Topological Transformation Approaches to Optimizing TCAM-Based Packet Classification Systems

Chad R. Meiners Alex X. Liu Eric Torng
Department of Computer Science and Engineering
Michigan State University
East Lansing, MI 48824, U.S.A.
{meinersc, alexliu, torng}@cse.msu.edu

ABSTRACT

Several range reencoding schemes have been proposed to mitigate the effect of range expansion and the limitations of small capacity, large power consumption, and high heat generation of TCAM-based packet classification systems. However, they all disregard the semantics of classifiers and therefore miss significant opportunities for space compression.

In this paper, we propose new approaches to range reencoding by taking into account classifier semantics. Fundamentally different from prior work, we view reencoding as a topological transformation process from one colored hyperrectangle to another where the color is the decision associated with a given packet. We present two orthogonal, yet composable, reencoding approaches, domain compression and prefix alignment. Our techniques significantly outperform all previous reencoding techniques. In comparison with the state-of-the-art results, our experimental results show that our techniques achieve at least 7 times more space reduction in terms of TCAM space for an encoded classifier and at least 3 times more space reduction in terms of TCAM space for a reencoded classifier and its transformers.

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, Security

Keywords

Packet Classification, TCAM, Range Encoding

1. INTRODUCTION

Packet classification is the core mechanism that enables many networking devices, such as routers and firewalls, to perform services such as 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 basic classification problem is to compare each packet with a list of predefined rules and find the first (i.e., highest priority) rule that the packet matches. Table 1 shows an example classifier of two rules. The format of these rules is based upon the format used in Access Control Lists on Cisco routers.

Rule	Src. IP	Dest. IP	Src. Port	Dest. Port	Prot.	Action
r_1	1.2.3.0/24	192.168.0.1	[1,65534]	[1,65534]	TCP	accept
r_2	*	*	*	*	*	discard

Table 1: An example packet classifier

Packet classification is often a performance bottleneck for routers as they need to classify every packet. Achieving wire speed packet classification has long been a networking goal. Although software-based packet classification has been extensively studied [28], using Ternary Content Addressable Memories (TCAMs) to perform hardware-based packet classification has become the de facto industrial standard [1,13].

A traditional random access memory chip receives an address and returns the content of the memory at that address. A TCAM chip, however, works in a reverse manner: it receives content and returns the address of the *first* entry where the content lies in the TCAM in constant time (*i.e.*, a few CPU cycles). Exploiting this hardware feature, TCAM-based packet classifiers store a rule in each entry as an array of 0's, 1's, or *'s (*don't-care* values). A packet header (*i.e.*, a search key) matches an entry if and only if their corresponding 0's and 1's match. Given a search key to a TCAM, the hardware circuits compare the key with all its occupied entries in parallel and return the index (or sometimes the content, depending on chip configuration,) of the first matching entry.

Unfortunately, TCAM-based solutions may not scale up to meet the classification needs of the rapidly growing Internet where packet classifiers are growing rapidly in size. First, current TCAMs have limited capacity. The largest available TCAM chip has a capacity of 36Mb [2], while 2Mb and 1Mb chips are the most popular [4]. Furthermore, the well known range expansion problem exacerbates the problem of limited capacity TCAMs. In a typical classifier rule, the three fields of source and destination IP addresses and protocol type are specified as prefixes where all the *s are at the end of the ternary string, so the fields can be directly stored in a TCAM. However, the other two fields of source and destination port numbers are specified in ranges (*i.e.*, integer intervals), which need to be mapped to one or more

SIGMETRICS/Performance'09, June 15–19, 2009, Seattle, WA, USA.

prefixes before being stored in a TCAM. This can lead to a large 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 1. Second, TCAM chip size growth has been and will likely continue to be slow due to their extremely high circuit density. Finally, even if larger TCAM chips were available, their deployment may be limited due to their high power consumption, large footprints, and high cost. TCAM chips consume lots of power and generate lots of heat because every memory access searches the entire active memory in parallel, and TCAM power consumption grows linearly with the number of ternary bits searched in each memory access [31]. Power constrains deployed TCAM chip size when systems designers must obey a "power budget", *e.g.*, TCAM components may use 10% of an entire board's power budget. Likewise, TCAM chips occupy 6 times (or more) board space than an equivalent SRAM which leads to TCAMs having high costs, even in large quantities. Due to these issues, the most popular TCAM chips are the 1Mb and 2Mb chips even though a 6Mb TCAM chip is commercially available.

Range reencoding schemes have been proposed to improve the scalability of TCAM-based systems, primarily by mitigating the effect of range expansion [5,6,13,19,22,23,30,32]. The basic idea is to first reencode a classifier into another classifier that requires less TCAM space and then reencode each packet correspondingly such that the decision made by the reencoded classifier for the reencoded packet is the same as the decision made by the original classifier for the original packet. Range reencoding has two possible benefits: rule width compression so that narrower TCAM entries can be used and rule number compression so that fewer TCAM entries can be used.

We observe that all previous reencoding schemes suffer from one fundamental limitation: they all ignore the decision associated with each rule and thus the classifier's decision for each packet. Disregarding classifier semantics leads all previous techniques to miss significant opportunities for space compression. Fundamentally different from prior work, we view reencoding as a topological transformation process from one colored hyperrectangle to another where the color is the decision associated with a given packet. Furthermore, we also view reencoding as a classification process that can be implemented with small TCAM tables, which enables fast packet reencoding. We present two orthogonal, yet composable, reencoding approaches, domain compression and prefix alignment. In domain compression, we transform a given colored hyperrectangle, which represents the semantics of a given classifier, to the smallest possible "equivalent" colored hyperrectangle. This leads to both *optimal rule width compression* as well as rule number compression. In prefix alignment, on the other hand, we strive for rule number compression only by transforming a colored hyperrectangle to an equivalent "prefix-friendly" colored hyperrectangle where the ranges align well with prefix boundaries, minimizing the costs of range expansion.

Domain Compression: In most packet classifiers, many coordinates (*i.e.*, values) within a field domain are equivalent. The idea of domain compression is to reencode the domain so as to eliminate as many redundant coordinates as possible. This leads to both rule width and rule number compression. From a geometric perspective, domain compression "squeezes" a colored hyperrectangle as much as possible. For example, consider the colored rectangle in Figure 1(A) that represents the classifier in Figure 1(H). In field F_1 represented by the X-axis, all values in $[0, 7] \cup [66, 99]$ are equivalent; that is, for any $y \in F_2$ and any $x_1, x_2 \in [0, 7] \cup [66, 99]$, packets (x_1, y) and (x_2, y) have the same decision. Therefore, when reencoding F_1, we can map all values in $[0, 7] \cup [66, 99]$ to a single value, say 0. By identifying such equivalences along all dimensions, the rectangle in Figure 1(A) is reencoded to the one in Figure 1(D), whose corresponding classifier is shown in Figure 1(I). Figures 1(B) and (C) show the two transforming tables for F_1 and F_2, respectively. We use "a" as a shorthand for "*accept*" and "d" as a shorthand for "*discard*".

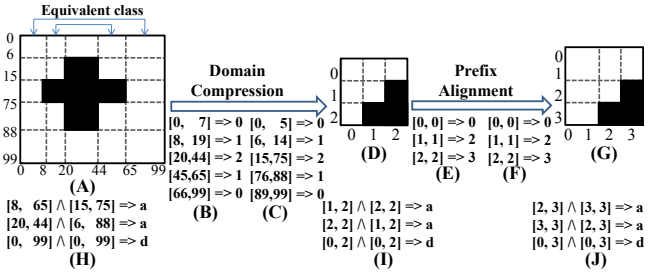


Figure 1: Example of topological transformations

Prefix Alignment: In prefix alignment, we "shift", "shrink", or "stretch" ranges by transforming the domain of each field to a new "prefix-friendly" domain so that the majority of the reencoded ranges either are prefixes or can be expressed by a small number of prefixes. This will reduce the costs of range expansion and leads to rule number compression with a potentially small loss in rule width compression. For example, consider the packet classifier in Figure 1(I), whose corresponding rectangle is in Figure 1(D). Range expansion will yield 5 prefix rules because interval $[1, 2]$ or $[01, 10]$ cannot be combined into one prefix. However, by transforming the rectangle in Figure 1(D) to the one in Figure 1(G), the range expansion of the resulting classifier, as shown in Figure 1(J), will have 3 prefix rules because $[2, 3]$ is expanded to 1*. Figures 1(E) and (F) show the two transforming tables for F_1 and F_2, respectively.

Our domain compression and prefix alignment techniques have several nice properties. First, they are powerful in reducing TCAM space. They achieve 3 to 7 times space reduction in comparison with state-of-the-art results. Second, they can be easily implemented on existing hardware by using TCAM to perform reencoding. Third, not only are they composable, they can also be composed with many other TCAM optimization and reencoding schemes proposed in prior work because the reencoded classifier produced by domain compression contains range rules and the prefix alignment technique can take any prefix classifier as its input.

We implemented our algorithms and conducted experiments on both real-world and synthetic classifiers. In comparison with the state-of-the-art results, the results show that our techniques achieve at least 8 times more space reduction with transformers excluded and approximately 3 times more space reduction with transformers included.

The rest of the paper proceeds as follows. We start by reviewing previous work in Section 2 and formally defining relevant terms in Section 3. In Section 4, we give an overview of our topological transformation approaches. In Sections 5

and 6, we present the technical details of the two topological transformation approaches. We discuss implementation issues in 7. Experimental results are presented in Section 8. We draw conclusions in Section 9. In the appendix, we define terms and notations that we use throughout the paper and present the formal proof of one result.

2. RELATED WORK

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

Circuit Modification: Spitznagel *et al.* proposed adding comparators at each entry level to better accommodate range matching [25]. While this research direction is important, such solutions are hard to deploy due to high cost [13].

Classifier Compression: These optimizations convert a given packet classifier to another semantically equivalent classifier that requires fewer TCAM entries. The schemes in [3, 8, 27] focus on one-dimensional and two dimensional packet classifiers. The redundancy removal algorithms in [16–18] can reduce TCAM usage by eliminating all the redundant rules in a packet classifier. In [7], Dong *et al.* proposed schemes to reduce range expansion by repeatedly expanding or trimming ranges to prefix boundaries and then using the redundancy removal algorithm in [16] to verify the correctness of a modification. Their observations on prefix boundaries have a similar flavor to our prefix alignment approach, although their methods are fundamentally different. In [20] Meiners, *et al.* proposed a greedy algorithm that finds locally minimal solutions along each field and combines these solutions into a smaller equivalent packet classifier. In [21], Meiners *et al.* proposed the first algorithm that can compress a given classifier into a non-prefix ternary classifier.

Range Reencoding: Previous range reencoding schemes fall into two categories: those that only consider rule number compression, often at the expense of rule width [5, 6, 13, 19, 32] and those that attempt to both compress rule number and rule width [22, 23, 30]. In [19], Liu proposed a scheme that allocates specific TCAM column bits to represent ranges in a manner similar to Lakshman and Stiliadis' software bitmap classification method [12]. Lakshminarayan *et al.* [13] proposed a scheme called fence encoding, which encodes interval ranges as a range of unary numbers. Fence encoding has an expansion factor of one, meaning all ranges can be encoded with one string, but the number of unary bits required for each rule is prohibitive. To reduce rule width, Lakshminarayan *et al.* proposed DIRPE, which compresses the width of fence encodings at the expense of a larger expansion factor. Bremler-Barr and Hendler [5] proposed SRGE, which utilizes the structural properties of binary reflected gray codes to reduce range expansion without increasing rule width. Lunteren and Engbersen proposed a hierarchy of three methods, P^2C, that can be used to compress both rule number and rule width [30]. Two methods guarantee an expansion factor of one but have potentially larger rule widths. The third method has the best rule width compression at the cost of expansion factors greater than one. Pao *et al.* proposed a prefix inclusion method (PIC) that achieves better rule width compression than P^2C [22, 23]. Che *et al.* [6,32] and Pao *et al.* [22,23] propose using TCAMs to reencode packets.

Reencoding has been used in software based packet classification. Lakshman and Stiliadis proposed to reencode each field's value into a bitmap that specifies the containment relationship among values and rules [12]. Given a reencoded packet, this method uses customized parallel AND gates to perform an intersection of these bitmaps and ultimately find the first matching rule. Srinivasan *et al.* proposed an encoding method called *cross-producting* that assigns a unique number to each disjoint range within a classifier field and constructs a lookup table for the cross product of the numbers associated with each field [26]. Gupta and McKeown proposed Recursive Flow Classification (RFC) [10], an optimized version of the cross-producting scheme that uses recursive cross-producting tables to reduce the space requirements of regular cross producting tables. Furthermore, they map disjoint ranges that are contained by the same set of rules into a single value. RFC's mapping tables use a weaker equivalence relation than our domain compression technique, so they do not achieve as much compression as we do. Unfortunately, these software based reencoding methods are difficult to deploy because the required RAM to perform the reencoding is extremely large. By using TCAMs to perform reencoding, we overcome this memory issue.

3. FORMAL DEFINITIONS

We now formally define the concepts of fields, packets, and packet classifiers. 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 w bits, denoted $D(F_i)$, is $[0, 2^w - 1]$. A *packet* over the d fields $F_1, \cdots, F_d$ is a d-tuple $(p_1, \cdots, p_d)$ 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 address, destination IP address, source port number, destination port number, and protocol type. The lengths of these packet fields are 32, 32, 16, 16, and 8, respectively. We use Σ to denote the set of all packets over fields $F_1, \cdots, F_d$. 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 nonnegative integer interval. A *prefix* $\{0,1\}^k\{*\}^{w-k}$ with k leading 0s or 1s for a packet field of length w denotes the integer interval $[\{0,1\}^k\{0\}^{w-k}, \{0,1\}^k\ \{1\}^{w-k}]$. For example, prefix 01** denotes the interval $[0100, 0111]$. 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.

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

A sequence of rules $\langle r_1, \cdots, r_n \rangle$ 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 a packet 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* $\mathbb{C}$ is a sequence of rules that is complete. The size of $\mathbb{C}$, denoted $|\mathbb{C}|$, is the number of rules in $\mathbb{C}$. A packet classifier $\mathbb{C}$ is a *prefix packet classifier* if and only if every rule in $\mathbb{C}$ is a prefix rule.

Two rules in a packet classifier may *overlap*; that is, a single packet may match both rules. Furthermore, two rules in a packet classifier may *conflict*; that is, the two rules not only overlap but also have different decisions. Packet classifiers typically resolve such 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 $\mathbb{C}$. The decision that packet classifier $\mathbb{C}$ makes for packet p is denoted $\mathbb{C}(p)$.

We can think of a packet classifier $\mathbb{C}$ as defining a many-to-one mapping function from Σ to DS. Two packet classifiers $\mathbb{C}_1$ and $\mathbb{C}_2$ are *equivalent*, denoted $\mathbb{C}_1 \equiv \mathbb{C}_2$, if and only if they define the same mapping function from Σ to DS; that is, for any packet $p \in \Sigma$, we have $\mathbb{C}_1(p) = \mathbb{C}_2(p)$. A rule is *redundant* in a classifier if and only if removing the rule does not change the semantics of the classifier.

In a typical packet classifier rule, the fields of source IP, destination IP, and protocol type are specified in prefix format, which can be directly stored in TCAMs; however, the remaining two fields of source port and destination port are specified as ranges (*i.e.*, non-negative integer intervals), which are typically converted to prefixes before being stored in TCAMs. This leads to *range expansion*, the process of converting a non-prefix rule to 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 $w-$bit range results in a set containing $2w - 2$ prefixes [11]. The next step is to compute the cross product of the set of prefixes for each field, resulting in a potentially large number of prefix rules.

4. TOPOLOGICAL TRANSFORMATION

Given a d-dimensional classifier $\mathbb{C}$ over fields $F_1, \cdots, F_d$, a topological transformation process produces two separate components. The first component is a set of *transformers* $\mathbb{T} = \{\mathbb{T}_i \mid 1 \leq i \leq d\}$ where transformer $\mathbb{T}_i$ transforms $D(F_i)$ into a new domain $D'(F_i)$. Together, the set of transformers $\mathbb{T}$ transforms the original packet space Σ into a new packet space Σ'. The second component is a transformed d-dimensional classifier $\mathbb{C}'$ over packet space Σ' such that for any packet $(p_1, \cdots, p_d) \in \Sigma$, the following condition holds:

$$\mathbb{C}(p_1, \cdots, p_d) = \mathbb{C}'(\mathbb{T}_1(p_1), \cdots, \mathbb{T}_d(p_d))$$

Each of the d transformers $\mathbb{T}_i$ and the transformed packet classifier $\mathbb{C}'$ are implemented in TCAM.

The TCAM space needed by our transformation approach is measured by the total TCAM space needed by the $d + 1$ tables: $\mathbb{C}'$, $\mathbb{T}_1$, $\cdots$, $\mathbb{T}_d$. We define the space used by a classifier or transformer in a TCAM as the number of entries (*i.e.*, rules) multiplied by the width of the TCAM in bits: *space = # of entries × TCAM width*. Although TCAMs can be configured with varying widths, they do not allow arbitrary widths. The width of a TCAM typically can be set at 36, 72, 144, and 288 bits (per entry). The primary goal of the transformation approach is to produce $\mathbb{C}', \mathbb{T}_1, \cdots, \mathbb{T}_d$ such that the TCAM space needed by these $d + 1$ TCAM tables is much smaller than the TCAM space needed by the original classifier $\mathbb{C}$. Most previous reencoding approaches ignore the space required by the transformers and only focus on the space required by the transformed classifier $\mathbb{C}'$. Note that we can implement the table for the protocol field using SRAM if desired since the field has only 8 bits.

There are two natural architectures for storing the $d + 1$ TCAM tables $\mathbb{C}', \mathbb{T}_1, \cdots, \mathbb{T}_d$: the *multi-lookup architecture* and the *pipelined-lookup architecture*.

In the multi-lookup architecture, we store all the $d + 1$ tables in one TCAM chip. For each table, we prepend a $\lceil \log(d + 1) \rceil$ *table ID* bit string to every entry. Figure 2 illustrates the packet classification process using the multi-lookup architecture when $d = 2$. Suppose we use the table IDs 00, 01, and 10 for the three tables $\mathbb{C}'$, $\mathbb{T}_1$, and $\mathbb{T}_2$, respectively. Given a packet (p_1, p_2), we first concatenate $\mathbb{T}_1$'s table ID 01 with p_1 and use the resulting bit string $01|p_1$ as the search key for the TCAM. Let ${p_1}'$ denote the search result. Second, we concatenate $\mathbb{T}_2$'s table ID 10 with p_2 and use the resulting bit string $10|p_2$ as the search key for the TCAM. Let ${p_2}'$ denote the search result. Third, we concatenate the table ID 00 of $\mathbb{C}'$ with ${p_1}'$ and ${p_2}'$ and use the resulting bit string $00|{p_1}'|{p_2}'$ as the search key for the TCAM. The search result is the final decision for the given packet (p_1, p_2).

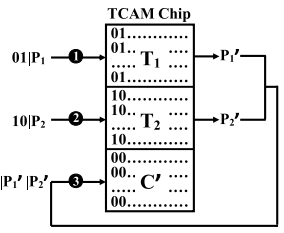


Figure 2: Multi-lookup

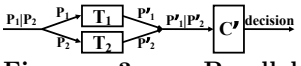


Figure 3: Parallel pipelined-lookup

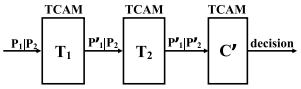


Figure 4: Chained pipelined-lookup

There are two natural pipelined-lookup architectures: parallel pipelined-lookup and chained pipelined-lookup. In both, we store the $d + 1$ tables in $d + 1$ separate TCAMs, so table IDs are no longer needed. In the parallel pipelined-lookup architecture, the d transformer tables $\mathbb{T}$, laid out in parallel, form a two-element pipeline with the transformed classifier $\mathbb{C}'$. Figure 3 illustrates the packet classification process using the parallel pipelined-lookup architecture when $d = 2$. Given a packet (p_1, p_2), we send p_1 and p_2, in parallel over separate buses, to $\mathbb{T}_1$ and $\mathbb{T}_2$, respectively. Then, the search result ${p_1}'|{p_2}'$ is used as a key to search on $\mathbb{C}'$. This second search result is the final decision for the given packet (p_1, p_2). Figure 4 illustrates the packet classification process using the chained pipelined-lookup architecture when $d = 2$.

The main advantage of the multi-lookup architecture is that it can be easily deployed since it requires minimal modification of existing TCAM-based packet processing systems. Its main drawback is a modest slowdown in packet processing throughput because $d + 1$ TCAM searches are required to process a d-dimensional packet. In contrast, the main advantage of the two pipelined-lookup architectures is high packet processing throughput. Their main drawback is that the hardware needs to be modified to accommodate $d + 1$ TCAM chips (or d chips if use SRAM for the protocol field).

Implementing our reencoding schemes on pipelined-lookup architectures actually improves packet processing throughput over conventional TCAM implementations. While the width of TCAM entries can be set to 36, 72, 144, or 288 bits, the typical TCAM bus width is 72 bits. Thus, the conventional TCAM lookup approach, which uses a TCAM entry width of 144 bits, requires four TCAM bus cycles to process a packet. Because all the tables produced by our reencod-

ing schemes have widths less than 36 bits, we can set the TCAM width to be 36. Thus, using a pipelined-lookup architecture, we can achieve a classification throughput of one packet per cycle; using multi-lookup architectures, we can achieve a classification throughput of one packet per twelve cycles.

5. DOMAIN COMPRESSION

We now describe our *domain compression* technique. The basic idea is to simplify the logical structure of a classifier by mapping the domain of each field $D(F_i)$ to the smallest possible domain $D'(F_i)$. We implement domain compression by exploiting the equivalence classes that any classifier $\mathbb{C}$ defines on the domain of each of its fields. Domain compression is especially powerful because it contributes to both rule width compression, which allows us to use 36 bit TCAM entries instead of 144 bit TCAM entries, and rule number compression because each transformed rule r' in classifier $\mathbb{C}'$ will contain fewer equivalence classes than the original rule r did in classifier $\mathbb{C}$. Through domain compression and redundancy removal, $\mathbb{C}'$ typically has far fewer rules than $\mathbb{C}$ did, something no other reencoding scheme can achieve.

Our domain compression algorithm consists of three steps: (1) computing equivalence classes, (2) constructing transformer $\mathbb{T}_i$ for each field F_i, and (3) constructing the transformed classifier $\mathbb{C}'$.

5.1 Step 1: Compute Equivalence Classes

We first formally define the equivalence relation that classifier $\mathbb{C}$ defines on each field domain and the resulting equivalence classes. We use the notation Σ_{-i} to denote the set of all $(d-1)$-tuple packets over the fields (F_1, $\cdots$, F_{i-1}, F_{i+1}, $\cdots$, F_d) and p_{-i} to denote an element of Σ_{-i}. Then we use $\mathbb{C}(p_i, p_{-i})$ to denote the decision that packet classifier $\mathbb{C}$ makes for the packet p that is formed by combining $p_i \in D(F_i)$ and p_{-i}.

DEFINITION 5.1 (EQUIVALENCE CLASS). *Given a packet classifier $\mathbb{C}$ over fields $F_1, \cdots, F_d$, we say that $x, y \in D(F_i)$ for $1 \leq i \leq d$ are* equivalent *with respect to $\mathbb{C}$ if and only if $\mathbb{C}(x, p_{-i}) = \mathbb{C}(y, p_{-i})$ for any $p_{-i} \in \Sigma_{-i}$. It follows that $\mathbb{C}$ partitions $D(F_i)$ into* equivalence classes. *We use the notation $\mathbb{C}\{x\}$ to denote the equivalence class that x belongs to as defined by classifier $\mathbb{C}$.*

In domain compression, we compress every equivalence class in each domain $D(F_i)$ to a single point in $D'(F_i)$. The crucial tool of our domain compression algorithm is the Firewall Decision Diagram (FDD) [9]. A *Firewall Decision Diagram* (FDD) with a decision set DS and over fields $F_1, \cdots, F_d$ 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 are called *terminal* nodes. (2) Each node v has a label, denoted $F(v)$, such that

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

(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: (i) *Consistency*: $I(e) \cap I(e') = \emptyset$ for any two distinct edges e and e' in $E(v)$. (ii) *Completeness*: $\bigcup_{e \in E(v)} I(e) = D(F(v))$.

We define a *full-length ordered FDD* as an FDD where in each decision path all fields appear exactly once and in the same order. For simplicity, we use the term "FDD" to mean "full-length ordered FDD" if not otherwise specified. Given a classifier $\mathbb{C}$, the FDD construction algorithm in [15] can convert it to an equivalent full-length ordered FDD f.

After an FDD f is constructed, we can reduce f's size by merging isomorphic subgraphs. A full-length ordered FDD f is *reduced* if and only if it satisfies the following two conditions: (1) no two nodes in f are isomorphic; (2) no two nodes have more than one edge between them. 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. A reduced FDD is essentially a canonical representation for packet classifiers.

The first step of our domain compression algorithm is to convert a given d-dimensional packet classifier $\mathbb{C}$ to d equivalent reduced FDDs f_1 through f_d where the root of FDD f_i is labeled by field F_i. Figure 5(a) shows an example packet classifier over two fields F_1 and F_2 where the domain of each field is [0,63]. Figures 5(b) and (c) show the two FDDs f_1 and f_2, respectively. The FDDs f_1 and f_2 are almost reduced except that the terminal nodes are not merged together for illustration purposes.

The crucial observation is that each edge out of reduced FDD f_i's root node corresponds to one equivalence class of domain $D(F_i)$. For example, consider the the classifier in Figure 5(a) and the corresponding FDD f_1 in Figure 5(b). Obviously, for any p_1 and $p_1{}'$ in $[7, 11] \cup [16, 19] \cup [39, 40] \cup [43, 60]$, we have $\mathbb{C}(p_1, p_2) = \mathbb{C}(p_1{}', p_2)$ for any p_2 in [0,63], so it follows that $\mathbb{C}\{p_1\} = \mathbb{C}\{p_1{}'\}$.

THEOREM 5.1 (EQUIVALENCE CLASS THEOREM). *For any packet classifier $\mathbb{C}$ over fields $F_1, \cdots, F_d$ and an equivalent reduced FDD f_i rooted at an F_i node v, the labels of v's outgoing edges are all the equivalence classes over field F_i as defined by $\mathbb{C}$.*

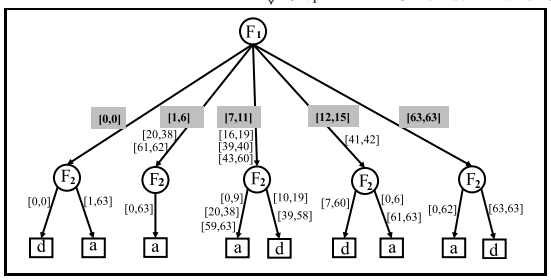
5.2 Step 2: Construct Transformers

Given a packet classifier $\mathbb{C}$ over fields $F_1, \cdots, F_d$ and the d equivalent reduced FDDs $f_1, \cdots, f_d$ where the root node of f_i is labeled F_i, we compute transformer $\mathbb{T}_i$ as follows. Let v be the root of f_i with m outgoing edges $e_1, \cdots, e_m$. First, for each edge e_j out of v, we choose one of the ranges in e_j's label to be a representative label, which we call the *landmark*. By Theorem 5.1, all the ranges in e_j's label belong to the same equivalence class, so any one of them can be chosen as the landmark. For each equivalence class, we choose the range that intersects the fewest number of rules in $\mathbb{C}$ as the landmark breaking ties arbitrarily. We then sort edges in the increasing order of their landmarks. We use L_j and e_j to denote the landmark range and corresponding edge in sorted order where edge e_1 has the smallest valued landmark L_1 and edge e_m has the largest valued landmark L_m. Our transformer $\mathbb{T}_i$ then maps all values in e_j's label to value j where $1 \leq j \leq m$. For example, in Figures 5(b) and

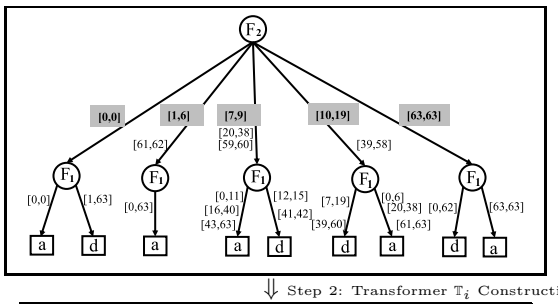
F_1	F_2	Decision
$[12, 15]$	$[7, 60]$	Discard
$[41, 42]$	$[7, 60]$	Discard
$[20, 38]$	$[0, 63]$	Accept
$[0, 63]$	$[20, 38]$	Accept
$[7, 60]$	$[10, 58]$	Discard
$[1, 63]$	$[0, 62]$	Accept
$[0, 62]$	$[1, 63]$	Accept
$[0, 63]$	$[0, 63]$	Discard

(a)

⇓ Step 1: FDD Construction and Reduction



(b)



(c)

⇓ Step 2: Transformer $\mathbb{T}_i$ Construction

F_1	Decision
$[0, 0]$	0
$[1, 6] \cup [20, 38] \cup [61, 62]$	1
$[7, 11] \cup [16, 19] \cup [39, 40] \cup [43, 60]$	2
$[12, 15] \cup [41, 42]$	3
$[63, 63]$	4

(d)

F_2	Decision
$[0, 0]$	0
$[1, 6] \cup [61, 62]$	1
$[7, 9] \cup [20, 38] \cup [59, 60]$	2
$[10, 19] \cup [39, 58]$	3
$[63, 63]$	4

(e)

⇓ Step 3: Classifier $\mathbb{C}'$ Construction

F_1	F_2	Decision
$[3, 3]$	$[2, 3]$	Discard
$\emptyset$	$[2, 3]$	Discard
$\emptyset$	$[0, 4]$	Accept
$[0, 4]$	$\emptyset$	Accept
$[2, 3]$	$[3, 3]$	Discard
$[1, 4]$	$[0, 3]$	Accept
$[0, 3]$	$[1, 4]$	Accept
$[0, 4]$	$[0, 4]$	Discard

(f)

⇓

F_1	F_2	Decision
$[3, 3]$	$[2, 3]$	Discard
$[2, 3]$	$[3, 3]$	Discard
$[1, 4]$	$[0, 3]$	Accept
$[0, 3]$	$[1, 4]$	Accept
$[0, 4]$	$[0, 4]$	Discard

(g)

Figure 5: Example of domain compression

(c), the greyed ranges are chosen as the landmarks of their corresponding equivalence classes, and Figures 5(d) and (e) show transformers $\mathbb{T}_1$ and $\mathbb{T}_2$ that result from choosing those landmarks.

5.3 Step 3: Construct Transformed Classifier

We now construct transformed classifier $\mathbb{C}'$ from classifier $\mathbb{C}$ using transformers $\mathbb{T}_i$ for $1 \leq i \leq d$ as follows. Let $F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d \rightarrow \langle decision \rangle$ be an original rule in $\mathbb{C}$. The domain compression algorithm converts $F_i \in S_i$ to $F_i{}' \in S_i{}'$ such that for any landmark range L_j $(0 \leq j \leq m-1)$, $L_j \cap S_i \neq \emptyset$ if and only if $j \in S_i{}'$. Stated another way, we replace range S_i with range $[a, b] \subseteq D'(F_i)$ where a is the smallest number in $[0, m-1]$ such that $L_a \cap S_i \neq \emptyset$ and b is the largest number in $[0, m-1]$ such that $L_b \cap S_i \neq \emptyset$. Note, it is possible no landmark ranges intersect range S_i; in this case a and b are undefined and $S_i{}' = \emptyset$. For a converted rule $r' = F_1{}' \in S_1{}' \wedge \cdots \wedge F_d{}' \in S_d{}' \rightarrow \langle decision \rangle$ in $\mathbb{C}'$, if there exists $1 \leq i \leq d$ such that $S_i{}' = \emptyset$, then this converted rule r' can be deleted from $\mathbb{C}'$.

Consider the rule $F_1 \in [7, 60] \wedge F_2 \in [10, 58] \rightarrow$ *discard* in the example classifier in Figure 5(a). For field F_1, the five landmarks are the five greyed intervals in 5(b), namely [0,0], [1,6], [7,11], [12,15], and [63, 63]. Among these five landmarks, [7,60] overlaps with [7,11] and [12,15], which are mapped to 2 and 3 respectively by transformer $\mathbb{T}_1$. Thus, $F_1 \in [7, 60]$ is converted to $F_1{}' \in [2, 3]$. Similarly, for field F_2, [10,58] overlaps with only one of F_2's landmarks, [10, 19], which is mapped to 3 by F_2's mapping table. Thus, $F_2 \in [10, 58]$ is converted to $F_2{}' \in [3, 3]$.

We now prove that $\mathbb{C}'$ together with $\mathbb{T}$ is semantically equivalent to $\mathbb{C}$.

THEOREM 5.2. *Consider any classifier $\mathbb{C}$ and the resulting transformers $\mathbb{T}$ and transformed classifier $\mathbb{C}'$. For any packet $p = (p_1, \cdots, p_d)$, we have*

$$\mathbb{C}(p_1, \cdots, p_d) = \mathbb{C}'(\mathbb{T}_1(p_1), \cdots, \mathbb{T}_d(p_d)).$$

PROOF. For each field F_i for $1 \leq i \leq d$, consider p's field value p_i. Let $L(p_i)$ be the landmark range for $\mathbb{C}\{p_i\}$. We set $x_i = \min(L(p_i))$. We now consider the packet $x = (x_1, \cdots x_d)$ and the packets $x(j) = (x_1, \ldots x_{j-1}, p_j, \ldots, p_d)$ for $0 \leq j \leq d$; that is, in packet $x(j)$, the first j fields are identical to packet x and the last $d-j$ fields are identical to packet p. Note $x(0) = p$ and $x(d) = x$. We now show that $\mathbb{C}(p) = \mathbb{C}(x)$. This follows from $\mathbb{C}(x(0)) = \mathbb{C}(x(1)) = \cdots = \mathbb{C}(x(d))$. Each equality follows from the fact that x_j and p_j belong to the same equivalence class within $D(F_j)$.

Let r be the first rule in $\mathbb{C}$ that packet x matches. We argue that p' will match the transformed rule $r' \in \mathbb{C}'$. Consider the conjunction $F_i \in S_i$ of rule r. Since x matches rule r, it must be the case that $x_i \in S_i$. This implies that $L(p_i) \cap S_i \neq \emptyset$. Thus, by our construction $p_i{}' = \mathbb{T}_i(p_i) = \mathbb{T}_i(x_i) \in S_i{}'$. Since this holds for all fields F_i, packet p' matches rule r'. We also argue that packet p' will not match any rule before transformed rule $r' \in \mathbb{C}'$. Suppose packet p' matches some rule $r_1{}' \in \mathbb{C}'$ that occurs before rule r'. This implies that for each conjunction $F_i \in S_i$ of the corresponding rule $r_1 \in \mathbb{C}$ that $L(p_i) \cap S_i \neq \emptyset$. However, this implies that $x_i \in S_i$ since if any point in $L(p_i)$ is in S_i, then all points in $L(p_i)$ are in S_i. It follows that x matches rule $r_1 \in \mathbb{C}$, contradicting our assumption that rule r was the first rule that x matches in $\mathbb{C}$. Thus, it follows that p' cannot match rule $r_1{}'$. It then follows that r' will be the first rule in $\mathbb{C}$ that p' matches and the theorem follows. □

6. PREFIX ALIGNMENT

We now describe our *prefix alignment* approach. The basic idea is to "shift", "shrink", or "stretch" ranges by transforming the domain of each field to a new "prefix-friendly" domain so that the majority of the reencoded ranges either are prefixes or can be expressed by a small number of prefixes. This will reduce the costs of range expansion with perhaps a small penalty in rule width.

We first solve the special case where $\mathbb{C}$ has only one field F. We develop an optimal solution using dynamic programming techniques. We then use this solution as a building block to perform prefix alignment on multi-dimensional classifiers. Finally, we compose domain compression and prefix alignment together.

6.1 Prefix Alignment Overview

The one-dimensional prefix alignment problem is equivalent to the following "cut" problem. Consider the three ranges $[0, 12]$, $[5, 15]$, and $[0, 15]$ over domain $D(F_1) = [0, 15]$ in classifier $\mathbb{C}$ in Figure 6(A), and suppose the transformed domain $D'(F_1) = [00, 11]$ in binary format. Because $D'(F_1)$ has a total of 4 elements, we want to identify three cut points $0 \le x_1 < x_2 < x_3 \le 15$ such that if $[0, x_1] \in D(F_1)$ transforms to $00 \in D'(F_1)$, $[x_1 + 1, x_2] \in D(F_1)$ transforms to $01 \in D'(F_1)$, $[x_2+1, x_3] \in D(F_1)$ transforms to $10 \in D'(F_1)$, and $[x_3 + 1, 15] \in D(F_1)$ transforms to $01 \in D'(F_1)$, the range expansion of the transformed ranges will have as few rules as possible. For this simple example, there are two families of optimal solutions: those with x_1 anywhere in $[0, 3]$, $x_2 = 4$, and $x_3 = 12$, and those with $x_1 = 4$, $x_2 = 12$, and x_3 anywhere in $[13, 15]$. For the first family of solutions, range $[0, 12]$ is transformed to $[00, 10] = 0* \cup 10$, range $[5, 15]$ is transformed to $[10, 11] = 1*$, and range $[0, 15]$ is transformed to $[00, 11] = **$. In the second family of solutions, range $[0, 12]$ is transformed to $[00, 01] = 0*$, range $[5, 15]$ is transformed to $[01, 11] = 01 \cup 1*$, and range $[0, 15]$ is transformed to $[00, 11] = **$. The classifier $\mathbb{C}'$ in Figure 6(A) shows the three transformed ranges using the first family of solutions. In both examples, the range expansion of the transformed ranges only has 4 prefix rules while the range expansion of the original ranges has 7 prefix rules.

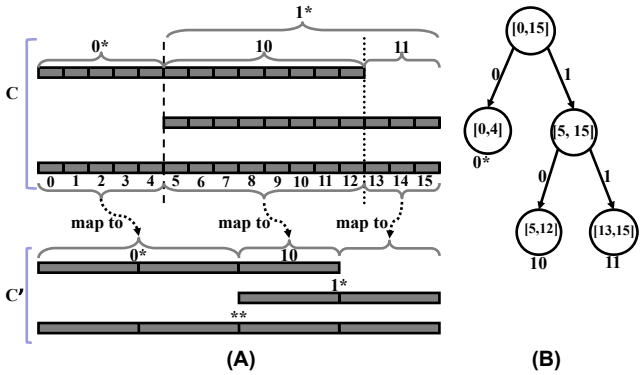


Figure 6: Example of 1-D prefix alignment

We now illustrate how to compute an optimal solution using a divide and conquer strategy. We first observe that we can divide the original problem into two subproblems by choosing the middle cut point. We next observe that a cut point should be the starting or ending point of a range, if possible, in order to reduce range expansion. Suppose the target domain $D'(F_1)$ is $[0, 2^b - 1]$. We first need to choose the middle cut point $x_{2^{b-1}}$, which will divide the problem into two subproblems with target domains $[0, 2^{b-1} - 1] = 0\{*\}^{b-1}$ and $[2^{b-1}, 2^b - 1] = 1\{*\}^{b-1}$ respectively. Consider the example in Figure 6(A), the x_2 cut point partitions $[0, 15]$ into $[0, x_2]$, which transforms to prefix $0*$, and $[x_2 + 1, 15]$, which transforms to prefix $1*$. The second observation implies either $x_2 = 4$ or $x_2 = 12$. Suppose we choose $x_2 = 4$; that is, we choose the dashed line in Figure 6(A). This produces two subproblems where we need to identify the x_1 cut point in the range $[0, 4]$ and the x_3 cut point in $[5, 15]$. In the two subproblems, we include each range trimmed to fit the restricted domain. For example, ranges $[0, 12]$ and $[0, 15]$ are trimmed to $[0, 4]$ in the first subproblem. In the second subproblem, ranges $[5, 15]$ and $[0, 15]$ are trimmed to $[5, 15]$ while range $[0, 12]$ is trimmed to $[5, 12]$. We must maintain each trimmed range even if there may be duplicates. In the first subproblem, the choice of x_1 is immaterial since both trimmed ranges span the entire restricted domain. In the second subproblem, the range $[5, 12]$ dictates that $x_3 = 12$ is the right choice.

We represent this divide and conquer process of computing cut points as a binary cut tree. Figure 6(B) depicts the tree where we select $x_2 = 4$ and $x_3 = 12$. This tree also encodes the transformation from the original domain to the target domain: *all the values in a terminal node are mapped to the prefix represented by the path from the root to the terminal node.* For example, as the path from the root to the terminal node of $[0, 4]$ is 0, all values in $[0, 4] \in D(F_1)$ are transformed to $0*$.

In domain compression, we considered transformers that mapped points in $D(F_i)$ to points in $D'(F_i)$. In prefix alignment, we consider transformers that map points in $D(F_i)$ to prefix ranges in $D'(F_i)$. If this is confusing, we can also work with transformers that map points in $D(F_i)$ to points in $D'(F_i)$ with no change in results; however, transformers that map to prefixes more accurately represent the idea of prefix alignment than transformers that map to points. Because we will perform range expansion on $\mathbb{C}'$ before performing any further optimizations including redundancy removal, we can ignore rule order. We can then view a one-dimensional classifier $\mathbb{C}$ as a multiset of ranges S in $D(F_1)$.

6.2 One-dimensional Prefix Alignment

We next present the technical details of our dynamic programming solution to the prefix alignment problem by addressing four issues.

6.2.1 Correctness of Prefix Alignment

We prove that prefix alignment preserves the semantics of the original classifier by first defining the concept of *prefix transformers* and then showing that prefix alignment must be correct when prefix transformers are used.

Given a prefix P, we use $\min P$ and $\max P$ to denote the smallest and the largest values in P, respectively.

DEFINITION 6.1 (PREFIX TRANSFORMERS). *A transformer $\mathbb{T}_i$ is an order-preserving prefix transformer from $D(F_i)$ to $D'(F_i)$ for a packet classifier $\mathbb{C}$ if $\mathbb{T}_i$ satisfies the following three properties. (1) (prefix property) $\forall x \in D(F_i)$, $\mathbb{T}_i(x) = P$ where P is a prefix in domain $D'(F_i)$; (2) (order-preserving property) $\forall x, y \in D(F_i)$, $x < y$ implies either $\mathbb{T}_i(x) = \mathbb{T}_i(y)$ or $\max \mathbb{T}_i(x) < \min \mathbb{T}_i(y)$; (3) (consistency property) $\forall x, y \in D(F_i)$, $\mathbb{T}_i(x) = \mathbb{T}_i(y)$ implies $\mathbb{C}\{x\} = \mathbb{C}\{y\}$.*

The following Lemma 6.1 and Theorem 6.1 easily follow from the definition of prefix transformers.

LEMMA 6.1. *Given any prefix transformer $\mathbb{T}_i$ for a field F_i, for any $a, b, x \in D(F_i)$, $x \in [a, b]$ if and only if $\mathbb{T}_i(x) \subseteq [\min \mathbb{T}_i(a), \max \mathbb{T}_i(b)]$.*

THEOREM 6.1 (PREFIX ALIGNMENT THEOREM). *Given a packet classifier $\mathbb{C}$ over fields $F_1, \cdots, F_d$, and d prefix transformers $T = \{\mathbb{T}_i \mid 1 \leq i \leq d\}$, and the classifier $\mathbb{C}'$ constructed by replacing any range $[a, b]$ over field F_i $(1 \leq i \leq d)$ by the range $[\min \mathbb{T}_i(a), \max \mathbb{T}_i(b)]$, the condition $\mathbb{C}(p_1, \cdots, p_d) = \mathbb{C}'(\mathbb{T}_1(p_1), \cdots, \mathbb{T}_d(p_d))$ holds.*

6.2.2 Find Candidate Cut Points

We next identify candidate cut points using the concept of atomic ranges. For any multiset of ranges S (a multiset may have duplicate entries) and any range x over domain $D(F_1)$, we use $S@x$ to denote the set of ranges in S that contain x.

DEFINITION 6.2 (ATOMIC RANGE SET). *Given a multiset S of ranges, the union of which constitute a range denoted $\bigcup S$, and a set of ranges S', S' is the atomic range set of S if and only if the following four conditions hold: (1) (coverage property) $\bigcup S = \bigcup S'$; (2) (disjoint property) $\forall x, y \in S'$, $x \cap y = \emptyset$; (3) (atomicity property) $\forall x \in S$ and $\forall y \in S'$, $x \cap y \neq \emptyset$ implies $y \subseteq x$; (4) (maximality property) $\forall x, y \in S'$ and $\max x + 1 = \min y$ implies $S@x \neq S@y$.*

For any multiset of ranges S, there is a unique atomic range set of S, which we denote as $AR(S)$. Because of the maximality property of atomic range set, the candidate cut points correspond to the end points of ranges in $AR(S)$. We now show how to compute S-start points and S-end points. For any range $[x, y] \in S$, define the points $x - 1$ and y to be S-end points, and define the points x and $y + 1$ to be S-start points. Note that we ignore $x - 1$ if x is the minimum element of $\bigcup S$ and $y + 1$ if y is the maximum element of $\bigcup S$. Let $(s_1, \cdots, s_m)$ and $(e_1, \cdots, e_m)$ be the ordered list of S-start points and S-end points. It follows that for $1 \leq i \leq m - 1$ that $s_i \leq e_i = s_{i+1} + 1$. Thus, $AR(S) = \{[s_1, e_1], \cdots, [s_m, e_m]\}$.

For example, if we consider the three ranges in classifier $\mathbb{C}$ in example Figure 6(A), range $[0, 12]$ creates S-start point 13 and S-end point 12, range $[5, 15]$ creates S-end point 4 and S-start point 5, and range $[0, 15]$ creates no S-start points or S-end points. Finally, 0 is an S-start point and 15 is an S-end point. This leads to $AR(S) = \{[0, 4], [5, 12], [13, 15]\}$.

6.2.3 Choose Target Domain Size

We next choose the number of bits b used to encode domain $D'(F_1)$. This value b imposes constraints on legal prefix transformers. Consider $S = \{[0, 4], [0, 7], [0, 12], [0, 15]\}$ with $AR(S) = \{[0, 4], [5, 7], [8, 12], [13, 15]\}$. If $b = 2$, then the only legal prefix transformer maps $[0, 4]$ to 00, $[5, 7]$ to 01, $[8, 12]$ to 10, and $[13, 15]$ to 11. If $b = 3$, there are many more legal prefix transformers including one that maps $[0, 4]$ to 000, $[5, 7]$ to 001, $[8, 12]$ to $01*$, and $[13, 15]$ to $1**$. In this case, the second prefix transformer is superior to this first prefix transformer.

We include b as an input parameter to our prefix alignment problem. We initialize b as $\lceil \log_2 |AR(S)| \rceil$, the smallest possible value, and compute an optimal prefix alignment for this value of b. We then increment b and repeat until no improvement is seen. We choose a linear search as opposed to a binary search because computing the optimal solution for b bits requires an optimal solution for $b - 1$ bits.

6.2.4 Choose Optimal Cut Points

We now show how to compute the optimal cut points given b bits. We view a one-dimensional classifier $\mathbb{C}$ as a multiset of ranges S in $D(F_1)$ and formulate the prefix alignment problem as follows: *Given a multiset of ranges S over field F_1 and a number of bits b, find prefix transformer $\mathbb{T}_1$ such that the range expansion of the transformed multiset of ranges S' has the minimum number of prefix rules and $D'(F_1)$ can be encoded using only b bits.*

We present an optimal solution using dynamic programming. Given a multiset of ranges S, we first compute $AR(S)$. Suppose there are m atomic ranges $R_1, \cdots, R_m$ with S-start points s_1 through s_m and S-end points e_1 through e_m sorted in increasing order. For any S-start point s_x and S-end point s_y where $1 \leq x \leq y \leq m$, we define $S \Cap [x, y]$ to be the multiset of ranges from S that intersect range $[s_x, s_y]$; furthermore, we assume that each range in $S \Cap [x, y]$ is trimmed so that its start point is at least s_x and its end point is at most s_y. We then define a collection of subproblems as follows. For every $1 \leq x \leq y \leq m$, we define a prefix alignment problem $PA(x, y, b)$ where the problem is to find a prefix transformer $\mathbb{T}_1$ for $[s_x, e_y] \subseteq D(F_1)$ such that the range expansion of $(S \Cap [x, y])'$ has the smallest possible number of prefix rules and the transformed domain $D'(F_1)$ can be encoded in b bits. We use $cost(x, y, b)$ to denote the number of prefix rules in the range expansion of the optimal $(S \Cap [x, y])'$. The original prefix alignment problem then corresponds to $PA(1, m, b)$ where b can be arbitrarily large.

The prefix alignment problem obeys the optimal substructure property. For example, consider $PA(1, m, b)$. As we employ the divide and conquer strategy to locate a middle cut point that will establish what the prefixes $0\{*\}^{b-1}$ and $1\{*\}^{b-1}$ correspond to, there are $m - 1$ choices of cut points to consider: namely e_1 through e_{m-1}. Suppose the optimal cut point is e_k where $1 \leq k \leq m - 1$. Then the optimal solution to $PA(1, m, b)$ will build upon the optimal solutions to subproblems $PA(1, k, b-1)$ and $PA(k+1, m, b-1)$. That is, the optimal transformer for $PA(1, m, b)$ will simply append a 0 to the start of all prefixes in the optimal transformer for $PA(1, k, b - 1)$ and a 1 to the start of all prefixes in the optimal transformer for $PA(k + 1, m, b - 1)$. Moreover, $cost(1, m, b) = cost(1, k, b-1) + cost(k+1, m, b-1) - |S@[1, m]|$. We subtract $|S@[1, m]|$ in the above cost equation because ranges that include all of $[s_1, e_m]$ are counted twice, once in $cost(1, k, b-1)$ and once in $cost(k+1, m, b-1)$. However, as $[s_1, e_k]$ transforms to $0\{*\}^{b-1}$ and $[s_{k+1}, e_m]$ transforms to $1\{*\}^{b-1}$, the range $[s_1, e_m]$ can be expressed by one prefix $\{*\}^b = 0\{*\}^{b-1} \cup 1\{*\}^{b-1}$.

Based on this analysis, Theorem 6.2 shows how to compute the optimal cuts and binary cut tree. As stated earlier, the optimal prefix transformer $\mathbb{T}_1$ can then be computed from the binary cut tree.

THEOREM 6.2. *Given a multiset of ranges S with $|AR(S)| = m$, $cost(l, r, b)$ for any $b \geq 0, 1 \leq l \leq r \leq m$ can be computed as follows. For any $1 \leq l < r \leq m$, and $1 \leq k \leq m$, and $b \geq 0$:*

$$\text{cost}(l, r, 0) = \infty,$$
$$\text{cost}(k, k, b) = |S@[k, k]|,$$

and for any $1 \leq l < r \leq m$ *and* $b \geq 1$

$$\text{cost}(l, r, b) = \min_{k \in \{l, \ldots, r-1\}} \left(\begin{array}{c} \text{cost}(l, k, b-1) \\ + \\ \text{cost}(k+1, r, b-1) \\ - \\ |S@[l, r]| \end{array} \right) \square$$

Note that we set $\text{cost}(k, k, 0)$ to $|S@[k, k]|$ for the convenience of the recursive case. The interpretation is that with a 0-bit domain, we can allow only a single value in $D'(F_1)$; this single value is sufficient to encode the transformation of an atomic interval.

6.3 Multi-Dimensional Prefix Alignment

We now consider multi-dimensional prefix alignment. Unfortunately, while we can optimally solve the one-dimensional problem, there are complex interactions between the dimensions that complicate the multi-dimensional problem. In particular, the total range expansion required for each rule is the product of the range expansion required for each field. Thus, there may be complex tradeoffs where we sacrifice one field of a rule but align another field so that the costs do not multiply. The complexity of the multi-dimensional prefix alignment problem is currently unknown.

We present a hill-climbing solution where we iteratively apply our one-dimensional prefix alignment algorithm one field at a time. Because the range expansion of one field affects the numbers of ranges that appear in the other fields, we run prefix alignment for each field more than once. We stop when running prefix alignment in each field fails to improve the solution. More precisely, for a classifier $\mathbb{C}$ over fields $F_1, \ldots, F_d$, we first create d identity prefix transformers $\mathbb{T}_1^0, \ldots, \mathbb{T}_d^0$. We define a *multi-field prefix alignment iteration* k as follows. For i from 1 to d, generate the optimal prefix transformer $\mathbb{T}_i^k$ assuming the prefix transformers for the other fields are $\{\mathbb{T}_1^{k-1}, \ldots, \mathbb{T}_{i-1}^{k-1}, T_{i+1}^{k-1}, \ldots, T_d^{k-1}\}$. Our iterative solution starts at $k = 1$ and preforms successive multi-field prefix alignment iterations until no improvement is found for any field.

6.4 Composing with Domain Compression

While domain compression and prefix alignment can be used individually, they can be easily combined to achieve superior compression. Given a classifier $\mathbb{C}$ over fields $F_1, \ldots, F_d$, we first perform domain compression resulting in a transformed classifier $\mathbb{C}'$ and d transformers $\mathbb{T}_1^{dc}, \ldots, \mathbb{T}_d^{dc}$; then, we perform prefix alignment on the classifier $\mathbb{C}'$ resulting in a transformed classifier $\mathbb{C}''$ and d transformers $\mathbb{T}_1^{pa}, \ldots, \mathbb{T}_d^{pa}$. To combine the two transformation processes into one, we merge each pair of transformers $\mathbb{T}_i^{dc}$ and $\mathbb{T}_i^{pa}$ into one transformer $\mathbb{T}_i$ for $1 \leq i \leq d$. We apply the optimal algorithm in [27] to compute the minimum possible transformers $\mathbb{T}_i$ for $1 \leq i \leq d$. When running prefix alignment after domain compression, computing the atomic ranges and candidate cut points is unnecessary because each point $x \in D'(F_i)$ for $1 \leq i \leq d$ belongs to its own equivalence class in $D'(F_i)$ which implies $[x, x]$ is an atomic range.

7. DISCUSSION

7.1 TCAM Update

Packet classification rules periodically need to be updated. The common practice for updating rules is to run two TCAMs in tandem where one TCAM is used while the other is updated [14]. All our approaches are compatible with this current practice. Because our algorithms are efficient and the resultant TCAM lookup tables are small, updating TCAM tables can be efficiently performed.

If an application requires very frequent rule updates (at a frequency less than a second, for example), we can handle such updates in a batch manner by chaining the TCAM chips in our proposed architecture after a TCAM chip of normal width (144 bits), which we call the "hot" TCAM chip. When a new rule comes, we add the rule to the top of the hot TCAM chip. When a packet comes, we first use the packet as the key to search in the hot chip. If the packet has a match in the hot chip, then the decision of the first matching rule is the decision of the packet. Otherwise, we feed the packet to the TCAM chips in our architecture described as above to find the decision for the packet. Although the lookup on the hot TCAM chip adds a constant delay to per packet latency, the throughput can be much improved by pipelining the hot chip with other TCAM chips. Using batch updating, we only need to run our topological transformation algorithms to recompute the TCAM lookup tables when the hot chip is about to fill up.

7.2 Rule Logging

Packet classifiers sometimes allow rule logging; that is, recording the packets that match some particular rules. Our algorithm handles rule logging by assigning each rule that is logged a unique decision. Our experiments show that even when all rules in a classifier have unique decisions, our algorithm still achieves significant TCAM space reduction.

8. EXPERIMENTAL RESULTS

We evaluate the effectiveness and efficiency of our topological transformation approaches on both real-world and synthetic classifiers. Although our two approaches can be used independently, they are much more effective when used together. Thus, we only report results for both techniques combined, and we finish by running the redundancy removal algorithm in [16] on the transformed classifier $\mathbb{C}''$.

8.1 Evaluation Metrics

Given a TCAM optimization algorithm A and a classifier $\mathbb{C}$, let $A(\mathbb{C})$ denote the resulting classifier, $W(A(\mathbb{C}))$ denote the number of bits to represent each rule in $A(\mathbb{C})$, $TW(A(\mathbb{C}))$ denote the minimum TCAM entry width for storing $A(\mathbb{C})$ given choices 36, 72, 144, or 288, $|A(\mathbb{C})|$ denote the number of rules in $A(\mathbb{C})$, and $B(A(\mathbb{C})) = TW(A(\mathbb{C})) \times |A(\mathbb{C})|$, which represents the total number of TCAM bits required to store $A(\mathbb{C})$. The main goal of TCAM optimization algorithms is to minimize $B(A(\mathbb{C}))$. We use *Direct* to denote direct range expansion algorithm, so $B(\mathit{Direct}(\mathbb{C}))$ represents the baseline we compare against, $W(\mathit{Direct}(\mathbb{C})) = 104$, $TW(\mathit{Direct}(\mathbb{C})) = 144$, and $B(\mathit{Direct}(\mathbb{C})) = 144 \times |\mathit{Direct}(\mathbb{C})|$. Below is the summary of our notations:

For any A and $\mathbb{C}$, we measure overall effectiveness by the compression ratio $CR(A(\mathbb{C})) = \frac{B(A(\mathbb{C}))}{B(\mathit{Direct}(\mathbb{C}))}$. To isolate the

A	TCAM opt. scheme
Direct	direct range expansion
$\mathbb{C}$	packet classifier
$A(\mathbb{C})$	resulting classifier
$W(A(\mathbb{C}))$	width of rules in $A(\mathbb{C})$
$\|A(\mathbb{C})\|$	number of rules in $A(\mathbb{C})$
$TW(A(\mathbb{C}))$	minimum TCAM width for rules in $A(\mathbb{C})$
$B(A(\mathbb{C}))$	$TW(A(\mathbb{C})) \times \|A(\mathbb{C})\|$, total bits of $A(\mathbb{C})$

Figure 7: Summary of notation

factors that contribute to the success of our approaches at compressing classifiers, we define the *Rule Number Ratio* of A on $\mathbb{C}$ to be $RNR(A(\mathbb{C})) = \frac{|A(\mathbb{C})|}{|\mathbb{C}|}$, which is often referred to as *expansion ratio*, and the *Rule Width Ratio* of A on $\mathbb{C}$ to be $RWR(A(\mathbb{C})) = \frac{W(A(\mathbb{C}))}{104}$. When we consider a set of classifiers S where $|S|$ denotes the number of classifiers in S, we generalize our metrics as follows. *Average compression ratio* of A for S is $CR(A(S)) = \frac{\sum_{\mathbb{C}\in S} CR(A(\mathbb{C}))}{|S|}$, *average rule number ratio* of A for S is $RNR(A(S)) = \frac{\sum_{\mathbb{C}\in S} RNR(A(\mathbb{C}))}{|S|}$, and *average rule width ratio* of A for S is $RWR(A(S)) = \frac{\sum_{\mathbb{C}\in S} RWR(A(\mathbb{C}))}{|S|}$

We use RL to denote a set of 25 real-world packet classifiers that we performed experiments on. RL is chosen from a larger set of real-world classifiers obtained from various network service providers, where the classifiers range in size from a handful of rules to thousands of rules. We eliminated structurally similar classifiers from RL because similar classifiers exhibited similar results. We created RL by randomly choosing a single classifier from each set of structurally similar classifiers. We then split RL into two groups, RLa and RLb where $RNR(Direct(\mathbb{C})) \leq 2$ for all $\mathbb{C} \in RLa$ and $RNR(Direct(\mathbb{C})) > 40$ for all $\mathbb{C} \in RLb$. We have no classifiers where $2 \leq RNR(Direct(\mathbb{C})) \leq 40$. It turns out $|RLa| = 12$ and $|RLb| = 13$. By separating these classifiers into two groups, we can determine how well our techniques work on classifiers that do suffer significantly from range expansion as well as those that do not.

Because packet classifiers are considered confidential due to security concerns making it difficult to acquire a large quantity of real-world classifiers, we generated a set of synthetic classifiers SYN with the number of rules ranging from 250 to 8000. The predicate of each rule has five fields: source IP, destination IP, source port, destination port, and protocol. We based our generation method upon Singh *et al.*'s [24] model of synthetic rules. We also performed experiments on TRS, a set of 490 classifiers produced by Taylor&Turner's Classbench [29]. These classifiers were generated using the parameters files downloaded from Taylor's web site `http://www.arl.wustl.edu/~det3/ClassBench/index.htm`. To represent a wide range of classifiers, we chose a uniform sampling of the allowed values for the parameters of smoothness, address scope, and application scope.

To stress test the sensitivity of our algorithms to the number of decisions in a classifier, we created a set of classifiers RL_U (and thus RLa_U and RLb_U) by replacing the decision of every rule in each classifier by a unique decision. Similarly, we created the set SYN_U. Thus, each classifier in RL_U (or SYN_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.

8.2 Effectiveness

8.2.1 Results on real-world and synthetic classifiers

Table 2 shows the average compression ratio, rule size ratio, and rule number ratio for our algorithm on all eight data sets. Figures 8 through 13 show the specific compression ratios, rule width ratios, and rule number ratios for all of our real-world classifiers; the black bars represent the increases in each quantity that arise from assigning each rule a unique decision. In each figure, we sort the classifiers by the number of rules in the original classifier. We present compression ratio and rule number ratio data with and without transformers. The data without transformers facilitate comparison with most previous reencoding schemes. The data with transformers depicts the true space savings of our methods.

	compression		rule width	rule number	
	w.o. T	with T		w.o. T	with T
RL	2.6%	10.3%	10.6%	27.6%	123.3%
RL_U	7.0%	16.2%	14.5%	63.2%	176.8%
RLa	5.3%	20.8%	14.2%	22.7%	87.4%
RLa_U	14.4%	33.1%	18.5%	62.5%	140.3%
RLb	0.1%	0.5%	7.2%	32.2%	156.4%
RLb_U	0.2%	0.6%	10.8%	63.8%	210.6%
SYN	0.6%	2.5%	10.4%	2.7%	11.8%
SYN_U	9.3%	12.4%	16.0%	43.9%	58.9%
TRS	1.0%	2.7%	15.7%	9.7%	23.3%

Table 2: Average compression ratio, rule width ratio, and rule number ratio for 9 data sets (with transformers included and excluded)

Our algorithm achieves significant compression on both real-world and synthetic classifiers. On RL, our algorithm achieves an average compression ratio of 10.3% if we count TCAM space for transformers and 2.6% if we do not. These savings are attributable to both rule width and rule number compression. The average rule width compression ratio is 10.6%, which means that a typical encoded classifier only requires 11 bits, instead of 104 bits, to store a rule. However, the actual savings that rule width compression contributes to average compression ratio is only 25% because the encoded classifiers will always use 36 bit wide TCAM entries of 36, which is the smallest possible TCAM width. In comparison, direct range expansion would use 144 bit wide TCAM entries. That is, $TW(A(\mathbb{C})) = 32$ for all the classifiers in RL (actually in all data sets including RL_U, SYN, and SYN_U). The remaining savings is due to rule number compression. Note that the average rule number compression ratio without transformers is 27.6%; that is, domain compression and redudancy removal eliminate an average of 72% of the rules from our real-life classifier sets. In comparison, the goal of all other reencoding schemes is an average rule number compression ratio without transformers of 100%. On other data sets, our algorithm also performs well. For example, for Taylor's rule set TRS, we achieve an average compression ratio of 2.7% with transformers included and 1.0% with transformers excluded.

8.2.2 Sensitivity to classifier efficiency

Our algorithm is effective for both efficiently specified classifiers and inefficiently specified classifiers. The efficiently

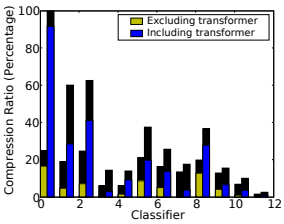


Figure 8: Compression ratio of RLa and RLa_U

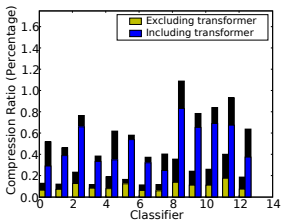


Figure 9: Compression ratio of RLb and RLb_U

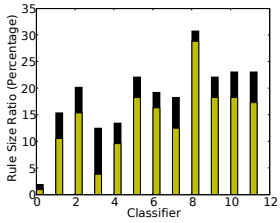


Figure 10: Rule size ratio of RLa and RLa_U

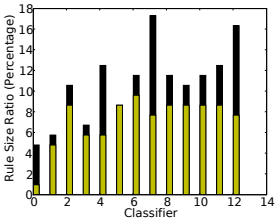


Figure 11: Rule size ratio of RLb and RLb_U

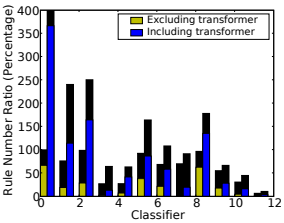


Figure 12: Rule number ratio of RLa and RLa_U

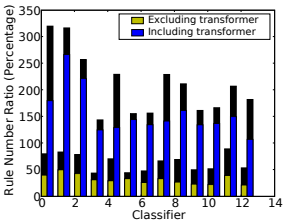


Figure 13: Rule number ratio of RLb and RLb_U

specified classifiers in RLa experience relatively little range expansion; the inefficiently specified classifiers in RLb experience significant range expansion. Not surprisingly, our algorithm provides roughly 40 times better compression for RLb than for RLa with average compression ratios of 0.5% and 20.8%, respectively. In both sets, TCAM width compression contributes 25% savings. The difference is rule number compression. On efficient classifiers, our algorithm provides modest rule number compression (even though the average rule number ratio without transformers for RLa is 22.7%). On inefficient classifiers, our algorithm provides tremendous rule number compression.

8.2.3 Sensitivity to number of unique decisions

Our algorithm's effectiveness is only slightly diminished 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, our algorithm achieves an average compression ratio of 16.2% with transformers included and 7.0% with transformers excluded; and on SYN_U, our algorithm achieves an average compression ratio of 12.4% with transformers included and 9.3% with transformers excluded. In particular, the TCAM width is unaffected as our algorithm still uses 36 bit wide TCAM entries.

8.2.4 Comparison with state-of-the-art results

Our algorithm outperforms all existing reencoding schemes by at least a factor of 3.16 including transformers and by at least a factor of 7.24 excluding transformers. We first consider the width of TCAM entries. We have 36 bit wide TCAM entry width while the smallest TCAM width achieved by prior work is 72 [22, 23]. Therefore, on TCAM entry width, our algorithm is 2 times better than the best known result. Next, we consider the number of TCAM entries. Excluding TCAM entries for transformers, the best rule number ratio that any other method can achieve on RL is 100% whereas we achieve 27.6%. *Therefore, excluding TCAM entries for transformers, our algorithm is at least* 7.24 ($= 2 \times 100\%/27.6\%$) *times better than the optimal TCAM reencoding algorithm that does not consider classifier semantics.* In comparison with PIC [22, 23], the best known TCAM-based reencoding algorithm, the transformers in PIC use at least the same number of TCAM entries as our algorithm because our domain compression technique may map multiple intervals to one decision whereas PIC maps each interval to a unique decision. Thus, including TCAM entries for transformers, the best average rule number ratio that PIC can achieve on RL is 195.7%($= 123.3\% - 27.6\% + 100\%$). *Therefore, including TCAM entries for transformers, our algorithm is at least* 3.16 ($= 2 \times 195.7\%/123.3\%$) *times better than PIC.*

Our algorithm also significantly outperforms prior classifier compression schemes. In [20], we demonstrated that TCAM Razor outperforms redundancy removal [16–18] and Dong's algorithm [7], although Dong's algorithm was implemented on a different set of packet classifiers. Thus, we compare our algorithm to TCAM Razor and a new classifier compression scheme, Bit Weaving [21]. On the same 25 real-world classifiers, TCAM Razor [20], Bit Weaving [21], and our algorithm achieve average compression ratios of 24.5%, 23.6%, and 10.3%, respectively.

8.3 Efficiency

We implemented our algorithms on the Microsoft .Net framework 2.0 and performed our experiments on a desktop PC running Windows XP with 3G memory and a single 3.4 GHz Pentium D processor. On RL, the minimum, mean, median, and maximum running times were 0.000, 0.060, 0.005, and 0.904 seconds; on RL_U, the minimum, mean, median, and maximum running times were 0.001, 0.316, 0.016, and 4.696 seconds. Table 3 shows running time of some representative classifiers in RL and RL_U. On synthetic rules, the running time grows linearly with the number of rules in a classifier, where the average running time for classifiers of 8000 rules is 4.0 seconds.

# Rules	Time (sec.)	Time (sec.) with unique decisions
511	0.4	2.1
1183	1.2	4.5
1308	1.0	8.0
3928	0.7	7.1

Table 3: Running time (5 classifiers in RL and RL_U)

9. CONCLUSIONS AND FUTURE WORK

We make three major contributions in this paper. First, we propose a novel topological view of the TCAM reencoding process where we consider the semantics of the packet classifier. Second, we present two techniques, domain compression and prefix alignment, for realizing such a view. These techniques are not only composable, they can be composed with other TCAM optimization and reencoding schemes proposed. Third, we implemented our algorithms and conducted extensive experiments on both real-life and synthetic packet classifiers. The experimental results show that our

techniques achieve at least 7.2 times more space reduction with transformers excluded and at least 3.16 times more space reduction with transformers included.

Our work opens up new problems for future research. One problem is to find an optimal choice of landmarks for each equivalence class in the domain compression technique that leads to the smallest final classifier. Another is to find an optimal solution to the multi-dimensional prefix alignment problem or prove it is NP-hard. We also plan to study more potential combinations of our techniques with other TCAM optimization and reencoding schemes.

Acknowledgement

The work of Alex X. Liu is supported in part by the National Science Foundation under Grant No. CNS-0716407.

10. REFERENCES

[1] Cypress semiconductor. http://www.cypress.com/.

[2] Integrated device technology. http://www.idt.com/.

[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 Symposium on Discrete Algorithms (SODA)*, January 2007.

[4] J. Bolaria and L. Gwennap. A guide to search engines and networking memory. *http://www.linleygroup.com*, 2004.

[5] A. Bremler-Barr and D. Hendler. Space-efficient TCAM-based classification using gray coding. In *Proc. 26th Annual IEEE Conf. on Computer Communications (Infocom)*, May 2007.

[6] H. Che, Z. Wang, K. Zheng, and B. Liu. DRES: Dynamic range encoding scheme for tcam coprocessors. *IEEE Transactions on Computers*, 57(7):902–915, July 2008.

[7] Q. Dong, S. Banerjee, J. Wang, D. Agrawal, and A. Shukla. Packet classifiers in ternary CAMs can be smaller. In *Proc. Sigmetrics*, pages 311–322, 2006.

[8] R. Draves, C. King, S. Venkatachary, and B. Zill. Constructing optimal IP routing tables. In *Proc. IEEE INFOCOM*, pages 88–97, 1999.

[9] M. G. Gouda and A. X. Liu. Firewall design: consistency, completeness and compactness. In *Proc. 24th IEEE Int. Conf. on Distributed Computing Systems (ICDCS-04)*, pages 320–327, March 2004.

[10] P. Gupta and N. McKeown. Packet classification on multiple fields. In *Proc. ACM SIGCOMM*, pages 147–160, 1999.

[11] P. Gupta and N. McKeown. Algorithms for packet classification. *IEEE Network*, 15(2):24–32, 2001.

[12] T. V. Lakshman and D. Stiliadis. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. In *Proc. ACM SIGCOMM*, pages 203–214, 1998.

[13] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary. Algorithms for advanced packet classification with ternary CAMs. In *Proc. ACM SIGCOMM*, pages 193 – 204, August 2005.

[14] P. C. Lekkas. *Network Processors - Architectures, Protocols, and Platforms*. McGraw-Hill, 2003.

[15] A. X. Liu and M. G. Gouda. Diverse firewall design. In *Proc. Int. Conf. on Dependable Systems and Networks (DSN-04)*, pages 595–604, June 2004.

[16] A. X. Liu and M. G. Gouda. Complete redundancy detection in firewalls. In *Proc. 19th Annual IFIP Conf. on Data and Applications Security, LNCS 3654*, pages 196–209, August 2005.

[17] A. X. Liu and M. G. Gouda. Complete redundancy removal for packet classifiers in tcams. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, to appear. The conference version was published in the 19th Annual IFIP Conf. on Data and Applications Security (DBSec), LNCS 3654, 2005.

[18] A. X. Liu, C. R. Meiners, and Y. Zhou. All-match based complete redundancy removal for packet classifiers in TCAMs. In *Proc. 27th Infocom*, 2008.

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

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

[21] C. R. Meiners, A. X. Liu, and E. Torng. Bit weaving: A non-prefix approach to compressing packet classifiers in tcams. Technical Report MSU-CSE-09-1, Department of Computer Science and Engineering, Michigan State University, January 2009.

[22] D. Pao, Y. Li, and P. Zhou. Efficient packet classification using TCAMs. *Computer Networks*, 50(18):3523–3535, 2006.

[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 Computers & Digital Techniques*, 1(5):572–580, September 2007.

[24] S. Singh, F. Baboescu, G. Varghese, and J. Wang. Packet classification using multidimensional cutting. In *Proc. ACM SIGCOMM*, pages 213–224, 2003.

[25] E. Spitznagel, D. Taylor, and J. Turner. Packet classification using extended TCAMs. In *Proc. 11th IEEE Int. Conf. on Network Protocols (ICNP)*, pages 120– 131, November 2003.

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

[27] S. Suri, T. Sandholm, and P. Warkhede. Compressing two-dimensional routing tables. *Algorithmica*, 35:287–300, 2003.

[28] D. E. Taylor. Survey & taxonomy of packet classification techniques. *ACM Computing Surveys*, 37(3):238–275, 2005.

[29] D. E. Taylor and J. S. Turner. Classbench: A packet classification benchmark. In *Proc. Infocom*, 2005.

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

[31] 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. Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 105–113, October 2005.

[32] K. Zheng, H. Che, Z. Wang, B. Liu, and X. Zhang. DPPC-RE: TCAM-based distributed parallel packet classification with range encoding. *IEEE Transactions on Computers*, 55(8):947–961, August 2006.