Appears in Proceedings of ICNP, 2003

Packet Classification Using Extended TCAMs

Ed Spitznagel, David Taylor, Jonathan Turner
Applied Research Laboratory, Washington University, St. Louis, MO 63130-4899
{ews1,det3,jst}@arl.wustl.edu

Abstract

CAMs are the most popular practical method for implementing packet classification in high performance routers. Their principal drawbacks are high power consumption and inefficient representation of filters with port ranges. A recent paper [11] showed how partitioned TCAMs can be used to implement IP route lookup with dramatically lower power consumption. We extend the ideas in [11] to address the more challenging problem of general packet classification. We describe two extensions to the standard TCAM architecture. The first organizes the TCAM as a two level hierarchy in which an index block is used to enable/disable the querying of the main storage blocks. The second incorporates circuits for range comparisons directly within the TCAM memory array. Extended TCAMs can deliver high performance (100 million lookups per second) for large filter sets (100,000 filters), while reducing power consumption by a factor of ten and improving space efficiency by a factor of three.

1. Introduction

Packet classification is a key technology for modern high performance routers. Packets received at a router input are classified to determine both the output port the packet should be sent to and to determine what, if any, special handling it should receive. Packet classification can be used to provide expedited forwarding of certain types of packets, to enforce security restrictions or to trigger traffic monitoring. The growing complexity of the Internet is creating new applications for packet classification, placing additional demands on the packet classification subsystem of routers and other packet handling devices.

In the general packet classification problem, packets are classified according to a set of packet *filters*, which define patterns that are matched against incoming packets. Typically, packet filters specify possible values of the source and destination address fields of the IP header, the protocol field (often including flags) and the source and destination port numbers (for TCP and UDP). The address fields are often specified as *address prefixes*, although arbitrary *bit masks* of the address fields are commonly allowed in packet filters and this feature is used in real filter sets, although relatively infrequently. Filters typically specify a range of port numbers for matching packets. Protocols can be either specified exactly or as a *wildcard*. Some systems allow protocol values to be specified by bit masks as well, although it's not clear how useful that feature is.

A small example of a filter set appears in Figure 1. Here, the address fields are shown as four bits, rather than 32 to simplify the example. A dash in an address field indicates a bit position where the mask bit is zero. A dash for an entire entry indicates a wildcard which is matched by any packet. When a packet is received, the filter set is consulted to find the *first* matching filter in the set. The packet is then processed according to the specified action. So for example, a packet with a source address of 0101, a destination address of 0011, a protocol field specifying TCP, a source port of 4 and a destination port of 6 would be forwarded to output port 5, since it matches the second filter in the set, but not the first. On the other hand, a packet with a source address of 1111 and a destination address of 0100 would be dropped (regardless of other fields), since the first matching filter is number 6.

Historically, the applications of *general* packet classification have been limited to relatively low performance systems with relatively small numbers of packet filters. This has made it possible to match every incoming packet against the ordered list of filters and stop when the first matching

	source address	dest address	protocol	source port	dest port	action
1.	1-01	01--	TCP	2-4	7	fwd 3
2.	01--	0--1	TCP	3-9	2-6	fwd 5
3.	110-	10--	UDP	1-7	4-6	fwd 2
4.	1101	101-	ICMP	-	-	fwd 7
5.	00-1	010-	UDP	4	5	fwd 0
6.	111-	01--	-	-	-	drop
7.	0101	----	-	-	-	fwd 4
8.	1---	0---	-	-	-	drop

Figure 1. Example packet filter set

This work supported by the National Science Foundation, ANI-9813723 and the Defense Advanced Research Projects Agency, Contract N660001-01-1-8930.

filter in the list is encountered. This method does not scale effectively to high performance systems that must process tens of millions of packets per second and that may have much larger filter sets. While currently, most general packet filter sets are fairly small (most have a few hundred entries and very few exceed a few thousand), the size is expected to grow substantially in the future.

The packet classification problem has been studied extensively in recent years. One early effort [13] proposed a *grid-of-tries* to look up 2D filters defined on source and destination address. While very effective for 2D filters, it could not be applied directly to larger numbers of dimensions. The *tuple-space search* technique [14] is another general approach, in which filters are separated into classes based on the number of bits specified in each dimension. This allows a given class to be probed quickly using hashing, but since many classes may have to be probed for a given packet, it does not yield very high performance. The *Recursive Flow Classification* (RFC) algorithm [GU99a] has received much attention in recent years. RFC trades-off memory space against time to achieve faster lookups. While this is a legitimate choice, the space efficiency of RFC can be surprisingly poor. It can use more than a kilobyte per filter, roughly 50 times the memory needed to represent the filter. Another recent algorithm, *Hicuts* [GU99b] is similarly profligate in its use of memory. The *Extended Grid of Tries* [2] and *Hypercuts* [17] algorithms are the first algorithms for the general problem that show some promise of achieving high performance, without requiring excessive amounts of memory.

Perhaps the most popular method for packet classification problem in practice, is to use *Ternary Content Addressable Memory* (TCAM). TCAMs stores data patterns in the form of (value, bit mask) pairs. A *query word* can be simultaneously compared against all the stored patterns. A query word q is said to *match* a stored pattern (v,m) if $q \,\&\, m = v \,\&\, m$, where the ampersand denotes the bit-wise logical and operation. One bit of TCAM storage can be implemented using 16 transistors [MO94] compared to 6 transistors for a word of SRAM. This 2.7x penalty, makes TCAMs less attractive than SRAM-based algorithms that use the same number of bits. However, as discussed earlier, high performance algorithms using SRAM typically use very large amounts of memory per stored filter, which offsets the cost advantage of SRAM.

TCAMs suffer from two other shortcomings, in addition to their relatively high cost per bit. First, TCAMs require large amounts of power, more than 100 times the power of a similar amount of SRAM. They can account for a major part of the power consumption of a router line card. A recent paper [11] showed how *partitioned TCAMs* could significantly reduce TCAM power consumption in IP route lookup. While this is of some interest, the availability of efficient SRAM-based route lookup algorithms [3, 12, 15, 18] limits its impact. In this paper, we explore how similar ideas can be applied to the more difficult problem of general packet classification. Another significant shortcoming of TCAMs is their inability to efficiently handle filters containing port number ranges. Such filters must be handled using multiple TCAM entries, and in the worst-case, it may take *hundreds* of TCAM entries to represent a single filter. We propose an extension to TCAMs that enables them to handle port ranges directly and argue that the added implementation cost of this extension is amply compensated by the improved handling of port ranges.

filter set	filters	with ranges	TCAM entries	storage efficiency
1	279	26	949	29%
2	183	24	553	33%
3	68	12	128	53%
4	158	10	418	37%
5	264	176	1638	16%

Figure 2. Effect of ranges on TCAM efficiency

Section II describes the extensions to TCAMs that are needed to enable high performance and cost-effective solutions to the general packet classification problem. Section III describes a general algorithm for organizing a filter set in an extended TCAM to enable fast lookup. Section IV briefly presents the method we use to generate large packet filter sets, which reflect the characteristics of the much smaller filter sets that are typically available for use by researchers. In Section V we present results evaluating the performance of our algorithm, under a wide range of conditions. Concluding remarks are provided in Section VI.

2. Extended TCAMs

Ternary CAMs are perhaps the most popular implementation method for packet classification in high performance routers. TCAMs are becoming available in configurations with up to 18 Mbits, roughly half the size of the largest SRAMs. An 18 Mbit TCAM offers enough storage for up to 128K IPv4 filters, which is large enough to meet most near term needs for general packet classification solutions.

As discussed above TCAMs have two major drawbacks. First, they consume a large amount of power, and second, they are inefficient when applied to filters with port number ranges. Some TCAMs have a feature that allows a query to be applied to a subset of the TCAM entries, instead of the entire set. Reference [11] has shown how such *partitioned TCAMs* can provide a lower power solution to the problem of IP lookup. We extend the partitioned TCAM concept and show that if we organize the set of filters in this extended TCAM appropriately, we can perform a lookup for a single packet, using a limited number of the TCAM blocks, rather than the entire TCAM, reducing the power consumption by more than an order-of-magnitude. To make this strategy effective, the TCAM must have a fairly large number of independent storage blocks. For example, we might organize a 128K filter TCAM into 512 blocks of 256 filters each. A lookup algorithm that limited its search to no more than say ten blocks would use just a few percent of the power that

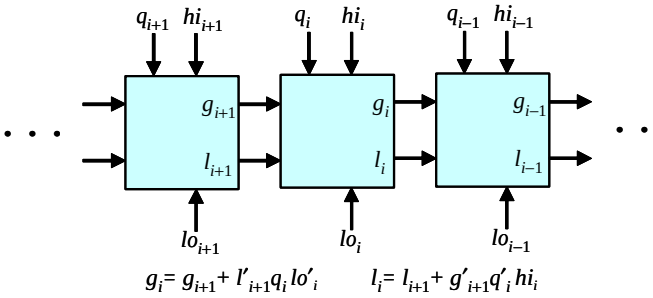

Figure 3. Iterative structure of range check circuit

would be required if every lookup were applied to the entire TCAM.

Our modification to the TCAM architecture adds a special storage block called an *index* to an ordinary partitioned TCAM. Each word in the index is associated with one of the main storage blocks. Conceptually, when a lookup is performed on the modified TCAM, the index is consulted first, and then for each word in the index that matches the query word, a lookup is performed on the corresponding storage block. The lookups in the storage blocks are done in parallel and the address of the first matching entry in each block is returned. To resolve matches across multiple blocks, we associate a priority with each filter, which corresponds to its position in the original ordered list. The priority field of the first matching filter in a block is returned along with the action field of the matching filter. The priorities are compared to determine which of the matching filters has the highest priority. The action field of this filter is returned as the result of the lookup. The modified TCAM can be pipelined to maintain the same operating frequency as a conventional TCAM. The index lookup is done on the first clock tick, followed by the lookup in the storage block on a second clock tick, followed by the priority resolution on a third clock tick. An extended TCAM with a clock rate of 100 MHz can perform 100 million lookups per second.

As mentioned above, a TCAM lookup is performed by comparing a *query word* against a set (value, mask) pairs. A word *q matches* a stored pair (*v*,*m*) if $q \,\&\, m = v \,\&\, m$. The (value, mask) matching paradigm works well for matching IP addresses, but is not well-suited to matching port number ranges. The usual way to handle a port number range in a filter, is to replace each filter with several filters, each covering a portion of the desired port range. This requires splitting the range into smaller ranges that can be expressed as (value, mask) pairs. For example, the range 2-10 can be partitioned into the set of patterns 001-, 01--, 100- and 1010, where the dashes denote bit positions where the mask is zero.

In general, any sub-range of a *k* bit field can be partitioned into $2(k-1)$ such patterns. Since port numbers are 16 bits each, this means that a range in either the source or destination port number field can require as many as 30 distinct TCAM entries. The problem becomes much worse if ranges

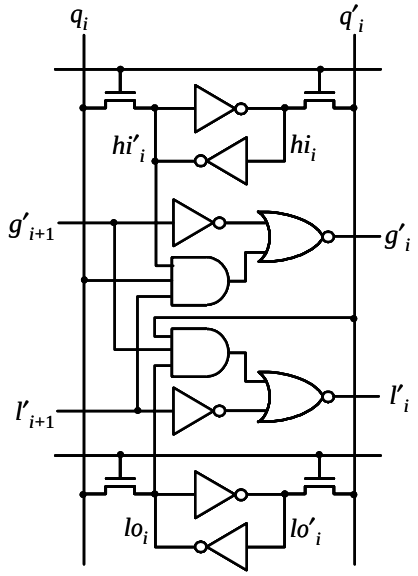

Figure 4. Range check sub-circuit

are present in both the source and destination port number fields. In this case, we need a filter for *all combinations* of the sub-ranges for the two fields. This means that a single packet filter may require 900 TCAM entries. In practice, things aren't nearly this bad, but they are still bad enough. Filter sets often use the port range 1024-65,535. This can be split into just six filters, but a filter containing this range in both the source and destination port number fields still needs 36 TCAM entries. If even 10% of the filters in a large filter set contained such port number ranges, the average number of TCAM entries per filter would be 4.5, greatly increasing the effective cost of a TCAM-based solution. (We note that reference [5] describes a more efficient way to represent a *set* of ranges, but this method cannot be applied to matching multi-dimensional filters in TCAMs.)

To better understand the magnitude of this issue, we studied five real-world filter sets and determined the minimum number of TCAM entries required to represent the set, assuming that port ranges were decomposed in the most efficient way. The results appear in Figure 2, which shows the number of TCAM entries required to represent each of the filter sets and the resulting storage efficiency. The storage efficiency ranges from as little as 16% to 53%, with an average of 34%, tripling the effective cost of TCAM-based solutions.

One way to handle port ranges better is to extend the TCAM functionality to directly incorporate port range comparisons in the device. Such a TCAM would store a pair of 16 bit values (*lo*,*hi*) for each port number field and include circuitry to compare a query word *q* against the stored values. Figure 3 shows the iterative structure of the required *range check* circuit. The circuit consists of a separate stage for each bit and the comparison proceeds from the most significant bits to the least significant bits. The inter-stage

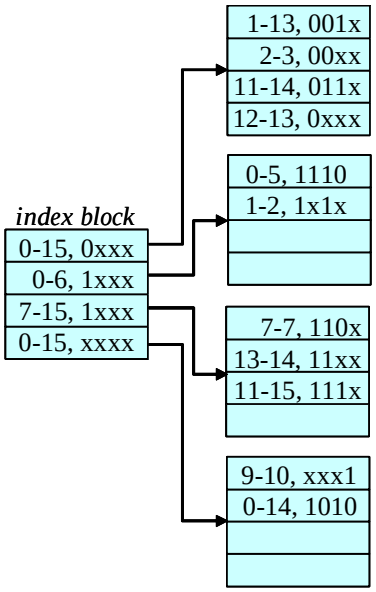

Figure 5. Search in an extended TCAM

signal g_i is high whenever the value of the high order bits of q (down to bit i) are numerically larger than the high order bits of *hi*. Similarly, the inter-stage signal l_i is high whenever the value of the high order bits of q are smaller than the high order bits of *lo*. The query value is in range if both g_0 and l_0 are low.

Figure 4 shows a circuit implementing the required logic for each stage, along with the storage elements for *lo* and *hi*. A standard CMOS implementation of this circuit uses 32 transistors, twice as many as the standard TCAM storage cell. However, the impact of this on the cost of the entire TCAM is much smaller. The two port number fields represent 22% of a 144 bit TCAM word, suitable for IPv4 (allowing 16 bits for the protocol field including flags, plus 32 bits for an action field and priority). Doubling the transistor count for this portion of the TCAM increases the total number of transistors per word by 22%. While this is a non-trivial increase, it is a far smaller price to pay than the price implied by the inefficient representation of port ranges in standard TCAMs. In any application where port number ranges are present in more than a few percent of the filters, the added cost is easily justified. We use the term *extended TCAM* to refer to a TCAM with both of the modifications described.

3. Classifiying Packets with Extended TCAMs

To use extended TCAMs for packet classification, we need to partition the filter set into storage blocks and then associate each storage block with an appropriate *index filter*. The extended TCAM search will first identify all matching index filters, and then query the storage blocks associated with those matching index filters. This process is illustrated in Figure 5, which shows a set of two dimensional filters on four bit fields (one defined using ranges, one defined using bit-masks) and the organization of those filters into TCAM blocks with an index block to the left (the figure does not show the priority and action fields). To perform a lookup on a packet with field values (2,10), we first check the index block and discover that the second and fourth index filters match the packet. Searching the second block, we find the matching filter (1-2, 1x1x) and searching the fourth block, we find the matching filter (0-14, 1010). In this example, the TCAM blocks are large enough for just four filters each, but a realistic implementation of the method would use TCAMs with blocks capable of storing hundreds of filters. Also note that the index block need not have the same number of entries as the storage blocks.

The key to making the search power-efficient is to organize the filters so that only a few TCAM blocks must be searched in order to find the desired matching filter for a given packet. We define the problem of organizing the filters precisely below, but first we introduce the following definition.

Definition. Let f_1 and f_2 be filters defined on the same multi-dimensional space. We say that f_1 *covers* f_2 if the region of the space that is defined by f_1 completely contains the region defined by f_2. Similarly, we say a set of filters F covers a filter f if the region defined by the union of the filters in F completely contains the region defined by f.

Filter Grouping Problem. Given a set F of filters and integers k, m and r, find a set S of at most m filters and a bipartite graph $G = (V,E)$ with $V=F \cup S$ and $E \subseteq F \times S$, that satisfy the following conditions.

- for every f in F, the neighbors (in G) of F cover f,
- for every s in S, the degree (in G) of s is at most k,
- no point in the multi-dimensional space on which the filters are defined is covered by more than r members of S.

S defines the set of index filters. The graph specifies the assignment of original filters to index filters and their associated storage blocks. The degree of a vertex for an index filter is equal to the number of filters in the storage block associated with that index filter. The bound k, on the degree, limits the number of filters per block; the bound m, on the size of S, limits the number of index filters and hence the number of TCAM blocks needed to hold the index; and the bound r, on the number of index filters covering any point in the space, limits the number of TCAM storage blocks that must searched (in addition to the index). The problem can be converted into an optimization problem, by minimizing any one of the three parameters, while leaving the other two as bounds.

We use a heuristic filter grouping algorithm to organize the filters. The algorithm proceeds in a series of *phases*. Each phase recursively divides the multi-dimensional space into ever smaller *regions*, so each phase produces a separate

partition of the space. During each *step* in a phase, a region in the space is selected and divided into two parts with approximately the same number of filters. The algorithm returns a set *S* of index filters and a subset of the original filter set for each of the index filters.

In all but the last phase, each sub-region created in that phase is associated with a set of filters that are contained entirely within the sub-region. These filters are *assigned* to this enclosing sub-region and are then ignored in later phases. The last phase also partitions the space, but some of the filters that remain at this stage, may not fall entirely within any of the sub-regions. Such filters are assigned to all the sub-regions created in the last phase which they intersect, meaning that there will be multiple TCAM blocks containing copies of these filters.

A basic operation of the algorithm is to *cut* a region *r* of the multidimensional space into two sub-regions r_1 and r_2 along one of the multiple dimensions. We represent each region by an index filter. To cut a filter along a "bit-mask dimension", we select one of the mask bits for that dimension that is equal to zero, change it to 1 in both of the sub-regions and assign the corresponding bit of the value field to 0 in one sub-region and 1 in the other. All other fields of r_1 and r_2 are inherited from *r*. To cut a filter along a "range dimension", we simply divide a range (*lo*,*hi*) into two sub-ranges (*lo*,*m*) and (*m*+1,*hi*) where $lo \le m < hi$.

Let F_i be the set of filters that remain to be processed at the start of phase *i* and let S_i be the set of sub-regions (index filters) created by the algorithm during phase *i*. At the start of phase *i*, we let S_i be the entire multidimensional space. For any given *r* in S_i, let $\sigma(r)$ denote the set of filters in F_i that lie entirely within *r* and let $\chi(r)$ denote the set of filters in F_i that intersect the region defined by *r* but do not lie entirely within *r*. In all but the last phase, we repeat the following step until for every region *r* in S_i, $\sigma(r)$ contains at most βk filters, where *k* is the size of the TCAM block and β is a parameter of the algorithm to be chosen later.

Let *r* be a region in S_i which maximizes $|\sigma(r)|$.

Consider cuts that divide *r* into two sub-regions r_1 and r_2 that satisfy

$$\max\left\{\left|\sigma(r_1)\right|,\left|\sigma(r_2)\right|\right\} \le \alpha\left|\sigma(r)\right|$$

where α is another parameter, to be chosen later. Among all such cuts, select one that maximizes

$$\left|\sigma(r_1) \cup \sigma(r_2)\right|$$

Replace S_i with $S_i \cup \{r_1, r_2\}$.

At the end of the phase, for each region *r*, we assign up to *k* filters in $\sigma(r)$ to region *r*. If $\sigma(r)$ contains more than *k* filters, we select the *k* filters that have the largest volume in the multi-dimensional space. The filters assigned to region *r* will share a TCAM storage block in the final result and *r* will be

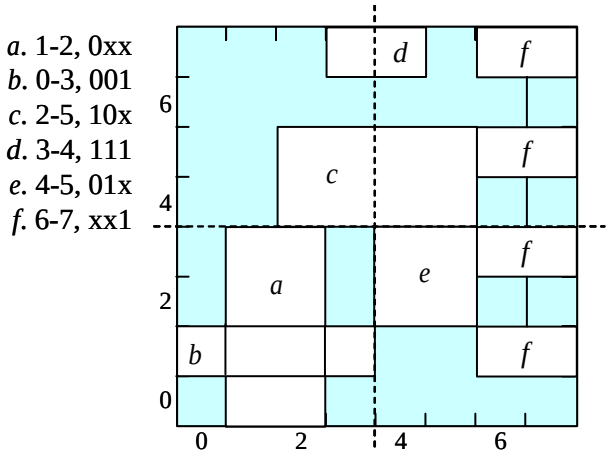

Figure 6. One step of the filter grouping algorithm

the corresponding index filter. All filters that are assigned to a region in phase *i* are excluded from the filter set F_{i+1} used in the next phase.

Figure 6 illustrates one of the basic steps. In this example, there are two dimensions. The horizontal axis is associated with a range dimension, and the vertical axis with a bit-mask dimension. Two candidate cuts are indicated by the dashed lines. Both cuts satisfy the splitting condition when α=2/3. In this case, the algorithm would select the horizontal cut, since this cut puts five of the six filters entirely within one of the two sub-regions, while the vertical cut places only four within one of the sub-regions.

The basic step in the last phase is similar.

Let *r* be a region in S_i that maximizes $\left|\sigma(r) \cup \chi(r)\right|$.

Consider cuts that divide *r* into two sub-regions r_1 and r_2 that satisfy

$$\max\left\{\left|\sigma(r_1) \cup \chi(r_1)\right|,\left|\sigma(r_2) \cup \chi(r_2)\right|\right\} \le \alpha\left|\sigma(r) \cup \chi(r)\right|$$

Among all such cuts, select one that maximizes

$$\left|\sigma(r_1) \cup \sigma(r_2)\right|$$

If there are no cuts that satisfy this condition, select a cut that satisfies the condition used in the earlier phases.

Replace S_i with $S_i \cup \{r_1, r_2\}$.

The last phase terminates when all sub-regions *r* in S_i satisfy

$$\left|\sigma(r) \cup \chi(r)\right| \le k$$

or a splitting operation results in no decrease in $\left|\sigma(r) \cup \chi(r)\right|$. In the latter case, the last phase "fails" and the algorithm terminates. We can then re-run the algorithm, specifying a larger number of phases. We can continue in this manner, specifying more and more phases until the algo-

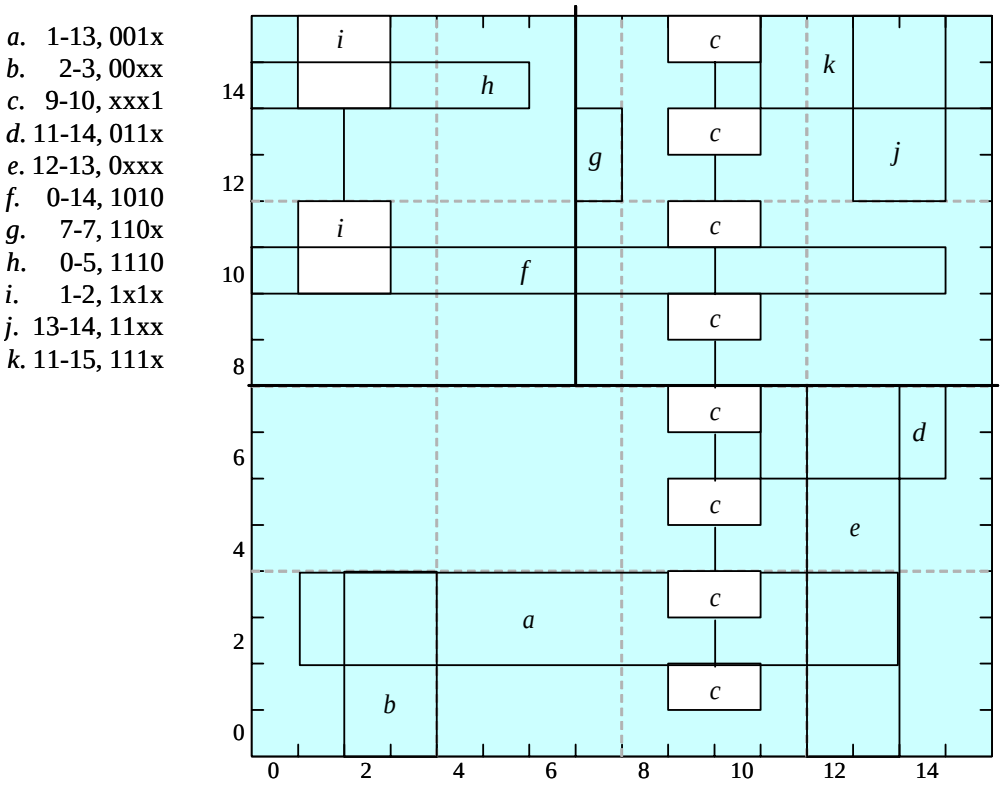

Figure 7. Example of partitioning algorithm

rithm successfully runs to completion. Alternatively, we can avoid most of the redundant computation implied by this procedure, by simply rolling back to the start of the last phase on failure and continuing from that point.

After the (successful) completion of the last phase, we let $S = \bigcup_i S_i$. The elements of *S* define our index filters. For each *r* in *S*, the filters that were assigned to *r* during the execution of the algorithm are placed in the storage block associated with *r*.

Figure 7 shows the result of an execution of the algorithm with *k*=4. The first phase results in the three subregions (0-15, 0xxx), (0-6, 1xxx) and (7-15, 1xxx), containing 4, 2 and 3 filters, respectively. The final phase terminates with two filters in the region (0-15, xxxx). This produces the TCAM configuration in Figure 5.

4. Evaluating Packet Classification Algorithms

It has been observed that "real" packet classification filter sets exhibit a considerable amount of structure. In response, several algorithmic techniques have been developed which exploit this structure to accelerate search time or reduce storage requirements. Consequently, the performance of these approaches is subject to the statistical composition of the filter set.

Despite the influence of filter set composition on the performance of packet classification algorithms and devices, no benchmark suite of filter sets or systematic methodology exists for standardized performance evaluation. Due to security and confidentiality issues, access to large "real" filter sets for statistical study and performance measurements of new classification techniques has been limited to a small subset of the research community.

Performance evaluations using real filter sets are restricted by the size and structure of the sample data. Some researchers have proposed ad hoc methods, such as independently selecting filter fields from a one-dimensional distribution, to generate synthetic filter sets or modify the composition and number of filters in the filter set.

In order to facilitate future research and provide a foundation for a meaningful benchmark, we developed a technique for generating large synthetic filter sets which model the statistical structure of a seed filter set [16]. Along with scaling filter set size, the tool provides mechanisms for systematically altering the number and composition of filters as depicted in Figure 8. Two adjustments, *smoothing* and *scope*, provide high-level adjustments for filter set generation and an abstraction from the low-level statistical characteristics.

Previous work reported many statistical characteristics of filter sets useful for constructing fast search algorithms. In a similar fashion, we extract statistics from seed filter sets in order to construct larger synthetic filter sets with similar structure and characteristics. Selection of the relevant statistics is based upon experience garnered from a study of five

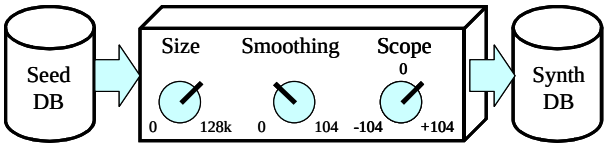

Figure 8. Conceptual view of the filter set generator

firewall filter sets of modest size. This study paid particular attention to the five-dimensional structure of the filter sets and the "correlation" among fields. In the context of our discussion, "correlation" is generally defined as the probability that two fields share the same value and will be defined more explicitly for the contexts in which it is used.

The first important characteristic of seed filter sets is the *tuple* distribution. We define the filter 5-tuple as a vector containing the following fields.

- $t[0]$ - source address prefix length, [0...32]
- $t[1]$ - destination address prefix length, [0...32]
- $t[2]$ -, source port range width, the number of port numbers covered by the range, $[0...2^{16}]$
- $t[3]$ - destination port range width, the number of port numbers covered by the range, $[0...2^{16}]$
- $t[4]$ - protocol specification, Boolean value denoting whether or not a protocol is specified, [0,1]

The tuple defines the five-dimensional structure of the filter without specifying the actual values for address prefixes, port ranges, and protocol fields. In order to provide a high-level measure of the specificity of the tuples in a seed filter set, we define a metric, *scope*, to be the base 2 logarithm of the number of possible packet headers covered by the filter.

$$scope = (32 - t[0]) + (32 - t[1]) + \lg t[2] + \lg t[3] + 8(1 - t[4])$$

A seed filter set contains a finite list of tuples which are derived from the filters. From the given seed filters, we generate a tuple distribution consisting of unique tuples and associated weights corresponding to the relative frequency of their occurrence in the seed filter set. For each filter generated for the synthetic filter set, a tuple is selected from the tuple distribution according to the probabilities dictated by the relative weights. We refer to this selected tuple as the *target tuple*.

As filter sets scale in size, we anticipate that new tuple specifications will emerge in the set of unique tuples. In order to allow for this possibility, we enable the creation of new tuples in a structured manner that preserves the structure present in the seed filter set. Using scope as a measure of distance, we would like new tuples to emerge "near" an existing tuple. We define a smoothing parameter, *r*, that controls the maximum distance between a new tuple and the original tuple from which it "spawns". For values of *r* greater than zero, the process selects a radius *i* in the range $[0 \ldots r]$ from a truncated geometric distribution; hence, the probability that we select the target tuple or a tuple near the original tuple is greater than the probability that we select a tuple farther away. Once a radius is selected, the new tuple is generated by randomly selecting a direction of scope adjustment (increase or decrease) for each field, then randomly selecting *i* tuple fields and adjusting the scope by one unit for each chosen field, according to the chosen direction. In order to illustrate the effect of smoothing, the joint source/destination address prefix length distribution for a seed filter set and the resulting synthetic filter set created with $r = 16$ are shown in Figure 9.

In addition to smoothing, we provide control over the average scope of the synthetic filter set via a scope parameter *s*. Tuning the average scope of the synthetic filter set is useful for exploring its effect on the performance and capacity of algorithms as well as modeling different network environments. For example, as the number of flow-specific filters in a filter set increases we expect the average scope of the filters to decrease. The smoothing parameter, *s*, is a relative adjustment to the average scope of the filter set; hence, it may take on values in the range [-104…104]. Adjusting

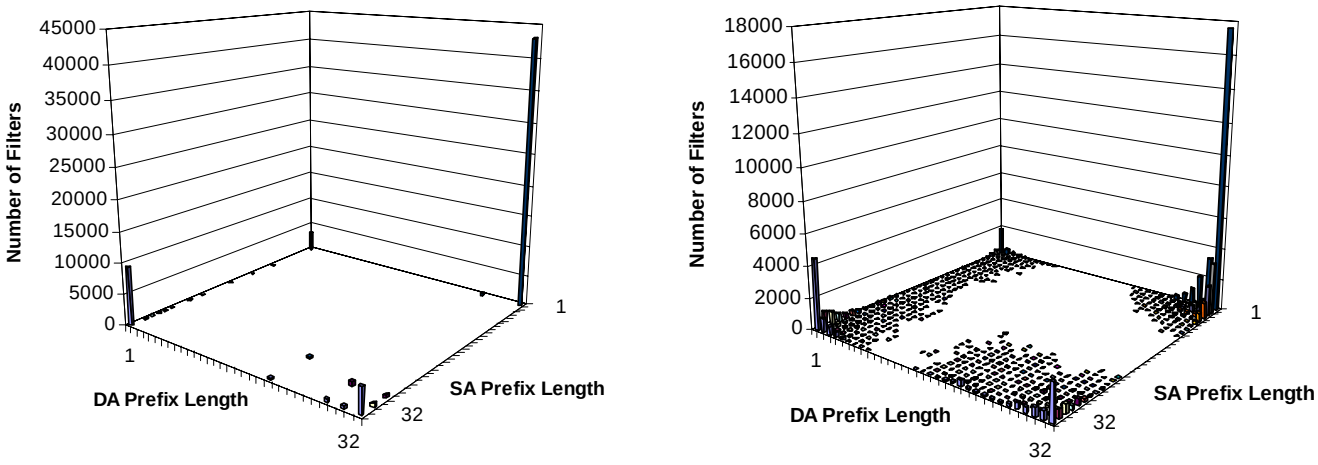

Figure 9. Source and destination prefix length distributions for seed filter set (left) and smoothed version (right)

the average scope of the filter set consists of adjusting the tuple specifications of the filters in a manner very similar to the smoothing adjustment. Once a target tuple is selected and adjusted for smoothing, a random tuple field is selected as the starting point for scope adjustment. The scope of the selected field is modified according to the sign of *s* (scope increased by one or decreased by one). Tuple fields are adjusted in turn until *s* scope adjustments have been made or no more scope adjustments are possible.

The remaining steps of the synthetic generation process essentially "fill in" filter fields as dictated by the tuple. While the tuple captures the joint source/destination address prefix length distribution, it does not characterize the structure of the address trees or similarity between address prefixes. We model address tree structure as follows. For each level in each address trie for the seed filter set, we compute the frequency that a node has zero children, one child, and two children. This is used to create a probability distribution that guides the generation of source and destination addresses. For nodes with two children, we also compute *skew*, which is the ratio of the weights of the left and right subtrees of the node. Subtree weight is defined to be the number of filters specifying prefixes in the subtree. An average skew value for each level is computed and stored with the child probability distributions for the level. We argue that the combination of the child and skew distributions capture the most important characteristics of the data.

Using the child and skew distributions, a source address tree specifying a branching probability for each node is dynamically created during the filter set generation process. The similarity between source and destination address prefixes is captured through a "correlation" distribution, which is defined as the probability that the source address prefix and destination address prefix are the same up to a given level. The child and skew distributions along with the "correlation" distribution is used to dynamically generate the destination address tree during the synthetic generation process.

While the tuple distribution captures the interdependence of port range widths and other tuple fields, it does not characterize the choice of range bounds given a range width. Analysis of the specified port ranges in the set of firewall filter sets yielded an interesting result: all port range widths other than one (fully specified) correspond to a single port range. This property makes selecting of bounds for port ranges given the range width trivial. For fully specified ports, we make a uniform random selection from the range of possible port values. For tuples that fully specify both port ranges, we account for the probability that both source and destination ports are the same.

Finally, we must select a protocol if the tuple dictates a specified protocol field. Clearly, a relationship exists between protocol specifications and port ranges. Based our analyses, we chose to employ three probability distributions for protocol selection. The first distribution is used when both port ranges are unspecified; the second is used when one port range is specified and the other is unspecified; and the third is used when both port ranges are specified. The three distributions may differ significantly within a single seed filter set.

In the next section, we report on how we have used the synthetic filter set generator to evaluate the performance of the filter grouping algorithm for extended TCAMs on a wide range of filter sets. We plan to continue to develop the filter set generator, with the objective of creating a standard benchmark and evaluation methodology for packet classification algorithms and devices.

5. Performance Results

There are three key parameters that affect the performance of the filter grouping algorithm, α, β and the block size, *k*. These parameters affect the two key performance metrics of interest, the power efficiency and the storage efficiency.

As our measure of power efficiency, we use the quantity $(b+sk)/N$, where *b* is the number of number of storage blocks used by the partitioning algorithm (this is equal to the number of entries in the index), *s* is the maximum number of storage blocks that must be searched for any packet (this is equal to the number of phases used by the filter grouping algorithm) and *N* is just the total number of filters in the original filter set. The numerator represents the total number of words in the TCAM that must be searched to perform a lookup (including the index) and the denominator is the number that would have to be searched if we were using a conventional TCAM. We refer to this quantity as the *power fraction* and we seek to make it as small as possible. As our measure of storage efficiency, we use $N/(b+bk)$. Here, the denominator is the sum of the number of words used in the index plus the number used in the storage blocks required by the algorithm.

We start our performance study by determining how the parameter choices affect the performance metrics. Figure 10 shows the dependence on α. We observe that for small values of α, the power fraction is relatively high, but that it drops under .05 for $\alpha \geq .75$. The storage efficiency shows no strong dependence on α but is best for values between .6 and .95. Figure 11 shows how the performance depends on β. Here, we observe that the power fraction increases from about .02 to just over .04. While this is a large relative increase, the absolute magnitude of the power fraction remains acceptably small across the whole range of values. (A power fraction of .05 is small enough to ensure that the TCAM power consumption remains a small fraction of the overall power consumption of a router line card.) The impact of β on storage efficiency is more significant, increasing the storage efficiency from about .72 to over .95.

Figure 12 shows how the performance is affected by the size of the TCAM blocks. We observe that the power fraction is strongly dependent on the block size, but generally

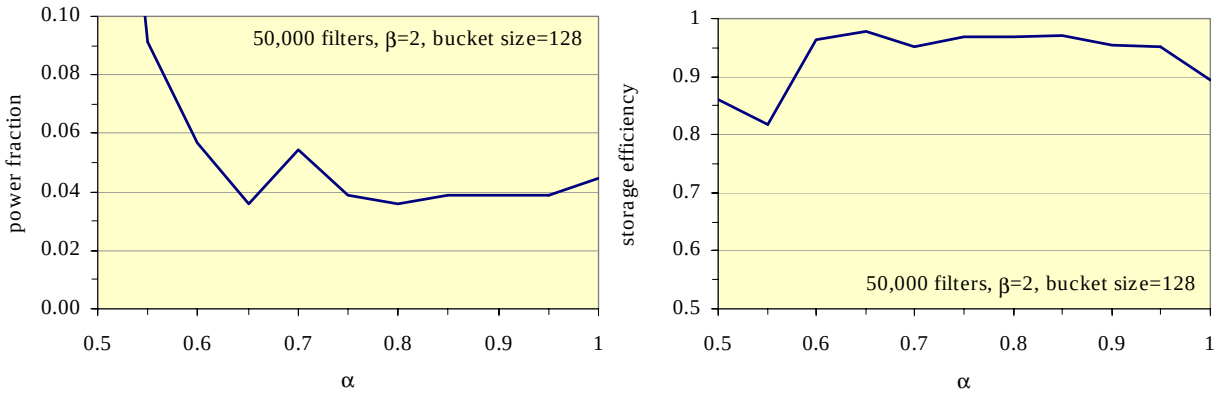

Figure 10. Dependence of filter grouping algorithm performance on α

stays below 5% for the largest filter set for block sizes from 32 to 128. We note that the optimal block size varies with the number of filters, with larger filter sets favoring larger block sizes. The power fraction is determined by the two terms, b/N and sk/N. The first term (which accounts for the power used to search the index) decreases as the block size grows, since with large blocks, we need fewer storage blocks. The second term grows with the block size. While s decreases as the block size k grows, the rate of decrease in s diminishes as k gets large. It's the combination of these opposing effects that produce the sharp minimum seen in the power fraction. It may seem curious that the power fraction is larger for smaller filter sets. This is because the power fraction is measured relative to the power that would be consumed by a TCAM that has exactly the right number of entries for the given filter set. While the absolute power levels are smaller for the smaller filter sets, the reduction that can be obtained using extended TCAMs is also smaller. Figure 12 also shows that the storage efficiency peaks roughly coincide with the best power reductions.

Based on these and other supporting data, we have concluded that α=.8 and β=2 are the best choices for the first two parameters. For the block size, the best choice appears to be the largest power of 2 that is less than $(1/2)N^{1/2}$. Of course, the block size is a little different from the other parameters, since it is a fixed characteristic of the extended TCAM device. So, this rule should be applied using the target device capacity. This will define the worst-case situation for that specific device, since power usage will generally be reduced for smaller filter sets.

Figure 13 shows how the performance metrics are affected by the size of the filter set. Results are shown for two different seed filter sets. As mentioned earlier, the relatively high power fractions for small filter sets reflect the fact that with small filter sets, the power required for a suitably sized TCAM is already small, leaving less room for improvement. For larger filter sets, where power reduction is most important, the power fraction generally stays below 5%. The storage efficiency also seems to improve slightly with filter set size.

Figure 14 shows how the performance varies as a function of the smoothness adjustment. For even small increases in r, we observe fairly marked deterioration in the power fraction and a smaller deterioration in the storage efficiency.

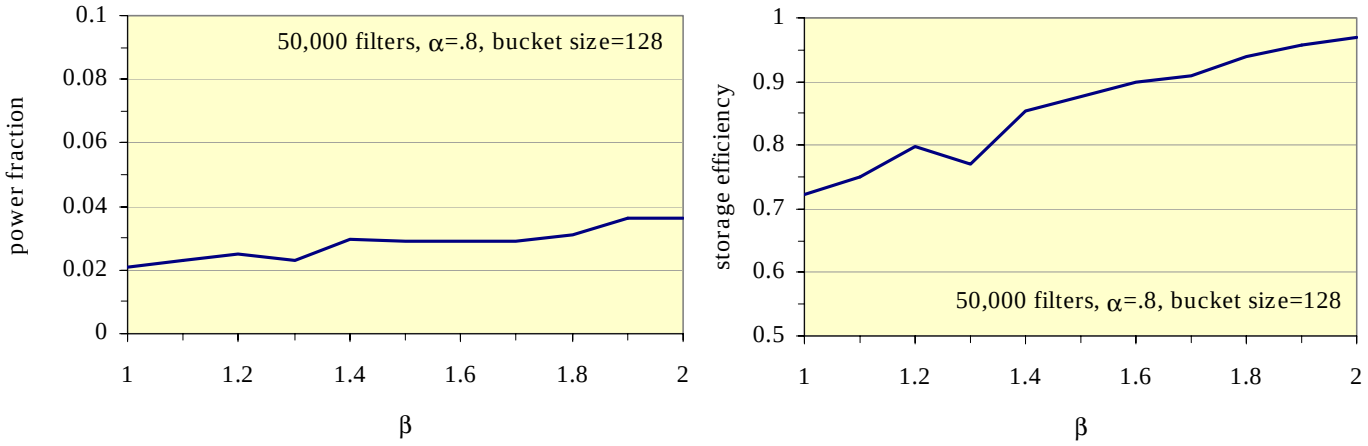

Figure 11. Dependence of filter grouping algorithm performance on β

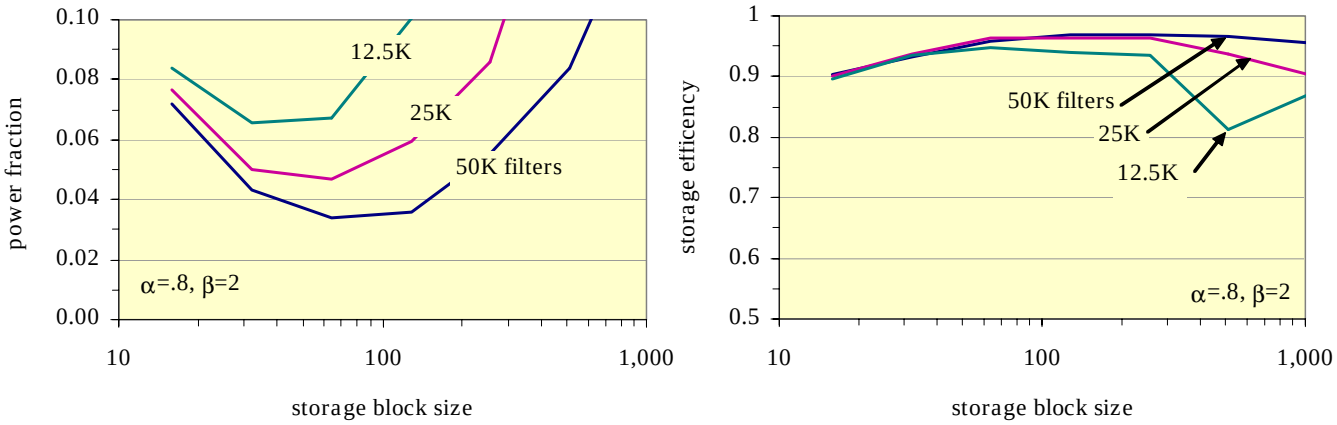

Figure 12. Dependence of filter grouping algorithm performance on block size (*k*)

While the results still represent a 10:1 improvement over the standard TCAM, the degraded performance is somewhat disturbing. At this time, we are not in a position to explain it fully.

Figure 15 shows the effect of the scope adjustment. A value of 0 means that the scope adjustment was not changed. A negative scope adjustment corresponds to more specific filters, a positive scope adjustment to less specific filters. We expect that as packet filter sets grow in size, there will be a natural tendency for the filters to become more specific. Hence, it is worth understanding how packet classification algorithms will perform under these circumstances. For these results, the smoothness adjustment was set to 16, so we observe the same, relatively large power fractions that we noted above. We see a marked improvement in power fraction at a scope adjustment of −16 or less. This seems to make sense intuitively. As filters become more specific, we expect the filter grouping algorithm to separate them more easily. We expect that other packet classification algorithms are also likely to perform better as filter sets become more specific.

6. Concluding Remarks

Extended TCAMs appear to be a promising solution for high performance packet classification. They provide the performance needed for OC-768 links, have moderate power requirements and are far more storage-efficient than high performance algorithmic solutions. There are several additional issues that remain to be explored.

Perhaps the most important issue to be studied is incremental updates. The filter grouping procedure described here is a computationally expensive procedure, which cannot reasonably be performed whenever we add or remove a filter. One way to handle incremental updates, in an extended TCAM with a fixed block size and total capacity is to attempt to structure the filter set so that all storage blocks have roughly equal amounts of free space. This can be achieved by running the filter grouping algorithm with a value of *k* that is smaller than the actual block size. Subsequent filter additions can be handled by assigning them to storage blocks with unused space that cover the same part of the multi-dimensional space that is spanned by the new filter. If new filters follow the same distribution as filters already in the filter set, one can reasonably expect most new filters to fit easily within the structure defined by the algorithm for the original filter set. Eventually, the addition of new filters will cause some storage blocks to fill up. If all the storage blocks that cover a given region of the space become full, it will become impossible to accommodate new filters that span

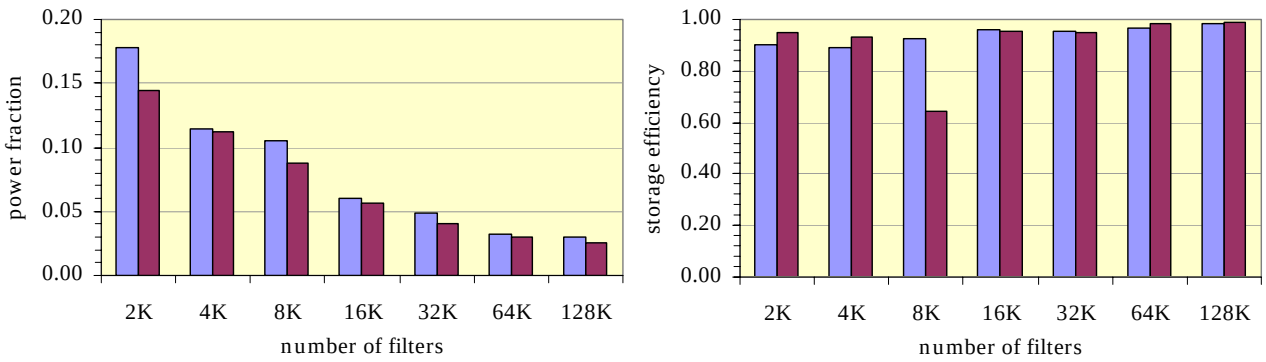

Figure 13. Effect of filter set size for two seed filter sets

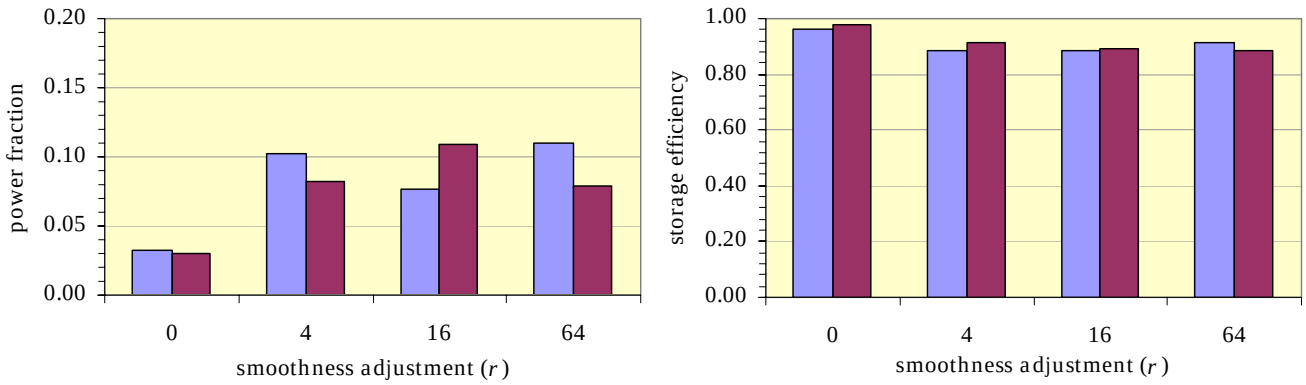

Figure 14. Effect of smoothness adjustment for two seed filter sets

that part of the space. This problem can be avoided by periodically reorganizing the TCAM, after re-running the filter grouping algorithm. Alternatively, it may be possible to reorganize the TCAM incrementally.

It's natural to ask if there might be an advantage to using a two level index, instead of one level. When the block size gets small, the size of the index grows (since there are more storage blocks) and the power used to perform the index lookup becomes significant. Since the index is just a set of filters, we can apply the filter grouping algorithm to the index in exactly the same way, to construct an index into the set of index filters. This allows us to use smaller storage blocks, without being limited by the power consumption of the index. Unfortunately, smaller storage blocks have drawbacks that may negate the power advantage. First, smaller blocks add area overhead to the TCAM device. Second, it seems likely that smaller blocks won't handle incremental updates as well as large blocks. Indeed, the incremental update issue may well motivate the use of larger blocks than one might otherwise choose, based on power considerations alone.

Finally, it should be noted that the filter grouping algorithm could reasonably be used as the basis for a more conventional packet classification algorithm using random access memory. A sequential algorithm can simulate the extended TCAM behavior by searching the index sequentially, then searching the appropriate set of storage blocks (sequentially). Our power reduction measure corresponds directly to the fraction of the filter set that such an algorithm would have to examine. A version of the algorithm using a multi-layer index might well be competitive with other algorithmic solutions. Indeed, the filter grouping algorithm has similarities to the Hypercuts algorithm [17] and was in fact, partly inspired by Hypercuts. A key difference is the use of multiple partitions of the multi-dimensional space. While it's not clear that this approach would lead to a competitive, sequential algorithm, the question appears worthy of further investigation.

REFERENCES

1. Baboescu, F. and G. Varghese. "Scalable packet classification," *Proc. of ACM Sigcomm*, 9/2001.
2. Baboescu, F., S. Singh, and G. Varghese. "Packet Classification for Core Routers: Is there an alternative to CAMs?" *Proceedings of Infocom*, 2003.
3. Brodnik, A., S. Carlsson, M. Degemark, S. Pink.. "Small Forwarding Tables for Fast Routing Lookups," *Proc. ACM SIGCOMM*, 1997.
4. Buddhikot, M. S. Suri, and M. Waldvogel. "Space decomposition techniques for fast layer-4 switching," *Proc. of PHSN*,

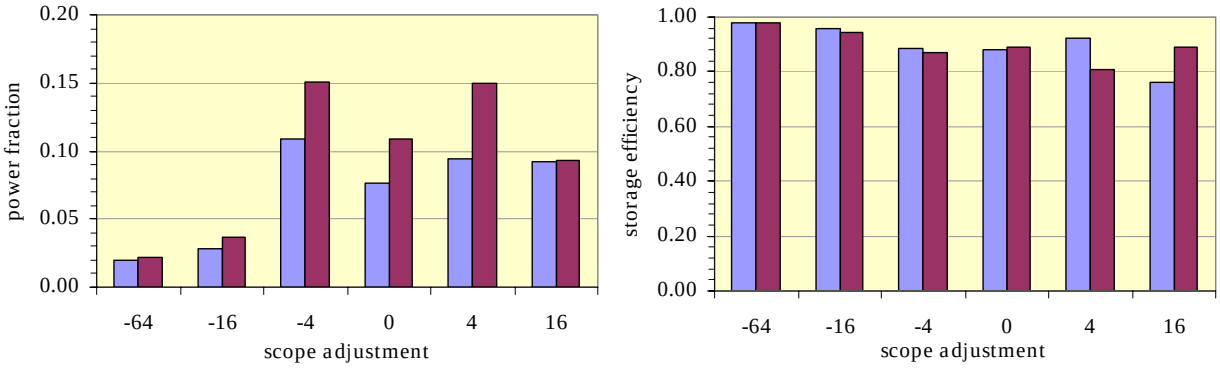

Figure 15. Effect of scope adjustment for two seed filter sets

8/99.

5. Feldman, A. and S. Muthukrishnan, "Tradeoffs for packet classification," *Proc. of Infocom*, 3/2000.
6. Gupta, S. Lin, and N. McKeown, "Routing lookups in hardware at memory access speeds," *Proceedings of Infocom* 4/98.
7. Gupta, P. and N. McKeown. "Packet Classification on Multiple Fields," *Proc. ACM Sigcomm* 9/99.
8. Gupta, P. and N. McKeown, "Packet classification using hierarchical intelligent cuttings," *Proc. of Hot Intercon-nects*, 8/99.
9. Lakshman, T. and D. Stidialis, "High speed policy-based packet forwarding using efficient multi-dimensional range matching," *Proc. of ACM Sigcomm*, 9/98.
10. Montoye, Robert K., "Apparatus for Storing don't care in a content addressable memory cell," United States Patent #5,319,590, 6/94.
11. Narlikar, Girija, Anindya Basu, Francis Zane. "CoolCAMs: Power-Efficient TCAMs for Forwarding Engines," *Proc. of Infocom*, 5/2003.
12. Srinivasan, V. and G. Varghese. "Fast IP Lookups using Controlled Prefix Expansion," *Proc. ACM Sigmetrics*, 6/98.
13. Srinivasan, V., G. Varghese, S. Suri, and M. Waldvogel. "Fast and scalable layer 4 switching," *Proc. of ACM Sigcomm* 9/98.
14. Srinivasan, V., S. Suri, and G. Varghese,. "Packet classification using tuple space search," *Proc. of ACM Sigcomm*, 9/99.
15. Taylor, David E., John W. Lockwood, Todd Sproull, Jonathan S. Turner, David B. Parlour. "Scalable IP Lookup for Programmable Routers," *Proc. of Infocom*, 6/02.
16. Taylor, David E. and Jonathan Turner. "Towards a Packet Classification Benchmark," Washington University Computer Science Department Technical Report, WUCS-03-42, 5/2003.
17. Varghese, George, et. al. "Packet Classification Using Multidimensional Cuts," to appear in *Proceedings of SIGCOMM*, 2003.
18. Waldvogel, M. G. Varghese, J. Turner, B. Plattner. "Scalable High-Speed Prefix Matching," *ACM Transactions on Computer Systems, 2001*.