This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2009 proceedings.

Layered Interval Codes for TCAM-based Classification*

Anat Bremler-Barr
School of Computer Science
The Interdisciplinary Center
Herzliya, Israel
Email: bremler@idc.ac.il

David Hay†
Department of Electrical Engineering
Politecnico di Torino
Torino, Italy
Email: hay@tlc.polito.it

Danny Hendler
Computer Science Department
Ben-Gurion University
Beer-Sheva, Israel
Email: hendlerd@cs.bgu.ac.il

***Abstract*—Ternary content-addressable memories (TCAMs) are increasingly used for high-speed packet classification. TCAMs compare packet headers against all rules in a classification database in parallel and thus provide high throughput.**

TCAMs are not well-suited, however, for representing rules that contain range fields and prior art algorithms typically represent each such rule by multiple TCAM entries. The resulting *range expansion* can dramatically reduce TCAM utilization because it introduces a large number of redundant TCAM entries. This redundancy can be mitigated by making use of extra bits, available in each TCAM entry.

We present a scheme for constructing efficient representations of range rules, based on the simple observation that sets of disjoint ranges may be encoded much more efficiently than sets of overlapping ranges. Since the ranges in real-world classification databases are, in general, non-disjoint, the algorithms we present split ranges between multiple *layers* each of which consists of mutually disjoint ranges. Each layer is then coded independently and assigned its own set of extra bits.

Our layering algorithms are based on approximations for specific variants of interval-graph coloring. We evaluate these algorithms by performing extensive comparative analysis on real-life classification databases. Our analysis establishes that our algorithms reduce the number of redundant TCAM entries caused by range rules by more than 60% as compared with best range-encoding prior art.

I. Introduction

Packet classification is an indispensable building block of numerous Internet applications in the areas of routing, monitoring, security, and multimedia. The routers use a *classification database* that consists of a set of *rule*s (a.k.a. *filter*s). Each such rule specifies a pattern, based on packet header fields, such as the source/destination addresses, source/destination port numbers and the protocol type. Each rule is associated with an action to apply to the packets that matched the pattern rule. Packet classification is often a performance bottleneck in the network infrastructure since it lies in the critical data path of routers. It is therefore important to design packet classification solutions that scale to millions of key search operations per second.

*This work was supported by a Cisco research grant.

†This work was done while the 2'nd author was a postdoc fellow in the Computer Science department at Ben-Gurion University.

Ternary content-addressable memory (TCAM) devices are increasingly used in the Industry for performing high-speed packet classification. TCAM enables parallel matching of a key against all entries and thus provides high throughput that is unparalleled by software-based solutions. A TCAM is an associative memory hardware device that can be viewed as an array of fixed-width entries. Each TCAM entry consists of ternary digits: 0, 1, or '*' (don't-care). When a key matches multiple TCAM entries, the TCAM returns the index of the first matching entry. This index is then used to locate the information specifying which actions to apply to the packet. For classification applications, TCAMs are typically configured to be 144 bits wide. This leaves a few dozens of unused bits, called *extra bits*, per each TCAM entry.

A significant obstacle to the efficient use of TCAMs for packet classification is the fact that they are not well suited for representing rules that contain *range fields*, such as port fields. The traditional technique for range representation is the *prefix expansion* technique [1], in which a range is represented by a set of prefixes, such that each is stored in a single TCAM entry. For example, the range $[1, 6]$ can be represented by the prefix set $\{001, 01*, 10*, 110\}$. Hence, a single rule may require multiple TCAM entries, resulting in *range expansion*. Range expansion was found to cause an increase of more than 16% in TCAM space requirements for real-word databases [2].

A. Our Contributions

We present a scheme for constructing efficient representations of range rules, by making efficient use of TCAM extra bits. The scheme is based on the simple observation that sets of disjoint ranges may be encoded much more efficiently than sets of overlapping ranges. Since the ranges in real-world classification databases are, in general, non-disjoint, the algorithms we present split the ranges between multiple *layers*, each of which consists of mutually disjoint ranges. Each layer is then coded independently and assigned its own set of extra bits. We call the resulting encoding scheme a *Layered Interval Code* (LIC). Roughly speaking, our goal is to find a minimum-size LIC code, where the *size* of a LIC code is the number of extra bits it uses.

We consider two problems related to the space-efficient

978-1-4244-3513-5/09/$25.00 ©2009 IEEE

construction of LIC codes. An instance of the *minimum-space LIC* (MLIC) problem consists of a set $\mathcal{S}$ of ranges. The MLIC problem is to output a LIC code for the ranges of $\mathcal{S}$ that uses a minimum number of bits. We call the second problem the *budgeted minimum-space LIC* (BMLIC) problem. An instance of BMLIC consists of a set $\mathcal{S}$ of *weighted* ranges and a positive integer b, the number of available bits. Under this restriction, it is required to output a LIC code for a maximum-weight subset $\mathcal{S}'$ of $\mathcal{S}$, such that the code size of $\mathcal{S}'$ is at most b.

This paper presents several novel algorithms that solve the MLIC and BMLIC problems. Our algorithms use approximations for specific variants of interval-graph coloring as layering building blocks and exhibit different tradeoffs between implementation complexity and space efficiency. We have conducted extensive performance analysis, comparing our algorithms with prior art, by using a large real-life classification database consisting of more than $223K$ rules. Our empirical results show that all of our algorithms reduce the average number of redundant TCAM entries required to represent a range rule decreases by more than 60% as compared to best range-encoding prior art.

In practice, classification databases change over time. To maintain space efficiency, every practical range encoding scheme whose encoding is a function of the database ranges-distribution (such schemes are named *database-dependent*) must change its encoding in accordance with these changes. First and foremost, it is crucial to guarantee *hot updates*; namely, that while TCAM entries are being updated, the device can still be used to classify incoming packets. Our scheme supports a novel and efficient hot updates mechanism that can be combined with any database-dependent encoding scheme (including ours). This algorithm requires only a single extra bit per TCAM entry.

On the theoretical side, we formalize the MLIC problem and prove that it is NP-hard by a reduction from the *circular arc graph coloring* problem, whose NP-hardness was established in [3]. We then prove that an algorithm presented in [4] for the closely related *chromatic sum* problem [5] is a polynomial-time 2-approximation algorithm for MLIC. We use this approximation algorithm as a building block in some of the algorithms we present. Finally, we formalize the BMLIC problem and use a reduction from the MLIC problem to establish that it also is NP-hard.

II. Background

The issue of using TCAM devices for packet classification has received considerable attention from the research community over the past few years. A key issue dealt with by researchers in this regard is that of improving the space-efficiency of TCAM memory range representation. This issue was considered both from the algorithmic [6]–[9] and the architectural [10] perspectives.

Spitznagel, Taylor, and Turner, introduced *Extended TCAM* [10], which implements range matching directly in hardware in addition to reducing power consumption by over 90% relative to standard TCAM. While this may represent a promising long-term solution, it seems that changing the ternary nature of TCAM entries while maintaining reasonable per-bit cost and addressing scalability issues will not be accomplished in the near future.

The traditional technique for range representation, originated by Srinivasan et al. [1], is to represent a range by a set of prefixes, each of which can be stored by a single TCAM entry. The worst-case expansion ratio when using prefix expansion for ranges whose endpoints are W-bits numbers is $2W-2$. The problematic range is $R_w = [1, 2^w - 2]$, whose smallest cover set is $\{01{\star}^{w-2}, 001{\star}^{w-3}, 0001{\star}^{w-4}, \cdots, 0^{w-1}1, 10{\star}^{w-2}, 110{\star}^{w-3}, \cdots, 1^{w-1}0\}$.

As observed by Taylor [2], a single rule that includes two 16-bit range fields could, in the worst-case, require $(2 \cdot 16 - 2)^2 = 900$ entries. Though such rules are rare in practice, analysis of real-world databases, provided by [2], revealed that prefix expansion may increase the number of required TCAM entries by a factor of more than 6.

Prior art that deals with the range expansion problem can roughly be divided to two main categories: database-independent and database-dependent range encoding algorithms. The encoding of a range by a database-dependent scheme is a function of the distribution of ranges in the database in which it occurs. In contrast, the encoding of a specific range by a database-independent scheme does not change across different databases. Lakshminarayana et al. present a database-independent range encoding algorithm that is based on the concept of *fence encoding* [8]. Bremler-Barr and Hendler [11] present a database-independent algorithm that is based on the observation that small ranges, which occur frequently in real-world databases, are encoded more efficiently by using *Gray code* [12]. Both these works also present hybrid versions that are database-dependent.

The first database-dependent prior art is due to Liu [6]. The basic idea is to use the available extra bits as a bit map: a single extra bit is assigned to each selected range r in order to avoid the need to represent r by prefix expansion. If range r is assigned extra bit i, then the i'th extra bit is set in all TCAM entries that include r; all other extra bits are set to 'don't care' in these entries. In the search key, extra bit i is set to 1 if the key falls in range r or set to 0 otherwise.

This basic scheme eliminates redundancy for every range that is assigned an extra bit. However, since the number of ranges whose redundancy may be eliminated is bounded by the number of extra bits, this solution does not scale. Indeed, as observed by [8], the number of unique ranges in today's classification databases is around 300 and is anticipated to continue to grow in the future. This growth is expected to accelerate, due to the use of TCAMs by Intrusion Detection Systems (IDS), which add new range fields, such as a packet-length field.

To alleviate this scalability problem, *Region Partitioning*

was proposed in [6]. This partitioning may split a range into multiple sub-ranges. Each such sub-range is encoded by two numbers: the region number into which it falls, and the sub-range number within that region. Unfortunately, in practice, this algorithm often results in high expansion (See our experiment results in Section VI), since a range is divided to multiple sub-ranges and the encoding of each sub-range required a separate TCAM entry.

Che et al. present a bit-map based scheme, called DRES, that employs a *dynamic range selection* algorithm for selecting the ranges that are to be assigned extra bits [13]. DRES is a greedy algorithm that assigns extra bits to the ranges with highest prefix expansion. The key difference between DRES and our scheme is that, whereas DRES assigns a single bit per range, we use the much more compact LIC coding. This allows our scheme to give better 'bang for the (extra) bit', since we are able to eliminate more redundancy using the same number of bits as compared with DRES.

A technique very similar to LIC was proposed in [7] by Lunteren and Engbersen but no explicit algorithm was presented by that work.

III. The Layered Interval Encoding Scheme

The LIC encoding scheme is based on the simple observation that, while encoding n arbitrary ranges may require n bits, only $\log(n+1)$ bits are required to encode n *disjoint* ranges. Our algorithm is composed of three stages: a *layering stage*, a *bit allocation stage*, and an *encoding stage*. Note that all these stages are done in a pre-processing phase, thus lookup operations still take only a single TCAM cycle.

In the *layering stage*, we partition the intervals[1] that occur in the database into a set of interval-sets $\mathcal{C}$, such that all the intervals in any interval-set $L \in \mathcal{C}$ are mutually disjoint. We call the set $\mathcal{C}$ the *layering* of the intervals. In Section III-B we propose several polynomial-time layering algorithms that are based on approximation algorithms for the BMLIC and MLIC problems.

In the *bits allocation stage*, we iterate over extra bits, deciding for each to which layer it should be assigned. Roughly speaking, the bit is assigned to the layer for which the total weight of intervals that can now be coded by using this bit is maximum. This process minimizes range-expansion, while not exceeding the extra bits budget. We describe the bits allocation stage in detail in Section III-C.

Finally, in the *encoding stage* we construct the corresponding search keys and TCAM entries, based on the outputs of the layering and the bits allocation stages. The selected ranges of each layer are encoded independently (of other layers) by using the extra-bits budget. Roughly speaking, a search key is constructed by encoding, for each layer, the single range of this layer to which the search key falls; if no such range exists, we encode a value representing the "area" outside all of this layer's ranges. A range-entry is constructed by encoding the range (at the single layer to which it belongs), while using 'don't care' bits for all other layers. This stage is described in more detail in Section III-D.

In the following, we describe each of the above stages in more detail. Before that, we define required terminology in Section III-A. For presentation simplicity, the following description assumes that each rule contains at most a single range field. This assumption is later removed in Section IV-A.

A. Terminology

A packet header consists of fields, each of which is a bit string. A *key* is a collection of κ fields from the packet header. Keys are matched against classification *rules* stored in *entries* of a *classification database*.

Rules consist of κ fields matching the corresponding key fields. A packet p with header $h(p)$ matches a rule R if each of the κ key fields of $h(p)$ matches the corresponding field of R. Each rule field f can specify one of three types of *matches*: *exact match prefix match* or *range match*.

In general, rules containing a range field (a.k.a *range rules*) cannot be represented by a single TCAM entry and *range encoding schemes* are used to encode each such rule as a set of TCAM entries. An encoding scheme maps each range r to a set of TCAM entries that represent it, called the *cover set of* r.

The quality of the range encoding scheme E is measured by the number of additional entries it requires for encoding range rules. Specifically, the *expansion* of a range r is the size of its cover set under scheme E, while the *basic redundancy* of a range r is defined to be r's expansion minus 1. The *expansion* of a rule R is the product of the expansion of all its κ fields, where the expansion of non-range fields is defined as 1. Similarly, the *basic redundancy* of a rule R is its expansion minus 1.

Let $\mathcal{D}$ be a classification database such that some of its rules contain range r. Under scheme E, the *total redundancy* of r in $\mathcal{D}$ is the product of r's basic redundancy and the number of rules of $\mathcal{D}$ in which r occurs. In other words, r's total redundancy is the total number of redundant entries that are required if we use encoding scheme E for r in $\mathcal{D}$. The *database expansion factor* of a database $\mathcal{D}$ using scheme E is the relative increase in the number of entries required to represent $\mathcal{D}$ in TCAM using coding scheme E.

Since this paper deals with encoding range fields, we focus our attention on rules containing such fields. The *range redundancy factor* of a database $\mathcal{D}$ using scheme E is the average number of TCAM entries required to represent the range-rules of $\mathcal{D}$ using E. Clearly, the lower the range redundancy factor of a scheme E is, the more efficient it is.

We consider two variations of the LIC problem: MLIC and BMLIC. Formally, these problems are coloring problems for

[1]Throughout this section, we mostly use the term 'interval' instead or 'range', since the approximations we employ are based on algorithms for interval-graph coloring.

interval graphs [2].

Definition 1: Let G be an interval graph. The *minimum space layered interval-code* (MLIC) problem is to find a legal coloring $\mathcal{C}$ of G that minimizes $\sum_{i=1}^{|\mathcal{C}|} \log_2(n_i+1)$, where n_i is the number of the nodes of G assigned color i by $\mathcal{C}$. We say that a coloring $\mathcal{C}$ of G *has LIC code-size* b if the expression above is b for that coloring. We let $L(G,\mathcal{C})$ denote the LIC code-size associated with coloring $\mathcal{C}$.

Let $\mathcal{I}$ denote the set of intervals that occur in the database. The input to the MLIC problem is the interval graph corresponding to $\mathcal{I}$ and its output is a layering of the intervals in $\mathcal{I}$ that uses a minimum number of extra bits. The MLIC problem is closely related to the *chromatic sum* (a.k.a. *sum coloring*) problem [5], in which vertices are colored using natural numbers and the *sum* of the colors needs to be minimized. Intuitively, the objective function of both these problems improves (i.e., decreases) as the distribution of intervals to colors becomes asymmetric.

Definition 2: Let $G=\langle V,E\rangle$ be a weighted interval graph such that $w(v)$ is the weight of vertex $v \in V$, and let $b \in \mathbb{N}$ be a LIC code-size budget. The *budgeted minimum layered interval-code (BMLIC)* problem is to find a subset $V' \subseteq V$ and a coloring $\mathcal{C}$ of the subgraph of G induced by V', such that $\sum_{v\in V'} w(v)$ is maximum under the constraint that $\mathcal{C}$ has LIC code-size of at most b.

Let $\mathcal{I}$ denote the set of *weighted intervals* that occur in the database, where the weight of each interval r in $\mathcal{I}$ is r's total redundancy in the database. Also, let b denote the number of extra bits in each TCAM entry. A solution of the BMLIC problem with inputs $\mathcal{I}$ and b returns a LIC encoding of size at most b for a subset $\mathcal{I}'$ of $\mathcal{I}$ that saves a maximum number of redundant TCAM entries.

Note that all the intervals in $\mathcal{I} \setminus \mathcal{I}'$ must be encoded using a *database-independent scheme*. The common practice is to encode them using the *prefix expansion scheme*, but in the general scheme it can be any other database-independent scheme: A database-independent scheme E' is a *fall-back scheme* for database-dependent scheme scheme E, in case all the ranges that E cannot encode using the available extra bit budget are encoded using E'.

B. The Layering Stage

Let $G=\langle V,E\rangle$ be an interval graph. We evaluate the following four layering algorithms:

a) Maximum Size Independent Sets (MSIS): This is a greedy layering algorithm that works iteratively. Let $G_0=G$. In iteration $i \geq 1$, the algorithm finds a *maximum size independent set*, L_i, of the interval graph $G_{i-1}=\langle V_{i-1},E_{i-1}\rangle$ and defines $G_i=\langle V_i,E_i\rangle$ to be the subgraph of G_{i-1} induced by $V_{i-1} \setminus L_i$. The algorithm stops when V_i is empty. Let $\mathcal{C}=\{L_1,L_2,\ldots\}$. $\mathcal{C}$ is a coloring of G since if two vertices belong to the same layer L_i they are not adjacent in G. Bar-Noy et al. [14] show that MSIS is a 4-approximation to the chromatic sum problem.

[2]An *interval graph* is an undirected graph $\langle V,E\rangle$. Each node $v \in V$ corresponds to an interval I_v. For every pair of distinct nodes $v,u \in V$, E contains an edge between v and u if and only if $I_v \cap I_u \neq \emptyset$. A (vertex) *coloring* of a graph G is an assignment of colors to the nodes of G such that no two adjacent vertices are assigned with the same color.

Algorithm 1 Maximum Size Colorable Sets (MSCS)

```
1: layering procedure MSCS(Graph G = ⟨V, E⟩)
2:     k := 0
3:     while |A_k| ≠ |V| do
4:         k := k + 1
5:         A_k := Maximum size k-colorable set of graph G.
                  ▷ Using ALGORITHM LB [4, PP. 114-115]
6:     end while
7:     if k = 1 then
8:         return {A_1}
9:     end if
10:    C_1 := MSCS(GRAPH(G, A_1))
11:    for i = 2 to k do
12:        C_i := MSCS(GRAPH(G, A_i \ A_{i-1}))
13:    end for
14:    return ⋃_{i=1}^{k} C_i
15: end procedure
```

b) Maximum Size Colorable Sets (MSCS): An *i-colorable* set of a graph G is a subset of G's vertices that can be colored with i colors. MSCS finds a maximum size i-colorable set of G, denoted A_i, for each $1 \leq i \leq \chi(G)$, where $\chi(G)$ denotes the chromatic number of G. The algorithm proceeds recursively on each sub-graph induced by the set of vertices $A_i \setminus A_{i-1}$ (where $A_0=\emptyset$) and returns the union of all the layerings that result from these recursive calls (see Algorithm 1). The procedure *GRAPH* (not shown) returns the sub-graph of its first parameter, G, induced by the vertices-set passed as its second parameter. Nicolso et al. present a slightly different but equivalent version of MSCS and prove that it is a 2-approximation to the chromatic sum problem [4]. We prove in Section VII that MSCS is also a 2-approximation for the MLIC problem.

Algorithms MSIS and MSCS do not take interval weights into consideration. If the budget of available extra bits is significantly smaller than that required for an optimal solution of MLIC, then the weight of the layering obtained may be far from optimal regardless of how bits are partitioned in the bits allocation stage (explained shortly). This motivates the following two heuristic algorithms.

c) Maximum Weight Independent Sets (MWIS): same as MSIS, except that we iteratively find a maximum *weighted* independent set.

d) Maximum Weight Colorable Sets (MWCS): same as MSCS, except that, instead of finding maximum size k-colorable sets (Line 5 in the pseudo-code of Algorithm 1), MWCS finds *maximum weight* k-colorable sets. Finding a maximum-weight k-colorable set of an interval graphs can be done in polynomial time by transforming the graph into a directed acyclic network graph and solving the *minimum cost flow* problem [15]. The minimum cost flow problem, in

Algorithm 2 Bit Auction

```
1: int[] procedure BIT-AUCTION(layering C, int b)
2:     assigned[|C|], p[|C|]: arrays of integers, initially all 0
3:     for all L_i ∈ C do
4:         ⟨L[i,1], L[i,2], ··· , L[i,|L_i|]⟩ := Sort L_i's intervals in
5:                                              decreasing weight order
6:     end for
7:     while b > 0 do
8:         for all L_i ∈ C do
9:             p[i] := max_{k∈[1,b]} (1/k) Σ_{j=2^{assigned[i]}}^{2^{assigned[i]+k}-1} w(L[i,j])
10:        end for
11:        m := arg max_i p[i]
12:        assigned[m] = assigned[m] + 1
13:        b := b − 1
14:    end while
15:    return assigned
16: end procedure
```

turn, can be solved in polynomial time by using the widely-used paradigms of *augmentation paths* and *negative cycle cancelations* [16].

C. The Bits Allocation Stage

In this stage we assign extra bits to layers and determine which intervals are to be encoded. This process is done by the *Bit-Auction* algorithm whose pseudo-code appears as Algorithm 2. The $assigned$ array stores the numbers of bits assigned to layers: $assigned[i]$ is the number of bits assigned to the i'th layer of C. The algorithm works in b iterations, where b is the number of available extra bits. Each iteration may be viewed as an auction, in which layers compete for the next available bit. The intervals in each layer are sorted in decreasing order of their weight (where ties are broken arbitrarily) (line 4). If a layer L_i has already been assigned $assigned[i]$ bits, then assigning it additional k bits allows us to encode L_i's next $2^{assigned[i]+k} - 2^{assigned[i]}$ intervals. Assume that all intervals are sorted in decreasing weight order within each layer and let $L[i][j]$ denote the interval of L_i with the j'th largest weight. Then the *per bit* decrease in redundancy gained by allocating the next k bits to layer L_i is: $\frac{1}{k}\sum_{j=2^{assigned[i]}}^{2^{assigned[i]+k}-1} w(L[i][j])$. We compute the above quantity for each layer L_i (line 9), and assign the next bit to a layer for which this quantity is maximal (line 11-13). Note that the first bit allocated to each layer accommodates only *a single* interval, since value 0 is interpreted as 'not in layer'.

Also note that, given a layering $\mathcal{C}$ that contains n intervals and a constant number of extra-bits b, an optimal bits allocation can be computed polynomially by an exhaustive search on all possible allocations. However, this algorithm has a prohibitive time-complexity of $O(n \cdot \log n + n \cdot |\mathcal{C}|^b)$. The Bit-Auction algorithm, on the other hand, is fast (our implementation has time-complexity $O(n \cdot \log n + n \cdot b^2)$) and achieves excellent results in practice. We do not know if it is optimal.

D. The Encoding Stage

The essence of the encoding stage is implemented by the *Encode* procedure that assigns codes to the intervals that were selected for encoding in the bits allocation stage. These codes are then used to determine the range-field in the TCAM entries and the value of the extra bits in the search keys.

The code for each interval is determined by the layer it belongs to (by definition, each interval belongs to a single layer). As a result of the bits allocation stage, each layer L_i is assigned a range of extra bit indices (k_{i_1}, k_{i_2}), of length $assigned[i]$, that is dedicated for encoding interval of that layer. The intervals of layer L_i are therefore encoded within these boundaries in a decreasing order of their weight. For example, if $assigned[i] = 3$, the interval with largest weight is assigned value 001, the interval with the second largest weight is assigned value 010, and so on. All bits outside these boundaries (that is, with index at most k_{i_1} or at least k_{i_2}) are set to $\star$. The pseudo code of this stage is given by the *Encode* procedure (see Algorithm 3) that uses the following notations: $\text{bin}(j)$ denotes the binary representation of the integer j and $\star^k$ denotes a vector of k 'don't-care' bits. The concatenation of strings A and B is denoted by $A \cdot B$. The procedure returns as output the *code* array (line 13). Entry $code[i][j]$ stores the code assigned to interval j of layer i, where, as before, intervals are sorted within each layer in decreasing order of weight. The output of the *Encode* procedure is used to set the *code* of the RANGES.

The number of extra-bits b is limited, hence not all ranges may be assigned LIC codes. Ranges that are not assigned a LIC code can be represented using any prior art database-independent encoding scheme, such as prefix expansion or Gray code [12]. If a rule contains such a range, then the extra bits corresponding to that field are set to 'don't-care's and the range-field is encoded by the fall-back technique. If a rule's range-field *was* assigned a LIC code then the field value is set to 'don't-care's.

The second task of the encoding stage is to determine the values of search keys' extra bits: For each search key value x, the extra bits are determined by performing a bitwise OR of the vector 0^b with the codes of all the ranges that contain x. These codes are retrieved from the output of the *Encode* procedure. We assume here that an OR of a proper bit value (0 or 1) with $\star$ returns the proper bit value. Note that, since ranges of different layers use different extra bits, and since the ranges within the same layer are mutually disjoint (that is, at most one range in each layer can contain x), this bit-wise OR operation effectively concatenates the codes of all the ranges intersected by the entry; bits corresponding to layers in which no range is intersected by the entry are set to 0 by this operation. Also, it is important to notice that the encoding stage modifies only the extra bit values; the rest of the search-key's bits remain unchanged.

Algorithm 3 Encode

```
1: int[][] procedure ENCODE(layering C, int[] assigned)
2:     for all L_i ∈ C do
3:         k_1 := Σ_{m=1}^{i-1} assigned[m]
4:         k_2 := Σ_{m=i+1}^{|C|} assigned[m]
5:         for all r_j ∈ L_i do
6:             if j ≤ 2^{assigned[i]} − 1 then
7:                 code[i][j] := *^{k_1} · bin(j) · *^{k_2}
8:             else
9:                 code[i][j] := ⊥
10:            end if
11:        end for
12:    end for
13:    return code
14: end procedure
```

IV. Implementation Consideration

In this section, we explain how the basic LIC scheme, described thus far, is modified to support multiple range fields. We also provide more details about the architecture of the LIC scheme.

A. Supporting Multiple Range Fields

In the previous sections, for presentation simplicity, we have mostly ignored the fact that real-life classification-database ranges may occur in more than a single field. In the following we describe how our scheme supports the two IP header range-fields: the source-port (s-port) and destination-port (d-port) fields. We note that our scheme can be easily extended to support a larger number of range-fields.

Our algorithm maintains two separate range sets: one for the ranges that occur in the s-port field and another for the ranges that occur in the d-port field. The layering stage described in Section III-B is performed separately for each of these sets. Nevertheless, these two sets are not independent since the existence of multiple range-fields influences the calculation of range weights. As an example, consider a range r that occurs in the s-port field and another range s that occurs in the d-port field. If r and s occur in the same rule, then the contribution of r (s) to the total database redundancy depends on the basic redundancy of s (r) and whether or not s (r) is assigned a LIC code.

To accommodate that, the weight attributed to r on account of a rule where it co-occurs with s is computed as $b(r) \cdot b(s) - b(s)$, where $b(x)$ denotes the basic redundancy of x. This is exactly the number of redundant entries that can be saved in the representation of the rule if r is assigned a LIC code assuming s is not assigned a LIC code.

The bits allocation stage is done on the union of layers that were obtained for both range sets, with the following modification applied to the algorithm described in Section III-C: At the end of each algorithm iteration, the weight of a range r that co-occurs with a range s is decreased if s was assigned a LIC code in the course of the iteration. Finally, the encoding stage (Section III-D) is performed separately for each of the range sets.

B. LIC Scheme Architecture

In all database-dependent range-encoding schemes, incoming headers have to be preprocessed to create a search key. For the LIC scheme, the values of all fields (i.e. the source/destination IP, source/destination ports and protocol fields) are simply copied from the header to the corresponding fields of the search key. Preprocessing is required only for determining the values of the extra bits. Our scheme employs pre-computed tables that map the ranges to the extra bits for fast preprocessing. We now describe the preprocessing process in more detail.

Two direct-access lookup tables are used, called SOURCE-KEYS and DEST-KEYS, for storing the extra bits of the search key corresponding to the specific value of s-port and d-port fields, respectively (see Section III-D for details about the encoding procedure of a specific search key). The SOURCE-KEYS and DEST-KEYS tables are accessed by using the s-port and d-port field values, respectively, as indices. The extracted values are then concatenated to construct the extra-bits part of the search key.

These two tables are small and can thus be stored in fast memory. Each of the port fields is 16-bit wide, and hence the two tables have a total of $128K$ entries. Assuming the typical $b = 36$ and SRAM word-size of 32 bits, the two tables occupy at most $384K$ SRAM words altogether. [3]

V. Hot Updates Support

In general, the rules of a classification database change over time. To maintain space efficiency, every practical database-dependent range-encoding scheme must eventually change its encoding in accordance with these changes. First and foremost, it is crucial to guarantee that, while TCAM entries are being updated, one can still use the device to classify incoming packets. In that case, we say that the scheme supports *hot updates*.

We next present a general scheme for hot updates that can be combined with any database-dependent range-encoding scheme. Our approach uses a single extra bit for every rule, called the *phase bit*.

We assume the existence of two procedures, called ADD_ENTRY and DELETE_ENTRY, that respectively add and delete a single TCAM entry. Efficient implementations of such procedures were presented in prior art (see, e.g., [17]). Our algorithm alternates between 0-phases and 1-phases, depending on the value of a global *phase* variable.

Whenever enough updates have occurred to trigger an encoding re-computation, the following two steps are performed:

[3] Direct-access tables become too big for future protocols such as IPv6, where the s-port/d-port width occupy more bits. For such protocols, the SOURCE-KEYS and DEST-KEYS tables can be implemented on a TCAM device similarly to the scheme proposed by [13].

Algorithm	Expansion	Redundancy
Prefix Expansion [1]	2.68	6.53
Region Partitioning [6]	1.64	2.51
hybrid DIRPE [8]	1.2	-
hybrid SRGE [11]	1.03	1.2
DRES [13]	1.025	0.09
LIC/MSIS and Prefix Expansion	1.0076	0.029
LIC/MSIS and SRGE	1.0061	0.024
LIC/MSCS and Prefix Expansion	1.0088	0.034
LIC/MSCS and SRGE	1.0075	0.029
LIC/MWIS and Prefix Expansion	1.0089	0.034
LIC/MWIS and SRGE	1.0074	0.029
LIC/MWCS and Prefix Expansion	1.0088	0.034
LIC/MWCS and SRGE	1.0074	0.029

TABLE I
EXPANSION AND REDUNDANCY FACTORS, USING 36 EXTRA BITS, FOR DIFFERENT RANGE ENCODING ALGORITHMS.

Layering Algorithm	Number of Required Bits
LIU	235
LIC/MWCS	96
LIC/MWIS	94
LIC/MSCS	86
LIC/MSIS	85
Optimal MLIC Solution	85

TABLE II
THE NUMBER OF BITS REQUIRED TO ENCODE ALL THE RANGES THAT OCCUR IN OUR DATABASE AS A FUNCTION OF THE ENCODING SCHEME EMPLOYED.

1) We employ the ADD_ENTRY procedure for adding all the rules that we want to encode using the new encoding. The phase bits of these rules are set to the value of the new phase (i.e, new-phase = 1 - phase). Note that, right after adding these entries, each range-rule might be represented *twice* in the database, since the range may be encoded in both the current database and the database of the previous phase. Non-range rules, however, are never duplicated.
2) After performing step 1, newly-encoded rules are still ineffective, since the phase-bit's value in all search keys is set to the previous phase. The algorithm therefore proceeds to update the search keys. Since all relevant range-rules are currently duplicated in the TCAM device, search-key updates may proceed lazily. After the updates complete, the phase is switched to the new-phase. Now all the obsolete rules can be deleted from the TCAM by invoking the DELETE_ENTRY procedure. Since these rules are never matched in the course of the new phase, deletions may proceed lazily as well.

VI. EXPERIMENTAL RESULTS

We evaluate the performance of the LIC scheme on a real-life database, which is a collection of 120 separate rule files originating from various applications (such as firewalls, acl-routers and intrusion prevention systems) collected over the year 2004. This is the same database that was used by [2], [8], [11].

Our database consists of a total of approximately $223,000$ rules that contain 280 unique ranges. Approximately 28% of the rules contain range fields and about half of these include the range $[1024, 2^{16} - 1]$. When applied to this database, prefix expansion results in an expansion factor of 2.68 and redundancy factor of 6.53 (see Table I).

Table I shows the expansion and redundancy factors obtained by prior art and by our algorithms when 36 extra bits are available, which is the common case in contemporary IPv4 TCAM classifiers. When all extra bits are exhausted, our LIC scheme can use any independent encoding algorithm as a fall-back scheme. We evaluated the LIC scheme using two fall-back schemes: (binary) prefix expansion and SRGE [11]. The SRGE algorithm is a database-independent encoding scheme that encodes ranges using binary-reflected Gray code.

It is clear from the data of Table I that all LIC algorithms, regardless of the fall-back scheme they use, obtain smaller expansion and redundancy factors as compared to prior art algorithms. Specifically, the redundancy factor obtained by all LIC algorithms is smaller by more than 62% as compared to DRES. It can also be seen that using SRGE as a fall-back scheme reduces redundancy as compared to using (binary) prefix expansion. SRGE saves 17.2% in redundancy when used in conjunction with LIC/MSIS and 14.7% when used in conjunction with LIC/MSCS, LIC/MWIS or LIC/MWCS as compared to a binary prefix expansion fall-back scheme.

Table II shows the number of bits required to encode all the ranges that occur in our database when using different encoding schemes. The naïve Liu algorithm [6] requires 235 bits, since a single bit must be used for every range with expansion larger than 1. All variants of the LIC scheme achieve a very significant decrease in the number of required extra bits as compared to the Liu algorithm. Surprisingly, LIC with the simple MSIS layering algorithm achieves the best result - only 85 bits. This is, in fact, the optimal solution of the MLIC problem on our database. Note that the DRES algorithm is not shown in Table II since, when the number of extra bits is not restricted, it degenerates to the Liu algorithm.

It can be observed from the data of Table II that when the number of extra bits is not restricted (i.e. when we solve the MLIC problem rather than the BMLIC problem), the un-weighted layering algorithms (that is, MSIS and MSCS) are superior to the weighted layering algorithms (MWIS and MWCS) for the following reason: If all ranges can be encoded, then there is no use in giving higher-weight ranges precedence when constructing layers. When solving MLIC, the layering algorithm should *ignore* weights; taking weights into consideration will, in general, increase the number of bits eventually used.

Figure 1 compares the redundancy factor of different range encoding algorithms. Since the curves of LIC/MSIS and LIC/MWCS are almost identical to those of LIC/MSCS and

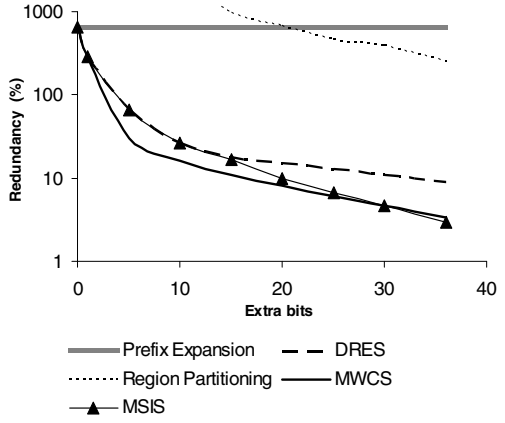

Fig. 1. Redundancy factor as a function of the number of extra bits for different encoding schemes

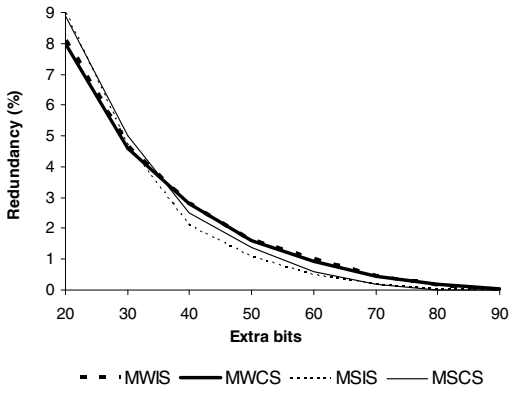

Fig. 2. Redundancy factor as a function of the number of extra bits for different layering algorithms.

LIC/MWIS, we only show the former curves in Figure 1. Regardless of the number of extra-bits available, our scheme outperforms all prior-art dependent range encoding algorithms. LIC, using either one of the four layering algorithms we analyze, reduces the redundancy factor by 50%-70% (depending on the number of available extra bits) as compared with DRES [13], which is the best prior art range encoding algorithm. Specifically, when 36 extra bits are available (as is typical for IPv4 TCAM-based classification databases), our scheme (using any of the 4 layering algorithms we described) reduces the redundancy factor as compared to DRES by either 62.2% or 67.8%, depending on whether the fall-back scheme used is prefix expansion or Gray coding, respectively. Note that the region partitioning algorithm [6] improves over prefix expansion only when the number of extra bits approaches 20 and even then achieves only a minor improvement of the redundancy factor.

Figure 2 shows the redundancy factor obtained by LIC when used with the different layering algorithms we analyze. It can be seen that the weighted layering algorithms (MWIS and MWCS) are superior when the number of extra bits is 34 or less but the un-weighted algorithms fare better when the number of extra bits exceeds 34. As mentioned before, the benefit gained from giving precedence to high-weight ranges in the layering process decreases when the number of extra bits increases.

VII. Theoretical Results

We start by proving that the MLIC problem, formally defined in Section III-B, is NP-hard. MLIC is similar to the chromatic sum problem [5], [18]. Marx [19] provided a short and elegant proof that chromatic sum is NP-hard on interval graphs. We employ similar ideas to prove NP-hardness of MLIC.

A graph G is a *circular arc graph* if it is the intersection graph of arcs on a circle. In other words, the vertices of G have a one-to-one correspondence with a set of arcs on a circle so that two vertices in G are adjacent in G if and only if their corresponding arcs intersect each other.

Given a circular graph G and integer k, the *circular graph coloring problem* is to decide whether G can be colored by at most k colors. We prove the NP-hardness of MLIC by a reduction from this problem, whose NP-hardness was established in [3].

Theorem 1: The Minimum space Layered Interval-Code (MLIC) problem is NP-hard.

Proof: Let G be a circular arc graph with n arcs and let k be an integer. The circular graph coloring problem it to decided whether G can be colored by using k colors. A circular arc representation of G can be found in polynomial time [20]. It can be assumed that the arcs are open: two arcs that share only an endpoint are assumed not to intersect each other. Let x be an arbitrary point on the circle that is not the endpoint of any of the arcs. If x is contained in more than k arcs, then a polynomial-time test can determine that G cannot be colored with k colors. Hence, it can be assumed that x is contained in exactly k arcs, since if x is contained only in the arcs $a_1, \cdots, a_{k'}$, for $k' < k$, then we can add $k - k'$ sufficiently small arcs that intersect only $a_1, \cdots, a_{k'}$. Clearly, this cannot increase the chromatic number of G beyond k.

Let $a_1, \cdots, a_k$ be the arcs that contain x. Split each arc a_i into two parts l_i and r_i at point x. Let x be the clockwise (respectively counter-clockwise) endpoint of l_i (respectively r_i) and let G' denote the resulting graph. G' is an interval graph, since x is not contained in any interval of G'. It follows that G' has an interval representation where the left endpoint of each interval l_i is 0, the right endpoint of each r_i is some positive M, and no interval extends to the left of 0 or to the right of M. We can modify the left endpoint of l_i to $-i+1$ and the right endpoint of r_i to $M+i-1$ without changing G'.

Let $R = k^7(n+k+1)$ and for every $i \in [k]$ define $R_i = (k-$

$i+1)R$. Let S_x be a set of $R/2$ equal-sized and contiguously-extending intervals, each of length $2/R$, such that $\inf\{I \in S_x\} = x$ and $\sup\{I \in S_x\} = x+1$; that is, S_x consists of $R/2$ intervals in $(x, x+1)$.

We proceed with constructing another graph denoted G''. G'' is constructed by adding the following sets of intervals into graph G': For each $i \in \{-k, \ldots, -1\}$ we add $-i$ copies of the set S_i; similarly, for each $j \in \{M, \ldots, M+k-1\}$ we add $j - M + 1$ copies of the set S_j. We denote these set of added intervals SL and SR respectively.

The hardness result is derived from the following lemma, whose prove is omitted for lack of space.

Lemma 2: A circular arc graph G can be colored with k colors if and only if the corresponding graph G'' has a coloring with LIC-code size less than $B = 1/k^5 + \sum_{i=1}^{k} \log_2 R_i$.

■

The hardness of the BMLIC problem follows by reduction from the MLIC problem:

Theorem 3: The Budgeted Minimum Layered Interval-Code problem is NP-hard.

Proof: Let $G = \langle V, E \rangle$ be an instance of the MLIC problem. We show that the minimum LIC-code of G can be found in polynomial time by an oracle machine with an oracle for the BMLIC problem.

This is done by iteratively running the BMLIC oracle with input G and increasing b by 1 in every iteration (starting with $b = 1$). The algorithm stops when the oracle first returns G (that is, the graph G can be colored completely with an interval code size exactly b). The solution for the MLIC problem is therefore the coloring $\mathcal{C}$ returned by the oracle (and the optimal MLIC value is b).

Since a coloring $\mathcal{C}'$ of G that colors each vertex $v \in V$ with a unique color is always feasible and satisfies $L(G, \mathcal{C}') = |V|$, the optimal MLIC value is always bounded by $|V|$. Therefore, the algorithm described above runs in polynomial time and the claim follows. ■

We next show that the MSCS algorithm presented in Section III-B (see Algorithm 1 for the pseudo-code) is a polynomial 2-approximation algorithm for the MLIC problem. The algorithm and the proof follow closely the 2-approximation algorithm for *sum coloring on interval graph* (and its proof) presented in [4]. The proof is omitted for lack of space.

Theorem 4: Algorithm MSCS is a polynomial 2-approximation algorithm for the MLIC problem.

VIII. Discussion

This paper presents the *Layered Interval Code* (LIC) range-encoding scheme for constructing space-efficient TCAM representations of range rules. The gist of our scheme is the use of graph-coloring algorithms to partition the ranges between multiple *layers*. Each of these layers consists of mutually disjoint ranges that are encoded independently by using the extra-bits that are typically available in TCAM-based classifier entries. We present several LIC construction algorithms that are based on approximation algorithms for specific variants of interval-graph coloring. We also present a novel hot updates algorithm (used by our LIC scheme) that can be used with any database-dependent encoding scheme. We evaluate these algorithms by performing extensive comparative analysis on real-life classification databases. Our analysis establishes that our algorithms reduce the number of redundant TCAM entries caused by range rules by more than 60% as compared with best range-encoding prior art.

Acknowledgments: The authors are deeply indebted to Cisco Systems, Will Eatherton, and David Taylor for kindly providing us the classification database we used for this work. We would also like to thank Yehuda Afek, Karthik Lakshminarayanan, Anand Rangarajan , and Srinivasan Venkatachary for assisting us in the process of obtaining access to the database.

References

[1] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," in *ACM SIGCOMM*, Sep. 1998, pp. 191–202.
[2] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Comput. Surv.*, vol. 37, no. 3, pp. 238–275, 2005.
[3] M. R. Garey, D. S. Johnson, G. L. Miller, and C. H. Papadimitriou, "The complexity of coloring circular arcs and chords," *SIAM Journal on Algebraic and Discrete Methods*, vol. 2, no. 1, pp. 216–227, 1980.
[4] S. Nicoloso, M. Sarrafzadeh, and X. Song, "On the sum coloring problem on interval graph," *Algorithmica*, vol. 23, pp. 109–126, 1999.
[5] E. Kubicka, "The chromatic sum of a graph," Ph.D. dissertation, Western Michigan University, 1989.
[6] H. Liu, "Efficient mapping of range classifier into ternary-cam," in *Hot Interconnects*, 2002.
[7] J. van Lunteren and T. Engbersen, "Fast and scalable packet classification," *JSAC*, 2003.
[8] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for advanced packet classification with ternary CAMs," in *ACM SIGCOMM*, 2005.
[9] F. Yu and R. H. Katz, "Efficient multi-match packet classification with tcam," in *HOTI*, 2004.
[10] E. Spitznagel, D. Taylor, and J. Turner, "Packet classification using extended TCAMs," in *ICNP*, 2003.
[11] A. Bremler-Barr and D. Hendler, "Space-efficient tcam-based classification using gray coding," in *IEEE INFOCOM*, 2007, pp. 1388–1396.
[12] F. Gray, "Pulse code communication," 1953, united States Patent Number 2632058.
[13] H. Che, Z. Wang, K. Zheng, and B. Liu, "Dres: Dynamic range encoding scheme for tcam coprocessors," *IEEE Transaction on Computers*, vol. 57, no. 6, 2008.
[14] A. Bar-Noy, M. Bellare, M. M. Halldorsson, H. Shachnai, and T. Tamir, "On chromatic sums and distributed resource allocation," *Information and Computation*, vol. 140, no. 2, pp. 183–202, 1998. [Online]. Available: citeseer.ist.psu.edu/bar-noy98chromatic.html
[15] M. C. Carlisle and E. L. Lloyd, "On the k-coloring of intervals," *Discrete Applied Mathematics*, vol. 59, no. 3, pp. 225–235, 1995.
[16] R. E. Tarjan, *Data structures and network algorithms*. Society for Industrial and Applied Mathematics, 1983.
[17] D. Shah and P. Gupta, "Fast updating algorithms for tcams," *IEEE Micro*, vol. 21, no. 1, pp. 36–47, 2001.
[18] K. J. Supowit, "Finding a maximum planar subset of nets in a channel," *IEEE Transactions on Computer-Aided Design*, vol. 6, no. 1, pp. 93–94, 1987.
[19] D. Marx, "A short proof of the NP-completeness of minimum sum interval coloring," *Operations Research Letters*, vol. 33, no. 4, pp. 382–384, 2005.
[20] A. Tucker, "An efficient test for circular-arc graphs," *SIAM Journal on Computing*, vol. 9, no. 1, pp. 1–24, 1980.