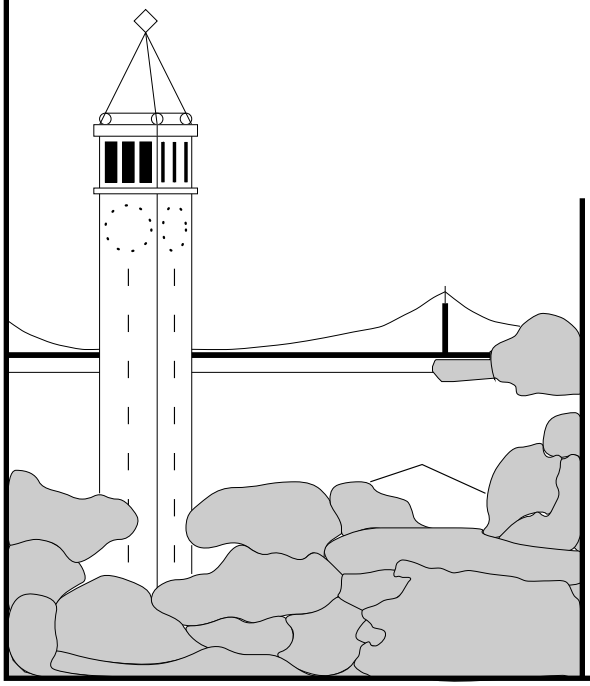
SSA: A Power and Memory Efficient Scheme to Multi-Match Packet Classification

Fang Yu
fyu@eecs.berkeley.edu

T. V. Lakshman
lakshman@research.bell-labs.com

Martin Austin Motoyama
m_moto@uclink.berkeley.edu

Randy H. Katz
randy@eecs.berkeley.edu



Report No. UCB/CSD-5-1388

May 2005

Computer Science Division (EECS)
University of California
Berkeley, California 94720

This technical report is supported by UC Micro grant number 03-041 and 02-032 with matching support from NTT MCL, HP, Cisco, and Microsoft.

SSA: A Power and Memory Efficient Scheme to Multi-Match Packet Classification

Fang Yu[1] T. V. Lakshman[2] Martin Austin Motoyama[1] Randy H. Katz[1]

[1]EECS Department, UC Berkeley
fyu@eecs.berkeley.edu m_moto@uclink.berkeley.edu randy@eecs.berkeley.edu

[2]Bell Laboratories, Lucent Technologies
lakshman@bell-labs.com

Abstract

New network applications like intrusion detection systems and packet-level accounting require multi-match packet classification, where all matching filters need to be reported. Ternary Content Addressable Memories (TCAMs) have been adopted to solve the multi-match classification problem due to their ability to perform fast parallel matching. However, TCAM is expensive and consumes large amounts of power. None of the previously published multi-match classification schemes is both memory and power efficient. In this paper, we develop a novel scheme that meets both requirements by using a new Set Splitting Algorithm (SSA). The main idea of SSA is that it splits filters into multiple groups and performs separate TCAM lookups into these groups. It guarantees the removal of at least half the intersections when a filter set is split into two sets, thus resulting in low TCAM memory usage. SSA also accesses filters in the TCAM only once per packet, leading to low power consumption. We compare SSA with two best known schemes: MUD [1] and Geometric Intersection-based solutions [2]. Simulation results based on the SNORT filter sets show that SSA uses approximately the same amount of TCAM memory as MUD, but yields a 75% to 95% reduction in power consumption. Compared with Geometric Intersection-based solutions, SSA uses 90% less TCAM memory and power at the cost of one additional TCAM lookup per packet.

1. Introduction

In packet classification, an incoming packet is compared against a set of filters. Most traditional applications only require the highest priority match, e.g., the longest prefix match. However, many new applications demand multi-match packet classification, where all matching filters need to be reported. For example, for accounting purposes, multiple counters may need to be updated for a given packet [1]. Therefore, multi-match classification is necessary to identify the relevant counters for each packet.

Another application of multi-match classification is network intrusion detection systems, which monitor packets in a network and detect malicious intrusions or DoS attacks. Systems like SNORT [3] employ thousands of rules that contain intrusion patterns. Figure 1.a gives an example of a SNORT rule that detects a MS-SQL worm probe. Figure 1.b is a rule for detecting an RPC old password overflow attempt. Each rule has two components: a rule header and a rule option. The rule header is a classification filter that consists of five fixed fields: protocol, source IP, source port, destination IP, and destination port. The rule option specifies intrusion patterns used for scanning packet payloads. Rule headers may have overlaps, so a packet may match multiple rule headers (both examples blow). Multi-match classification is used to find all the rule headers that match a given packet in order to identify the related rule options that must later be checked.

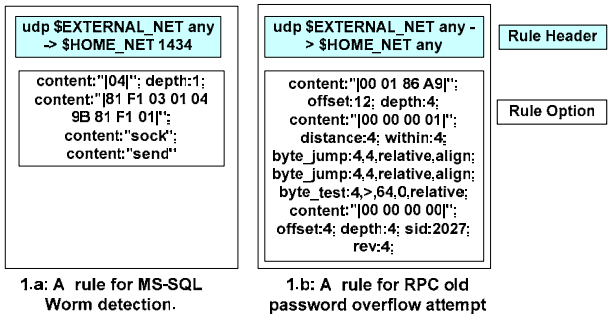


Figure 1. SNORT rule examples.

Multi-match classification is usually the initial step in choosing a set of functions (e.g., update a counter) related to a packet. Because multi-match classification is performed on every packet, we do not want this operation to bottleneck the system. To maintain high packet processing rates, we need an approach that has a *deterministic* and *high lookup rate*. Hence, TCAMs were adopted to solve the multi-match classification problem [1, 2] as they can perform fast parallel searches across all filters in hardware. However, TCAMs are expensive and consume high amounts of power. In some high end routers, TCAMs consume around 30 to 40 percent of the total line card power. As line cards are

stacked together, TCAMs impose a high cost on the cooling system. Figure 2 shows the power consumption of a 9Mbits TCAM based on the data from a TCAM manufacture. The energy used by a TCAM grows linearly with the number of entries searched in parallel and scales with the frequency of TCAM accesses. To be cost and energy efficient, TCAM-based multi-match solutions must *use an economic TCAM memory size* and *perform a limited number of TCAM lookups for each packet.*

None of the previously published multi-match classification schemes can meet all of the requirements above. For example, the MUD scheme proposed by Lakshminarayanan et al. encodes the extra bits in each TCAM entry to support range and multi-match lookup [1]. The amount of TCAM memory needed is linear in the size of the filter sets. However, the algorithm needs at least k TCAM lookups to get k matching results and all the entries in the TCAM are accessed during each lookup. This results in a long processing time and high power consumption for packets that match many filters. Another previously proposed Geometric Intersection-based solution can report classification results with just one TCAM lookup [2]. However, achieving this speed requires that all filter intersections (regions of overlap between filters) be inserted as new filters in the TCAM. Theoretically, N filters with F fields can create $O(N^F)$ intersections. Therefore, this approach is not cost or energy efficient when filters have many intersections.

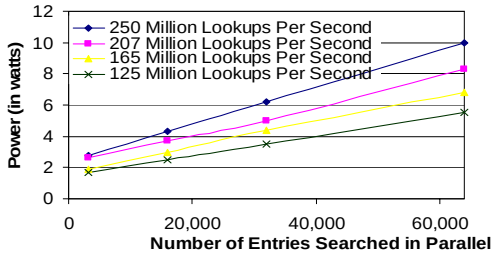


Figure 2. Power consumption for a 9Mbits TCAM.

In this paper, we propose a Set Splitting Algorithm (SSA), which works by splitting the filters into several sets and performing separate TCAM lookups for each set. The benefits of SSA are summarized as follows:

- **Low Memory Usage.** We will show later that it is not necessary to include the intersections caused by filters of different sets in the TCAM. Each time a filter set is split into two sets, SSA guarantees the removal of at least 50% of the intersections needed in TCAMs.
- **Low Power Consumption.** SSA uses a small amount of TCAM memory and accesses each TCAM entry once per packet. Hence, the power consumption level is low.
- **Deterministic Lookup Rates**. If SSA splits filters into k sets, then k TCAM lookups are needed. The number of TCAM lookups is independent of the input packet.
- **Supports Parallelism.** The filter sets generated by SSA are uncorrelated. Thus, the lookups into these filter sets can be parallelized or pipelined.
- **Low Update Cost.** Since filters are split into uncorrelated sets, the update cost is local to one set.

Simulation results based on the SNORT rule sets and synthesized large filter sets show that SSA removes over 95% of the intersections when splitting the filter set into two sets and almost eliminates the need to include extra intersections when splitting the filter sets into four sets. The total number of TCAM entries used is less than 1.2 times the original filter set size, which is 90% less space than the Geometric Intersection-based solutions Compared to MUD, SSA uses a similar amount of TCAM memory. However, SSA is faster than MUD, since each packet requires either two or four TCAM lookups (depending on whether the original filter set is split into two or four sets), while MUD requires 20 TCAM lookups in the worst case. In addition, although SSA involves multiple TCAM lookups, each lookup is performed on different sets. Each entry in the TCAM is accessed only once per packet. Hence, SSA yields a 75% to 95% reduction in power consumption over MUD, as MUD may require accessing each TCAM entry up to 20 times for one packet.

The rest of paper is organized as follows. We start by giving a brief introduction to TCAM in Section 2. Then we analyze the existing solutions for multi-match classification in Section 3. Our SSA scheme is presented in Section 4. We show the effectiveness of our approach by comparing it with two previously published TCAM-based solutions (MUD and Geometric Intersection-based solution), and two representative software-based solutions (EGT-PC[4] and HiCuts [5]) in Section 5. Finally, we state our conclusions in Section 6.

2. Introduction to TCAM

A TCAM consists of a list of fixed-length entries. Each entry has several cells that can be used to store a string. A TCAM works as follows: given an input string, it performs a parallel comparison of the string against all entries contained in memory and reports a bit vector of the matching results. This bit vector is passed to a priority encoder, and the lowest index match result is usually reported (shown in Figure 3). The lookup time (e.g., 4 ns [6]) is deterministic for any input.

Unlike a binary CAM that has two states: (0 or 1), each cell in a TCAM can take one of three states: 0, 1, or '?' (do not care). With the 'do not care' state, TCAMs can be used for matching variable prefix CIDR IP addresses and thus can be used in high-speed IP lookups [7, 8]. TCAMs report a longest prefix match result with just one lookup time (e.g., 4 ns). Furthermore, there is a one-to-one correspondence between the bits in the filter sets and memory bits required.

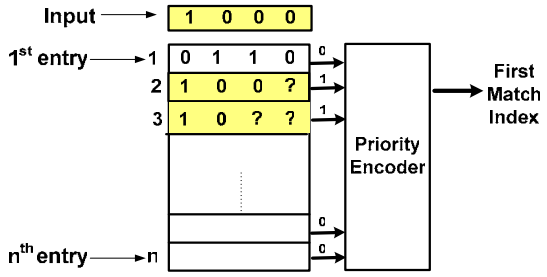


Figure 3. A TCAM.

However, TCAMs have some limitations. TCAMs cost about 30 times more per bit of storage than DDR SRAMs [9]. In addition, TCAMs consume 150 times more power per bit than SRAMs. TCAM power consumption grows linearly with the number of entries searched in parallel and is also directly related to the number of TCAM accesses, as mentioned in Section 1.

3. Related Work

Different approaches have been proposed to save TCAM space and reduce TCAM power consumption. For example, approaches like CoolCAMs [10] and load balancing TCAMs [11, 12] partition the TCAM so that for a given packet, only several partitions are searched to decrease power consumption. These approaches are designed for the one dimensional packet classification (destination IP lookup). Spitznagel et al. [8] proposed a solution for the multi-dimensional case. They organized the TCAM as a two level hierarchy where an index block is used to enable/disable the query process of the main blocks. They also incorporated circuits for direct range comparisons. These approaches are all designed for single match packet classifications that require reporting only the highest priority match. In the rest of this section, we will review current solutions to the multi-match classification problem.

3.1 Bit Vector Solution

Currently, commercial TCAMs only report one matching result (usually the first match). This is because TCAMs have priority encoding circuits that take the matching vector and output the first matching index as previously shown in Figure 3. If we remove that priority encoder, the TCAM can output a bit vector of matching results, one bit for each entry. This solution works very efficiently when each matching result is connected directly to a hardware processing unit [13]. The related follow-up processing can be triggered immediately, and these follow-up processing units can run in parallel.

However, if we don't have the whole system built with the aforementioned hardware, the bit vector solution doesn't work efficiently. In the common packet classification architecture, a processor (CPU or Network Processing Unit (NPU)) is connected to a TCAM. The processor sends packet information to the TCAM, and the TCAM sends back matching results. With the matching results, the processor performs the relevant operations (e.g., send to a port, update a counter) on the packet. If the TCAM returns a matching vector, the processor needs to step through the vector to extract the matching results. This is not an efficient approach when the number of entries N is large and the matching vector is sparse. First, the rate to transfer the N bit vector is limited by memory bandwidth. Second, processing complexity is $O(N)$ to extract the matching results.

Is it possible to change the priority encoder to output the matching results only? This is difficult to accomplish, because the number of matching entries and how they are spread over the N bit vector vary from application to application. It is also hard for TCAM vendors to come up with a general design that works efficiently for all applications.

3.2 Current Industrial Solutions

Some commercial TCAMs support multiple matching. There is a valid bit for each TCAM entry that indicates whether or not to compare this entry with the input. The valid bits of all entries are initially set to valid. Given an input, the first match will be reported in the first cycle. The valid bit for the first matching entry is then unset, and the TCAM will subsequently ignore that entry. The TCAM then performs another lookup, and the second match is reported. This process continues until there are no more matching results. Finally, all the valid bits are reset to valid for the next packet.

As analyzed in [1], identifying k matching results requires $7k$ cycles, as it takes 6 cycles to invalidate and later revalidate an entry. Furthermore, all entries, excluding those that match the packet, are searched k times. Hence, the energy consumption level is high when packets match many entries.

3.3 MUD Solutions

Lakshminaryanan et al. proposed a novel solution to support both range matching and multiple matching in TCAMs [1]. Their approach is based on the observation that some commercially available TCAMs have 144 bits per entry, while the 5-tuple typically used for packet classification has only 104 bits. They proposed a scheme called Multi-match Using Discriminators (MUD). The basic idea is to encode the index of the entry and include the encoded value in each TCAM entry. For example, for the eighth entry in TCAM, put 1000 (binary form of 8) after the filter in that entry. When searching for a match, MUD appends the input packet with a set of discriminators. The packet information is compared with the filters in parallel, while the discriminators are compared with the encoding of the indexes in parallel. The initial discriminators are all set to don't cares, meaning one can match results at any index. After finding a matching result at index j, the TCAM is searched again with a discriminator field value that is

'greater than *j*' to get the second match result. The scheme needs to expand '*greater than j*' to prefixes, so MUD may need multiple TCAM lookups to obtain the second matching result. The authors showed that MUD needs *1+d+(k-2)*(d-1)* TCAM lookups to get *k* matching results, where *d* is the logarithm of the number of entries in TCAM ($d=\log_2 N$). The worst case lookups can be decreased to *1+d*(k-1)*/r with DIRPE, where *r* (smaller than *d*) is a parameter used in DIRPE [1].

Compared to commercial solutions, MUD doesn't need to store the per-search state in the TCAM (i.e., invalidating the previously matched TCAM entries) to get the multi-match results. Therefore, it can be used in multi-threaded environments. However, it shares the same problem with commercial solutions: the number of TCAM accesses needed per packet is linear to the number of matching results. We will show later in Section 6 that a packet can match up to 12 unique filters for the SNORT rule sets and thus requires a maximum of 20 TCAM lookups. This is the worst case performance, but a common packet can result in many TCAM lookups as well. For example, a regular HTTP packet matches at least 4 unique filters. A Napster file-sharing packet can match 8 unique filters and thus requires a maximum of 15 TCAM lookups. In addition, all the TCAM entries are accessed during each TCAM lookup, so the power consumption of MUD is high when packets match many entries.

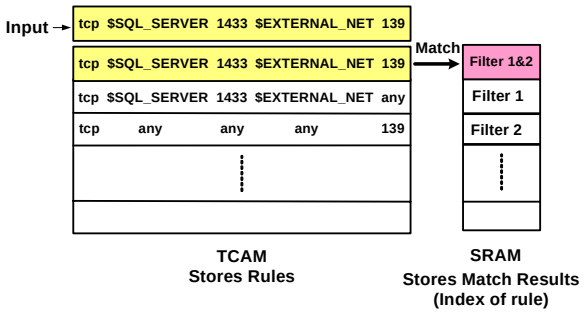


Figure 4. Geometric Intersection-based scheme

3.4 Geometric Intersection-based Solutions

The geometric intersection-based approach inserts filter intersections in the TCAM [2]. An illustrative example is shown in Figure 4. If we only put filter 1 and 2 into the TCAM, a packet matching both filters cannot get all the matching results. Therefore, intersection filters (e.g., the filter in the first entry of the TCAM) are created to handle a packet that matches both filters. These intersection filters and the original filters are inserted into the TCAM and compared in parallel with the input packet. Only one TCAM lookup is needed. Afterwards, the result can be used as an index into SRAM to get all the matching results as shown in Figure 4. Theoretically, the number of intersection filters can be $O(N^F)$, where N is the total number of filters, and F is the number of fields. Real-world filter sets are typically simpler than the theoretical worst case, but we still observe that the SNORT rule header set creates intersections that are ten times the original filter set size [2]. This approach is expensive both in memory and power consumption when the filter set has many intersections.

3.5 Software solutions

Most existing software-based packet classification algorithms are designed for single-match classification. The most relevant work on multi-match classification is on filter conflict detection in [14]. They showed that even for the single-match classification problem, classification filters can intersect and thus introduce conflicts. There are cases where commonly used conflict resolution schemes that are based on filter ordering do not work. The authors proposed a solution to solve the problem by adding new filters in a manner similar to the geometric intersection-based solution.

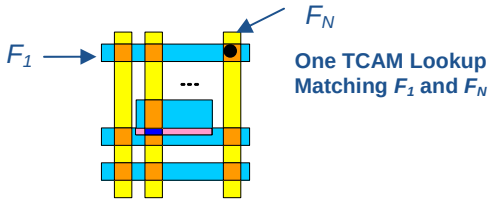
There have been extensive studies on the single-match classification problem, and some of them can be extended to report multi-match results. For example, Grid of Tries and Extended Grid of Tries (EGT) [4, 15] use the source and destination IP addresses to build a trie. By traversing the tree, we can get all the related filters. Multi-match results can be obtained by comparing the input with these filters one by one. Other heuristic algorithms like Recursive Flow Classification (RFC) [16], HiCuts and Hypercuts [5, 17] work well for real world filter sets for single-match classification.

These software solutions are developed based on typical characteristics of single match filter sets. For example, the authors of the EGT scheme observed that in typical single match filter sets, at most 20 rules will match any packet when considering the source and destination fields. Hence, one can search the tree based on the source and destination fields and then perform a linear search among the returned filters. This appears to be an economic solution. However, this observation is no longer valid for multi-match classification sets. In SNORT rule sets, there are many wildcards in the source and destination addresses. One packet can match up to 153 filters when considering source and destination IP addresses only. We will show in Section 6 that two representative algorithms (EGT-PC and HiCut) either need to compare the input packet with many filters one by one, or need a large amount of memory when applied to the SNORT rule sets.

4. A Memory and Power Efficient Solution to Multi-match Classification

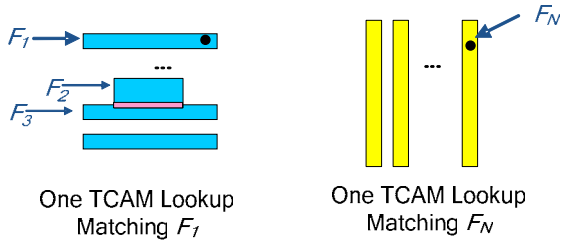
We want to develop a scheme that is both memory and energy efficient. In addition, we want our algorithm to be fast: access TCAMs as few times as possible. The Geometric Intersection-based solution is fast, requiring only one TCAM lookup per packet. However, intersection filters

results in high storage requirements and power consumption. This problem, however, can be solved if we are willing to sacrifice time for storage space and power consumption. Let's look at the example in Figure 5, all the filters intersect with each other in a two dimensional space. Putting the filters into the TCAM with all the intersections requires $O(N^2)$ TCAM entries, and we can obtain the classification result with one TCAM access. However, if we split the filters into two groups, we can check the two groups separately and report the matching results from both groups as shown in Figure 6. This approach can reduce the number of TCAM entries to $N+1$, but it costs two TCAM lookups.



Storage cost: N filters +$O(N^2)$ intersection
Classification speed: 1 TCAM lookup time

Figure 5. Include all intersections in the TCAM.



Storage cost: N filters +1 intersection (F_2 and F_3)
Classification speed: 2 TCAM lookups time

Figure 6. Separate filters into two sets and perform TCAM lookups separately.

We still need to keep intersections that are generated by filters in the same set. For example, in Figure 6, the intersection of filter F_2 and F_3 still needs to be included in the TCAM so that we can distinguish a packet matching both filters from packets matching only one of the filters. However, intersections of filters from different sets are no longer needed in the TCAM because separate TCAM lookups will be performed on each set. Hence, if a packet matches multiple filters from different sets, all the matches will be reported separately.

Note that although we separate filters into two sets here, we don't necessarily require two TCAMs. TCAM vendors now provide a blocking feature that can divide a TCAM into several blocks and allow users to selectively search one or several blocks in parallel. With this feature, different sets of filters can be put into different blocks of the same TCAM and be accessed separately. Since different sets generated by SSA are logically independent, we can even parallelize the lookups in different sets to achieve a higher rate.

If we split the filter sets so that most intersections occur between filters in different sets, we do not need to include these intersections in the TCAM. This saves TCAM space and also reduces power consumption. Note that we only consider the intersections that are different from the original filters because these intersections require extra TCAM entries. For example, in Figure 7, although F_1 and F_2 overlap, the intersection is the same as F_2. We don't need to include this intersection in the TCAM because when a packet matches F_2, we can simply report a matching of both F_2 and F_1. However, when F_2 and F_3 intersect to create intersection I, we do need to consider I because it is different from F_2 and F_3, thus requiring an extra TCAM entry. F_1 doesn't contribute to the generation of this intersection, so I = intersect of F_2 and F_3 only. In other words, I can be eliminated only when F_2 and F_3 are in different sets,

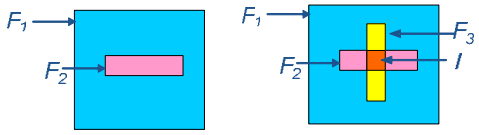


Figure 7. Example of filter intersections.

This approach is memory and power efficient. These benefits come at the cost of more TCAM lookups. Hence, we want to find an algorithm that can automatically separate filters into a minimum number of sets, and keep the total number of filters (original filter set plus the remaining intersection filters) smaller than the TCAM capacity.

4.1 Mathematical Formulation

Suppose there are N filters $F_1, F_2, \ldots, F_N$, and filters create M intersections $I_1, I_2, \ldots, I_M$. For example, I_1= *intersection of* (F_1, F_5, F_6). We want to separate the filters into several sets. We define the residual intersection set I' as intersections generated by filters in the same set, which is different from filters in that set. Our objective is to separate filters into a minimum number of sets that satisfy $N + |I'| <$ TCAM size, which is an NP hard problem.

Suppose we restrict the problem by dividing up the filters into two sets rather than multiple sets. Our new goal then would be to find a way to separate filters into two sets so that number of residual intersection is minimum. Unfortunately, this problem is still NP hard and is known as the *maximum set splitting* or *maximum hypergraph cut* problem [18].

To our knowledge, the best known approximation algorithm for the maximum set splitting problem yields a performance ratio of 0.72 to the optimum solution [19]. However it requires quadratic programming, hence the solution will be too slow to be useful as there are usually hundreds to thousands of filters in the multi-match filter sets [1, 3]. We develop an efficient algorithm SSA to quickly divide filters into multiple sets so as to reduce the need of including intersection filters. SSA guarantees the removal

of at least half of the intersections each time the filter set is split into two sets. In addition, it has a low complexity of *O(NM)*, where *N* is the total number of filters, and *M* is the total number of intersections.

4.2 Johnson's Algorithm for the Maximum Satisfiability Problem

SSA is based on Johnson's algorithm [20], which we briefly summarize in this section. The algorithm solves the maximum satisfiablity problem defined as follows. Let *L* be *N* literal pairs $L=\{\{F_1, \overline{F_1}\}, \{F_2, \overline{F_2}\},.., \{F_N, \overline{F_N}\}\}$. Each literal can take either a true value or a false value. There are *M* clauses, with each clause consisting of a subset of literals either in a positive or negative form, e.g., $C_1 = \{F_1 \vee \overline{F_2} \vee F_6\}$. The goal is to find an assignment of *L* that satisfies the maximum number of clauses. Define *K* as the minimum number of literals in each clause. For $K \geq 2$, this problem is known to be NP complete [18].

```
Assign each clause with |C| literals a weight = 2^-|C|;
While (not all literals assigned weight yet){
    Pick any remaining literal Fi;
    If the total weight of clauses containing Fi > those containing
    ¬Fi {
        Assign Fi a true value;
        Remove all clauses containing Fi ;
        Double the weight of clauses containing ¬Fi ;
    }else{
        Assign Fi a false value;
        Remove all clauses containing ¬Fi ;
        Double the weight of clauses containing Fi
    }
}
```

Figure 8. Johnson's algorithm.

Johnson's algorithm is an approximation algorithm for the maximum satisfiability problem. As presented in Figure 8, it works as follows: At the very beginning, it assigns each clause *C* a weight = $2^{-|C|}$, where $|C|$ denotes the number of literals in *C*. For example, weight of $C_1 = \{F_1 \vee \overline{F_2} \vee F_6\}$ is 2^{-3}. Next, the algorithm examines literals one by one. For any literal F_i that has not been assigned a value yet, if the weight of all clauses containing F_i is higher than the clauses containing $\overline{F_i}$, assign F_i a true value. Now, all the clauses containing F_i are all satisfied and hence can be removed. Clauses containing $\overline{F_i}$ are not satisfied yet, so we want the other literals within the clauses to have a higher probability of being selected as true later. Hence, the algorithm multiplies the weight of all the clauses containing $\overline{F_i}$ by 2. For the case where the weight of all clauses containing $\overline{F_i}$ is higher than the clauses containing F_i, the algorithm performs the opposite actions.

Johnson's algorithm is very simple, it has *O(NM)* complexity. It is proven that Johnson's algorithm can satisfy at least $(2^K-1)/2^K$ fraction of the total clauses [20]. For instance, when $K=2$, it can satisfies at least ¾ of the clauses. Johnson's algorithm achieves the best approximable bound for $K>2$ [20].

4.3 Set Splitting Algorithm (SSA)

In this section, we present SSA, which works by splitting filters that generate intersections into different sets, so that their intersections don't need to be included in TCAMs. Every filter corresponds to a literal in the maximum satisfiablity problem, where literals can either take a true value or false value. For every intersection, two clauses are added into the clause set, one with all positive literals, and one with all negative literals. For example, if I_1 represents the intersection of F_1, F_2 and F_6, add two clauses: $C=\{F_1 \vee F_2 \vee F_6\}$ and $C'=\{\overline{F_1} \vee \overline{F_2} \vee \overline{F_6}\}$. If there are *M* intersections, the total number of clauses is *2M*. Then, we run Johnson's algorithm and assign each filter F_i either a true or a false value. If the literal takes a true value, the corresponding filter is placed in one set. Otherwise, it is put in the other set. Figure 9 shows the pseudo code.

```
Reduce the filter set splitting problem into a max satisfiablity problem:
    Each filter Fi corresponds to a literal
    For each intersection Ii generated by j filters: Fx1, Fx2, …, Fxj, add two clauses:
        C={ Fx1 ∨ Fx2 ∨ ,…, ∨ Fxj)
        C'=( ¬Fx1 ∨ ¬Fx2 ∨ ,…, ∨ ¬Fxj )
Run Johnson's algorithm to assign each filter Fi a true value or false value
    Put Fi in set one if it is true.
    Put Fi in set two if it is negative.
```

Figure 9. Set splitting algorithm (SSA).

Lemma: If both clauses of an intersection are satisfied, this intersection is no longer needed in the TCAM.

Proof: Suppose I_1, which is the intersection of F_1, F_2, and F_6, has both clauses $C=\{F_1 \vee F_2 \vee F_6\}$ and $C'=\{\overline{F_1} \vee \overline{F_2} \vee \overline{F_6}\}$ satisfied. This means that at least one of (F_1, F_2, F_6) is true and one of them is false. According to the algorithm, these filters are split into different sets. Thus this intersection does not need to be represented in the TCAM.□

Theorem: SSA can remove at least 50% of the intersections each time the filter set is split into two sets.

Proof: Each clause is generated by an intersection; hence, it has at least two literals. In other words, $K >= 2$. From Johnson's results, at least $(2^K-1)/2^K = ¾$ of the clauses are satisfied. There are a total of $2M$ clauses, which means $2M*3/4 = 1.5M$ of the clauses are satisfied. Since there are M intersections, and each intersection corresponds to two clauses, at least $0.5M$ of the intersections have both clauses satisfied and hence can be removed according to the lemma. □

If we want to split the filter set further into more subsets, this result still holds. Whenever the filter set is split, we can decrease at least 50% of the intersections. For example, if we split the filters into two sets, and then split both sets again, we can decrease the number of intersections to less than ¼ of the original number.

The complexity of the SSA is same as Johnson's algorithm: $O(NM)$. Note that we didn't impose any restriction on the order for literals to be examined in the Johnson's algorithm. In our simulation, we select literals based on the ratio of positive weight (total weight of clauses containing positive literals) to negative weight. This results in a complexity of $O(NM+N^2)$.

5. Simulation Results

In this section, we compare SSA with two previously proposed TCAM approaches: MUD and Geometric Intersection-based approaches. In addition, we also will present the results of applying software-based solutions (EGT-PC and HiCuts) to the multi-match filter sets. We use the SNORT [3] rule header benchmark as the test sets for multi-match classification. Since the SNORT rule header sets are fairly small, we also test our algorithms on synthesized larger filter sets.

5.1 Evaluation Metrics

We are going to show the tradeoffs between different solutions in terms of the following metrics.

- *Memory consumption*. We use the total number of TCAM entries to reflect memory consumption because all entries have the same width (e.g., 144 bits).
- *Speed*. Memory lookup is usually the bottleneck of a packet classification system. For the three TCAM-based solutions (SSA, MUD and Geometric Intersection-based solution), the number of SRAM accesses needed per packet is equal to the number of TCAM accesses[i]. This is because we can use the TCAM output as an index to fetch the results stored in SRAM, as illustrated in Figure 4 of Section 3. Hence, we report the maximum number of TCAM lookups per packet to reflect the worst case classification rate.
- *Power consumption*. As shown in Figure 2 of Section 1, energy used by a TCAM grows linearly with the number of entries searched in parallel and is directly related to the number of TCAM accesses. Hence, we use the total TCAM entries accessed per packet as a metric for power consumption. It is defined as the product of the number of TCAM entries accessed per lookup and the number of lookups per packet.
- *Update costs*. We randomly select 90% of the filter set as the base filter set and use the remaining 10% as the update filters. We test the insertion cost in terms of the number of newly inserted filters because the deletion of old filters is easier in TCAM-based approaches (just mask those entries out). Note that inserting a filter into a first match TCAM may require moving the existing filters. There are existing approaches that prepare empty spaces in TCAMs, or associate priority [21] or extra circuits with each entry [22]. Here, we only concentrate on the number of newly inserted filters as the metric of update cost.

5.2 Results on the SNORT Rule Header Set

We test all the versions of SNORT after 2.0. Although each rule set has around 1700-2000 rules, many rules share a common rule header (filter). The unique rule headers in each version vary from 240 to 257 as shown in Table 1. Note that we omitted the versions that share the same rule headers with the previous version.

Table 1. SNORT rule headers statistics.

Version	Release Date	Filter Set Size
2.0.0	4/14/2003	240
2.0.1	7/22/2003	255
2.1.0	12/18/2003	257
2.1.1	2/25/2004	263

TCAM Memory Consumption

The Geometrical Intersection-based solution inserts all the intersections into the TCAM. The second column in Table 2 records the number of extra intersections (different from the original filters) that need to be included. Although it is well below the theoretical upper bound $O(N^F)$, it is still roughly 10 times the size of the original filter set. After applying SSA to divide the filters into 2 sets (SSA-2), the number of extra filters that need to be included in TCAM falls below 55, which means that we removed more than 98% of the intersections. When splitting the filters into four sets (SSA-4), SSA almost removes the need to include extra filters caused by intersections. Note that the SNORT rule set contains range and negation (e.g., not port 80). Filters containing range and negation may need to be mapped into multiple TCAM entries. There are many existing solutions to these two problems. For example, we can use extra encodings [1, 23] or hardware circuits [8] to solve the range

[i] SSA and Geometric Intersection-based solution may take longer than one SRAM access time if the matching results are longer than the memory bandwidth. However, it shouldn't be a serious issue as a packet matches a maximum of 12 filters in the SNORT rule sets and pipelining data transfer scheme can be used to decrease the time.

problem and the negation removing scheme proposed in [2] to efficiently map negations into TCAM. In this paper, we assume that range and negation can be efficiently solved using the previous solutions. This assumption won't affect the comparison of the three TCAM-based solutions because all of them face similar percents of ranges and negations.

Table 3 shows the total number of TCAM entries required by the three solutions. MUD uses the least number of TCAM entries: just the number of filters. The Geometric Intersection-based solution needs to include many intersections filters. Hence, it consumes the most number of TCAM entries. SSA dramatically removes the extra intersection filters that need to be inserted into the TCAM. The number of TCAM entries needed is extremely close to the MUD solution.

Table 2. Total number of extra intersections filters in TCAMs.

Version	Geometric Intersection-based	SSA-2		SSA-4	
		Extra Inter-sections	Saving	Extra Inter-sections	Saving
2.0.0	3453	46	98.67%	1	99.97%
2.0.1	3754	47	98.75%	1	99.97%
2.1.0	3758	47	98.75%	0	100%
2.1.1	4067	55	98.65%	0	100%

Table 3. Total number of TCAM entries used.

Version	MUD	Geometric Intersection-based	SSA-2	SSA-4
2.0.0	240	3693	286	241
2.0.1	255	4009	302	256
2.1.0	257	4015	304	257
2.1.1	263	4330	318	263

Classification Speed

For a given packet, only one TCAM lookup is required by the Geometric Intersection-based solution. For our SSA solution, the number of lookups is deterministic. If the filter sets are split into two sets, then two separate TCAM lookups are needed. These two TCAM lookups access different sets of filters. Because no logical relationship exists between them, these two lookup processes can be fully parallelized. If we split the filter sets into four sets, then four TCAM lookups are needed.

The number of TCAM lookups needed by MUD varies from packet to packet. Packets that don't match any filters need only one TCAM lookup. However, for the SNORT filter sets, one packet can match a maximum of 12 filters. Hence, for such packets, MUD takes at least 12 TCAM lookups to get all the matching results. The worst case number of TCAM lookups used by MUD is $1+d*(12-1)/r$, where d is the logarithm of the total number of filters and r is the chunk size. For the SNORT rule sets, d is 8 or 9 and r can be 5 or less. Hence, the number of TCAM lookups can be as high as 20.

The above analysis is the worst case performance of MUD. Since the number of TCAM lookups is workload dependent, we looked at some high frequency packets. A HTTP packet matches at least 4 unique filters and thus requires 5 to 9 TCAM lookups. A Napster file-sharing packet can match 8 unique filters and thus requires 9 to 15 TCAM lookups.

Update Cost

The update cost for the Geometric Intersection-based solution is high, as newly inserted filters may intersect with the existing filters and result in potentially many insertion operations. Table 4 shows the average update costs per newly inserted filter in terms of the number of newly generated filters (including intersection filers) in the TCAM. The third column also shows the maximum number of insertions. One filter can generate up to 157 new TCAM entry insertions. This scheme will obviously be slow during the update process.

Table 4. Update cost in terms of newly inserted filters.

Version	MUD	Geometric Intersection-based		SSA-2		SSA-4	
		Avg	Max	Avg	Max	Avg	Max
2.0.0	1	31.73	157	1.33	17	1.002	2
2.0.1	1	35.24	135	1.34	19	1	1
2.1.0	1	34.71	135	1.36	20	1.002	2
2.1.1	1	36.00	172	1.41	26	1.006	2

The number of newly inserted filters is always 1 for MUD, as it doesn't need to include any intersections. The average insertion cost decreases to almost 1 when using SSA-2. However, the max update cost is still high (around 20). If the filter set is split into 4 sets, the number of newly inserted filters is reduced to approximately 1 and the worst case cost is 2, which is similar to the MUD solution.

Power Consumption

The power consumption of the MUD-based solution is related to the incoming packets. As we explained before, in the worst case, a packet may need 20 TCAM accesses to get all the matching results. For each access, all the entries in TCAM are compared in parallel. Hence, the total number of TCAM entries accessed is 20 times the filter set size in the worst case, which is over 4000 entries as shown in the top curve of Figure 10. If all the packets are HTTP packets, then each packet needs at least 5 TCAM lookups, meaning that it accesses at least 1000 entries. A Napster packet needs at least 9 TCAM lookups, and hence access at least 2000 entries.

The Geometric Intersection-based algorithm only performs one TCAM lookup. Therefore, the number of TCAM entries accessed per packet is the same as the number of TCAM entries used. It is around 4000 due to large number of intersections included in the TCAM. For SSA, although we perform several TCAM lookups, they look into different groups of filters (stored in different TCAM blocks or different TCAMs). Each TCAM entry is

accessed only once per packet. Hence, the total number of TCAM entries accessed is just the number of entries in the TCAMs (less than 300). The energy used by SSA-4 and SSA-2 is similar as shown in the bottom curves in Figure 10. SSA-2 saves at least 90% of the energy consumption compared to the Geometric Intersection-based solution. Compared to MUD, it saves over 95% compared to the worst case performance, and saves 76% for HTTP packets and 87% for Napster file-sharing packets.

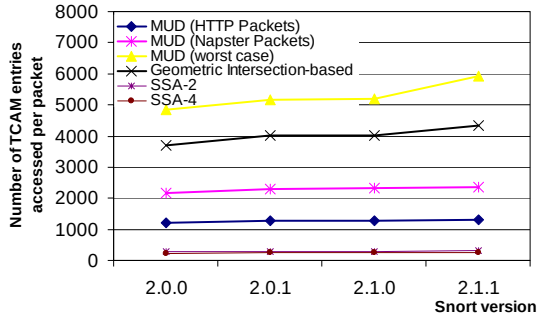


Figure 10. TCAM entries accessed per packet.

5.3 Results on a Synthesized Multi-match Filter Set

Since the SNORT rule header set is fairly small, we generate synthetic multi-match classification test sets to test our algorithms. We take a real-world single-match classification set used in a core router (more than 3000 filters) and randomly insert new filters into it. Each field in these new filters is randomly selected from old filter sets according to their appearance frequencies. Since these new filters are randomly selected, they are likely to intersect with the old filters to create intersections. Because the performance of different algorithms is largely dependent on the number of intersections, we test our algorithms on different intersection rates, defined as the percentage of newly inserted filters to the size of the original filter set. We start at 5%, which constitutes around 150 new filters, and test to 100%, meaning that the number of newly inserted filters reaches the same size as the original filter set size (3060 filters).

Table 5. Total number of extra intersections filters in TCAMs.

Insertion Factor	Geometric Intersection-based	SSA-2		SSA-4	
		Intersections	Saving	Intersections	Saving
0	359	22	93.87%	2	99.44%
0.05	418	20	95.22%	1	99.76%
0.1	488	45	90.78%	3	99.39%
0.2	733	52	92.91%	4	99.45%
0.3	1080	112	89.63%	1	99.91%
0.4	1312	78	94.05%	9	99.31%
0.6	2086	171	91.80%	9	99.57%
0.8	2488	208	91.64%	4	99.84%
1	2883	229	92.06%	7	99.76%

Although the synthesized filter sets are larger than the SNORT rule sets, they generate relatively fewer intersections, as shown in the second column of Table 5. SSA also works well for these large filter sets. Splitting the filter set into 2 sets removes over 90% of the extra intersections, while splitting the set into 4 sets almost eliminates the need to include extra intersections.

Due to space limitations, we will not present detailed results on the classification speed and energy consumption. SSA-2 requires only 2 memory lookups per packet. However, MUD solution can match more than 20 filters in the worst case and thus requires more than 20 TCAM accesses per packet. Hence, SSA-2 is faster and more energy efficient than MUD.

5.4 Software Solutions

We apply the popular EGT-PC and HiCuts algorithm to the SNORT rule sets. EGT-PC can directly support multi-match results. We made slight modifications to the HiCuts algorithm to output all matching results.

EGT-PC is optimized based on the characteristics of the typical single match filter sets, as discussed in Section 3.5. For the SNORT rule sets, many rules apply to the same common source and destination addresses. Therefore, if we search the EGT-PC tree based on the source and destination addresses, the algorithm may return up to 153 filters for an input packet. All these filters have to be compared to the packet one by one in software, making the approach highly inefficient.

When we apply HiCuts to the SNORT rule sets, the high number of filter intersections cause filters to be copied to multiple leaf nodes. We optimized the HiCuts algorithm to report filters directly when they encounter a node that is fully covered by these filters to reduce filter duplications. However, these cases are not common, and we still see that the filters are copied many times at the leaf nodes. Table 6 shows that for the SNORT 2.0.0 filter set, a filter is copied on average 745019/240=3108 times. This high degree of duplication results in a high SRAM storage requirement. More than 40MBytes of SRAM are needed for the SNORT filter sets. This requirement goes beyond the largest single chip SRAM density (around 8MBytes today). In addition, a high degree of filter duplication will cause large update costs, as the insertion of a new filter or deletion of an old filter needs to touch all the leaf nodes containing that filter.

Table 6. Applying HiCuts to the SNORT rule set.

Version	Tree Height	Number of Filters in Leaf Nodes	SRAM Used (KB)
2.0.0	18	745,019	41,000
2.0.1	19	803,645	46,297
2.1.0	19	820,415	47,160
2.1.1	18	827,651	49,378

As we can see from the above results, due to the different characteristics of multi-match and single-match

filter sets, the two single match software-based solutions do not work efficiently on the SNORT rule sets.

6. Conclusions

Multi-match packet classification is a key packet-processing operation needed in important applications such as network intrusion detection and packet-level accounting. Previously proposed TCAM-based approaches suffer from high power consumption or high memory usage. We develop a new set splitting algorithm (SSA) that reduces the TCAM memory and power consumption by 90% when tested on the SNORT rule set. This is accomplished by using a novel scheme that splits filters into multiple TCAM blocks such that, for each split, the number of overlaps within each TCAM block is reduced by a factor of at least two. Our algorithm can support fast updates and also generates deterministic lookup rates. To our knowledge, this is the first power efficient scheme for high-speed multi-match packet classification.

References

[1] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for Advanced Packet Classification with Ternary CAMs," *Proc. ACM Sigcomm*, 2005.

[2] F. Yu and R. H. Katz, "Efficient Multi-Match Packet Classification with TCAM," *Hot Interconnects*, August, 2004.

[3] "SNORT Network Intrusion Detection System." http://www.snort.org.

[4] F. Baboescu, S. Singh, and G. Varghese, "Packet Classification for Core Routers: Is there an alternatives to CAMs?," *Proc. IEEE Infocom*, 2003.

[5] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," *Proc. Hot Interconnects*, August, 1999.

[6] "5512GLQ TCAM from NetLogic Microsystems."

[7] P. Gupta and N. McKeown, "Algorithms for Packet Classification," *Proc. IEEE Network*, 2001.

[8] E. Spitznagel, D. Taylor, and J. Turner, "Classification Using Extended TCAMs," *Proc. IEEE International Conference on Network Protocols (ICNP)*, 2003.

[9] D. E. Taylor, "Survey Taxonomy of Packet Classification Techniques," *Proc. Tech Report, WUCSE-2004-24*, May 2003.

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

[11] R. Panigrahy and S. Sharma, "Reducing TCAM Power Consumption and Increasing Throughput," *Proc. Hot Interconnects*, 2001.

[12] K. Zheng, H. Che, Z. Wang, and B. Liu, "an ultra high throughput and power efficient TCAM based IP lookup engine," *Proc. IEEE Infocom*, 2004.

[13] H. Song and J. W. Lockwood, "Efficient packet classification for network intrusion detection using FPGA," *Proc. International Symposium on Field Programmable Gate Arrays (FPGA)*, Monterey, USA, 2005.

[14] V. Srinivasan and G. Varghese, "Faster IP lookups using controlled prefix expansion," *Proc. ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, 1998.

[15] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and Scalable Layer Four Switching," *Proc. Sigcomm*, Spetember 1998.

[16] P. Gupta and N. McKeown, "Packet classification on multiple fields," *Proc. SIGCOMM*, August 1999.

[17] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet Classification Using Multidimensional Cutting," *Proc. Sigcomm*, August 2003.

[18] P. Crescenzi and V. Kann, "A compendium of NP optimization problems." http://www.nada.kth.se/~viggo/wwwcompendium/node145.html.

[19] G. Andersson and L. Engebretsen, "Better Approximation Algorithms and Tighter Analysis for SET SPLITTING and NOT-ALL-EQUAL SAT," *Proc. technical reports, ECCCTR: Electronic Colloquium on Computational Complexity*, 1998.

[20] D. S. Johnson, "Approximation algorithms for combinatorial problems," *Proc. the fifth annual ACM symposium on Theory of computing*, 1973.

[21] M. Hidell, P. Sjödin, and O. Hagsand, "Router Architectures," *Tutorial at Networking 2004*.

[22] M. Kobayashi, T. Murase, and A. Kuriyama, "A longest prefix match search engine for multi-gigabit ip processing," *Proc. International Conference on Communications (ICC 2000)*.

[23] H. Liu, "conference on Measurement and modeling of computer systems," *Proc. Hot Interconnects*, 2001.