PC-TRIO: A Power Efficient TCAM Architecture for Packet Classifiers

Tania Banerjee and Sartaj Sahni,
Department of Computer and Information Science and Engineering,
University of Florida, Gainesville, FL 32611
{tmishra, sahni}@cise.ufl.edu
Gunasekaran Seetharaman, AFRL, Rome, NY, USA
Gunasekaran.Seetharaman@rl.af.mil

***Abstract*—PC-TRIO is an indexed TCAM architecture for packet classification. In addition to index TCAMs, PC-TRIO uses wide SRAM words. On our packet classifier datasets, PC-TRIO reduced TCAM power by 96% and lookup time by 98% on an average, compared to PC-DUOS+ [28] that does not use indexing or wide SRAMs. PC-DUOS+ was shown to be better than STCAM, which is a single TCAM architecture conventionally used for packet classification [28]. In this paper, we also 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+, which demonstrates the effectiveness of indexing and usage of wide SRAMs in reducing power and lookup time for packet classifiers.**

***Index Terms*—Packet classifier, incremental updates, power, TCAM.**

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.

This material is based upon work funded by AFRL, under AFRL Contract No. FA8750-10-1-0236. (b) Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of AFRL. Approved for public release: (88ABW-2011-4143)

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 [19]. 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.

TCAMs are augmented with wide SRAMs and index TCAMs for scalability and low power consumption. These techniques are simple and have produced significant benefits in packet forwarding. We extend the applicability of these techniques to packet classification, in this paper. Packet classification is a harder problem than packet forwarding because each rule in a classifier typically consists of five fields, namely, source, destination, protocol, source port and destination port, and a corresponding action. In contrast, a rule in a packet forwarding table consists of only one field, that is destination prefix, and a corresponding next hop. The difficulties in using an indexed TCAM architecture in packet classification are discussed below. We also discuss ways to overcome these difficulties and contributions of PC-TRIO.

A. Problems in storing a classifier in an indexed TCAM

There are two problems in mapping a packet classifier to an indexed TCAM architecture. The first problem is that the overlapping of rules may lead a higher priority rule to be stored in a bucket of entries corresponding to a lower priority index. This is more likely to happen when we have TCAM buckets of fixed sizes.

The second problem is the need to include a $covering\ rule$ in each TCAM bucket which introduces redundancy. 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. However, the redundancy so introduced, in the face of range expansion, could lead to a huge increase in the number of rules to be stored in the TCAM. The following example illustrates these points.

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*

	Filter		Action	Priority
R#	Destination	Source		
R1	00*	1000	A1	1
R2	0*	0101	A2	2
R3	000*	01*	A3	3
R4	*	*	A4	4

Fig. 1. An example classifier

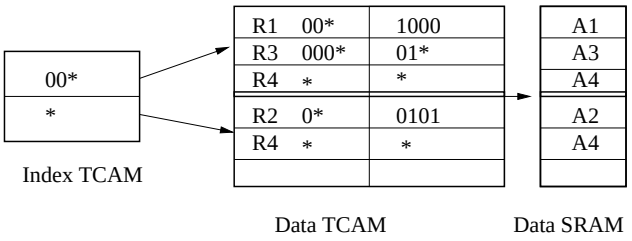


Fig. 2. Classifier rules stored in a indexed TCAM

data TCAM has two buckets and in this setup, the index TCAM uses bits from the destination prefix of each rule to index into the buckets of the data TCAM. 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 R3 - 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, R2 is the best matching rule and A2 is the desired action.

Rule R4 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.

B. Overcoming these problems

If we have a set of independent rules, then both the problems described above are solved. Two rules are $independent$ iff no packet is matched by both the rules. Independent rules may be placed anywhere in the TCAM since a lookup matches only one rule and priority resolution is not needed. So a TCAM storing only independent rules does not need a priority encoder. Further, covering rules are not needed. Thus, given a set of independent rules, we can easily identify the indexes without messing up the priority ordering. This helps us in adding wide SRAMs as well as index TCAMs and we present PC-TRIO which is an indexed TCAM architecture for packet classification. To the best of our knowledge, PC-TRIO is the first indexed TCAM architecture for packet classification.

C. Contribution

The main contributions in this work are as follows. Firstly, we now have a low power TCAM architecture for packet classification. In fact, power consumption in PC-TRIO is up to 1/30 th of the power consumed by an STCAM which is a conventional TCAM with port range expansion. The reduction in power consumption is possible because of selectively activating only a portion of the TCAM during lookup. Further, with the use of wide SRAMs, we are able to reduce the port range expansion problem. The start and end points of a port are now stored in the SRAM whenever possible. During lookup, the content of the wide SRAM word is processed by a specialized and fast hardware. Secondly, PC-TRIO has faster lookup times and hence higher throughput. PC-TRIO completes a lookup in at least 1/4th the time taken by PC-DUOS+ [28], and PC-DUOS+ itself takes about half the time taken by a search in a STCAM. Thirdly, PC-TRIO is highly scalable. With the rapid growth of the Internet, one may expect more and more filter rules to be added to packet classifiers. PC-TRIO, with its use of wide SRAMs can store a larger number of classifier rules in a given TCAM, and has a low growth index. Finally, PC-TRIO makes it possible to implement fast and incremental TCAM updates. In fact, for a given insert or delete rule request, PC-TRIO takes two TCAM writes on an average to update the TCAM.

The rest of the 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 begin with a description of the related work. Section II-A presents the research on TCAM based packet classifiers. Section II-B describes the existing indexed TCAM architectures for packet forwarding tables. We discuss the challenges in using indexed TCAM architecture for packet classification in Section I-A. Section I-B shows how these challenges can be tackled.

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. A common goal in these approaches is to store a classifier rule in a single TCAM entry. The schemes in [1], [6], [7], [9], [14], [17] 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. PC-TRIO, with the aid of wide SRAMs, can store multiple classifier rules in a single TCAM entry. The common bits in these rules are stored in the TCAM entry, while the remaining bits in each rule, including the source and destination ports, are stored in the corresponding wide SRAM entry. Thus, PC-TRIO obtains considerable savings of TCAM power. Additionally, since TCAM lookup time is proportional to the number of TCAM entries, PC-TRIO supports faster lookups.

Liu et al. [13], approach the range expansion problem by splitting a d-dimensional classifier into multiple lower dimensional classifiers each of which can be stored in its own small TCAM, leading to effective reduction of the range expansion problem. The algorithm effectively contains the range expansion problem, reduces power, improves the lookup time and supports batch updates. The power reduction achieved is between 35.5% and 89.6% and throughput increased between 1.34 and 5.89 times compared to TCAMs implementing direct range expansion, which we call STCAM here. PC-TRIO obtains a power reduction between 56.5% and 96.6% and throughput increased between 3.92 and 149.92 times compared to PC-DUOS+ [28]. Power consumption in PC-DUOS+ is comparable to STCAM, but PC-DUOS+ has up to twice the throughput of a STCAM.

Compressing packet classifiers by removing redundancies is an effective strategy to reduce TCAM power consumption. The approaches in [4], [10], [12], [15], [16], [21] present algorithms that transform an input classifier to an equivalent smaller classifier. These algorithms quite naturally contain port range expansions. The equivalently smaller classifier can be stored in PC-TRIO to obtain further savings in power and lookup time. While the compression techniques bring about 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.

Ma and Banerjee introduced index TCAMs to store a pre-classifier in SmartPC [20]. The pre-classifier in SmartPC is the index that classifies each incoming packet based on two header fields (source and destination) only, at the first stage. A major problem with the pre-classifier construction heuristic is the assumption that each classifier rule occupies one TCAM entry in a block. This assumption points to the fact that SmartPC completely ignores the range expansion problem and hence is unusable in its current form for packet classification using TCAMs. If the problem in SmartPC is corrected, it is still very different than PC-TRIO, since PC-TRIO uses wide SRAMs, and PC-TRIO has a more advanced mechanism for identifying the first stage indexes, that does not restrict the number of dimensions, as SmartPC does to two. Moreover, the incremental update strategy of SmartPC results in new rules being added to the general TCAM for most of the time. This increases the number of general TCAM blocks and hence degrades the power benefits over time since the general blocks are always activated on lookup.

Fast updates for packet classifiers is gradually becoming important, and there have been a number of approaches to implement fast incremental updates. Song and Turner [8] describe an algorithm for fast updates in which 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 computed block number 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 [24] and PC-DUOS+ [28] use dual TCAMs for representation and incremental update of classifiers. PC-DUOS+ improves lookup performance too, although power consumption is the same as STCAM storing the classifier rules after a direct expansion of the port ranges. PC-TRIO is different from PC-DUOS+ in terms of both TCAM architecture and associated algorithms. PC-DUOS+ uses two TCAMs whereas PC-TRIO uses five TCAMs and four wide SRAMs. Both PC-DUOS+ and PC-TRIO identify independent classifier rules. However, PC-DUOS+ uses a directed graph representation for the rules to identify a single set of independent rules, whereas PC-TRIO uses a multi-dimensional trie representation to identify two sets of independent rules. Overall, compared to PC-DUOS+, PC-TRIO uses indexing, wide SRAMs and different algorithms and data structures to find independent rules, which further improves lookup performance and significantly reduces TCAM power consumption.

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

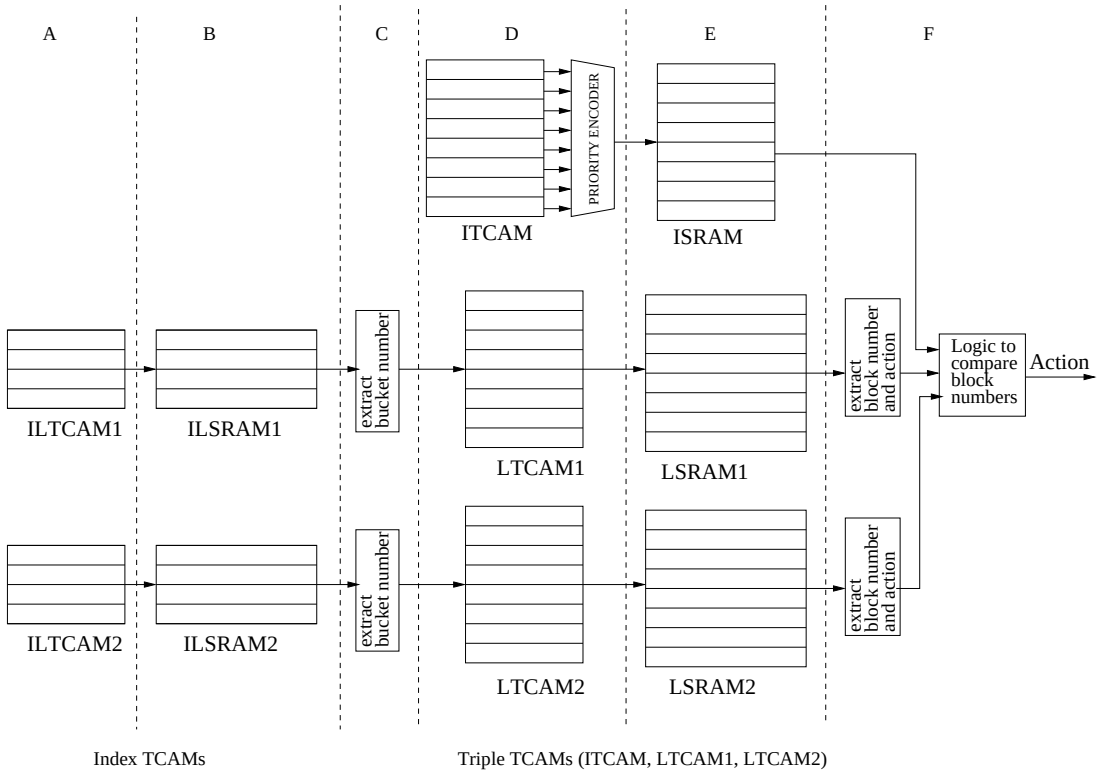


Fig. 3. PC-TRIO Architecture

[22] and DUO [23] obtained further reduction in power and TCAM space for packet forwarding, using the indexing and wide SRAM schemes. In particular, DUO [23] 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 [25], [26].

III. PC-TRIO

PC-TRIO is a comprehensive TCAM based solution for low power consumption and fast lookup in packet classification, and includes an architecture and associated algorithms. The TCAM architecture enables parallel and pipelined access for faster lookups and power savings. The algorithms help to fill in the various TCAMs in the system to ensure correct results, and also to efficiently update the TCAMs when the classifier is updated.

Given a classifier, the first step in PC-TRIO is to partition the rule set to eliminate overlap among the rules. In particular, PC-TRIO partitions the rule set into three subsets, out of which two subsets contain non-overlapping or independent rules and the third subset contains the remaining rules. The heuristic presented in the paper for partitioning the rule set uses a multi-dimensional trie to represent classifier rules with the five standard fields, namely, prefixes to source and destination addresses, protocol type, source port and destination port. A multi-dimensional trie is a scalable, space efficient representation of the classifier rules [27], [30]. PC-TRIO uses tries in the control plane to fill in the TCAMs initially, and to perform subsequent updates. To generate the first subset of independent rules, a multi-dimensional trie is constructed with the source prefix as the first dimension and our heuristic is applied. To obtain the second subset of independent rules, another multi-dimensional trie is constructed using the remaining rules, this time with the destination prefix as the first dimension, and the heuristic is applied again. Our partitioning strategy is based on the observation that most of the prefixes are very specific and do not overlap, irrespective of whether the prefixes represent a source or a destination. In fact, using our strategy, the number of rules in the third subset containing the remaining rules was only 5% on an average on our tests. So, even though the remaining rules may be used to generate more independent sets - making the number of overlapping rules even smaller, there is no tangible benefit in terms of power and performance than what is achieved with two independent sets. In fact, it may adversely affect power and performance, as we will soon see.

The second step in PC-TRIO is to store the three subsets of rules in the TCAM architecture as described in Section III-A. Each of the two independent subsets of rules, is stored in its own indexed TCAM and wide SRAM system. The third subset of remaining rules is stored in a regular TCAM-SRAM system, and not in an indexed TCAM or wide SRAM, due to the problems described in Section I-A. Thus the three subsets of rules are stored in three independent TCAM systems, that are looked up in parallel to search for a match. So, at the final stage, the architecture uses a logic circuit to compare the priority of the three rules matched from the three subsets and returns the action corresponding to the highest priority match.

In this context, we note that having more than two independent sets will incur higher hardware cost since indexed TCAM,

wide SRAM systems have to be added for the independent sets. This will result in higher power consumption and also decreased performance since the final stage logic to compare priorities will become a bottleneck.

The PC-TRIO architecture is presented in detail in Section III-A. The algorithms for storing the TCAMs using multi-dimensional tries and priority graphs are discussed in Section III-B, while the updating algorithms are discussed in Section III-C. The differences of PC-TRIO with related multi-TCAM 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 LSRAMs and index TCAMs. The wide LSRAMs store multiple actions and part of rules in a single word. Thus PC-TRIO needs additional hardware to extract block number (associated with priority of rule) and action corresponding to the matching rule(s) from a returned LSRAM word. The index TCAMs are labeled as ILTCAM1 and ILTCAM2 corresponding to LTCAM1 and LTCAM2, respectively. The index TCAMs also have wide associated SRAMs, namely, ILSRAM1 and ILSRAM2. These wide LSRAM words store bucket indexes to access the next level TCAMs, and bits from the indexes themselves. Thus, PC-TRIO employs additional hardware to extract bucket number from multiple entries in the wide SRAM word. The bucket number obtained is used to activate a specific bucket in the corresponding LTCAM. 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 2 stages, ABC and DEF marked in Figure 3. In the first stage, the ILTCAMs are searched (A), ILSRAMs are accessed using the address of the matching ILTCAM1 and ILTCAM2 entries (B), and the matching wide ILSRAM words are processed to obtain the corresponding bucket index for LTCAM1 and LTCAM2 (C). In second stage, the bucket indexes obtained at the first stage are used to search the corresponding buckets in the LTCAMs, the ITCAM is also searched (D). Then, the ISRAM and the LSRAMs are accessed using the addresses of the matching TCAM entries (E). Finally, the contents of the wide LSRAM words are processed and the best action is chosen from the three actions returned by the ISRAM, LSRAM1 and LSRAM2 by comparing the priorities of the corresponding rules (F). Though the two stage pipeline described above has higher latency, it has the same throughput compared to a single stage implementation. Such two stage pipeline architectures are common in packet forwarding and the associated latency is generally acceptable.

B. Storing rules in TCAMs

Given an input of packet classifier rules, various preprocessing steps are applied on the rules before storing the classifier in PC-TRIO. 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 and final step is to store the remaining rules in the ITCAM in priority order, which is discussed in Section III-B4.

1) Representing Classifier Rules:

a) Priority Graph: The classifier rules are represented in a priority graph, which 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.

Example: *Suppose a classifier has four two-field rules as shown in Figure 4. Figure 5 shows the priority graph for this classifier. Consider rules R1 and R4. These two rules overlap with each other and rule R1 is of higher priority compared to rule R4. Thus there is an edge between the vertices corresponding to rules R1 and R4 and the direction of the edge is from rule R1 to R4. On the other hand, there is no edge between the rules R1 and R2, because the destination prefix fields of these rules are non-overlapping. For example, R2 matches destination addresses with 1 on the first octet, whereas R1 matches those with 0 on the first octet. Thus, the sets of addresses matched by R1 and R2 are disjoint. In Figure 5, the block number of each rule is given by the side of the corresponding vertex inside a square. corresponding square.*

Index	Rule (Source, Destination)	Priority
R1	(0.0.0.0/0, 0.0.0.0/16)	1
R2	(1.0.0.0/8, 1.0.0.0/16)	2
R3	(0.0.0.0/8, 0.0.0.0/8)	3
R4	(0.0.0.0/0, 0.0.0.0/0)	4

Fig. 4. A classifier with four rules

b) Multi-dimensional Trie: A trie is a binary tree used to store prefixes. The left child of a node of a trie represents a 0 and the right child a 1. Figure 6 shows an example of a trie representing five prefixes, where, '*' represents a sequence of trailing don't care bits. We need a multi-dimensional trie to represent a classifier rule, with each field being represented by

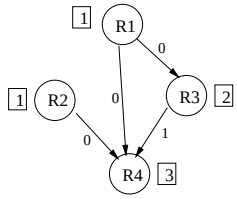


Fig. 5. A priority graph

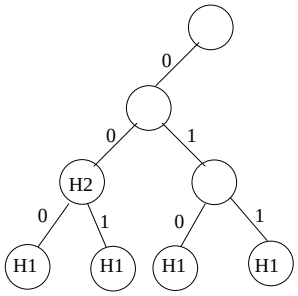


Fig. 6. Trie for example prefix set $C = \{000*, 001*, 010*, 011*, 00*\}$. Packet with address 000* is forwarded to next hop $H1$.

a trie. An example of a multi-dimensional classifier is given below.

Example: *Consider a two dimensional classifier with five rules described as follows: The corresponding multi-*

R#	Rule (Source, Destination)	Priority
R1	(*, 00*)	1
R2	(*, 01*)	2
R3	(010*, *)	3
R4	(11*, *)	4
R5	(*, *)	5

Fig. 7. A classifier with five rules

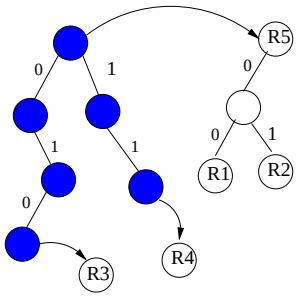


Fig. 8. Multi-dimensional trie, with source as the outermost trie

dimensional trie is illustrated in Figure 8. In this Figure, the outermost trie (whose nodes are colored blue) represents the source prefixes of the rules, while the next level tries represent the destination prefixes. The branching out of a next level trie from a node is represented by arrows in the figure. The next level trie nodes that represent the rules are marked with R#.

c) Leaves of leaves set: One of the very first challenges in designing PC-TRIO is identifying the maximum number of rules that are pairwise independent in a classifier. This is an NP-complete problem which can be easily proved by reducing

```
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 and right child.
        if (node contains root of a next-level 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. 9. Selecting protocol nodes for leaves of leaves set

an instance of the problem of finding maximal independent set in a graph to the problem of finding the maximum number of independent rules.

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* [24] of protocol nodes in a multi-dimensional trie using the algorithm in Figure 9.

The nodes belonging to the leaves of leaves set in a trie are obtained by traversing the multi-dimensional trie from the root to the leaves of the source trie and then from these leaves into their attached destination trie and then from the leaves of the destination trie into the leaves of their attached innermost trie for the protocol field. The leaves of leaves set of rules corresponding to the multi-dimensional trie of Figure 8 is {R3, R4}. The rule R1 is not in the leaves of leaves set since its destination trie, branches out of a non-leaf node of the source trie. Similarly, R2 is not in the leaves of leaves set. Rule R5 does not even belong to a leaf of the destination trie, and of course, the destination trie itself branches out of a non-leaf node in the source trie.

Theorem 1: The rules in a leaves of leaves set of a multi dimensional trie are mutually independent.

Proof: Consider two rules R1 and R2 in the leaves of leaves set that are overlapping, i.e. not mutually independent of each other. In other words, there are some common addresses matched by both the rules R1 and R2. Note that, for two rules to be overlapping, all the fields in the rule must be overlapping as well [26].

Now, consider the outermost trie. The prefixes for the corresponding field in R1 and R2 trace up to leaf level of this trie, since R1 and R2 belong to the leaves of leaves set. There are two possibilities, either the leaf in the outermost trie traced by R1 is the same, or different than that traced by R2. If it is different, then the corresponding prefix fields are non-overlapping, and hence R1 and R2 are non-overlapping. But this is in contradiction with our selection for R1 and R2. So, both the rules branch out of the same leaf of the outermost trie.

This logic may, similarly, be applied to the next level trie and to all the other tries in sequence in the multi-dimensional trie. Then we get that all the fields of R1 and R2 are the same and so R1 and R2 cannot be different rules. In other words,

R1=R2. ■

The second step in finding a set of independent rules is needed for a packet classifier with source and destination port ranges. The leaves of leaves set built so far, has independent entries. However, it is possible that for each entry in the leaves of leaves set, there are more than one rule with the same source/destination/protocol prefixes, but different source and destination port ranges. Therefore, in this second step, we process each leaves of leaves set, and identify a set of independent rules. This is done, by building a small priority graph with rules only in the leaves of leaves set.

It is easy to check that vertices in the priority graph with in-degree 0 comprise a set of independent rules. If there were any two rules that overlap, the corresponding vertices will have an edge between them. In that case, one of the vertices will not satisfy the condition of in-degree = 0. Thus, vertices in the priority graph with in-degree 0 comprise a set of independent rules. The vertices corresponding to rules R1 and R2 in Figure 5 have in-degree 0.

d) Exploiting characteristics of a classifier: A set of classifier rules mainly consist of four classes of rules, namely, those with more specific source address prefixes, those with more specific destination address prefixes, those with more specific source and destination address prefixes, and those with both the prefixes as non-specific. A prefix is more specific when the prefix is long and hence includes a small set of addresses. An example of a very specific prefix is, 156.122.78.226/30, which is an IPv4 prefix of length 30, and includes only 4 addresses. In contrast, a non-specific prefix includes a large set of addresses. An extreme non-specific prefix is '*'. This observation helps us in identifying two sets of independent rules. The first set is obtained by constructing a multi-dimensional trie with the destination prefix as the outermost trie, while the second set is obtained by constructing another trie where source prefixes constitute the outermost trie.

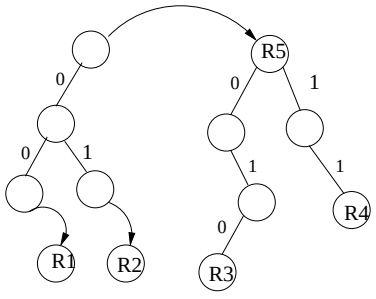


Fig. 10. Multi-dimensional trie, with destination as the outermost trie

Example: *Figure 10 represents a version of the multi-dimensional trie of Figure 8 in which the outermost trie is the destination trie. The leaves of leaves set in this trie is {R1, R2}. Note that the union of the leaves of leaves set, for the multi-dimensional tries of Figures 10 and 8, is {R1, R2, R3, R4}, which covers all the rules except R5.*

Thus, we construct two multi-dimensional tries for PC-TRIO. A multi-dimensional trie, Trie1, is first created in which 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. 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.

We identify a set of independent rules from Trie1 using the two step process that includes identifying the leaves of leaves set and building a priority graph. 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*>. Thus all rules are represented in Trie1, and only those rules that are not in the leaves of leaves set for Trie1 are included in the construction of Trie2. 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. The Trie1 and Trie2 for the rules of Figure 7 are given in Figures 11 and 12

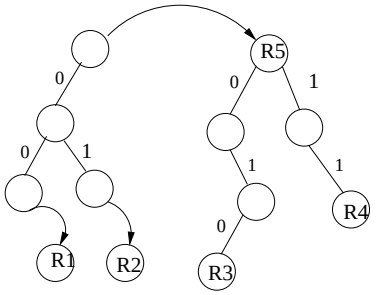


Fig. 11. Trie1 for classifier rules in Figure 7

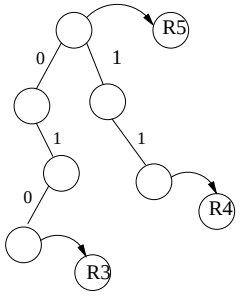


Fig. 12. Trie2 for classifier rules in Figure 7.

respectively. Note that the rules R1 and R2 are not included in Trie2. The rules in Trie1 and Trie2 are stored in LTCAM1 and LTCAM2 respectively.

2) Storing rules in the LTCAM1: So far, we have identified the independent rules to be stored in the LTCAM1. In this

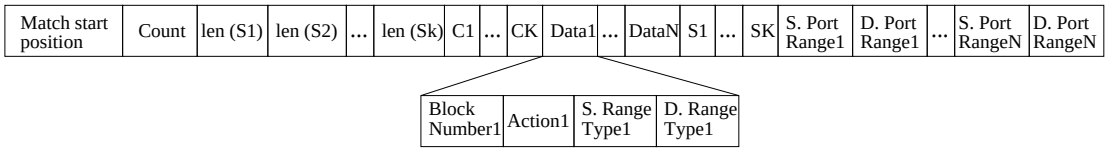


Fig. 13. Data encoding in a wide SRAM word

section we describe how these rules are stored in the LTCAM1 subsystem which comprises the LTCAM1 and LSRAM1, and the corresponding index structure ILTCAM1 and ILSRAM1. The method is identical for the LTCAM2 subsystem. The process of storing rules in the LTCAM1 subsystem is described in five subsections below. First, the format of storing information in a wide LSRAM word is discussed (Section III-B2a), then we describe the creation of LTCAM1 entries using the process of carving (Section III-B2b). Next we describe partial port range expansion (Section III-B2c) that may be necessary, and finally, the creation of ILTCAM1 and ILSRAM1 entries (Section III-B2d).

a) 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 13 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 word, a whole range is never entered and only one port number is entered for a range with the same start and end points.

b) Creating LTCAM1 entries: A trie is carved into subtries to assign rules to the wide SRAM words. Before carving takes place, we calculate the number of SRAM bits needed to store the rules in Trie1. The Trie1 is carved using the carving heuristic $visit\_postorder$ of DUO [23] that has been enhanced for multi-dimensional tries. The heuristic traverses the trie in post order manner, and whenever a subtrie is found that requires SRAM bits less than or equal to the size of an SRAM word, that subtrie is a potential candidate for carving. The carving happens if it creates an independent (disjoint) entry for LTCAM1. The path starting from the root of Trie1

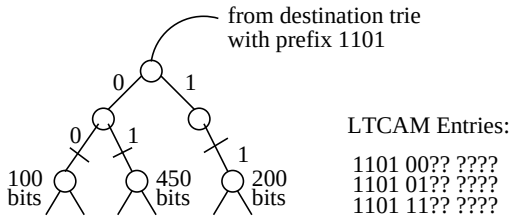


Fig. 14. Nodes in a source trie is being carved.

to the root of the subtrie defines an LTCAM1 entry.

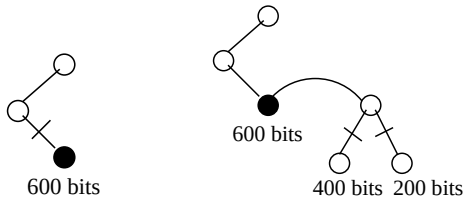
Example *Figure 14 shows a portion of a source trie that hangs off a destination trie, where carving takes place at nodes 00, 01, and 11 of the source trie. The path from the root to the node of the destination trie from which the source trie hangs off is 1101. Thus, after carving the node at 00 on the 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 14 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 13.*

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 [23].

c) 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 15(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 15(b), and carve nodes on this trie to



(a) a protocol node

(b) a new source port trie is attached to the protocol node

Fig. 15. 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 15(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.

d) 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

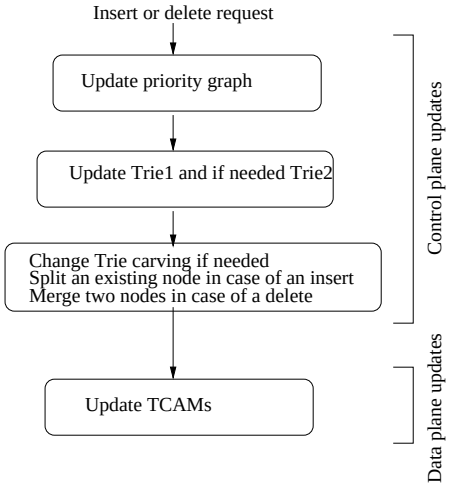


Fig. 16. Flow for incremental updates

Figure 16 gives the overall flow of the updates that take place when a request to insert or delete a rule is received. After an update request is issued, 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

Fig. 17. Sequence of development of TCAM architectures

	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. 18. Differences among the architectures

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 PC-DUOS+ [28] may be used. For the ITCAM of PC-TRIO, we implemented the DLFS_PLO (Distributed and Linked Free Space with Prefix Length Ordering) 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 PC-DUOS+ is relatively simple as all the rules in the LTCAM are independent so a new rule may be inserted anywhere in

the TCAM. However, we still need to locate a free slot. The LTCAM memory management algorithm 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 this memory management algorithm for its LTCAM1 and LTCAM2.

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

Figure 17 illustrates our sequence of development of various TCAM architectures for packet classification. We first developed PC-DUOS[24], which uses two TCAMs for packet classification, which was improved in PC-DUOS+[28] resulting is faster lookup times and updates. Both PC-DUOS and PC-DUOS+ partition the classifier into two subsets of rules - the first subset consisting of independent rules and the second subset containing the remaining rules. The two subsets are stored in two regular TCAMs. Next, PC-DUOS+ was extended to PC-DUOS+W by using an index TCAM and wide SRAM for the independent subset of rules. Following PC-DUOS+W, we developed PC-TRIO, in which the classifier is partitioned into three subsets. Figure 18 summarizes the differences among these architectures.

In addition to architectural and algorithmic differences from 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+ [28]. PC-DUOS+ was compared with STCAM (Single TCAM) which is conventionally used for packet classification. This comparison in [28] showed that PC-DUOS+ is superior to STCAM. Hence we preclude STCAM in our comparison with PC-TRIO. We first give the setup used by us for the experiments in Section IV-A and then describe our benchmarks 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++, and compared their performance on an x86 Linux box with a 64-bit, 1.2GHz CPU. It is difficult to get real life packet classifiers from ISPs, mainly due to security reasons. So, we generated test classifiers using ClassBench [5], which is a well known tool for generating synthetic classifiers and packet traces. The classifiers generated using ClassBench closely model real life packet classifiers. The three different types of classifiers modeled by ClassBench are access control lists, firewalls and IP chains. The 12 seed files included in ClassBench contain the basic parameters used to generate the classifiers of a specific type. Each generated rule has the traditional 5-field filter, namely, source address, destination address, source port range, destination port range, and protocol. We generated the test classifiers, by using the seed files and specifying the number of rules in each classifier.

Further, 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 [31] and a TCAM power and timing model [18] 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 [35].

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], [24].

B. Datasets

We used three benchmarks derived from ClassBench [5]. The first benchmark consists of large classifiers, each classifier containing about 100,000 rules, with one classifier for each of the 12 seed files in ClassBench. This benchmark is used to compare the number of TCAM entries, power, lookup performance and space requirements of PC-TRIO, PC-DUOS+W and PC-DUOS+ [28].

The second benchmark consists of medium sized classifiers, with each classifier containing about 20,000 rules. For each of the 12 seed files we generated 10 such datasets. Thus, this benchmark consists of 120 classifiers and is used to analyze the variance of PC-TRIO performance.

The third set of benchmarks was reused from [28]. There are 13 datasets here which are used to compare incremental update performance of PC-TRIO, with PC-DUOS+ [28] and PC-DUOS+W.

C. Results from the first benchmark comprising large classifiers

1) Number of TCAM entries: Using wide SRAM words to store portions of classifier rules, reduces the number of TCAM entries. Figure 19 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

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	1.50	21085	182	0.19	2.00
2	acl2	74298	101857	19421	31	1122.39	37491	19421	6.35	33.16	36593	18439	6.04	151.93
3	acl3	99468	131243	30859	40	1640.47	52632	30859	9.47	81.19	26823	1017	0.40	2.89
4	acl4	99334	127320	25189	39	1730.46	49912	25189	7.98	47.96	34034	6547	2.32	26.11
5	acl5	98117	105375	1535	32	2072.16	32932	1535	1.29	0.41	34993	2209	0.77	5.68
6	fw1	89356	142085	91473	43	2466.72	98425	91473	28.52	2318.82	26610	4864	1.60	16.71
7	fw2	96055	129249	27084	39	1543.76	43146	27084	8.90	86.77	22196	1494	0.53	3.63
8	fw3	80885	117731	39199	36	1007.04	51228	39199	11.99	215.81	26269	7479	2.38	30.69
9	fw4	84056	211403	116149	64	3182.03	131505	116149	35.46	2139.81	27617	4894	1.60	15.86
10	fw5	84013	111989	55650	34	930.94	65598	55650	17.00	616.09	22361	3454	1.15	9.72
11	ipc1	99198	112154	22165	34	1288.02	41920	22165	6.82	45.81	23894	567	0.26	2.10
12	ipc2	100000	100000	30133	30	784.69	47247	30133	9.23	114.37	20195	0	0.09	1.45

Fig. 19. Number of TCAM entries, ITCAM entries and TCAM power and lookup time in PC-DUOS+, PC-DUOS+W, PC-TRIO

Index	Dataset	PC-DUOS+				PC-TRIO				Improvement Ratio	
		Power (mW)		Lookup Time (ns)		Power (mW)		Lookup Time (ns)			
		Average	Std. Dev.	Average	Std. Dev.	Average	Std. Dev.	Average	Std. Dev.	Power	Time
1	acl1	5511.72	15.54	228.64	1.31	315.87	11.90	1.53	0.04	17.45	149.92
2	acl2	6018.23	46.56	145.75	2.45	434.11	23.18	1.40	0.04	13.86	103.90
3	acl3	5819.47	25.88	113.39	1.28	2140.47	45.20	22.60	1.08	2.72	5.02
4	acl4	5651.50	55.68	113.54	3.07	2460.07	80.46	28.95	2.46	2.30	3.92
5	acl5	3543.97	116.27	80.37	6.07	1395.74	71.70	9.92	1.33	2.54	8.10
6	fw1	6916.64	83.74	109.66	3.13	350.62	5.29	1.27	0.01	19.73	86.05
7	fw2	6104.80	26.05	147.12	2.47	201.82	0.95	1.22	0.00	30.25	120.77
8	fw3	5989.51	92.88	93.51	5.44	377.30	7.30	1.33	0.01	15.87	70.17
9	fw4	10212.47	72.43	209.17	2.88	781.97	12.46	2.63	0.07	13.06	79.48
10	fw5	5480.81	37.00	63.46	0.76	424.71	7.25	1.43	0.02	12.90	44.45
11	ipc1	5211.18	19.70	108.23	1.19	811.96	8.68	2.58	0.05	6.42	41.89
12	ipc2	4755.23	2.09	61.58	7.29	188.05	0.39	1.22	0.00	25.29	50.63

Fig. 20. Average power and lookup time for PC-DUOS+ and PC-TRIO

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 21(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 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-B2c). So all LTCAM entries in PC-DUOS+W and PC-TRIO were at most 72 bits.

2) Power: Figure 19 gives the TCAM power consumption during a lookup, while Figure 21(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 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.

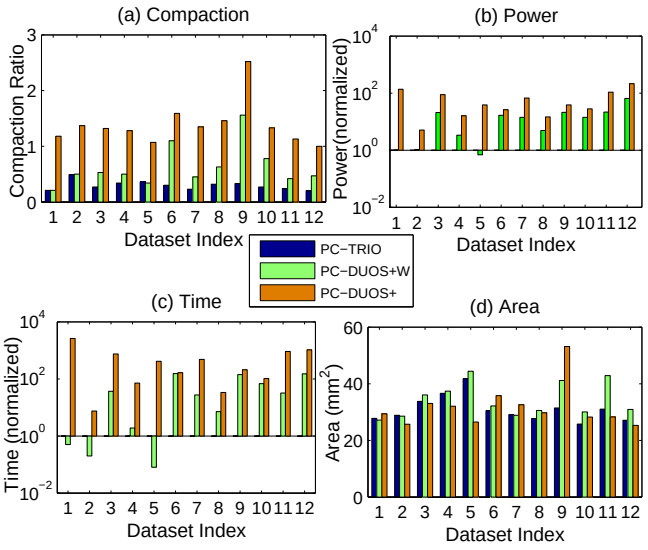


Fig. 21. Comparison of compaction ratio, total power, lookup time and area

3) Lookup Performance: Figure 21(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 19) 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 19). 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 [22], where area of a single cell is multiplied by the number of cells and then adjusted for wiring overhead. Figure 21(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+.

D. Results from the second dataset with medium classifiers

Figure 20 gives the average TCAM power and lookup times and their standard deviation for PC-DUOS+ and PC-TRIO. From the last couple of columns in this figure, we observe that power consumption in PC-TRIO is between 1/2.3 and 1/30 th of the power consumtion in PC-DUOS+. Lookup time in PC-TRIO is 3.92 to 149 times faster compared to that in PC-DUOS+. The least power saving was for the access control list test, acl4, while the most savings were obtained for the firewall test, fw2. Similarly, acl4 has the least improvement in lookup time, whereas acl1 has the maximum improvement.

The performance of PC-DUOS+ or PC-TRIO ultimately depends on how many rules are stored in the ITCAM, since that's the TCAM that cannot be optimized. For example, PC-DUOS+ had about 6000 rules in the ITCAM for acl4, whereas PC-TRIO had more than 9000. This is not surprising since PC-DUOS+ and PC-TRIO use different algorithms to extract independent rules. Despite having a larger ITCAM, PC-TRIO shows improvement in power and lookup time because of indexing.

The standard deviation for power and lookup time in both PC-DUOS+ and PC-TRIO is small, which shows that the individual power and timing numbers are very close to the average.

E. Results from the third benchmark with small to medium sized classifiers

The classifiers here are being reused from the work on PC-DUOS+ [28] to check the update performance of PC-TRIO in comparison with PC-DUOS+ and PC-DUOS+W.

1) Update Performance: Figure 22 shows the average number of TCAM writes used per update. PC-TRIO needs about 2

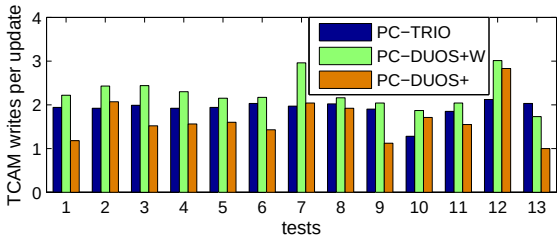


Fig. 22. TCAM writes

TCAM writes on an average and is comparable to PC-DUOS+ for updates. PC-TRIO 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 LTCAM have the highest priority compared to overlapping rules.

F. 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 [32], [33] and it was found found that the design successfully met the timing constraints with a 500MHz clock. The results are presented

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

Fig. 23. Timing and power results for additional hardware

in the Figure 23. 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 [34], 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+ [28] 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] C. R. Meiners, A. X. Liu, E. Torng, and J. Patel, SPliT: Optimizing Space, Power, and Throughput for TCAM-Based Classification, *ANCS* 2011.

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

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

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

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

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

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

[20] Y. Ma and S. Banerjee, A Smart Pre-Classifer to Reduce Power Consumption of TCAMs for Multi-dimensional Packet Classifcation, *SIGCOMM*, 2012.

[21] C. R. Meiners, A. X. Liu, and E. Torng, Topological Transformation Approaches to Optimizing TCAM-Based Packet Classification Systems, *SIGMETRICS/Performance*, 2009.

[22] T. Mishra and S.Sahni, PETCAM – A Power Efficient TCAM For Forwarding Tables, *IEEE Transactions on Computers*, Volume 61, No. 1, January 2012, 3-15

[23] T. Mishra and S.Sahni, DUO – Dual TCAM Architecture for Routing Tables with Incremental Update, Green TCAM-based Internet Routers, *Handbook of Energy-Aware and Green Computing*, Chapman-Hall/CRC Press, 2011.

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

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

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

[27] K. S. Kim and S. Sahni, Efficient Construction of Pipelined Multibit-Trie Router-Tables, *IEEE/ACM Transactions on Networking*, 11, 4, 2003, 650-662.

[28] T. Mishra, S. Sahni and G. Seetharaman, PC-DUOS+: A TCAM Architecture for Packet Classifiers, *IEEE Transactions on Computers*, To appear.

[29] T. Mishra, S. Sahni and G. Seetharaman, PC-TRIO: An Indexed TCAM Architecture for Packet Classifiers, *ISCC*, 2012.

[30] W. Lu and S. Sahni, Packet classification using space efficient pipelined multibit tries, *IEEE Transactions on Computers*, 57, 5, 2008, 591-605.

[31] 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

[32] 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.

[33] 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.

[34] 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.

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

Tania Banerjee-Mishra received Ph.D. in Computer Science from University of Florida in 2012. She did her Integrated M.Sc. in Mathematics and M.Tech in CSDP from IIT, Kharagpur. She is currently a Post Doc at University of Florida.

Sartaj Sahni is a Distinguished Professor and Chair of Computer and Information Sciences and Engineering at the University of Florida. He is also a member of the European Academy of Sciences, a Fellow of IEEE, ACM, AAAS, and Minnesota Supercomputer Institute, and a Distinguished Alumnus of the Indian Institute of Technology, Kanpur. In 1997, he was awarded the IEEE Computer Society Taylor L. Booth Education Award "for contributions to Computer Science and Engineering education in the areas of data structures, algorithms, and parallel algorithms", and in 2003, he was awarded the IEEE Computer Society W. Wallace McDowell Award "for contributions to the theory of NP-hard and NP-complete problems". Dr. Sahni was awarded the 2003 ACM Karl Karlstrom Outstanding Educator Award for "outstanding contributions to computing education through inspired teaching, development of courses and curricula for distance education, contributions to professional societies, and authoring significant textbooks in several areas including discrete mathematics, data structures, algorithms, and parallel and distributed computing." Dr. Sahni has published over three hundred research papers and written 15 texts. His research publications are on the design and analysis of efficient algorithms, parallel computing, interconnection networks, design automation, and medical algorithms.