Algorithms for Advanced Packet Classification with Ternary CAMs

Karthik Lakshminarayanan
Univ. of California, Berkeley

Anand Rangarajan
Cypress Semiconductor

Srinivasan Venkatachary
Cypress Semiconductor

ABSTRACT

Ternary content-addressable memories (TCAMs) have gained wide acceptance in the industry for storing and searching Access Control Lists (ACLs). In this paper, we propose algorithms for addressing two important problems that are encountered while using TCAMs: *reducing range expansion* and *multi-match classification*.

Our first algorithm addresses the problem of expansion of rules with range fields—to represent range rules in TCAMs, a single range rule is mapped to multiple TCAM entries, which reduces the utilization of TCAMs. We propose a new scheme called *Database Independent Range PreEncoding* (DIRPE) that, in comparison to earlier approaches, reduces the worst-case number of TCAM entries a single rule maps on to. DIRPE works without prior knowledge of the database, scales when a large number of ranges is present, and has good incremental update properties.

Our second algorithm addresses the problem of finding multiple matches in a TCAM. When searched, TCAMs return the first matching entry; however, new applications require either the first few or all matching entries. We describe a novel algorithm, called *Multi-match Using Discriminators* (MUD), that finds multiple matches without storing any per-search state information in the TCAM, thus making it suitable for multi-threaded environments. MUD does not increase the number of TCAM entries needed, and hence scales to large databases.

Our algorithms do not require any modifications to existing TCAMs and are hence relatively easy to deploy. We evaluate the algorithms using real-life and random databases.

Categories Subject Descriptors

C.2.6 [**Internetworking**]: Routers.

General Terms

Algorithms, Performance, Design.

Keywords

Packet classification, Ternary CAMs, Multi-match, Range.

1. INTRODUCTION

High-speed packet classification algorithms that scale to large multi-field databases have become a widespread requirement for a variety of applications such as network security appliances, quality of service filtering and load balancers. For classifying packets, a router employs a *classification database* (also called a policy database) which has several *access control lists* (ACLs). Each ACL consists of *rules* that are applied on incoming or outgoing packets. While the syntax of these rules varies based on the router vendor, the semantics of the rules allows similar classification information to be specified—the rules allow the definition of various patterns based on the packet header. Furthermore, for each rule, the set of actions to be taken on packets that match the rule is also specified.

Designing algorithms that scale to millions of rules and millions of searches per second has been and continues to be an important stream of research. Several advances in algorithmic approaches that use off-chip random access memories have been made in the past few years. Recursive Flow Classification [8], Crossproducting [18, 20], HyperCuts [15], Extended Grid-of-Tries [1] are some examples; refer to [19, 23] for details of these techniques.

However, in the past few years, the industry has increasingly employed *Ternary Content Addressable Memories* (TCAMs) for performing packet classification [5, 9, 12]. A large class of current- and next-generation systems that require up to a few hundred thousand rules have adopted TCAMs for packet classification at multi-gigabit speeds.[1] The number of TCAM devices that have been deployed worldwide in 2004 is over 6 million [3].

A TCAM is a memory device that stores data as a massive array of fixed-width ternary entries. A ternary entry is a string of bits where each bit is either 0, 1 or x (don't care). Given a search key, the TCAM searches the key in parallel against all the ternary entries stored in the TCAM and produces the first match as the result. TCAMs provide two main characteristics that make them suitable for router design: *deterministic search throughput* and *deterministic capacity*. Current TCAMs can support up to 133 million searches per second for 144-bit wide keys, and can store 128K ternary entries that are 144 bits wide in a single device.

1.1 Problems

While TCAMs are well-suited for performing high-speed searches on databases with ternary entries, the following problems and trends reduce the efficiency of TCAMs.

Range rules: To store a rule with *range fields*, multiple TCAM entries are needed, which reduces the efficiency of TCAMs [11,16]. In IP router ACLs, the port fields usually have ranges. As ranges cannot be directly stored in TCAMs, traditionally, ranges are converted to a corresponding set of prefixes, and each prefix is stored in a separate TCAM entry (see Section 3.1.1). When this range-to-prefix expansion technique is applied on the port fields, which are 16 bits wide, a rule with a single range field can expand to 30 TCAM entries in the worst case. By analyzing router ACL databases dated 1998 and 2004, we provide evidence for the following temporal trends that make the range expansion problem an important one. Table 1 (see Section 3) provides the actual numbers.

[1]Due to power and cost considerations, current-generation TCAMs face scalability challenges. A cost-effective approach to supporting millions of rules at high speeds is still a topic of research.

SIGCOMM'05, August 21–26, 2005, Philadelphia, Pennsylvania, USA.

- Number of rules in router ACL databases is increasing.
- Percentage of rules with one range field is increasing.
- Percentage of rules with two range fields is increasing.
- Number of unique ranges is increasing.

Multi-match classification: Security applications and accounting applications require the *first k* matching entries, or in some cases *all* the matching entries, for a given key [25]. TCAMs do not natively support finding multiple matches; they report only the first matching entry.

1.2 Our Contributions

In this paper, we propose algorithms for addressing the range expansion and multi-match classification problems using off-the-shelf TCAMs. Our first algorithm, *Database Independent Range PreEncoding* (DIRPE), reduces the worst-case number of TCAM entries a single rule maps on to, when compared to earlier schemes. DIRPE works without knowledge of the database, scales when a large number of ranges is present, and has good incremental update properties. Our second algorithm, *Multi-match Using Discriminators* (MUD), enables multiple matches to be found using a TCAM without storing any per-search state information in the TCAM, thus making it suitable for multi-threaded packet processing environments. MUD can scale to large databases since it does not expand the number of TCAM entries needed. The benefits of MUD come at the cost of extra search cycles; however, we show that MUD can still support multi-match classification at multi-gigabit link speeds.

Both our algorithms utilize unused bits in the TCAM array to encode relevant information. Though the algorithms solve seemingly different problems, we draw similarities between the algorithms, and apply similar ideas for both the algorithms.

Our schemes do not require any change to existing TCAMs and hence are relatively easy to deploy. This metric is important as TCAMs are complex devices and architectural changes that modify TCAMs involve millions of dollars of investment and more than two years of development time. Hence, algorithmic approaches that utilize current TCAMs to solve a problem are preferable.

The rest of the paper is organized as follows. In Section 2, we provide some background and describe the metrics and terminology used in the paper. In Section 3, we present our database-independent range encoding algorithm. In Section 4, we describe our multi-match classification algorithm. Related work, evaluation of the schemes, and comparison with earlier approaches are presented in the corresponding sections themselves.

2. PACKET PROCESSING ENVIRONMENT

Figure 1 shows a packet processor connected to a set of TCAMs. Packets are classified using a classification database consisting of several access control lists (ACLs), each of which holds several rules. The control plane software maintains the ACLs and stores them in the TCAMs. When data packets arrive, the packet processor parses the packet headers, and forms keys to search the appropriate ACLs (based on factors such as the interface the packet arrives on). A typical search key is of the form:

<acl-id><proto><src-ip><dest-ip><src-port><dest-port>

A parallel search is performed on the entries stored in the TCAM. The search returns the index of the first entry that matches the search key. A memory location corresponding to the result index is used to store the *action* to be taken when a search key matches the entry. Typical actions include permit/deny, update counters and replicate on a port.

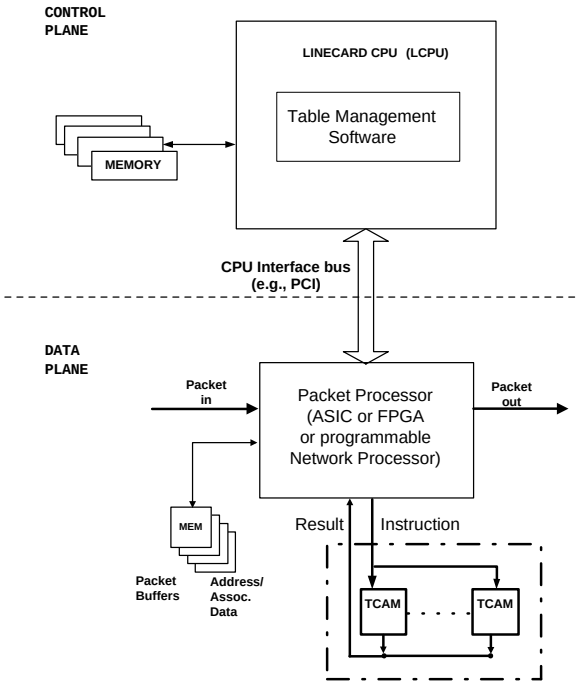


Figure 1: System picture of a router line card showing control and data planes.

TCAMs constitute a significant portion of the cost of a multi-gigabit linecard. For example, the price for a 10 gigabit linecard in the next couple of years is expected to be less than a thousand dollars [6]. However, TCAMs that can support 128K entries of 144-bit width are expected to cost over $200 for the next few years [5,9,12]. Hence, to design efficient, low-cost multi-gigabit linecards, it is critical to utilize TCAMs as efficiently as possible. Though today's TCAMs do not scale to millions of rules due to cost and power considerations, they are well-suited for storing databases with up to a few hundred thousand rules.

State-of-the-art TCAMs provide 18M ternary bits which are organized into 32, 64 or 128 blocks. Each block can be independently configured to have a width of 72, 144, 288 or 576 bits. After the fields of an ACL rule are encoded in the TCAM, typically there are some extra bits that are left unused. For example, most IPv4 ACLs consist of an identifier and a protocol (8 bits), destination address (32 bits), source address (32 bits), destination port (16 bits) and source port (16 bits)—a total of 104 bits. Usually, 4 more bits are used to encode miscellaneous fields. Since TCAMs are typically configured to be 144 bits wide, 36 extra bits remain unused.

We now describe some metrics for comparing algorithms used in the packet classification subsystem for multi-gigabit routers.

2.1 Metrics

Speed/Throughput: The system has to support a guaranteed throughput (in gigabits per second (Gbps)). To compute the guaranteed rate in millions of packets per second (MPPS), we assume a minimum packet size of 64 bytes [1].

The system requires a certain minimum throughput (measured in Millions of Packets Per Second (MPPS)), which is usually the wire-rate assuming smallest-sized packets. Assuming 64-byte packets, OC-48 corresponds to roughly 5 MPPS and OC-192 to 20 MPPS.

Capacity: Capacity is the number of rules that can be supported in the search subsystem. In our experiments, we compared the capacity of candidate algorithms using worst-case, real-life, and random databases.

Update Speed: Traditionally, rules are updated manually, and low update speeds of the order of a few hundred per second are acceptable. However, newer systems that perform real-time active response to hostile network events [4] require incremental updates at much higher speeds.

Overhead on Packet Processor: This is the cost of the extra logic, if any, that is needed as part of the packet processor.

Multi-threading Support: Most packet processors use many threads of execution to achieve high speed. Components such as TCAMs attached to the network processor are shared by all these threads. Hence, algorithms that utilize such components must take multi-threading into account.

In this paper, we compare various schemes using the metrics described above. For real-life comparisons, we provide results using some Internet Service Provider (ISP) databases that we obtained. Due to privacy reasons, we cannot reference the ISP. We also use the Snort database, a publicly available intrusion detection database [2].

2.2 Terminology

We now introduce some terminology that we use in the paper. Let N denote the number of rules in a database $\mathbf{F_{dat}}$. Let a range field be W bits wide. Let R denote the closed range $[s, e]$, where s and e are W-bit numbers.

A *key* S is a collection of K fields from the packet header H. These header fields are denoted $H[1], H[2], \ldots, H[K]$, where each field is a string of bits.

A *prefix* P is a bit string of length between 0 and W. *length*(P) denotes the number of bits in a prefix. (P padded with 0's is the smallest number encompassed by P and P padded with 1's is the largest number encompassed by P).

A *filter rule or ACL rule* F is a collection of K fields. Each field $F[i]$ in a rule can specify any of three kinds of matches: *exact match*, *prefix match*, or *range match*. A rule that has at least one of its fields having a range match specification is referred to as a *range rule*.

An *exact match* specification is a value specified for a rule field i. A header field $H[i]$ is an exact match for the rule field $F[i]$ if and only if $H[i] = F[i]$.

A *prefix match* specification is a prefix specified for a rule field i. A header field $H[i]$ is a prefix match for the rule field $F[i]$ if and only if the leading *length*$(F[i])$ bits of $H[i]$ are the same as $F[i]$.

A *range match* specification is a range of values $F[i] = val_{start}, \ldots, val_{end}$ for rule field i. A header field $H[i]$ is a range match for the rule field $F[i]$ if and only if $val_{start} \leq H[i] \leq val_{end}$.

A rule F is said to be a *matching rule* for a header H if and only if each field $H[i]$ of H matches the corresponding field $F[i]$ of F.

3. REDUCING EXPANSION OF RANGE RULES

Ternary CAMs are directly suited for storing ACL tables that have rules with wildcards. However, a range cannot be stored directly in a TCAM since a TCAM supports storing only 0, 1 and x (don't-care) states. Hence, storing range rules could take a large number of TCAM entries. In this section, we present DIRPE, a database-independent algorithm that reduces the expansion of range rules even in the worst-case. We first review earlier approaches and motivate our algorithm.

Statistic	1998 database	2004 database
Total number of rules	41190	215183
With single range field	4236 (10.3%)	54352 (25.3%)
With single range field excluding "≥ 1024" specification	553 (1.3%)	25311 (11.8%)
With two range fields	0 (0%)	3225 (1.5%)
Unique ranges in first field	62	270
Unique ranges in second field	0	37

Table 1: Number of rules with range fields in a collection of ACLs obtained in 1998 and 2004.

3.1 Earlier Approaches

3.1.1 Prefix Expansion of Ranges

A well-known method for representing range rules in TCAMs is to expand each range into a set of prefixes, which can then be used directly as the TCAM entries [19]. For rules with multiple range fields, the sets of prefixes corresponding to all the fields are crossproducted to get the TCAM entries. The worst-case expansion for a W-bit range is $2W-2$. A simple proof by construction is as follows. Consider the range $[1, 2^W-2]$. The smallest set of prefixes needed to cover this range is $\{01*, 001*, 0001*, \ldots, 0^{W-1}1, 10*, 110*, \ldots, 1^{W-1}0\}$. For a 16-bit range field, the worst-case expansion is 30. Hence, an IP ACL rule which has two 16-bit port fields can expand to $30 \times 30 = 900$ entries in the worst case.

3.1.2 Database-dependent Encoding of Ranges

To reduce the expansion of rules, additional bits in the TCAM can be used to encode the ranges that appear frequently. To illustrate this scheme, consider the example of an ACL database that contains the range R in several range rules. Consider the following encoding of an extra bit in TCAM: set the bit to 1 when the range specification in the rule encompasses the range R, and 0 otherwise. Furthermore, the extra bit in the search key is set to 1 if the key falls in the range R, and 0 otherwise. This encoding reduces the expansion of all rules that contain the range R to 1.

The simple scheme described above requires an extra bit for each distinct range that appears in the database to achieve a worst-case expansion of 1. To scale to databases with several unique ranges, *region-based range encoding* schemes have been proposed [11,22]. These schemes divide the ranges into many regions and use a hierarchical encoding; they first encode the regions and then the ranges within the regions. Since the encoding depends on the database, incremental updates of the rules are expensive.

Furthermore, to append the search key with the appropriate bits, the packet processor needs logic with a certain number of comparators. While the overhead on the processor is small for databases with a few unique ranges, as the number of unique ranges increases, the logic gets prohibitively large. Since Table 1 shows a trend of increasing number of unique ranges, we expect this problem to only worsen. An alternative to using the logic with comparators is to have a precomputed table that maps each possible key value into the appropriate extra bits. While precomputation is feasible today for 16-bit range fields (corresponds to a table with 64K entries), for larger widths, the table would get prohibitively large. For example, even a 24-bit field would require a table size of 16M entries.

3.1.3 Modifying TCAMs to Accommodate Ranges

TCAM modifications to accommodate range matching better—such as implementing comparators at each entry level—have been proposed [16]. Experimenting with such approaches is important. However, since TCAMs are massively parallel, circuit-intensive de-

Range	Prefixes	DIRPE with 1 extra bit
≥ 0	xx	$xx\ x$
≥ 1	$01, 1x$	$xx\ 1$
≥ 2	$1x$	$1x\ x$
≥ 3	11	$11\ x$

Table 2: Expansion of "$\geq$" ranges on a 2-bit field using prefix expansion and DIRPE. Notice that the representation is in ternary and not prefix format. The search key b_1b_0 would be replaced by b_1b_0c where $c = b_0\ OR\ b_1$)

vices, even small changes at a per-entry circuit level can, besides requiring several million dollars of investment, suffer from long lead times (of at least a few years) before they can be produced at acceptable speed, cost and power. Hence, while modifying TCAMs is not impossible, changing the ternary nature of the entries has many barriers; software-based algorithms that use existing TCAMs to better represent range rules are often preferred.

3.2 Database Independent Range PreEncoding (DIRPE)

3.2.1 Basic Ideas behind DIRPE

DIRPE is based on two simple ideas. First, instead of representing a range as a set of prefixes, we can represent it as a set of *arbitrary* ternary values. (For example, $0xx1x0$ is a ternary value that is not a prefix.) Second, additional unused bits in a TCAM array can be used to encode the ternary strings. Hence, the ternary values would be wider than the prefixes, but the total number of them would be less than the number of prefixes, even in the worst case. Since TCAMs have pre-defined widths, extra bits are available in each row "for free" after storing the bits corresponding to the rule.

We illustrate these ideas using a simple encoding for ranges of the form "$\geq$" on a 2-bit field. Using prefix expansion, the worst-case expansion of any range is 2. By using three (instead of two) bits in the TCAM to represent a range, the worst-case can be reduced to one TCAM entry (see Table 2). The search key for the range field b_1b_0 is augmented with the third bit using the equation $b_1\ OR\ b_0$. Logically, the third bit encodes whether the search key is a member of either 01 or $1x$.

3.2.2 DIRPE: Encoding Closed Ranges

We now describe a generic instantiation of DIRPE for encoding closed ranges on a W-bit range field. For now, let us assume that there is no restriction on the number of extra bits in the TCAM that we can use. Consider the following encoding, which we term as *fence encoding*, that maps a W-bit field to 2^W-1 bits: the encoding of a number i consists of i ones preceded by 2^W-1-i zeros.

As shown in Table 3, any closed range can be represented using fence encoding using a single ternary entry. In other words, 2^W-1 bits are sufficient for an encoding to reduce the worst-case expansion to 1. However, the following result shows that 2^W-1 bits are necessary. This result is surprising at first since 2^W bits are sufficient to represent arbitrary subsets, not just ranges.

THEOREM 1. *For achieving a worst-case row expansion of* 1 *for a W-bit range, 2^W-1 bits are necessary.*

We prove a simple lemma before we present the proof of the theorem. Let $f(R)$ denote the ternary encoding of a range R.

LEMMA 1. *Let R_1 be a completely contained subrange of R_2, then f must satisfy the following properties: (a) if a bit position i is specified (as 0 or 1) in $f(R_2)$, then it must be specified identically in $f(R_1)$, (b) there must be at least one don't-care bit in $f(R_2)$ that is specified (as 0 or 1) in $f(R_1)$.*

Range	Encoding
$= i$	$0^{2^k-i-1}1^i$
$\geq i$	$x^{2^k-i-1}1^i$
$< i$	$0^{2^k-i}x^{i-1}$
$[i, j]$	$0^{2^k-1-j}x^{j-i}1^i$

Table 3: Fence encoding of various types of ranges for a k-bit field (or a chunk). The encoding of a number i consists of 2^k-1-i zeros followed by i ones. Note that the notation a^b is used to denote both regular expressions (*e.g.,* 0^i) and exponentiation (*e.g.,* 2^k).

PROOF. If property (a) is not satisfied, let I be the set of all the positions where $f(R_2)$ has a bit $b_i = 0/1$ and $f(R_1)$ has an entry $b'_i \neq b_i$. Now, one can trivially construct an entry e with the i_{th} (for all $i \in I$) bit set to complement of b_i such that e matches R_1 but not R_2.

If property (b) is not satisfied, then combining with property (a), we have R_1=R_2. □

PROOF. (of theorem) Observe that the optimal way to represent the complete range R_n=$[0, 2^W-1]$ is $xx\ldots x$. Now, consider the range R_{n-1}=$[0, 2^W-2]$. It follows from lemma 1 that the representation of this range needs at least one more bit specified than in R_n. Without loss of generality, let us set the first bit to 0, hence giving the encoding $0x\ldots x$ to R_{n-1}. Proceeding thus, one needs to specify an extra bit all the way till the range $[0, 1]$. However, the last bit can be used to represent the range $[0, 0]$ also, by setting all the bits to zero. Hence, the total number of bits needed is 2^W-1. □

Since the number of unused bits in a TCAM array is much smaller than 2^W-1, the natural question is whether we can use the available unused bits to reduce the expansion of the database at all (if not reduce the expansion all the way to one).

We answer this question by generalizing the fence encoding technique. To form the ternary representation of a range, let us divide the field into multiple *chunks*, where each chunk represents a contiguous portion of the bits of W. Let W be split into l chunks, with chunk i having k_i bits. (Then, $W = k_0 + k_1 + \ldots + k_{l-1}$.) Here, k_0 corresponds to the most significant k_0 bits, k_1 corresponds to the next k_1 most significant bits and so on. Now, the value in the bit-strings corresponding to each of the chunks is mapped to their fence encoding, *i.e.,* each of the k_i bits is represented using $2^{k_i} - 1$ bits. The width of our new encoding is hence $W' = (2^{k_0} - 1) + \ldots + (2^{k_{l-1}} - 1)$.

To explain how ranges expand to ternary entries in this representation, we now present the analogy between the DIRPE encoding and multibit trie-based algorithms that have been used in the literature for improving the performance of IP lookups [7, 13, 17]. Compared to unibit tries, multibit tries reduce tree depth at the expense of larger node sizes. Analogous to reduction of tree depth in multibit tries, DIRPE achieves reduction in the number of TCAM entries. DIRPE uses additional bits per entry to encode the range field, which is analogous to having a larger node size per level.

Figure 2 shows a multibit trie view of DIRPE based on chunks. We refer to the number of chunks, l, as the number of levels also. In the example, W=8, R=[11, 54], l=3, k_0=2, k_1=3, and k_2=3. For a range $[s, e]$, *split chunk* is defined as the first chunk (or level) in which s and e differ. From the multibit view, it follows that each level of the multibit trie representation generates at most two entries: one corresponding to the branch corresponding to s and another corresponding to e. However, the split chunk generates only one entry; hence the worst-case expansion of a range is $2l-1$. The chunking strategy provides a tradeoff between worst-case expansion and the number of bits required for encoding the range. As the

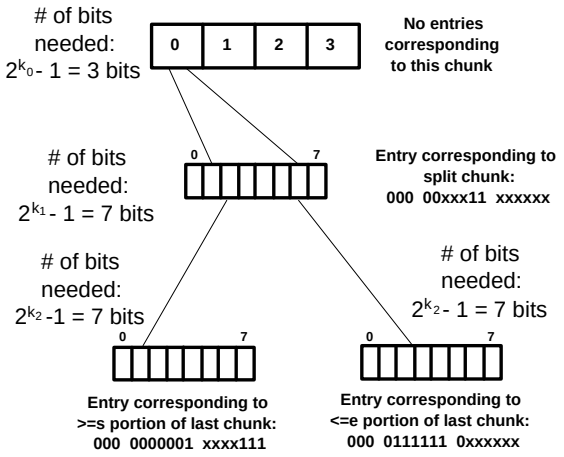


Figure 2: Multibit trie view of DIRPE. For the example, W=8 and R=[11,54]

Form_TCAM_Entries $(s, e, k_1, k_2, \ldots, k_l)$

if $(s = e)$
return the TCAM encoding of s

// Form entries for the split chunk
$c \leftarrow$ Find_Split_Chunk (s, e), where split chunk c is defined as the chunk such that $v_{s_c} \neq v_{e_c}$ and $v_{s_i} = v_{e_i}$ for all $i < c$
Form TCAM entry corresponding to chunk c that covers the range $v_{s_c} + 1, \ldots, v_{e_c} - 1$
Call this entry as *Split_Chunk_TCAM_Entry*

if $(c = l - 1)$
return the TCAM entry formed

// Form entries corresponding to $\geq s$ portion
Find j such that for $c < j < l$, $v_{s_j} \neq 0$ and $v_{s_i} = 0$ for all $i > j$
If no such j is found
Adjust *Split_Chunk_TCAM_Entry* to include v_{s_c}
else **for** $(i = c + 1$ to $j)$
Form TCAM entry corresponding to this chunk that covers the range $v_{s_i}, \ldots, 2^{k_i} - 1$

// Form entries corresponding to $\leq e$ portion
Find j such that for $c < j < l$, $v_{e_j} \neq 2^{k_i} - 1$ and $v_{e_i} = 2^{k_i} - 1$ for all $i > j$
If no such j is found
Adjust *Split_Chunk_TCAM_Entry* to include v_{e_c}
else **for** $(i = c + 1$ to $j)$
Form TCAM entry corresponding to this chunk that covers the range $0, \ldots, v_{e_i}$

return all TCAM entries formed

Form_Search_Key (*Key*)

Split *Key* into l chunks, *Key*$_0, \ldots,$ *Key*$_{l-1}$ of sizes $k_0, \ldots, k_{l-1}$ respectively

for $(i = 0$ to $l-1)$
$E_i \leftarrow$ Fence encoding of *Key*$_i$

// concatenate the fence encodings of the key chunks
return $E_1 E_2 \ldots E_{l-1}$

Metric	Prefix Expansion	Region-based Encoding (with *r* regions)	DIRPE (with *k*-bit chunks)	DIRPE + Region-based
Extra bits	0	$F(\log_2 r + \frac{2n-1}{r})$	$F(\frac{W(2^k-1)}{k} - W)$	$F(\frac{(2^k-1)\log_2 r}{k} + \frac{2n-1}{r})$
Worst-case capacity degradation	$(2W-2)^F$	$(2\log_2 r)^F$	$(\frac{2W}{k} - 1)^F$	$(\frac{2\log_2 r}{k})^F$
Cost of an incremental update	$O(W^F)$	O(N)	$O((\frac{W}{k})^F)$	O(N)
Overhead on the packet processor	None	Pre-computed table of size: $O((\log_2 r + \frac{2n-1}{r}) F.2^W)$ (*or*) O(nF) comparators of width W bits	$O(\frac{W.2^k}{k})$ logic gates	Both pieces of logic from previous two columns

Table 4: Comparison of key metrics (see Section 2.1 for the list of metrics) for different range encoding schemes.

number of levels l increases, the worst-case expansion increases, and since the chunk widths decrease, the width of the encoded range also decreases.

The pseudocode for forming the TCAM entries and the search key is presented above. Let R=$[s, e]$ denote the range, v_{s_i} and v_{e_i} denote the values of the chunk i of s and e respectively. We use the same example as above to illustrate the algorithm.

Note that the prefixes needed to represent R are $\{0001xxxx$, $000011xx$, $0000101x$, 00001001, $0010xxxx$, $001100xx$, $0011010x$, $00110110\}$—a total of 8 prefixes.

Recall that l=3, k_0=2, k_1=3, and k_2=3. Then, W'=$2^2 - 1+2^3 - 1+2^3 - 1$=17, v_{s_0}=0, v_{s_1}=1, v_{s_2}=3 and v_{e_0}=0, v_{e_1}=6, v_{e_2}=6. Note that R can be written as R=$[013 - 066]$, with the leading digit being a 2-bit number, and the trailing two digits being 3-bit octal numbers. The split chunk is 1. Following the algorithm, the ternary entries needed to represent R are $\{02x-05x = 000\ 00xxx11\ xxxxxxx, 013-017 = 000\ 0000001\ xxxx111, 060-066 = 000\ 0111111\ 0xxxxxx\}$—a total of 3 ternary entries.

3.3 Comparative Analysis and Evaluation of Range Encoding Schemes

Table 4 presents a summary of an analytical comparison of the different range encoding schemes based on the metrics (described in Section 2.1). We consider an ACL with N rules, with each rule having F range fields that are W bits wide. As mentioned in Section 3.1.1, the prefix expansion scheme expands to $(2W-2)^F$ entries in the worst-case.

For each range field, DIRPE with k-bit chunks results in W/k chunks. Each of the chunks takes 2^k-1 bits to represent, leading to a total of $(2^k-1).W/k$ bits. Hence, the number of extra bits needed is $(2^k-1).W/k-W$. The worst case expansion is $2W/k-1$, since there can be 2 entries corresponding to each of the W/k levels except the root level, which can have at most 1 entry. Finally, the additional logic introduced for modifying the search key can be implemented in a few hundred gates; the logic does not affect throughput of the search, though it adds a few cycles of latency to the search.

For region-based range encoding scheme, n unique ranges form at most $2n-1$ non-overlapping subranges, which are divided equally into r *regions*. $\log_2 r$ bits are needed to represent a region and $(2n-1)/r$ bits are needed to represent the non-overlapping subranges within that region, leading to a total of $\log_2 r + (2n-1)/r$ extra bits per entry. Any range can span many regions fully and at most 2 regions partially. The portion of the

Extra bits	DIRPE	Region-based Range Encoding
0	30	30
8	15	30
18	11	16
27	9	14
44	7	12

Table 5: Comparison of worst-case expansion of DIRPE and Region-based range encoding schemes [11, 22] for various extra bits available for a single field.To calculate worst case for region based scheme, we assume that the database has less than 1024 unique ranges. Note that using zero extra bits corresponds to using prefix expansion.

Extra bits	DIRPE	Region-based Range Encoding
0	2.69	2.69
8	2.08	2.33
18	1.79	2.17
36	1.57	1.58

Table 6: Comparison of expansion of DIRPE and database dependent region-based range encoding for a real-life database with 215K rules.

range in the fully spanned regions can be represented using prefix expansion using at most $2\log_2 r - 2$ entries, and the 2 partial regions using 1 entry each, leading to a worst-case expansion of $2\log_2 r$. We can further reduce the worst-case expansion of this scheme by applying DIRPE on the representation of the range on $W' = \log_2 r$ bits.

Table 5 compares the worst-case expansion of DIRPE and the region-based range encoding scheme for a database with 1024 unique ranges in a single field. For the latter scheme, the worst-case was calculated by picking the value of r which produced the lowest expansion while not exceeding the given number of extra bits. DIRPE outperforms the region-based scheme for the same number of additional bits used.

3.3.1 *Evaluation of DIRPE on Real-life and Random Databases*

Table 6 compares the expansion of DIRPE and region-based range encoding for a real-life database.[2] Despite being database-independent, DIRPE out-performs the database-dependent region-based scheme even on real-life databases. For this particular database, we found that there are 1408 unique ranges in one field and 256 unique ranges in another field. As the number of unique ranges increases, DIRPE would perform increasingly better. The fact that DIRPE has better worst-case expansion and update properties makes it an attractive choice in many systems.

In Figure 3, we plotted the relative size of the database that can be stored in a given amount of TCAM as a function of the number of DIRPE bits used, *i.e.,* assuming that one has a TCAM large enough to store a database, how larger a database can be stored in the same TCAM as a function of number of DIRPE bits. We observe that by using 32 total extra bits (recall that rules have two fields), we are able to accommodate about 50% more rules.

Figure 4 plots the variation of relative capacity improvement for random database with two range fields. The graph also plots the improvement in the worst-case bounds for comparison. Not surprisingly, the results for a random database are much better—by using 32 bits per entry for DIRPE, the stored database size can be doubled. When considering the worst-case prefix expansion, by using 32 bits for DIRPE, the size of the stored database can be quadrupled. To understand exactly how DIRPE improves the expansion, in Figure 5, we plot the frequency distribution of the number of rules

[2]We implemented the region-based range encoding scheme based on the description provided in [11].

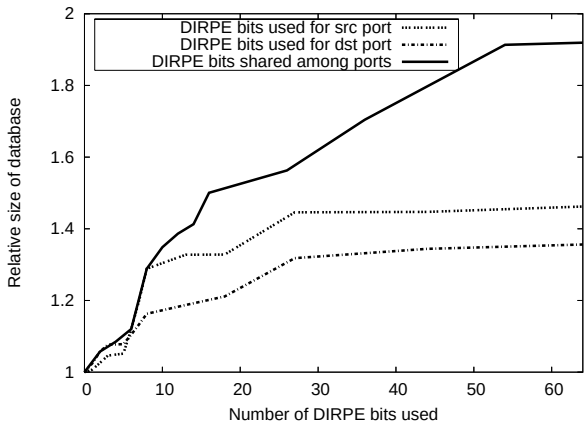


Figure 3: Relative size of the database that can be stored in a given amount of TCAM using DIRPE. Real-world database with 215K rules is used. The base for comparison (number of bits = 0) corresponds to expanding ranges to prefixes.

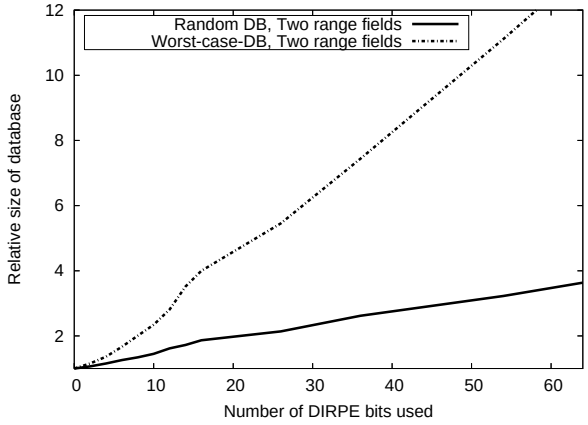


Figure 4: Relative size of random and worst-case databases that can be stored using DIRPE using various number of extra bits when rules have two range fields. The base for comparison (number of bits = 0) corresponds to expanding ranges to prefixes.

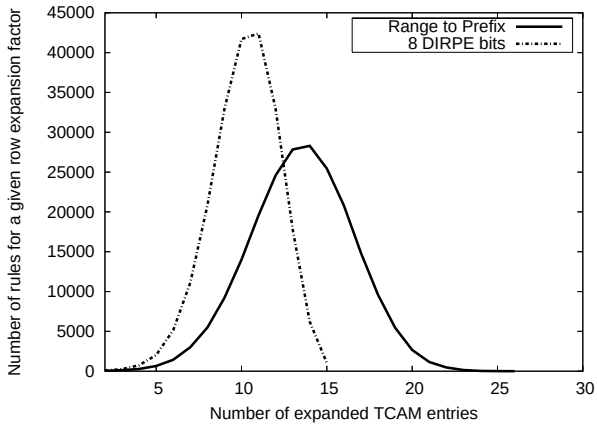


Figure 5: Distribution of expansion for different ranges on a random database with a single range field.

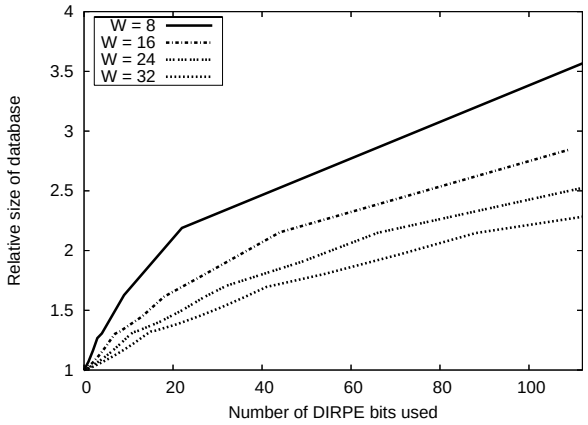


Figure 6: Variation of DIRPE effectiveness with width of field; the databases are randomly generated and have a single range field.

Statistic	1998 database Dest Port	2004 database Src Port	2004 database Dest Port
Range	1024-65535	1024-65535	16384-16480
Frequency	3683	20810	14999
Range	0-1023	1023-1340	1024-65535
Frequency	76	1328	10097
Range	33554-65535	0-1022	6970-7070
Frequency	39	1328	1328
Range	0-33432	1001-65535	2000-3467
Frequency	37	1328	1328
Remaining	Various	Various	Various
Frequency	401	847	7585

Table 7: Table showing frequency occurrence of various unique ranges in the 1998 and 2004 database. Note that the 1998 database does not have any source port range.

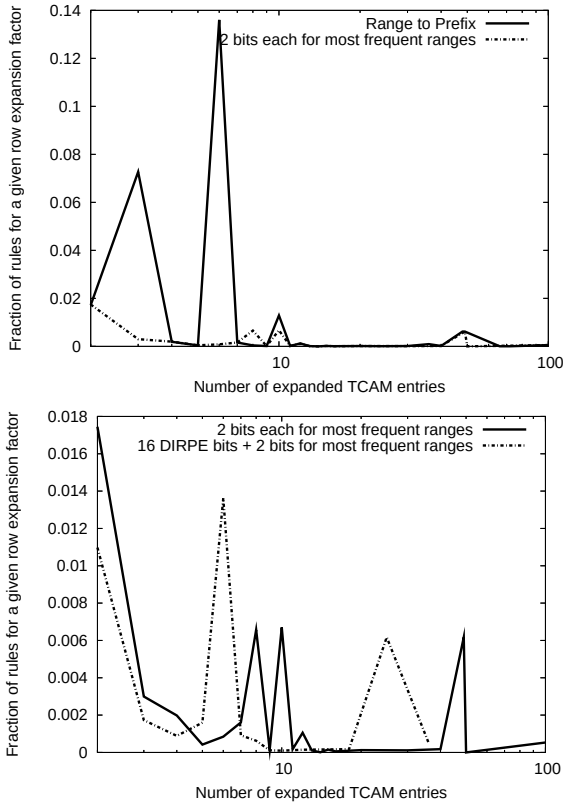


Figure 7: Distribution of expansion for different ranges on a real-life database with two range fields. The top figure shows how the frequency-based range encoding improves prefix expansion. The bottom figure shows how DIRPE further improves the expansion.

that expand to a given number of TCAM entries for prefix expansion and for DIRPE using 8 bits. Compared to prefix expansion, DIRPE pushes the distribution to the left uniformly, instead of reducing the expansion of a subset of ranges decided by the database.

An interesting question is how the effectiveness of DIRPE varies for different widths of the range field. Note that for fields of larger width, database-dependent schemes require prohibitively large support logic in the packet processor when databases have more than a few tens of unique ranges. Figure 6 shows the effect of number of extra bits used for various widths of the range field. Since the real-life databases have only 16-bit range fields, we used only random databases for this evaluation. From the figure, we observe that by using W extra bits for DIRPE (where W is the field width), there is a 50% increase in the database size that can be supported, and by using $2W$ extra bits, there is a 80% increase in the database size.

3.4 Hybrid Approaches

If a database has a few ranges predominantly, database-dependent schemes can be used in conjunction with DIRPE, while still retaining the property that incremental updates are efficient. Here, we consider a very simple variant of the database-dependent scheme—the k most frequent ranges are computed from the database, and a single bit is assigned to each of the ranges, thus reducing the expansion of all those ranges to 1.

Figure 7 plots the frequency distribution of the rules that expand to a certain number of TCAM entries for a real-life database. The first graph shows that using 2 bits per range field (based on the frequency of occurrence of a range in the database) gives significant improvement. The second graph shows that even when the database-dependent scheme is used, DIRPE gives further improvement over and above the database-dependent scheme.

Finally, we note that we found the usefulness of DIRPE is greater in the newer database as compared to the older one. If the trend of more range rules and more unique ranges continues (as illustrated by Table 7, which presents the frequency distribution of unique ranges in the databases from 1998 and 2004), we believe that the benefits of DIRPE will further increase in the future.

3.5 Practical Considerations in Using DIRPE

Recall from Section 2 that we have 36 extra bits available. For tables with a single 16-bit range field, the DIRPE scheme with strides $4, 3, 3, 3, 3$ can be used. [3] This encoding would use up 27 bits and reduce the worst-case range expansion from 30 entries per rule to 9 entries per rule. The remaining 9 bits can be used to encode the frequent ranges to achieve better real-life capacity.

For tables with two range fields, the field with more unique ranges can use strides $2, 2, 3, 3, 3, 3$, and the other field can use strides $2, 2, 2, 2, 2, 3, 3$. The extra bits used will be 18 and 13 respectively. The remaining 5 bits can be used to encode the frequent ranges for each of the two fields. These choices reduce the worst-case expansion from 900 to 143 entries per rule. Applying this on the 215K database, the expansion reduces from 2.69 to 1.12, a factor of more than two. Reducing the amount of TCAM needed on a linecard by a factor of two today is significant, and the promise of larger savings going forward (based on the database trends we have observed) makes this scheme attractive.

[3] For reducing the worst-case expansion for a given number of extra bits, we choose equal-width strides.

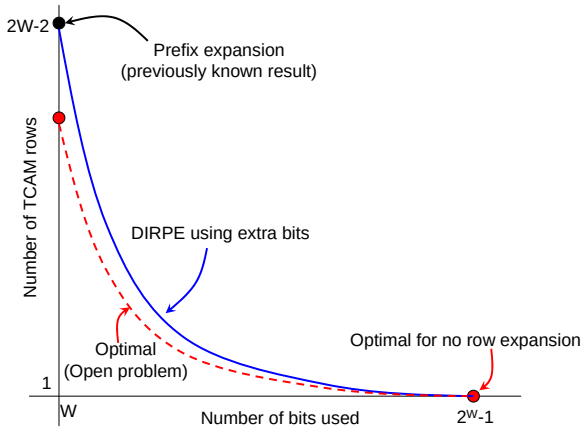


Figure 8: Our contribution towards database-independent range encoding schemes from a theoretical perspective. The previously known result was expansion of ranges to prefixes. Our algorithm DIRPE uses extra unused bits to achieve better worst-case expansion. For achieving a row expansion of 1, we have shown that $2^W - 1$ bits are necessary, but finding the optimal database-independent encoding is an open problem.

3.6 Future Directions

We have shown how we can reduce the row expansion of range rules by using the additional bits in a TCAM array in a database-independent manner. The question then is how far from optimal is DIRPE in the following sense: To bound the worst case row expansion to E, what is the minimum extra bits needed, and what is the corresponding encoding? Given B extra bits, what is the least worst case row expansion, and what is the corresponding encoding?

When the desired row expansion is 1, we have shown that $2^W - 1$ is necessary, but finding the optimal database-independent encoding is an open problem. Figure 8 captures the state-of-the-art in database-independent range encoding to the best of our knowledge.

4. MULTI-MATCH CLASSIFICATION

Traditional packet classification requires that, for a given search key, the best matching rule be found. However, recently, many applications such as load balancers and intrusion detection systems require finding multiple (or sometimes all) matches. TCAMs report only the first matching entry. To enable these applications, schemes to find multiple matches are necessary. We now formally define the multi-match classification problem.

Multi-match Classification Problem:

Consider a database $\mathbf{F_{dat}}$ consisting of N rules with cost $cost(F_i)$ associated with each rule F_i. The *multi-match classification problem* for finding (at most) k rules that match a search key S is defined as follows.

Find rules $F_1^{mult}, \ldots, F_k^{mult}$ in $\mathbf{F_{dat}}$ such that:

- Each of F_i^{mult} is a rule match for S.
- There is no other rule F_j in $\mathbf{F_{dat}}$ such that F_j is a match for S and $cost(F_j) < cost(F_i^{mult})$ for some $i \in [1, k]$.

We define the *multi-match degree* of an ACL database as the maximum number of rules that can potentially match a key. In other words, if the multi-match degree of a database is M, then there exists a key S such that M rules match S and there is no key that matches more than M rules. Figure 9 shows an example in which the multi-match degree is 3.

Figure 10 shows the distribution of multi-match degree across 112 ACLs in a router database with a total of $215K$ rules. The

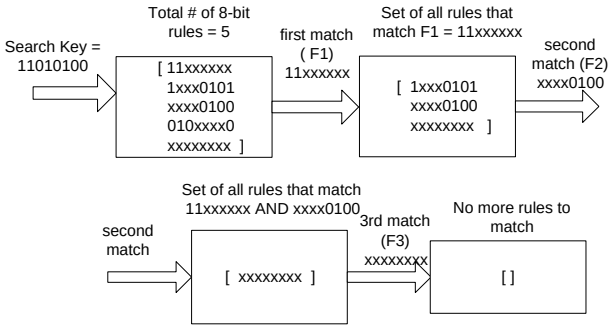


Figure 9: Example showing set of possible matches at every step of multi-match classification.

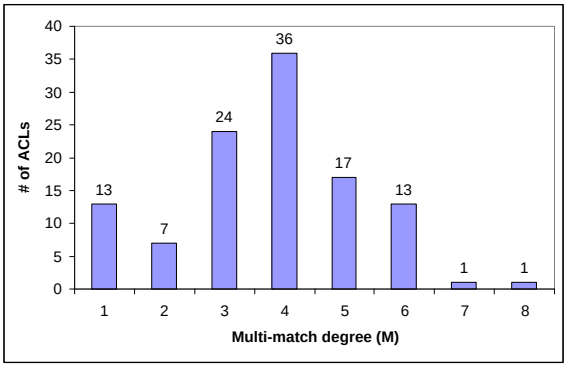


Figure 10: Frequency distribution of multi-match degrees (maximum number of rules that can match any key) for 112 ACLs.

largest ACL had 11781 rules. We now present a summary of the multi-match characteristics we observed from this database.

- The maximum multi-match degree is 8. The set of rules in one such multi-match group is shown in Figure 11.
- Most ACLs have a multi-match degree of 4 or 5.
- The multi-match degree does not correlate well with the size of the databases. Some ACLs with as few as 97 rules as well as some with as high as 3060 rules both have the same multi-match degree of 5. Another ACL with only 340 rules has a multi-match degree of 8.
- The snort database [2] has 2700 signatures, each having an ACL specified on the packet header, and further patterns based on packet content. We found that the set of ACLs that were found in all these signatures comprised 276 rules. The multi-match degree of all the unique ACL headers was 4.

Thus, real databases have several multiple matching rules; we now review earlier approaches to find multiple matches.

4.1 Earlier Approaches

4.1.1 *Entry-Invalidation Scheme*

Entry-invalidation scheme, one of the earliest and simplest schemes, maintains the state of a multi-match search in the database itself. A *valid bit*, an additional bit in the TCAM array, is associated with each entry in the TCAM. Searches are performed over a subset of TCAM entries by setting the valid bits for those entries only. When the search starts, the valid bit is set for all the entries. Let the rule that matches be F_j. Now, the valid bit of F_j is unset

```
access-list 105 tcp 81.184.207.0 0.0.0.255 148.79.89.193 0.0.0.0 eq 6000
access-list 105 tcp 81.184.207.0 0.0.0.255 148.79.89.193 0.0.0.0 gt 1023
access-list 105 ip 81.184.207.202 0.0.0.0 148.79.89.193 0.0.0.0
access-list 105 tcp 81.184.207.202 0.0.0.0 148.79.89.193 0.0.0.0 lt 20608
access-list 105 tcp 81.184.207.202 0.0.0.0 148.79.89.193 0.0.0.0 lt 20616
access-list 105 tcp 0.0.0.0 255.255.255.255 0.0.0.0 255.255.255.255
access-list 105 ip 0.0.0.0 255.255.255.255 148.0.0.0 0.255.255.255
access-list 105 ip 0.0.0.0 255.255.255.255 0.0.0.0 255.255.255.255
```

Figure 11: Real-life example of a multi-match group with multi-match degree of 8. *ip* rules will match *tcp* rules as well, since TCP is an IP-based protocol.

```
Multi-Match_MUD (key)
  Initialize discriminator prefix list:
    D ← {'xx...xx'}
  while (D is not empty)
    d ← D.pop(). Let d represent the range [s, e].
    R ← TCAM_Search(d, key)
    if (R != NULL)
      Let i be the index of rule R
      Let D' be the set of discriminator prefixes
        for the range [i + 1, e]
      D ← D.push(D')
  end-while
  return TCAM_Match_List
```

Figure 12: MUD search logic in the packet processor interfacing with TCAMs. The control plane software sets up the TCAM entries and chooses discriminators for each of the entries appropriately.

and the same search key is issued again. The process is repeated till the required number of (or no more) matches are found.

The total number of cycles taken for finding k matches is $k.(T_{invalidate\_write\_cycle} + T_{search\_cycle} + T_{revalidate\_write\_cycle})$. The search operation typically takes a single cycle while writes take 3 cycles. Hence, the total number of cycles per multi-match is 7.

The entry-invalidation scheme alters the state of the database during the course of the algorithm. Hence, it is infeasible to use this scheme in a multi-threaded environment such as today's packet processors, where searches are issued from multiple packet processing threads that have simultaneous access to the TCAM device.

To support multiple threads, the entry-invalidation scheme can be extended trivially to use many valid bits per TCAM entry, with each valid bit keeping track of a single thread (independent of the other bits). However, as the number of threads increases (today's packet processing systems support at least 64 threads and the number is rising), the overhead of the valid bits becomes prohibitive. For example, the Intel IXP2800 network processor supports up to 256 threads [10]. Since the width of each TCAM entry is fixed and allocated in discrete quantities (72, 144, 288 or 576 bits), allocating a large number of bits as valid bits might not be feasible as it would severely affect the size of databases that can be stored.

4.1.2 Geometric Intersection-based Scheme

The scheme described in [25] constructs the set of matching geometric intersections (cross-products) of fields and places them in the TCAM. While this elegant scheme has a high search throughput, it does not scale well in capacity—the number of TCAM entries needed per rule in an ACL might be very large. For all the 112 ACLs in our database, we noticed an expansion factor of 25-100. For example, we found a real-life ACL of 32 rules leading to 11263 TCAM entries.

In the next section, we present our multi-match scheme, which works well in multi-threaded systems, provides deterministic worst-case search bounds and scales in memory usage.

4.2 Multi-match Using Discriminators (MUD)

We first present the basic idea behind MUD. Let the result of searching a TCAM with a key be the rule with index j. To get the next matching result from the TCAM, we need to perform the search on all the entries *after index j*. To accomplish this, we use a simple idea: along with each TCAM entry, store a discriminator field that encodes the index of that entry. The TCAM entries after index j have discriminator field values that are greater than j. We expand '$> j$' to prefixes, and specify these prefixes in the discriminator field in subsequent searches to search through the TCAM entries that appear after j. The benefits that MUD offers—support for multi-threaded environment and low update cost—come at the expense of search cycles; however, as we show in Table 8, MUD can still support multi-match classification at multi-gigabit link speeds.

To specify prefixes in the search key, we use a well-known search capability called *global masking* that TCAMs provide. When a key is searched, TCAMs allow each bit position to be *masked out*, *i.e.,* set to x. If a bit position is masked out, then that bit position in the key will not be compared against the corresponding bit in each entry, but will be deemed to have matched. For example, using a global mask $xxxx111$ would mean that only the 3 least significant bits are actually compared, and the 4 most significant bits are not compared.

We now describe the algorithm in detail. To the original set of rules, $R_1, \ldots, R_N$, we prepend a discriminator field of d bits which indicates the index of the rule within the ACL; *i.e.,* rule R_i will have the value i in the field. The minimum number of bits required for the discriminator field for a database with N rules is $d = \log_2 N$.

When the search for a key S starts, the discriminator field in the key is masked out completely, resulting in the entire database being searched. Let the first match be rule R_j. The next search has to consider rules with index greater than j, *i.e.,* rules with discriminator greater than j. By using discriminators we have reduced the multi-match problem to the range representation problem. In the case of DIRPE, the rules had ranges that had to be represented as ternary strings. In the case of MUD, the discriminator in the search key has a range that needs to be represented as ternary strings.

Let us consider an example in which we use 4 bits for the discriminator field (*i.e.,* the database has at most 16 entries). Let the first match occur at index 5 (*i.e.,* 0101). The discriminator prefixes needed for searching the rest of the database (starting from index 6) are $011x$ and $1xxx$, representing the ranges $[6, 7]$ and $[8, 15]$ respectively. If any match is found with any of these two prefixes as the discriminator field in the search key, then the process is repeated recursively.

Figure 12 shows the steps that are needed to issue search keys for locating matches—it is very simple and can be easily included in any device interfacing with the TCAM.

4.3 Improving Performance of MUD

We now describe a few optimizations for improving the performance of MUD.

4.3.1 Assigning Discriminators to Sets of Entries

In the baseline MUD scheme, for a database with N rules, each rule is given unique discriminator values between 0 and $N-1$, resulting in $\log_2 N$ bits for the discriminator field. We present an optimization based on the following intuition: if several rules can be grouped into a set such that any search key can match at most one rule in this set, then all the rules in this set can be given the same discriminator value. For example, for getting all the matches from a database of prefixes, we would set the discriminator value to be the length of a prefix. This is because any key can match only one prefix among the prefixes of the same length.

A mask of a rule is a bit string indicating the location of specified bits (0 or 1) and wildcard bits (x) in the rule. To compute the mask, a 0/1 in the rule is replaced with a 1 and an x is replaced with a 0. For example, two rules $10x11xx0$ and $11x01xx1$ have the same mask 11011001. In an ACL database, there can only be one matching entry among rules with the same mask. Hence, the same discriminator value can be assigned to all entries with the same mask. This optimization would reduce the number of bits needed for the discriminator to $\log_2$(number of distinct masks). However, since order of multi-matches is not retained, for obtaining the first k matches, one would need to find all matches and pick the first k.

Our database has 215183 rules, which when expanded using range-to-prefix conversion correspond to 1694 distinct masks. Hence, for our database, the discriminator field can be encoded using 11 bits.

4.3.2 Assigning Discriminators using DIRPE

To represent N discriminator values (which could be the number of unique masks), at least $\log_2 N$ bits are needed. We now show that, by using a *wider* representation for the discriminator, we can reduce the worst-case number of searches.

Recall that to address the range expansion problem, we proposed DIRPE, which reduced the expansion of ranges by using additional bits. Analogous to the range expansion problem in which ranges appear in the ACL rules and are mapped to multiple TCAM entries, in the multi-match problem the ranges appear in the search key leading to multiple TCAM searches. Drawing from the analogy, for MUD, we can use the DIRPE algorithm to encode the discriminator keys corresponding to searching the range '$> j$', thus reducing the worst-case number of searches. Hence, even though the minimum number of discriminator bits needed is $\log_2$(number of distinct masks), by using DIRPE, we can add extra bits to reduce the worst-case number of searches.

Furthermore, in the case of MUD, since only ranges of the type '$> j$' are needed, the worst-case number of ternary entries is equal to the number of chunks used in DIRPE.

$d = \log_2$(number of unique discriminator values) is the minimum number of bits needed for the discriminator field. Dividing this into chunks of r bits each gives d/r chunks. Note that MUD issues several searches to cover all chunks. Hence we use the following encoding for the discriminator field: $\log_2(d/r)$ bits to indicate the chunk-id, $d - r$ bits to represent the bits leading up to the chunk and $2^r - 1$ bits to encode the range in the chunk. Thus, the discriminator width, d', is $\log_2(d/r) + d - r + 2^r - 1$.

Table 8 shows the links speeds that can be supported for various values of unique discriminators, number of discriminator bits using DIRPE and number of matches per multi-match. The main trend we observe is that for the same number of unique discriminators, greater the number of discriminator bits used (using DIRPE), higher is the link speed that can be supported. For example, compare the first three rows of the table. Also, the trend is independent of the number of discriminator values (see last two rows of the table for links speeds with 2048 unique discriminators) and the number of multi-matches (rows 4-6 show the same trend for 5 multi-matches). Since MUD does not increase the number of TCAM entries, we can replicate the tables if higher search throughput is desired.

# of Unique Disc. Values	Max. matches per multi-match	Disc. width using DIRPE	Link Gbps with MUD
512	4	9	2.40
512	4	13	5.00
512	4	15	7.81
512	5	9	1.84
512	5	13	3.91
512	5	15	6.25
2048	4	11	1.95
2048	4	15	4.03

Table 8: Link speed for finding all matches using the MUD scheme for different values of discriminator bits. We assume a base TCAM throughput of 125 million searches per second. Minimum size packet size = 64 bytes for wire speed operation (See [1]). 5MPPS corresponds to $2.5G$bps.

4.3.3 A Set Pruning-Based Approach

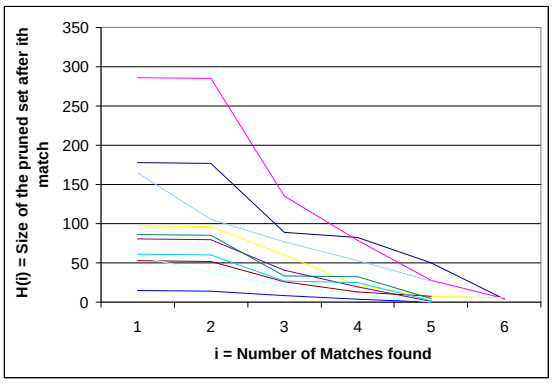
We also considered an algorithm that is based on set pruning approach for improving the performance of MUD. The algorithm is based on the simple idea: When some matches are found, the total number of entries that can *possibly* match the search key must reduce. Hence, after finding the first few matches, if the list of potential matches is small, then they can be searched quickly by a linear search.

Define a *pruned set* as the list of entries that can potentially match a key after i matches are found. Denote the size of the largest such pruned set after i matches by H_i. Using the real-life database, we found that H_i does not decrease below 10 till several matches are found (see Figure 13). In addition, the time for precomputation of pruned sets was also very large. Though this simple heuristic proved ineffective, our experiments shed some light on the characteristics of real-life databases.

4.4 Comparative Analysis of MUD and Other Schemes

Table 9 presents the comparison of MUD with earlier approaches based on the metrics in Section 2.1. The invalidation scheme does not work well in multi-threaded systems. Both the invalidation scheme and MUD require only one TCAM entry per rule, but the geometric intersection scheme does not scale well in the worst-case number of TCAM entries needed per rule. Even for the real-life database we used, for all the 112 ACLs, the expansion factor was between 25 and 100.

When rules change, updating ACL tables has traditionally involved manual intervention, and hence the ability to perform incremental updates has not been a serious requirement. But with the adoption of automated intrusion detection, there is an increasing need to update the tables incrementally. For MUD, the discriminator field in the entries need to be updated with their new index location. Hence, both the entry-invalidation scheme and the MUD scheme support high update rates—cost of updating N rules is $O(N)$ —same as that required to maintain the entries for single match classification. Update rates of few ten thousand up-



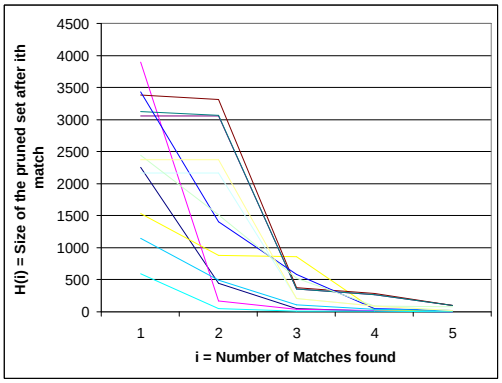


Figure 13: Size of pruned sets for ACLs with (a) < 300 rules (on the left), (b) > 500 rules (on the right). The x-axis shows the number of matches completed during a multi-match classification. The y-axis is the size of the largest pruned set after the i-th match has been found.

Metric	Entry Invalidation	Geometric Intersection-based	MUD
Multi-threading support	No	Yes	Yes
Worst-case TCAM entries for N rules	N	$O(N^F)$	N
Update cost	O(N)	$O(N^F)$	O(N)
Cycles for k multi-matches	7k	1	1 + d + (d-1)(k-2) with DIRPE: $1 + \frac{d(k-1)}{r}$
Extra bits	0	0	without DIRPE: d with DIRPE: $\log_2(d/r) + (d-r) + (2^r-1)$
Overhead on the packet processor	Small state machine logic; can be implemented using a few hundred gates or a few microcode instructions	None	Small state machine logic; can be implemented using a few hundred gates or a few microcode instructions

Table 9: Comparison of key metrics for multi-match schemes. k is the number of matching rules to be found, N is the number of rules in the database and F is the number of fields in a rule. $d = \log_2$(number of unique discriminator values). r is the chunk width ≥ 2

dates per second can be easily achieved. In contrast, the geometric intersection-based scheme can require several minutes of recomputation each time a new rule is added.

MUD supports high density, fast updates and multi-threading at the cost of extra searches through the TCAM. We now present a simple worst-case analysis for the number of TCAM searches for finding k matches. After the first match i is found, the subsequent searches correspond to the discriminator prefixes needed to represent $> i$, which is at most d prefixes. After the second match, note that the worst-case number of prefixes is at most $d - 1$. Hence, for finding k matching rules, when $d \geq (k - 1)$, the worst-case number of total searches is $1 + d + (k - 2)(d - 1)$. When using DIRPE with $d' = 1 + (d/r)(k - 1)$, the search throughput is further increased. The cost of additional searches does not affect the multi-match performance adversely: as shown in Table 8, MUD can support multi-match classification at multi-gigabit link speeds.

4.4.1 *Practical Considerations in Using MUD*

Both DIRPE and MUD use extra unused bits from the TCAM array. Hence, when DIRPE and MUD are used simultaneously (*e.g.,* when multi-match is performed using a database containing range rules), the available extra bits must be shared between the schemes based on the desired performance and density. Recall that typical TCAMs today have about 36 extra bits when used with IP ACLs. If MUD uses 12 bits (the large real-life dataset we used required 11 bits for MUD), DIRPE would have 24 bits. A possible way of splitting the bits is to assign 16 DIRPE bits to the first range field and 8 bits to the second range field; using such a split, we get an expansion of 1.31—which is comparable to the expansion obtained when all the bits were used for DIRPE.

4.5 Future Directions

While the deterministic search throughput of TCAMs makes it attractive compared to RAM, the cost factor makes it difficult to scale TCAMs to millions of rules. We plan to investigate how one can use a combination of TCAM and RAM to scale both single and multi-match classification to millions of rules with high search performance in real-life databases. Here, we present the basic idea we plan to pursue, and present preliminary results that show that the idea holds promise.

Several schemes for single match classification that divide the rules into buckets have been proposed [14, 21, 24]. If we partition the rules into buckets of size T using one such scheme, then we can store one TCAM entry corresponding to each bucket and store all the rules of a single bucket in RAM. During a search, the TCAM is first searched and then the buckets corresponding to all matching TCAM entries are searched. The number of rules that need to be accessed from RAM is then $M \times T$, where M is the multi-match degree of the TCAM. Since we found that even large router databases have small multi-match degree (see Figure 13), we expect this technique to work well in real databases. A similar observation on the two prefix fields in the rules is made in [1] and a corresponding extended grid-of-tries with path compression is described.

We implemented a simple variation of this heuristic in which we recursively walk down the rule tree, splitting the rules based on whether the bit in the rule is 0, 1, or x. Figure 14 shows the variation of the number of TCAM entries and the number of RAM entries that need to be accessed per classification, as a function of the bucket size, T. The desired search rate dictates the number of RAM accesses allowed per search. From the figure, we see that the number of TCAM entries needed reduces as the number of RAM accesses increases. Hence, depending on system requirements—available RAM bandwidth, desired search throughput and cost—we can choose the point in the tradeoff curve by using the appropriate value of the bucket size.

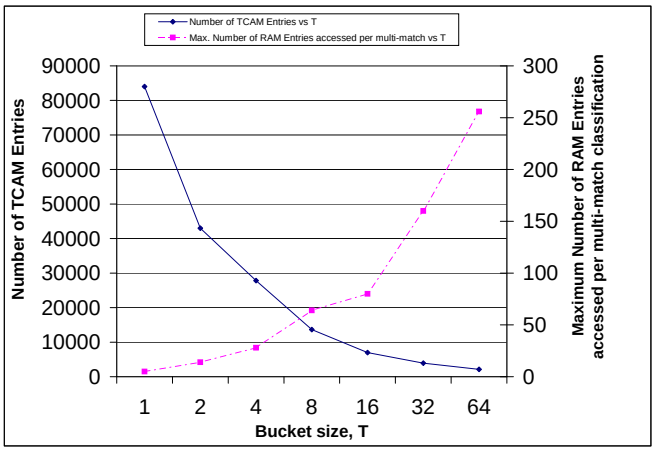


Figure 14: Variation of number of TCAM entries to be stored and maximum number of RAM entries to be accessed per classification, as a function of bucket size, T. The total number of RAM entries to be stored is the total number of rules in the database (after converting ranges to prefixes). A representative subset of the database was used.

5. CONCLUSIONS

Ternary CAMs have been widely used in the industry for packet classification for databases with up to hundreds of thousands of rules. In this paper, we presented algorithms that further the state-of-the-art for solving two important problems that arise while using TCAMs. Our first algorithm, *Database Independent Range PreEncoding* (DIRPE), addresses the problem of efficient representation of range rules in TCAMs. DIRPE reduces the worst-case expansion of range rules, scales to a large number of ranges, and has good incremental update properties.

Our second algorithm, *Multi-match Using Discriminators* (MUD), addresses the issue of finding multiple matches for a search key. The key benefit of MUD is that it does not store per-search state and hence is suited for multi-threaded environments. MUD does not increase the number of TCAM entries and hence scales to large databases. The benefits of MUD are obtained at the expense of additional searches; but we show that MUD can still support multi-match classification at multi-gigabit link speeds.

Our schemes do not require any change to TCAMs. Both the algorithms rely on extra bits in the TCAM entry; they can be used in conjunction by using disjoint sets of extra bits in the TCAM entry. We evaluated the algorithms using a large real-life router ACL database, using a randomly generated database, as well as using worst-case analysis.

We believe that the following future directions are interesting. The first direction deals with finding the optimal encoding of ranges when a certain number of ternary bits are available. The second direction is to investigate how TCAMs and RAMs can be combined to achieve deterministic search throughput at low costs while scaling to real-life databases with millions of rules; the preliminary results based on the simple heuristic we considered are encouraging.

6. REFERENCES

[1] F. Baboescu, S. Singh, and G. Varghese. Packet Classification for Core Routers: Is there an Alternative to CAMs? In *Proc. of IEEE INFOCOM*, 2003.

[2] J. Beale. *Snort 2.1 Intrusion Detection, Second Edn.* Syngress, 2004.

[3] J. Bolaria and L. Gwennap. A Guide to Search Engines and Networking Memory. `http://www.linleygroup.com/Reports/memory_guide.html`, April 2004.

[4] Computer Associates. eTrust Intrusion Detection System. `http://www3.ca.com`.

[5] Cypress Semiconductor Corp. Content addressable memory. `http://www.cypress.com/`.

[6] Dell'oro. Layer3 Switch Router Market. July 2004.

[7] W. Eatherton. Full Tree Bit Map: Hardware/Software IP Lookup Algorithms with Incremental Updates. *EE Masters Thesis, Washington University*, April 1999.

[8] P. Gupta and N. McKeown. Packet Classification on Multiple Fields. In *Proc. of ACM SIGCOMM*, 1999.

[9] Integrated Device Technology, Inc. Content addressable memory. `http://www.idt.com/`.

[10] Intel Corp. Intel IXP2800 Network Processor. `http://www.intel.com/design/network/products/npfamily/ixp2800.htm`.

[11] H. Liu. Efficient Mapping of Range Classifier into Ternary-CAM. In *Proc. of Hot Interconnects*, 2002.

[12] Netlogic Microsystems. Content addressable memory. `http://www.netlogicmicro.com/`.

[13] S. Nilsson and G. Karlsson. Fast Address Look-Up for Internet Routers. *Proceedings of IEEE Broadband Communications'98, Stuttgart, Germany*, April 1998.

[14] L. Qiu, G. Varghese, and S. Suri. Fast Firewall Implementations for Software and Hardware-based Routers. In *Proc. of ICNP*, 2001.

[15] S. Singh, F. Baboescu, G. Varghese, and J. Wang. Packet Classification using Multidimensional Cutting. In *Proc. of ACM SIGCOMM*, 2003.

[16] E. Spitznagel, D. Taylor, and J. Turner. Packet Classification Using Extended TCAMs. In *Proc. of ICNP*, 2003.

[17] V. Srinivasan and G. Varghese. Faster IP Lookups using Controlled Prefix Expansion. *ACM TOCS*, February 1999.

[18] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and Scalable Layer Four Switching. In *Proc. of ACM SIGCOMM*, 1998.

[19] D. E. Taylor. Survey and Taxonomy of Packet Classification Techniques. Technical Report WUCSE-2004-24, Washington Univ., St. Louis, May 2004.

[20] D. E. Taylor and J. Turner. Scalable Packet Classification using Distributed Crossproducting of Field Labels. Technical Report WUCSE-2004-38, Washington Univ., St. Louis, 2004.

[21] P. Tsuchiya. A Search Algorithm for Table Entries with Non-contiguous Wildcarding. Unpublished document, 1992.

[22] J. van Lunteren and T. Engbersen. Fast and Scalable Packet Classification. *IEEE JSAC*, 21:560–571, May 2003.

[23] G. Varghese. *Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices*. Morgan Kaufmann Publishers, Inc., 2004.

[24] T. Y. C. Woo. A Modular Approach to Packet Classification: Algorithms and Results. In *Proc. of IEEE INFOCOM*, 2000.

[25] F. Yu and R. H. Katz. Efficient Multi-Match Packet Classification with TCAM. In *Proc. of HotI*, 2004.