Packet Classification Using Binary Content Addressable Memory

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

***Abstract*—Packet classification is the core mechanism that enables many networking devices. Although using Ternary Content Addressable Memories (TCAMs) to perform high speed packet classification has become the widely adopted solution, TCAMs are very expensive, have limited capacity, consume large amounts of power, and generate tremendous amounts of heat because of their extremely dense and parallel circuitry. In this paper, we propose the first packet classification scheme that uses Binary Content Addressable Memories (BCAMs). BCAMs are similar to TCAMs except that in BCAMs, every bit has only two possible states: 0 or 1; in contrast, in TCAMs, every bit has three possible states: 0, 1, or * (*don't care*). Because of the high complexity in implementing the extra "*don't care*" state, TCAMs have much higher circuit density than BCAMs. As the power consumption, heat generation, and price grow non-linearly with circuit density, BCAMs consume much less power, generate much less heat, and cost much less money than TCAMs. Our BCAM based packet classification scheme is built on two key ideas. First, we break a multi-dimensional lookup into a series of one-dimensional lookups. Second, for each one-dimensional lookup, we convert the ternary matching problem into a binary string exact matching problem. To speed up the lookup process, we propose a number of optimization techniques including skip lists, free expansion, minimizing maximum lookup time, minimizing average lookup time, and lookup short circuiting. We evaluated our BCAM scheme on 17 real-life packet classifiers. On these classifiers, our BCAM scheme requires roughly 5 times fewer CAM bits than the traditional TCAM based scheme. The penalty is a throughput that is roughly 4 times less.**

I. Introduction

A. TCAM-Based Packet Classification

Packet classification is the core mechanism that enables many networking devices, such as routers and firewalls, to perform services such as packet filtering and traffic accounting, network address translation (NAT), quality of service (QoS), load balancing, virtual private networks (VPNs) 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 I shows an example packet classifier of two rules. The format of these rules is based upon the format used in Access Control Lists (ACLs) on Cisco routers. Note that we use the terms *packet classifiers*, *ACLs*, *rule lists*, and *lookup tables* interchangeably.

Packet classification is often the performance bottleneck for Internet routers as they need to classify every packet on the wire. Achieving wire speed packet classification has long been a networking goal. Ternary Content Addressable Memory (TCAM) based packet classification has been the widely adopted solution. Unlike traditional random access memory chip receives an address and returns the content of the memory at that address, a TCAM chip 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 the content, depending on the chip architecture and configuration) of the first matching entry.

Rule	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

TABLE I
AN EXAMPLE PACKET CLASSIFIER

Unfortunately, TCAM's high speed comes with several costs. To understand the costs of using TCAM, we must first understand TCAM chip architecture. A TCAM chip consists of many TCAM core cells and other supporting circuits and hardware. A TCAM core cell, the building block of a TCAM chip, serves two basic functions: bit storage and bit comparison. There are two types of TCAM core cells: a NOR-type TCAM cell and a NAND-type TCAM cell. A NOR-type TCAM cell consists of four comparison transistors and two SRAM cells that have a total twelve transistors. A NAND-type TCAM cell consists of four comparison transistors and two SRAM cells that have a total twelve transistors. Furthermore, each memory access searches all active TCAM memory implying TCAM power consumption is proportional to the number of bits searched.

These architectural constraints have several implications on TCAM deployment. First, due to their large footprint and large number of transistors, TCAM is *very expensive*, costing hundreds of dollars even in large quantities. TCAM chips often

978-14799-3360-0/14/$31.00 ©2014 IEEE

cost more than network processors. Second, TCAM has *limited capacity*. The largest available TCAM chip has a capacity of 72Mb, while 2Mb and 1Mb chips are most popular. As rules stored in TCAM must be ternary, TCAM capacity limitations are exacerbated by the well known range expansion problem. That is, converting packet classification rules to ternary rules typically results in a much larger number of TCAM rules. In a packet classification 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 remaining two fields of source and destination port numbers are specified in ranges, which need to be converted to one or more prefixes before being stored in a TCAM. 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. Third, the large power consumption of TCAM chips constrains their deployment when system designers must operate within a "power budget", *e.g.*, TCAM components may only use 10% of an entire board's power budget. The explains that even though 72Mb TCAM chips are available, the 1-2Mb chips are most popular [1].

B. Proposed Approach: BCAM-based Packet Classification

In this paper, we B-CLASS-d, the first packet classification scheme that uses Binary Content Addressable Memory (BCAM). BCAM works similar to TCAM except that in BCAM, every bit has only two possible states: 0 or 1; in contrast, in TCAM, every bit has three possible states: 0, 1, or *. A BCAM core cell is simpler than a TCAM core cell because it only has a binary state because there is no need for the third "*don't care*" state. Specifically, there are two types of BCAM core cells: a NOR-type BCAM cell and a NAND-type BCAM cell. A NOR-type BCAM cell consists of four comparison transistors and an SRAM cell that has six transistors. A NAND-type BCAM cell consists of three comparison transistors and an SRAM cell that has six transistors. This means TCAM circuitry is about two times denser than BCAM circuitry. This implies a TCAM chip consumes roughly twice as much power and roughly twice as much board space as an equivalent BCAM chip. Considering the number of transistors and the associated circuits, the circuitry of a TCAM is about two times more dense than that of a BCAM.

C. Defining CAM Space and Throughput

We will compare our B-CLASS-d scheme with the existing standard TCAM scheme. Because the two schemes are very different architecturally, we must carefully define space and time for both schemes to provide a reasonable comparison. We define the space used by a packet classifier in a CAM chip as the number of classifier entries or rules multiplied by the width of the CAM chip in bits. Recall that each TCAM bit requires roughly twice as much circuitry as a BCAM bit. The traditional TCAM scheme must accommodate the 104 bits in the five packet fields. Given the limited number of possible TCAM entry widths, this means each entry typically has 144 bits. On the other hand, in B-CLASS-d, each table will typically hold only a single packet field. Thus, each entry will be much narrower. In most cases, each entry is 72 bits. We define the throughput to be the number of CAM bus cycles required to classify a packet. In the traditional scheme, we only require one TCAM lookup. However, the TCAM bus is typically only 72 bits wide. Thus, classifying a packet typically requires at least 2 TCAM bus cycles. In B-CLASS-d, since each BCAM entry is at most 72 bits wide, we will require only one BCAM bus cycle per lookup.

We implemented B-CLASS-d, our BCAM based scheme, and conducted experiments on 17 real-life packet classifiers. For our set of classifiers, classifier preprocessing time was at most a second for almost all the classifiers. First, we compare the CAM space required by B-CLASS-d and the traditional TCAM scheme. Our results indicate that B-CLASS-d with light optimization requires roughly 5 times less space than the traditional TCAM scheme. On top of that, B-CLASS-d uses cheaper BCAM bits rather than TCAM bits. It is true that B-CLASS-d has less throughput than the traditional TCAM scheme. In particular, the average of the maximum number of BCAM lookups required by B-CLASS-d on the 17 classifiers is 8.4. Given that the traditional scheme requires 2 TCAM bus cycles, we see that B-CLASS-d has a maximum throughput that is roughly 4 times slower than the traditional TCAM scheme.

The rest of the paper proceeds as follows. We start by reviewing previous work in Section II. In Section III, we present our basic scheme as well as optimization techniques for one-dimensional packet classification using BCAM. In Section IV, we present our BCAM classification scheme for multi-dimensional packet classifiers. Experimental results are presented in Section V. We draw conclusions in Section VI.

II. Related Work

To the best of our knowledge, there is no prior work on BCAM based packet classification. There is some prior work that explores ways to address the hardware limitations of TCAMs and cope with the well-known range expansion problem. Such work falls into three broad categories: (1) *classifier minimization* (*e.g.*, [3], [6], [8], [9], [15]–[19], [21], [22], [25], [27], [29]), which converts a given classifier to a semantically equivalent one that requires fewer TCAM entries; (2) *range encoding* (*e.g.*, [5], [12], [20], [23], [24], [31]), which encodes the range fields (*i.e.*, source port and destination port) in a manner that reduces range expansion; and (3) *circuit modification* (*e.g.*, [28]), which modifies TCAM circuits to accommodate range comparisons. We make use of a new TCAM SPliT approach [26] which we discuss in more detail in Section IV-A.

III. One-dimensional Packet Classification

In this section, we present B-CLASS, a scheme for performing one-dimensional packet classification using BCAM. This scheme B-CLASS will be used as the basic building block for performing multi-dimensional packet classification using BCAM, that we describe in the next section.

A. The Basic Scheme

We first consider the following prefix membership verification problem: *given a w-bit prefix P and a w-bit binary number B, how can we store P in a BCAM such that we can use the BCAM to determine whether B matches P (*i.e.*, $B \in P$)*? Our solution leverages the prefix membership verification scheme in [13]. For prefix $P = \{0,1\}^k\{*\}^{w-k}$ with k leading 0's and 1's followed by $w-k$ $*$'s, the length of P, denoted as $\mathbb{L}(P)$, is defined to be k. The key observation is that if binary string B matches prefix P, then the first $\mathbb{L}(P)$ bits of P and B are the same. For example, if $B \in 01**$ (i.e., $x \in [0100, 0111]$), then the first two bits of B must be 01. Given a w-bit binary number $B = b_1b_2\cdots b_w$, *the prefix family of* B, denoted as $\mathbb{PF}(B)$, is defined as the set of $w+1$ prefixes $\{b_1b_2\cdots b_w, b_1b_2\cdots b_{w-1}*, \cdots, b_1*\cdots*, **...*\}$, where the i-th prefix is $b_1b_2\cdots b_{w-i+1}*\cdots*$. For example, $\mathbb{PF}(0101) = \{0101, 010*, 01**, 0***, ****\}$. It follows naturally that for any binary number B and prefix P, $B \in P$ if and only if $P \in \mathbb{PF}(B)$.

To store a prefix P in BCAM, we need to convert P to a binary number. This process is called *prefix numericalization*. A prefix numericalization function $\mathbb{N}$ takes a prefix as the input and outputs a unique binary number. The important property that $\mathbb{N}$ needs to satisfy is that for any two prefixes P_1 and P_2, $\mathbb{N}(P_1) = \mathbb{N}(P_2)$ *if and only if* $P_1 = P_2$. We use the following prefix numericalization scheme: Given a w-bit prefix $b_1b_2\cdots b_k*\cdots*$, we first replace every * by 0; second, we append $\lceil \log(w+1) \rceil$ bits whose value is equal to k. For example, $\mathbb{N}(101*) = 1010011$. We use $\mathbb{N}(\mathbb{PF}(B))$ to denote the resulting set of binary numbers after numericalizing every prefix in $\mathbb{PF}(B)$. For example, $\mathbb{N}(\mathbb{PF}(0101)) = \{0101100, 0100011, 0100010, 0000001, 0000000\}$. Based on the above concepts, it follows that for any binary number B and prefix P, $B \in P$ if and only if $\mathbb{N}(P) \in \mathbb{N}(\mathbb{PF}(B))$.

Our 1-dimensional scheme B-CLASS is composed of two algorithms. The first algorithm describes how to preprocess a classifier so that it can be stored in BCAM. The second algorithm classifies a packet. We first describe the classifier preprocessing. Given a 1-dimensional classifier C, we first convert it to an equivalent *minimum prefix classifier* C' using algorithms in [9], [29]. By minimum, we mean that no equivalent prefix classifier has strictly fewer rules than a minimum prefix classifier. Second, for each prefix P in C', we convert P to a binary number using the prefix numericalization described above. The preprocessing result $\mathbb{N}(C')$ is stored in a BCAM.

We then classify packets using the following algorithm. Given a packet represented as a w-bit binary number $B = b_1b_2\cdots b_w$, we first generate its prefix family $\mathbb{PF}(B)$. In particular, we generate the prefixes in $\mathbb{PF}(B)$ in the deceasing order of prefix length. We use $\mathbb{OPF}(B)$ to denote this ordered prefix family; that is, $\mathbb{OPF}(B) = \langle b_1b_2\cdots b_w, b_1b_2\cdots b_{w-1}*, \cdots, b_1*\cdots*, **...*\rangle$. Second, we numericalize every prefix in $\mathbb{OPF}(B)$ and get $\mathbb{N}(\mathbb{OPF}(B))$. The search process starts by testing whether the first element of $\mathbb{N}(\mathbb{OPF}(B))$ (*i.e.*, $\mathbb{N}(b_1b_2\cdots b_w)$) is in the BCAM. If yes, then return the corresponding decision; otherwise, continue to test whether the second element of $\mathbb{N}(\mathbb{OPF}(B))$ is in the BCAM. This process proceeds until the BCAM returns a match. Figure 1 illustrates B-CLASS in action.

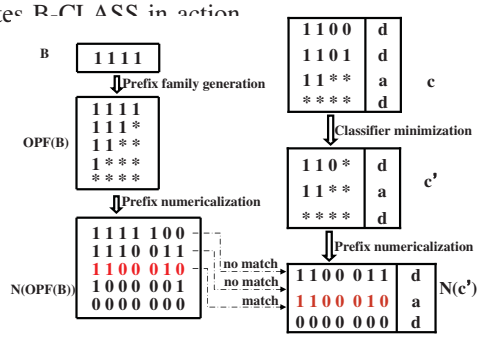


Fig. 1. Example of B-CLASS on a packet

B. Skip Lists

The idea of skip lists is based on two key observations. The first observation is that the prefix family generated from a packet contains prefixes of every length from 0 to w, but the minimum classifier converted from the input classifier often does not. For example, the minimum classifier C' in Figure 1 only contains prefixes of lengths 3, 2, and 0. Therefore, when searching for a packet B, we can ignore prefixes of B that have no equal length prefix in the minimum classifier. For the example in Figure 1, it is unnecessary to search prefixes 1111 and $1***$ because the minimum classifier C' does not have prefixes of length 4 or 1.

The second observation is that the rules in a minimum prefix classifier can be sorted in decreasing order of their prefix length without changing the semantics of the minimum classifier. More formally, for any 1-dimensional minimum prefix classifier C, let C^s denote the prefix classifier formed by sorting all the rules in C in decreasing order of prefix length.

C. Free Expansion

We first examine BCAM lookup times for B-CLASS with skip lists. Given a sorted minimal prefix classifier C, we use $\mathbb{LIST}(C)$ to denote the skip list built for C. Let k be the number of elements in $\mathbb{LIST}(C)$. We use PS_i $(0 \leqslant i \leqslant k-1)$ to denote the set of prefixes in C whose length is $\mathbb{LIST}(C)[i]$. More precisely, $PS_i = \{P | \exists r \in C \ s.t. \ \mathbb{P}(r) = P, and \ \mathbb{L}(P) = \mathbb{LIST}(C)[i]\}$. We define the set of all prefixes in PS_i to be *layer* i for classifier C. For any prefix P in PS_i $(0 \leqslant i \leqslant k-1)$ and for any packet p that will match P, the number of BCAM lookups is $i+1$. Consider the example in Figure 1. For packet $p = 1110$, the number of BCAM lookups is 2.

The idea of free expansion is based on two observations. First, a prefix in PS_i $(1 \leqslant i \leqslant k-1)$ can be replaced by a set

of $2^{\mathbb{LIST}(C)[i-1]-\mathbb{LIST}(C)[i]}$ prefixes of length $\mathbb{LIST}(C)[i-1]$. Consider the example in Figure 1. We can expand the prefix $11**$ into $110*$ and $111*$ (*i.e.*, replace the rule $11** \rightarrow a$ by two rules $110* \rightarrow a$ and $111* \rightarrow a$). After expanding a prefix $P \in PS_i$ $(1 \leqslant i \leqslant k-1)$, for all packets that match P, the number of BCAM lookups is reduced from $i+1$ to i.

The second observation is that after replacing a prefix P in PS_i $(1 \leqslant i \leqslant k-1)$ by a set S of $2^{\mathbb{LIST}(C)[i-1]-\mathbb{LIST}(C)[i]}$ prefixes of length $\mathbb{LIST}(C)[i-1]$, some rules with prefixes in S may become upward redundant. Consider the example in Figure 1. After expanding the prefix $11**$ into $110*$ and $111*$, the rule $110* \rightarrow a$ becomes upward redundant because of the first rule.

IV. Multi-dimensional Packet Classification

Our multi-dimensional BCAM packet classification scheme B-CLASS-d is based on the TCAM SPliT approach [26]. In this section, we first give a brief overview of TCAM SPliT. Then, we introduce B-CLASS-d, and we conclude with two optimizations for B-CLASS-d. Note, we now assume that B-CLASS includes skip lists and free expansion.

A. TCAM SPliT

TCAM SPliT is a TCAM based packet classification scheme that can achieve high TCAM space efficiency. The key observation is that the rules in a classifier often share a number of the same field values [4], [11], [30], [32]. For example, the prefix classifier in Figure 2(a) has only 4 distinct values for field F_1. Such information redundancy can lead to a multiplicative increase in the number of rules in a classifier as cross producting of field values to form rules is often necessary. This problem is exacerbated in TCAM packet classifiers by range expansion in the source and destination port fields.

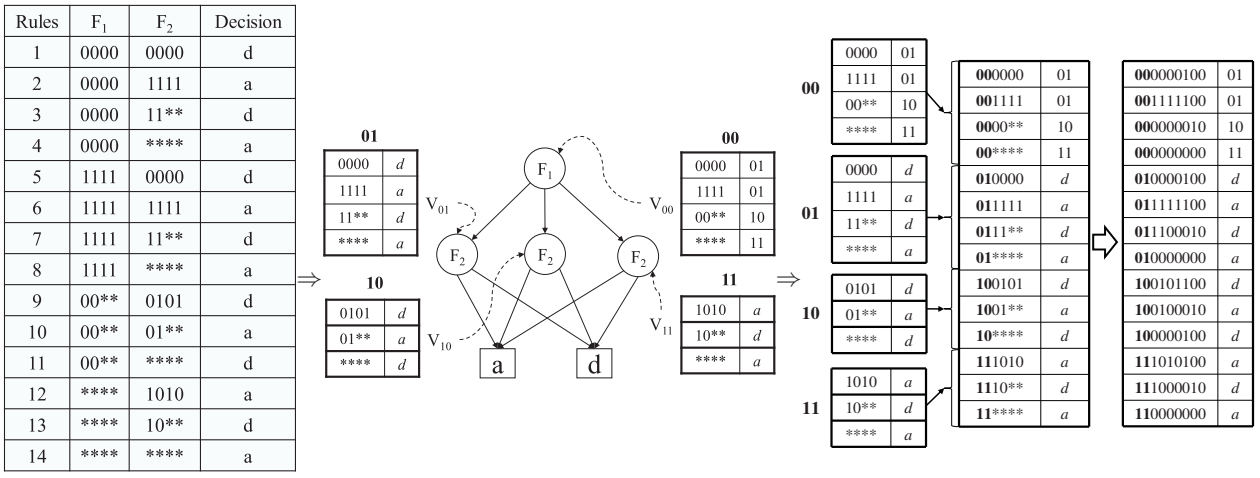
The basic idea of TCAM SPliT is to reduce the multiplicative effects of cross producting by decomposing *a d-dimensional packet classifier* into *d one dimensional classifiers*. This requires replacing *a single d-dimensional lookup* by a sequence of *d one-dimensional lookups* that are pipelined to improve classification throughput. Intuitively, TCAM SPliT is essentially an optimized version of the software-based decision tree packet classification technique. We create a decision diagram of small one-dimensional classifiers where each of these small classifiers will be implemented in TCAM. TCAM SPliT is attractive because the total TCAM space required to store these small and thin tables is significantly less than the TCAM space required to store a single d-dimensional table. For our purpose, the key feature of TCAM SPliT is that it converts a multi-dimensional lookup into a series of one-dimensional lookups, which allow us to the B-CLASS scheme presented in the previous section.

We create the decision tree structure by first converting the given classifier to an equivalent reduced firewall decision diagram [10]. 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 $F(v) \in \{F_1, \cdots, F_d\}$ if v is a nonterminal node and $F(v) \in DS$ if v is a terminal node. (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. An FDD construction algorithm, which converts a packet classifier to an equivalent full-length ordered FDD, is in [14]. For ease of presentation, in the rest of this proposal, we use the term "FDD" to mean "full-length ordered FDD" if not otherwise specified.

A *reduced* FDD f 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 brute force deep comparison algorithm for FDD reduction was proposed in [10]; however, it is not efficient. A more efficient FDD reduction algorithm that uses a novel fingerprinting technique to speed up node comparison was proposed in [2].

The second step of TCAM SPliT is to generate a one-dimensional lookup table from each nonterminal node in the FDD. Because of the completeness property of an FDD, each nonterminal node can be viewed as a one-dimensional classifier. For each such one-dimensional classifier, we apply the optimal polynomial-time algorithm for minimizing one-dimensional prefix classifiers in [29]. For example, Figure 2(b) shows the four minimal one-dimensional tables generated from the four nonterminal nodes in the FDD.

The third step of TCAM SPliT is to combine the one-dimensional tables together. There are two options for merging tables, which are designed for two classification architectures, multi-lookup and pipelined-lookup, respectively. For the multi-lookup architecture, we combine all the one-dimensional tables together to form a single table. Let m be the total number of one-dimensional tables. For each one-dimensional table, we assign a table ID of $\lceil \log m \rceil$ bits. For any table entry whose decision is a nonterminal node v, we replace v by v's corresponding table ID. Then, for every nonterminal node v, we prepend v's corresponding table ID to each entry in the table generated from v. After the multi-lookup TCAM table is built and stored in a TCAM for a d-dimensional packet classifier, the decision for a d-dimensional packet $(p_1, \ldots, p_d)$



(a) Prefix Classifiers (b) Reduced FDD and 1-D tables (c) Combine and numericalize 1-D tables

Fig. 2. Overview of the BCAM sequential decomposition

can be found by performing d searches on the TCAM. The first search key k_1 is formed by concatenating the root node's ID and p_1. Let $f(k_1)$ denote the search result of k_1. The second search key, k_2 is formed by concatenating $f(k_1)$ and p_2. This process continues until we compute $f(k_d)$, which is the decision for the packet. For example, given the two dimensional multi-lookup prefix table in 2c and a packet $(0011, 0010)$, the first search key is 000011, which returns 10. The second search key is 100010, which returns *d* (*i.e.*, discard) as the decision for the packet.

For the pipelined-lookup architecture, we only combine the tables of the same field into a single table. Let m_i be the number of nodes with label F_i in the reduced FDD. The ID assigned to each F_i node consists of $\lceil \log m_i \rceil$ bits. Similarly, for any table entry whose decision is a nonterminal node v, we replace v by v's corresponding table ID; for every nonterminal node v, we prepend v's corresponding table ID to each entry in the table generated from v. The resulting d tables are stored in d TCAM chips, and the d chips are chained together into a pipeline such that the search result of the i-th chip is part of the search key for the $(i+1)$-th chip. The result of the last chip is the decision for the packet. With such a chain, d packets can be processed in parallel in the pipeline.

B. B-CLASS-d: d-Dimensional BCAM Packet Classification

B-CLASS-d is built upon B-CLASS and the above multi-lookup TCAM SPliT scheme for TCAM based packet classification. We make two modifications to the single multi-lookup table. First, we numericalize every entry in the table. Second, for every entry in the table, we store the index of the corresponding skip list of the one-dimensional table that the entry belongs to. The skip lists of all one-dimensional tables that are stored in the SRAM associated with the BCAM. Figure 2(c) shows the multi-lookup BCAM table with skip list indexes omitted. We next formally analyze packet lookup time.

Given a reduced FDD and the minimal one-dimensional tables generated from nonterminal nodes, for any packet p, we can calculate the lookup time (*i.e.*, the number of BCAM lookups) of p. Given a packet $p = (p_1, \cdots, p_d)$, there is one and only one path in the FDD that the packet matches. Let $v_1, \cdots, v_d$ be the d nonterminal nodes starting from the root in the path, and $t_1, \cdots, t_d$ be the d corresponding one-dimensional tables. For $1 \leqslant i \leqslant d$, let P_i be the prefix of the first rule that p_i matches in table t_i. We call the sequence of prefix/table pairs $\langle P_1/t_1, \cdots, P_d/t_d \rangle$ the *lookup path* of packet p. The lookup path for packet $(0011, 0010)$ for the example in Figure 2(b) is $\langle 00**/00, \cdots, ****/10 \rangle$.

Let k_i be the lookup time for the packets that satisfy P_i in table t_i. In other words, P_i is in layer $k_i - 1$ of table t_i (*i.e.*, $\mathbb{L}(P_i) = \mathbb{LIST}(t_i)[k_i - 1]$). Thus, the lookup time for packet p is $\Sigma_{i=1}^{d} k_i$. We call $\Sigma_{i=1}^{d} k_i$ the lookup time for the lookup path $\langle P_1/t_1, \cdots, P_d/t_d \rangle$. For example, the lookup time for packet $(0011, 0010)$ on the multi-lookup table in Figure 2(c) is 5 because the first field 0011 needs 2 BCAM lookups on table 00 in 2(b) and the second field 0010 needs 3 BCAM lookups on table 10 in 2(b).

Next, we present techniques for minimizing the maximum lookup time and the average lookup time respectively. The key observation is that the BCAM may still have free space after storing a multi-lookup table. Such free BCAM space can be exploited to optimize lookup time.

C. Algorithm for Minimizing Maximum Packet Lookup Time

The key idea is that we can reduce the lookup time for some packets by expanding some prefixes in the 1-dimensional lookup tables. Our focus is to find the correct prefixes to expand such that the maximum lookup time can be reduced. For any lookup path $\langle P_1/t_1, \cdots, P_d/t_d \rangle$ where t_i is in an F_i table for each $1 \leqslant i \leqslant d$, we can calculate its lookup time as discussed above. By traversing the reduced FDD where each nonterminal node is associated with a one-dimensional table, we can easily calculate all the lookup paths that have the maximum lookup time. We call a lookup path that has the maximum lookup time a *maximum lookup path*. Given a set of maximum lookup paths, we next discuss how to bring the maximum lookup time down using available BCAM space.

For any prefix P in a minimum 1-dimensional lookup table t, we can reduce the lookup time for the packets whose first matching prefix is P by either expanding P directly or expanding all prefixes in a layer above P in t. More formally, assuming $\mathbb{L}(P) = \mathbb{LIST}(t)[i]$, expanding the single prefix P into $2^{\mathbb{LIST}(C)[i-1]-\mathbb{LIST}(C)[i]}$ prefixes of length $\mathbb{LIST}(C)[i-1]$ and expanding all prefixes of length $\mathbb{LIST}(C)[j]$ $(0 < j < i)$ have the same lookup time reduction effect for the packets that satisfy P. Consider the example of table 00 in Figure 2(b). If we want to reduce the lookup time of the packets whose first matching prefix is $****$, we can either expand $****$ to four prefixes $00**$, $01**$, $10**$, and $11**$, or we can expand $00**$ to four prefixes 0000, 0001, 0010, and 0011. We define the follow set as the *lookup reduction set for prefix P in table t*: $\{(\{P\}, t), (\{P' \mid \mathbb{L}(P') = \mathbb{LIST}(C)[i-1]\}, t), \cdots, (\{P' \mid \mathbb{L}(P') = \mathbb{LIST}(C)[1]\}, t)\}$. For example, for prefix $****$ in table 00 in Figure 2(b), the lookup reduction set is $\{(\{****\}, 00), (\{00**\}, 00)\}$, where 00 is the table ID.

Suppose that there are a total of m maximum lookup paths: $\langle P_{1,1}/t_{1,1},\ P_{1,2}/t_{1,2},\ \cdots\ P_{1,d}/t_{1,d}\rangle,\ \cdots, \langle P_{m,1}/t_{m,1},\ P_{m,2}/t_{m,2},\ \cdots P_{m,d}/t_{m,d}\rangle$. For example, the FDD in Figure 2(b) has 3 lookup paths with maximum lookup time of 6: $\langle * * * * /00, * * * * /01\rangle$, $\langle * * * * /00, * * * * /10\rangle$, $\langle ****/00, ****/11\rangle$. Let $R_{i,j}$ $(1 \leqslant i \leqslant m, 1 \leqslant j \leqslant d)$ be the lookup reduction set for prefix $P_{i,j}$ in its corresponding table $t_{i,j}$. We define $R_i = \cup_{j=1}^{d} R_{i,j}$ as the lookup reduction set for lookup path $\langle P_{i,1}/t_{i,1}, P_{i,2}/t_{i,2}, \cdots P_{i,d}/t_{i,d}\rangle$. For example, the lookup reduction set for lookup path $\langle ****/00, ****/01\rangle$ is $\{(\{****\}, 00), (\{00**\}, 00), (\{****\}, 01), (\{11**\}, 01)\}$; the lookup reduction set for lookup path $\langle ****/00, ****/10\rangle$ is $\{(\{****\}, 00), (\{00**\}, 00), (\{****\}, 10), (\{01**\}, 10)\}$; the lookup reduction set for lookup path $\langle ****/00, ****/11\rangle$ is $\{(\{****\}, 00), (\{00**\}, 00), (\{****\}, 11), (\{10**\}, 11)\}$.

To reduce the lookup time of all the m maximum lookup paths $\langle P_{1,1}/t_{1,1},\ P_{1,2}/t_{1,2},\ \cdots P_{1,d}/t_{1,d}\rangle,\ \cdots, \langle P_{m,1}/t_{m,1},\ P_{m,2}/t_{m,2},\ \cdots P_{m,d}/t_{m,d}\rangle$, we must choose at least one expansion option from every R_i for $1 \leqslant i \leqslant m$. The easiest way to do this is to choose an expansion option in $\cap_{i=1}^{m} R_i$ if this intersection set is not empty. For the three maximum lookup paths in Figure 2(b), by expanding either (****,00) or(00**,00), we can reduce the lookup time of all three paths. Otherwise, we need to choose more than one expansion option. However, we do not want to expand more than is necessary. We enumerate all other possible minimum expansions in the following manner. For each set R_i, we define $R'_i = R_i - \cap_{j=1}^{m} R_j$. We then consider all combinations of m elements from R'_i for $1 \leqslant i \leqslant m$. Note that the same expansion option may show up in multiple R'_i sets in which case we would use fewer than m expansion options. For example, to reduce the lookup time of all three maximum lookup paths in Figure 2(b), we can expand the three prefixes: $(* * **, 01)$, $(* * **, 10)$, and $(* * **, 11)$. Each expansion reduces the lookup time for one lookup path.

Our algorithm for minimizing maximum lookup time works as follows. We have an initialization phase where we determine our initial BCAM budget. Then we enter our greedy iterative phase where we first compute all the maximum lookup paths. Second, we compute all the expansion options that can reduce the lookup time of all maximum lookup paths. Third, we choose the expansion option that requires the fewest extra BCAM entries. When there is a tie, we choose the expansion option that can reduce the lookup time of more paths. We then update our BCAM budget and repeat our greedy process until the BCAM budget is used up.

D. Algorithm for Minimizing Average Packet Lookup Time

Rather than minimize the *maximum* lookup time, we may want to minimize *average* lookup time. This does require a probability distribution for packets. For simplicity of description, we assume each packet is equally likely to occur, though our greedy algorithm extends to general probability distributions. Given a classifier, a fixed BCAM budget, and a probability distribution on packets, the problem of finding the set of prefixes to expand that fit within our BCAM budget and improve average packet lookup time the most is an open problem with a knapsack flavor. In this paper, we propose the following simple greedy strategy. For each minimum 1-dimensional table t, assuming t's skip list $\mathbb{LIST}(t)$ has k elements, for each i $(1 \leqslant i < k)$, we calculate the cost of expanding all prefixes of length $\mathbb{LIST}(t)[i]$ and the gain measured by the number of packets whose lookup time will be reduced by one. Among all such expansion options, we choose the one with the largest $gain/cost$ ratio and update our BCAM budget.

The above process repeats until the average lookup time reduces very slowly or the BCAM is used up. We define a round as the process of choosing one prefix layer to expand. Let AVG_t to denote the average lookup time after round t. We predefine a small threshold ε. When $\frac{|AVG_t - AVG_{t-1}|}{AVG_{t-1}} < \varepsilon$, we terminate the process after round t. In our experiments, we chose $\varepsilon = 10^{-2}$.

Considering the lookup tables in Figure 2(b), the rule with the best gain for cost is the prefix $****$ in table 00. This prefix has a gain of $11 \times 16 = 176$ as 176 packets will have their lookup time reduced by one. The cost of this prefix is 2 as we need to introduce 3 prefixes, $01**$, $10**$, and $11**$, for a cost of 2 extra prefixes. This expansion thus has a $gain/cost$ ratio of $176/2 = 88$. Of course, this expansion can only be chosen if we have a BCAM budget of at least 2.

E. Lookup Short Circuiting

So far, we have assumed the use of *full-length FDDs* where in each decision path all fields appear exactly once. Actually, this constraint can be relaxed so that some paths may omit unnecessary fields when a node in the path contains only one outgoing edge. In this case, the node along with singleton outgoing edge can be pruned. Using FDDs that are not full-length has the advantage of reducing FDD size and consequently reducing the total number of tables. Furthermore, this optimization allows some specific decision paths to be performed with a reduced number of lookups, which will allow

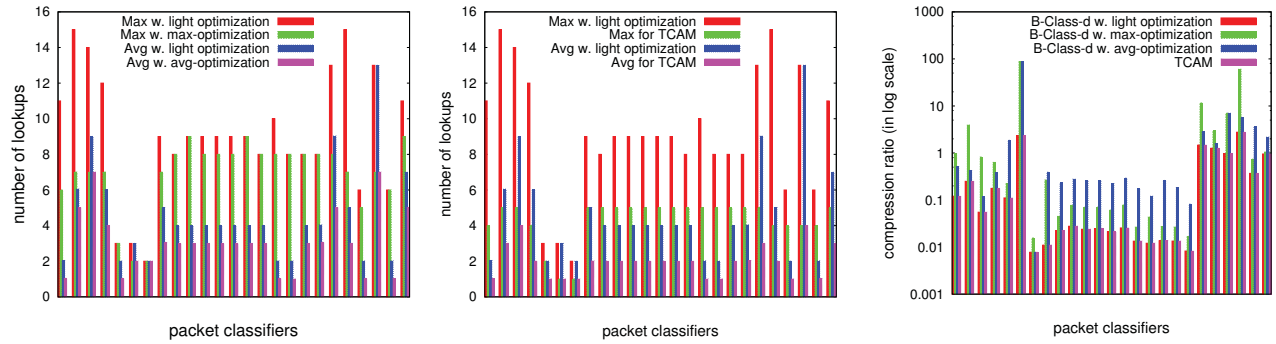


Fig. 3. Experimental results for real-life classifiers

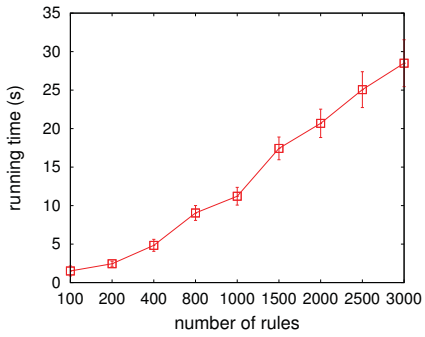


Fig. 4. Running time for synthetic classifiers

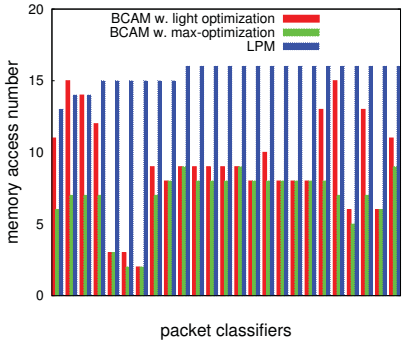


Fig. 5. Maximum lookup time for real classifiers

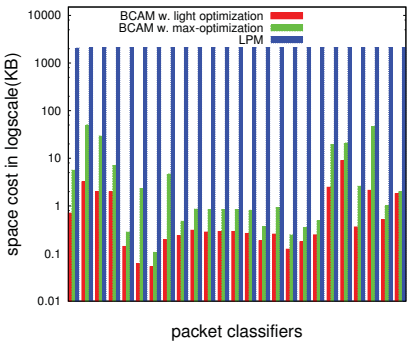


Fig. 6. Memory cost for real classifiers

for faster packet processing when the tables are processed in a multi-lookup fashion. Therefore, we call this optimization technique *lookup short circuiting*. Note this optimization technique requires storing field information in the decisions. This optimization is very useful as our experimental results in the next section demonstrate.

V. Experimental Results

We now report our experimental results on the effectiveness and efficiency of B-CLASS-d for a number of real-world packet classifiers as well as large synthetic packet classifiers. As a reminder, we use BCAM lookup time to refer to the number of BCAM lookups performed.

A. Metrics

We first define the metrics for measuring the BCAM space that B-CLASS-d in its various forms uses. Given a BCAM classification algorithm A and a classifier f, let $A(f)$ denote the BCAM space used by A for f, and $Direct(f)$ denote the TCAM space used by direct expansion for f. We define the space used by a classifier in a BCAM or TCAM chip as the number of classifier entries multiplied by the entry width measured in bits: $space = \#\ of\ entries \times CAM\ width$, where $CAM\ width$ is 72, 144, 288, or 576. Note that CAM width typically must be 72, 144, 288, or 576 bits wide. We define the following metrics regarding CAM space: *Compression ratio* of A on f: $CRatio(A, f) = \frac{A(f)}{Direct(f)}$; *Average compression ratio* of A for a set of classifiers S: $\overline{A(S)} = \frac{\sum_{f \in S} CRatio(A,f)}{|S|}$, where $|S|$ is the number of classifiers in S.

For all classifiers, we apply the optimization techniques of minimizing maximum lookup time and minimizing average lookup time in isolation for a better evaluation on the effectiveness of each technique, although minimizing maximum lookup time will reduce the average lookup time and minimizing average lookup time may reduce the maximum lookup time.

B. Effectiveness

We conducted experiments on a set of 25 real-life classifiers. We obtained these real-life classifiers from distinct network service providers and the classifiers range in size from dozens to hundreds of rules. We call the combination of skip lists, free expansion, and short circuiting techniques "*light optimization*" because they performed very efficiently in our experiments. We use *max-optimization* to denote the technique of minimizing the maximum lookup time in addition to light optimization, and *avg-optimization* to denote the technique of minimizing the average lookup time in addition to light optimization. Since the techniques of max-optimization and avg-optimization are sensitive to the available BCAM space, we assume the use of a 0.5 Mb BCAM chip in our experiments.

Figure 3 shows the effectiveness of our optimization techniques for all 17 packet classifiers. Note that without any optimizations, B-CLASS-d requires 109 lookups for every packet. For maximum lookup time, Figure 3(a) shows that *light optimization* brings down the maximum lookup time from 109 to 15 or lower with an average of 10.4, and max-optimization further reduces the maximum lookup time to 12 or lower with an average of 8.9. For average lookup time, Figure 3(a) shows that light optimization brings down the average lookup time

from 109 to 12.99 or lower with an average of 4.7, and max-optimization further reduces the average lookup time to 8.99 or lower with an average of 3.4.

In Figure 3(b), we compare the lookup time of B-CLASS-d with only light optimization and that of the sequential decomposition based TCAM multi-lookup scheme. Without the technique of short circuiting, the lookup time of the sequential decomposition based TCAM multi-lookup scheme is always 5. With the technique of short circuiting, the maximum lookup time for the TCAM scheme is 5 or lower with an average of 4.4. In comparison, the maximum lookup time for B-CLASS-d with light optimization is 15 or lower with an average of 10.4, and it drops to 12 or lower with an average of 8.9 if we apply max-optimization. With the technique of short circuiting, the average lookup time for the TCAM scheme is 3.9 or lower with an average of 1.8. In comparison, the average lookup time for B-CLASS-d with light optimization is 12.99 or lower with an average of 4.7, and it drops to 8.99 or lower with an average of 3.4 if we apply avg-optimization. Based on the above empirical data, we see that B-CLASS-d with light optimization often achieves a maximum or average lookup time that is often no more than twice that of the TCAM scheme. Given that BCAM is significantly cheaper than TCAM, it is cost-effective to use two BCAM chips to process packets in parallel to achieve similar throughput with the TCAM scheme. Furthermore, in non-peak modes, one BCAM chip can be powered down to conserve energy.

Figure 3(c) shows the compression ratios of the optimized versions of B-CLASS-d (*i.e.*, with light optimization, with max-optimization, and with avg-optimization) and the TCAM scheme (with short circuiting). The experimental results show that B-CLASS-d with light optimization achieves compression ratios that are similar to those achieved by the TCAM scheme. For the real-life classifiers, the compression ratios achieved by B-CLASS-d with light optimization range between 0.008 and 2.77 with an average of 0.44; similarly, the compression ratios achieved by the TCAM scheme range between 0.008 and 2.77 with an average of 0.44. We also observe that the techniques of max-optimization and avg-optimization do consume a significant amount of BCAM space. The compression ratios achieved by the BCAM scheme with max-optimization range between 0.02 and 87.33 with an average of 7.13; similarly, the compression ratios achieved by the BCAM scheme with avg-optimization range between 0.008 and 87.33 with an average of 4.65. However, these large compression ratios are less of a concern, except from an energy standpoint, because these techniques are constrained by the available BCAM space.

C. Efficiency

We implemented our BCAM packet classification scheme using Visual Basic on the Microsoft .Net framework 2.0 and Python. Our experiments were carried out on a desktop PC running Windows XP with 1G memory and a single 2.2 GHz AMD Opteron 148 processor. Most real world classifiers ran in under a second. Table II show the running time for three representative classifiers. The experimental results show that both B-CLASS-d with light optimization and B-CLASS-d with avg-optimization are very efficient; however, the results show that the techniques of max-optimization is much slower. Thus, we recommend the use of max-optimization when timing is not an immediate concern.

Number of Rules	with light optimization (second)	with avg-optimization (second)	with max-optimization (second)
661	2.45	3.99	69.77
87	0.99	2.94	168.03
42	0.65	0.78	0.85

TABLE II
RUNNING TIMES FOR THREE REAL-LIFE CLASSIFIERS

Because packet classifier rules are considered confidential due to security concerns, it is difficult to get many real-life packet classifiers for experiments. To address this issue and further evaluate the efficiency of our approaches, we generated a set of synthetic packet classifiers of 9 sizes, where each size has 100 independently generated classifiers. Every predicate of a rule in our synthetic packet classifiers has five fields: source IP address, destination IP address, source port number, destination port number, and protocol type. We first randomly generated a list of values for each field. For IP addresses, we generated several random class C addresses and then generated single IP addresses within the class C addresses; for ports we generated a random range; for protocols, we chose TCP, UDP, or ICMP. Every field also has the "*" value included in the list. We then generated a list of predicates by taking the cross product of these five lists and randomly selected from the cross product until we reached our desired classifier size by including a final default predicate. Finally, we randomly assigned one of two decisions, accept or discard, to each predicate to make a complete rule. The timing results on the synthetic rules for our BCAM classification scheme with light optimization are shown in Figure 4.

D. Ke-Longest Prefix Matching

We implemented the longest prefix matching algorithm described in [7] and built a packet classification scheme based on sequential decomposition. We evaluated both lookup time and memory cost of two schemes.

Figure 5 shows the maximum (worst case) memory access number for B-CLASS-d with only light optimization, B-CLASS-d with max-optimization and sequential decomposition based longest prefix matching scheme. For B-CLASS-d one BCAM lookup is counted as one memory access. The maximum lookup time for B-CLASS-d with light optimization is 15 or lower with an average of 10.4, and it drops to 12 or lower with an average of 8.9 if we apply max-optimization. The maximum lookup time for sequential decomposition based longest prefix matching is 16 with an average of 15.48. Note that we count one Bloom filter query or one hash table lookup as one memory access for the sequential decomposition based longest prefix matching scheme.

In Figure 6, we evaluated the memory cost of the three schemes. The experimental result shows that the memory cost for B-CLASS-d with light optimization is from 0.053 KB to

8.93 KB with an average of 1.08 KB, and it increases to from 0.105 KB to 50.2 KB with an average of 7.93 KB. Note that we assume 0.5 Mb BCAM budget. The memory cost for sequential decomposition based longest prefix matching is from 2083 KB to 2106 KB with an average of 2099 KB. We can see that our B-CLASS-d scheme requires at least two orders of magnitude less memory space than sequential decomposition based longest prefix matching scheme with better worst case performance.

VI. Conclusions and Future Work

The significance of this paper lies in B-CLASS-d, the first proposed BCAM packet classification scheme, as well as the several optimization techniques such as skip lists, free expansion, an algorithm for minimizing maximum lookup time, an algorithm for minimizing average lookup time, and lookup short circuiting. We implemented B-CLASS-d and conducted experiments on a number of real-life classifiers. Our experimental results validate the practicality of building high performance BCAM based packet classification systems. Furthermore, as this paper opens a new packet classification paradigm, there are many possible future research directions such as developing more efficient and effective algorithms for minimizing the maximum or average lookup time.

Acknowledgements

We would like to thank Ke Shen for his participation in the early stage of this work. He has helped us in implementing the proposed algorithms and conducting experiments. This work is supported in part by the National Science Foundation under Grant Numbers CNS-0916044, CNS-0845513, CNS-1017588, and CCF-1347953.

References

[1] A guide to search engines and networking memory. *http://www.linleygroup.com/pdf/NMv4.pdf*.

[2] Anonymous. Paper under double-blind review.

[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] F. Baboescu and G. Varghese. Scalable packet classification. In *Proc. ACM SIGCOMM*, pages 199–210, 2001.

[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] J. Daly, A. X. Liu, and E. Torng. A difference resolution approach to compressing access control lists. In *Proc. 32th Annual IEEE Conf. on Computer Communications (INFOCOM)*, Turin, Italy, April 2013.

[7] S. Dharmapurikar, P. Krishnamurthy, and D. Taylor. Longest prefix matching using bloom filters. In *Proc. ACM SIGCOMM*, 2003.

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

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

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

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

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

[13] A. X. Liu and F. Chen. Collaborative enforcement of firewall policies in virtual private networks. In *Proc. Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, pages 95–104, Toronto, Canada, August 2008.

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

[15] 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.

[16] A. X. Liu and M. G. Gouda. Complete redundancy removal for packet classifiers in tcams. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 21(4):424–437, April 2010.

[17] A. X. Liu, C. R. Meiners, and E. Torng. TCAM razor: A systematic approach towards minimizing packet classifiers in tcams. *IEEE/ACM Transactions on Networking*, 18(2):490–500, 2010.

[18] A. X. Liu, C. R. Meiners, and Y. Zhou. All-match based complete redundancy removal for packet classifiers in TCAMs. In *Proc. 27th Annual IEEE Conf. on Computer Communications (Infocom)*, pages 574–582, April 2008.

[19] A. X. Liu, E. Torng, and C. Meiners. Firewall compressor: An algorithm for minimizing firewall policies. In *Proc. 27th Annual IEEE Conf. on Computer Communications (Infocom)*, pages 691–699, April 2008.

[20] H. Liu. Efficient mapping of range classifier into Ternary-CAM. In *Proc. Hot Interconnects*, pages 95– 100, 2002.

[21] C. R. Meiners, A. X. Liu, and E. Torng. TCAM Razor: A systematic approach towards minimizing packet classifiers in TCAMs. In *Proc. 15th IEEE Conf. on Network Protocols (ICNP)*, pages 266–275, October 2007.

[22] C. R. Meiners, A. X. Liu, and E. Torng. Bit weaving: A non-prefix approach to compressing packet classifiers in TCAMs. In *Proc. 17th IEEE Conf. on Network Protocols (ICNP)*, pages 93–102, October 2009.

[23] C. R. Meiners, A. X. Liu, and E. Torng. Topological transformation approaches to optimizing tcam-based packet processing systems. In *Proc. ACM Int. Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 73–84, June 2009.

[24] C. R. Meiners, A. X. Liu, and E. Torng. Topological transformation approaches to tcam-based packet classification. *IEEE/ACM Transactions on Networking*, 19(1):237–250, February 2010.

[25] C. R. Meiners, A. X. Liu, and E. Torng. Bit weaving: A non-prefix approach to compressing packet classifiers in TCAMs. *IEEE/ACM Transactions on Networking (ToN)*, 20(2):488–500, April 2012.

[26] C. R. Meiners, A. X. Liu, E. Torng, and J. Patel. SPliT: Optimizing space, power, and throughput for TCAM-based classification. In *Proceedings of ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 200–210, 2011.

[27] E. Norige, A. X. Liu, and E. Torng. A ternary unification framework for optimizing tcam-based packet classification systems. In *Proc. 9th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, San Jose, California, October 2013.

[28] 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.

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

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

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

[32] T. Y. C. Woo. A modular approach to packet classification: Algorithms and results. In *Proc. IEEE INFOCOM*, pages 1213–1222, 2000.