Packet Classifiers In Ternary CAMs Can Be Smaller

Qunfeng Dong
University of Wisconsin
Madison, WI 53706
qunfeng@cs.wisc.edu

Suman Banerjee
University of Wisconsin
Madison, WI 53706
suman@cs.wisc.edu

Jia Wang
AT&T Labs - Research
Florham Park, NJ 07932
jiawang@research.att.com

Dheeraj Agrawal
University of Wisconsin
Madison, WI 53706
dheeraj@cs.wisc.edu

Ashutosh Shukla
University of Wisconsin
Madison, WI 53706
shukla@cs.wisc.edu

ABSTRACT

Serving as the core component in many packet forwarding, differentiating and filtering schemes, packet classification continues to grow its importance in today's IP networks. Currently, most vendors use Ternary CAMs (TCAMs) for packet classification. TCAMs usually use brute-force parallel hardware to simultaneously check for all rules. One of the fundamental problems of TCAMs is that TCAMs suffer from range specifications because rules with range specifications need to be translated into multiple TCAM entries. Hence, the cost of packet classification will increase substantially as the number of TCAM entries grows. As a result, network operators hesitate to configure packet classifiers using range specifications. In this paper, we optimize packet classifier configurations by identifying semantically equivalent rule sets that lead to reduced number of TCAM entries when represented in hardware. In particular, we develop a number of effective techniques, which include: trimming rules, expanding rules, merging rules, and adding rules. Compared with previously proposed techniques which typically require modifications to the packet processor hardware, our scheme does not require any hardware modification, which is highly preferred by ISPs. Moreover, our scheme is complementary to previous techniques in that those techniques can be applied on the rule sets optimized by our scheme. We evaluate the effectiveness and potential of the proposed techniques using extensive experiments based on both real packet classifiers managed by a large tier-1 ISP and synthetic data generated randomly. We observe significant reduction on the number of TCAM entries that are needed to represent the optimized packet classifier configurations.

Categories and Subject Descriptors

C.2.5 [**Computer Communication Networks**]: Local and Wide-Area Networks—*Internet*; C.2.6 [**Computer Communication Networks**]: Internetworking—*Routers*

General Terms

Algorithms, Design, Performance

Keywords

Packet Classification, Semantic Equivalence, Ternary CAM

SIGMetrics/Performance'06, June 26–30, 2006, Saint Malo, France.

1. INTRODUCTION

The problem of packet classification can be stated as follows: match an incoming packet against a *packet classifier*, which is a set of *rules* defined over a number of packet header *fields*. Packet classification is one of the most basic operations performed in IP networks. A simple example of packet classification is the packet forwarding operation in an IP router by which each incoming packet is classified based on its destination IP address field onto one of the forwarding table entries that specifies the outgoing link. Similarly, Access Control Lists (ACLs) in routers are classifiers in which each rule usually specifies five fields including source IP prefix, destination IP prefix, source port number (or range), destination port number (or range), as well as protocol type. Each field can also be specified as a wildcard, which means "do not care" and hence matches any value. Besides the fields used for matching, each rule also specifies a *decision* (or *action*) to be carried out. For example, the decision may be `deny` if packets that match the rule are classified as unwanted traffic. Some example rules are given in Table 1.

1.1 Packet classification

Each rule in a packet classifier typically specifies a conjunction of multiple clauses. In general, there is one clause for every field in the rule. Three types of match may be performed on the specified fields. An *exact match* returns `true` if and only if the packet field value and the rule field value being compared are the same and contain no wildcards, e.g., the protocol field clauses in Rules 1-3 of Table 1. A *prefix match* returns `true` if and only if the packet field value coincides with the rule field value in all non-wildcard bits, e.g., the Destination IP address field clause of Rule 1, and the Source IP address field clause of Rules 2 and 3. A *range match* returns `true` if and only if the packet field value

Rule #	Source IP	Destination IP	Source Port	Destination Port	Protocol	Action
1	*	10.112.*.*	5000-65535	*	UDP	deny
2	32.75.226.*	*	*	1001-2000	ICMP	deny
3	199.36.184.*	*	49152-65535	*	ICMP	deny
4	*	*	*	*	*	permit

Table 1: Example rules in a packet classifier.

is contained in the range specified by the rule field clause, e.g., the source port field clause in Rules 1 and 3, and the destination port field clause in Rule 2. When matching a packet against a rule, the specified fields of the packet are compared against their counterparts in the rule. If and only if every field match returns `true`, the rule is considered to *match* the packet, and vice versa.

In general, there are two representative models of packet classification. The most commonly seen model is the *first-match* model, where each rule is assigned a distinct priority and the rule set is typically sorted in decreasing order of priority. The objective of packet classification in this model is to find for each incoming packet the first (i.e., highest priority) rule in the rule set that matches the packet, and carry out the action specified by that rule. The other model is the *multi-match* model [26], where rules do not have priorities. The objective of packet classification in this model is to find for each incoming packet all the rules that match the packet, and carry out the action of each matching rule. As we will later see, it is this lack of an ordering of rules that disables multi-match solutions to be applicable to the first-match model. In this paper, we focus on solutions for the first-match model. By "packet classification", we refer to packet classification in the first-match model.

Beyond ACLs, any network service that requires differentiation of packets into one or more classes requires an efficient implementation of packet classification. Examples of such services, which include network address translation (NAT), virtual private networks (VPNs), and other quality of service mechanisms such as metering, traffic shaping, monitoring, policing, etc., are continuously growing in importance in today's networks.

1.2 State-of-the-art

The challenging aspect of packet classification is its need to be implemented at line speeds. Assuming 40-byte packets, if each incoming packet needs to be classified at OC-768 speeds, the total time budget available to a packet is 8 nanoseconds. This is expected to be further exacerbated as the number of rules in classifiers increases. For example, it has been envisioned that greater deployment of differentiated services will lead to packet classifiers that contain a few hundred thousand rules in edge routers [15].

A long thread of research has defined various algorithmic techniques for fast and efficient software-based packet classification [11, 20, 7, 21, 8, 25, 18, 2, 1, 24, 19, 10, 23]. Unfortunately, packet classification as a general problem is inherently hard. Overmars and van der Stappen [17] have shown that for packet classification over $d > 3$ fields, which is very common in real applications, the best known algorithms have either $O(\log n)$ search time at the cost of $O(n^d)$ memory space or $O(\log^{d-1} n)$ search time at the cost of $O(n)$ memory space, where n is the number of rules in the packet classifier. Therefore, in many scenarios, such software-based solutions prove to be fairly slow with respect to the available time budget (such as those pioneering algorithms [11, 20]) or consume exorbitant memory space (such as [7, 8]). Moreover, they unanimously exploit various statistic characteristics of real packet classifiers to improve speed and memory requirement, and hence are not considered **general**.

Given some of these limitations of software-based solutions, many router vendors favor hardware solutions based on *Ternary Content Addressable Memory (TCAM)* [16, 9, 4] because of its fast and stable lookup speed. Unlike software solutions, TCAMs work for all rule sets, not just "typical rule sets". Basically, TCAMs compare a given search key with all TCAM entries in parallel and is thus able to return the first matching entry in one single clock. The focus of this paper is on such TCAM-based packet classification systems.

1.3 Design challenges for TCAM-based systems

While TCAM-based solutions are fast, the following are four important issues that need to be tackled when designing such systems.

- *Inefficient representation of ranges:* An exact match or a prefix match clause can be efficiently represented by TCAMs using a single entry. However, a single range match clause may need to be partitioned into smaller ranges that can be represented in the form of prefixes, often requiring many TCAM entries. For example, rules often use the port range 1024–65535, which needs to be represented using the following 6 prefixes.

000001xxxxxxxxxx : 1,024 – 2,047
00001xxxxxxxxxxx : 2,048 – 4,095
0001xxxxxxxxxxxx : 4,096 – 8,191
001xxxxxxxxxxxxx : 8,192 – 16,383
01xxxxxxxxxxxxxx : 16,384 – 32,767
1xxxxxxxxxxxxxxx : 32,768 – 65,535

In general, an m-bit range may take up to $2(m-1)$ prefixes to represent (e.g. the range $[1, 2^m - 2]$). If the 16-bit source port range and destination port range are both specified, it may take as many as $30 \times 30 = 900$ TCAM entries to represent a single rule! Based on an analysis of 12 real packet classifiers, Taylor [22] reported that the number of TCAM entries needed can be as large as 6.2 times their number of rules. As a result, network operators hesitate to configure packet classifiers using range specifications.

- *Hardware cost:* TCAM costs have been a significant part of the cost of line cards or routers. Therefore, in order to reduce hardware costs of line cards and routers, an important design requirement of TCAM-based packet classifiers is to express the rule set using as few TCAM entries as might be possible.

- *Power consumption:* TCAMs can account for a considerable portion of the power consumption of a router line

card. Technically, power consumption is a key problem in large core routers. Economically, power supply and cooling costs account for a major part of an ISP's operational expenses [3]. Efficient representation of classifiers using as few TCAM entries as possible leads to greater power efficiency.

- *Area constraints:* For many routers, board area is a critical issue. TCAMs occupy much more physical space than SRAMs (which can be used for software-based packet classification systems). Hence, it is again important to fit a packet classifier into as small-sized a TCAM as possible.

1.4 Proposed approach and key contributions

Based on these observations, the goal of our work is to design TCAM-based packet classification systems that attempt to reduce the number of TCAM entries used in representing packet classifiers. We achieve this by using *semantically-equivalent* packet classifiers — given a packet classifier, we define a new packet classifier that takes the same action on any incoming packet as the original packet classifier but requires fewer TCAM entries to represent. This semantically equivalent packet classifier is constructed through an algorithmic sequence of operations which include *trimming* rules, *expanding* rules, *merging* rules, and sometimes even *adding* rules to meet the reduction objectives. Compared with previously proposed techniques (e.g. [14, 24, 12]) which typically require modifications to the packet processor hardware, our scheme does not require any hardware modification, which is highly preferred by ISPs. Moreover, our scheme is complementary to previous techniques in that those techniques can be applied on the rule sets optimized by our scheme.

The following, therefore, are the key contributions of this work.

- We propose a set of practical techniques for defining semantically equivalent packet classifiers that are smaller in TCAM size.
- Using extensive empirical results on random as well as real rule sets from routers of a large tier-1 ISP, we evaluate the effectiveness and potential of our proposed techniques in current and emerging applications. In our experiments, the proposed mechanisms often lead to a reduction of TCAM sizes by almost an order of magnitude.

1.5 Roadmap

The rest of the paper is organized as follows. In Section 2, we present the motivations of this work. The detailed solution is schemed in Section 3, and we evaluate its effectiveness and potential using empirical results based on real packet classifiers as well as random rule sets in Section 4. After reviewing related work in Section 5, we conclude the paper in Section 6.

2. MOTIVATIONS

In this section, we present the observations that motivate our proposed techniques for defining semantically equivalent packet classifiers requiring fewer TCAM entries to represent. The examples we use to illustrate the observations here are meant to be simple and intuitive. However, systematic solutions are non-trivial to design, and we will present such a design in details in Section 3.

Expanding rules: Consider the packet classifier shown in the second column of Table 2. If literally translated into TCAM entries, r_2 takes 4 TCAM entries as shown in the third column of Table 2. The observation is that we can safely expand the range of r_2 to be $x \in [64, 255]$ as shown in the fourth column of Table 2, which takes only 2 TCAM entries to represent. The semantics of the rule set remains the same while the number of TCAM entries needed reduces from 6 to 4 as shown in the fifth column of Table 2.

Trimming rules: Consider the packet classifier shown in the second column of Table 3. If literally translated into TCAM entries, r_2 takes 4 TCAM entries as shown in the third column of Table 3. The observation is that r_2 will only match packets with $x \in [128, 255]$, since packets with $x \in [100, 127]$ will first match r_1. Therefore, we can safely trim the range of r_2 to be $x \in [128, 255]$ as shown in the fourth column of Table 3, which takes only 1 TCAM entry to represent. The semantics of the packet classifier remains the same while the number of TCAM entries needed reduces from 5 to 2 as shown in the fifth column of Table 3.

Adding rules: Consider the packet classifier shown in the second column of Table 4. If literally translated into TCAM entries, r_1 takes 3 TCAM entries as shown in the third column of Table 4. The observation is that if we add a new rule r_0 to cover the small "hole" of $x \in [120, 127]$, we can safely expand r_1 to cover the range of $x \in [64, 127]$ as shown in the fourth column of Table 4, which takes only 1 TCAM entry to represent. In the new packet classifier, r_0 takes 1 TCAM entry and r_1 takes only 1 TCAM entry as well. The semantics of the packet classifier remains the same while the number of TCAM entries needed reduces from 4 to 3 as shown in the fifth column of Table 4.

Merging rules: Consider the packet classifier shown in the second column of Table 5. If literally translated into TCAM entries, r_2 and r_3 take 1 TCAM entry and 3 TCAM entries, respectively, as shown in the third column of Table 5. The observation is that r_2 and r_3 have the same decision and can be merged into one single rule r_2' as shown in the fourth column of Table 5. The semantics of the packet classifier remains the same while the number of TCAM entries needed reduces from 6 to 3, as shown in the fifth column of Table 5. Note that this merger can also be thought of as expanding either r_2 or r_3 into r_2' and then removing the other, which now becomes redundant. As we will later see, this is exactly how the idea of merging rules is carried out in our solution.

3. DESIGN

The observations and ad hoc solutions presented in Section 2 is a good starting point but not yet a complete roadmap toward our objective of defining a semantically equivalent packet classifier that takes fewer TCAM entries to represent. To be practical, we need a systematic solution. In this section, we present such a solution in details. Before we proceed to present the detailed design, we first build a formal foundation for understanding the problem.

3.1 Preliminaries

In the first-match model, a packet classifier is an ordered set $R = \{r_1, r_2, \cdots, r_n\}$ of rules, where each rule r_i is composed of two parts: a conjunctive *predicate* and a *decision*.

	Before expanding		After expanding	
	Rule	TCAM entries	Rule	TCAM entries
r_1:	$x \in [32, 79] \rightarrow deny$	001xxxxx $\rightarrow$ *deny*	$x \in [32, 79] \rightarrow deny$	001xxxxx $\rightarrow$ *deny*
		0100xxxx $\rightarrow$ *deny*		0100xxxx $\rightarrow$ *deny*
r_2:	$x \in [72, 255] \rightarrow permit$	01001xxx $\rightarrow$ *permit*	$x \in [64, 255] \rightarrow permit$	01xxxxxx $\rightarrow$ *permit*
		0101xxxx $\rightarrow$ *permit*		1xxxxxxx $\rightarrow$ *permit*
		011xxxxx $\rightarrow$ *permit*		
		1xxxxxxx $\rightarrow$ *permit*		

Table 2: Packet classifiers and their TCAM representations before/after expanding.

	Before trimming		After trimming	
	Rule	TCAM entries	Rule	TCAM entries
r_1:	$x \in [96, 127] \rightarrow deny$	011xxxxx $\rightarrow$ *deny*	$x \in [96, 127] \longrightarrow deny$	011xxxxx $\rightarrow$ *deny*
r_2:	$x \in [100, 255] \rightarrow permit$	011001xx $\rightarrow$ *permit*	$x \in [128, 255] \longrightarrow permit$	1xxxxxxx $\rightarrow$ *permit*
		01101xxx $\rightarrow$ *permit*		
		0111xxxx $\rightarrow$ *permit*		
		1xxxxxxx $\rightarrow$ *permit*		

Table 3: Packet classifiers and their TCAM representations before/after trimming.

	Before adding r_0		After adding r_0	
	Rule	TCAM entries	Rule	TCAM entries
r_0:			$x \in [120, 127] \rightarrow permit$	01111xxx $\rightarrow$ *permit*
r_1:	$x \in [64, 119] \rightarrow deny$	010xxxxx $\rightarrow$ *deny*	$x \in [64, 127] \rightarrow deny$	01xxxxxx $\rightarrow$ *deny*
		0110xxxx $\rightarrow$ *deny*		
		01110xxx $\rightarrow$ *deny*		
r_2:	$x \in [0, 255] \rightarrow permit$	xxxxxxxx $\rightarrow$ *permit*	$x \in [0, 255] \rightarrow permit$	xxxxxxxx $\rightarrow$ *permit*

Table 4: Packet classifiers and their TCAM representations before/after adding r_0.

	Before merging		After merging	
	Rule	TCAM entries	Rule	TCAM entries
r_1:	$x \in [96, 111] \rightarrow permit$	0110xxxx $\rightarrow$ *permit*	$x \in [96, 111] \rightarrow permit$	0110xxxx $\rightarrow$ *permit*
r_2 (r_2'):	$x \in [64, 95] \rightarrow deny$	010xxxxx $\rightarrow$ *deny*	$x \in [64, 127] \rightarrow deny$	01xxxxxx $\rightarrow$ *deny*
r_3:	$x \in [100, 127] \rightarrow deny$	011001xx $\rightarrow$ *deny*		
		01101xxx $\rightarrow$ *deny*		
		0111xxxx $\rightarrow$ *deny*		
r_4:	$x \in [0, 255] \rightarrow permit$	xxxxxxxx $\rightarrow$ *permit*	$x \in [0, 255] \rightarrow permit$	xxxxxxxx $\rightarrow$ *permit*

Table 5: Packet classifiers and their TCAM representations before/after merging.

Namely,

$$r_i : \bigwedge_{j=1}^{d} (x_j \in [l_j, h_j]) \longrightarrow decision.$$

Each x_j denotes a packet header field. Note that exact values and prefixes are both special cases of ranges. To facilitate our understanding and analysis, we can view each distinct decision as a distinct "color" and view the semantics of a packet classifier as a coloring of the d-dimensional space defined by the d packet header fields specified in the packet classifier, where the domain of each dimension is the domain of the corresponding packet header field. For example, the dimension corresponding to the source IP address field has a domain of $[0, 2^{32} - 1]$.

Within this d-dimensional space, the conjunctive predicate of each rule delimits a d-dimensional hypercube, which we refer to as the *definition region* of the rule. Each packet contains exact values in the specified fields and thus maps to a specific point in the d-dimensional space. The decision of a rule is the *color* that is used to color the definition region of that rule. A packet classifier as an ordered set of rules defines a coloring of the d-dimensional space with overwriting. Specifically, each point in the d-dimensional space may be contained in the definition region of multiple rules, but its color is defined to be the color of the highest priority rule whose definition region contains that point. The colors of lower priority rules are overwritten by that color. The region that is first colored by rule r is considered *dominated* by rule r, and we refer to this region as the *domination region* of rule r.

A packet classifier is considered *complete* if and only if every point within the d-dimensional space is colored and hence dominated by some rule in that packet classifier. In other words, a complete packet classifier partitions the d-dimensional space into the domination regions of its rules. In terms of packet classification, that means any packet is matched by some rule. Real packet classifiers are inherently complete, as they are defined to be able to handle any incoming packet. Two packet classifiers are *semantically equivalent* if and only if they color any point with the

same color. In terms of packet classification, that means they process any packet with the same decision.

3.2 Framework

Let us first consider an arbitrary rule r_i in an ordered rule set $R = \{r_1, r_2, \cdots, r_n\}$. To represent r_i using as few TCAM entries as possible, we need to figure out an answer to the following questions. Our answer to these questions will also shed some light on the design of an effective solution.

First, in order to preserve the semantics of the given rule set, what should we do? We suggest that the TCAM entries we use to represent r_i should cover the portion of r_i's definition region that is neither covered by higher priority rules (i.e., $\{r_1, r_2, \cdots, r_{i-1}\}$) nor covered by lower priority rules (i.e., $\{r_{i+1}, r_{i+2}, \cdots, r_n\}$) of the same color. Because otherwise, the coloring of this region will be different from its original coloring defined by the given packet classifier. For simplicity, we refer to this portion of r_i's definition region as the *core region* of rule r_i.

Second, while preserving the semantics of the given rule set, what freedom can we have when reducing the number of TCAM entries needed to represent r_i? Consider the region S covered by the TCAM entries we use to represent rule r_i. The general principle is that, if a point $p \in S$ is contained in the definition region of some rule in $\{r_1, r_2, \cdots, r_{i-1}\}$, the TCAM entries we use can assign an arbitrary color to p. Otherwise, since we want to preserve the semantics of the rule set after processing r_i, the TCAM entries should assign to p a color that is the same as the color assigned to p by the rule set.

Third, with the allowed freedom, how should we maneuver to reduce the number of TCAM entries needed? As motivated in Section 2, we can achieve our objective by trimming, expanding, adding and merging rules. Based on the understanding and insights we have achieved so far, we design our solution using these techniques as follows.

In particular, we start with a relatively simple solution that processes individual rules in a bottom-up fashion, seeking to represent individual rules using as few TCAM entries as possible. Meanwhile, after processing each rule, the semantics of the packet classifier should remain the same. This local improvement can be done by trimming, expanding and possibly adding rules. Later on, we consider the more complicated interaction between rules and extend the local improvement solution to more aggressively reduce the number of TCAM entries needed through merging rules. As we will see, adding rules and merging rules are both implemented as extensions to expanding rules in our design.

3.3 Trimming rules

Based on our above answer to the first question, as the first step of our solution we trim each rule r_i along all dimensions so that the definition region of r_i after trimming is the minimum d-dimensional hypercube that encloses the core region of r_i. To focus on our proposed techniques and main results, in this paper we use the algorithms proposed in [13] for computing the core region of each rule. The objective of [13] is to minimize the total number of rules instead of TCAM entries. Therefore, they either remove a rule (if its core region is empty) or leave the rule intact (if its core region is not empty). Partial trimming of rules as we propose here is not considered in [13]. We refer interested readers to [13] for the technical details of those algorithms.

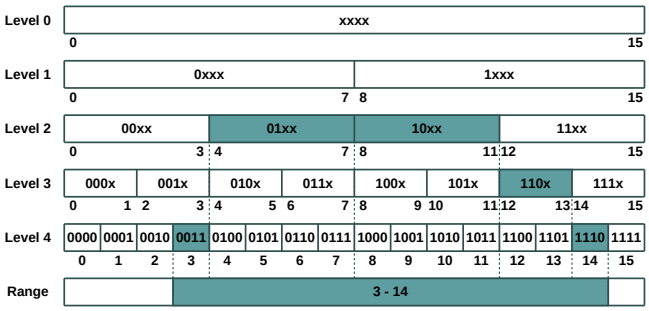


Figure 1: Partition of a range into prefixes.

Besides that, some additional issues specific to partial trimming of rules also need to be taken care of here. For example, for dimensions that are typically in the form of prefixes (e.g. source IP address and destination IP address), we prefer to keep them as a prefix after trimming. Because otherwise we risk dramatically increasing the number of TCAM entries needed to represent those dimensions and hence rule r_i. If necessary, we can always expand the trimmed rule r_i along these dimensions to be a (possibly smaller) prefix so that such a dimension of r_i can still be represented using one single TCAM entry. Note that at least we can restore such a dimension to its original prefix. For a dimension that is typically in the form of a non-prefix range (e.g. source port range and destination port range), we leave it as is for now and will explain how to process such ranges in upcoming subsections.

3.4 Expanding rules

Starting from the trimmed rule r_i, we may be able to reduce the number of TCAM entries needed to represent r_i by expanding r_i along the non-prefix range dimensions. This expansion operation has a limit. In particular, without additional post-processing, we should not expand r_i too much so that the semantics of the rule set is changed. Thus, we have delimited a scope within which r_i can be expanded. However, it still remains to figure out a specific strategy for expanding r_i.

We start by looking into the details of how should a range be expanded to reduce the number of prefixes needed to represent it. First of all, there are $m+1$ different levels of partition of an m-bit domain $[0, 2^m - 1]$ into a set of prefixes. For each $0 \leq i \leq m$, the level i partition cuts the entire domain into 2^i prefixes each of size 2^{m-i}. In particular, for each $0 \leq j \leq 2^i - 1$, the jth level i prefix starts at $j \cdot 2^{m-i}$. An example of a 4-bit domain is given in Figure 1. Range $[3, 14]$ can be represented using two level 2 prefixes `01xx` and `10xx`, one level 3 prefix `110x`, and two level 4 prefixes `0011` and `1110`.

Every prefix corresponds to exactly one TCAM entry. To reduce the number of TCAM entries needed, we need to reduce the number of prefixes by merging two or more contiguous prefixes into a larger prefix. Such a merger may be enabled by expanding the range such that the larger prefix we target is covered by the expanded range. From the illustration in Figure 1, it is easy to see that only prefixes on the left end (i.e., lower end) or the right end (i.e., upper end) can be merged through expansion. Then the question will be: *Which end of the range should we expand and how much should we expand it?*

$rule\_id$	the index of the rule in processing
idx	the index of the dimension in processing
$c[i]$	the color of the ith rule
$Pl[i][j]$	the lower end of the jth leftmost prefix of the range $x_i \in [l[i], h[i]]$
$Ph[i][j]$	the upper end of the jth leftmost prefix of the range $x_i \in [l[i], h[i]]$
$N[i]$	the number of TCAM entries needed to represent the range $x_i \in [l[i], h[i]]$
$\Delta[i]$	the maximum amount by which $N[i]$ can be decreased through a minimum expansion of the range x_i
$B[i]$	$B[i]$ denotes whether $N[i]$ can be reduced by expanding the rule along the range x_i or not
$lstride$	the amount by which the range needs to be expanded toward the left (i.e., lower) end
$rstride$	the amount by which the range needs to be expanded toward the right (i.e., upper) end

Table 6: Notations

```
    void Expand_Range (int rule_id, idx)
1   if N[idx] = 1
2       B[idx] = false;
3       return;
4   find the minimum prefix [lower, upper] that contains
        the leftmost 2 prefixes of dimension idx;
5   lstride = MAX;
6   if Allowable (rule_id, idx, lower, upper)
7       lstride = (Pl[idx][1] − lower) + (upper − Ph[idx][2]);
8   find the minimum prefix [lower, upper] that contains
        the rightmost 2 prefixes of dimension idx;
9   rstride = MAX;
10  if Allowable (rule_id, idx, lower, upper)
11      rstride = (Pl[idx][N[idx] − 1] − lower)+
            (upper − Ph[idx][N[idx]]);
12  if lstride = MAX && rstride = MAX
13      B[idx] = false;
14      return;
15  if lstride ≤ rstride
16      Pl[idx][1] = lower;
17      Ph[idx][2] = upper;
18      while [Pl[idx][1], Ph[idx][2]] is a prefix
19          merge the leftmost two prefixes;
20  if lstride > rstride
21      Pl[idx][N[idx] − 1] = lower;
22      Ph[idx][N[idx]] = upper;
23      while [Pl[idx][N[idx] − 1], Ph[idx][N[idx]]] is a prefix
24          merge the rightmost two prefixes;
25  update N[idx];
26      B[idx] = false;

    bool Allowable (int rule_id, idx, lower, upper)
1   insert before rule rule_id a new rule r such that
        dimension idx is [lower, upper] &&
        other dimensions are the same as rule rule_id;
2   compute the core region of rule r;
3   remove rule r;
4   if the core region is empty
5       return true;
6   else
7       return false;
```

Table 7: Pseudo code description of the simple greedy algorithm for expanding a range. Notations are annotated in Table 6.

One intuition is that in each expansion we prefer to expand a dimension as little as possible to enable a merger, so that hopefully we will be less constrained when expanding other dimensions. Based on this intuition, we propose the following simple greedy algorithm. Given the sequence of prefixes of a range, let us first consider the leftmost two prefixes. We want to find the smallest prefix that contains the leftmost two prefixes. In the example in Figure 1, the leftmost two prefixes are `0011` and `01xx`, and the smallest prefix that contains them is the level 1 prefix `0xxx`. In order to merge `0011` and `01xx` into `0xxx`, we need to expand the range by 3 (from $[3, 14]$ to $[0, 14]$). Then let us look at the rightmost two prefixes `1110` and `110x`. The smallest prefix that contains them is the level 2 prefix `11xx`. In order to merge `1110` and `110x` into `11xx`, we need to expand the range by only 1 (from $[3, 14]$ to $[3, 15]$). Therefore, we prefer to merge the rightmost two prefixes by expanding the range from $[3, 14]$ to $[3, 15]$. The pseudo code description of this simple greedy algorithm is presented as function `Expand_Range` in Table 7.

So far in our description of the range expansion algorithm, an important detail has been deliberately ignored in order to focus on the question we just proposed and answered above. Now we turn to the previously ignored problem of *whether an expansion should be allowed or not.* Our solution is very simple. In particular, we temporarily insert the expanded new rule r before the rule $rule\_id$ that is currently in processing, and then compute the core region of this new rule. If its core region is empty, that means expanding the rule $rule\_id$ does not change the semantics of the rule set and thus can be allowed. Otherwise, the semantics of the rule set will be changed by the proposed expansion and the expansion should not be allowed. The pseudo code description of this simple greedy algorithm is presented as function `Allowable` in Table 7. The correctness of this algorithm is easy to verify. Because the expanded new rule r changes the color of some point(s) if and only if its core region contains the point(s).

Now we have figured out how should we expand a range to reduce the number of prefixes needed to represent it. Note that some rules may specify range clauses on more than one fields. In such cases, another question that needs to be answered is, *In what order should we expand the non-prefix dimensions?* To facilitate our understanding of the problem, we establish for now the simplifying principle that each time we only seek to merge either the leftmost or the rightmost two prefixes. Although it is possible to merge more than two prefixes through one expansion, we can view such a merger as a series of mergers involving two prefixes, where all mergers except the first one can be done without range expansion. For example, by expanding the range in Figure 1 to $[3, 15]$ (if allowed), we can merge the rightmost three prefixes `1110`, `110x` and `10xx` into one single larger prefix `1xxx`.

```
    void Expand_Rule (int rule_id)
1   for each dimension i
2      B[i] = true;
3   bool done = false;
4   while (!done)
5      idx ← i such that
           B[i] = true && N[i] is minimum;
6      Expand_Range (idx);
7      done = true;
8      for each dimension i
9         if B[i] = true
10           done = false;
```

Table 8: Pseudo code description of the simple greedy algorithm for expanding a rule. Notations are annotated in Table 6.

```
    void Expand_Rule (int rule_id)
1   for each dimension i
2      B[i] = true;
3   bool done = false;
4   while (!done)
5      idx ← i such that
           B[i] = true && Δ[i]/N[i] is maximum;
6      Expand_Range (idx);
7      update Δ;
8      done = true;
9      for each dimension i
10        if B[i] = true
11           done = false;
```

Table 9: Pseudo code description of the improved greedy algorithm for expanding a rule. Notations are annotated in Table 6.

Alternatively, we can also view this merger as two consecutive mergers. The first merger involves expanding the range to $[3, 15]$ and merging the rightmost two prefixes `1110` and `110x` into `11xx`. The second merger involves no range expansion but merging the new rightmost two prefixes `10xx` and `11xx` into `1xxx`.

In general, let us assume that some rule r specifies k non-prefix range fields $x_1, x_2, \cdots, x_k$. Assume that for each $1 \leq i \leq k$, N_i prefixes are needed to represent the range $x_i \in [l_i, h_i]$. The total number of TCAM entries needed to represent rule r is $\prod_{i=1}^{k} N_i$. If we choose to expand the range $x_i \in [l_i, h_i]$ and decrement N_i by 1, the total number of TCAM entries needed to represent rule r is decreased by $\frac{\prod_{j=1}^{k} N_j}{N_i}$. Therefore, it follows that we would prefer to expand the range $x_i \in [l_i, h_i]$ such that N_i is the smallest and hence reducing N_i by 1 will lead to the largest reduction in the total number of TCAM entries needed to represent rule r. A pseudo code description of this simple greedy algorithm for choosing a dimension to expand is presented in Table 8.

In this simple greedy algorithm, if a dimension x_i is chosen for expansion, that means N_i is the smallest. As N_i is only decreasing, N_i will always be the smallest and hence we will keep expanding x_i until it can not or should not be further expanded. Therefore, this simple greedy algorithm expands ranges in non-decreasing order of their N_i values. Intuitively, this fixed order (except ties) may not be as effective as we expect. An improved greedy algorithm can take into account the maximum amount Δ_i by which N_i can be reduced through a minimum expansion. In particular, we prefer to expand the range $x_i \in [l_i, h_i]$ that will maximize $\frac{\Delta_i \times \prod_{j=1}^{k} N_j}{N_i}$ or $\frac{\Delta_i}{N_i}$. Intuitively, a minimum expansion of such a dimension x_i will yield the largest reduction in the total number of TCAM entries needed to represent rule r among all choices of x_i. Once such a range $x_i \in [l_i, h_i]$ is chosen, we perform a minimum expansion of it and merge all the prefixes that can be merged. Then, based on the expanded range, we update all the Δ_i values and N_i values, and choose the next dimension for expansion based on these new Δ_i values and new N_i values. A pseudo code description of this improved greedy algorithm is given in Table 9.

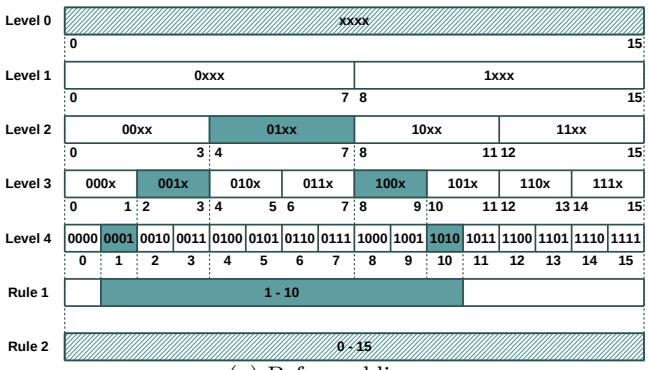
3.5 Adding rules

In Section 2, we have seen adding rules as an effective means of reducing the number of TCAM entries needed to represent a rule set. We now proceed to the design of a systematic implementation of this idea. Before that, we first need to know the answer to the following question: *When and how should we add rules?*

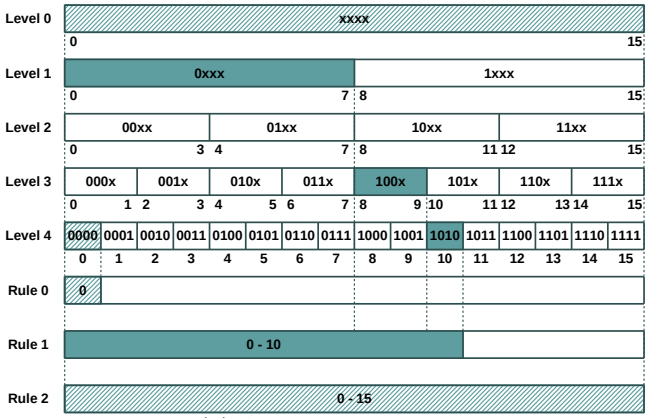
Very nicely, the idea of adding rules can be viewed as a natural choice during the process of expanding a rule. To facilitate our understanding, consider the example in Figure 2(a). When we try to expand rule 1 in order to merge its leftmost two prefixes, we find that the required expansion is unfortunately not allowed. Because otherwise we would change the color of $[0, 0]$, which is originally colored by rule 2. In the function `Allowable` in Table 7, we simply give up. Similarly, we have the same problem when trying to expand rule 1 toward its right end in order to merge its rightmost two prefixes, and we simply give up as well. In such cases where we can not proceed to expand a rule, adding rules arises as a natural step over the "hole". In the example in Figure 2(a), if we add a new rule 0 before the current rule in processing (i.e., rule 1) to "fill up" the hole of $[11, 15]$, we can safely expand rule 1 to cover $[1, 15]$ so that its rightmost two prefixes can be merged into one single prefix `1xxx`.

Now the question is: *Is there such a new rule that will enable the proposed expansion? If yes, what should the color of the new rule be?* We can use exactly the solution proposed as the function `Allowable` in Table 7, except that we here explore the possibility of adding a new rule r' if the core region of r is not empty. Along the dimension under expansion, rule r' covers the minimum range that is needed to enclose the core region of r. Along other dimensions, rule r' covers the same range as rule *rule_id* and r. In order to figure out the feasible color of r' (if such a color exists at all), for each possible color we temporarily insert a new rule r' of that color before the rule *rule_id*. We then compute the core region of rule r'. If its core region is empty, that means at the presence of r', expanding the rule *rule_id* will not change the semantics of the rule set and thus can be allowed. Otherwise, the semantics of the rule set will be changed by the proposed expansion and we continue to try other colors. If none of the colors works out, then the expansion is not allowed.

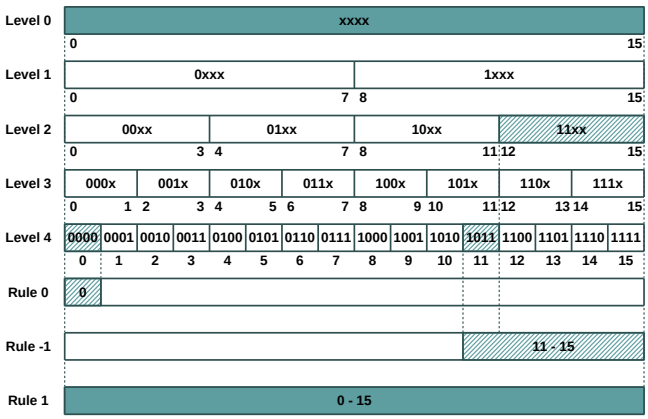
However, an allowed solution is not necessarily a desirable one. After all, adding rules is a technique that is meant to approach the ultimate objective of reducing the number of TCAM entries needed. For example, the proposal of adding a new rule to cover $[11, 15]$ will actually increase the num-



(a) Before adding



(b) After adding rule 0



(c) After adding rule -1 and removing rule 2

Figure 2: An example of adding rules.

ber of TCAM entries needed and hence is not a desirable choice. Because we are adding two more prefixes to remove only one. Therefore, we still have to calculate the cost efficiency of an allowed solution. If we are not reducing the number of TCAM entries needed, we should give up. For the example in Figure 2(a), the right choice is to add a new rule 0 before rule 1 to fill up the hole of $[0, 0]$. As is shown in Figure 2(b), we will be able to expand rule 1 to cover $[0, 10]$, which takes only 3 prefixes to represent. Consequently, the total number of TCAM entries needed is reduced from 6 to 5. A pseudo code description of the modified `Allowable` algorithm considering the additional choice of adding rules is given in Table 10.

bool Allowable (**int** *rule_id*, *idx*, *lower*, *upper*)
1 insert before rule *rule_id* a new rule r such that
  dimension *idx* is $[lower, upper]$ &&
  other dimensions are the same as rule *rule_id*;
2 compute the core region of rule r;
3 remove rule r;
4 **if** the core region is empty
5 **return true**;
6 **else for** each color c
7 insert before rule *rule_id* a new rule r' of color c such that
    dimension *idx* is minimum to enclose the core region of r &&
    other dimensions are the same as rule *rule_id*;
8 compute the core region of rule r';
9 **if** the core region of r' is empty
10 **if** the number of TCAM entries will be reduced
11 **return true**;
12 remove rule r';
13 **return false**

Table 10: Pseudo code description of the modified algorithm for determining whether a proposed range expansion should be allowed or not, considering the additional choice of adding rules. Notations are annotated in Table 6.

Interestingly, now the first proposal of adding a rule to fill up the hole of $[11, 15]$ becomes a good idea. As is shown in Figure 2(c), by adding a new rule -1 before rule 1, we can safely expand rule 1 to cover $[0, 15]$, which is represented by one single prefix `xxxx`. Meanwhile, rule 2 now becomes redundant and hence can be removed from the rule set. Consequently, we further reduce the total number of TCAM entries needed from 5 to 4.

3.6 Merging rules

So far in our `Expand_Range` algorithm, we have only been focusing on the potential benefit of reducing the number of TCAM entries needed to represent the current rule (we are processing), without considering the possibility of merging with other rules. Specifically, if expanding a rule r along a dimension (with possibly adding rules) will not reduce the number of TCAM entries needed to represent r, we will not perform the expansion. However, such "locally useless" expansions may potentially make some other rules or TCAM entries become redundant and hence we can reduce the total number of TCAM entries needed to represent the rule set after removing those redundant rules or TCAM entries. As we have discussed in Section 2, merging two rules can be viewed as expanding one rule to make the other rule redundant and then removing it. Throughout the discussion of our algorithm, we will always focus on the rules that have already been processed by our algorithm. Because rules to be processed may be changed after processing, which may invalidate the considerations we currently make.

Within a rule set $R = \{r_1, r_2, \cdots, r_n\}$, redundant rules can be categorized as either *upward redundant* or *downward redundant* [13], which can be formally defined as follows. The intuition is that, after removing redundant rules from a packet classifier, the action taken on any packet by the packet classifier should remain the same.

DEFINITION 1 (UPWARD REDUNDANCY). *Rule $r_i \in R$ is considered upward redundant if and only if the domination region of r_i is empty, i.e., the definition region of r_i is contained by the union of the definition regions of $r_1, r_2, \cdots, r_{i-1}$.*

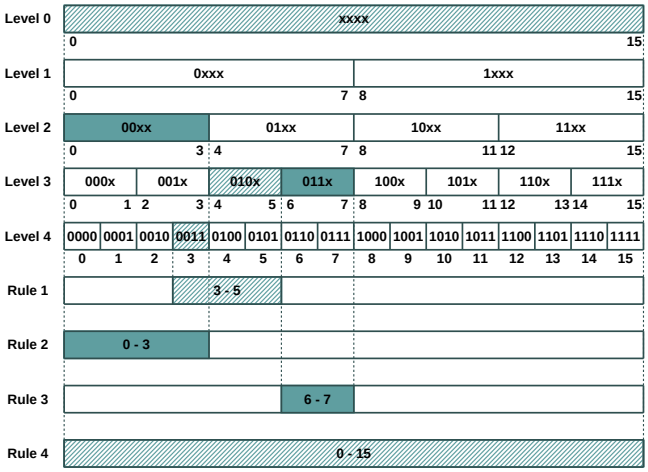
DEFINITION 2 (DOWNWARD REDUNDANCY). *Rule r_i is considered downward redundant if and only if the following conditions hold. (1) r_i is not upward redundant; (2) For each point p in the domination region of r_i, the first rule in $\{r_{i+1}, r_{i+2}, \cdots, r_n\}$ whose definition region contains p must have the same decision as r_i.*

It is clear that expanding the current rule will not make any already processed rule become downward redundant if originally it is not downward redundant. Thus, in our algorithm we only need to consider upward redundancy. If expanding a rule *rule_id* along dimension *idx* will reduce the number of TCAM entries needed to represent *rule_id*, we still expand it as usual. Otherwise, we consider the minimum expansion of rule *rule_id* along dimension *idx* (in terms of the number of additional TCAM entries needed) that will make some TCAM entry/entries of some processed rule(s) become upward redundant. If the total number of TCAM entries will be reduced, we will perform this minimum expansion. To avoid later repeated counting of those upward redundant TCAM entries, we immediately remove them from the rule set. Such a minimum expansion of *rule_id* repeats until at some point we cannot further decrease the total number of TCAM entries needed.

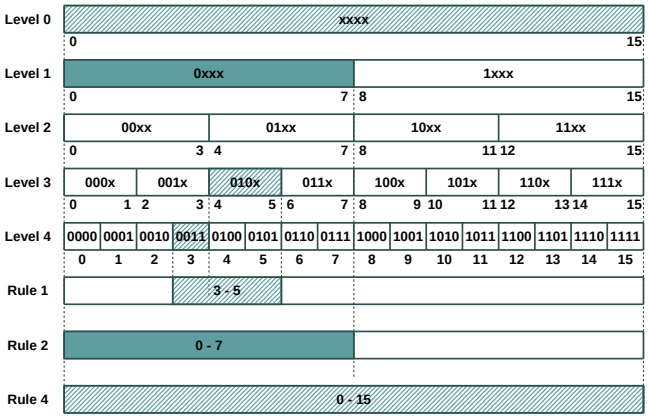
To take advantage of this technique, there are two places in our current algorithms that can be improved. (1) In lines 1–3 of the `Expand_Range` algorithm in Table 7, we give up expanding a range if it is already a prefix. With the additional option of merging rules, now we want to check out the possibility of expanding such a prefix to make some TCAM entry/entries of some already processed rules become upward redundant. If this is possible and the total number of TCAM entries needed will be reduced, we will expand the prefix and remove the TCAM entries which now become upward redundant. For example, consider the rules in Figure 3(a). Although rule 2 is already a prefix, we find that expanding it to cover $[0, 7]$ will make rule 3 upward redundant. After expanding rule 2 and removing rule 3, we reduce the total number of TCAM entries needed from 5 to 4, as is shown in Figure 3(b). Note that when processing rule 3, we do not expand it. Because as we have previously pointed out, at that point we only consider rules below rule 3 and thus we cannot benefit from expanding rule 3. (2) In line 10 of the `Allowable` algorithm in Table 10, when determining if the total number of TCAM entries will be reduced or not, we can take the possibility of merging rules into account as well.

3.7 Removing redundancy

During our bottom-up processing of individual rules, some rules or TCAM entries may become redundant due to various reasons such as rule expansion and insertion. Thus, after the whole processing is completed, we need to remove possibly upward redundant and downward redundant TCAM entries in a bottom-up round and a top-down round, respectively. Again, we refer interested readers to the literature [13] for technical details about how to remove upward redundancy and downward redundancy.



(a) Before merging



(b) After expanding rule 2 and removing rule 3

Figure 3: An example of merging rules.

3.8 Summary of design

To better convey a picture of our proposed techniques, we here briefly summarize the entire design of our solution. Four techniques are proposed, namely trimming rules, expanding rules, adding rules and merging rules. As the first "preprocessing" step of our scheme, we first trim each individual rule such that its trimmed definition region is the minimum hypercube that encloses its core region. If necessary, non-prefix range dimensions of the trimmed rule can be expanded to be the minimum prefix that contains the trimmed range.

After trimming the rules, we try to reduce the number of TCAM entries needed to represent the rule set by expanding, adding and merging rules. Specifically, adding rules and merging rules are both designed as extensions to the basic idea of expanding rules. In many cases, when simply trying to expand a rule is not possible or helpful, further reduction in the number of TCAM entries needed can be achieved by adding new rules and merging with other rules, which make rule expansion either possible or helpful.

Interestingly, we observe that repeatedly applying our proposed techniques can achieve significantly more reduction in the number of TCAM entries needed. This is because the new rules added by our scheme can also be optimized by our proposed techniques of expanding, adding and merging

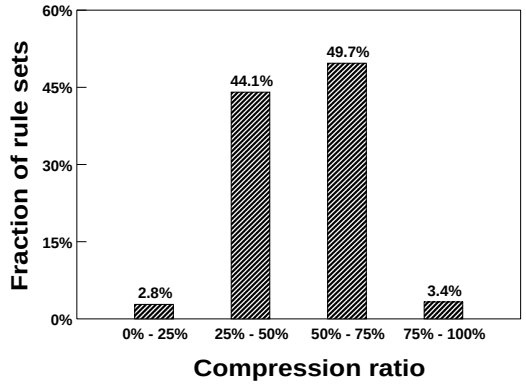


Figure 4: Optimization results of real rule sets.

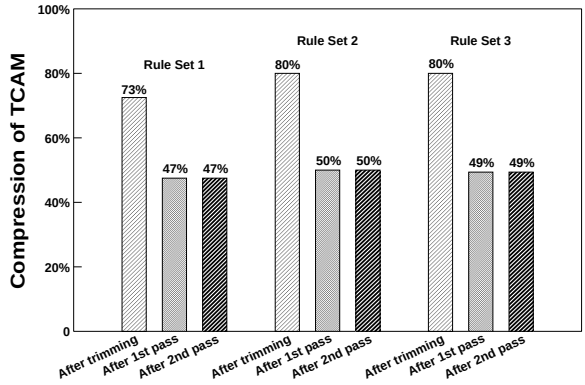


Figure 5: Optimization results of real rule sets.

rules. This interesting observation will be made clear in further details through experiments in Section 4.

4. EVALUATION

In this section, we evaluate the effectiveness as well as the potential of our proposed techniques. Two sets of experiments are conducted. We first evaluate the effectiveness of our techniques on a large collection of real packet classifiers provided by a tier-1 ISP. To study our scheme in further details and to evaluate its potential in emerging applications, we then conduct a set of experiments based on randomly generated rule sets. The results reported in this section are obtained by applying expanding, adding and merging rules for two *passes*. As we will explain later on, little further compression is obtained when applying these techniques for more than two passes.

4.1 Real rule sets

We carefully examined a large collection of thousands of packet classifiers obtained from routers of a large tier-1 ISP backbone network. The size of these packet classifiers varies from dozens to hundreds of rules. In the data set we have examined, packet classifiers typically have at most one port range in each rule. Rules that specify two port ranges are very rare. Therefore, the multiplicative explosion in the number of TCAM entries needed to represent a rule is rarely observed. Nevertheless, our proposed techniques of trimming, expanding, adding and merging rules can still significantly reduce the number of TCAM entries needed. Let us define *compression ratio* to be the ratio between the numbers of TCAM entries needed after compression and before compression. The average compression ratio achieved on real rule sets is $1 : 1.85 = 54\%$. Specifically, trimming rules reduces the number of TCAM entries needed by 32%; expanding, adding and merging rules reduce the number of TCAM entries needed by 14%. The distribution of compression ratios achieved on these rule sets is presented in Figure 4.

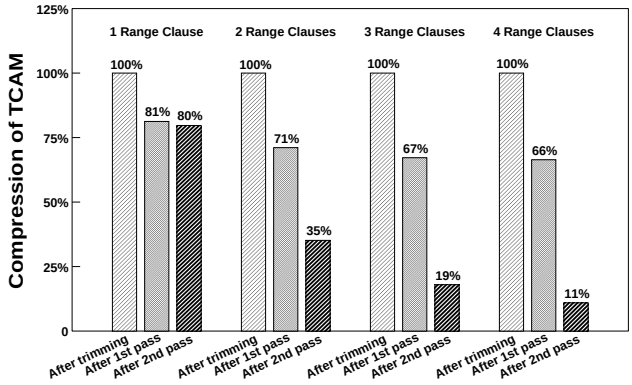


Figure 6: Optimization results of random rule sets.

Optimization results of three rule sets randomly selected from the collection of real rule sets are presented in Figure 5 for further details. On average, trimming rules reduces the number of TCAM entries needed by $20\% - 30\%$; expanding, adding and merging rules can further achieve approximately as much compression. For these rule sets, we observe few rules being added. Consequently, little further compression is achieved during the second pass.

4.2 Random rule sets

As finer and finer differentiation of packets becomes necessary, more and more fields are being checked by packet classifiers and hence range clauses are expected to be specified more and more frequently. For example, some firewall packet filters have been checking application level data for security purposes. To better evaluate the effectiveness and potential of our proposed techniques in emerging applications, we have also done simulations based on random rule sets, which are generated as follows. For source IP address and destination IP address, we generate a random IP address prefix for each of them. A random number is assigned to the protocol type field. Source port range and destination port range are generated as a random sub-range of $[0, 65535]$. The action of each rule is randomly picked from the set of actions observed in the real packet classifier. To avoid redundant rules, the last *default* rule (which matches any incoming packet that is not matched by any other rules) specifies an action that is different from any other rule. This is consistent with our observation that real rule sets typically have a default rule whose decision is different from the decision of most other rules. As we expect, redundant rules are rarely observed in our simulations.

For different numbers of range clauses, we run simulations on 100 such random rule sets, each of which contains 1000 rules. The average compression ratios are presented in Figure 6, and the average increase in the number of rules (which

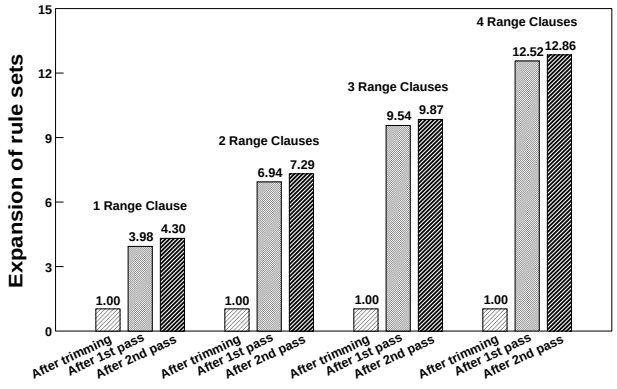


Figure 7: Optimization results of random rule sets.

we refer to as *expansion ratio*) is presented in Figure 7. As more range clauses are specified, our proposed techniques can achieve even higher compression ratios and compression ratio tends to decrease faster than the increase in the number of range clauses. For rule sets with 1 range clause, our proposed techniques can reduce the number of TCAM entries needed by 20%. For rule sets with 2 range clauses, the number of TCAM entries needed is reduced by a factor of roughly 3. For rule sets with 3 range clauses, we can reduce the number of TCAM entries needed by a factor of 5. For rule sets with 4 range clauses, our proposed techniques can reduce the number of TCAM entries needed by a factor of 9. The sharp decrease in the number of TCAM entries needed (Figure 6) is achieved by the sharp increase in the number of rules (Figure 7), which clearly reveals the effectiveness of expanding through adding rules.

Note that the compression is completely achieved by expanding, adding and merging rules, which is essentially different from the case of real rule sets. Intuitively, this is because randomly generated rules rarely overlap and interact with each other – the definition regions of two rules overlap if and only if they overlap on every dimension, which rarely happens in random rule sets. Meanwhile, the default rule has a different decision from other rules. Therefore, the other rules can barely be trimmed without changing the semantics of the rule set.

By comparing the results in Figures 6 and 7, we can also clearly understand the reason why the second pass can significantly compress the TCAM while more passes can hardly help. The reason is because the first pass adds a significant number of new rules, which make the overall compression achieved by the first pass not so striking but can be later optimized by the second pass. The optimization during the second pass is mostly expanding and merging rules. Only a small number of new rules are further added during the second pass. Therefore, little further compression can be achieved by more than two passes.

5. RELATED WORK

There has been two orthogonal lines of research on packet classification in the most commonly used first-match model. Given a packet classifier, a long thread of research [11, 20, 7, 21, 8, 25, 18, 2, 1, 24, 19, 10, 23] aims to design efficient solutions for performing packet classification. The other line of research aims to reduce the size of the given rule set, in terms of either the number of rules or the number of TCAM entries needed to represent that rule set. In some sense, the latter can be viewed as a preprocessing stage preceding the former: after compressing the rule set, packet classification solutions can apply on the compressed rule set and hence benefit from its reduced size. Our work in this paper belongs to the second type, and we here briefly review the related work of the second type.

Given the inherent hardness of general packet classification as a theory problem, some researchers have proposed to reduce the number of rules in a given rule set in order to improve the efficiency of packet classification. In his Ph.D. thesis [6], Gupta identifies *backward redundancy* and *forward redundancy*. In particular, a rule r_i in a rule set $R = \{r_1, r_2, \cdots, r_n\}$ is backward redundant if and only if R contains some other rule r_j such that (1) $j < i$ and (2) the definition region of r_j contains the definition region of r_i. A rule r_i is considered forward redundant if and only if R contains some other rule r_j such that (1) $j > i$, (2) the definition region of r_j contains the definition region of r_i, (3) r_i and r_j have the same decision, and (4) for any k such that $i < k < j$, either the definition regions of r_i and r_k do not overlap or r_i and r_k have the same decision. It is easy to verify that the backward redundant rules and the forward redundant rules in a rule set form a subset of the upward redundant rules and the downward redundant rules (which are identified by Liu and Gouda in [13]), respectively. In their earlier paper [5], Gouda and Liu design algorithms for removing redundant rules generated from a firewall decision diagram.

The objective of these previous work is to reduce the number of rules instead of reducing the number of TCAM entries needed to represent the given rule set. As we have seen in this paper, reducing the number of rules does not necessarily reduce the number of TCAM entries needed to represent the rule set, and vice versa. Actually, we propose adding rules as an effective technique for reducing the number of TCAM entries needed. Our simulation results based on random rule sets have demonstrated that our proposed scheme typically produces a semantically equivalent packet classifier that contains much more rules than the given rule set, but requires much less TCAM entries to represent.

Prior to this work, some other complementary techniques have also been proposed to reduce the size of TCAM needed. For example, range encoding techniques [14, 24, 12] aim to reduce the number of TCAM entries needed to represent a rule set by employing sophisticated schemes for encoding ranges specified in a rule set. Typically, range encoding techniques involve dividing an m-bit range into k blocks. Then, various encoding schemes can be applied on the blocks. Such inherent nature of these techniques makes them plagued by the following problems, which are not present in our scheme. First, modifications to the packet processor hardware are needed to interpret and perform the range encoding schemes. Second, some techniques (e.g. [14, 24]) exploit statistical features of the given rule set and hence are not considered general. If preferred, these complementary techniques can be applied on the rule set produced by our scheme to achieve further performance benefits.

6. CONCLUSIONS

TCAM as the *de facto* solution for packet classification suffers from multiplicative explosion in size due to range

specifications. Compressing the TCAM representation of packet classifiers without requiring hardware modification is a challenging problem of significant value. In this paper, we propose a set of practical techniques for defining semantically equivalent packet classifiers such that the amount of TCAM entries needed is much smaller. Thus, packet classification can be carried out using much smaller TCAMs without any hardware modification. Experiments conducted on thousands of real rule sets provided by a tier-1 ISP demonstrate that our proposed techniques reduce the number of TCAM entries needed by 46%, although these rule sets typically have only one range clause. Furthermore, experiments based on random rule sets with varying number of range clauses demonstrate that the power of our proposed techniques grows rapidly with the number of range clauses. These results clearly demonstrate the enormous potential of our proposed techniques in emerging applications.

7. REFERENCES

[1] F. Baboescu, S. Singh, and G. Varghese. Packet classification for core routers: is there an alternative to CAMs? In *IEEE INFOCOM*, 2003.

[2] F. Baboescu and G. Varghese. Scalable packet classification. In *ACM SIGCOMM*, 2001.

[3] A. Gallo. Meeting traffic demands with next-generation internet infrastructure. *Lightwave*, 18(5):118–123, May 2001.

[4] G. Gibson, F. Shafai, and J. Podaima. Content addressable memory storage device. United States Patent 6,044,005, March 2000.

[5] M. G. Gouda and A. X. Liu. Firewall design: Consistency, completeness and compactness. In *IEEE ICDCS*, 2004.

[6] P. Gupta. *Algorithms for Routing Lookups and Packet Classification*. Ph.d. thesis, Stanford University, 2000.

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

[8] P. Gupta and N. McKeown. Packet classification using hierarchical intelligent cuttings. In *HOTI*, 1999.

[9] R. A. Kempke and A. J. McAuley. Ternary CAM memory architecture and methodology. United States Patent 5,841,874, November 1998.

[10] M. E. Kounavis, A. Kumar, H. Vin, R. Yavatkar, and A. T. Campbell. Directions in packet classification for network processors. In *NP2 Workshop*, 2003.

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

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

[13] A. X. Liu and M. G. Gouda. Removing redundancy from packet classifiers. Technical Report TR-04-26, Department of Computer Sciences, The University of Texas at Austin, Austin, Texas, U.S.A., June 2004.

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

[15] C. Matsumoto. CAM vendors consider algorithmic altervatives. *EE Times*, May 2002.

[16] R. K. Montoye. Apparatus for storing "don't care" in a content addressable memory cell. United States Patent 5,319,590, June 1994.

[17] M. H. Overmars and A. F. van der Stappen. Range searching and point location among fat objects. *Journal of Algorithms*, 21(3):629–656, November 1996.

[18] L. Qiu, G. Varghese, and S. Suri. Fast firewall implemention for software and hardware based routers. In *IEEE ICNP*, 2001.

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

[20] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and scalable layer four switching. In *ACM SIGCOMM*, pages 191–202, September 1998.

[21] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Packet classification using tuple space search. In *ACM SIGCOMM*, 1999.

[22] D. E. Taylor. Survey & taxonomy of packet classification techniques. Technical Report WUCSE-2004-24, Department of Computer Science & Engineering, Washington University in Saint Louis, May 2004.

[23] D. E. Taylor and J. S. Turner. Scalable packet classification using distributed crossproducting of field labels. In *IEEE INFOCOM*, 2005.

[24] J. van Lunteren and T. Engbersen. Fast and scalable packet classification. *IEEE Journal on Selected Areas in Communications*, 21(4):560–571, 2003.

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

[26] F. Yu and R. H. Katz. Efficient multi-match packet classification with TCAM. In *Hot Interconnects*, 2004.