# PC-DUOS: Fast TCAM Lookup and Update for Packet Classifiers *

Tania 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 48% and our experiments demonstrate an improvement in update performance by up to 2.8 times in terms of the number of TCAM writes.*

**Keywords**
Packet classifiers, TCAM, updates.

## 1 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, 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.

*(a) 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-2011-2171

[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 begin in Section 2 by reviewing the background and related work. In Section 3 we describe our scheme of storing packet classifiers in TCAMs. An experimental evaluation of our scheme is done in Section 4 and we conclude in Section 5.

## 2 Background and Related Work

PC-DUOS (Packet Classifier - DUOS) is an extension of DUOS [9], which has been proposed for packet forwarding. DUOS, as shown in Figure 1, has 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 final stage logic 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

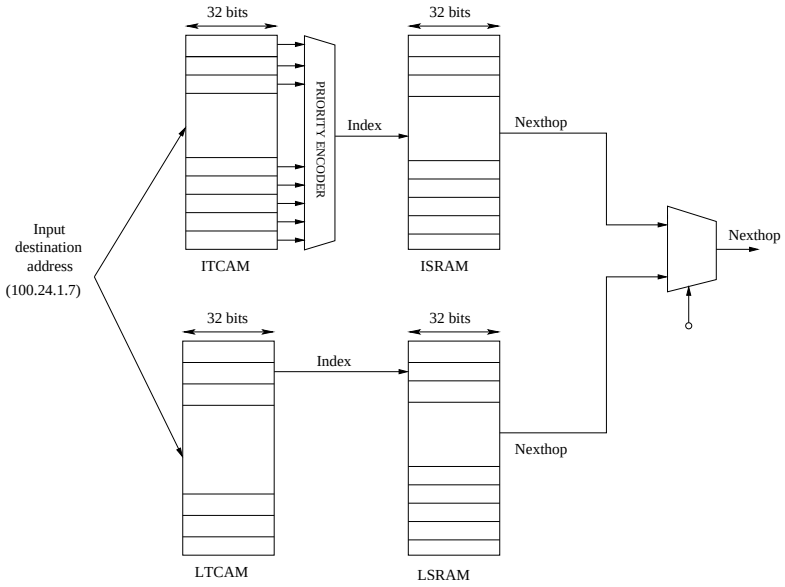


**Figure 1. Dual TCAM with simple SRAM**

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), which initially distributes the free space 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.

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.

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.

We use a simple TCAM (STCAM) architecture for comparing our PC-DUOS performance. 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. In this method a subset of the rules is identified such that within the subset, each rule overlaps with every other rule. 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. 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.

The authors in [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. Our PC-DUOS architecture performs lookup using a single TCAM search.

## 3 PC-DUOS: Methodology

PC-DUOS uses the two TCAM architecture of Figure 1. During lookup the LTCAM and ITCAM are searched in parallel using the packet header information. In case a match is found in the LTCAM, the ongoing search in the ITCAM is aborted. For this lookup strategy to yield correct results, the following requirements must hold:

1. No packet is matched by more than one rule in the LTCAM.

2. 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 ensure that these requirements are met.

### 3.1 Representing Classifier Rules

The classifier rules are represented using a multi-dimensional trie, where each dimension 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 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 consider first the simple encoding of a range by a set of prefixes[3, 2]. This encoding is well suited for a binary trie representation. Next, we consider the DIRPE scheme of [3] that requires the use of a multibit trie. Our methodology may also be applied to other range encoding schemes [11, 12].

### 3.2 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 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 nodes obtained by traversing the multidimensional trie from the root to the leaves of the outer level trie and then from these leaves into their attached next level tries and so on until we reach leaves of the innermost level tries are independent. We call this set of independent rules the *leaves of leaves* set.

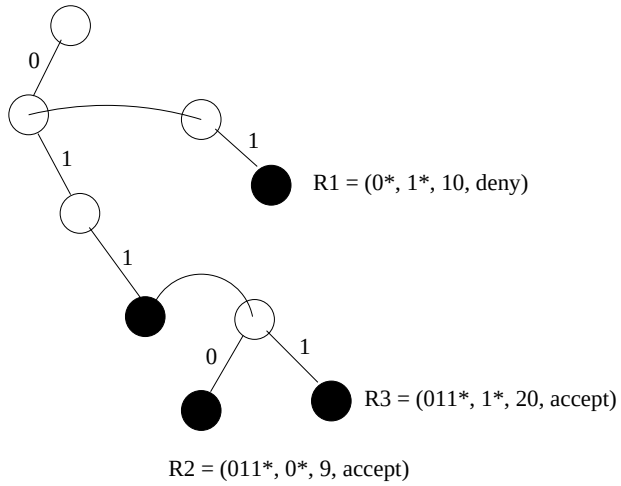


**Figure 2. Example of a two-dimensional trie storing three rules: R1, R2 and R3**

Figure 2 shows an example classifier, with three rules. The outer level trie (level 1) is on the destination prefix and the inner level (level 2) is on the source prefix. Rules R2 and R3 belong to the leaves-of-leaves set, because they are stored at leaves of their source trie and the source trie itself hangs off a leaf in the destination trie. On the other hand, R1 does not belong to the leaves-of-leaves set even though it is stored in a leaf of its source trie because this source trie hangs off a non-leaf node in the destination trie.

It is easy to see that the leaves-of-leaves set of rules are independent. The rule set is thus partitioned into two lists - leafList and otherList, where leafList is the leaves of leaves set, and otherList contains the remaining rules. In the example of Figure 2, R2 and R3 are added to the leafList and R1 is added to the otherList that is to contain the remaining rules.

Our rule placement strategy places all rules in otherList in the ITCAM. Additionally, it may be necessary to place some of the rules in leafList in the ITCAM as well so as to ensure that the remaining rules which will be placed in the LTCAM, satisfy the second requirement for lookup correctness. Thus, for each rule $R$ in leafList it is checked if there exists a higher priority rule in otherList (containing rules for insertion in ITCAM) that overlaps with $R$. If there is, then $R$ is added to the ITCAM. In Figure 2, R3, which is in leafList, is added to the ITCAM instead of the LTCAM, since it has lower priority compared to the otherList rule R1. Thus, in our three rule example of Figure 2, R1 and R3 are added to the ITCAM while R2 is added to the LTCAM. Since the rules assigned to the LTCAM

are independent, they can be entered in any order and there is no need for a priority encoder. Our LTCAM rule selection process guarantees that a match found in the LTCAM during lookup will be of the highest priority. Thus, during lookup, if a match is found in the LTCAM, then the corresponding action can be returned, without waiting for the ITCAM lookup to finish. The ITCAM lookup must go through a priority encoder to resolve multiple matches.

### 3.3 Storing rules in the ITCAM

The rules designated for the ITCAM, are assigned block numbers by creating a priority graph [2] for the rules. A priority graph contains a vertex for each ITCAM rule. 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. The priority graph is used in assigning block numbers to the rules as follows [2]. All vertices, to which there are no incoming edges, are assigned a block number of 0. All children of the vertices with block number zero are assigned a block number of 1 and so on. Rules that are assigned the same block number are grouped together in a single block. These blocks are entered in the TCAM in increasing order of assigned block numbers starting from the lowest address.

### 3.4 Update algorithms

The first step in processing a classifier update is to perform the update operation on the multi-dimensional trie. The second step is to rearrange the existing, overlapping rules in the ITCAM and to move the rules between the two TCAMs as needed to preserve the priorities and to ensure that the two requirements for memberships in LTCAM are still satisfied after the actual update is done.

As the trie is updated, a number of TCAM update requests are generated that are needed to maintain the two requirements for a rule to be in the LTCAM. For example, as a result of inserting a new rule, some existing LTCAM rules may no longer be leaves-of-leaves rules. For these rules, ITCAM insert requests are generated to insert them to the ITCAM and LTCAM delete requests are generated to delete them from the LTCAM. The ITCAM inserts must happen first so that a search returns consistent results.

Similarly, when a rule is deleted from the trie, some existing rules may now become leaves-of-leaves rules. For these rules it is further checked if they are the highest priority rules in the set of overlapping rules. If they are, then the second requirement for membership in LTCAM gets satisfied and for these rules a number of LTCAM insert and ITCAM delete requests are generated.

Thus, as a result of updating the trie a number of ITCAM and LTCAM insert and delete requests are generated. These requests still remain in the control-plane and are processed further as described below, before they are applied to the TCAMs.

#### 3.4.1 Other control-plane operations for updates

At this stage, the TCAM requests generated as a result of updating the multi-dimensional trie are further processed and run through the memory management algorithms for the ITCAM and the LTCAM. The memory management schemes from DUOS may be used here. For the ITCAM of PC-DUOS we implemented the DLFS_PLO scheme, as it had the best results in DUOS with respect to the required number of moves for prefix insertion and deletion. Also, for operations on the ITCAM, we use dynamically sized arrays for $top$, $bot$, $AV$ and $prev$ variables [9] in the implementation, since rule blocks are randomly added and deleted – unlike same length prefix blocks in packet forwarding that results in a maximum of 32 blocks for IPv4.

**ITCAM.insert** :
To insert a new rule in the ITCAM, firstly, the rule is added to the priority graph and a block number is assigned to it. It may be required to adjust the block numbers of existing rules overlapping with the new rule. Next, DLFS_PLO insert and delete requests are generated for the existing rules, whose block numbers are changed, to move them to the corresponding rule blocks. Secondly, a new DLFS_PLO insert request is generated for the new rule to insert it in the desired rule block. The method to assign and adjust block numbers during updates is explained in [2]. A DLFS_PLO insert or delete request may invoke a number of $move$ operations to create a free slot at a specific block or to manage freed slots.

Thirdly, to ensure that the second requirement for membership in LTCAM still holds for all the LTCAM rules, the priority of the new ITCAM rule is compared with that of the existing LTCAM rules. All overlapping LTCAM rules with priority lower than that of the new rule, must be inserted in the ITCAM using the first and second steps above and then deleted from the LTCAM.

**ITCAM.delete** :
The steps for deleting a rule from the ITCAM are as follows. Firstly, the rule is deleted from the priority graph, following which a DLFS_PLO delete request is generated to delete the rule from the ITCAM. Secondly, every ITCAM rule that now becomes leaf of leaves and has the highest priority among all overlapping rules, are moved to the LTCAM.

**LTCAM.insert and LTCAM.delete** :
For an LTCAM insert, the $AV$ variable in the control plane [9] is used to access an available slot and an insert request is generated for the new rule at that slot.

Similarly, to delete a rule from the LTCAM, a delete request is generated to set the valid bit for the rule slot to zero and to store the current $AV$ variable on the SRAM word for the freed slot and then $AV$ is updated to the address of the slot currently being freed. LTCAM inserts do not need rearrangement of existing LTCAM rules since there is no overlap with the existing rules.

## 4 Experimental Results

We evaluated the performance of PC-DUOS in updating packet classifiers using a set of synthetic IPv4 classifiers generated using Classbench [7]. Our experiments were run on an x86 Linux box with 64bit, 1200MHz CPU. Figure 3 shows the details of these datasets. The first column in this figure presents the names of the classifiers, the second column shows the seed files in Classbench from which these tests were derived, the third column shows the number of rules in each of the classifiers, and columns four to six give the number of insert and delete operations in the update traces as well as the total number of update operations for each dataset. We used 7 seed files based on access control lists (acl), firewalls (fw) and IP chains (ipc) to generate 7 classifiers. Each rule consists of the fields: source address, destination address, source port range, destination port range, protocol. We created an update trace from the classifiers by marking rules for insertion/deletion randomly, and later removing the rules marked for insertion. The corresponding insert, and delete operations are shuffled and then written to the update file. To update the classifier in a consistent fashion, change operations are implemented as a succession of insert and delete commands, that is insert the new, changed rule first and then delete the existing rule being changed. Hence, in our update simulation we did not add change operations.

| DataSet | seed_file | #Rules | #Inserts | #Deletes | #Total-updates |
|---|---|---|---|---|---|
| test1 | acl1_seed | 30075 | 69300 | 29700 | 99000 |
| test2 | fw1_seed | 7989 | 28800 | 7200 | 36000 |
| test3 | ipc1_seed | 15338 | 34300 | 14700 | 49000 |
| test4 | acl2_seed | 53970 | 45000 | 45000 | 90000 |
| test5 | fw5_seed | 5571 | 45900 | 5100 | 51000 |
| test6 | acl4_seed | 34254 | 5000 | 5000 | 10000 |
| test7 | ipc1_seed | 5165 | 94050 | 4950 | 99000 |

**Figure 3. Synthetic classifiers and update traces used in the experiments**

The smoothness and scope parameters in ClassBench used to generate the packet classifiers are as follows- smoothness: 2; address scope: 0.5, application scope: -0.1. From the description of these parameters in [7], the values used ensure that the generated files contain a fair amount of overlapping rules. Note that the number of overlapping rules directly impact the update performance as the overlapping rules have to be rearranged to maintain priority ordering in the ITCAM. If we look at the priority graphs for the tests, we find that the number of levels of nodes in these graphs is between 27 and 73. This indicates a close resemblance to real life packet classifiers in terms of the maximum length of a chain of overlapping updates [2].

We have run two sets of experiments on our datasets. The first set uses prefix expansion of ranges[3, 2], whereas the second set uses DIRPE [3] for representing source and destination port ranges. DIRPE is implemented by introducing multi-bit tries for source and destination port ranges. For experiments with DIRPE, we assume that 36 bits are available for encoding each port range in a rule. With this assumption, we use strides 223333 for both source and destination port range tries in 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 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. The memory management scheme DLFS_PLO is used for the STCAM to allot a free slot for rule insertion or to manage a freed up slot following rule deletion. 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).

Figures 4 and 5 show the number of TCAM writes and the total computation time for processing the updates starting from the first to the last update using STCAM and PC-DUOS architectures. The processing time includes everything except the actual time to perform the TCAM writes. The write ratio is obtained by dividing the number of writes in STCAM, by the number of writes (sum of ITCAM and LTCAM writes) in PC-DUOS.

| Data-Sets | #Rules | PC-DUOS | | | STCAM | | write ratio |
|---|---|---|---|---|---|---|---|
| | | #ITCAM writes | #LTCAM writes | Time(s) | #writes | Time(s) | |
| test1 | 30075 | 1301 | 131731 | 10 | 373933 | 3295 | 2.81 |
| test2 | 7989 | 120146 | 72209 | 1635 | 261330 | 2362 | 1.36 |
| test3 | 15338 | 47702 | 54398 | 291 | 126953 | 1124 | 1.24 |
| test4 | 53970 | 98768 | 115130 | 1366 | 287333 | 2763 | 1.2 |
| test5 | 5571 | 181379 | 54357 | 3693 | 514424 | 3964 | 2.18 |
| test6 | 34254 | 32062 | 12412 | 47 | 73516 | 185 | 1.65 |
| test7 | 5165 | 231437 | 108681 | 1611 | 533531 | 6880 | 1.57 |

**Figure 4. Number of TCAM writes in PC-DUOS and STCAM using prefix representation scheme**

We observe from Figures 4 and 5 that there is an improvement in the write ratio with PC-DUOS. The improvement is in the range [1.2, 2.81] for the prefix representation scheme and [1.19, 2.82] for the DIRPE based scheme. A signifi-

| Data-Sets | #Rules | PC-DUOS | | | STCAM | | write ratio |
|---|---|---|---|---|---|---|---|
| | | #ITCAM writes | #LTCAM writes | Time(s) | #writes | Time(s) | |
| test1 | 30075 | 1301 | 115092 | 8 | 327675 | 2469 | 2.82 |
| test2 | 7989 | 78628 | 27238 | 928 | 126225 | 935 | 1.19 |
| test3 | 15338 | 44055 | 44681 | 265 | 110346 | 780 | 1.24 |
| test4 | 53970 | 59067 | 89660 | 921 | 205568 | 1725 | 1.38 |
| test5 | 5571 | 82656 | 30702 | 2012 | 297624 | 2702 | 2.63 |
| test6 | 34254 | 12391 | 9639 | 31 | 43017 | 118 | 1.95 |
| test7 | 5165 | 207274 | 89389 | 1449 | 521653 | 4808 | 1.76 |

**Figure 5. Number of TCAM writes in PC-DUOS and STCAM using DIRPE[3]**

cant improvement in update processing time is seen for the test1, which is faster by 329 times for the prefix representation scheme and 308 times faster that the STCAM scheme when DIRPE is used. The test1 is derived from a seed file for access control lists. It contains a large percentage of very specific rules for access control. For example, following is a less specific rule - block access to a resource for packets originating from any source address and any port number. Such rules are stored in the ITCAM of PC-DUOS and for test1, only 2% of the rules were assigned to ITCAM. ITCAM updates are more CPU intensive as they require updating the priority graph and invoking the memory management scheme to create empty slots at the right place. LTCAM updates, on the other hand, are faster since rules are independent and can be added to any free slot and hence a simpler memory management scheme.

Thus, the performance of PC-DUOS depends on the number of rules in the ITCAM and also on the number of rules that overlap with a new rule being inserted. For example, if a newly inserted rule overlap with a large number of rules, it may require the overlapping rules to be moved to different blocks to maintain priority relationship. These two factors are primarily responsible for the differences in improvement in processing time between PC-DUOS and STCAM. The varied total time numbers for the tests that we see on the columns marked Time, are due to the two factors above as well as due to differences in the total number of updates being processed for each of these tests.

Assuming the number of lookup requests satisfied by the LTCAM is proportional to the number of rules stored in it, we can estimate the improvement in average lookup time on our datasets. For a single lookup finding a match in the LTCAM, the improvement is 50%[1]. The best case in our dataset is test1, which has 98% of the rules in the LTCAM. The improvement in average lookup time for this test is 49%. The worst case is test2 with DIRPE, which contains only 40% of the rules in the LTCAM, and for this test the improvement in average lookup time is 20%.

## 5 Conclusion

A new scheme PC-DUOS is proposed for packet classifier lookup and update. Two TCAMs are used which have been named as LTCAM and ITCAM. PC-DUOS stores high priority independent rules in the LTCAM. The remaining rules are stored in the ITCAM. During lookup for highest priority rule matching, both ITCAM and 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 a 50% [1] improvement in TCAM search time. The average improvement in lookup time is estimated to be between 20% to 48% for the tests in our data set. The distribution of rules to the two TCAMs makes updates faster by reducing the number of TCAM writes by up to 2.82 times and reducing the control-plane processing time by up to 329 times. The maximum reduction in control-plane processing time is observed for ACL lists.

## 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, http://www.cise.ufl.edu/˜sahni/papers/petcam.pdf

[9] T. Mishra and S.Sahni, DUOS – Simple Dual TCAM architecture for routing tables with incremental update, *IEEE Symposium on Computers and Communications*, 2010.

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

[11] 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.

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