for each rule based on the rule's implicit priority, which is derived from the position of the rule in the classifier, and the implicit priority values of the overlapping rules. The block number so computed is stored along with the rule in the TCAM using unused TCAM bits. A new rule may be placed anywhere in the TCAM. This relieves the TCAM of moving existing rules to maintain priority ordering. Instead, during lookup, multiple lookups per packet are performed to identify the best matching rule. Mishra, Sahni and Seetharaman in PC-DUOS [21] and PC-DUOS+ [24] use dual TCAMs for representation and incremental update of classifiers.

B. Forwarding tables with indexed TCAMs

The concept of using an index TCAM for a forwarding table was proposed by Zane et al. [2] and further refined by Lu and Sahni in [3]. A forwarding table can be viewed as a one dimensional packet classifier, containing only destination prefixes. Zane et al. [2] proposed a 2-level TCAM architecture in which the first level TCAM is an index to the partitions in the second level TCAM. We refer to a partition in a TCAM as a $bucket$. The partitions and indexes are constructed by $carving$ the binary trie representing the prefixes in the forwarding table.

Lu and Sahni in [3], further augment the traditional 1-level TCAM lookup structure as well as the 2-level TCAM structure of Zane et al. [2] with wide SRAMs and store the suffixes of several prefixes in a single wide SRAM word. This enables a reduction in both power consumption and total TCAM memory requirement. Mishra and Sahni, in PETCAM [19] and DUO [20] obtained further reduction in power and TCAM space for packet forwarding, using the indexing and wide SRAM schemes. In particular, DUO [20] is a dual TCAM architecture used for packet forwarding that uses efficient memory management algorithms for the two TCAMs. These algorithms help DUO in executing consistent incremental updates [22], [23].

C. Problems in storing a classifier in an indexed TCAM

There are two problems in mapping a packet classifier to an indexed TCAM architecture with wide SRAMs. Recall that during a TCAM lookup, the contents in the SRAM word corresponding to the first matching rule is returned. A constraint on the size of a wide SRAM word (and also that on the size of a TCAM bucket), makes it impossible to guarantee that the first matching word will contain the highest priority rule matching the packet. For example, consider the classifier with 4 rules in Figure 1, where each rule has two fields - a destination, and a source. The classifier is mapped to the indexed TCAM in Figure 2. The data TCAM has two buckets

Filter		Action	Priority
Destination	Source		
00*	1000	A1	1
0*	0101	A2	2
000*	01*	A3	3
*	*	A4	4

Fig. 1. An example classifier

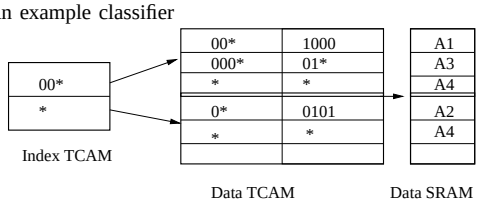

Fig. 2. Classifier rules stored in a indexed TCAM

and the index TCAM uses bits from the destination prefix of each rule, to index into the buckets of the data TCAM. In this setup, assuming that addresses are 4 bits, suppose a packet arrives with destination and source addresses as 0000 and 0101 respectively. The best matching rule from Figure 2 is the second rule on the first bucket of the data TCAM and A3 is returned as the action to be applied on the packet. However, from the table in Figure 1, A2 is the desired action. Thus if there are multiple matching rules on a TCAM, then all the corresponding SRAM words must be processed to return the action of the matching rule with the highest priority, and this will take more than one TCAM clock cycle to finish a search. This is the first problem.

The second problem is about the $covering\ rules$ of a wide SRAM word or a data TCAM bucket. A $covering\ prefix$ [2], [3], in the context of packet forwarding tables, is a default prefix for a TCAM bucket. The presence of covering prefixes in a TCAM bucket makes every search in the TCAM bucket return at least one match. In a packet classifier, covering rules similarly guarantee that a search on a TCAM bucket matches at least one rule. The fourth rule in Figure 1 is a covering rule and hence entered in both the TCAM buckets in Figure 2. A packet classifier may have several covering rules for a TCAM bucket. Further, different TCAM buckets may need the same covering rules which makes it necessary to store a single rule multiple times in the TCAM, once in every TCAM bucket for which it is a covering rule. Having a rule replicated as such in the TCAM, is unacceptable specially considering the fact that the replicated rules themselves may undergo range expansion.

D. Overcoming these problems

The dual TCAM architecture presented for PC-DUOS [21] and PC-DUOS+ [24], as well as the PC-TRIO architecture presented in this paper, makes it possible to get around both

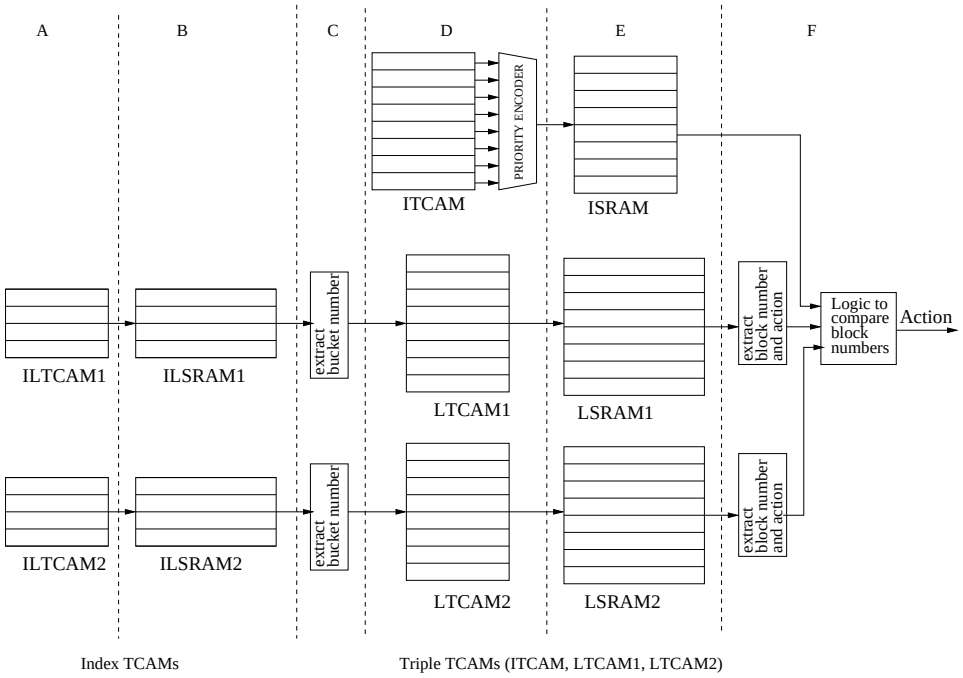

Fig. 3. PC-TRIO Architecture

the problems mentioned about using wide SRAMs and index TCAMs with a TCAM for packet classifiers. The LTCAM (Leaf TCAM) of PC-DUOS stores independent rules. Two rules are independent iff no packet matches both the rules. Storing a set of independent rules in a TCAM, ensures that at most one TCAM entry matches during a search and we simply process the corresponding SRAM word. The ITCAM (Interior TCAM) of PC-DUOS stores all the remaining rules which includes the covering rules. During a lookup both TCAMs are searched in parallel, and in case there is no match on the LTCAM, the ITCAM returns the action for the matching rule with the highest priority. Note that the LTCAM of PC-DUOS is a suitable candidate for augmenting with wide SRAM words and an index TCAM, since at most one TCAM entry matches during a search. The rules in the ITCAM, on the other hand, are not independent and hence multiple TCAM entries will match during a search. Thus, the ITCAM is not a suitable candidate for using with it a wide SRAM or an index TCAM.

III. PC-TRIO

The PC-TRIO architecture is presented in Section III-A. The algorithms for storing and updating the TCAMs are discussed in Sections III-B and III-C. The differences with related architectures are presented in Section III-D.

A. The Architecture

Figure 3 illustrates the PC-TRIO architecture. It primarily consists of three TCAMs, the ITCAM (Interior TCAM), the LTCAM1 (Leaf TCAM) and the LTCAM2. The corresponding associated SRAMs are: ISRAM, LSRAM1 and LSRAM2, respectively. The LTCAMs store independent rules, hence both the TCAMs are augmented with wide SRAMs and index TCAMs. ILTCAM1 and ILTCAM2 are the index TCAMs for LTCAM1 and LTCAM2, respectively. The index TCAMs also have wide associated SRAMs, namely, ILSRAM1 and ILSRAM2. Since the rules stored in the two LTCAMs and the two ILTCAMs are independent, at most one rule (in each LTCAM and ILTCAM) will match during a search. So these TCAMs do not need a priority encoder. A priority encoder assists in resolving multiple TCAM matches and is used with the ITCAM to access the ISRAM word corresponding to the highest priority matching rule in the ITCAM.

A lookup in PC-TRIO is pipelined with 6 stages marked A-F in Figure 3. In the first stage A, the ILTCAMs are searched. The ILSRAMs are accessed, using the address of the matching ILTCAM1 and ILTCAM2 entries in stage B. The matching wide ILSRAM words are processed in stage C to obtain the corresponding bucket index for LTCAM1 and LTCAM2. In stage D, the bucket indexes so obtained are used to search the corresponding buckets in the LTCAMs. The ITCAM is also searched in this stage. In the next stage E, the ISRAM, and the LSRAMs are accessed using the addresses of the matching TCAM entries. In the final stage F, the contents of the wide LSRAM words are processed and the best action is chosen from the at most three actions returned by the ISRAM, LSRAM1 and LSRAM2 by comparing the priorities of the corresponding rules.

B. Storing rules in TCAMs

There are several steps of processing a packet classifier to store the rules in the TCAMs. The first step is to create a priority graph and multi-dimensional tries for the rules in the classifier. This is further discussed in Section III-B1. In the second and third steps, the LTCAM1 and LTCAM2 subsystems are populated as discussed in Sections III-B2 and III-B3, respectively. The fourth step is to store the remaining rules in the ITCAM in priority order, which is discussed in Section III-B4.

1) Representing Classifier Rules: The classifier rules are represented in a priority graph, which contains one vertex for

each rule in the classifier. A priority graph contains one vertex for each rule in the classifier. There is a directed edge (u, v) from vertex u to vertex v iff (a) the rules corresponding to u and v overlap (i.e., at least one packet matches both rules) and (b) the priority of u is more than that of v (we assume that overlapping rules have different priority). For the directed edge (u, v), we say that u is the parent of v and v is the child of u. The priority graph is used to assign block numbers to rules/vertices as follows [8]. All vertices with in-degree 0 are assigned the block number 1. Each remaining vertex v is assigned a block number equal to

$$1 + \max_{(u,v)\in E} \{block\ number\ of\ u\}$$

where E is the set of edges in the priority graph. Thus a child of any vertex is assigned a block number that is at least one more than the block number of this vertex.

Next we create a multi-dimensional trie, Trie1, where each dimension represents one field of a rule. Initially, Trie1 is three-dimensional, with the three fields, source, destination and protocol of a classifier rule used for this purpose. The fields appear in the following order in the trie: <*destination, source, protocol*>. We assume that the destination and source fields as well as the protocol field of the filters are specified as prefixes. So, these are represented in a trie in the standard way with the left child of a node representing a 0 and the right child a 1. A classifier rule, along with its source and destination port ranges, is stored on the protocol node that is arrived at after traversing the trie starting from its root, using first the destination, then the source and finally the protocol fields of the rule.

We identify a set of independent rules as described in Section III-B2. All the remaining rules are used to create another multi-dimensional trie, Trie2, in which fields in a filter rule appear in the order <*source, destination, protocol*>. Note that the source and destination tries are switched in Trie2, with respect to Trie1. So, while destination trie is the outermost trie in Trie1, in Trie2, source is the outermost trie.

2) Storing rules in the LTCAM1: The process of storing rules in the LTCAM1 subsystem is described in five subsections below. First, independent rules are identified (Section III-B2a), next, the format of storing information in a wide LSRAM word is discussed (Section III-B2b), then we describe the creation of LTCAM1 entries using the process of carving (Section III-B2c). Next we describe partial port range expansion (Section III-B2d) that may be necessary, and finally, the creation of ILTCAM1 and ILSRAM1 entries (Section III-B2e).

a) Identifying Independent Rules: Recall that two rules are $independent$ iff no packet is matched by both rules. For the LTCAM1, we are interested in identifying the largest set of rules that are pairwise independent. To find an independent rule set in acceptable computing time, we relax the "largest set" requirement and instead look for a large set of independent rules using a two step process. In the first step, we create a *leaves of leaves set* [21] of protocol nodes in a multi-dimensional trie using the algorithm in Figure 4. The nodes belonging to the leaves of leaves set in Trie1 are obtained by traversing the multi-dimensional trie from the root to the leaves of the destination trie and then from these leaves into their attached source trie and then from the leaves of the source trie into the leaves of their attached innermost trie for the protocol field.

```
Algorithm: findNode(node) Inputs:
node: a trie node, initially set to the root of a multi-dimensional trie.
Output:
a leaves of leaves set of protocol nodes storing classifier rules.
    for each child i of node
        findNode(node→child[i]);
    endfor
    if (node is a leaf) // true if node has no left or right child.
        if (node contains root of a trie)
            findNode(node→trie→root);
        else // node belongs to trie for the last field (protocol)
            append protocol node to leaves of leaves set
        endif
    endif
```

Fig. 4. Selecting protocol nodes for leaves of leaves set

In the second step, for each protocol node in the leaves of leaves set, we identify a set of independent rules stored in that protocol node by building a small priority graph with rules only in that protocol node. Vertices in the priority graph with in-degree 0 comprise a set of independent rules. A collection of independent rules from all protocol nodes in the leaves of leaves set, gives us the rules to be entered in the LTCAM1.

b) Wide SRAM Word Format: Once the rules to be stored in LTCAM1 are identified, subtries of the multi-dimensional trie are carved and rules in the protocol nodes in a subtrie are stored in a LSRAM1 word. In particular, for each rule in a protocol node we store the rule's source and destination port ranges, block number, and action. We also store the suffix of a protocol node, which is the path from the root of the carved subtrie to the protocol node. Figure 5 shows a format for encoding this information in a wide SRAM word. The fields in this format are described briefly as follows:

1) *Match start position*: This field specifies the positions of the first bit in the source, destination and protocol fields of a packet header starting from which suffixes of protocol nodes in the SRAM word must be matched.
2) *Count*: This is the number of protocol nodes in the leaves of leaves set stored in the SRAM word.
3) *len(Si)*: This field specifies the length of the suffix for protocol node i in the SRAM word.
4) *Ci*: This gives the number of classifier rules stored for protocol node i.
5) $Data_j$: $Data_1, \cdots, Data_N$ give details of the N rules in the carved subtrie. The rules for protocol node 1 of this subtrie come first, followed by those of the second protocol node and so on. $Data_j$ gives the block number, action, source and destination port range types for the jth classifier rule.
6) *Si*: This field stores the suffix for protocol node i.
7) *Port ranges*: Stores the port ranges for the N rules.

There are three types of ranges found in a classifier. These are: a whole range ([0-65535]), a range with the same start and end point, and a range with different start and end points. The port range type subfield in the Data field represents these three types of ranges using two bits. To save space in a SRAM

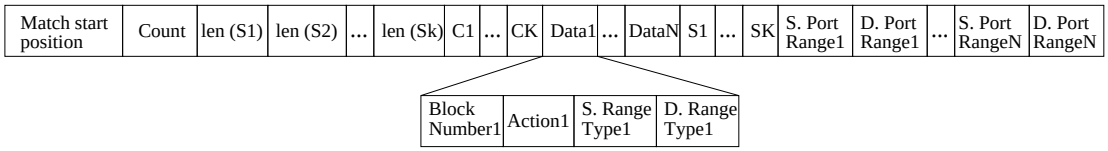

Fig. 5. Data encoding in a wide SRAM word

word, a whole range is never entered and only one port number is entered for a range with the same start and end points.

c) Creating LTCAM1 entries: A trie is carved into subtries to assign rules to the wide SRAM words. The Trie1 is carved using the carving heuristic $visit\_postorder$ of DUO [20] that has been enhanced for multi-dimensional tries. This carving heuristic creates independent (disjoint) entries for the

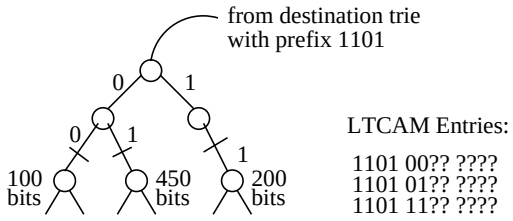

Fig. 6. Nodes in a source trie is being carved.

LTCAM1. The path starting from the root of Trie1 to the root of the subtrie defines an LTCAM1 entry. Figure 6 shows a portion of a source trie that hangs off a destination trie, where carving takes place at nodes 00, 01, and 11 of the source trie. The path from the root to the node of the destination trie from which the source trie hangs off is 1101. Thus, after carving the node at 00 on the source trie, the LTCAM1 entry is 1101 00?? ????, assuming addresses and protocol fields are represented using 4 bits each. Similarly, the two other LTCAM1 entries in this example are 1101 01?? ???? and 1101 11?? ????. Figure 6 also shows a size assignment (in bits) on the three nodes where carving takes place. These sizes are computed for all the trie nodes even before the carving algorithm is invoked. The size assigned to a trie node represents the number of LSRAM1 bits needed to store all the classifier rules (for LTCAM1) in a subtrie rooted at that node. For example, for a subtrie rooted at the source node 01, the number of bits needed to store the action, block number, port ranges of classifier rules and suffixes of protocol nodes present in this subtrie, is 450. If the actual width of a SRAM word is, say, 500 bits, then the rules in this subtrie will fit in an SRAM word and we may carve at the source node 01. A corresponding LSRAM1 entry is constructed for the classifier rules in the format given by Figure 5. The carving heuristic carves a node n on the trie when any of the following two conditions is true. Here, p is the parent of n in the trie.

- C1) The size assigned to n is less than the width of a SRAM word, but that assigned to p is more than the the width of a SRAM word.
- C2) A descendant of p was carved.

The second condition ensures that the carving creates disjoint TCAM entries [20].

d) Partial port range expansion: : It is possible that the SRAM bits needed to store the classifier rules for LTCAM1 on a protocol node exceeds the capacity of a wide SRAM word. This case is shown in Figure 7(a) where the black node is a protocol node in the leaves of leaves set and the size assigned to it is 600 bits. Suppose the width of the SRAM word is 500 bits. Then to avoid overflowing an SRAM word, we must split the rules in the protocol node, into two or more SRAM words. Instead of replicating the LTCAM1 entry for each of the split SRAM words, we create a source port range trie as shown in Figure 7(b), and carve nodes on this trie to

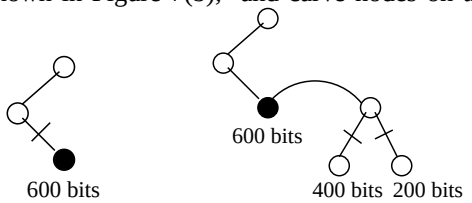

Fig. 7. Prefixes in forwarding table before and after applying updates

create independent LTCAM1 entries. Each node in the source port trie inherits those classifier rules (for LTCAM1) from the protocol node that have their source port range overlap with the port range represented by the trie node. Thus multiple copies of a rule may be created, one for each trie node with port range overlapping the source port range of the rule. After the source port trie is created, the carving heuristic resumes its traversal along the source port trie, and carves source port nodes if they satisfy either condition C1, or C2. In the example of Figure 7(b), two LTCAM1 entries are created, one each for the two carved nodes. These LTCAM1 entries differ on the first bit on the source port field, with one entry having a 0 while the other having a 1. If the classifier rules in a leaf node of the source port trie overflows an SRAM word, then a destination port trie is created for the destination port ranges on rules of that leaf node, and the carving heuristic finds appropriate nodes to carve on the destination port trie.

The source and destination port tries are thus created in PC-TRIO only when necessary, and then, to minimize the range expansion problem we use multi-bit tries for storing the port ranges. The bits used to arrive at a node in the multi-bit trie define an LTCAM1 entry.

e) Creating ILSRAM1 and ILTCAM1 entries: After carving Trie1 to create suffixes for entering into LSRAM1, we carve Trie1 again a second time, to create subtries that contain LTCAM1 entries. All LTCAM1 entries in a subtrie are entered in a LTCAM1 bucket. Thus, at the end of this carving step, the LTCAM1 entries are partitioned into buckets. The bits from the root of the multi-dimensional trie to a carved node defines an index that points to an LTCAM1 bucket.

After partitioning the LTCAM1 into buckets, Trie1 is carved a third and final time. This time, a carved subtrie contains indexes to LTCAM1 buckets. Suffixes of these indexes, along with the corresponding LTCAM1 bucket indexes, are stored in the ILSRAM1, and the bits on path from the root of the Trie1 to a carved node define an ILTCAM1 entry.

3) Storing rules in LTCAM2: This is done exactly as for LTCAM1, by processing the rules stored in Trie2. In particular, Trie2 undergoes carving in a similar manner as described for Trie1 and the LTCAM2 system is populated. The remaining rules, i.e. rules that are stored neither in the LTCAM1 nor in the LTCAM2 subsystem, are stored in the ITCAM.

4) Storing rules in the ITCAM: The ITCAM does not have a wide ISRAM, hence, a rule to be entered in the ITCAM, must have its port range stored in the ITCAM itself. An ISRAM word contains the action and block number of a classifier rule stored in the corresponding ITCAM entry. We use DIRPE to encode these port ranges on the ITCAM. DIRPE is suitable for incremental updates, unlike database dependent range encoding schemes. However, if fast incremental updates are not needed, then any range encoding scheme may be chosen for the ITCAM.

C. Incremental Updates

When an update request is received, the priority graph is updated as described in Section III-C1. Then Trie1 and, if necessary, Trie2 are updated as described in Section III-C2. As the tries are updated, it may be necessary to carve the tries at different trie nodes. This is discussed in Section III-C3. Updating the TCAMs is discussed in Section III-C4.

1) Updating the priority graph: To insert a new rule, the first step is to store the rule in the priority graph. A new vertex v is created for the rule. The existing rules that overlap with v are identified and new edges are formed between v and the vertices of overlapping rules, with directions of the edges set from the higher to the lower priority rules. Then, a block number is assigned to v, which is one more than the maximum block number of the nodes from which v has an incoming edge. If the block number of a child vertex is not more than that assigned to v, the child's block number is updated so that it is at least one more than the block number of v. If the rule r corresponding to this child vertex is stored in ITCAM, then, r must be moved to the ITCAM block represented by its updated block number, and the ISRAM entry for r is also updated with the changed block number. On the other hand, if r is in one of the LTCAMs, then, we simply change r's block number in the corresponding LSRAM entry. Updates to block numbers are propagated to all vertices reachable from v.

To process a delete request, the vertex corresponding to the rule along with the incident edges is removed from the priority graph.

2) Updating the tries: To insert a new rule, the rule is first added to Trie1. If the rule is an independent rule in a protocol node in the leaves of leaves set, then it is inserted in the LTCAM1. Otherwise, the rule is added to Trie2. If the rule is an independent rule in a protocol node in the leaves of leaves set for Trie2, then the rule is inserted in the LTCAM2. Otherwise, the rule is inserted in the ITCAM.

If a new rule is stored in the LTCAM1 or the LTCAM2, then some of the existing rules in that TCAM may no longer be independent. If such a non-independent rule exists in the LTCAM1, then that rule is added to the Trie2 and if the rule can be added to the LTCAM2 it is moved from the LTCAM1 to the the LTCAM2. Otherwise, the rule is moved from the LTCAM1 to the ITCAM. Similarly, a new rule added to the LTCAM2 may cause some of the existing LTCAM2 rules to be moved to the ITCAM.

To delete a rule, the rule is deleted from Trie1 and also from Trie2 if it was stored in Trie2. The rule is then deleted from the TCAM that stores the rule.

3) Updating the trie carving: We now discuss the dynamics of creation and merging of LSRAM words when a new rule is added or an existing rule is deleted. Both Trie1 and Trie2 contain nodes that were carved to create TCAM and SRAM entries. We describe how these entries change for Trie1. The process is similar for Trie2. When a rule is added to Trie1 at node t, if there is an ancestor a of t, where carving was done to create a wide LSRAM1 word s, and if there is space in s to place the action, block number, port ranges of the new rule, then, the new rule is placed in s. If there is no space in s, then the contents of s are split, by carving descendants of a to create two or more LTCAM1 entries. If, on the other hand, t does not have an ancestor a, then one of the two things below may happen. If there is an ancestor b of t, such that b has at least one carved descendant and the subtrie rooted at b needs fewer SRAM bits than the width of a SRAM word to represent the classifier rules, then b is carved. As a result, the new rule is stored with some existing rules in a new SRAM word. Note that the existing rules, have additional suffix bits in the newly created SRAM word and old LTCAM1 entries for the existing rules are deleted. If no such b exists, a new LTCAM1 entry is created by carving at t. The corresponding LSRAM1 word contains only the newly added rule.

When a rule in an LTCAM1 is deleted, then the rule is first removed from the LSRAM1 word. If the LSRAM1 word becomes empty, then the corresponding LTCAM1 word is deleted. Otherwise, if the contents of the LSRAM1 word can be merged with another LSRAM1 word then a new LTCAM1 entry is created while the LTCAM1 entries for the merged words are deleted.

The algorithms to merge and split buckets on the LTCAMs are similarly based on manipulating the carving in Trie1 and Trie2.

4) Updating the TCAMs: To insert or move a rule in a TCAM we need a free slot at an appropriate location in the TCAM. This slot can be obtained efficiently using memory management algorithms developed for TCAMs. In particular, the memory management schemes from DUO [20] may be used. For the ITCAM of PC-TRIO, we implemented the DLFS_PLO scheme, as its the most efficient scheme known to us for moving free slots to a desired location in a TCAM. In the DLFS_PLO initial rule placement scheme, free slots are kept in the region between two blocks. Additionally, there may be free slots $within$ a block. So a list of free slots is maintained for each block on the TCAM, with the list being empty initially. As rules are deleted from a block, the freed slots are added to the list for that block. Thus, DLFS_PLO requires no moves for most of the time to get or return a free slot.

The memory management scheme for the LTCAM of DUO is relatively simple as all the rules in the LTCAM are independent so a new rule may be inserted anywhere in the

	PC−DUOS	PC−DUOS+	PC−TRIO	PC−DUOS+W
1.	Uses single LTCAM	Uses single LTCAM	Uses two LTCAMs	Uses two LTCAMs
2.	No wide SRAMs or index TCAMs	No wide SRAMs or index TCAMs	Uses wide SRAMs and index TCAMs	Uses wide SRAM and index TCAM
3.	LTCAM stores highest priority independent rules	LTCAM stores highest priority independent rules	LTCAMs store independent rules	LTCAM stores highest priority independent rules
4.	Aborts ITCAM search when LTCAM search succeeds	Aborts ITCAM search when LTCAM search succeeds	Waits for ITCAM search to finish	Aborts ITCAM search when LTCAM search succeeds
5.	Independent rules are filtered leaves of leaves set in trie	Independent rules are vertices in priority graph with indegree=0	Independent rules are leaves of leaves set in trie	Independent rules are vertices in priority graph with indegree=0

Fig. 8. Differences among the architectures

TCAM. However, we still need to locate a free slot. The LTCAM memory management algorithm of DUO creates a linked list of the free slots. When a free slot is needed, a slot is obtained from the head of the free slot list. PC-TRIO uses the memory management algorithm in DUO for its LTCAM1 and LTCAM2.

D. Differences among PC-DUOS, PC-DUOS+, PC-DUOS+W and PC-TRIO

We note that the methodology used in this paper for PC-TRIO may be used to add index TCAMs and wide SRAMs to PC-DUOS+ to arrive at a new architecture PC-DUOS+W. Although PC-DUOS [21] may be similarly extended to obtain PC-DUOSW, we do not consider this extension here as PC-DUOS+ was shown to be superior to PC-DUOS [24]. Figure 8 highlights the differences among PC-DUOS, PC-DUOS+, PC-DUOS+W and PC-TRIO.

Unlike the other architectures, PC-TRIO does not guarantee that the rules in the LTCAMs are of the highest priority among all overlapping rules. Thus, PC-TRIO must wait for an ITCAM lookup to complete even if there are matching rules in the LTCAMs. Although the rule assignment algorithms for PC-TRIO may be modified so that the LTCAM rules are the highest priority among all overlapping rules (and thus avoid having to wait for an ITCAM lookup to complete in cases when a match is found in an LTCAM), doing so retards the performance of PC-TRIO to the point where it offers little or no power and lookup time benefit over PC-DUOS+W.

IV. Experimental Results

We compare PC-TRIO, with PC-DUOS+W and PC-DUOS+ [24]. We first give the setup used by us for the experiments in Section IV-A and then describe our datasets in Section IV-B. Finally we present our results in Section IV-C.

A. Setup

We programmed the rule assignment, trie carving and update processing algorithms of PC-TRIO using C++. We designed a circuit for processing wide SRAM words using Verilog and synthesized it using Synopsys Design Compiler to obtain power, area and gate count estimates. We used CACTI [25] and a TCAM power and timing model [17] to estimate the power consumption and search time for the SRAMs and the TCAMs respectively. The process technology used in the experiments is 70nm and the voltage is 1.12V. It is assumed that the TCAMs are being operated at 360MHz [29].

The TCAM and SRAM word sizes used are consistent for all the architectures used in the comparison. The word size is 144 bits for the TCAMs. For SRAMs we have different word sizes depending upon the TCAMs they are used with. The ISRAM words of all the architectures, as well as the LSRAM words of PC-DUOS+, are 32 bits wide. The LSRAM1 and LSRAM2 words of PC-TRIO and the LSRAM words of PC-DUOS+W are 512 bits, while the ILSRAMs are 144 bits wide. The bucket size for LTCAMs in PC-TRIO and PC-DUOS+W is set to 65 TCAM entries. PC-DUOS+ uses DIRPE [1] to encode port ranges. The classifier rules stored in the ITCAMs of PC-TRIO and PC-DUOS+W also use DIRPE to encode port ranges. Since the TCAM word size is set to 144 bits, we assume that 36 bits are available for encoding each port range in a rule. With this assumption, we use the strides 223333 as these give us minimum expansion of the rules [1], [21].

B. Datasets

We used two sets of benchmarks derived from ClassBench [5]. The first set of benchmarks consists of 12 datasets each containing about 100,000 classifier rules and is generated from seed files in ClassBench. This dataset is used to compare the number of TCAM entries, power, lookup performance and space requirements of PC-TRIO, PC-DUOS+W and PC-DUOS+ [24].

The second set of benchmarks was reused from [24]. There are 13 datasets here which are used to compare incremental update performance of PC-TRIO, with PC-DUOS+ [24] and PC-DUOS+W.

C. Results

1) Number of TCAM entries: Using wide SRAM words to store portions of classifier rules, reduces the number of TCAM entries. Figure 9 gives the results of storing our datasets in the three architectures. The first, second and third columns show the index, name, and the number of classifier rules, respectively, of a dataset. The fourth, fifth and sixth and seventh columns give for PC-DUOS+, the total number of TCAM entries, the number of ITCAM entries, the TCAM power and lookup time, respectively. Similarly, the eighth, ninth, tenth and eleventh columns give the corresponding numbers for PC-DUOS+W and the remaining four columns give those for PC-TRIO.

Figure 10(a) gives the TCAM compaction ratio of the three architectures, obtained by dividing the number of TCAM entries for each dataset by the number of rules in the classifier. PC-DUOS+ does not use wide SRAMs, hence there is no

Index	Dataset	#Rules	PC-DUOS+				PC-DUOS+W				PC-TRIO			
			Entries	#ITCAM	Watts	Time(ns)	Entries	#ITCAM	Watts	Time(ns)	Entries	#ITCAM	Watts	Time(ns)
1	acl1	99309	117033	379	36	2624.39	21146	379	0.23	0.50	21085	182	0.19	1.00
2	acl2	74298	101857	19421	31	1122.39	37491	19421	6.35	30.36	36593	18439	6.04	149.43
3	acl3	99468	131243	30859	40	1640.47	52632	30859	9.47	80.49	26823	1017	0.40	2.19
4	acl4	99334	127320	25189	39	1730.46	49912	25189	7.98	45.95	34034	6547	2.32	24.12
5	acl5	98117	105375	1535	32	2072.16	32932	1535	0.53	0.41	34993	2209	0.77	4.98
6	fw1	89356	142085	91473	43	2466.72	98425	91473	27.92	2318.82	26610	4864	1.60	15.01
7	fw2	96055	129249	27084	39	1543.76	43146	27084	8.30	86.77	22196	1494	0.53	3.18
8	fw3	80885	117731	39199	36	1007.04	51228	39199	11.99	215.21	26269	7479	2.38	30.09
9	fw4	84056	211403	116149	64	3182.03	131505	116149	35.46	2139.21	27617	4894	1.60	15.16
10	fw5	84013	111989	55650	34	930.94	65598	55650	17.00	615.49	22361	3454	1.15	9.02
11	ipc1	99198	112154	22165	34	1288.02	41920	22165	6.82	45.11	23894	567	0.26	1.40
12	ipc2	100000	100000	30133	30	784.69	47247	30133	9.23	113.77	20195	0	0.09	0.75

Fig. 9. Number of TCAM entries, ITCAM entries and TCAM power and lookup time in PC-DUOS+, PC-DUOS+W, PC-TRIO

compaction, instead, there is expansion to handle port ranges. Thus, the compaction ratio for PC-DUOS+ is at least 1 for every dataset. The compaction achieved by PC-TRIO is more than that of PC-DUOS+W for almost all the datasets. This is because, PC-TRIO has fewer ITCAM entries and therefore stores more rules in wide SRAM words. For acl5, PC-DUOS+W identified more independent rules compared to PC-TRIO. The algorithm to identify independent rules is the same for PC-DUOS+W and PC-DUOS+ which results in identical ITCAM entries for these two architectures.

No classifier rules in the LTCAMs of PC-DUOS+W and PC-TRIO needed partial port range expansion (Section III-B2d). So all LTCAM entries in PC-DUOS+W and PC-TRIO were at most 72 bits.

2) Power: Figure 9 gives the TCAM power consumption during a lookup, while Figure 10(b) gives the normalized total power obtained for each dataset by dividing the total TCAM and SRAM power in an architecture by that of PC-TRIO during a lookup. The vertical axis is scaled logarithmically

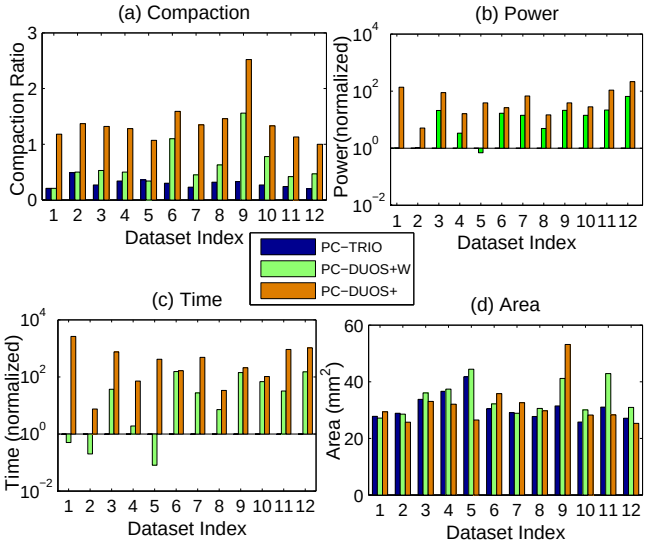

Fig. 10. Comparison of compaction ratio, total power, lookup time and area

and based at 1. PC-TRIO uses less power for all datasets except acl5. The average improvement in power with PC-TRIO is 96% relative to PC-DUOS+, and 65% relative to PC-DUOS+W. The average improvement in power with PC-DUOS+W is 71%, relative to PC-DUOS+. The maximum improvement with PC-TRIO is observed for ipc2 (99%) and the minimum for acl2 (80%), compared to PC-DUOS+. The maximum improvement with PC-DUOS+W is observed for acl1 (99%) and the minimum for fw1 (35%), compared to PC-DUOS+. The maximum improvement with PC-TRIO is observed for ipc2 (98%) and the minimum for acl1 (2%), compared to PC-DUOS+W.

3) Lookup Performance: Figure 10(c) gives the average lookup time, normalized with respect to that of PC-TRIO. TCAM search time is proportional to the number of TCAM entries. Hence, PC-DUOS+ requires the maximum time.

PC-DUOS+W is faster than PC-TRIO for the ACL tests acl1, acl2 and acl5. For these datasets, the number of ITCAM entries in PC-DUOS+W and PC-TRIO (columns 9 and 13 of Figure 9) are comparable. Thus, the ITCAM search times are comparable, as are the number of lookups served by the ITCAMs. This, coupled with the fact that ITCAM searches are slower, give PC-DUOS+W an immediate advantage since it, unlike PC-TRIO, aborts an ITCAM search after finding a match in the LTCAM. However, for these three tests, the lookup times using PC-TRIO are quite reasonable (column 15 of Figure 9). For the other datasets PC-TRIO has fewer rules in the ITCAM, which makes PC-TRIO lookups faster even though it has to wait for ITCAM search to finish.

The average improvement in lookup time with PC-TRIO and PC-DUOS+W (relative to PC-DUOS+) are 98% and 76%, respectively. The average improvement in lookup time with PC-TRIO (relative to PC-DUOS+W) is 68%. The maximum improvement using PC-TRIO rather than PC-DUOS+ is observed for acl1 (99.96%) and the minimum for acl2 (86.6%). The maximum improvement using PC-DUOS+W rather then PC-DUOS+ is observed for acl1 (99.98%) and the minimum for fw1 (5%). The maximum improvement with PC-TRIO rather than PC-DUOS+W is observed for tests fw1, fw4 and ipc2 (99%) and the minimum for acl4 (47%).

4) Space requirements: We obtained SRAM area from CACTI results and estimated TCAM area using the same technique as used in PETCAM [19], where area of a single cell is multiplied by the number of cells and then adjusted for wiring overhead. Figure 10(d) gives the total area needed for the TCAMs and associated SRAMs. The total area is comparable for the three architectures. PC-TRIO and PC-DUOS+W have lower TCAM area (due to fewer TCAM entries) and higher SRAM area (due to wider SRAM words) than PC-DUOS+.

5) Update Performance: Figure 11 shows the average number of TCAM writes used per update on the datasets from [24]. PC-TRIO needs comparable number of writes as PC-DUOS+ and hence supports efficient and consistent incremental updates. PC-DUOS+W needs more writes than PC-TRIO to preserve the property that all rules stored in the

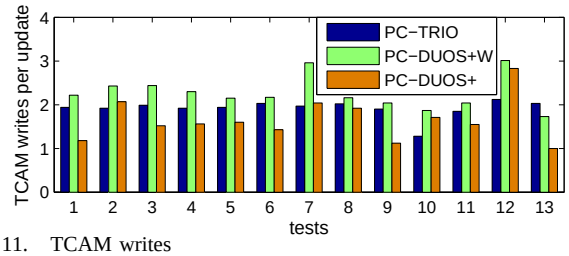

Fig. 11. TCAM writes

LTCAM have the highest priority compared to overlapping rules.

6) Characteristics of the logic that processes wide SRAM words: A circuit designed to process the contents of a wide LSRAM word was synthesized using a 0.18 μm library [26], [27] and it was found found that the design successfully met the timing constraints with a 500MHz clock. The results are

Process	Time (ns)	Throughput (Msps)	Voltage (V)	Power (mW)	Gate Count
0.18μm	2	500	1.8	61.13	59724

Fig. 12. Timing and power results for additional hardware

presented in the Figure 12. The throughput is represented in terms of million searches per second (Msps). An example of a TCAM with a speed of 143MHz (effectively, 143 Msps) is found in [28], using 0.13μm technology. It is expected that the delay overhead and throughput of our design will improve on using a 0.13μm library. Thus, our design can operate at the same speed as that of a TCAM.

V. Conclusion

We presented an indexed TCAM architecture, PC-TRIO, for packet classifiers. The methods to add indexing and wide SRAMs were applied on PC-DUOS+ [24] to obtain another indexed TCAM architecture PC-DUOS+W. These two architectures were then compared with PC-DUOS+. Both PC-TRIO and PC-DUOS+W may be updated incrementally. The average improvement in TCAM power and lookup time using PC-TRIO were 96% and 98%, respectively, while that using PC-DUOS+W were 71% and 76%, respectively, relative to PC-DUOS+.

PC-DUOS+W performed better on the ACL datasets compared to the other types of classifiers. There was 86% reduction in TCAM power, and 98% reduction in lookup time with PC-DUOS+W on the ACL datasets on an average compared to PC-DUOS+. Even though PC-DUOS+W lookup performance was better than that of PC-TRIO on three ACL tests, PC-TRIO lookup performance was quite reasonable and in fact, using PC-TRIO, there was a reduction in TCAM power by 94% and lookup time by 97% on an average for the ACL tests, compared to PC-DUOS+.

So, we recommend PC-TRIO for packet classifiers.

References

[1] K. Lakshminarayan, A. Rangarajan and S. Venkatachary, Algorithms for Advanced Packet Classification with Ternary CAMs, *SIGCOMM*, 2005.

[2] F. Zane, G. Narlikar and A. Basu, CoolCAMs: Power-Efficient TCAMs for Forwarding Engines, *INFOCOM*, 2003.

[3] W. Lu and S. Sahni, Low Power TCAMs For Very Large Forwarding Tables, *INFOCOM*, 2008.

[4] R. Draves, C. King, S. Venkatachary, and B.Zill, Constructing Optimal IP Routing Tables, *INFOCOM*, 1999.

[5] D. E. Taylor and J. S. Turner, ClassBench: A Packet Classification Benchmark, *TON*, 15, 3, Jun 2007, 499-511.

[6] H. Che, Z. Wang, K. Zheng and B. Liu, DRES: Dynamic Range Encoding Scheme for TCAM Coprocessors, *TOC* 57, 7, Jul 2008, 902-915.

[7] A. Bremler-Barr, D. Hay and D. Hendler, Layered Interval Codes for TCAM-based Classification, *INFOCOM* 2009.

[8] H. Song and J. Turner, Fast Filter Updates for Packet Classification using TCAM, Routing Table Compaction in Ternary-CAM, *GLOBECOM*, 2006

[9] D. Pao, P. Zhou, B. Liu, and X. Zhang, Enhanced Prefix Inclusion Coding Filter-Encoding Algorithm for Packet Classification with Ternary Content Addressable Memory, *Computers & Digital Techniques, IET*, 1, 5, Sep 2007, 572-580.

[10] S. Suri, T. Sandholm and P. Warkhede, Compressing Two-Dimensional Routing Tables, *Algorithmica*, 35, 4, 2003, 287-300.

[11] E. Spitznagel, D. Taylor, and J. Turner, Packet Classification Using Extended TCAMs, *ICNP*, 2003, 120-131.

[12] C. R. Meiners, A. X. Liu, and E. Torng, TCAM Razor: A Systematic Approach Towards Minimizing Packet Classifiers in TCAMs, *ICNP*, 2007, 266-275.

[13] H. Liu, Efficient Mapping of Range Classifier into Ternary-CAM, *Hot Interconnects*, 2002, 95-100.

[14] A. X. Liu, C. R. Meiners, and Y. Zhou, All-Match Based Complete Redundancy Removal for Packet Classifiers in TCAMs, *INFOCOM*, 2008, 574-582.

[15] Q. Dong, S. Banerjee, J. Wang, D. Agrawal, and A. Shukla, Packet Classifiers in Ternary CAMs can be Smaller, *SIGMETRICS*, 2006, 311-322.

[16] J. van Lunteren and T. Engbersen, Fast and Scalable Packet Classification, *IJSAC*, 21, 4, May 2003, 560-571.

[17] B. Agrawal and T. Sherwood, Ternary CAM Power and Delay Model: Extensions and Uses, *TVLSI*, 16, 5, May 2008, 554-564.

[18] O. Rottenstreich and I. Keslassy, Worst-Case TCAM Rule Expansion, *INFOCOM*, 2010.

[19] T. Mishra and S.Sahni, PETCAM – A Power Efficient TCAM For Forwarding Tables, *http://www.cise.ufl.edu/~sahni/papers/petcam.pdf*,

[20] T. Mishra and S.Sahni, DUO – Dual TCAM Architecture for Routing Tables with Incremental Update, *http://www.cise.ufl.edu/~sahni/papers/duo.pdf*

[21] T. Mishra, S.Sahni, and G. Seetharaman, PC-DUOS: Fast TCAM Lookup and Update for Packet Classifiers, *ISCC*, 2011.

[22] Z. Wang, H. Che, M. Kumar, and S.K. Das, CoPTUA: Consistent Policy Table Update Algorithm for TCAM without Locking, *TOC*, 53, 12, Dec 2004, 1602-1614.

[23] T. Mishra and S. Sahni, CONSIST - Consistent Internet Route Updates *ISCC*, 2010.

[24] T. Mishra, S. Sahni and G. Seetharaman, PC-DUOS+: A TCAM Architecture for Packet Classifiers *http://www.cise.ufl.edu/~sahni/papers/pcduos+.pdf*

[25] N. Muralimanohar, R. Balasubramonian and N. P. Jouppi, Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0, *ISM* December 2007, 3-14

[26] J. B. Sulistyo, J. Perry and D. S. Ha, Developing Standard Cells for TSMC 0.25um Technology under MOSIS DEEP Rules, *Virginia Tech, Technical Report VISC-2003-01* Nov 2003.

[27] J. B. Sulistyo and D. S. Ha, A New Characterization Method for Delay and Power Dissipation of Standard Library Cells, *VLSI Design* 15, 3, Jan 2002, 667-678.

[28] H. Noda, K. Inoue, M. Kuroiwa, F. Igaue and K. Yamamoto, A Cost-Efficient High-Performance Dynamic TCAM With Pipelined Hierarchical Searching and Shift Redundancy Architecture, *IJSSC*, 40, 1, Jan 2005, 245-253.

[29] Renesas R8A20410BG 20Mb Quad Search Full Ternary CAM. http://am.renesas.com/products/memory/TCAM/tcam_root.jsp.