PC-DUOS+: A TCAM Architecture for Packet Classifiers

Tania Banerjee-Mishra 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
Gunasekaran.Seetharaman@rl.af.mil

***Abstract*—We propose algorithms for distributing the classifier rules to two TCAMs (ternary content addressable memories) and for incrementally updating the TCAMs. The performance of our scheme is compared against the prevalent scheme of storing classifier rules in a single TCAM in priority order. Our scheme results in an improvement in average lookup speed by up to 49% and an improvement in update performance by up to 3.84 times in terms of the number of TCAM writes.**

***Index Terms*—Packet classifiers, TCAM, updates.**

I. Introduction

Internet packets are classified into different flows based on the packet header fields. This classification of packets is done 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 filter in the classifier, the corresponding 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, Transmission Control Protocol (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 corresponding to 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 costs.

[4], [5] survey the many solutions that have been proposed for packet classifiers. Among these, TCAMs have widely been used for packet classification as they support high speed lookups and are simple to use. Each bit of a TCAM may be set to one of the three states 0, 1, and x (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 whereas 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 in one TCAM cycle.

We present a TCAM architecture, update algorithms and a TCAM lookup mechanism in this paper for packet classifiers. The two main contributions of our method are fast TCAM lookup for packet classifiers and an advanced incremental update scheme that requires few TCAM writes per update. The fast lookup scheme is enabled by exploiting a TCAM without a priority encoder to store selected rules in a classifier. TCAMs without an encoder reduce TCAM search latencies by 50%[1]. The incremental update scheme uses a novel TCAM rule insertion algorithm that either shifts the overlapping rules with lower priority (compared to a new rule) downwards or those with higher priority upwards to allot a slot for the new rule, assuming TCAM addresses increase downwards. Conventionally, the overlapping rules with lower priority are shifted downwards in the TCAM for packet classifier and forwarding table updates. The new insertion scheme reduces the number of TCAM writes by up to 52% on our tests, compared to the conventional scheme. Further, this algorithm may be used with any TCAM architecture for packet classifiers, and is not limited to PC-DUOS+.

We begin in Section II by reviewing the background and related work. In Section III we describe our scheme of storing packet classifiers in TCAMs. An experimental evaluation of our scheme is done in Section IV and we conclude in Section V.

II. Background and Related Work

We first describe the PC-DUOS+ architecture in Section II-A and then present the related work in the field of TCAM based packet classifiers in Section II-B. Section II-C describes a simple TCAM and how classifier rules are stored, whereas Section II-D describes the differences between PC-DUOS+ and PC-DUOS.

A. The Architecture

DUOS [9] is a dual-TCAM architecture with simple SRAM (word size of 32 bits) used for packet forwarding. The same basic architecture is used in PC-DUOS (Packet Classifier - DUOS) [15], and in PC-DUOS+, which is an extension of

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

PC-DUOS. The TCAM architecture used in DUOS is shown in Figure 1. There are two TCAMs, labeled as the ITCAM (Interior TCAM) and the LTCAM (Leaf TCAM). DUOS also employs a binary trie in the control plane of the router to represent the prefixes in the forwarding table. The prefixes found in the leaf nodes of the trie are stored in the LTCAM, and the remaining prefixes are stored in the ITCAM. The prefixes stored in the LTCAM are independent and therefore at most one LTCAM prefix can match a specified destination address. Hence the LTCAM doesn't need a priority encoder. Prefix lookup works in parallel on both the TCAMs. If a match is found in the LTCAM then that is guaranteed to be the longest matching prefix and the corresponding next hop is returned. At the same time the ongoing lookup process on the ITCAM (which takes longer due to the priority resolution step) is aborted. Thus, if a match is found on the LTCAM, the overall lookup time is shortened by about 50% [1]. The logic on the final stage in Figure 1 that chooses between the two next hops could be moved ahead and placed between the TCAM and SRAM stages. In that case, the logic receives one "matching index" input from the LTCAM and another from the ITCAM. If a match is found in the LTCAM, the index from LTCAM input is used to access the LSRAM, otherwise, the ITCAM index is used to access the ISRAM. Further, if a match is found in the LTCAM, the ITCAM lookup is aborted.

The memory management schemes used in DUOS are highly efficient. The ITCAM needs to store the prefixes in decreasing order of length, for example, so that the first matching prefix is also the longest matching prefix. DUOS [9] uses a memory management scheme (Scheme 3, also known as DLFS_PLO - Distributed and Linked Free Space with Prefix Length Ordering), which initially distributes the free space

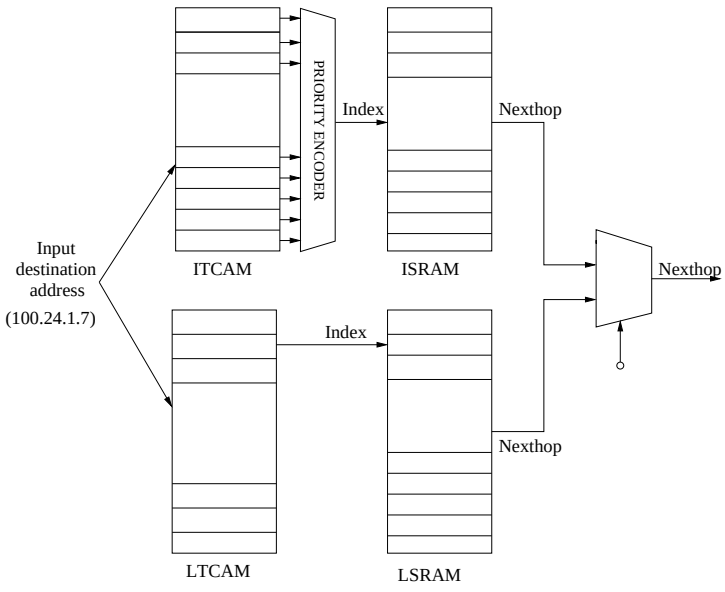


Fig. 1. Dual TCAM Architecture

available in a TCAM between blocks of prefixes (of same length) in proportion to the number of prefixes in a block. A free slot needed to add a new prefix is moved from a location that requires the minimum number of moves. As a prefix is deleted, the freed slot is added to a list of free spaces for that prefix block. Each prefix block has its own list of free slots. With this scheme even with 99% prefix occupancy in the TCAM and 1% free space, the total number of prefix moves using DLFS_PLO is at most 0.7% of the total number of prefix inserts and deletes.

To support lock-free updates, so the TCAMs can be updated without locking them from lookups, DUOS implements consistent update operations that rule out incorrect matches or erroneous next hops during lookup. For consistent updates, it is assumed that:

1) Each TCAM has two ports, which can be used to simultaneously access the TCAM from the control plane and the data plane.
2) Each TCAM entry/slot is tagged with a valid bit, that is set to 1 if the content for the entry is valid, and to 0 otherwise. A TCAM lookup engages only those slots whose valid bit is 1. The TCAM slots engaged in a lookup are determined at the start of a lookup to be those slots whose valid bits are 1 at that time. Changing a valid bit from 1 to 0 during a data plane lookup does not disengage that slot from the ongoing lookup. Similarly, changing a valid bit from 0 to 1 during a data plane lookup does not engage that slot until the next lookup.

Additionally, the availability of the function $waitWriteValidate$ is assumed which writes to a TCAM slot and sets the valid bit to 1. In case the TCAM slot being written to is the subject of an ongoing data plane lookup, the write is delayed till this lookup completes. During the write, the TCAM slot being written to is excluded from data plane lookups. Similarly, the availability of the function $invalidateWaitWrite$, is assumed. This function sets the valid bit of a TCAM slot to 0 and then writes an address to the associated SRAM word in such a way that the outcome of the ongoing lookup is unaffected. All these assumptions for DUOS are also made by our PC-DUOS and PC-DUOS+ architectures.

B. Existing Work on Packet Classifiers in TCAMs

When packet classifiers are stored in a TCAM, the main attributes for optimization are TCAM space and power consumption, updating mechanism and lookup performance. A classifier rule often needs more than one TCAM entry to represent port ranges, and this is called the port range expansion problem. Various approaches have been proposed in the literature to alleviate the range expansion problem. The schemes in [3], [14], [12], [17], [18], [19], [26] 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. [20] 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.

A reduction of port range expansion helps to reduce TCAM power consumption. Other strategies for TCAM power reduction are compressing packet classifiers by removing redundancies is an effective strategy to reduce TCAM power

consumption. The approaches in [21], [22], [25], [23], [24] 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.

The problem of incorporating updates to packet classifiers stored in TCAMs has been studied in [6] and [2]. The authors in [6] present a method for consistent updates when the classifier updates arrive in a batch. All deletes in an update batch are first performed to create empty slots in the TCAM. Then the relative priority of the relevant rules (for example rules overlapping with a new rule being inserted) is determined and the existing rules are moved accordingly to reflect any change in priority ordering as the entire batch of updates is applied. Following the ordering of existing rules, new rules are inserted in appropriate locations. A problem with the algorithm of [6] is that it performs the deletes in the update batch first. This could lead to temporary inconsistencies in lookup [10].

Given a packet classifier, a naive approach is to store it in a TCAM by entering each rule sequentially as they appear in the classifier and distribute all the empty slots between rules. As mentioned in [2], this approach could lead to high power consumption during look as the whole TCAM has to be searched including the empty entries. On the other hand, if the empty entries are kept together at the higher addresses of the TCAM, then those may be excluded from lookups. However, if the empty spaces are kept at one end of the TCAM, then it would require a large number of rule moves to create an empty slot at a given location. Specifically, all the rules in the TCAM, below the slot to be emptied must be moved below.

Song and Turner [2] describe a fast TCAM update scheme on packet classifiers. In their method, the classifier rules are entered arbitrarily in the TCAM and are not arranged according to decreasing order of priority. They ensure that the action corresponding to the highest priority matching rule is returned by performing multiple searches on the TCAM. Specifically, they assign a priority (which we call block number here) to each rule and encode the block number as a TCAM field and allow the highest priority TCAM match to be found using $\log_2 n$ searches, where n is the total number of block values assigned in the classifier. The highest priority match corresponds to the rule with the minimum block number. The rule and its assigned block number are entered in the TCAM. Even though this method does not incur TCAM writes due to rule moves for maintaining consistent block numbers for overlapping rules or to create an empty slot at the right place for inserting a new rule, this method involves a number of TCAM writes as the assigned block numbers of rules change due to inserts or deletes. Moreover, lookup speed is slowed down since multiple TCAM searches are required and these searches cannot be pipelined as they take place on the same TCAM.

Zheng et al. [13] describe a distributed and parallel TCAM architecture, DPPC-RE (Distributed and Parallel Packet Classification with Range Encoding), for packet classification in which the rules are stored in K ($>$1) TCAMS with some rules possibly being stored in more than 1 TCAM. The rule to TCAM distribution strategy ensures that an incoming packet can be matched only by the rules in a single TCAM. As a result, the packets in an incoming stream may be distributed to appropriate TCAMs for classification achieving an expected throughput that is about $K/3$ (when key encoding is used[1]) times the throughput of a simple TCAM classifier (Section II-C). However, since an adversary may inject a packet stream all of whose packets require the same TCAM for lookup, the worst-case throughput is about a third (the same) that of a simple TCAM classifier when key encoding is (not) used. Further, even though it may be possible to initialize the TCAMs to have a roughly equal share of the rules, as updates are performed, the TCAM occupancy can become unbalanced. The strategies suggested in [13] to re-balance the TCAMs delay processing of the packet stream as the affected TCAMs are not available for lookup during re-balancing. In the extreme case, expensive redistribution algorithms have to be run and the entire set of rules redistributed to the K TCAMs. The strategy of [13] also requires substantial additional hardware (a distributer, K processing units, a mapper, etc.) to orchestrate the system. PC-DUOS+, on the other hand, requires much less additional hardware and no re-balancing. Further, the TCAMs are never locked out for lookup and so there is never a delay in processing the packet stream. PC-DUOS+, which uses two TCAMs and no key encoding, obtains an expected throughput of about twice that of a simple single TCAM. This is comparable to the expected throughput of DPPC-RE when $K = 2$ and key encoding is not used. However, the PC-DUOS+ architecture is much simpler, requires no re-balancing and has no lockout of lookups. Since some components (e.g. dispatcher and mapper) of DPPC-RE must operate at K times the speed of a TCAM, the scalability of DPPC-RE is limited to a technology dependent value K_{max}. When this limit is reached, the expected throughput may be further doubled by replacing each TCAM of DPPC-RE with a PC-DUOS+.

The idea of splitting a given set of rules to speed up lookup was used by Kasnavi et al. [16] for packet forwarding. They use different treatment for prefixes that are 16-bits or less and those that are more than 16 bits in length to achieve a low power, high throughput lookup system.

C. Simple TCAM (STCAM)

We use a STCAM architecture for performance comparison. The STCAM is a modification over the naive TCAM in that the rules are grouped by block numbers, which reduces the number of required moves when a free slot is needed. The required number of moves is now bounded by the total number of blocks. The block numbers are assigned to the rules using the algorithm presented in [2], based on a priority graph. A priority graph is a representation of the rules in a classifier. Each rule is represented by a vertex in the priority graph. There is an edge between two vertices iff the corresponding

[1]We assume that a rule fits in a TCAM slot. So, when key encoding is used, two TCAM searches (one for each of the source and destination ports) are needed to encode the key and a third TCAM search is needed for rule matching.

rules overlap. The direction of the edge is from the vertex for the higher priority rule to the vertex for the lower priority rule.

Example: *Suppose a classifier has four two-field rules as shown in Figure 2. Figure 3 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 R3, because the source prefix fields of these rules are non-overlapping. For example, R3 matches source addresses with 0 on the first octet, whereas R1 matches those with 1 on the first octet. Thus, the sets of addresses matched by R1 and R3 are disjoint.*

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

Fig. 2. A classifier with four rules

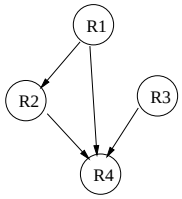


Fig. 3. A priority graph

The method of assigning block numbers to the rules is explained as follows: a subset of the rules is identified such that within the subset, each rule overlaps with every other rule. Such a subset, in this example, are the rules R1, R2 and R4. Each rule in the subset is assigned a different block number based on its priority. Block numbers can be reused for different non-overlapping rule subsets. Thus, rules with the same block number are all non-overlapping or independent. Two rules are independent iff there is no packet that matches both the rules. The rules R2 and R3 are independent rules in this example. Filters are grouped based on their assigned block numbers. The group with the lowest block number is of highest priority and these rules are stored in the lowest memory addresses of the TCAM. Figure 4 shows how the rules are grouped into three blocks in the TCAM, where 'x' represents an octet of 'don't care' bits.

D. Difference between PC-DUOS+ and PC-DUOS

PC-DUOS+ differs from PC-DUOS in the way the selection of rules for the LTCAM is made. PC-DUOS filters the *leaves of leaves* set in a multi-dimensional trie to keep only the highest priority rules among all overlapping rules. The rules in the filtered leaves of leaves set is then entered in the LTCAM. PC-DUOS+, on the other hand, uses a priority graph to select rules for the LTCAM. PC-DUOS+ also uses new

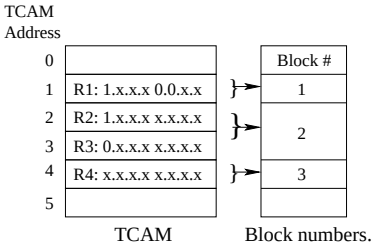


Fig. 4. Rules of Figure 2 in a TCAM; and mapping of TCAM addresses to block numbers

algorithms for ITCAM rule insertion which require fewer moves to rearrange rules for priority based adjustments. The ITCAM insert algorithm is not specific to PC-DUOS+, and can be used with any packet classifier architecture to reduce the number of writes during an update.

III. PC-DUOS+: Methodology

PC-DUOS+ uses the 2-TCAM architecture used in PC-DUOS[15] (Figure 1). During lookup, the LTCAM and ITCAM are searched in parallel using the packet header information. If a match is found in the LTCAM, the ongoing search in the ITCAM is aborted. When the ITCAM search is aborted, lookup time is reduced by about 50%[1], because the LTCAM has no priority encoder. For this lookup strategy to yield correct results, the following requirements must hold:

- R1) No packet is matched by more than one rule in the LTCAM.
- R2) When a packet is matched by a rule in the LTCAM, the matched rule must be the highest priority matching rule.

The algorithms used for storing and updating rules in the TCAMs are discussed in detail below.

A. Storing Rules in TCAMs

Figure 5 shows the overall flow of storing rules in the ITCAM and the LTCAM. The first phase involves creating a priority graph and a multi-dimensional trie for the rules in the classifier. This is further discussed in Section III-A1. The second phase in our methodology consists of identifying a set of highest priority independent rules and storing these in the LTCAM, which is discussed in Section III-A2. In the third phase, the remaining rules are stored in the ITCAM in priority order. This is discussed in Section III-A3.

1) Representing Classifier Rules: The classifier rules are represented in a priority graph as well as in a multi-dimensional trie. A priority graph contains one vertex for each rule in the classifier. There is a directed edge between two vertices iff the two rules overlap and the direction of the edge is from the higher to the lower priority rule. Two rules overlap iff there exists at least one packet that matches both the rules.

Each dimension in a multi-dimensional trie represents one field of the rule. The fields in a filter rule appear in the following order in the trie: <*destination, source, protocol, source port range, destination port range*>. We assume that the destination and source fields of the filters are specified

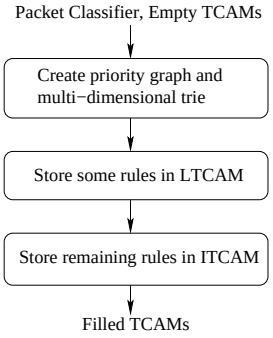


Fig. 5. Flow diagram for storing packet classifiers in TCAMs

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. Ranges may be handled in one of many ways. In this paper, we use the DIRPE scheme of [3] that requires the use of a multi-bit trie. Our methodology may also be applied to other range encoding schemes, such as those in [12] and [14].

2) Storing rules in the LTCAM: Recall that two rules are $independent$ iff no packet is matched by both rules. For the LTCAM we are interested in identifying the largest set of rules that are pairwise independent. Note that every independent rule set satisfies the first requirement (R1) for a lookup to work correctly. 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. It is easy to see that the rules in the vertices of the priority graph with in-degree 0 are independent rules. Further, these rules are also the highest priority rules among all rules that overlap with them. This satisfies the second requirement (R2) for a lookup to work correctly. Hence, we choose to enter these rules into the LTCAM. All remaining rules are entered in the ITCAM.

3) Storing rules in the ITCAM: The rules to be stored in the ITCAM, are assigned block numbers. The priority graph is used to assign block numbers as follows [2]. All vertices, to which there are no incoming edges, are assigned a block number of 1. All children of the vertices with block number 1 are assigned a block number of 2 and so on. A $parent$ of a vertex v in the priority graph, is a vertex from which there is an incoming edge to v. Similarly, a $child$ of v is a vertex to which there is an out-going edge from v. Thus a child of any vertex is assigned a block number that is at least one more than that of this vertex. A $path$ in a graph is a sequence of vertices such that from each vertex there is an edge to the next vertex in the sequence. A non-trivial path is a path with at least two vertices. An $ancestor$ of a vertex v is a node that has a non-trivial path to v. A $descendant$ of v is a vertex to which there is a non-trivial path from v. In other words, a descendant of v has v as one of its ancestors.

In the block assignment scheme, rules that are assigned the same block number are independent and hence grouped together in a single block. These blocks are entered in the TCAM in increasing order of the assigned block numbers. In our implementation, each vertex v in the priority graph has a field $v{\rightarrow}hpri$ which stores a pseudo priority associated with the block number of the vertex. While $v{\rightarrow}hpri$ equals the block number of v in PC-DUOS, in PC-DUOS+, $priorityMap(v{\rightarrow}hpri)$ is the block number for rule v. The example below shows the initial assignment of rules to an ITCAM and the pseudo-priorities assigned to the rules. when PC-DUOS and PC-DUOS+ are used.

Example: *The rules in Figure 2 are inserted in ITCAM in priority order as shown in Figure 6. Its an initial assignment and there are three blocks. Rule R1 is placed in the first block, rules R2 and R3 are placed in the second and rule R4 in the third block. The first block is assigned to TCAM address 1, the second to TCAM addresses 2-3, and the third to 4. The pseudo-priorities assigned to the blocks are 1, 2, and 3 respectively. We will see later how the pseudo priorities change as new rules are added.*

TCAM Address		Block #	Pseudo−priority	
			PC−DUOS	PC−DUOS+
0				
1	R1: 1.x.x.x 0.0.x.x	1	1	1
2	R2: 1.x.x.x x.x.x.x	2	2	2
3	R3: 0.x.x.x x.x.x.x			
4	R4: x.x.x.x x.x.x.x	3	3	3
5				

Fig. 6. Pseudo priority assignments in PC-DUOS and PC-DUOS+

When the priority graph is constructed for the initial classifier, $v{\rightarrow}hpri$ equals the block number of v and $priorityMap$ is an identity mapping. However, as we insert and delete rules, $v{\rightarrow}hpri$ may no longer equal the block number of v (in fact, $v{\rightarrow}hpri$ may not be an integer) and $priorityMap$ is no longer an identity mapping.

To build the priority graph, we first iterate over the classifier rules and for each rule, identify all rules that overlap with it. A trie-based algorithm to determine the rules that overlap a given rule is presented in Figure 7. For simplicity, the algorithm is specified for the case when rules have only two fields - destination and source prefix. Its extension to rules with a larger number of fields is straight forward. Given a rule, the algorithm first extracts the values for the different fields for the rule, and traverses the trie along these prefix paths until all overlapping rules are found. For each overlapping rule found, a directed edge is added to the priority graph. The priority graph is a directed acyclic graph and block numbers are assigned using an iterative process.

Even though in the worst case all the trie nodes have to be explored for finding overlapping rules (this happens, for example, when $ruleInstance$ is the root of the multi-dimensional trie and thus represents a classifier rule with wild-carded fields) this approach works well on average and, in fact, it makes the computation in PC-DUOS+ scalable during the initial setup as well as while processing the updates. In contrast, the simple approach of iterating over all the rules of the classifier to compare overlaps and priorities, quickly becomes a performance bottleneck as the number of rules in the classifier increases.

Algorithm: findOverlappingRules($ruleInstance$**)**
Inputs:
$ruleInstance$: a binary trie node representing a rule and storing its action.
Src: source prefix obtained from $ruleInstance$.
$Dest$: destination prefix obtained from $ruleInstance$.
Output:
list: a list of rules overlapping with the input rule

```
nodeD = root of destination trie;
for (i=0; i<length of destination prefix; ++i)
    if (root of a source trie is stored at nodeD)
        nodeS = root of source trie
        for (j=0; j<length of source prefix and nodeS; ++j)
            if nodeS stores a rule R
                append R to list.
            branchBitS = Src[j]; // jth bit of Src prefix
            nodeS = nodeS→child[branchBitS];
                // nodeS→child[0] and nodeS→child[1] are
                // the two chidren of nodeS in the trie
        endfor
        if (nodeS != NULL) then
            visit all nodes in subtrie rooted at nodeS
            if (any node stores a rule R)
                append R to list.
            endif
        endif
    endif
    branchBitD = Dest[i];
    nodeD = nodeD→child[branchBitD];
endfor
for nodeX in subtrie rooted at nodeD
    if (any node stores a rule R)
        append R to list.
    endif
endfor
```

Fig. 7. Find overlapping rules by trie traversal

Function: Trie.insert
Trie.insert(rule, action);
This function inserts a rule and its action into the control-plane multi-dimensional trie.
Function: Trie.delete
Trie.delete(rule);
This function deletes a rule from the control plane trie.
Function: Trie.change
Trie.change(rule, action);
This function changes the action associated with a prefix.

Fig. 8. Table of control-plane trie functions

B. Update algorithms

When an update request is received, the priority graph and the multi-dimensional trie are updated. Section III-B1 describes how this is done. Next the existing ITCAM rules that overlap with the rule involved in the update are $rearranged$ to ensure that the highest priority rules are still matched after the update is complete. Rules may also be moved from the ITCAM to the LTCAM or vice versa as a result of the updates. This step is discussed in Section III-B2.

1) Update the priority graph and the trie: This is the first step in the update process. The multi-dimensional trie is updated with the help of functions as described in Figure 8.

The priority graph is updated next. If the update is a delete request, then the vertex for the rule to be deleted (together with incident edges) is removed from the priority graph and rules corresponding to vertices whose in-degree becomes 0 are moved from the ITCAM to the LTCAM. Each rule that is to be so moved is first inserted into the LTCAM and then deleted from the ITCAM using insert/delete procedures described in Sections III-B2b and III-B2d. If the update is an insert, then a new vertex is added to the priority graph. All rules overlapping with the new rule are found, and a new edge is added for each overlapping rule. Overlapping rules are identified by traversing the trie using the algorithm of Figure 7. After adding a new vertex v to the priority graph, $v{\rightarrow}hpri$ is calculated. If v has no incoming edges $v{\rightarrow}hpri$ is set to 0 and the new rule in v is placed in the LTCAM. Otherwise, v is placed in the ITCAM.

If v is placed in the ITCAM then $v{\rightarrow}hpri$ is set either by moving v's ancestors upward or its descendants downward or by moving neither descendants nor ancestors. These three possibilities are shown in Figures 9(c), (d) and (e). Figure 9(a) depicts a portion of the original graph. The number next to each vertex shows the $hpri$ value on that vertex. The newly added vertex v is colored black in Figure 9(b). In Figure 9(c), $v{\rightarrow}hpri$ is set based on v's parent $hpri$ so that v will be placed in the ITCAM block below that of its parent. Note that the $hpri$ of v's child must be updated too and the child is moved one block downward, thus avoiding v and its child being placed in the same ITCAM block. Such updates propagate to all descendants. In Figure 9(d), $v{\rightarrow}hpri$ is set based on the $hpri$ of v's child so that v will be placed in the block above that of its child. The $hpri$ of v's parent is updated so that the parent is moved one block upward and these updates propagate to all ancestors. Figure 9(e) shows a case where a new block is inserted between the parent block and the child block, and the hpri associated with the new block is 3.5. Thus $v{\rightarrow}hpri$ is set to 3.5, and neither the descendants nor the ancestors of the new block are moved.

Figure 10 shows the algorithm to set $v{\rightarrow}hpri$. Figure 11 shows how the descendants are moved downwards. In Figure 10, we first calculate the number of moves to set $v{\rightarrow}hpri$ when descendants are moved downwards (childMoves) and when the ancestors are moved upwards (parentMoves). These calculations are based on the flow diagram in Figure 13(b). Suppose $v{\rightarrow}hpri$ is set by moving descendants downwards, and the block number corresponding to the maximum $hpri$ of the parent vertices is B. Then v is assigned to a block $B+1$ and no child vertex of v can be in a block lower than $B+2$. If a child vertex is found to be in a block lower than $B+2$ by mapping the child's $hpri$, then that child must be moved to an appropriate block, which could be either block $B+2$ or some higher block such as $B+3$, $B+4$, etc. Such updating happens recursively for all descendants as shown in Figure 11. The algorithm to set $v{\rightarrow}hpri$ by moving ancestors upwards is similar.

Example: *This example highlights the difference in moving the descendants downwards versus the ancestors upwards upon an insert. We start with the TCAM assignment shown*

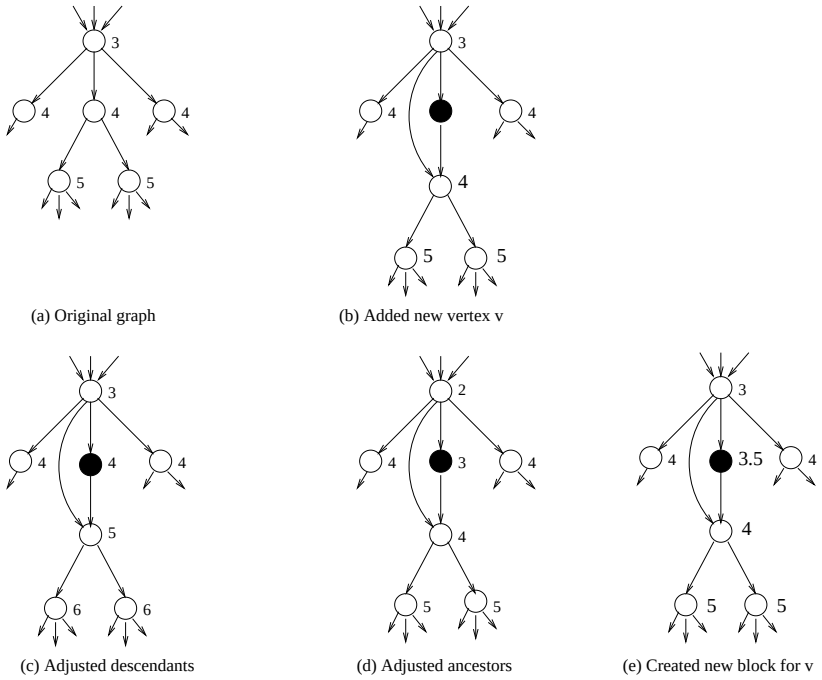


Fig. 9. Setting $hpri$ on a new vertex in a priority graph

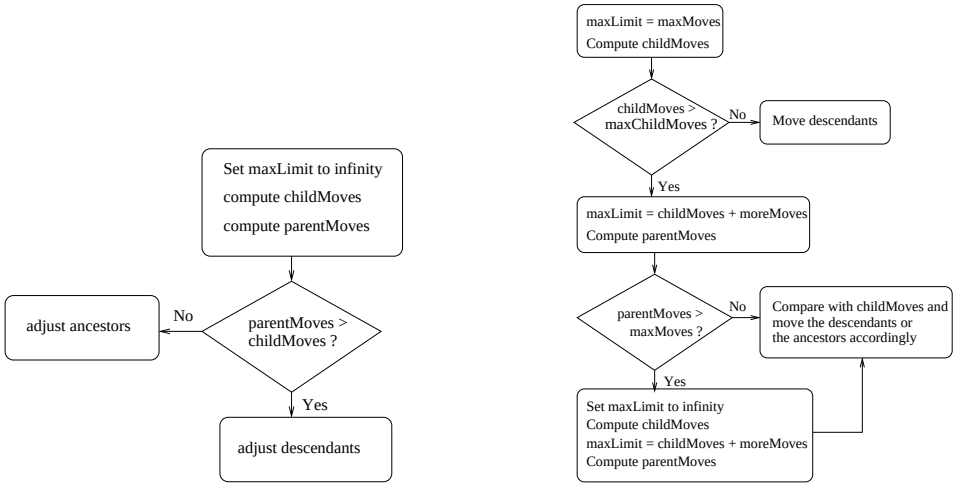
in Figure 6. We insert a new rule R:(1.0.0.0/8, 0.0.0.0/8), with priority lower than R1, but higher than R2. When the descendants are moved downwards, we need three TCAM writes. The first TCAM write copies R4 to position 5, the second write copies R2 to position 4 and the third write inserts the new rule R at position 3. The pseudo priority of the blocks preserve identity mapping. Descendants are moved downwards to create a free slot during an insert by both PC-DUOS and the update scheme in [2]. This strategy is also selectively used in PC-DUOS+.

Now consider the other situation when the ancestors are moved upwards. In this case we need just two TCAM writes. The first TCAM write copies R1 to position 1, and the second TCAM write inserts the rule R to position 2. In this case the relative rule ordering is still the same. However, now the first block cannot continue to have a pseudo-priority of 1, since the second block (consisting of R) is already assigned 1. So, the priority of the first block is changed to a number lower than 1 (but greater than 0, since rules with priority 0 are inserted in the LTCAM) and we set it to 0.5. This is shown in Figure 12(a). Thus identity mapping is not preserved any more.

PC-DUOS+, initially estimates the number of TCAM writes that would be needed when the descendants are moved downwards, versus when the ancestors are moved upwards and the direction that yields a lower number of TCAM writes is selected for moving the rules. This step is computationally intensive, with a worst case complexity of $O(NL)$, where N is the number of vertices in the priority graph and L is number of vertices on the longest path. L is also referred to as the *maximum chain length* of the priority graph. The worst case happens when each vertex is connected to every other vertex. The flowchart in Figure 13(a) shows an unoptimized decision diagram that causes a performance bottleneck. In this case, the actual number of moves is computed for both the cases when the descendants and the ancestors are moved. childMoves stores the number of rule moves computed when the descendants are moved and parentMoves stores the corresponding number when the ancestors are moved. $maxLimit$ is the maximum number of rule moves when descendants or parents are moved. $maxLimit$ is set to infinity and does not play any role in this decision diagram.

To avoid this performance bottleneck, we calculate the number of moves based on the flowchart in the Figure 13(b). A threshold $maxLimit$ is set to avoid calculating the exact number of moves. When $maxLimit$ is set, "Compute childMoves" and "Compute parentMoves" return the number of moves if its below $maxLimit$, otherwise they return $maxLimit$. The following example shows how the flowchart helps in speeding the calculations. Suppose, the exact number of child and parent moves are 870 and 3050 respectively. If it is found that childMoves=870, then before computing parentMoves, we set $maxLimit$ to a number larger than 870 (say 900), and then call 'Compute parentMoves'. The parentMoves returned is 900, instead of 3050, which is sufficient information for the algorithm in Figure 10, as it tells that parentMoves are more than childMoves. Thus, the time to count parentMoves up to 3050 is saved by setting $maxLimit$ appropriately. The flowchart variables maxMoves, maxChildMoves and moreMoves, were set to 500, 50 and 100 in our experiments. This filtered out the less than 10 percent of the child or parent move calculations that took 90 percent of the total processing time.

We use another optimization in the ITCAM rule placement strategy, where a new block is inserted into the TCAM between two existing blocks as shown in Figure 9(e) and on lines 18 and 43 of Figure 10. If the maximum block number of the parents of v is B and the minimum block number of its children is $B + 1$, then instead of moving all children in



(a) Inefficient flow (b) Efficient flow

Fig. 13. Decision diagrams for priority adjustment of descendants vs. ancestors

block $B+1$ to $B+2$ or all parents in block B to $B-1$, a new block is created in the ITCAM between the blocks B and $B+1$ and $v{\rightarrow}hpri$ is set to the average of the hpri-s of the two blocks (i.e. (hpri_of(B) + hpri_of($B+1$)) /2). The new rule for v is then added to the new block. If the new rule is to be added on top of the topmost ITCAM block as on line 38 of Figure 10, then $v{\rightarrow}$hpri is set to (hpri_of(B) /2). Recall that the vertices with in-degree 0 are assigned an block number 0. Hence the assignment of hpri, ensures that the block number is still greater than 0. Addition of a new block must be done judiciously, since it requires an extra move while bringing in a free slot to a particular block B when the newly inserted block is between the free space pool and B. This is shown in Figure 14. While it takes just a single rule move in block C to get a free slot in block B in the setup of Figure 14(a), it takes 2 rule moves when a new block A is added as shown in Figure 14(b). This optimization is enabled by the user settable 'isInsertFlag' on lines 17 and 42 of Figure 10. Figure 9(e) shows that $v{\rightarrow}hpri$ for the new vertex v is set to 3.5. A new block is added between the parent and the child blocks in this case.

For consistent updates [10], [11], if the vertices are to be moved downwards, then the moves may be executed in increasing order of priority starting from the lowest priority rule and after all the descendants are moved, the new rule is added. If the vertices are moved upwards, then the moves may be executed in decreasing order of priority, starting from the highest priority rule. After all the ancestors are moved, the new rule is added. Lines 55-59 of Figure 10 ensure that nodes are moved to their assigned slots in the reverse order of visiting them. Thus, the node last visited for updating $hpri$ is the first to be moved to its assigned slot. This preserves update consistency for both the cases when the descendants are moved downwards and the parents upwards. The new rule is added at the end (Line 60).

2) Updating the TCAMs: TCAM updates are generated after updating the priority graph. Rules may be moved from the ITCAM to the LTCAM or vice versa or they may be moved within the ITCAM for rearrangement of overlapping rules. To insert or move a rule in a TCAM we need a free slot at an appropriate location. This slot can be obtained efficiently using memory management algorithms. In particular, the memory management schemes from DUOS may be used here. For the ITCAM of PC-DUOS+ as well as PC-DUOS, 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. The memory management scheme for LTCAM is relatively simple as all the rules in the LTCAM are independent so a new rule can be inserted anywhere in the TCAM. However, we still need to locate a free slot. The LTCAM memory management algorithm of DUOS 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-DUOS+, as well as PC-DUOS, uses the memory management algorithm for DUOS for its LTCAM [9].

Since the blocks grow both ways, up as well as down, PC-DUOS+ has a modified initial rule placement policy as shown in Figure 15 where 25% of the free slots (represented by white blocks) are placed on the top of the TCAM (that is, covering the lowest addresses) and another 25% are kept at the bottom of the TCAM (covering the highest addresses). The remaining 50% of the free slots are distributed to the region between the blocks in proportion to the number of rules in a block.

a) ITCAM.insert: :
To insert a new rule in the ITCAM, a free slot is first made available at the desired block. A free slot may be present in the same block in which case no moves are needed to get it from the free slot list of the block. If there is no free slot in the block, then a free slot may be obtained from the inter-block region on the top or the bottom of the block. No moves are needed in this case too. If there is no free slot in the inter-block region adjacent to the block, then a free slot is moved from the nearest neighboring block where its available.

Algorithm: insertRule(v)
Inputs:
Rule stored in vertex v in the priority graph.
User settable 'isInsertFlag' which is used for optimization and explained in Section III-B1

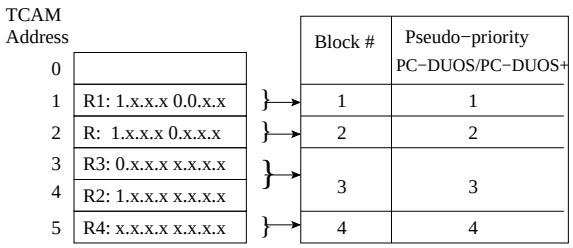
```
1  maxP = max(parent→hpri) from ITCAM parents of v;
2  minC = min(child→hpri) from children of v;
3  // Default values are maxP: −1 and minC: infinity
4  childMoves = parentMoves = 0;
5  if (maxP ≥ minC) then
6      compute childMoves to push descendants down and
7      parentMoves to push ancestors up according to Figure 13(b).
8  endif
9  // Get block BC corresponding to minC. If v has no outgoing
10 // edges, then BC − 1 is the last block in the ITCAM.
11 BC = priorityMap(minC);
12 BP = priorityMap(maxP);
13 if (v has a parent vertex in the ITCAM and
14       parentMoves < childMoves) then
15    // Move ancestors upwards
16    targetBlock = BC − 1;
17    if (BC − 1== BP and isInsertFlag) then
18       targetBlock = create a new block between BP and BC.
19    endif
20    // Function reversePriorityMap returns pseudo-priority
21    // corresponding to targetBlock.
22    v→hpri = reversePriorityMap(targetBlock);
23    assign slot in targetBlock for v;
24    if (v→hpri ≤ maxP) begin
25       sort the parent vertices in a decreasing order of hpri;
26       for each parent of v
27          if (v→hpri ≤ parent→hpri)
28             if (parent is in ITCAM) moveParentUp(parent);
29    endif
30 else // Move descendants downwards
31    // Initially, the highest priority rules in ITCAM have hpri
32    // set to 2. So, targetBlock is initialized to that block.
33    targetBlock = priorityMap(2);
34    if (v has no parent in the ITCAM) then
35       if (there exists a block BC − 1) then
36          targetBlock = BC − 1;
37       else if (isInsertFlag) then
38          targetBlock = create a new block on top of BC.
39       endif
40    else
41       targetBlock = BP + 1;
42       if (BP + 1 == BC and isInsertFlag) then
43          targetBlock = create a new block between BP and BC.
44       endif
45    endif
46    v→hpri = reversePriorityMap(targetBlock);
47    assign slot in targetBlock for v;
48    if (v→hpri ≥ minC) begin
49       sort the descendant vertices in an increasing order of hpri
50       for each child of v
51          if (v→hpri ≥ child→hpri) moveChildDown(child);
52    endif
53 endif
54 // Process nodeList from moveParentUp/moveChildDown
55 for each w in nodeList starting from the last one
56    slotW = current TCAM slot occupied by the rule of w;
57    write the rule of w in the assigned slot;
58    free slotW;
59 endfor
60 write the rule of v in the assigned slot.
```

Fig. 10. Insert a rule in the ITCAM

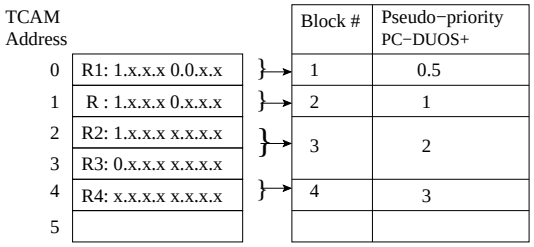
Algorithm: moveChildDown(child)
Input: Rule stored in vertex 'child' in the priority graph.

```
mP = find max(parent→hpri) from all parents of child
mC = find min(child→hpri) from all children of child
if (mP < child→hpri and child→hpri < mC) return;
block = priorityMap(maxP) + 1;
child→hpri = reversePriorityMap(block);
assign a slot in block for child; append child to nodeList;
if (child→hpri ≥ mC) begin
    sort the descendant vertices in an increasing order of hpri
    for each childi of child
       if (child→hpri ≥ childi→hpri) moveChildDown(childi);
endif
```

Fig. 11. Moving descendants downward in the ITCAM



(a) Descendants move down (3 TCAM writes)

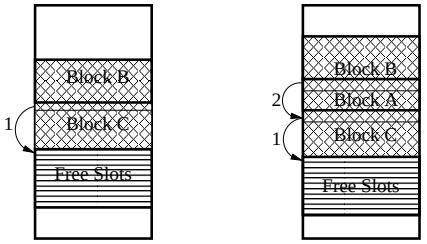


(b) Ancestors move up (2 TCAM writes)

Fig. 12. Comparison of the number of TCAM writes to insert a new rule R

To insert a new block between two blocks in the ITCAM, it is first checked if there is a free slot between the top and bottom blocks. If there are free slots in the region between the top and the bottom blocks, then the rule in the new block is inserted there in such a way that there are some free slots above and below the new block. Otherwise, free slots for the new block are moved in from the nearest neighboring block that has free slots.

b) ITCAM.delete: :
After deleting the vertex corresponding to the rule in the



(a) 1 move (b) 2 moves

Fig. 14. Number of moves to get a free slot to block B, moves are shown by arrows.

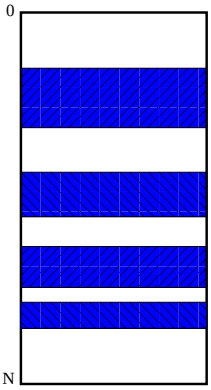


Fig. 15. Initial ITCAM layout

priority graph, the valid bit on the corresponding TCAM slot is set to 0. DLFS_PLO frees up the block if the rule deleted is the last rule in the block. Otherwise, the freed slot is prepended to the head of the list of free slots in the block.

c) ITCAM.change: :
Suppose the specified change is with respect to the fields of a rule, then such a change is implemented as an insert followed by a delete. The insert adds the changed rule to the same block as the old rule, while the delete removes the old rule from this block. If the change is in the priority of the rule, then, we revisit all the incoming and outgoing edges of the corresponding vertex v in the priority graph and reverse the edges appropriately to maintain the edge direction from the higher to the lower priority rule. Then the block number is freshly calculated for v, and the rule is moved to a block at a higher address (if the priority was lowered) or to a block at a lower address (if the priority of the rule was increased) in the ITCAM. If the vertex v does not have any incoming edge following the update, it is moved to the LTCAM.

d) LTCAM.insert, LTCAM.delete and LTCAM.change: :
To insert a new rule in the LTCAM, a free slot is obtained from the head of the LTCAM free slot list. If a rule is deleted from the LTCAM, then the valid bit of the slot is set to 0 and the freed up slot is prepended to the head of the free slot list.

For incorporating a changed rule, if the change is with respect to the fields of a rule, then the changed rule is simply inserted in the LTCAM and the old rule deleted. If the change is in the priority of a rule in such a way that the corresponding vertex now has an incoming edge, then the rule is moved to the ITCAM. Otherwise, if the rule continues to be the highest priority rule among all overlapping rules even after the change, then nothing needs to be done.

IV. Experimental Results

The experimental setup is described in Section IV-A. The results obtained for lookup and update performance are described in Sections IV-B and IV-C.

A. Setup

We programmed the lookup and update algorithms for STCAM, PC-DUOS and PC-DUOS+ in 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 [7], 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.

To test the performance of TCAM lookup, we generated packet traces using ClassBench with 100,000 packet headers simulated lookups in PC-DUOS and PC-DUOS+. To test the update algorithms of PC-DUOS+, we generated an update sequence, by randomly marking some of the rules in a dataset for insertion and some others for deletion. The rules marked for insertion were removed from the dataset to arrive at the initial configuration for the classifier. A random permutation of the removed rules (i.e., those marked for insertion) together with those marked for deletion define the update sequence. Since we use the same update sequence on PC-DUOS+, PC-DUOS and the STW architectures, the comparison of update performance is fair. We generated two sets of tests to experiment with our update algorithms.

For the first set of tests, we generated a classifier of about 10000 rules, from each seed file. Then for a classifier we generated 10 insert sequences each comprising of a different set of 5000 rules to be inserted. Then for each insert sequence we generated 10 different permutations. Thus we generated 100 tests for a seed file, where each test has an insert sequence of 5000 rules.

Figure 16 describes the second set of tests, in which we generate a single test from each file, and these tests contain a longer sequence of updates which include deletes. In Figure 16, the first and second columns give the indexes and names of the classifiers, the third column shows the seed files in ClassBench from which these tests were derived, the fourth column shows the number of rules in the initial configuration of a classifier, and columns five to seven give the number of insert and delete operations in the update sequence. We used 12 seed files based on access control lists (acl), firewalls (fw) and IP chains (ipc) to generate the 13 classifiers. Out of these 13 tests, the first seven were used in [15].

We use DIRPE [3] to store the port ranges in the TCAM. DIRPE was implemented by using multi-bit tries for source and destination port ranges. We assume that 36 bits are available for encoding each port range in a rule. With this assumption, we use strides 223333 for our experiments, which give us minimum expansion of the rules. The stride value 223333 indicates that for a given port number (16 bits), the root of the port range trie will use the first two bits to branch to one of its four possible child nodes at level 1. Each node at level 1 uses the next two bits to branch to one among its four possible child nodes at level 2. A node at the level 2, on

Index	Dataset	seed_file	#Rules	#Inserts	#Deletes
1	acl1	acl1_seed	30075	69300	29700
2	fw1	fw1_seed	7989	28800	7200
3	ipc1	ipc1_seed	15338	34300	14700
4	acl2	acl2_seed	53970	45000	45000
5	fw5	fw5_seed	5571	45900	5100
6	acl4	acl4_seed	34254	5000	5000
7	ipc2	ipc1_seed	5165	94050	4950
8	acl3	acl3_seed	19745	2976	3124
9	acl5	acl5_seed	19492	12500	12500
10	fw2	fw2_seed	16668	15000	15000
11	fw3	fw3_seed	16841	33400	16600
12	fw4	fw4_seed	12882	10000	10000
13	ipc3	ipc2_seed	20000	15000	15000

Fig. 16. Synthetic classifiers and update traces used in the experiments

the other hand, uses the next 3 bits to branch to one among its eight possible child nodes at the level 3, and so on. Thus, all the 16 bits (2 + 2 + 3 + 3 + 3 + 3 = 16) are used to traverse the trie and arrive at the last node (at the 6th level) representing the port number.

We compare our results with those from a single TCAM setup (STCAM) as is commonly used today for packet classification. In this setup, all rules are entered into the TCAM in priority order. The ordering is needed only for rules that overlap. If two rules do not overlap, their relative ordering does not matter. We use a priority graph for the whole set of rules to track the block numbers of the rules as well as to compute adjustments to block numbers as new rules are inserted. We do not compare PC-DUOS+' update performance with that of the work in [2], since PC-DUOS+' lookup performance is far superior to the worst case of [2], which is at least 4 times slower in the worst case, on our datasets (obtained as logarithm of the number of blocks).

We analyze the results based on two perspectives – improvement in lookup performance and improvement in update performance.

B. Lookup Performance

We computed the overall improvement in lookup time using the lookup traces generated using ClassBench.

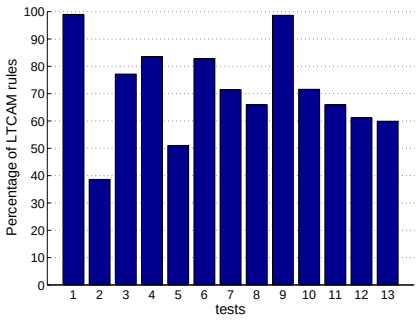


Fig. 17. Percentage of rules in LTCAM of PC-DUOS+

Figure 17 shows the percentage of rules entered in the LTCAM of PC-DUOS+. For example, test 13 has about 60% of the rules in the LTCAM and the remaining 40% in the ITCAM.

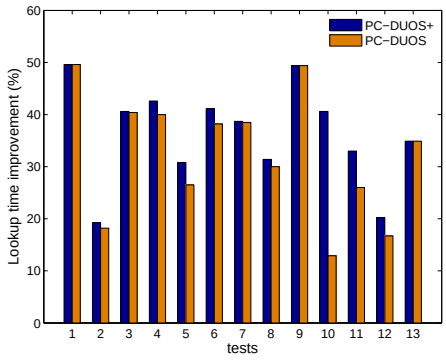


Fig. 18. Comparison of improvement in lookup time between PC-DUOS and PC-DUOS+, with respect to STCAM

Figure 18 shows the improvement in average lookup time of PC-DUOS and PC-DUOS+ with respect to STCAM. Recall that during a lookup, if a match is found in LTCAM then the lookup finishes faster. Having a large number of rules in the LTCAM makes the probability of finding a match in the LTCAM, higher. Thus the tests 1 (acl1) and 9 (acl5) with 99% of their rules in the LTCAM, had the LTCAM serve almost all the lookups. The improvement in average lookup time on these tests was, therefore, almost 50%, which is the maximum improvement achievable by yanking a priority encoder.

PC-DUOS+ had better improvement in lookup time particularly for the tests based on forwarding tables, namely, tests 2, 5, 10, 11, 12, and comparable performance on the other tests based on ACL lists and IP chains. The observed improvement is because the LTCAM of PC-DUOS+ on those tests contained more rules than the LTCAM of PC-DUOS, which is a result of the different algorithm employed in PC-DUOS+ for selecting rules for the TCAMs.

Figure 19 presents the details on the number of rules in the ITCAM and LTCAM and the percentage improvement in lookup performance of PC-DUOS+. The first three columns give the dataset index, name and the number of rules respectively. The fourth and fifth columns give the number of rules in the ITCAM and LTCAM, respectively. The sixth and seventh columns give, respectively, the number of lookups and the percentage improvement in average lookup time.

Index	Dataset	#Rules	#ITCAM	#LTCAM	#Lookups	%Improve
1	acl1	30075	305	29731	120301	49.6
2	fw1	7989	4885	3068	103857	19.2
3	ipc1	15338	3504	11834	107618	40.6
4	acl2	53970	8875	45095	107940	42.6
5	fw5	5571	2689	2796	105430	30.8
6	acl4	34254	5882	28372	103104	41.1
7	ipc2	5165	1476	3689	98136	38.7
8	acl3	19745	6737	13007	102851	31.4
9	acl5	19492	260	19209	97460	49.4
10	fw2	16668	4739	11929	100008	40.6
11	fw3	16841	5688	10986	103794	33
12	fw4	12882	5004	7878	103266	20.2
13	ipc3	20000	8027	11973	100163	34.9

Fig. 19. Number of rules in ITCAM and LTCAM of PC-DUOS+ and improvement in lookup time relative to STCAM

Seed	STCAM			PC-DUOS			PC-DUOS+			PC-DUOS+*	
	Avg. writes	Std. Dev.	#Min	Avg. writes	Std. Dev.	#Min	Avg. writes	Std. Dev.	#Min	Avg. writes	Std. Dev.
acl1	2.19	4.04	1	1.4	2.58	69	1.245	0.16	30	1.283	0.123
acl2	99.7	372	0	4.59	2.29	0	1.59	0.43	100	1.78	0.409
acl3	51.04	161.34	0	1.8	0.69	0	1.62	0.298	100	1.84	0.286
acl4	28.1	252	1	1.87	0.913	0	1.6	0.58	99	1.73	0.43
acl5	3.42	2.53	0	1.13	0.1	100	1.18	0.07	0	1.21	0.11
fw1	83.7	88.1	2	14.99	15.06	0	2.57	2.203	98	3.69	3.376
fw2	3.18	3.84	0	1.92	2.07	0	1.4	0.071	100	1.496	1.063
fw3	54.4	498	14	8.98	8.95	0	2.36	1.34	86	2.36	4.28
fw4	29.7	45.5	0	14.3	23.9	0	7.66	17.36	100	9.85	22.7
fw5	16.9	18.1	0	7.62	10.63	0	2	2.95	100	3.95	9.66
ipc1	9.72	8.99	0	2.23	1.35	0	1.34	0.35	100	1.55	0.39
ipc2	1.022	10.22	0	1	0	100	1	0	100	1	0

Fig. 20. Data obtained on TCAM writes

C. Update Performance

We first present the results from the setup in which we ran 100 tests for each seed file, where each test executed 5000 insert operations. Figure 20 shows the data we collected from these runs. The data in each column is described as follows. The column 'Avg. writes' gives the average TCAM writes per insert operation which is obtained by first getting the average for each of the 100 tests and then dividing the average by 5000, which is the number of inserts for each test. 'Std. dev.' gives the standard deviation, which is obtained by calculating the average TCAM writes per insert for each of the 10 classifiers for a seed file and then computing the difference with the average (for the 100 tests) from the previous column. from the difference of average TCAM writes (for each of the 100 tests) from the average of the average in the previous column. '#Min' gives the number of tests (out of the total of 100) for which the given architecture resulted in a minimum value of the average TCAM writes.

The PC-DUOS+* is a version of PC-DUOS+, that implements only the standard "descendants move downwards" algorithm for ITCAM rule insertion. PC-DUOS+* will be used to compare improvements only due to the new insertion algorithm of PC-DUOS+, thereby nullifying the effects of the other enhancements in PC-DUOS+ such as selection of rules for the TCAMs, with respect to PC-DUOS.

We briefly describe the variable setup used for our experiments. The "isInsertFlag" of Figure 10 was set to true for all ACL and IP chain based classifiers in our experiments. The reason is these classifiers had a small number of rules in the ITCAM and the degree of overlap among the rules was low, in contrast to the third type of classifiers that are based on forwarding rules. If the flag is set to true for the latter, then a lot of small blocks are added in between, which increases the cost to move a free slot.

The graph in Figure 21 shows a comparison of the average number of writes for these tests. The observations on these tests are as follows:

1) The average number of TCAM writes (from column 'Avg. writes') is the minimum for PC-DUOS+ for all the tests except the one based on seed file acl5 which represents ACL based tests. For the remaining tests, PC-DUOS+ performed better than PC-DUOS by an amount between 10-82%. For the tests based on acl5, PC-DUOS+ was very close to PC-DUOS.

2) The difference in the number of TCAM writes between the PC-DUOS+ versions (PCDUOS+* in which only descendants

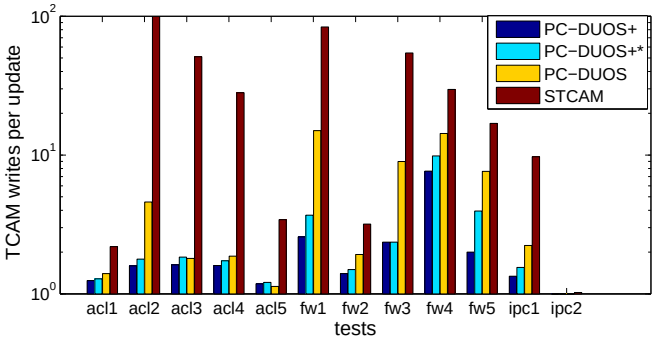


Fig. 21. Number of TCAM writes with respect to PC-DUOS+

are moved versus the original version PC-DUOS+), highlights the performance benefits of our ITCAM insertion agorithm. The insertion algorithm improved the number of TCAM writes by up to 49% for test fw5.

3) PC-DUOS+ obtained the highest number of minimum average writes out of 100 tests (from column #Min) for all the tests except those based on seed files acl1 and acl5. The number of PC-DUOS+ writes for the tests for which its not the lowest, is very close to the lowest score for those tests. For the tests on ipc2, PC-DUOS+ tied with PC-DUOS on the minimum average writes, both requiring exactly the same number of writes.

4) Standard deviation is small for PC-DUOS+, indicating better fidelity to the average.

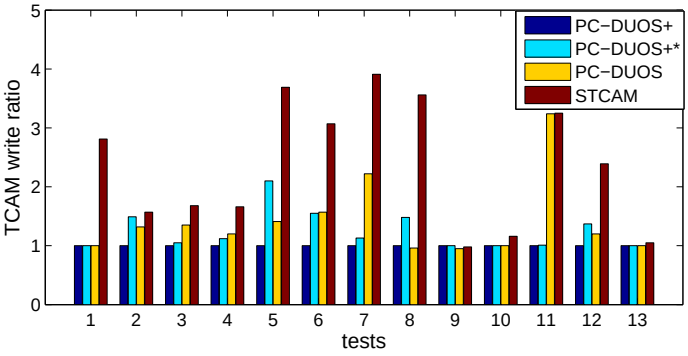


Fig. 22. Number of TCAM writes with respect to PC-DUOS+

Now, we present the results from the second set on tests

Index	DataSet	PC-DUOS+		STCAM	
		#Average TCAM writes	#Worst case TCAM writes	#Average TCAM writes	#Worst case TCAM writes
1	acl1	1.18	31	3.31	53
2	fw1	2.22	4161	3.5	12014
3	ipc1	1.33	1607	2.25	3945
4	acl2	1.37	2487	2.28	6194
5	fw5	1.58	2540	5.84	26020
6	acl4	1.39	364	4.3	9821
7	ipc2	1.34	1793	5.27	17285
8	acl3	2	789	7.13	10946
9	acl5	1.14	9	1.124	11
10	fw2	1.71	9	1.98	9
11	fw3	1.44	259	1.87	14693
12	fw4	2.91	1483	6.97	6996
13	ipc3	1	1	1.05	6

Fig. 23. Average and worst case TCAM writes for PC-DUOS+

for update performance. Figure 22 shows the ratios of TCAM writes needed to process the test update sequence by PC-DUOS+, PC-DUOS [15] and STCAM, where a ratio is computed by dividing the actual number of writes for each of the architectures, by the actual number of writes for PC-DUOS+. A noticeable improvement in the number of writes is observed compared to STCAM for almost all the tests except for tests 9 (acl5), 10 (fw2) and 13 (ipc3). Test 7 (ipc2) requires up to 3.84 times more writes using an STCAM compared to PC-DUOS+, while test 11 (fw3) requires up to 3.23 times the number of writes using PC-DUOS compared to PC-DUOS+.

From Figure 22 we observe that tests 9 (acl5), 10 (fw2) and 13 (ipc3) need almost similar number of writes in all the three setups, namely, PC-DUOS+, PC-DUOS and STCAM. The reason is the degree is overlap among the classifier rules is very small in these tests. As a result, when a rule is inserted, it hardly requires any rule moves to adjust the priorities of the descendants or ancestors, which reduces the required number of TCAM writes.

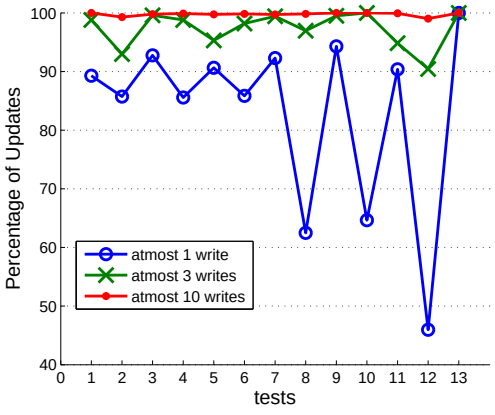


Fig. 24. Percentage of updates that require 1 write, $\leq$ 3 and $\leq$ 10 writes

Figure 23 gives the average and the worst case TCAM writes for PC-DUOS+ and STCAM. The average writes for PC-DUOS+ are lower than the corresponding numbers for STCAM. The worst case writes for PC-DUOS+ is lower than those for STCAM for all tests except test 9 (acl5). The number of TCAM writes in the worst case for PC-DUOS+ is quite high, even though from Figure 24 we observe that more than

Index	Data-Sets	PC-DUOS+			PC-DUOS		STCAM	
		Writes	Time(s)	Writes*	Writes	Time(s)	Writes	Time(s)
1	acl1	116408	9.15	116420	116393	8	327675	2469
2	fw1	80210	377	119569	105866	928	126225	935
3	ipc1	65592	124	68812	88736	265	110346	780
4	acl2	124043	185	139147	148727	921	205568	1725
5	fw5	80658	926	169560	113358	2012	297624	2702
6	acl4	13996	7.5	21674	22030	31	43017	118
7	ipc2	133396	540	150839	296663	1449	521653	4808
8	acl3	12233	14.69	18056	11798	47	43494	97
9	acl5	28742	0.9	28752	27366	1.2	28090	167
10	fw2	51304	158	51306	51402	792	59406	488
11	fw3	28710	175	28862	92955	1057	93426	1098
12	fw4	58243	74	79789	69958	362	139434	378
13	ipc3	30000	211	30000	30000	287	31647	416

Fig. 25. Total TCAM writes in PC-DUOS+, PC-DUOS and STCAM

99% of the rules require at most 10 writes.

Figure 25 shows the actual number of TCAM writes for inserting or deleting rules in the different datasets and the time taken to perform these updates in PC-DUOS+, PC-DUOS and STCAM. The fifth column 'Writes*' gives the number of writes using PC-DUOS+*. Using the ITCAM insertion algorithm of PC-DUOS+, the number of writes reduced by up to 52% for test fw2, compared to PC-DUOS+*. Compared to PC-DUOS, the number of TCAM writes using PC-DUOS+, reduced by up to 69% and compared to STCAM, the number of TCAM writes reduced by up to 74%.

V. Conclusion

PC-DUOS+ is proposed for packet classifier lookup and update. Two TCAMs named LTCAM and ITCAM are used. PC-DUOS+ stores the highest priority independent rules in the LTCAM. The remaining rules are stored in the ITCAM. During lookup for highest priority rule matching, both the ITCAM and the LTCAM are searched in parallel. Since the LTCAM stores independent rules, at most one rule may match during lookup in the LTCAM and a priority encoder is not needed. If a match is found in the LTCAM during lookup, it is guaranteed to be the highest priority match and the corresponding action can be returned immediately yielding up to 50% improvement in TCAM search time relative to STCAM.

Compared to PC-DUOS, PC-DUOS+ has a new rule selection algorithm for allotment of rules to the ITCAM and the LTCAM. PC-DUOS+ also has a new ITCAM insertion algorithm which improves the update performance in terms of the number of TCAM writes, as well as the update processing time in the control plane. The ITCAM rule insertion algorithm moves the overlapping rules by either shifting the descendants downwards, or the ancestors upwards. This is in contrast to the conventional method of always shifting the descendants downwards, and improves the number of ITCAM writes by up to 52% compared to the conventional method. This insert algorithm is not limited to PC-DUOS+, and may be used with any architecture to improve the TCAM update performance. The average improvement in lookup time is found to be between 19% to 49% for the tests in our dataset. The distribution of rules to the two TCAMs makes updates faster by reducing the average number of TCAM writes by up to 3.84 times (for ipc2) and reducing the control-plane processing time by up to

270 times (for acl1). The maximum reduction in control-plane processing time is observed for the ACL tests.

The throughput of PC-DUOS+ is comparable to that of DPPC-RE [13] when $K = 2$ TCAMs are used and key encoding is not used. However, the PC-DUOS+ architecture is much simpler, requires no re-balancing of rules and has no lockout of lookups. Further, when the scalability limit of DPPC-RE is reached, performance may be doubled by replacing each TCAM of DPPC-RE with a PC-DUOS+.

REFERENCES

[1] M. Akhbarizadeh and M. Nourani, Efficient Prefix Cache For Network Processors, *IEEE Symp. on High Performance Interconnects*, 41-46, 2004.

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

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

[4] D. E. Taylor, Survey and taxonomy of packet classification techniques *ACM Computing Surveys (CSUR)* Volume 37 Issue 3, September 2005, 238-275 .

[5] S. Sahni, K. Kim, and H. Lu, Data structures for one-dimensional packet classification using most-specific-rule matching, *International Journal on Foundations of Computer Science*, 14, 3, 2003, 337-358.

[6] 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, December 2004, 1602-1614.

[7] D. E. Taylor and J. S. Turner, ClassBench: A Packet Classification Benchmark, *IEEE/ACM Transactions on Networking*, Volume 15, No. 3, June 2007, 499-511

[8] 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

[9] T. Mishra and S.Sahni, Green TCAM-based Internet routers, *Handbook of Energy-Aware and Green Computing*, Chapman-Hall/CRC Press, 2012

[10] T. Mishra and S. Sahni, CONSIST - Consistent Internet Route Updates *IEEE Symposium on Computers and Communications*, 2010.

[11] 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, December 2004, 1602-1614.

[12] H. Che, Z. Wang, K. Zheng and B. Liu, DRES: Dynamic Range Encoding Scheme for TCAM Coprocessors, *IEEE Transactions on Computers*, 57, 7, July 2008, 902-915.

[13] K. Zheng, H. Che, Z. Wang, B. Liu, and X. Zhang, DPPC-RE: TCAM-Based Distributed Parallel Packet Classification with Range Encoding, *IEEE Transactions on Computers*, 55, 8, August 2006, 947-961.

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

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

[16] S. Kasnavi, P. Berube, V. C. Gaudet and J. N. Amaral, A Multizone Pipelined Cache for IP Routing, *Computer Networks* Volume 52, No. 2, February 2008, 303-326

[17] 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.

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

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

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

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

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

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

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

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

[26] Y-K. Chang, A Two Level TCAM Architecture for Ranges, *IEEE Transactions on Computers* 55, 12, Dec 2006, 1614-1629.

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, working under the supervision of Prof. Sartaj Sahni.

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.

Dr. Guna Seetharaman is a Principal Engineer for Computer Vision, and Information Fusion and Exploitation, at the Information Directorate, Air Force Research Laboratory, Rome, NY. He is currently focused on high performance computing, computer vision, machine learning, content-based image retrieval, persistent surveillance and computational science and engineering. He served as an associate professor of computer science and engineering, at the Air Force Institute of Technology (2003-2008) and University of Louisiana at Lafayette (1988-2003). He was also a CNRS Invited professor at University of Paris XI on multiple tenures between1998-2005; and, held a visiting distinguished professorship at Indian Institute of Technology, Mumbai, in 2006.

Dr Seetharaman initiated and established computer vision and robotics laboratories at The University of Louisiana at Lafayette. He co-founded Team Cajunbot . a participant in DARPA Grand Challenge. He led the LiDAR data processing and obstacle detection efforts in Team Cajun Bot, demonstrated in 2005 and 2007 DARPA Grand Challenges. He was a member of the AFIT based core team for demonstrating and transitioning a wide area persistent imaging and surveillance system known as Angel Fire. He currently leads efforts on power, latency and communications optimized video processing across video camera networks.

He has published more than 150 peer- reviewed articles in: Computer Vision, low-altitude aerial imagery, Parallel Computing, VLSI-signal processing, 3D Displays, Nano-Technology, micro-optics, and 3D Video analysis. He co-organized the DOE/ONR/NSF Sponsored Second International Workshop on Foundations of Decision and Information Fusion, in 1996 (Washington DC), and the IEEE Sixth International Workshop on Computer Architecture for Machine Perception, New Orleans, 2003. He guest edited IEEE COMPUTER special issue devoted to Unmanned Intelligent Autonomous Vehicles, Dec 2006. He also guest-edited a special issue of the EURASIP Journal on Embedded Computing in the topics of Intelligent Vehicles. He is an active member of the IEEE, and ACM. He is also a member of Tau Beta Pi, Eta Kappa Nu and Upsilon Pi Epsilon. He is a Paul Harris Fellow of the Rotary International.