A Smart Pre-Classifier to Reduce Power Consumption of TCAMs for Multi-dimensional Packet Classification

Yadi Ma
University of Wisconsin
Madison, WI, USA
yadi@cs.wisc.edu

Suman Banerjee
University of Wisconsin
Madison, WI, USA
suman@cs.wisc.edu

ABSTRACT

Ternary Content-Addressable Memories (TCAMs) has become the industrial standard for high-throughput packet classification. However, one major drawback of TCAMs is their high power consumption, which is becoming critical with the boom of data centers, the growing classifiers and the deployment of IPv6. In this paper, we propose a practical and efficient solution which introduces a smart pre-classifier to reduce power consumption of TCAMs for multi-dimensional packet classification. We reduce the dimension of the problem through the pre-classifier which pre-classifies a packet on two header fields, source and destination IP addresses. We then return to the high dimension problem where only a small portion of a TCAM is activated and searched for a given packet. The smart pre-classifier is built in a way such that a given packet matches at most one entry in the pre-classifier, which make commodity TCAMs sufficient to implement the pre-classifier. Furthermore, each rule is stored only once in one of the TCAM blocks, which avoids rule replication. The presented solution uses commodity TCAMs, and the proposed algorithms are easy to implement. Our scheme achieves a median power reduction of 91% and an average power reduction of 88% on real and synthetic classifiers respectively.

Categories and Subject Descriptors

C.2.6 [**Computer-Communication Networks**]: Internetworking—*Routers*

Keywords

Packet classification, SmartPC, power consumption

1. INTRODUCTION

Given a set of strictly ordered rules in a classifier, the packet classification problem is to find out the first (highest priority) rule that matches each incoming packet at a router.

Many common packet classification techniques define rules on multiple header fields, e.g., a 5-tuple on source and destination IP addresses, source and destination port numbers and protocol. Packet classification plays a significant role in many networking services and functions that require network traffic (i.e. packets) to be distinguished and isolated into different flows for suitable processing. Examples of such services include: QoS, security, packet filtering (e.g., deny all packets from a known source), policy routing (e.g., route all VoIP traffic over a separate network), traffic shaping (e.g., ensure that no one source overloads the network), and so on.

There are two major trends of previous work on packet classification: RAM-based algorithmic solutions and hardware-based TCAM solutions. The most prevailing RAM-based solutions include decision-tree based schemes such as HiCuts [8], HyperCuts [19], EffiCuts [23], et al. Algorithmic methods do not scale effectively to high performance systems that must process tens of millions of packets per second.

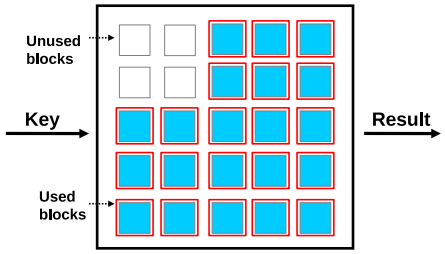
TCAMs allow simultaneous match of an incoming packet against all the stored rules of a classifier within a single memory access and a fixed number of clock cycles. TCAM has become the *de facto* industrial standard for packet classification in high performance routers. However, one major disadvantage of using TCAMs is the high power consumption. The main component of power consumptions of TCAMs is proportional to the number of searched entries [26]. A typical 18Mbit TCAM device can consume up to 15 Watts of power when all the entries are searched. This can be a significant power overhead for the router, switch, or other networking hardware that embeds them. Furthermore, as enterprises and operators continue to seek larger and bigger classifiers, the energy footprint of these devices would continue to grow even further. In this paper, we present a new solution for reducing the energy footprint of TCAMs when applied to the problem of multi-dimensional packet classification by carrying out the classification in two pipelined stages.

1.1 Our pre-classifier approach

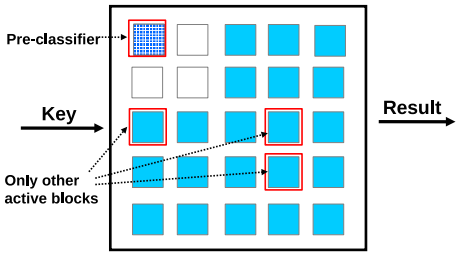
TCAM vendors have been providing mechanisms to address and activate smaller portions of the TCAMs, called blocks. A TCAM block is a contiguous, fixed-sized chunk of TCAM entries, much smaller than the size of the entire TCAM, as shown in Figure 1. If the width of a TCAM block is 80 bits and there are 2K entries in each block, the size of a block is 160K bits. Further suppose the 104 bit header fields of an IPv4 packet (32 bit source or destination IP address, 16 bit source or destination port, plus 8 bit protocol) would occupy two entries in a block. Therefore, each block can hold up to 1K rules. A large classifier of size 50K requires 50 such blocks to store the rules. To reduce power consumption of TCAMs, we take advantage of this mechanism to select and activate a small number of blocks, instead of all the 50 blocks, for each packet classification.

Activating a small number of blocks is only feasible if we can somehow ensure that the incoming packet will be correctly matched by searching the entries stored in these active blocks. We achieve

SIGCOMM'12, August 13–17, 2012, Helsinki, Finland.



(a) TCAM is composed of a number of blocks. For each incoming packet, all the blocks that are used for packet classification are activated.



(b) A TCAM block is used for pre-classifier. For each incoming packet, the pre-classifier is consulted first. Based on its result, only 3 blocks are activated.

Figure 1: An overview of TCAM power saving.

this goal by using a *pre-classifier*. In particular, we construct a specific pre-classifier and shuffle the rules in the original classifier into different blocks so that only the pre-classifier and a small number of TCAM blocks need to be activated to classify a packet. The novelty of our approach lies in finding a good way to form pre-classifier entries so as to minimize the number of blocks activated for each packet classification.

In Figure 1(a), we show an example TCAM composed of 25 blocks. In this example, 21 blocks are used to store the rules in a classifier while 4 remain unused (could be used for other purposes, e.g., IP lookups for packet forwarding). All the 21 blocks that are used to store the original classifier need to be activated for every packet classification. In our modified system, shown in Figure 1(b), one block is used to store pre-classifier entries [1]. The pre-classifier, in this example, requires only three other blocks be activated for correctly matching the incoming packet. This achieves a power reduction of $(21-4)/21 * 100\% = 81\%$. For a 18Mbit TCAM, this approach could bring down its power consumption from 15 Watts to less than 3 Watts.

Our proposed pre-classifier is motivated by previous work [8, 7, 19, 9] which suggest that although large classifiers could contain hundreds of thousands of rules, for *any individual packet* there are usually a small number of "intercepting" rules — rules that match the given packet. (Note that different packets can have completely different intercepting rules and hence it is possible a large fraction of the rules are actually compared to classify a large number of packets.) The power consumption could be reduced if we can somehow shuffle the rules in a classifier into TCAM blocks, such that for each incoming packet, only a subset of blocks that contain all the rules intercepting with the packet are searched.

In this paper we create this pre-classifier by reducing the dimensionality of the problem. Instead of the typical 5-dimensional rules, we construct the pre-classifier by creating rules in two dimensions only: the source and destination IP address ranges. This choice provides an adequate reduction in storage requirements, yet provides adequate ability to classify packets. In fact, the results of the pre-classifier ensure that the packet can then be correctly classified by comparing against only a few other TCAM blocks (which contain suitably shuffled rules). We present a detailed analysis to explain the efficacy of this choice of two dimensions in Section 3.

Finally, this design and use of the pre-classifier is made possible by new features of TCAMs, which support turning on and off a specific subset of blocks. The time and overheads of turning on and off such blocks is negligible. Furthermore, such hardware supports multiple searches in parallel, e.g., Cisco's TCAM4 which allows four parallel searches [3].

The notion of using multiple stages to implement packet classification techniques is not entirely new. Two of the most related examples of this approach for TCAMs are CoolCAMs [26], and packet classification using extended TCAMs [20]. Our work differs from such prior approaches in the specific algorithmic techniques in creating pre-classifiers, and our approach is very deployable with current TCAMs for the problem of packet classification. For instance, CoolCAMs[26] was designed for the problem of IP lookups, in which classification is performed in a single dimension — destination IP address. Their proposed hash-based and trie-based approaches for solving this one dimensional problem do not extend to the multi-dimensional packet classification problem that we are addressing, and required new algorithmic considerations. Similarly, extended TCAMs assumes a new type of ternary match hardware that are not available in current commodity TCAMs. The specifics of our multi-dimensional packet classification problem and the specific restrictions of commodity TCAMs make our solution uniquely applicable to this important problem domain.

1.2 Summary of our contributions

In this paper, we propose a **Smart P**re-**C**lassifier to reduce power consumption of TCAMs for packet classification (SmartPC). With SmartPC, the actual packet classification occurs in two stages: first a given packet is classified by a pre-classifier which provides information on which TCAM block needs to be activated in the next stage; in the second stage, the block from first stage plus a few *general* blocks are activated and searched in parallel for a match for the packet. SmartPC can typically bring down the TCAM's power consumption from 15 Watts to less than 3 Watts. The two-dimensional pre-classifier contains non-overlapping entries, which are formed by combining and expanding rules in the original classifier. The rules in the original classifier are thus reorganized into TCAM blocks so that the rules covered by an entry in the pre-classifier are stored in the same TCAM block. Those rules that do not fit in a block are marked as general rules and stored in general blocks.

We present a detailed design of SmartPC and evaluate it on a large amount of real classifiers from ISPs and synthetic classifiers generated by ClassBench [21]. The experimental results suggest huge power reductions on both real and synthetic classifiers. With block size 128, SmartPC achieves a median of 91% and an average of 88% power reductions on these classifiers with the highest reduction reaching 98%.

2. BACKGROUND

2.1 Problem statement of packet classification

Packet classification is performed using a packet classifier, also called a policy database, flow classifier, or simply a classifier. A

[1]This figure shows one way of implementing a pre-classifier. In practice, a pre-classifier could also be implemented in software, as discussed later in Section 4.

Table 1: A simple example with 8 rules on 5 fields

$Rule$	F_1	F_2	F_3	F_4	F_5	Action
$R0$	000*	111*	10	*	UDP	$action_0$
$R1$	000*	10*	01	10	TCP	$action_1$
$R2$	000*	01*	*	11	TCP	$action_0$
$R3$	0*	1*	*	01	UDP	$action_2$
$R4$	0*	0*	10	*	UDP	$action_1$
$R5$	000*	0*	*	01	UDP	$action_1$
$R6$	*	*	*	*	UDP	$action_3$
$R7$	*	*	*	*	TCP	$action_4$

classifier is a collection of rules or policies. Given a set of strictly ordered rules in a classifier, the packet classification problem is to find out the first rule that matches each incoming packet at a router. Each rule is associated with an action. After classification, the corresponding action will be performed on each packet.

Suppose a classifier at a router contains a set of N rules, and each rule contains K fields. We consider the case where $K = 5$. The five fields are source IP address, destination IP address, source port, destination port and protocol type, respectively (shown as F_1 through F_5 in Table 1). There are three types of matches that a field can have: prefix match (source or destination IP address), exact match (protocol type), or range match (source or destination port).

Table 1 shows a simple example with eight rules on five fields. In a prefix match, the rule field should be a prefix of the header field. Suppose header field 2 of a packet is 1010. In Table 1, it matches the second field, F_2, of rule $R1$. In exact match, the header field of a packet should match the rule field exactly. For example, protocol type could be TCP or UDP. In a range match, the header value should lie in the range specified by the rule. If each of the header fields of a packet P matches each of the corresponding fields in a rule R, the packet P is said to match rule R. If P matches multiple rules, the first rule (with the minimum index) is returned.

2.2 Geometric view of packet classification

We can view a 32-bit prefix like 001* as a range of addresses from $0010\cdots00$ to $0011\cdots11$ on the number line from 0 to 2^{32}. If prefixes can be viewed as line segments geometrically, two-dimensional rules correspond to rectangles, three-dimensional rules to cubes, and so on. A given packet header is a point. The problem of packet classification reduces to finding the highest-priority multi-dimensional region that contains a given point. This is a classic problem in computational geometry, named point location problem. The point location problem is defined as follows: given a partition of K-dimensional space into N disjoint regions, determine the region where a query point lies. Numerous results have been reported in the literature [10, 6], most of which deal with the case of non-overlapping regions or specific arrangement of hyperplanes or hypersurfaces of bounded degree.

2.3 Basics of TCAMs

Ternary Content Addressable Memories (TCAMs) are fully associative memories that allow three matching states, "0", "1" or "X"(wildcard). A "X" state matches both 0s and 1s in the corresponding input bit. This feature makes TCAMs particularly attractive for packet classification and route lookup applications which require longest prefix matches.

The rules are stored in the TCAM array in the order of decreasing priority. Given a packet header to classify, the TCAM performs a comparison against all of its entries in parallel, and a priority encoder selects the first matching rule. TCAMs allow simultaneous match of a packet against a large number of rules within a single memory access, while conventional trie-based designs may require multiple memory accesses for a single packet. Therefore, TCAMs are very popular for designing high-throughput packet classification solutions. However, TCAMs do suffer from some deficiencies such as high power consumption and so on.

3. PROPERTIES OF REAL CLASSIFIERS

A number of previous work have mentioned important properties of real classifier [8, 7, 19, 9]. However, these studies are based on a very small number of classifiers (e.g., in [9], the authors analyzed four ACLs), which may not represent the wide range of real classifiers.

We performed an analysis of more than 200 real classifiers ranging in size from 3 to 15,181. These classifiers were provided to us by a large networking vendor and these are sample classifiers they use for testing their own classification systems. However, the specifics of the classifier are under an NDA and unfortunately cannot be released.

Rule overlapping.

A classifier could contain a large amount of rules, but the number of rules intercepting with a given packet is usually significantly smaller than the theoretical upper bound [8, 7, 19, 9]. Even though only a small number of rules are intercepting with a packet, all the multi-dimensional rules stored in a TCAM need to be searched, which leads to high power consumption. It is reasonable to reduce the dimension for the purpose of power efficiency. Since source and destination addresses are more specific than the other three header fields, we consider these two dimensions as candidates. Therefore we are more interested in what is the maximum number of source and destination IP prefix pairs that need to be searched for any incoming packet.

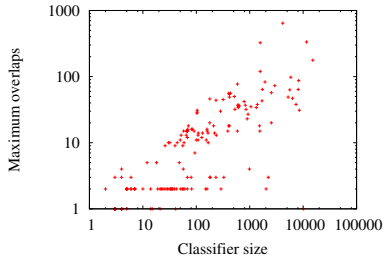
This question is equivalent to the following geometry problem: given a set of axis-aligned rectangles, where each rectangle represents a two-dimensional rule with a range of values in each dimension, what is the maximum number of rectangles that overlap at any point? We employ a line sweep algorithm [22] in which a vertical sweep line moves from left to right. When it crosses the left edge of a rectangle, the rectangle is added to an active set. When it crosses the right edge, the rectangle is removed from the active set. In the inner loop, a horizontal sweep line moves top-down. When it crosses the upper or lower horizontal edge of a rectangle in the active set, we increment or decrement a counter that says how many rectangles overlap at the current point. The maximum value of the counter is the number we are looking for.

We run this line sweep algorithm on the classifiers and the results are shown in Figure 2(a), where x-axis represents classifier size, and y-axis shows the maximum number of overlapping rules in the two-dimensional space (i.e., source and destination IP addresses). Similarly, we plot the number of rules that contain wildcards in the two dimensions as shown in Figure 2(b). The wildcard rules match any packet and intercept with any other rules in the two-dimensional space, therefore, they form a subset of the set of maximum overlapping rules.

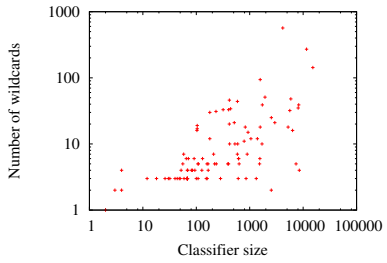
From the plots, we observed that:

- In the two-dimensional space, the maximum number of overlapping rules and the number of wildcard rules of a classifier is an order of magnitude smaller than the classifier size.

Based on these properties, in Section 4, we build a two-dimensional pre-classifier and design an efficient classification algorithm to reduce the power consumption of TCAMs.



(a) Maximum number of overlapping rules in the two-dimensional space



(b) Number of wildcard rules in the two-dimensional space

Figure 2: Properties of real classifiers (both axes are in log scale)

4. DESIGN

To reduce the power consumption of TCAMs, we propose SmartPC, in which we pre-process a classifier and shuffle its rules into blocks in a TCAM, so that only a few TCAM blocks, instead of all the blocks that are used to store the classifier, will be searched to find the desired matching rule for any given packet.

SmartPC reduces the complex multi-dimensional packet classification problem to a much simpler problem by introducing a *pre-classifier* which classifies each incoming packet on two header fields, source and destination IP addresses. Hereafter, we use "rules" to refer to five-dimensional rules in the original classifier, and "entries" to refer to two-dimensional entries in the pre-classifier. The rules are shuffled into TCAM blocks such that each pre-classifier entry covers a number of rules in one of the blocks. Note that when rules are shuffled to TCAM blocks, the relative order of the rules is preserved in each TCAM block, since commodity TCAMs select the match from the first (highest priority) rule.

4.1 Design overview

SmartPC is motivated by our large-scale analysis of real classifiers as shown in Section 3, from which we observe that in a classifier, the maximum number of source-destination pairs matching a given packet is much smaller than the size of the classifier.

The main idea behind our work is that we group source-destination pairs in a classifier into clusters based on their locations in the two-dimensional space. The two-dimensional space is thus divided by non-overlapping rectangles of different sizes. Each rectangle *covers* a cluster of rules and represents an entry in the pre-classifier for the classifier, as discussed in Section 4.2.

Given the ranges of the source and destination IP addresses of a pre-classifier entry p,

$$p = \{[sl_p, sh_p], [dl_p, dh_p]\}$$

and the ranges of the two header fields of a rule r,

$$r = \{[sl_r, sh_r], [dl_r, dh_r]\}$$

we say p covers r if $sl_p \leq sl_r \leq sh_r \leq sh_p$ and $dl_p \leq dl_r \leq dh_r \leq dh_p$, where sl, sh represent lower and upper bound of source address, and dl and dh stand for lower and upper bound of destination address.

We then shuffle the rules in the classifier such that each pre-classifier entry is associated with a TCAM block, named *specific block*, which contains up to a block size of rules that intercept with the entry in the two-dimensional space. If the number of rules that intercept with the pre-classifier entry is larger than one block size, the extra rules are stored in TCAM blocks named *general blocks*. When classifying a packet, the pre-classifier is first consulted. If a match is found in the pre-classifier, then its associated specific block plus the general blocks are activated and searched in parallel for the final result; otherwise, if no match is found in the pre-classifier, only the general blocks are searched.

We propose a classification system for SmartPC as shown in Figure 3. The classification process of a packet occurs in the following steps:

- Each incoming packet is first matched against Index TCAM, which contains pre-classifier entries. The Index TCAM indexes into an associated Index SRAM, which contains the ID of the specific TCAM block. This block will be searched in the second stage lookup.
- In the second stage, only the specific block found in the first stage, if any, together with general blocks are activated and involved in two parallel searches.
- Finally the matches from the specific block and general blocks are resolved by priority, and the action of the higher priority match is returned as the final result.

The above steps can be pipelined to maintain the same operating frequency as commodity TCAMs. It is worth noting that although latency grows slightly due to pipelining, throughput remains unchanged. Throughput is the main metric of interest.

4.2 Construction of a pre-classifier

In SmartPC, we propose the concept of *pre-classifier*, which is motivated by Storm [14], where sampled incoming traffic (i.e., packets) are expanded and combined to form evolving rules which are stored in a rule cache. Those evolving rules are updated over time and are used as popular rules to classify incoming packets. In SmartPC, we adapt the idea of expanding rule, while we expand and combine original rules in a classifier to construct entries in the proposed pre-classifier. Our approach is not dependent on traffic pattern and the process only occurs once for each classifier [2]. Although this insight about expanding and combining rules is derived out of Storm, it is worth noting that SmartPC employs an intelligent pre-classifier to cluster overlapping rules while Storm is focusing on a multi-core approach.

How to build an effective and efficient pre-classifier is of crucial importance. In SmartPC, each entry in a pre-classifier is constructed by expanding and combining the rectangles formed by source and destination fields of the rules in the original classifier. The rules in the classifier are reorganized, with each pre-classifier entry pointing to a TCAM block that contains all the rules covered by the corresponding pre-classifier entry. The number of original

[2]The pre-classifier does not change over time unless the rules in the classifier change.

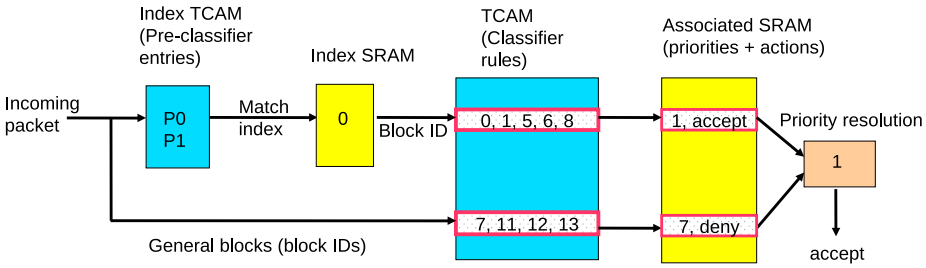


Figure 3: Packet classification system for SmartPC.

rules covered by a pre-classifier entry is restricted to be less than a pre-defined parameter, named *block size* in SmartPC. If the inclusion of a rule causes the number of rules covered by a pre-classifier entry to exceed the block size, the rule will be marked as *general* [3]. All the rules marked as general are stored in general blocks, which will be searched in parallel with the specific block associated with the matching pre-classifier entry, if there is any, for each incoming packet.

Our experimental results show that usually only a small portion of rules in a classifier are marked as general (e.g., the fraction of general rules ranges from 0 to around 20% for all the classifiers we evaluated), therefore the number of general blocks will be much smaller than the total number of TCAM blocks used to store the whole classifier.

Example on how to build a pre-classifier.

Table 2 shows a five-dimensional classifier containing 14 rules. The source and destination addresses are in the format of address/prefix-length. Figure 4(a) is a graphical representation of the source and destination address fields of this classifier, where x-axis and y-axis represent destination address and source address, respectively.

Figure 4 demonstrates the process of building a pre-classifier for the given classifier, assuming a block size of 5 (note that real block size could be much larger, and this small number is for demonstration purposes only). Initially there is no existing pre-classifier entry, thus rule 0 becomes the first pre-classifier entry $P0$ as shown in Figure 4(a). Then we try to expand $P0$ to cover other rules. In Figure 4(b), $P0$ is expanded to cover rule 1. Since rule 5 intercepts with the expanded pre-classifier entry $P0$, we keep expanding $P0$ to cover rule 5 as shown in Figure 4(c). Rule 6 lies inside $P0$. Expansion of P0 to cover rule 7 fails, since the expanded $P0$ would cover more than 5 rules. Therefore 7 is marked as general and will be stored in general blocks. Rule 8 lies inside $P0$, thus so far $P0 = \{0, 1, 5, 6, 8\}$. The size of $P0$ reaches block size. Other intercepting rules 11, 12 and 13 are also marked as general.

We continue to form new pre-classifier entries. Rule 2 forms a new pre-classifier entry $P1$. Similarly, $P1$ is expanded to cover rules 3, 4, 9 and 10. Thus $P1 = \{2, 3, 4, 9, 10\}$ as shown in Figure 4(d).

Heuristic algorithm.

We present the pseudo code on how to build a pre-classifier in Algorithm 1.

The inputs to the algorithm include a classifier C, and an empty pre-classifier P. The properties *isDone* and *isGeneral* of each rule in the classifier are initialized to false, where isDone indicates whether the rule is already covered by a pre-classifier entry, and isGeneral indicates whether it is marked as a general rule. The *size* property of each pre-classifier entry is updated to show the current number of rules covered by the pre-classifier entry. When this number reaches BLOCK-SIZE, which is the pre-defined TCAM block size, the pre-classifier entry can not be expanded any further.

In the outer loop of the algorithm, each rule in the classifier is processed to form or expand a pre-classifier entry. In the inner loop, the existing pre-classifier entries are checked one by one, until a pre-classifier entry is found which can be expanded to include the current rule without exceeding BLOCK-SIZE. If none is found at the end of the inner loop, and the rule is not marked as general, this rule forms a new pre-classifier entry by itself, which may be expanded later.

Algorithm 1 BuildPreClassifier()

```
1:  for each rule i in C do
2:    success = false
3:    if (i.isDone) || (i.isGeneral) then
4:      continue
5:    end if
6:    for each pre-classifier entry j in P do
7:      if (j.size = BLOCK-SIZE) then
8:        continue
9:      end if
10:     expandedEntry = j
11:     if ExpandPreEntry(i, j, expandedEntry) then
12:       success = true
13:       j = expandedEntry
14:       for each rule k in C do
15:         if k is covered by j then
16:           k.isDone = true
17:         end if
18:       end for
19:       break {j is successfully expanded to cover i}
20:     end if
21:     if i.isGeneral then
22:       break
23:     end if
24:   end for
25:   if ((!success) && (!i.isGeneral)) then
26:     a new pre entry k = i{no existing pre entry can be expanded and i does not intercept with any existing pre-entry}
27:     k.size = 1
28:   end if
29: end for
```

Algorithm 1 calls ExpandPreEntry, shown as Algorithm 2, in which a rule is marked as general if it intercepts with the current pre-classifier entry but the inclusion of the rule causes the size of

[3]Though a rule with more wildcards (or shorter prefix lengths on source and destination addresses) is more likely to be marked as general, rules that are marked as general are not necessarily the rules with more wildcards.

Table 2: A classifier which contains 14 rules

Rule#	Src_addr	Dst_addr	Src_port	Dst_port	protocol
0	228.128.0.0/9	0.0.0.0/0	0 : 65535	0 : 65535	0x01
1	223.0.0.0/9	0.0.0.0/0	0 : 65535	0 : 65535	0x06
2	0.0.0.0/1	175.0.0.0/8	0 : 65535	0 : 65535	0x06
3	0.0.0.0/1	225.0.0.0/8	0 : 65535	0 : 65535	0x01
4	0.0.0.0/1	225.0.0.0/8	0 : 65535	0 : 65535	0x06
5	128.0.0.0/1	123.0.0.0/8	0 : 65535	0 : 65535	0x06
6	128.0.0.0/1	37.0.0.0/8	0 : 65535	0 : 65535	0x06
7	0.0.0.0/0	123.0.0.0/8	0 : 65535	0 : 65535	0x06
8	178.0.0.0/7	0.0.0.0/1	0 : 65535	0 : 65535	0x06
9	0.0.0.0/1	172.0.0.0/7	0 : 65535	0 : 65535	0x06
10	0.0.0.0/1	226.0.0.0/7	0 : 65535	0 : 65535	0x06
11	128.0.0.0/1	120.0.0.0/7	0 : 65535	0 : 65535	0x01
12	128.0.0.0/1	120.0.0.0/7	0 : 65535	0 : 65535	0x06
13	128.0.0.0/1	38.0.0.0/7	0 : 65535	0 : 65535	0x06

the entry to exceed BLOCK-SIZE, or it causes entries in the pre-classifier to be overlapping, or it results in conflicts with other rules. This algorithm keeps expanding a pre-classifier entry to cover non-general rules intercepting with it until its size reaches BLOCK-SIZE, or all the intercepting rules in this round are included. This function returns true if the expansion succeeds, and false otherwise.

Algorithm 2 ExpandPreEntry(i, j, expandedEntry)

```
1: tmp = expandedEntry
2: expandedEntry = new entry formed by expanding j to cover i
3: expandedEntry.size += 1
4: if expandedEntry.size > BLOCK-SIZE then
5:     i.isGeneral = true
6:     expandedEntry = tmp
7:     return false {fail to expand}
8: end if
9: if expandedEntry overlaps with other pre entries || RuleConflict(expandedEntry) then
10:    if i intercepts with j then
11:        i.isGeneral = true
12:    end if
13:    expandedEntry = tmp
14:    return false {fail to expand}
15: end if
16: return true {succeed to expand}
```

Algorithm 2 and Algorithm 3 calls each other recursively. Algorithm 3 checks all the non-general rules in classifier C that are intercepting with expandedEntry for conflicts. It returns true if a conflict is found or BLOCK-SIZE is exceeded; otherwise, it returns false.

When Algorithm 1 terminates, a list of pre-classifier entries are formed, and those rules that are not covered by the pre-classifier are marked as general. These general rules are stored in general TCAM blocks. All the rules covered by a pre-classifier entry are stored in the same TCAM block, with their orders in the original classifier preserved. In spite of this, the rules covered by other entries can also be stored in the same TCAM block if they can fit in the block. This is because the rules covered by different entries will not overlap. We only need to separate the blocks that store general rules from other blocks. It is a bin-packing problem to minimize the number of TCAM blocks when organizing the rules covered by pre-classifier entries into TCAM blocks. Simple and efficient heuristics such as best-fit decreasing and first-fit decreasing could be used to solve the problem.

Algorithm 3 RuleConflict(expandedEntry)

```
1: for each rule k in C do
2:     if k.isGeneral || k is already covered || k does not intercept with expandedEntry then
3:         continue
4:     end if
5:     if k lies inside expandedEntry then
6:         expandedEntry.size += 1
7:         if expandedEntry.size > BLOCK-SIZE then
8:             return true
9:         end if
10:    else if k intercepts with expandedEntry then
11:        tmp = expandedEntry
12:        if ExpandPreEntry(k, j, expandedEntry) then
13:            break {succeed}
14:        else
15:            expandedEntry = tmp
16:        end if
17:    end if
18: end for
19: return false {no conflict}
```

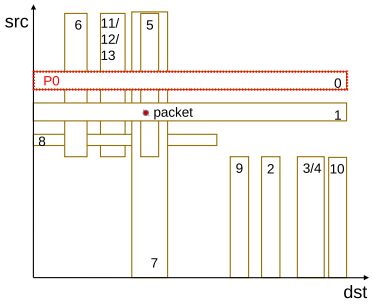
Algorithm analysis.

Our goal is to build a pre-classifier which minimizes the number of general rules. However, an optimal solution is hard to find. Though our proposed heuristic algorithm may not be optimal, it is proved to be effective and efficient, achieving a median power reduction of 91% on real classifiers. Smarter heuristics might exist (and the slack in our experiments is obviously upper bounded by around 9%).

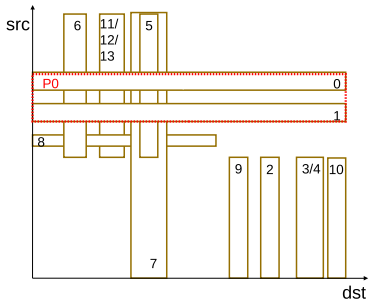
In the worst case, all the rules are overlapping in the 2-dimensional space and all are marked general, resulting in no power reduction. As expected, this is not common in practice since the maximum number of overlapping rules is an order of magnitude smaller than classifier size (see Section 3).

4.3 Packet classification with SmartPC

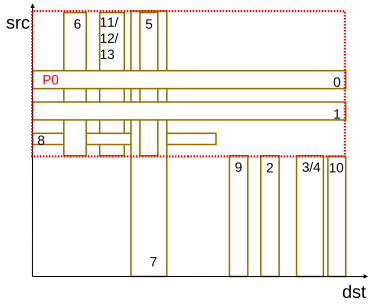
When a packet as shown in Figure 4(a) arrives, the pre-classifier entries $P0$ and $P1$ are consulted first, and a matching entry $P0$ is found. Figure 3 shows the whole process of the classification. The TCAM block pointed to by $P0$, which contains rules (0, 1, 5, 6, 8), together with the general TCAM block are activated and searched in parallel. Finally, the priorities of two matches, rule 1 and 7, are compared, and the action of the higher priority rule 1, "accept", is returned as the final classification decision for the packet.



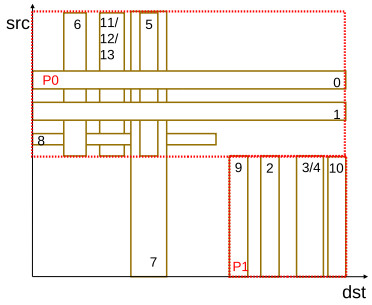
(a) Rule 0 in the classifier forms the first pre-classifier entry P0.



(b) Expand P0 to cover rule 1.



(c) Rule 5 intercepts with P0, expand P0 to cover 5. P0={0,1,5,6,8}.



(d) The second pre-classifier entry P1={2,3,4,9,10}.

Figure 4: An example on how to build pre-classifier (assume block size = 5). Two pre-classifier entries are formed and four rules (7, 11, 12, 13) are marked as general.

4.4 Properties of pre-classifiers

We build an intelligent pre-classifier that satisfies the following properties for the purpose of simplicity and power-efficiency. Note that pre-classifier entries do not necessarily cover the whole two-dimensional space.

Entries in a pre-classifier are non-overlapping. This property guarantees that any incoming packet matches at most one entry in the pre-classifier. If the pre-classifier is implemented in TCAM, there is no need to consider the priorities of pre-classifier entries. While if pre-classifier entries are allowed to overlap, the situation becomes complicated when a given packet matches multiple entries in the pre-classifier, since we need to decide which entry has higher priority. In addition, this property provides more flexibility to the implementation of a pre-classifier as discussed below.

Each rule in a classifier is either covered by only one pre-classifier entry, or marked as general. In other words, each rule is associated with no more than one pre-classifier entry. This property ensures that rules in a classifier are not replicated in TCAM blocks, i.e., each rule is stored only once in one of the blocks. Therefore, our approach is storage-efficient. The only extra storage in SmartPC is the storage used for storing the entries in a pre-classifier.

4.5 Implementation of a pre-classifier

Considering the properties of pre-classifiers, there are a number of approaches to implement a pre-classifier.

For the special case of two-dimensional non-overlapping rectangles (which is the case in our proposed pre-classifier), a number of geometric solutions have been reported with logarithmic time complexity and near-linear space complexity [10, 6]. However, geometric approaches are not efficient enough for pre-classifiers. For the special case of pre-classifiers on just two header fields, Grid of Tries [24] is an efficient algorithmic solution.

TCAM is another alternative where a pre-classifier is stored in TCAM block(s). Our experimental results suggest that a pre-classifier usually contains a small number of entries which can fit in one or a few TCAM blocks. Plus TCAMs are much faster than SRAMs (used in geometric and algorithmic solutions). Though geometric and algorithmic solutions may incur lower power consumption than TCAM-based methods, we decide to store pre-classifiers in TCAMs.

4.6 Rule update

Rule update overhead of SmartPC is generally less than that of regular TCAMs. In SmartPC, the ordering of TCAM entries is kept within one specific block or within a small number of general blocks when rules are added or deleted, while with regular TCAMs, the ordering of TCAM entries has to be kept throughout all the blocks.

To add a rule, the simplest approach is to insert the rule into the general blocks [4]. We can rerun our algorithms and construct from scratch if the number of general blocks exceeds some predefined number. A slightly sophisticated approach could be that if the rule is covered by a pre-classifier entry (i.e., the pre-classifier entry matches the rule), and the specific block associated with this pre-classifier entry is not full, the rule is stored into the specific block; otherwise, the rule can be simply inserted into the general blocks. Depending on rule add/delete frequencies, we may leave some holes in each block (e.g., by under loading the blocks when constructing pre-classifiers) to reduce the overhead of moving rules.

To delete a rule, the block that stores the rule is first found and the rule is deleted. Only when the deletion of the rule changes the ranges of the corresponding pre-classifier entry, the pre-classifier entry is updated.

[4]Please note that the ordering of rules (i.e. rule priorities) must be kept when inserting rules into blocks.

Table 3: Summary of real classifiers

Name	Size	MaxOverlaps	Wildcard
R1	5233	49	18
R2	5626	63	32
R3	5874	98	48
R4	6339	47	16
R5	7356	38	5
R6	8063	64	35
R7	8475	31	4
R8	10054	1	0
R9	11574	334	271
R10	15181	177	143

An update operation can be considered as a delete operation followed by an add operation, as described above.

In summary, only one specific block or a few general blocks plus at most one pre-classifier entry need to be modified for each rule add/delete, while regular TCAMs may have to modify all the blocks.

5. EXPERIMENTAL RESULTS

We start this section with an overview of our experimental methodology. We then evaluate the power reduction of our scheme on real classifiers and synthetic classifiers as well. We also look into the performance of pre-classifiers.

5.1 Experimental methodology

We evaluate the performance of SmartPC using a number of real classifiers and synthetic classifiers generated by ClassBench [21]. As we discussed in Section 3, the real classifiers were provided to us by a large networking vendor and the specifics of the classifier are under an NDA and unfortunately cannot be released. So we are presenting some statistics in Table 3.

Since the power consumption problem is more critical for large classifiers, we only show results from 10 real classifiers with more than 5,000 rules. The largest real classifier we have access to has a size of 15,181. Larger real classifiers are hard to find since ISPs or CAM venders do not release their classifiers. Therefore, we also show results on synthetic classifiers of sizes 50,000 and 100,000. Even larger synthetic classifiers show similar trend and the results are not shown in the paper. Though our focus is on large classifiers, we also evaluated the performance of small real classifiers. SmartPC could achieve significant power reductions on these classifiers as well.

An important parameter is TCAM block size, which is usually dependent on hardware design. In this paper, we show results with different block sizes, to demonstrate how the performance of our scheme is affected by block sizes. This work might aid TCAM designers in choosing the right block size.

Table 3 summarizes the properties of the real classifiers used in the evaluations. In this table, we show the size of each classifier, the maximum number of overlapping rules in two dimensions: source and destination IP addresses, and the number of rules that are wildcard(covers the whole space) on the two dimensions (as discussed in Section 3). These wildcard rules need to be searched for every packet classification and will be marked as general. MaxOverlaps indicates the maximum number of rules that need to be searched for a given packet. This number gives us a rough estimation of block size. As seen here, MaxOverlaps of most classifiers are smaller than 128, and all of them are smaller than 512. We evaluated real classifiers with six different block sizes, 32, 64, 128, 256, 512 and 1024, respectively.

Table 4: Summary of synthetic classifiers

Name	Size	MaxOverlaps	Wildcard
S1	9802	22	4
S2	9416	126	57
S3	9497	76	18
S4	9624	82	12
S5	7255	28	0
S6	99823	27	5
S7	87039	249	79
S8	99836	89	47
S9	99866	81	38
S10	99220	10	0

Similarly, we summarize the properties of ten synthetic classifiers in Table 4. We generate these classifiers using ClassBench by setting rule size parameters to 10k and 100k and using five different classifier seeds. The properties MaxOverlaps and Wildcard are similar to those of real classifiers.

To evaluate our scheme, we use power reduction as the main metric, since the goal of this work is to reduce power consumptions of TCAMs. As mentioned in Section 1, the main component of power consumptions of TCAMs is proportional to the number of searched entries [26]. Therefore, we employ a simple linear power model to estimate the power reductions, though the real reductions may be slightly different. Suppose a classifier contains N rules, and TCAM block size is B. In the default scheme without using SmartPC, all the TCAM blocks that are used to store the classifier, defined as X, must be searched for any packet classification, where $X = \lceil N/B \rceil$.

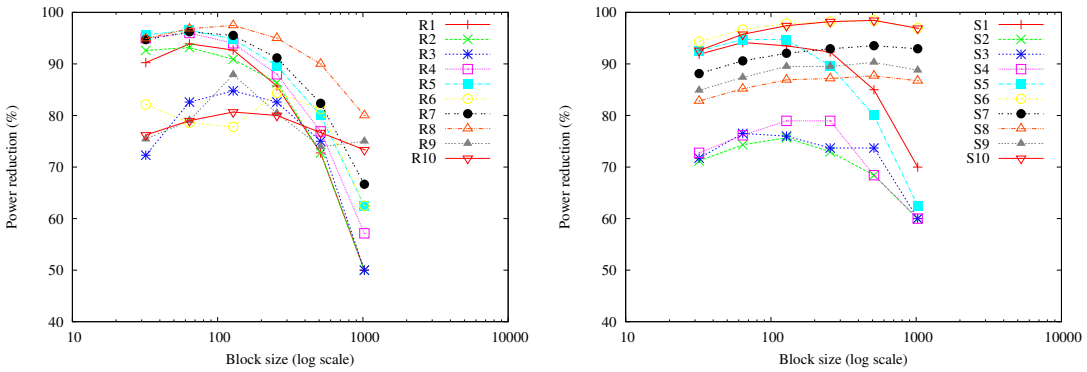
To classify a packet using SmartPC, the pre-classifier, and at most one specific TCAM block plus general TCAM blocks are activated. The power consumption of pre-classifiers depends on how pre-classifiers are implemented. Here we assume pre-classifier entries are stored in TCAM blocks, and we count its power consumption as well. Suppose G out of N rules are marked as general, and P pre-classifier entries are formed. With SmartPC, at most $Y = \lceil P/B \rceil + 1 + \lceil G/B \rceil$ blocks must be activated for each packet classification. Therefore, the percentage of power reduction with SmartPC is $\frac{X-Y}{X} \times 100\%$. We use this definition to evaluate power reductions of SmartPC.

5.2 Power reductions

We evaluate the percentage of power reductions on real and synthetic classifiers by SmartPC. We first evaluate how block sizes affect the power reductions on all the classifiers. We evaluate these classifiers using five different block sizes, 32, 64, 128, 256, 512, and 1024, respectively. We plot the power reductions in Figure 5, where x-axis represents block size and y-axis shows the percentage of power reductions.

As shown in Figure 5(a), regardless of classifier sizes, our scheme achieves huge power reductions on real classifiers, ranging from 78% to 97%. We further observe that the power reduction of each classifier fluctuates with block sizes. For most classifiers, the maximum power reduction occurs at block sizes 64, 128 and 256. With block size 128, the average and median power reductions are 92% and 87%, respectively.

Similarly, we evaluate the percentage of power reductions on synthetic classifiers. As shown in Figure 5(b), SmartPC also achieves huge power reductions on synthetic classifiers. For every classifier and block size combination, the power reduction is more than 60%, with the highest reduction reaching 98%. With block size 128, the average and median power reductions are 91% and 88%, respectively. It is worth noting that power reductions on the five larger



(a) Real classifiers (b) Synthetic classifiers

Figure 5: The percentage of power reductions (note that y-axis starts at 40).

Table 5: Number of general rules and pre-classifier entries for real classifiers with block sizes 64, 128 and 256.

Name	# general rules			# pre entries		
	64	128	256	64	128	256
R1	85	101	159	101	50	25
R2	146	146	146	103	52	28
R3	824	520	302	103	55	29
R4	62	62	62	103	52	27
R5	11	11	11	123	62	31
R6	1513	1462	516	108	53	35
R7	11	11	11	179	88	48
R8	0	0	0	234	120	58
R9	2055	1004	1792	224	131	62
R10	2799	2479	2380	304	169	86

Table 6: Number of general rules and pre-classifier entries for synthetic classifiers with block sizes 64, 128 and 256.

Name	# general rules			# pre entries		
	64	128	256	64	128	256
S1	203	130	91	252	129	64
S2	2075	2030	1903	223	110	54
S3	1971	1931	1877	178	90	42
S4	2028	1776	1530	179	88	45
S5	119	73	67	167	97	48
S6	344	316	290	2855	1561	824
S7	5810	5490	4883	2302	1153	593
S8	11509	11191	11318	3147	1604	861
S9	9401	8699	9198	3091	1573	819
S10	787	558	379	3286	1748	866

classifiers (S6 through S10) are bigger than those on the smaller ones.

By comparing Figure 5(a) and Figure 5(b), we find that the five 10K synthetic classifiers (S1 through S5) present similar trend of power reductions as the real classifiers (R1 through R10). However, the five 100k classifiers (S6 through S10) show bigger power reductions, as they appear in the upper part in Figure 5(b). This suggests that SmartPC achieves more power reductions for larger classifiers. The intuition behind this is that the number of intercepting rules with a given packet and the number of wildcard rules in the two-dimensional space do not increase proportionally with classifier size (as discussed in Section 3). Consequently, the number of pre-classifier entries and general blocks do not scale as well. According to the definition of the percentage of power reduction ($\frac{X-Y}{X} \times 100\%$), in the case of larger classifiers, Y does not increase proportionally with X. Therefore, SmartPC usually achieves more power reductions on larger classifiers.

To further understand the behaviors of our proposed algorithms for building pre-classifiers, we summarize the number of general rules, and the number of pre-classifier entries of real classifiers and synthetic classifiers, as shown in Table 5 and Table 6, for block sizes 64, 128, and 256 respectively.

We make the following observations from these tables:

- With the increase of block size, there is a general trend of non-increasing number of general rules. This is because that with smaller block sizes, rules have bigger chances to be marked as general, while with larger block sizes pre-classifier entries are more likely to be expanded successfully to cover more rules.

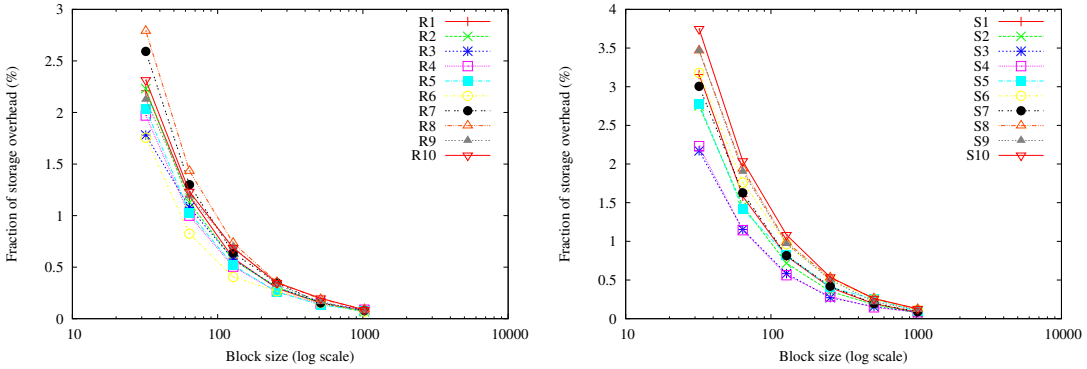
- With the increase of block size, the pre-classifier sizes decrease, since the number of entries in a pre-classifier is roughly proportional to N/B.

- The number of general rules and pre-classifier entries are much smaller than the size of a classifier. Therefore, SmartPC can achieve huge savings in power consumptions since only the pre-classifier, general rules and a block of specific rules will be activated.

5.3 Storage overhead

In SmartPC, we need extra storage to store pre-classifier entries. Pre-classifier entries are two-dimensional on source and destination addresses, so each pre-classifier entry occupies 64 bits, while each five-dimensional rule occupies 104 bits. We show the percentage of pre-classifier size compared to the size of a whole classifier on real and synthetic classifiers in Figure 6. For example, R1 contains 5233 rules. With block size 128, there are 50 pre-classifier entries for R1. Therefore, we calculated the percentage of storage overhead as $\frac{50\times64}{5233\times104} \times 100 = 0.59\%$. Though the actual extra storage depends on TCAM specifications (e.g., width of TCAM, and etc), the numbers shown in Figure 6 provide an estimation of the storage overhead of pre-classifiers. As we can see, the percentage of extra storage decreases as block size increases, since the number of pre-classifier entries is proportional to N/B. We observe that the overhead is pretty small, less than 4% for every classifier. More than 80% of power reductions on the classifiers justify these storage overhead.

SmartPC may introduce holes in blocks, since some blocks may not be full. As discussed in Section 4.2, classifier rules associated with multiple pre-classifier entries can be stored into the same TCAM block as long as the block can fit them. We implemented a



(a) Real classifiers (b) Synthetic classifiers

Figure 6: The storage overhead of SmartPC.

variant of first-fit algorithm to minimize the number of blocks used and found no significant increase of blocks. Considering regular TCAMs also leave holes in blocks to reduce overhead of rule insertions, we only focus on the extra storage incurred by pre-classifiers.

5.4 Comparison of SmartPC with naive-divide

To prove that the huge power reductions come from the intelligence of the proposed pre-classifiers, we compare SmartPC with a naive approach, named naive-divide, which recursively divides the multi-dimensional space into smaller non-overlapping regions. As in the extended TCAMs paper [20], up to a block size number of rules that lie entirely in each region are assigned to the region and these rules are stored in the same TCAM block. Different from the extended TCAMs paper, naive-divide executes one phase, rather than multiple phases, so that there will be only one match in the index TCAM, instead of multiple matches. Naive-divide eliminates the need for multi-match TCAMs. In naive-divide, those rules that do not fit in the blocks are treated as general rules as in SmartPC.

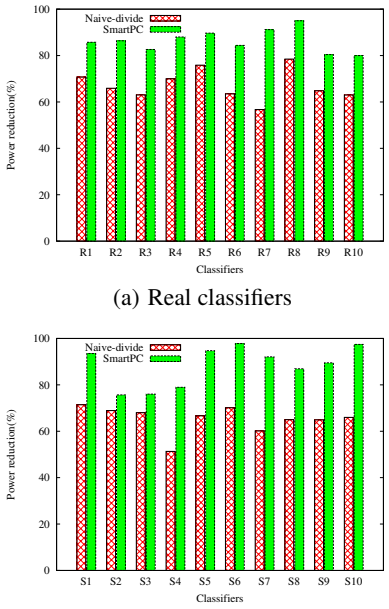
In Figure 7, we compare the power reductions of SmartPC and naive-divide on real and synthetic classifiers. Here we show results with block size 128, while other block sizes show similar performance. SmartPC outperforms naive-divide on every classifier we evaluated. On average, SmartPC achieves 87% power reductions on real classifiers, while naive-divide only achieves 67%. SmartPC outperforms naive-divide by 14% to 34% on real classifiers, with an average of 20%. With synthetic classifiers, SmartPC achieves 88% power reductions on average, while naive-divide achieves 65%. SmartPC outperforms naive-divide by 7% to 32% on synthetic classifiers, with an average of 23%.

The reductions of SmartPC come from the intelligence of the pre-classifiers. In SmartPC, the structures of the rules are taken into account when building the pre-classifier entries. While in naive-divide, the multi-dimensional space can be divided arbitrarily, resulting in a larger amount of general rules. Compared to naive-divide, SmartPC results in smaller number of active TCAM blocks by reducing the number of general rules. Therefore, SmartPC achieves higher power reductions.

5.5 Discussion

5.5.1 *The effect of block size on power reductions*

To understand how block sizes affect power reductions, we plot the number of blocks that need to be activated with default scheme (without using power reduction optimizations) and SmartPC in Figure 8, where x-axis is block size, y-axis represents the number of TCAM blocks that need to be activated, and both axes are in log



(a) Real classifiers

(b) Synthetic classifiers

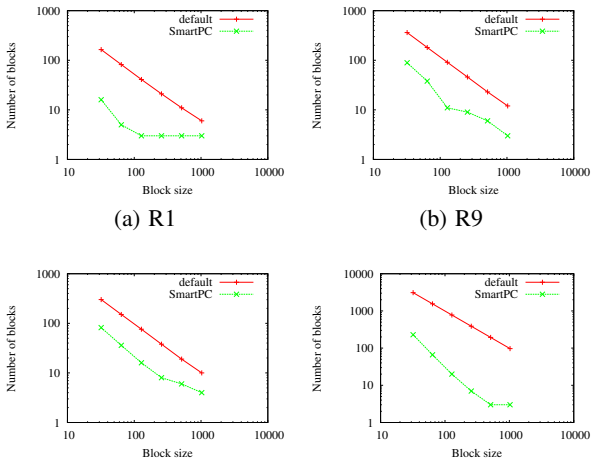
Figure 7: Compare power reductions of naive-divide and SmartPC on real and synthetic classifiers with block sizes 128.

scale. For brevity, we use classifiers R1, R9, S4 and S10 as representatives, while other classifiers show similar trends.

In the default scheme, the number of active blocks decrease linearly with block size because the number of active blocks equals to $\lceil N/B \rceil$, resulting in a straight line for each classifier in Figure 8. While in SmartPC, there is a non-linear relationship between the number of active blocks and block size. Initially, the number of active blocks decreases quickly with block size, till it reaches some point when the decreasing speed reduces. We can see this point occurs at block size 64 for R1 (as shown in 8(a)), 128 for R9 (8(b)), 256 for S4 (8(c)) and 512 for S10 (8(d)). Accordingly, the maximum power reductions are achieved at these points as shown in Figure 5.

Moreover, we discover that the number of active blocks for R1 reaches a constant value of 3 when the block size exceeds 128. 3 blocks is the minimum number of blocks that have to be searched in our scheme [5], which include a block storing the pre-classifier, a general block and a specific block. Similarly, we also observe this

[5]This is generally true, while it is possible that the number of gen-



(a) R1 (b) R9

(c) S4 (d) S10

Figure 8: Number of blocks activated with default scheme and SmartPC (both axes are in log scale)

behavior with other classifiers. For example, the number of active blocks for S10 also reaches 3 when the block size is greater than 512.

5.5.2 *The effect of prefix distribution and prefix length*

Using ClassBench [21] tool, we generate classifiers with different prefix distributions by adjusting the *smoothness* parameter which may take a value from 0 to 64. A value 0 maintains the distributions specified in a parameter file to the rule generator, while a value of 64 models a uniform distribution. We observe that with the increase of smoothness from 0 to 64, both the number of general rules and the size of the pre-classifier decrease, resulting in larger power reductions. We set smoothness to 2 when generating the ten synthetic classifiers to evaluate the performance of SmartPC in a relatively rigorous environment.

We also generate classifiers with more or less specific address prefixes by varying *address scope*, a parameter in range of -1.0 to 1.0 which adjusts the average scope of the address prefixes. With less specific address prefixes (shorter prefix lengths), the number of rules marked as general increases, while the size of the associated pre-classifier decreases. Though the number of general rules and the size of pre-classifier fluctuate with the parameter, there is no obvious change in the percentage of power reductions in SmartPC.

5.5.3 *Power reductions on small classifiers and IPV6 classifiers*

Though the focus of our work is on large classifiers, small classifiers can also benefit from SmartPC. From above analysis, our scheme could provide power reductions on a classifier which occupies more than three TCAM blocks, since generally three blocks is the minimum number of blocks that are activated in our scheme.

Although we do not have access to IPV6 classifiers, the algorithms presented in Section 4 apply to IPV6 classifiers without any modifications. We expect similar performance on IPV6 classifiers.

6. RELATED WORK

There are two main threads of research on packet classification: RAM-based algorithmic approaches and TCAM-based approaches. A lot of intelligent algorithmic solutions are proposed and many of them are based on decision trees, e.g., HiCuts [8], HyperCuts [19], HyperSplit [18], Modular packet classification [25], Common Branches [4], and EffiCuts [23]. The core issue of algorithmic approaches centers on the tradeoff between memory usage and speed.

Wire speed packet classification motivated the development of hardware-based solutions. Ternary Content Addressable Memories (TCAMs) has become the *de facto* industrial standard for packet classification in high performance routers. However, TCAMs suffer from several primary deficiencies: high power consumption, high cost per bit relative to other memory technologies, and storage inefficiency. Accordingly, prior work in optimizing TCAM-based systems fall into four broad categories: power efficiency [26, 20], circuit modification [20, 11], classifier compression [5, 15] and range reencoding [17]. In this paper, we focus on power-efficient TCAM solutions, while reencoding or compression or reducing the number of entries is an orthogonal problem, and our solution can be combined with these types of work.

CoolCAMs [26] divides a TCAM device into multiple partitions. An IP lookup becomes a two stage process where only one partition is selected in the first stage and the partition is queried in the second stage. However, CoolCAMs is limited to the problem of IP forwarding where the destination address of an incoming packet is matched against the longest matching prefix in a routing table.

To apply TCAMs to the more difficult packet classification problem where incoming packets are matched against multi-dimensional rules in TCAMs, extended TCAMs [20] extends the partitioned TCAM concept in [26]. A filter set is partitioned into multiple blocks, each of which is associated with an appropriate index filter. Upon a query, all matching index filters are first identified and then the blocks associated with those matching index filters are queried. However, packets follow up on multiple matches in the index TCAM, which makes this approach infeasible in two ways. First, if legacy chips are used, the bitmap with the blocks can not be transferred to activate on each search. Second, it requires a new type of ternary match hardware that returns all matches because commodity TCAMs give only the first result.

Except [26] and [20], there is significant recent work in the space of power efficiency that we are aware of, but they are within companies and are of proprietary nature. We are under NDA with one company and can not refer to them.

Classifier compression optimizations convert a given classifier into another semantically equivalent classifier that requires fewer TCAM entries. While these techniques would also reduce power consumptions, SmartPC is complementary to compression techniques, and if combined, power consumptions could be further reduced. In [5], by expanding, trimming, adding and merging rules, the authors identify semantically equivalent classifiers that lead to fewer TCAM entries. The redundancy removal algorithm in [12] can reduce TCAM usage by eliminating all the redundant rules in a packet classifier. To address the prefix expansion problem of TCAMs, TCAM Razor [15] proposed a greedy algorithm that finds locally minimal solutions along each dimension and combines these solutions into a smaller equivalent packet classifier. In Bit Weaving [16], the authors proposed the first algorithm that can compress a given classifier into a non-prefix ternary classifier.

Range reencoding schemes cope with range expansion by developing a new representation for important packets and intervals. Previous range reencoding schemes fall into two categories: database independent encoding schemes [1, 11], where each rule is encoded according to standard encoding scheme, and database dependent encoding schemes [2, 17, 13], where the encoding of each rule depends on the intervals present within the classifier. While range

eral rules is zero for some classifiers, therefore bring down this number to 2.

reencoding schemes mitigate the effects of prefix expansion, they require either extra hardware or more per packet processing time.

In [9], the authors explored the structure and properties of four ACLs, and provided a guideline for designing classification algorithm which states that a multi-dimensional classification problem should be split into two stages. Though we also employ a two-stage process, there are major differences. The focus of [9] is on properties of ACLs and the authors did not design any specific packet classification algorithm. Furthermore, we employ a pre-classifier to pre-classify on source and destination addresses, and then a full classifier on all the five fields. While in [9], the first stage is classification on source-destination pairs and the second is on other fields. The results from the two stages are merged to get the final result.

7. CONCLUSION

In this paper, we performed a large scale analysis of important properties of more than 200 real classifiers. Based on our analysis, we propose SmartPC, a smart pre-classifier for power-efficient packet classification using TCAMs. In SmartPC, the rules in a classifier are shuffled into TCAM blocks such that each pre-classifier entry is associated with a TCAM block. To classify a packet in SmartPC, the two-dimensional pre-classifier is first consulted which directs the search to at most one specific TCAM block that contains rules intercepting with the packet. Then the specific block plus a few general blocks are searched in parallel and two matches are generated. Finally the action from the higher priority match is returned as the final result. SmartPC uses commodity TCAMs, and the algorithms for building pre-classifiers are easy to implement. We evaluated SmartPC with real and synthetic classifiers. SmartPC achieves more than 80% power reductions on most classifiers with less than 4% storage overhead. With block size 128, SmartPC achieves a median power reduction of 91% and an average power reduction of 88% on these classifiers. To demonstrate the effectiveness of pre-classifiers, we compared SmartPC with naive-divide, a naive approach that recursively divides the multi-dimensional space into smaller regions. SmartPC outperforms naive-divide for every classifier, with 20% more power reductions on average. SmartPC is a practical and promising solution to address the high power consumption of TCAMs and can find its applications in data centers.

8. ACKNOWLEDGEMENTS

All authors are supported in part by the following grants of the US NSF: CNS-1040648, CNS-0916955, CNS-0855201, CNS-0747177, CNS-1064944, and CNS-1059306.

9. REFERENCES

[1] A. Bremler-barr and D. Hendler. Space-efficient TCAM-based classification using gray coding. In *IEEE INFOCOM*, 2007.

[2] H. Che, Z. Wang, K. Zheng, and B. Liu. DRES: Dynamic range encoding scheme for tcam coprocessors. *IEEE Transactions on Computers*, 57:902–915, 2008.

[3] Cisco. Cisco catalyst 4500 series supervisor engine 6-e centerflex technology. http://www.cisco.com/en/US/prod/collateral/switches ps5718/ps4324/prod_white_paper0900aecd806dc821.html.

[4] E. Cohen and C. Lund. Packet classification in large ISPs: Design and evaluation of decision tree classifiers. In *ACM SIGMETRICS*, 2005.

[5] Q. Dong, S. Banerjee, J. Wang, D. Agrawal, and A. Shukla. Packet classifiers in ternary CAMs can be smaller. In *ACM SIGMETRICS*, 2006.

[6] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15:317–340, May 1986.

[7] P. Gupta and N. Mckeown. Packet classification on multiple fields. In *ACM SIGCOMM*, 1999.

[8] P. Gupta and N. Mckeown. Packet classification using hierarchical intelligent cuttings. In *Hot Interconnects VII*, 1999.

[9] M. E. Kounavis, A. Kumar, H. Vin, R. Yavatkar, and A. T. Campbell. Directions in packet classification for network processors. In *HPCA-9*, 2003.

[10] T. V. Lakshman and D. Stiliadis. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. In *ACM SIGCOMM*, 1998.

[11] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary. Algorithms for advanced packet classification with ternary CAMs. In *ACM SIGCOMM*, 2005.

[12] A. X. Liu, C. R. Meiners, and Y. Zhou. All-match based complete redundancy removal for packet classifiers in TCAMs. In *IEEE INFOCOM*, 2008.

[13] H. Liu. Efficient mapping of range classifier into Ternary-CAM. In *HOT Interconnects*, 2002.

[14] Y. Ma, S. Banerjee, S. Lu, and C. Estan. Leveraging parallelism for multi-dimensional packet classification on software routers. In *ACM SIGMETRICS*, 2010.

[15] C. Meiners, A. Liu, and E. Torng. TCAM Razor: A systematic approach towards minimizing packet classifiers in TCAMs. In *ICNP*, 2007.

[16] C. R. Meiners, A. X. Liu, and E. Torng. Bit weaving: A non-prefix approach to compressing packet classifiers in TCAMs. In *ICNP*, 2009.

[17] C. R. Meiners, A. X. Liu, and E. Torng. Topological transformation approaches to optimizing TCAM-based packet classification systems. In *ACM SIGMETRICS*, 2009.

[18] Y. Qi, L. Xu, B. Yang, Y. Xue, and J. Li. Packet classification algorithms: From theory to practice. In *IEEE INFOCOM*, 2009.

[19] S. Singh, F. Baboescu, G. Varghese, and J. Wang. Packet classification using multidimensional cutting. In *ACM SIGCOMM*, 2003.

[20] E. Spitznagel, D. Taylor, and J. Turner. Packet classification using extended TCAMs. In *ICNP*, 2003.

[21] D. Taylor and J. Turner. ClassBench: A packet classification benchmark. http://www.arl.wustl.edu/∼ det3/ClassBench/index.htm.

[22] TopCoder. Line sweep algorithms. http://community.topcoder.com/tc?module=Static&d1= tutorials&d2=lineSweep.

[23] B. Vamanan, G. Voskuilen, and T. N. Vijaykumar. EffiCuts: optimizing packet classification for memory and throughput. In *ACM SIGCOMM*, 2010.

[24] G. Varghese. *Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices*. Morgan Kaufmann, 2005.

[25] T. Y. Woo. A modular approach to packet classification: algorithms and results. In *IEEE INFOCOM*, 2000.

[26] F. Zane, G. Narlikar, and A. Basu. CoolCAMs: Power-efficient TCAMs for forwarding engines. In *IEEE INFOCOM*, 2003.