TCAM Razor: A Systematic Approach Towards Minimizing Packet Classifiers in TCAMs

Alex X. Liu, Chad R. Meiners, and Eric Torng

***Abstract*—Packet classification is the core mechanism that enables many networking services on the Internet such as firewall packet filtering and traffic accounting. Using ternary content addressable memories (TCAMs) to perform high-speed packet classification has become the *de facto* standard in industry. TCAMs classify packets in constant time by comparing a packet with all classification rules of ternary encoding in parallel. Despite their high speed, TCAMs suffer from the well-known range expansion problem. As packet classification rules usually have fields specified as ranges, converting such rules to TCAM-compatible rules may result in an explosive increase in the number of rules. This is not a problem if TCAMs have large capacities. Unfortunately, TCAMs have very limited capacity, and more rules mean more power consumption and more heat generation for TCAMs. Even worse, the number of rules in packet classifiers has been increasing rapidly with the growing number of services deployed on the Internet. In this paper, we consider the following problem: given a packet classifier, how can we generate another semantically equivalent packet classifier that requires the least number of TCAM entries? In this paper, we propose a systematic approach, the *TCAM Razor*, that is effective, efficient, and practical. In terms of effectiveness, TCAM Razor achieves a total compression ratio of 29.0%, which is significantly better than the previously published best result of 54%. In terms of efficiency, our TCAM Razor prototype runs in seconds, even for large packet classifiers. Finally, in terms of practicality, our TCAM Razor approach can be easily deployed as it does not require any modification to existing packet classification systems, unlike many previous range encoding schemes.**

***Index Terms*—Algorithm, packet classification, router design, ternary content addressable memory (TCAM) optimization.**

I. Introduction

PACKET classification, which is widely used on the Internet, is the core mechanism that enables routers to perform many networking services such as firewall packet filtering, virtual private networks (VPNs), network address translation (NAT), quality of service (QoS), load balancing, traffic accounting and monitoring, differentiated services (Diffserv), etc. The function of a packet classification system is to map each packet to a decision (i.e., action) according to a sequence (i.e., ordered list) of rules, which is called a packet classifier. Each rule in a classifier has a predicate over some packet header fields and a decision to be performed upon the packets that match the predicate. To resolve possible conflicts among rules in a classifier, the decision for each packet is the decision of the first (i.e., highest priority) rule that the packet matches. Table I shows an example classifier of two rules. The format of these rules is based upon the format used in access control lists (ACLs) on Cisco routers.

TABLE I
EXAMPLE PACKET CLASSIFIER

#	Source IP	Dest. IP	Source Port	Dest. Port	Protocol	Action
r_1	1.2.3.0/24	192.168.0.1	[1,65534]	[1,65534]	TCP	accept
r_2	*	*	*	*	*	discard

A. Motivation

To process the neverending supply of packets at wire speed, using ternary content addressable memories (TCAMs) to perform packet classification has become the *de facto* standard for high-speed routers on the Internet [12]. A TCAM is a memory chip where each entry can store a packet classification rule that is encoded in ternary format. Given a packet, the TCAM hardware can compare the packet with all stored rules in parallel and then return the decision of the first rule that the packet matches. Thus, it takes $O(1)$ time to find the decision for any given packet. In 2003, most packet classification devices shipped were TCAM-based [4]. More than 6 million TCAM devices were deployed worldwide in 2004 [4].

Despite their high speed, TCAMs have their own limitations with respect to packet classification. 1) *Range expansion:* TCAMs can only store rules that are encoded in ternary format. In a typical classification rule, source IP address, destination IP address, and protocol type are specified in prefix format, which can be directly stored in TCAMs, but source and destination port numbers are specified in ranges (i.e., integer intervals), which need to be converted to one or more prefixes before being stored in TCAMs. This can lead to a significant increase in the number of TCAM entries needed to encode a rule. For example, 30 prefixes are needed to represent the single range $[1, 65534]$, so $30 \times 30 = 900$ TCAM entries are required to represent the single rule r_1 in Table I. 2) *Low capacity:* TCAMs have limited capacity. The largest available TCAM chip has a capacity of 36 Mb [1], while 2- and 1-Mb chips are the most popular [4]. 3) *High power consumption and heat generation:* TCAM chips consume large amounts of power (30 times as

Manuscript received February 14, 2008; revised May 13, 2009; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor T. Wolf. First published October 30, 2009; current version published April 16, 2010. This work was supported in part by the National Science Foundation under Grants CNS-0716407, CNS-0916044, and CNS-0845513. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. A preliminary version of this paper was published in the *Proceedings of the 15th IEEE International Conference on Network Protocols* (ICNP), Beijing, China, October 2007 (pp. 226–275).

The authors are with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824 USA (e-mail: alexliu@cse.msu.edu; meinersc@cse.msu.edu; torng@cse.msu.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2009.2030188

TABLE II
TCAM RAZOR OUTPUT FOR THE EXAMPLE CLASSIFIER IN TABLE I

#	Source IP	Dest. IP	Source Port	Dest. Port	Protocol	Action
r_1	1.2.3.0/24	192.168.0.1	0	*	*	discard
r_2	1.2.3.0/24	192.168.0.1	65535	*	*	discard
r_3	1.2.3.0/24	192.168.0.1	*	0	*	discard
r_4	1.2.3.0/24	192.168.0.1	*	65535	*	discard
r_5	1.2.3.0/24	192.168.0.1	[0,65535]	[0,65535]	TCP	accept
r_6	*	*	*	*	*	discard

much power as a comparably sized SRAM chip given the same number of memory accesses [13]) and generate large amounts of heat. Power consumption of TCAMs increases linearly with the capacity of the TCAM [2] as well as the number of rules stored in the TCAM [2], [28]. Power consumption together with the consequent heat generation is a serious problem for core routers and other networking devices. 4) *Large board space occupation:* TCAMs occupy much more board space than SRAMs. A given capacity TCAM chip occupies six times (or more) board space than an equivalent capacity SRAM [13]. For networking devices such as routers, area efficiency of the circuit board is a critical issue. 5) *High hardware cost:* TCAM chips are *expensive*, costing hundreds of dollars even in large quantities. In modern routers, TCAM chips sometimes cost more than network processors [14].

B. Problem Statement

In this paper, we consider the following TCAM minimization problem: *given a classifier, how can we generate another semantically equivalent classifier that requires the least number of TCAM entries?* Two classifiers are (semantically) equivalent if and only if they have the same decision for every packet. For example, the two classifiers in Tables I and II are equivalent; however, the one in Table I requires 900 TCAM entries, and the one in Table II requires only six TCAM entries.

Solving this problem helps to address the limitations of TCAMs. As we reduce the number of TCAM entries required, we can use smaller TCAMs, which results in less board space and lower hardware cost. Furthermore, reducing the number of rules in a TCAM directly reduces power consumption and heat generation because the energy consumed by a TCAM grows linearly with the number of ternary rules it stores [2].

C. Summary and Limitations of Prior Art

All previous efforts on minimizing d-dimensional ($d > 2$) rule lists employ equivalent transformations at the *rule level*. That is, they only consider rewriting each *rule* in place. For example, all prior range encoding schemes [5], [6], [12], [19], [21], [22], [27], [29] try to minimize the effects of range expansion on each individual rule. *The fundamental weakness of this rule-level perspective is that it looks at the problem locally by examining one rule at a time*. This local, not global, perspective ignores many compression opportunities such as reordering the rule list. Furthermore, previous rule-level approaches ignore the decisions associated with each rule and thus the semantics of classifiers. Thus, when these approaches are given two rule lists that are syntactically different but semantically the same, they typically generate two different output lists. Therefore, the effectiveness of rule-level approaches partially depends on how the input classifier is specified. Two previous papers do consider classifier semantics when minimizing rule lists [3], [26], but these results only apply when $d = 1$ or 2, whereas real-life classifiers have $d = 4$ or 5 or more.

D. New Perspective, Challenges, and Our Approach

We perform equivalent transformation at the *list level*. We take a rule list as a whole and try to generate another small, but semantically equivalent, rule list. List-level transformation is more powerful than rule-level transformation because it incorporates the classifier semantics and considers more optimizations. However, it is more challenging due to the "curse of dimensionality." That is, these rule list minimization problems can be solved optimally in polynomial time for one-dimensional prefix and range rule lists [8], [18], [26], but no polynomial time algorithms are known even for their two-dimensional versions. In fact, Applegate *et al.* showed that the range rule list optimization problem is NP-hard for $d = 2$ [3]. The one-dimensional ternary rule list optimization problem is essentially the Boolean expression minimization problem, which is an NP-complete problem [5]. As we add dimensions, the problems become more and more difficult because of the complex interactions among multidimensional objects.

To cope with the complex rule interactions, we propose a novel approach, called *TCAM Razor*, to perform list-level equivalent transformation. We first decompose a multidimensional rule list optimization problem into multiple one-dimensional versions of the problem using decision diagrams. Second, we optimally solve each of the one-dimensional rule list optimization problems. Finally, we reconstruct a solution to the multidimensional problem from the multiple one-dimensional solutions, again using the structure of decision diagrams. This approach is highly *effective* because it harnesses the power of the optimal solutions for one-dimensional problems. On a set of real-life classifiers, Razor achieves a total compression ratio of 29.0%, which is significantly better than the previously published best result of 54%. This approach is highly *efficient* because a decision diagram can be constructed efficiently and each one-dimensional problem can be solved efficiently (and optimally). Running Razor on a classifier with thousands of rules takes only a few seconds. This approach is also highly *practical* because it can be easily deployed without any modification of existing classification systems. In comparison, many prior solutions require hardware and architecture modifications to existing classification systems, making their adoption by networking manufacturers and ISPs much harder. TCAM Razor works for classifiers with any number of distinct decisions. Its effectiveness gracefully degrades as the number of decisions increases.

We name our solution "TCAM Razor" following the principle of Occam's razor: "*Of two equivalent theories or explanations, all other things being equal, the simpler one is to be preferred.*" In our context, of all classifiers that are equivalent, the one with the least number of TCAM entries is preferred.

The rest of this paper proceeds as follows. We start by reviewing related work in Section II. In Section III, we formally define the TCAM Minimization Problem and related terms. In Section IV, we discuss the weighted one-dimensional TCAM minimization problem. In Section V, we give a solution to the multidimensional TCAM minimization problem. The optimization techniques are presented in Section VI. In Section VII, we show the experimental results on a collection of real-life classifiers. Finally, we give concluding remarks in Section VIII.

II. Related Work

Prior work on optimizing TCAM-based packet classification systems fall into three broad categories: TCAM modification, range reencoding, and classifier minimization.

1) TCAM Modification: The basic idea is to modify TCAM circuits for packet classification purposes. For example, Spitznagel *et al.* proposed adding comparators to better accommodate range matching [25]. While important, this direction leads to solutions that are hard to deploy due to high cost[12].

2) Range Encoding: The basic idea is to reencode ranges that appear in a classifier and store the reencoded rules in a TCAM. Packets also need to be reencoded online accordingly. Some encoding schemes have been proposed [5], [12], [19]–[23], [27]. While the TCAM circuit does not need to be modified to implement range encoding, the system hardware does need to be reconfigured to allow for preprocessing of packets. Nevertheless, these range encoding schemes are somewhat orthogonal to our work and combining our approaches with range encoding schemes could be interesting future work.

3) Classifier Minimization: The basic idea is to convert a given classifier to another semantically equivalent classifier that requires fewer TCAM entries. Our work, along with [3], [7], [8], [16], [17], and [26], falls into this category.

Three papers focus on one-dimensional and two-dimensional classifiers. Draves *et al.* proposed an optimal solution for one-dimensional classifiers in the context of minimizing routing tables in [8]. Subsequently, Suri *et al.* developed a dynamic programming formulation for two-dimensional prefix rule lists, but this formulation is not optimal even for $d=2$ and is extremely inefficient when extended to larger d [26]. Recently, Applegate *et al.* focused on the two-dimensional case and only considered strip rules where one field must contain the entire field domain. They developed clever insights that led them to find an optimal strip rule algorithm without being overly constrained by the input rule list syntax. Unfortunately, it is not obvious how to generalize their algorithm to more dimensions or other rule formats.

There are only two prior methods for minimizing classifiers with more than two dimensions: redundancy removal [16] [17] and rule trimming/expanding [7]. Removing redundant rules from classifiers obviously results in less TCAM space usage. TCAM Razor significantly outperforms redundancy removal because it finds many more opportunities for minimizing classifiers. In [7], Dong *et al.* observed that both expanding and trimming ranges so they correspond to prefixes boundaries can result in fewer TCAM entries. TCAM Razor outperforms the heuristics of Dong *et al.* because of two major reasons. First, although TCAM Razor and Dong *et al.*'s heuristics both process classifiers one dimension at a time, TCAM Razor is guaranteed to achieve optimal compression on that dimension, but Dong *et al.*'s heuristics are not. Specifically, TCAM Razor handles all the special cases that Dong *et al.* identify in a systematic fashion. Second, TCAM Razor reduces the influence of classifier syntax (i.e., how classifiers are specified) by converting the given classifier to its canonical representation (i.e., a reduced decision diagram). On the other hand, Dong *et al.* process rules specified in the original manner, looking at one rule at a time for optimization possibilities.

III. Formal Definitions

We now formally define the concepts of fields, packets, packet classifiers, and the TCAM Minimization Problem. A *field* F_i is a variable of finite length (i.e., of a finite number of bits). The domain of field F_i of b bits, denoted $D(F_i)$, is $[0, 2^b-1]$. A *packet* over the d fields $F_1, \ldots, F_d$ is a d-tuple $(p_1, \ldots, p_d)$, where each $p_i (1 \leq i \leq d)$ is an element of $D(F_i)$. Packet classifiers usually check the following five fields: source IP, destination IP, source port, destination port, and protocol type. The length of these fields are 32, 32, 16, 16, and 8, respectively. We use Σ to denote the set of all packets over fields $F_1, \ldots, F_d$. It follows that Σ is a finite set and $|\Sigma| = |D(F_1)| \times \cdots \times |D(F_d)|$, where $|\Sigma|$ denotes the number of elements in set Σ and $|D(F_i)|$ denotes the number of elements in set $D(F_i)$.

A *rule* has the form $\langle predicate \rangle \rightarrow \langle decision \rangle$. A $\langle predicate \rangle$ defines a set of packets over the fields F_1 through F_d, and is specified as $F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d$ where each S_i is a subset of $D(F_i)$ and is specified as either a prefix or a range. A *prefix* $\{0,1\}^k\{*\}^{b-k}$ with k leading 0 s or 1 s for a packet field of length b denotes the range $[\{0,1\}^k\{0\}^{b-k}, \{0,1\}^k\{1\}^{b-k}]$. For example, prefix 01** denotes the range $[0100, 0111]$. A rule $F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d \rightarrow \langle decision \rangle$ is a *prefix rule* if and only if each S_i is represented as a prefix.

When using a TCAM to implement a classifier, we typically require that all rules be prefix rules. However, in a typical classifier rule, some fields such as source and destination port numbers are represented as ranges rather than prefixes. This leads to *range expansion*, the process of converting a rule that may have fields represented as ranges into one or more prefix rules. In range expansion, each field of a rule is first expanded separately. The goal is to find a minimum set of prefixes such that the union of the prefixes corresponds to the range. For example, if one 3-bit field of a rule is the range $[1, 6]$, a corresponding minimum set of prefixes would be 001, 01*, 10*, 110. The worst-case range expansion of a b-bit range results in a set containing $2b-2$ prefixes [11]. The next step is to compute the cross product of each set of prefixes for each field, resulting in a potentially large number of prefix rules. In Section I, the range expansion of rule r_1 in Table I resulted in $30 \times 30 = 900$ prefix rules.

A packet $(p_1, \ldots, p_d)$ *matches* a predicate $F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d$ and the corresponding rule if and only if the condition $p_1 \in S_1 \wedge \cdots \wedge p_d \in S_d$ holds. We use α to denote the set of possible values that $\langle decision \rangle$ can be. For firewalls, typical elements of α include accept, discard, accept with logging, and discard with logging.

A sequence of rules $\langle r_1, \ldots, r_n \rangle$ is *complete* if and only if for any packet p, there is at least one rule in the sequence that p matches. To ensure that a sequence of rules is complete and thus is a classifier, the predicate of the last rule is usually specified as $F_1 \in D(F_1) \wedge \cdots F_d \in \wedge D(F_d)$. A *packet classifier* f is a

TABLE III
NOTATION

Symbol	Description
F_i	field i
$D(F_i)$	domain of F_i
d	# of dimensions
b	# of bits
Σ	set of all packets
f	a classifier
p	packet
$f(p)$	decision of f on p
$\{f\}$	set of all classifiers equivalent to f
$\mathcal{P}$	prefix
$f_{\mathcal{P}}$	classifier equivalent to f on $\mathcal{P}$

Symbol	Description
$Decision(r_i)$	decision of rule r_i
d_i	decision i
w_{d_i}	cost of decision i
$Cost(f)$	cost of classifier f
$C(f_{\mathcal{P}})$	minimum cost of classifiers equivalent to $f_{\mathcal{P}}$
$C(f_{\mathcal{P}}^{d_i})$	minimum cost of classifiers that are equivalent to $f_{\mathcal{P}}$ and whose last rule's decision is d_i

sequence of rules that is complete. The size of f, denoted $|f|$, is the number of rules in f. A classifier f is a *prefix classifier* if and only if every rule in f is a prefix rule.

Two rules in a classifier may overlap; that is, there exists at least one packet that matches both rules. Furthermore, two rules in a classifier may conflict; that is, the two rules not only overlap, but also have different decisions. Classifiers typically resolve conflicts by employing a first-match resolution strategy where the decision for a packet p is the decision of the first (i.e., highest priority) rule that p matches in f. The decision that classifier f makes for packet p is denoted $f(p)$.

We can think of a classifier f as defining a many-to-one mapping function from Σ to α, where Σ denotes the set of all possible packets and α denotes the set of all possible decisions. Two classifiers f_1 and f_2 are *equivalent*, denoted $f_1 \equiv f_2$, if and only if they define the same mapping function from Σ to α; that is, for any packet $p \in \Sigma$, we have $f_1(p) = f_2(p)$. For any classifier f, we use $\{f\}$ to denote the set of classifiers that are equivalent to f. Now, we are ready to define the *TCAM Minimization Problem*.

Definition 3.1 (TCAM Minimization Problem): Given a classifier f_1, find a prefix classifier $f_2 \in \{f_1\}$ such that for any prefix classifier $f \in \{f_1\}$, the condition $|f_2| \leq |f|$ holds.

Table III lists the notations used throughout this paper.

IV. ONE-DIMENSIONAL TCAM MINIMIZATION

We first consider the special problem of *weighted* one-dimensional TCAM minimization, whose solution is used in the next section as a building block for multi-dimensional TCAM minimization. Given a one-dimensional packet classifier f of n prefix rules $\langle r_1, r_2, \ldots, r_n \rangle$, where $\{Decision(r_1), Decision(r_2), \ldots, Decision(r_n)\} = \{d_1, d_2, \ldots, d_z\}$ and each decision d_i is associated with a cost w_{d_i} (for $1 \leq i \leq z$), we define the cost of packet classifier f as follows:

$$Cost(f) = \sum_{i=1}^{n} w_{\text{Decision}(r_i)}.$$

Based upon the above definition, the problem of weighted one-dimensional TCAM minimization is stated as follows.

Definition 4.1 (Weighted One-Dimensional TCAM Minimization Problem): Given a one-dimensional packet classifier f_1 where each decision is associated with a cost, find a prefix packet classifier $f_2 \in \{f_1\}$ such that for any prefix packet classifier $f \in \{f_1\}$, the condition $Cost(f_2) \leq Cost(f)$ holds.

The problem of one-dimensional TCAM minimization (with uniform cost) has been studied in [8] and [26] in the context of compressing routing tables. In this paper, we extend the dynamic programming solution in [26] to solve the weighted one-dimensional TCAM minimization problem based on the following three observations:

1) For any one-dimensional packet classifier f on $\{*\}^b$, we can always change the predicate of the last rule to be $\{*\}^b$ without changing the semantics of the packet classifier. This follows from the completeness property of packet classifiers.
2) Consider any one-dimensional packet classifier f on $\{*\}^b$. Let f' be f appended with rule $\{*\}^b \rightarrow d$, where d can be any decision. The observation is that $f \equiv f'$. This is because the new rule is redundant in f' since f must be complete. A rule in a packet classifier is *redundant* if and only if removing the rule from the packet classifier does not change the semantics of the packet classifier.
3) For any prefix $\mathcal{P} \in \{0,1\}^k\{*\}^{b-k} (0 \leq k \leq b)$, one and only one of the following conditions holds: a) $\mathcal{P} \in \{0,1\}^k 0 \{*\}^{b-k-1}$; b) $\mathcal{P} \in \{0,1\}^k 1 \{*\}^{b-k-1}$; c) $\mathcal{P} = \{0,1\}^k\{*\}^{b-k}$.

This property allows us to divide a problem on $\{0,1\}^k\{*\}^{b-k}$ into two subproblems: one on $\{0,1\}^k 0\{*\}^{b-k-1}$ and the other one on $\{0,1\}^k 1\{*\}^{b-k-1}$. This divide-and-conquer strategy can be applied recursively.

Based on the above three observations, we formulate an optimal dynamic programming solution to the weighted one-dimensional TCAM minimization problem.

Let $\mathcal{P}$ denote a prefix $\{0,1\}^k\{*\}^{b-k}$. We use $\underline{\mathcal{P}}$ to denote the prefix $\{0,1\}^k 0\{*\}^{b-k-1}$, and $\bar{\mathcal{P}}$ to denote the prefix $\{0,1\}^k 1\{*\}^{b-k-1}$. Given a one-dimensional packet classifier f on $\{*\}^b$, we use $f_{\mathcal{P}}$ to denote a packet classifier on $\mathcal{P}$ such that for any $x \in \mathcal{P}$, $f_{\mathcal{P}}(x) = f(x)$, and we use $f_{\mathcal{P}}^d$ to denote a similar packet classifier on $\mathcal{P}$ with the additional restriction that the final decision is d. We use $C(f_{\mathcal{P}})$ to denote the minimum cost of a packet classifier t that is equivalent to $f_{\mathcal{P}}$, and we use $C(f_{\mathcal{P}}^d)$ to denote the minimum cost of a packet classifier t' that is equivalent to $f_{\mathcal{P}}$ and the decision of the last rule in t' is d. Given a one-dimensional packet classifier f on $\{*\}^b$ and a prefix $\mathcal{P}$ where $\mathcal{P} \subseteq \{*\}^b$, *$f$ is consistent on $\mathcal{P}$* if and only if $\forall x, y \in \mathcal{P}, f(x) = f(y)$.

Our dynamic programming solution to the weighted one-dimensional TCAM minimization problem is based on the following theorem, the proof of which shows how to divide a problem into subproblems and how to combine solutions to subproblems into a solution to the original problem.

Theorem 4.1: Given a one-dimensional packet classifier f on $\{*\}^b$, a prefix $\mathcal{P}$ where $\mathcal{P} \subseteq \{*\}^b$, the set of all possible decisions $\{d_1, d_2, \ldots, d_z\}$ where each decision d_i has a cost $w_{d_i} (1 \leq i \leq z)$, we have that

$$C(f_{\mathcal{P}}) = \min_{i=1}^{z} C(f_{\mathcal{P}}^{d_i})$$

where each $C(f_{\mathcal{P}}^{d_i})$ is calculated as follows:

1) If f is consistent on $\mathcal{P}$, then

$$C(f^{d_i}_{\mathcal{P}}) = \begin{cases} w_{f(x)}, & \text{if } f(x) = d_i \ \forall x \in \mathcal{P} \\ w_{f(x)} + w_{d_i}, & \text{if } f(x) \neq d_i \ \forall x \in \mathcal{P}. \end{cases}$$

2) If f is not consistent on $\mathcal{P}$, then

$$C(f^{d_i}_{\mathcal{P}}) = \min \begin{cases} C(f^{d_1}_{\underline{\mathcal{P}}}) + C(f^{d_1}_{\bar{\mathcal{P}}}) - w_{d_1} + w_{d_i}, \ldots \\ C(f^{d_{i-1}}_{\underline{\mathcal{P}}}) + C(f^{d_{i-1}}_{\bar{\mathcal{P}}}) - w_{d_{i-1}} + w_{d_i} \\ C(f^{d_i}_{\underline{\mathcal{P}}}) + C(f^{d_i}_{\bar{\mathcal{P}}}) - w_{d_i} \\ C(f^{d_{i+1}}_{\underline{\mathcal{P}}}) + C(f^{d_{i+1}}_{\bar{\mathcal{P}}}) - w_{d_{i+1}} + w_{d_i}, \ldots \\ C(f^{d_z}_{\underline{\mathcal{P}}}) + C(f^{d_z}_{\bar{\mathcal{P}}}) - w_{d_a} + w_{d_i}. \end{cases}$$

Proof:

1) The base case of the recursion is when f is consistent on $\mathcal{P}$. In this case, the minimum-cost prefix packet classifier in $\{f_{\mathcal{P}}\}$ is clearly $\langle \mathcal{P} \rightarrow f(x) \rangle$, and the cost of this packet classifier is $w_{f(x)}$. Furthermore, for $d_i \neq f(x)$, the minimum-cost prefix packet classifier in $\{f_{\mathcal{P}}\}$ with decision d_i in the last rule is $\langle \mathcal{P} \rightarrow f(x), \mathcal{P} \rightarrow d_i \rangle$ where the second rule is redundant. The cost of this packet classifier is $w_{f(x)} + w_{d_i}$.
2) If f is not consistent on $\mathcal{P}$, we divide $\mathcal{P}$ into $\underline{\mathcal{P}}$ and $\bar{\mathcal{P}}$. The crucial observation is that an optimal solution f^* to $\{f_{\mathcal{P}}\}$ is essentially an optimal solution f_1 to the subproblem of minimizing $f_{\underline{\mathcal{P}}}$ appended with an optimal solution f_2 to the subproblem of minimizing $f_{\bar{\mathcal{P}}}$. The only interaction that can occur between f_1 and f_2 is if their final rules have the same decision, in which case both final rules can be replaced with one final rule covering all of $\mathcal{P}$ with the same decision. Let d_x be the decision of the last rule in f_1 and d_y be the decision of the last rule in f_2. Then, we can compose f^* whose last rule has decision d_i from f_1 and f_2 based on the following cases:
   A) $d_x = d_y = d_i$: In this case, f can be constructed by listing all the rules in f_1 except the last rule, followed by all the rules in f_2 except the last rule, and then the last rule $\mathcal{P} \rightarrow d_i$. Thus, $Cost(f) = Cost(f_1) + Cost(f_2) - w_{d_i}$.
   B) $d_x = d_y \neq d_i$: In this case, f can be constructed by listing all the rules in f_1 except the last rule, followed by all the rules in f_2 except the last rule, then rule $\mathcal{P} \rightarrow d_x$, and finally rule $\mathcal{P} \rightarrow d_i$ Thus, $Cost(f) = Cost(f_1) + Cost(f_2) - w_{d_x} + w_{d_i}$.
   C) $d_x \neq d_y, d_x = d_i, d_y \neq d_i$: We do not need to consider this case because $C(f^{d_i}_{\underline{\mathcal{P}}}) + C(f^{d_y}_{\bar{\mathcal{P}}}) = C(f^{d_i}_{\underline{\mathcal{P}}}) + (C(f^{d_y}_{\bar{\mathcal{P}}}) + w_{d_i}) - w_{d_i} \geq C(f^{d_i}_{\underline{\mathcal{P}}}) + C(f^{d_i}_{\bar{\mathcal{P}}}) - w_{d_i}$.
   D) $d_x \neq d_y, d_x \neq d_i, d_y = d_i$: Similarly, we do not need to consider this case.
   E) $d_x \neq d_y, d_x \neq d_i, d_y \neq d_i$: Similarly, we do not need to consider this case. □

Fig. 1 illustrates a one-dimensional TCAM minimization input instance where $f(00) = f(10) = \text{accept}$ and $f(01) = f(11) = \text{discard}$. Graphically, the black bar denotes decision accept and the white bar denotes decision discard.

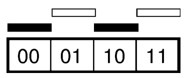


Fig. 1. A one-dimensional TCAM minimization instance.

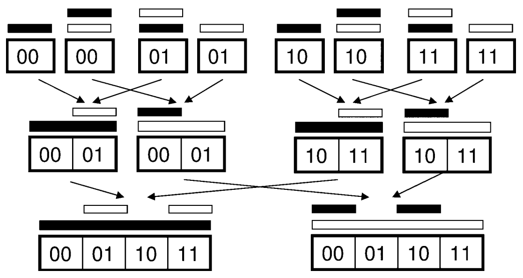


Fig. 2. Illustration of dynamic programming algorithm.

Fig. 2 illustrates how we compute an optimal solution for this instance using dynamic programming. In this instance, we have four distinct domain values (00, 01, 10, 11) and two distinct decisions accept and reject for eight base cases that are listed across the top of Fig. 2: f^{accept}_{00}, f^{discard}_{00}, f^{accept}_{01}, f^{discard}_{01}, f^{accept}_{10}, f^{discard}_{10}, f^{accept}_{11}, and f^{discard}_{11}. Here, f^{accept}_{00} denotes a solution for the domain value 00 with a default rule of accept. Four base cases including f^{accept}_{00} require one rule because the decision for the domain value matches the decision of the default rule. The other four base cases including f^{discard}_{00} require two rules: the first rule to set the correct decision for the given domain value and the second default rule with a different decision. The middle and bottom of the figure show how to use subsolutions to compute f^{accept}_{0*}, f^{discard}_{0*}, f^{accept}_{1*}, f^{discard}_{1*}, f^{accept}_{**}, and f^{discard}_{**}. For example, to find the solution for f^{accept}_{0*}, we must consider two cases: compose the solutions from f^{accept}_{00} and f^{accept}_{01} or compose the solutions from f^{discard}_{00} and f^{discard}_{01}. In this example, the first choice costs two rules and the second choice costs three rules (where the last rule is a default accept rule), so the lower cost first choice is selected. This process repeats for each subproblem.

V. Multidimensional TCAM Minimization: The Basics

In this section, we present TCAM Razor, our algorithm for minimizing multidimensional prefix packet classifiers. A key idea behind TCAM Razor is processing one dimension at a time using the weighted one-dimensional TCAM minimization algorithm in Section IV to greedily identify a local minimum for the current dimension. Although TCAM Razor is not guaranteed to achieve a global minimum across all dimensions, it does significantly reduce the number of prefix rules in real-life packet classifiers.

A. Conversion to Firewall Decision Diagrams

We represent classifiers using firewall decision diagrams, a special kind of decision tree [10]. A *firewall decision diagram* (FDD) with a decision set DS and over fields $F_1, \ldots, F_d$ is an acyclic and directed graph that has the following five properties: 1) There is exactly one node that has no incoming edges. This node is called the *root*. The nodes that have no outgoing edges

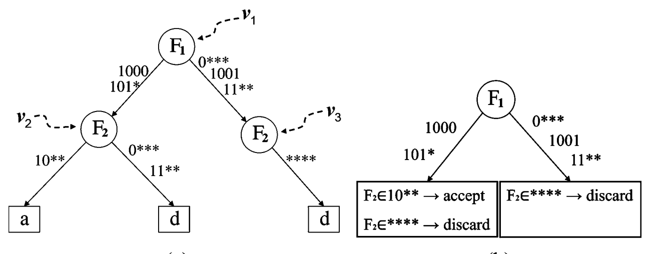


Fig. 3. An FDD and its "virtual" one-dimensional classifier.

are called *terminal* nodes. 2) Each node v has a label, denoted $F(v)$, such that

$$F(v) \in \begin{cases} \{F_1, \ldots, F_d\}, & \text{if } v \text{ is a nonterminal node} \\ DS, & \text{if } v \text{ is a terminal node.} \end{cases}$$

3) Each edge $e : u \rightarrow v$ is labeled with a nonempty set of integers, denoted $I(e)$, where $I(e)$ is a subset of the domain of u's label (i.e., $I(e) \subseteq D(F(u))$). 4) A directed path from the root to a terminal node is called a *decision path*. No two nodes on a decision path have the same label. 5) The set of all outgoing edges of a node v, denoted $E(v)$, satisfies the following two conditions: a) *Consistency*: $I(e) \cap I(e') = \emptyset$ for any two distinct edges e and e' in $E(v)$. b) *Completeness*: $\bigcup_{e \in E(v)} I(e) = D(F(v))$.

Fig. 3(a) shows an example FDD over two fields F_1, F_2, where the domain of each field is $[0, 15]$. Note that in labeling terminal nodes, we use letter "a" as a shorthand for "accept" and letter "d" as a shorthand for "discard."

Given a classifier f_1, we can construct an equivalent FDD f_2 using the FDD construction algorithm in [15]. An improved FDD construction algorithm is presented in Section VI.

B. Multidimensional TCAM Minimization

We start the discussion of our greedy solution by examining the FDD in Fig. 3(a). We first look at the subgraph rooted at node v_2. This subgraph can be seen as representing a one-dimension packet classifier over field F_2. We can use the weighted one-dimensional TCAM minimization algorithm in Section IV to minimize the number of prefix rules for this one-dimensional packet classifier. The algorithm takes the following three prefixes as input:

$$\begin{aligned} &10{*}{*} \quad \text{(with decision accept and cost 1)} \\ &0{*}{*}{*} \quad \text{(with decision accept and cost 1)} \\ &11{*}{*} \quad \text{(with decision accept and cost 1).} \end{aligned}$$

The one-dimensional TCAM minimization algorithm will produce a minimum (one-dimensional) packet classifier of two rules as shown in Table IV(a). Similarly, from the subgraph rooted at node v_3, we can get a minimum classifier of one rule as shown in Table IV(b).

Next, we look at the root v_1. As shown in Fig. 3(b), we view the subgraph rooted at v_2 as a decision with a multiplication factor or cost of 2 and the subgraph rooted at v_3 as another decision with a cost of 1. The graph rooted at v_1 can be thought of as

TABLE IV
MINIMUM PACKET CLASSIFIERS FOR (a) v_2 AND (b) v_3

#	F_1	Decision
1	10**	accept
2	****	discard
minimum packet classifier for v_2		

(a)

#	F_1	Decision
1	****	discard
minimum packet classifier for v_3		

(b)

TABLE V
(a) MINIMUM 1-D CLASSIFIER FOR v_1 AND (b) THE FINAL 2-D CLASSIFIER

#	F_1	Decision
1	1001	go to node v_3
2	10**	go to node v_2
3	****	go to node v_3
minimum 1-D classifier for v_1		

(a)

#	F_1	F_2	Decision
1	1001	****	discard
2	10**	10**	accept
3	10**	****	discard
4	****	****	discard
final 2-D classifier			

(b)

a "virtual" one-dimensional classifier over field F_1, where each child has a multiplicative cost. Thus, we can use the one-dimensional TCAM minimization algorithm in Section IV to minimize the number of rules for this "virtual" one-dimensional classifier. The algorithm takes the following five prefix rules and the associated decision costs as input:

$$\begin{aligned} &1000 \quad \text{(with decision } v_2 \text{ and cost 2)} \\ &101{*} \quad \text{(with decision } v_2 \text{ and cost 2)} \\ &0{*}{*}{*} \quad \text{(with decision } v_3 \text{ and cost 1)} \\ &1001 \quad \text{(with decision } v_3 \text{ and cost 1)} \\ &11{*}{*} \quad \text{(with decision } v_3 \text{ and cost 1).} \end{aligned}$$

The weighted one-dimensional TCAM minimization algorithm will produce the "virtual" one-dimensional classifier of three rules as shown in Table V(a). Combining the "virtual" 1-D classifier in Table V(a) and the two 1-D classifiers in Table IV, we get a 2-D classifier of 4 rules as shown in Table V(b).

C. Removing Redundant Rules

Next, we observe that rule r_3 in the classifier in Table V(b) is redundant. If we remove rule r_3, all the packets that used to be resolved by r_3 (that is, all the packets that match r_3 but do not match r_1 and r_2) are now resolved by rule r_4, and r_4 has the same decision as r_3. Therefore, removing rule r_3 does not change the semantics of the packet classifier. Redundant rules in a packet classifier can be removed using the algorithms in [16] and [17]. Finally, after removing redundant rules, we get a packet classifier of three rules from the FDD in Fig. 3(a).

D. The Algorithm

To summarize, TCAM Razor, our multidimensional TCAM minimization algorithm, consists of the following four steps:

Step 1: Convert the given classifier to an equivalent FDD.
Step 2: Use the FDD reduction algorithm described in the next section to reduce the size of the FDD. This step will be explained in more detail in the next section.
Step 3: Generate a classifier from the FDD in the following bottom-up fashion. For every terminal node, assign

a cost of 1. For a nonterminal node v with z outgoing edges $\{e_1, \ldots, e_z\}$, formulate a one-dimensional TCAM minimization problem as follows. For every prefix $\mathcal{P}$ in the label of edge e_j, $(1 \le j \le z)$, we set the decision of $\mathcal{P}$ to be j and the cost of $\mathcal{P}$ to be the cost of the node that edge e_j points to. For node v, we use the weighted one-dimensional TCAM minimization algorithm in Section IV to compute a one-dimensional prefix classifier with the minimum cost. We then assign this minimum cost to the cost of node v. After the root node is processed, generate a classifier using the prefixes computed at each node in a depth first traversal of the FDD. The cost of the root indicates the total number of prefix rules in the resulting classifier.

Step 4: Remove all redundant rules from the resulting classifier.

E. TCAM Updates

In most applications, such as router ACLs and firewalls, the rule sets are relatively static. Therefore, we propose using the bank mechanism in TCAMs to handle rule list updates. TCAMs are commonly configured into a series of row banks. Each bank can be individually enabled or disabled; entries from disabled banks are not examined in TCAM searches. We propose storing the compressed classifier before update in the active banks and the one after update in the disabled banks. Once the writing is finished, we activate the banks containing the newly compressed classifier and deactivate the banks containing the old one.

In some applications, there may be more frequent updates of the rule set. In such applications, there typically is a small static set of rules at the top of the classifier followed by a large dynamic set of rules. Furthermore, updates are typically the insertion of new rules to the top of the dynamic set of rules or the deletion of recently added dynamic rules.[1] We can support this update pattern by storing the small static set of rules at the top of the TCAM and then running TCAM Razor on the dynamic set of rules and storing this compressed set of rules in the bottom of the TCAM. New dynamic rules are appended to the top of the compressed set of rules in a bottom-up fashion. This update scheme is correct because adding rules to the top of the compressed set of rules is functionally equivalent to adding rules to the top of the original set of rules. Thus, TCAM Razor only needs to run when the TCAM is almost full rather than when each new rule is added. Note, we may not include specific dynamic rules when running TCAM Razor if they are likely to be deleted in the near future. Instead, we run TCAM Razor on the remainder of the dynamic rules and retain these likely-to-be-deleted rules at the front of the compressed set of rules.

VI. Multidimensional TCAM Minimization: The Optimization Techniques

In this section, we discuss the following two optimization techniques that we implemented to reduce the running time and memory usage of TCAM Razor: lazy copying in FDD construction and hashing in FDD reduction.

[1]Learned from private conversation with a large ISP

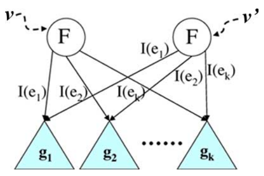


Fig. 4. Lazy copying of subgraphs.

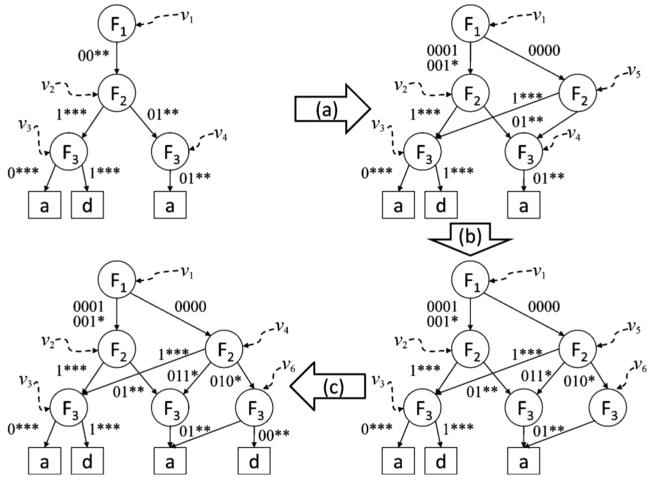


Fig. 5. Example of lazy copying.

A. Lazy Copying in FDD Construction

The FDD construction algorithm in [15] performs deep copying of subgraphs when splitting edges. This is inefficient in terms of both running time and memory usage. In TCAM Razor, we use the technique of lazy copying, which is explained as follows. Consider the subgraph (of an FDD) in Fig. 4. The root of this subgraph is v, and v has k outgoing edges $e_1, e_2, \ldots, e_k$, which point to the subgraphs $g_1, g_2, \ldots, g_k$, respectively. When we need to make another copy of this subgraph, instead of making a deep copy of the whole subgraph, we only make another copy of the root of the subgraph. Let v' denote the new node. Node v' has the same label as v and also has k outgoing edges $e'_1, e'_2, \ldots, e'_k$, where each e'_i has the same label $I(e_i)$ as e_i, and also points to the same subgraph g_i that e_i points to.

Each time we need to modify a node v, we first need to check its in-degree (i.e., the number of edges that point to v): If its in-degree is 1, then we can directly modify v; if its in-degree is greater than 1, then we need to first make a lazy copy of the subgraph rooted at v, and then modify the new node v'. To the outside, lazy copying looks like deep copying, but it reduces unnecessary copying of subgraphs (and promotes the sharing of common subgraphs) in the constructed FDD as much as possible.

Fig. 5 shows the process of appending rule $(F_1 \in 0000) \wedge (F_2 \in 010*) \wedge (F_3 \in 0***) \rightarrow d$ to node v_1 of the partial FDD in the upper-left side of the figure. A partial FDD is a diagram that has all the properties of an FDD except the completeness property.

In step (a), we split the single edge leaving v_1 into two edges, where v_5 is a shallow copy of v_2. In step (b), we further split the edge labeled $01**$ into two edges, where v_6 is a shallow copy of v_4. In step (c), we add the edge labeled $00**$ to v_6.

```
Algorithm 1: Lazy Copying Based FDD Construction
  Input: A packet classifier $f$ of a sequence of rules $\langle r_1, \cdots, r_n\rangle$
  Output: An FDD $f'$ such that $f$ and $f'$ are equivalent
1  build a decision path with root $v$ from rule $r_1$;
2  for $i := 2$ to $n$ do APPEND($v$, $r_i$);

3  APPEND($v$, $(F_m \in S_m) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle dec\rangle$ )
   /*$F(v) = F_m$ and $E(v) = \{e_1, \cdots, e_k\}$*/
4  if $(S_m - (I(e_1) \cup \cdots \cup I(e_k))) \neq \emptyset$ then
5      add an outgoing edge $e_{k+1}$ with label
       $S_m - (I(e_1) \cup \cdots \cup I(e_k))$ to $v$;
6      build a decision path from rule
       $(F_{m+1} \in S_{m+1}) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle dec\rangle$, and make
       $e_{k+1}$ point to the first node in this path;
7  if $m < d$ then
8      for $j := 1$ to $k$ do
9          if $I(e_j) \subseteq S_m$ then
10             if $Indegree(Target(e_j)) > 1$ then
11                 (1) create a new node $v'$ labeled the same as
                   $Target(e_j)$;
12                 (2) let $e_j$ point to $v'$;
13                 /*suppose $Target(e_j)$ has $h$ outgoing edges
                   $\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_h$*/
14                 (3) create $h$ new outgoing edges
                   $\varepsilon'_1, \varepsilon'_2, \cdots, \varepsilon'_h$ for $v'$, where each new edge $\varepsilon'_t$
                   point to $Target(\varepsilon_t)$ for $1 \leq t \leq h$;
15             APPEND($Target(e_j)$,
               $(F_{m+1} \in S_{m+1}) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle dec\rangle$);
16         else if $I(e_j) \cap S_m \neq \emptyset$ then
17             (1) create a new node $v'$ labeled the same as
               $Target(e_j)$;
18             (2) create a new edge $e$ from $v$ to $v'$ with label
               $I(e_j) \cap S_m$;
19             (3) replace the label of $e_j$ by $I(e_j) - S_m$;
20             /*suppose $Target(e_j)$ has $h$ outgoing edges
               $\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_h$*/
21             (4) create $h$ new outgoing edges $\varepsilon'_1, \varepsilon'_2, \cdots, \varepsilon'_h$ for
               $v'$, where each new edge $\varepsilon'_t$ point to $Target(\varepsilon_t)$
               for $1 \leq t \leq h$;
22             (5) APPEND( $Target(e_j)$,
               $(F_{m+1} \in S_{m+1}) \wedge \cdots \wedge (F_d \in S_d) \rightarrow \langle dec\rangle$ );
```

The pseudocode for the lazy-copying-based FDD construction algorithm is in Algorithm 1.

B. Hashing in FDD Reduction

To further reduce the number of rules generated by our algorithm, after we convert a packet classifier to an equivalent FDD, we need to reduce the size of the FDD. An FDD is *reduced* if and only if it satisfies the following three conditions: 1) no two nodes are isomorphic; 2) no two nodes have more than one edge between them; 3) no node has only one outgoing edge. Two nodes v and v' in an FDD are *isomorphic* if and only if v and v' satisfy one of the following two conditions: 1) both v and v' are terminal nodes with identical labels; 2) both v and v' are nonterminal nodes, and there is a one-to-one correspondence between the outgoing edges of v and the outgoing edges of v' such that every pair of corresponding edges have identical labels and they both point to the same node.

We next show an example where FDD reduction helps to reduce the number of prefix rules generated from an FDD. Consider the two equivalent FDDs in Fig. 6, where (a) is nonreduced and (b) is reduced. If we run our multidimensional TCAM minimization algorithm on the two FDDs, we will produce four prefix rules as shown in Table VI(a) and two prefix rules as shown in Table VI(b), respectively.

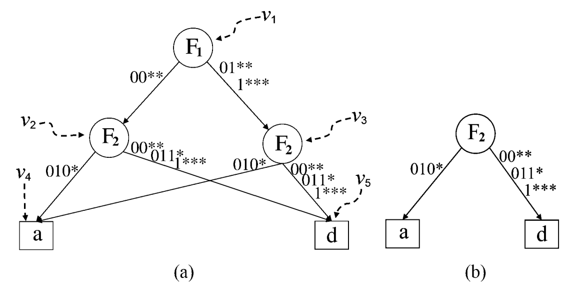


Fig. 6. Example of FDD reduction: (a) before and (b) after FDD reduction.

TABLE VI
RULES FROM THE FDDS IN (a) FIG. 6(a) AND (b) FIG. 6(b)

#	F_1	F_2	Decision
1	00**	010*	accept
2	00**	****	discard
3	****	010*	accept
4	****	****	discard
Rules generated from Fig. 6(a)			

(a)

#	F_1	F_2	Decision
1	****	010*	accept
2	****	****	discard
Rules generated from Fig. 6(b)			

(b)

A brute-force deep comparison algorithm for FDD reduction was proposed in [10]. In TCAM Razor, we use a more efficient FDD reduction algorithm that processes the nodes level by level from the terminal nodes to the root node using signatures to speed up comparisons. This algorithm works as follows.

Starting from the bottom level, at each level, we compute a signature for each node at that level. For a terminal node v, set v's signature to be its label. For a nonterminal node v, suppose v has k children $v_1, v_2, \ldots, v_k$, in increasing order of signature ($Sig(v_i) < Sig(v_{i+1})$ for $1 \leq i \leq k-1$), and the edge between v and its child v_i is labeled with E_i, a sequence of nonoverlapping prefixes in increasing order. Set the signature of node v as $Sig(v) = h(Sig(v_1), E_1, \ldots, Sig(v_k), E_k)$, where h is a one-way and collision-resistant hash function such as MD5 [24] and SHA-1 [9]. For any such hash function h, given two different input x and y, the probability of $h(x) = h(y)$ is extremely small.

After we have assigned signatures to all nodes at a given level, we search for isomorphic subgraphs as follows. For every pair of nodes v_i and $v_j (1 \leq i \neq j \leq k)$ at this level, if $Sig(v_i) \neq Sig(v_j)$, then we can conclude that v_i and v_j are not isomorphic; otherwise, we explicitly determine if v_i and v_j are isomorphic. If v_i and v_j are isomorphic, we delete node v_j and its outgoing edges and redirect all the edges that point to v_j to point to v_i. Furthermore, we eliminate double edges between node v_i and its parents. For example, the signatures of the nonroot nodes in Fig. 6(a) are computed as follows:

$$\begin{aligned} Sig(v_4) &= a \\ Sig(v_5) &= d \\ Sig(v_2) &= h(Sig(v_4), 010*, Sig(v_5), 00**, 011*, 1***) \\ Sig(v_3) &= h(Sig(v_4), 010*, Sig(v_5), 00**, 011*, 1***). \end{aligned}$$

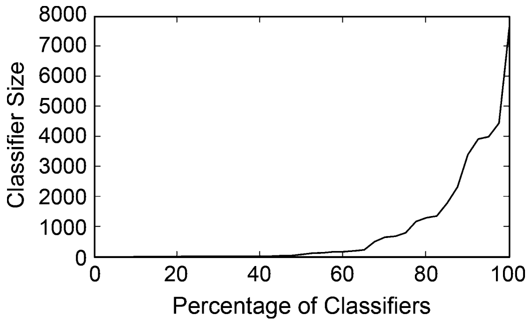


Fig. 7. Classifier sizes of RL.

At the end, for any nonterminal node v, if v has only one outgoing edge, we remove v and redirect the incoming edge of v to v's single child. As this step does not affect the number of rules generated from the FDD, we can skip it in practice.

VII. Experimental Results

We now evaluate TCAM Razor's effectiveness and efficiency.

A. Effectiveness

1) Methodology: We first define the metrics for measuring the effectiveness of TCAM Razor. Note that in cases where TCAM Razor cannot produce smaller classifiers than redundancy removal alone, TCAM Razor will return the classifier produced by redundancy removal. Let f denote a classifier, S denote a set of classifiers, and A denote a classifier minimization algorithm. We let $|f|$ denote the number of rules in f, $A(f)$ denote the prefix classifier produced by applying A on f, and $Direct(f)$ denote the prefix classifier produced by applying direct range expansion on f. The *compression ratio* of A on f is defined as $\frac{|A(f)|}{|Direct(f)|}$, and the *expansion ratio* of A on f is defined as $\frac{|A(f)|}{|f|}$. We define the following four metrics for assessing the effectiveness of A on a set of classifiers S. The *average compression ratio* of A over $S = \frac{\Sigma_{f\in S}\frac{|A(f)|}{|Direct(f)|}}{|S|}$. The *total compression ratio* of A over $S = \frac{\Sigma_{f\in S}|A(f)|}{\Sigma_{f\in S}|Direct(f)|}$. The *average expansion ratio* of A over $S = \frac{\Sigma_{f\in S}\frac{|A(f)|}{|f|}}{|S|}$. The *total expansion ratio* of A over $S = \frac{\Sigma_{f\in S}|A(f)|}{\Sigma_{f\in S}|f|}$.

2) Experimental Data: We obtained 65 real-life classifiers from different network service providers that range in size from dozens to thousands of rules. Although this collection of classifiers is diverse, some classifiers from the same source have similar structure and exhibited similar results under TCAM Razor. To prevent this repetition from skewing the performance data, we divided the 65 packet classifiers into 40 structurally distinct groups and randomly chose one from each of the 40 groups to form our experimental data set denoted as RL. Most classifiers have two decisions; however, one classifier has three decisions and three classifiers have four decisions. Fig. 7 shows the distribution of classifier sizes for RL.

To stress test the sensitivity of TCAM Razor on the number of decisions in a classifier, we created a set of classifiers RL_U by replacing the decision of every rule in each classifier by a unique decision. Thus, each classifier in RL_U has the maximum possible number of distinct decisions. Such classifiers might arise in the context of rule logging where the system monitors the frequency that each rule is the first matching rule for a packet.

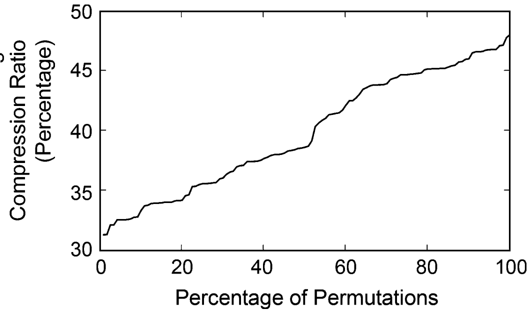


Fig. 8. Ratios for each permutation.

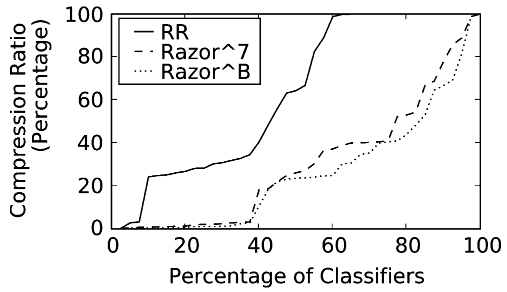


Fig. 9. Compression ratios for RL.

3) Variable Ordering: There are $5! = 120$ variable orders that Razor can use to convert a classifier into an equivalent FDD because there are $5! = 120$ different permutations of the five packet fields (source IP, destination IP, source port, destination port, and protocol type). We use Razor^p to denote the TCAM Razor algorithm using permutation p. For a given classifier f, we use Razor^B to denote the TCAM Razor algorithm using the best of the 120 permutations for f specifically. To find the variable order that achieves the best average compression ratio, for each permutation p, we computed the average compression ratio Razor^p on RL. Fig. 8 shows the corresponding accumulated percentage graph. From this figure, we see that the average compression ratios of 120 permutations range from 31.3% to 48.8%. The seventh permutation *protocol type, source IP, destination IP, source port, destination port* achieves the best average compression ratio, as well as the best total compression ratio, which is 29.0%.

4) Sensitivity To Number of Unique Decisions: TCAM Razor's effectiveness gracefully degrades as we increase the number of unique decisions in a classifier. In the extreme case where we assign each rule a unique decision in RL_U, Razor^7 achieves an average compression ratio of 48.3% and a total compression ratio of 55.8%.

5) Compression and Expansion Ratios: Fig. 11 shows the average compression and expansion ratios of Razor^7, Razor^B, and RR (denoting the redundancy removal algorithm) on both RL and RL_U. Figs. 9 and 10 show the accumulated percentage graphs for the compression and expansion ratios of Razor^7, Razor^B, and RR on RL. We draw two conclusions from

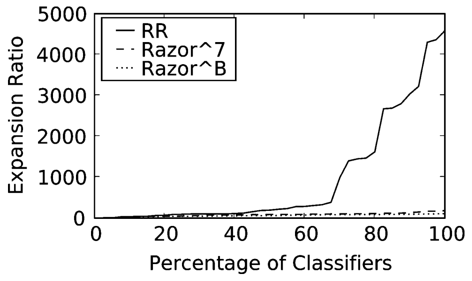


Fig. 10. Expansion ratios for RL.

# Rules	Seconds
691	0.47
807	0.81
1183	0.51
1308	1.10
1365	2.29
1794	2.99
2331	3.69
3410	3.03
3928	2.60
4004	5.13
4456	7.76
7652	21.419

Fig. 12. Razor^7 running time.

	Compression Ratio					
	Average			Total		
	RR	Razor^7	Razor^B	RR	Razor^7	Razor^B
RL	64.0 %	31.3 %	27.2 %	78.2 %	29.0 %	25.1 %
RL_U	78.9 %	48.3 %	46.1 %	86.1 %	55.8 %	46.6 %

	Expansion Ratio					
	Average			Total		
	RR	Razor^7	Razor^B	RR	Razor^7	Razor^B
RL	957.1 %	83.4 %	61.0 %	201.3 %	74.6 %	64.7 %
RL_U	2109.3 %	156.4 %	142.9 %	221.5 %	143.5 %	120.0 %

Fig. 11. Effectiveness of TCAM Razor.

our experimental results. First, the effectiveness of Razor^7 is close to Razor^B. For example, the average compression ratios of Razor^7 and Razor^B are 31.3% and 27.2%, respectively. Second, TCAM Razor significantly outperforms the redundancy removal algorithm. For example, the average compression ratio of Razor^7 is more than two times better than that of RR.

It is difficult to compare our results directly with those of Dong *et al.* [7] because we have access to neither their programs nor the classifiers they experimented with. However, Dong *et al.* reported a total compression ratio[2] of 54% on their real-life classifiers.

B. *Efficiency of TCAM Razor*

We implemented TCAM Razor using Visual Basic on Microsoft .Net framework 2.0. In our experiments, we tested TCAM Razor on RL. Our experiments were carried out on a desktop PC running Windows Vista with 4 G memory and a four CPU 2.4-GHz Intel Q6600. Note that TCAM Razor is not multithreaded, so only a single core is utilized for the experiments. Fig. 12 shows the total running time of Razor^7 for some representative classifiers. We see that TCAM Razor is very efficient. For classifiers with thousands of rules, it only takes a few seconds.

VIII. Conclusion

TCAMs have become the *de facto* industry standard for packet classification. However, as the rules in packet classifiers grow in number and complexity, the viability of TCAM-based solutions is threatened by the problem of range expansion. In this paper, we propose TCAM Razor, a systematic approach to minimizing TCAM rules for packet classifiers. While TCAM Razor does not always produce optimal packet classifiers, in our experiments with 40 structurally distinct real-life packet classifier groups, TCAM Razor achieves an average compression ratio of 31.3% and 29.0%, respectively. Unlike other solutions that require modifying TCAM circuits or packet processing hardware, TCAM Razor can be deployed today by network administrators and ISPs to cope with range expansion.

[2]By clarifying with the authors, the term "average compression ratio" in [7] is actually what we define as "total compression ratio" in this paper.

Acknowledgment

The authors thank the anonymous reviewers for their constructive comments. Their suggestions greatly improved this paper.

References

[1] Integrated Device Technology, Inc. content addressable memory. [Online]. Available: http://www.idt.com/
[2] B. Agrawal and T. Sherwood, "Modeling TCAM power for next generation network devices," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2006, pp. 120–129.
[3] D. A. Applegate, G. Calinescu, D. S. Johnson, H. Karloff, K. Ligett, and J. Wang, "Compressing rectilinear pictures and minimizing access control lists," in *Proc. ACM-SIAM SODA*, Jan. 2007, pp. 1066–1075.
[4] J. Bolaria and L. Gwennap, "A guide to search engines and networking memory," [Online]. Available: http://www.linleygroup.com
[5] A. Bremler-Barr and D. Hendler, "Space-efficient TCAM-based classification using gray coding," in *Proc. IEEE INFOCOM*, May 2007, pp. 1388–1396.
[6] H. Che, Z. Wang, K. Zheng, and B. Liu, "DRES: Dynamic range encoding scheme for TCAM coprocessors," *IEEE Trans. Comput.*, vol. 57, no. 7, pp. 902–915, Jul. 2008.
[7] Q. Dong, S. Banerjee, J. Wang, D. Agrawal, and A. Shukla, "Packet classifiers in ternary CAMs can be smaller," in *Proc. ACM Sigmetrics*, 2006, pp. 311–322.
[8] R. Draves, C. King, S. Venkatachary, and B. Zill, "Constructing optimal IP routing tables," in *Proc. IEEE INFOCOM*, 1999, pp. 88–97.
[9] D. Eastlake and P. Jones, "US secure hash algorithm 1 (Sha1)," RFC 3174, 2001.
[10] M. G. Gouda and A. X. Liu, "Structured firewall design," *Comput. Netw. J.*, vol. 51, no. 4, pp. 1106–1120, Mar. 2007.
[11] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Network*, vol. 15, no. 2, pp. 24–32, Mar.–Apr. 2001.
[12] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for advanced packet classification with ternary CAMs," in *Proc. ACM SIGCOMM*, Aug. 2005, pp. 193–204.
[13] C. Lambiri, Senior Staff Architect IDT, Private communication, 2008.
[14] P. C. Lekkas, *Network Processors—Architectures, Protocols, and Platforms*. New York: McGraw-Hill, 2003.
[15] A. X. Liu and M. G. Gouda, "Diverse firewall design," in *Proc. Int. Conf. DSN*, Jun. 2004, pp. 595–604.
[16] A. X. Liu and M. G. Gouda, "Complete redundancy removal for packet classifiers in TCAMS," *IEEE Trans. Parallel Distrib. Syst.*, 2005, to be published.
[17] A. X. Liu, C. R. Meiners, and Y. Zhou, "All-match based complete redundancy removal for packet classifiers in TCAMs," in *Proc. IEEE INFOCOM*, Apr. 2008, pp. 111–115.

[18] A. X. Liu, E. Torng, and C. Meiners, “Firewall compressor: An algorithm for minimizing firewall policies,” in *Proc. IEEE INFOCOM*, Phoenix, AZ, Apr. 2008, pp. 176–180.

[19] H. Liu, “Efficient mapping of range classifier into ternary-CAM,” in *Proc. Hot Interconnects*, 2002, pp. 95–100.

[20] C. R. Meiners, A. X. Liu, and E. Torng, “Topological transformation approaches to optimizing TCAM-based packet classification systems,” in *Proc. ACM SIGMETRICS*, Jun. 2009, pp. 73–84.

[21] D. Pao, Y. Li, and P. Zhou, “Efficient packet classification using TCAMs,” *Comput. Netw.*, vol. 50, no. 18, pp. 3523–3535, 2006.

[22] D. Pao, P. Zhou, B. Liu, and X. Zhang, “Enhanced prefix inclusion coding filter-encoding algorithm for packet classification with ternary content addressable memory,” *IET Comput. Digital Tech.*, vol. 1, no. 5, pp. 572–580, Sep. 2007.

[23] D. Pao, P. Zhou, B. Liu, and X. Zhang, “Enhanced prefix inclusion coding filter-encoding algorithm for packet classification with ternary content addressable memory,” *IET Comput. Digital Tech.*, vol. 1, pp. 572–580, Apr. 2007.

[24] R. Rivest, “The MD5 message-digest algorithm,” RFC 1321, 1992.

[25] E. Spitznagel, D. Taylor, and J. Turner, “Packet classification using extended TCAMs,” in *Proc. IEEE ICNP*, Nov. 2003, pp. 120–131.

[26] S. Suri, T. Sandholm, and P. Warkhede, “Compressing two-dimensional routing tables,” *Algorithmica*, vol. 35, pp. 287–300, 2003.

[27] J. van Lunteren and T. Engbersen, “Fast and scalable packet classification,” *IEEE J. Sel. Areas Commun.*, vol. 21, no. 4, pp. 560–571, May 2003.

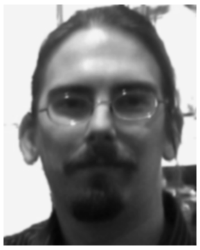
[28] F. Yu, T. V. Lakshman, M. A. Motoyama, and R. H. Katz, “SSA: A power and memory efficient scheme to multi-match packet classification,” in *Proc. Symp. ANCS*, Oct. 2005, pp. 105–113.

[29] K. Zheng, H. Che, Z. Wang, B. Liu, and X. Zhang, “DPPC-RE: TCAM-based distributed parallel packet classification with range encoding,” *IEEE Trans. Comput.*, vol. 55, no. 8, pp. 947–961, Aug. 2006.

Alex X. Liu received the Ph.D. degree in computer science from the University of Texas at Austin in 2006.

He is currently an Assistant Professor with the Department of Computer Science and Engineering, Michigan State University, East Lansing. His research interests focus on networking, security, and dependable systems.

Dr. Liu received the IEEE & IFIP William C. Carter Award in 2004 and a NSF CAREER Award in 2009.



Chad R. Meiners received the Ph.D. degree in computer science from Michigan State University, East Lansing, in 2009.

He is currently a Research Associate with the Department of Computer Science and Engineering, Michigan State University. His research interests include networking, algorithms, and security.

Eric Torng received the Ph.D. degree in computer science from Stanford University, Stanford, CA, in 1994.

He is currently an Associate Professor and Graduate Director with the Department of Computer Science and Engineering, Michigan State University, East Lansing. His research interests include algorithms, scheduling, and networking.

Dr. Torng received a NSF CAREER award in 1997.