Encoding Short Ranges in TCAM Without Expansion: Efficient Algorithm and Applications

Anat Bremler-Barr, *Member, IEEE*, Yotam Harchol, *Member, IEEE*, David Hay, *Member, IEEE*, and Yacov Hel-Or, *Member, IEEE*

***Abstract*— We present range encoding with no expansion (RENÉ)— a novel encoding scheme for short ranges on *Ternary content addressable memory* (TCAM), which, unlike previous solutions, does not impose *row expansion*, and uses bits proportionally to the maximal range length. We provide theoretical analysis to show that our encoding is the closest to the lower bound of number of bits used. In addition, we show several applications of our technique in the field of packet classification, and also, how the same technique could be used to efficiently solve other hard problems, such as the *nearest-neighbor search* problem and its variants. We show that using TCAM, one could solve such problems in much higher rates than previously suggested solutions, and outperform known lower bounds in traditional memory models. We show by experiments that the translation process of RENÉ on switch hardware induces only a negligible 2.5% latency overhead. Our nearest neighbor implementation on a TCAM device provides search rates that are up to four orders of magnitude higher than previous best prior-art solutions.**

***Index Terms*— Computer networks, switching systems, information retrieval, search methods, nearest neighbor search.**

I. Introduction

TERNARY content addressable memories (TCAMs) have become highly popular in networking equipment and network processing units. TCAMs are used for high-speed IP lookup and packet classification in switches and routers [1], [2]. Software defined networking (SDN) schemes such as OpenFlow [3] rely on TCAM as the main hardware for their data path. TCAM was also suggested to be used for other computationally intensive tasks such as pattern matching [4], [5], heavy-hitters detection [6], and similarity search in databases [7].

TCAM is an associative memory module. It is composed of an array of ternary words, each consisting of ternary digits, namely: 0, 1, or *. The '*' digits serve as 'wild cards' that can be matched with either '0' or '1'. Given a query word, TCAM returns the first location in the memory array that matches the query. This process is illustrated in Figure 1.

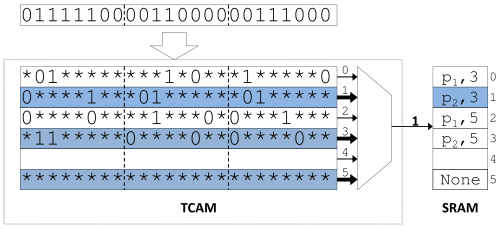

Fig. 1. Diagram of the TCAM lookup process. The query is compared to all entries in parallel and the index of the first matching entry is used to find the result.

Multi-field packet classification is becoming more and more important in modern network architectures, such as SDN and *network function virtualization* (NFV) [8]. Specifically, recently suggested SDN frameworks perform more network functionalities on switches, such as load balancing [9], DDoS prevention [10], and quality of service (QoS) [11]. The initiative for NFV suggests to implement higher level tasks such as deep packet inspection and caching as virtual software services, and make traffic flow through them using smart classification rules. All such frameworks heavily rely on multi-field packet classification. Many of these fields are better expressed as ranges.

While TCAMs become more and more popular, it is still a hard problem to efficiently represent range rules on such memories. Over the last decade there has been an intense line of research on range encoding on TCAM [12]–[24]. Aside from propositions to rearchitect TCAM devices to natively support range rules [13], these solutions can roughly be classified as either *database-independent* or *database-dependent* encoding schemes. Database-independent schemes encode all possible ranges using the same technique, thus allowing fast *hot updates* [12], [14], [15]. However, these schemes use exponential TCAM row expansion, where a row is expanded into several rows, exponentially to the number of range fields in it.

Database-dependent schemes trades the hot updates flexibility for more compact codes [17], [20], but usually performs well only when the number of encoded ranges is small, as the produced code is proportional to the number of ranges in database. Database-dependent schemes do not scale for large number of ranges, as we show in Section IV-A. Therefore, this paper focuses on a database-independent approach.

In this paper we present a database-independent range encoding scheme, called RENÉ - *Range Encoding with No Expansion* - that *eliminates row expansion completely* when ranges are *short enough*. The code produced by RENÉ is *proportional to the maximal range length*, not to the number

Manuscript received June 14, 2017; revised September 26, 2017; accepted January 19, 2018; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y. Ganjali. Date of publication February 5, 2018; date of current version April 16, 2018. This work was supported in part by the European Research Council under the European Union's Seventh Framework Programme FP7/2007-2013/ERC under Grant 259085 and in part by the Israeli Centers of Research Excellence Program under Center 4/11. Partial and preliminary versions of this paper appeared in ACM DaMoN 2015 and ACM SPAA 2016. *(Corresponding author: Yotam Harchol.)*

A. Bremler-Barr and Y. Hel-Or are with the Interdisciplinary Center Herzliya, School of Computer Science, Herzliya 46150, Israel.

Y. Harchol is with the Electrical Engineering and Computer Science Department, University of California at Berkeley, Berkeley, CA 94720 USA (e-mail: yotam@eecs.berkeley.edu).

D. Hay is with the School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem 91904, Israel.

Digital Object Identifier 10.1109/TNET.2018.2797690

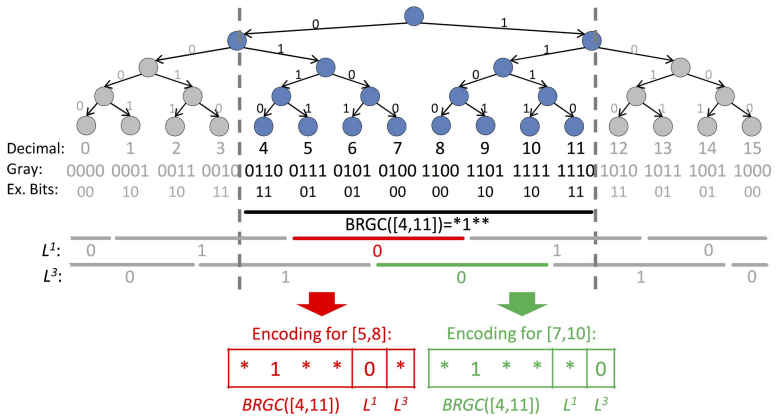

Fig. 2. Toy example: A binary-reflected Gray code (BRGC) encoding tree and the encoding of ranges $[5-8]$ and $[7,10]$ using our scheme. In this example, maximal range length is 4. Ranges are divided into layers of non-overlapping ranges. Two layers contain only ranges that can be encoded using Gray code and therefore are not shown. Containing range $[4,11]$ is encoded based on the BRGC values, which forms the first left four bits. Extra bits correspond to layers: Fifth bit to layer L^1 and sixth bit to layer L^3. If a range belongs to a layer, then the value of the bit corresponding to that layer is the binary value of the range for this layer. Otherwise, that bit is set to '*'. The total number of bits is proportional to the maximal encoded length and is independent of the number of encoded ranges.

of ranges, as in database-dependent schemes. In many cases, as we show in this paper, ranges are limited in length. For example, it was shown in [15] that in real-life packet classification tables more than 60% of the TCP port ranges are short. Moreover, packet classification also uses other range fields, where all ranges are short (such as IP ToS or TTL). On some fields one may apply quantization and categorization to reduce the length of ranges without hurting classification accuracy (e.g. packet length). Nonetheless, RENÉ can be combined with other approaches to represent a wider spectrum of ranges if necessary.

In addition to packet-classification, where TCAM has already been selected as de-facto industry standard, we propose in this paper using a TCAM as a co-processor to CPU in order to solve hard problems from other domains in computer science. Specifically, we show how an encoding scheme such as RENÉ, which requires no row expansion, can be used to practically and efficiently solve the *nearest neighbor search* problem and its variants, removing the infamous curse of dimensionality from them.

Multidimensional nearest neighbor search (NN) lies at the core of many computer science applications. Given a database of objects and a query, we wish to find the object in the database most similar to the query object. Commonly, the objects are mapped to points in high-dimensional metric space. In this context, given a query point $q \in \mathbb{R}^d$ and a set of points $S = \{p_i\}_{i=1}^n$, $p_i \in \mathbb{R}^d$, the goal is to find a point $p \in S$ most similar to the query point q under some distance metric. In addition to the exact NN, many variants of this problem exist, including k-nearest neighbor, approximate nearest neighbor, fixed-radius near neighbors, and more. The NN and its variants are utilized in a wide range of applications, such as spatial search, object recognition, image matching, image segmentation, classification and detection, to name a few [25]–[29].

When the dimensionality of the points is small, many solutions were proven to be very effective. These include mainly space partitioning techniques [30]–[32]. However, when the number of the data points is large (in the order of tens of thousands or higher) and the dimensionality is high (in the order of tens or hundreds), the exact solutions break down and produce exponential time complexity[1] [33], [34]. This problem is widely known as the *curse of dimensionality*.

To overcome the curse of dimensionality, *approximated nearest neighbor* (ANN) solutions are commonly used. In particular, a c-ANN is a solution where the distance of the retrieved point from q is at most c times the true distance from the nearest point. For the ANN problem, probabilistic dimensionality reduction such as *locality sensitive hashing* (LSH) [33] was proven to be useful, with query time sub-linear in n but linear in d. For very-high dimensional space this may still pose a problem [35]. Note also that the solution provided by LSH is correct only with high probability.

To our knowledge, we are the first to present a database-independent encoding scheme for short ranges on TCAM with no row expansion. In a nutshell, RENÉ divides all ranges of some length h_{max} into h_{max} layers of disjoint ranges. Using the *binary-reflected Gray code* (BRGC) [36], which was shown to be more expressive for ranges than binary representation [15], it focuses on a specific area where the encoded range is. Using additional bits, it exactly points to the encoded range inside the area in focus, where a single additional bit represents the location of the range inside the layer it belongs to. A toy example is shown in Figure 2.

Using a general conjunction operator we present next, we are able to encode all ranges with length up to h_{max}. The total length of RENÉ's code for a w-bits field, when encoding ranges of *up to* length h_{max}, is $w - \log_2(h_{max}) + h_{max} - 1$. This means that RENÉ is feasible on contemporary TCAMs for ranges up to length of 512, depending on the available space on TCAM and number of range fields. We also present a theoretical analysis and show that at least $max(h_{max} - 1, w)$ bits are required to encode short ranges of up to length h_{max} in a w-bits field. RENÉ is closer to this lower bound than any previously-suggested technique.

[1]Exact brute-force search works in time that is linear to n and d, but is very slow for high n and d. Space partitioning techniques are exponential in d.

We show several applications for RENÉ in the area of packet classification, along with an implementation of such an application on a powerful OpenFlow switch. We also show by experiments that the penalty in latency for translating values using RENÉ is negligible.

We evaluate and experiment our nearest-neighbor algorithms on a real TCAM device and achieve search rates up to four orders of magnitude higher than previous best prior-art solutions [7], [33], [37].

II. Background

A. Ternary Content Addressable Memory (TCAM)

Contemporary TCAM devices operate at very high rates, between hundreds of millions to more than one billion queries per second [38], [39]. These devices have about 20-40 megabits of memory that can be configured to accommodate entries of up to about 640-bits wide (the wider the entry, the fewer entries can be stored on the chip).

The downsides of TCAM are that it is power hungry, tends to generate high heat (thus requiring extra cooling), and relatively expensive, compared to a standard DRAM chip. A high-density TCAM consumes 12 to 15 W per chip when the entire memory is used [2]. However, compared to compute units and coprocessors such as CPU or GPU, TCAMs' power requirement, heating and price are actually lower, and become similar only when connecting multiple TCAMs in parallel, as usually done in high end networking equipment. For example, Intel's E7-4870 CPU consumes 130 W [40], and Nvidia's Tesla K80 GPU consumes up to 300 W [41]. Another downside could be that currently, a TCAM cannot be easily deployed on a standard PC, as they are manufactured for networking equipment.

However, due to their impressive adoption for networking devices, TCAMs are becoming larger, faster, less power hungry and less expensive. We speculate that this trend will continue. Inspired by the adoption of Graphics Processing Units (GPUs) for general purpose parallel computing in recent years, in Section V we also suggest that TCAMs may be useful for other tasks outside the networking field.

B. Range Encoding on TCAM

The problem of range encoding on TCAM has received considerable attention in the context of packet classification.

The traditional technique for range representation [12] is prefix expansion, where a range is represented 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 a w-bit field is $2w-2$ and for an entry with d ranges it is $(2w-2)^d$. Lakshminarayanan *et al.* [14], and Bremler-Barr and Hendler [15] suggest encoding schemes other than binary: In [14], Lakshminarayanan *et al.* propose DIRPE, a hierarchical version of fence encoding. Bremler-Barr and Hendler [15] propose SRGE - an encoding based on binary reflected Gray code [36]. However, these works do not reduce the range expansion to one, or, in the case of [14], it requires an infeasible exponential memory size to do that. SRGE [15] points out that more ranges can be expressed by using Gray code than when using binary representation, but it only reduces the worst case of row expansion to $2w-4$. DIRPE [14] suggests a tradeoff between row expansion and the number of bits required to code the range. For encoding without expansion, it demands the unfeasible number of 2^w-1 bits.

The database-dependent range encoding techniques design the encoding to efficiently encode the ranges that specifically appear in the database. These techniques [16]–[20] use extra bits, in addition to the w bits of the range field. The basic idea [18] is to use the extra bits as a bit map: a single extra bit is assigned to each selected range in order to avoid the need to represent it by prefix expansion.

Several works [16]–[20] deal with the scalability problem of this basic technique, which requires one bit per range. However, all these solutions still require either very long rows, proportionally to the number of encoded ranges, or they trade that for row expansion. Moreover, some of these solutions demand extra logic or extra memory that makes them useless in our case, where the number of ranges is high.

In [42] and [43] it was suggested to use negation rules on TCAM instead of row expansion, when applicable, such that rules may specify the opposite of a range and a corresponding opposite action (e.g. 'deny' instead of 'accept' in ACL). This reduces worst-case expansion factor to w but does not eliminate it, and is only applicable in certain scenarios.

In [44] and [45] it was suggested to use the independence of order between entries [44], or the independence between ingress and egress linecards [45], in order to reduce the number of bits used to represent multi-field forwarding tables. Specifically, this reduces the width of TCAM entries and compacts longer ranges to shorter ranges. It is possible to use our proposed encoding scheme can be used on top of the result tables provided by these works. In such a case, this will allow our scheme to represent longer ranges efficiently. However, our scheme is database independent and using such techniques will force it to be database-dependent.

Other works [21]–[24], [46], [47] improve the overall TCAM memory requirements for classification rules, or split the rules into multiple TCAM chips [48], [49]. However, these works do not focus specifically on range encoding, and can be used on top of most of the range encoding techniques including the one proposed in this paper.

Other works use TCAM for similarity search in databases. Shinde *et al.* [7] encode probabilistic hash functions on TCAM to implement locality-sensitive hashing [33]. Afek *et al.* [50] use TCAM to implement priority queues with a constant time lookup operation and as a by-product, to provide a TCAM-based sorting algorithm with $O(n)$ time.

The limitation of all the methods has inspired a suggestion to change the TCAM hardware [13], to implement range matching directly in hardware. However this solution changes TCAM architecture dramatically, and it does not seem feasible in the near future, since TCAM is a popular memory chip that exists in tens of millions of routers and switches today. Moreover, the solution harms the flexibility of TCAM implementation, where every entry is simply encoded as a string of ternary bits, regardless of the fields type and borders.

Gray code [36] was originally designed for error correction purposes in communication networks. However, as shown

in [15], the reflectivity of this encoding can be used in TCAMs, along with the ternary representation provided by this device, to represent intervals. As ternary gray code cannot be used to represent all possible intervals, it was suggested in [20] to divide intervals into layers, and use binary encoding of those layers to represent all intervals. In this work we rely on both ideas.

C. Nearest-Neighbor Search

The nearest neighbor search problem is formally defined by the following definition for points in a discrete space of dimension d:

Definition 1: Given a set of data points $S = \{p_i\}_{i=1}^n$, $p_i \in \mathbb{Z}^d$ and a query point $q \in \mathbb{Z}^d$, THE NEAREST-NEIGHBOR PROBLEM is to find the point $s^* = \arg\min_{s \in S} D(s, q)$, where $D(s, q)$ is a distance between s and q.

As discussed above, in order to overcome the curse of dimensionality, the accuracy of the solution is sometimes compromised. The c–APPROXIMATE NEAREST-NEIGHBOR PROBLEM (c-ANN) searches for a point $p \in S$ such that $D(p, q) < c \cdot D(s^*, q)$, where s^* is the nearest point to q.

One important generalization of the nearest neighbor problem that can be solved with minor adaptations of our framework is the THE k–NEAREST-NEIGHBOR PROBLEM, which finds a set $S' \subseteq S$ of k points such that for each $p' \in S'$ and $p \in S \setminus S'$, $D(p', q) \leq D(p, q)$. We show these adaptations in Section V-B.

A simpler problem that we will use as a building block in our algorithms only searches for a neighbor close enough to the query point, or discovers that there is no such neighbor at all:

Definition 2: Given a set of points $S = \{p_i\}_{i=1}^n$, $p_i \in \mathbb{Z}^d$, and a query point $q \in \mathbb{Z}^d$, let $d^* = \min_{s \in S} D(s, q)$. The r–NEAR-NEIGHBOR REPORT PROBLEM is to find the point $s' \in S$ such that $D(s', q) \leq r$ if $d^* \leq r$, and to return *false* if $d^* > r$.

Note that under ℓ_∞, a solution for the r–NEAR-NEIGHBOR REPORT PROBLEM is a data point within the d-dimensional cube of edge length $2r$ that is centered in the query point. Thus, our framework for solving c-ANN can be viewed as solving (either in parallel or sequentially) a series of r–NEAR-NEIGHBOR REPORT PROBLEM instances for increasing values of r. As pointed out in [51], this solves the c-APPROXIMATE NEAREST NEIGHBOR PROBLEM, where the approximation ratio is determined by the maximum ratio between consecutive values of r.

One method for solving the NEAR NEIGHBOR REPORT PROBLEM is *bucketing* [51]. The idea behind it is to divide the d-dimensional space into a grid of d-dimensional cells. Given a point q located in some cell, we look for its nearest neighbor, p, in the cell. Then, we search for other points p' in adjacent cells such that $D(p', q) < D(p, q)$. If such a point p' exists, we update p to be p'. We then continue looking for such points in adjacent cells until the distance of the cell's boundary from q is larger than $D(p, q)$. Other, more advanced space-partitioning techniques to solve the NEAR NEIGHBOR REPORT PROBLEM are *kd-Trees* [52] and *Random Projection Trees* [53]. However, all these methods are only useful if the dimension of the search space (d) is low (e.g., around 10 or 20) [33]. An experimental study [34] has in fact shown that such approaches scale poorly with the number of dimensions d, and even when $d > 10$, they may perform worse than a brute-force scan.

Locality sensitive hashing (LSH) [51] is another technique for solving instances of the NEAR NEIGHBOR REPORT PROBLEM; its goal is to be more useful in higher dimensions. The idea of this technique is to find a family of hash functions that map neighboring points to the same hash bucket with high probability, so that if two points are in the same bucket, they are likely to be close-enough neighbors. One could create different LSH solvers for different NEAR NEIGHBOR REPORT PROBLEM instances and thus provide an approximated solution for the nearest-neighbor problem. The size of the hash function family depends on the size of the data set, rather than on the space dimension. LSH was further investigated and later works provided better approximation and running times [33], [54]. Building on the LSH idea, Lv *et al.* [55] proposed reducing the number of hash functions by multiple probing of hash buckets that are likely to contain query results. Another LSH-based approach is *locality sensitive B-trees* (LSB-Trees) [56], which improves the running time and quality of results. In Section V-E we show, however, that for high-quality results, the computation time of LSH can be relatively long, and incomparable to the computation time required by our solution.

A different approach to tackle the *curse of dimensionality* is to use parallel hardware. For example, graphics processing units (GPUs), which are currently fully programmable using CUDA and OpenCL, have hundreds of computing cores, and can help reduce the effect of higher dimensions. Two fast nearest neighbor search implementations were presented in [37], [57], and [58]. These implementations, both written in CUDA, basically perform a multithreaded brute-force scan of the data set using a GPU. A GPU was also used to implement a parallel version of the LSH algorithm [59]. While these approaches leverage the parallelism of GPUs and provide much faster solutions than previous approaches, we show in Section V-E how TCAM can provide an even more time-efficient solution to the nearest-neighbor problem.

To the best of our knowledge, using TCAM for nearest neighbor search has been considered only once, by Shinde *et al.* [7] who proposed the TLSH scheme, where a TCAM device is utilized to implement LSH with a series of TCAM lookup cycles. In this scheme, each database point is mapped to a ternary code, where each ternary digit is generated using a random projection and dividing the projected line into m bins whose assignments alternate between ternary digits $[0, *, 1, *, 0, *, 1, \cdots]$. As the '*' digit can match both '0' and '1', this assignment blurs the boundaries between the 0 and 1 bins such that the ternary hashed representation of nearby points matches with high probability. Our algorithms, however, are deterministic and take a completely different approach, as we will highlight in the rest of the paper.

III. ENCODING SCHEME FOR SHORT RANGES

Our goal is to encode a range up to a certain length h_{max} using a single TCAM entry of as few bits as possible. Such

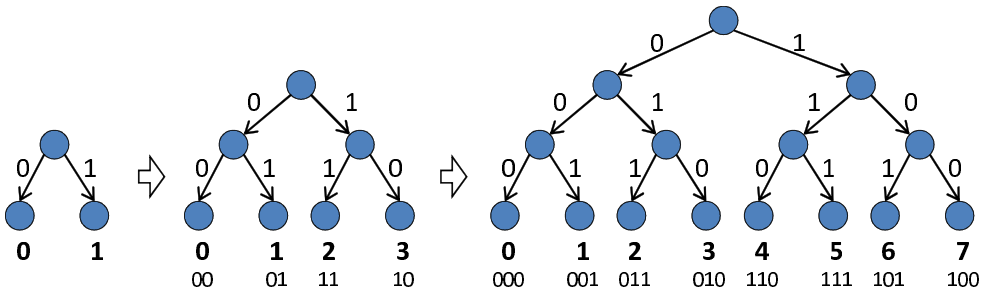

Fig. 3. Encoding trees for Binary Reflected Gray Codes of length 1, 2, and 3 bits.

code will allow encoding classification rules with multiple ranges without row expansion at all.

A. Basic Definitions

We begin with some basic definitions that will be used throughout the rest of this section. First, we define a ternary bit-wise comparison:

Definition 3: Let $a = a_0, \ldots, a_m$ and $b = b_0, \ldots, b_m$ be two ternary words ($a_i, b_i \in \{0, 1, *\}$). a **matches** b, denoted $a \approx b$, if and only if for every $i \in \{0, \ldots, m\}$, either $a_i = b_i$, or a_i is $*$, or b_i is $*$.

RENÉ encodes ranges in discrete spaces. We begin by defining an encoding function **`tcode`** for values and ranges. Let $\mathcal{U} = [0, 2^w) \subset \mathbb{N}_0$ be a range on the natural line. RENÉ's encoding function **`tcode`** encodes either a value $v \in \mathcal{U}$, or a range $R \subseteq \mathcal{U}$. It is important to note that RENÉ treats $\mathcal{U}$ as a cyclic 'wrap-around' space and thus throughout this paper, any range $[x, y)$ refers in fact to $[x, y \bmod 2^w)$.

The result of the encoding function is either a binary word (for exact values) or a ternary word (for ranges), and we expect that a ternary match **`tcode`**$(v) \approx$ **`tcode`**(R) will imply that the value v is inside the range R. This is formally defined as follows:

Definition 4: An encoding function **`tcode`** is **admissible** if for every value $v \in \mathcal{U}$ and every range $R \subseteq \mathcal{U}$, **`tcode`**$(v) \approx$ **`tcode`**(R) if and only if $v \in R$. Furthermore, for any point $v \in \mathcal{U}$, **`tcode`**(v) does not contain '$*$' symbols.

B. Binary-Reflected Gray Code for TCAM

The *binary-reflected Gray code* (BRGC) [36] is a binary encoding of integers in a contiguous range such that the codes of any two consecutive numbers differ by a single bit. A b-bits BRGC is constructed recursively by reflecting a $(b-1)$-bits BRGC.[2]

Definition 5: The $BRGC$ encoding function $BRGC(v, 2^w)$ encodes a point p (where $0 \leqslant p < 2^w$) with w bits. It is defined recursively as follows:

$$BRGC(0,1) = \varepsilon$$
$$BRGC(p, 2^w) = \begin{cases} 0 \cdot BRGC(p, 2^{w/2}) & \text{if } p < 2^{w/2} \\ 1 \cdot BRGC(2^w - p - 1, 2^{w/2}) & \text{otherwise} \end{cases}$$

where ε is the empty word and '$\cdot$' denotes concatenation.

[2]The BRGC of a value x can be directly calculated using the following formula: $x \oplus (x \gg 1)$, where $\oplus$ and $\gg$ are the bitwise operations of XOR and Right Shift, respectively.

For example, $BRGC(4, 8) = 1 \cdot BRGC(3, 4) = 11 \cdot BRGC(0, 2) = 110 \cdot BRGC(0, 1) = 110$. An example for values in $[0, 16)$ is shown in Figure 2.

Figure 3 shows the recursive process of constructing BRGC for $n = \log w = 3$ bits. We begin with $n = 1$, where 0 is encoded as 0 and 1 is encoded as 1. To construct the code for $n = 2$, the code is duplicated and reflected, and an additional leading 0-bit (1-bit) is added to the first (second) part. The result is can be viewed as a tree with two levels, where the encoding of a number (a leaf) is the sequence of binary digits (transitions) that lead to it. This process can continue recursively for any number of bits. The BRGC codeword can also be computed directly using a simple formula

By wildcarding some of the bits of a BRGC codeword we can create a ternary range representation. For example, as can be seen in Figure 2, the ternary word `*1**` matches all values in the range $[4, 11]$. In fact, when looking at this tree representation of the BRGC encoding, we observe that all ranges that exactly contain a full sub-tree, or two adjacent full sub-trees, can be represented using a single *ternary* BRGC codeword (namely, a BRGC codeword where some of the 0-1 bits were replaced by '$*$' symbols).

Before formulating and proving this observation we define the following terms that will be used in the proof:

- *k-prefix* is a ternary word in which the k least significant bits are '$*$' and the rest are either 0 or 1.
- *k-semi-prefix* is a ternary word in which the k least significant bits are '$*$', one additional bit is also '$*$', and the rest are either 0 or 1.

We now formulate and prove the following theorem:

Theorem 1: If all values are BRGC-encoded, then a single ternary BRGC codeword suffices to admissibly encode a range $R = [x, y \bmod 2^w)$ if and only if there exist non-negative integers i, k, for which $x = i \cdot 2^k$ and $y = (i+2) \cdot 2^k$. Specifically, one of the following cases holds:

1) If i is even, the $(k+1)$ least significant bits of the codeword are $*$, and the rest are either 0 or 1. Thus, the ternary codeword is $(k+1)$-prefix.
2) If i is odd, the k least significant bits of the codeword are $*$, one additional bit is $*$, and the rest are either 0 or 1. Thus, the ternary codeword is k-semi-prefix.

Proof: The proof follows by induction on k: For $k = 0$, ranges are $[i, i+2)$. These ranges are simply two adjacent leaves in the BRGC tree representation, and, by definition of Gray code, they differ in a single bit only.

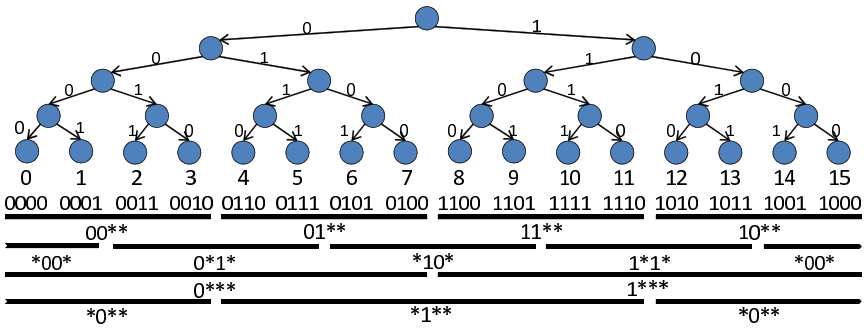

Fig. 4. BRGC encoding tree for points in range $[0, 16)$, and ternary representations of intervals of lengths $4, 8$ that can be represented using this encoding.

If i is even, then there is an even number of leaves before these two, and thus these two leaves are siblings and have a common direct ancestor x with height $k+1=1$. Thus, the $(k+1)$-prefix that is the concatenation of the $\log(w)-1$ bits that represent the path to x with one $*$, represents the range $[i, i+2)$ (case 1 in Theorem 1).

If i is odd, then the two leaves do not have a common direct ancestor. However, they do have some common ancestor up in the higher levels of the tree, let it be at height h. Thus, their representation differ in the h^{th} bit. Since the codewords of i and $i+1$ differ only in one bit, there are no additional bits where they differ. Therefore, the k-semi-prefix in which the h^{th} bit is $*$ and the rest of the bits are as in i and $i+1$ BRGC codewords, represents the range $[i, i+2)$ (case 2 in Theorem 1).

We assume that the lemma is correct for some k, and show that it is correct also for $k+1$: Let $R = [i \cdot 2^{k+1}, (i+2) \cdot 2^{k+1})$ be a range of length 2^{k+2}, for some positive integer i. Let $j_1 = 2i, j_2 = 2i+2$, and let $R_1 = [j_1 \cdot 2^k, (j_1+2) \cdot 2^k), R_2 = [j_2 \cdot 2^k, (j_2+2) \cdot 2^k)$ be two ranges of length 2^{k+1}. Then, $R = R_1 \cup R_2$, and since j_1 and j_2 are even, R_1, R_2 can be represented using $(k+1)$-prefixes (case 1 in Theorem 1).

If i is even, then in the tree representation of the BRGC encoding there exists an even number of subtrees of height $k+1$ before the subtree that represents R_1. Thus, the roots of the subtrees that represent R_1, R_2 are siblings, and the first $w-k-2$ bits of their ternary BRGC codewords are equal. Wildcarding the $k+2$ least significant bit would yield a $(k+2)$-prefix that represents their union and thus represents R.

If i is odd, then the roots of the subtrees that represent R_1, R_2 are not siblings, but they do have some common ancestor. We denote the right bound of R_1 as $x = (2i+2) \cdot 2^k - 1$ and the left bound of R_2 as $y = (2i+2) \cdot 2^k$. x and y are two consecutive integers and thus their BRGC representation differ in one bit only. If we assume towards a contradiction that the difference is in one of the $k+1$ least significant bits, then both BRGC codewords of these points share the same prefix of length that is greater than $w-k-1$, so the two values can be represented using the same subtree of height $k+1$, which is impossible as $R_1 \neq R_2$. Thus, the difference between the two codewords is in one of the $w-k-1$ most significant bits, and the $(k+1)$-semi-prefix that has '$*$'s in this bit and in the $k+1$ least significant bits, represents the union of R_1 and R_2 which is R. ■

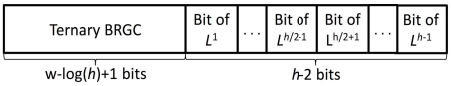

Fig. 5. Encoding structure for a value or range of length h.

Figure 4 illustrates the BRGC encoding tree for all points in the range $[0, 15]$, and the ternary encoding for all intervals of lengths $4, 8$ that satisfy the condition above.

Theorem 1 implies that when encoding ranges of length h, the $\log_2(h)-1$ least significant bits are always '$*$' thus one can save TCAM space by omitting these uninformative bits.

C. An Encoding Function for Ranges

We call those ranges that can be encoded using a single ternary BRGC codeword *trivial ranges*, and all other ranges *nontrivial ranges*. In this section, we extend the BRGC encoding scheme so that it can encode in a single ternary word nontrivial ranges as well. We append *extra bits* to the end of BRGC codewords, as depicted in Figure 5: $w - \log_2(h) + 1$ bits are used for the binary BRGC encoding of a value $v \in \mathcal{U}$, or for the ternary BRGC encoding of some trivial range R. To encode nontrivial ranges of length $h = 2^k$ ($k \in \mathbb{N}_0$), at most $h-2$ bits are added as extra bits.

The key idea of RENÉ is to divide all ranges of some length $h = 2^k$ ($k \in \mathbb{N}_0$) into h *layers*, such that a layer L_h^i is the set of all ranges $[x, x+h)$ for which $x \bmod h = i$. Note that two of these layers contain only trivial ranges (L_h^0 and $L_h^{2^{k-1}}$). We are left with $h-2$ layers that contain nontrivial ranges. We assign an extra bit for each layer of nontrivial ranges. The value of this bit alternates between adjacent ranges in the same layer, such that for any pair of consecutive ranges in the same layer, the value of the bit corresponding to this layer is different. Hence, for a value $v \in \mathcal{U}$, $\texttt{tcode}(v)$ is the $1 + w - log_2(h)$ most significant bits of $BRGC(v)$, concatenated with $h-2$ extra bits. The value of the i^{th} extra bit corresponds to layer L_h^i and is set to $\left\lfloor \frac{v-i}{h} \right\rfloor \bmod 2$.

For nontrivial ranges we define their *cover range* as follows:

Definition 6: For any nontrivial range of length $h = 2^k$ ($k \in \mathbb{N}_0$), $R = [x, x+h)$, let the **cover range** of R, denoted by $\texttt{cover}(R)$, be the range $[\lfloor x/h \rfloor \cdot h, (\lfloor x/h \rfloor + 2) \cdot h)$.

We first notice the following property of cover ranges:

Lemma 1: For any range $R = [x, x+h)$ of length $h = 2^k$ ($k \in \mathbb{N}_0$), $\texttt{cover}(R)$ fully contains R.

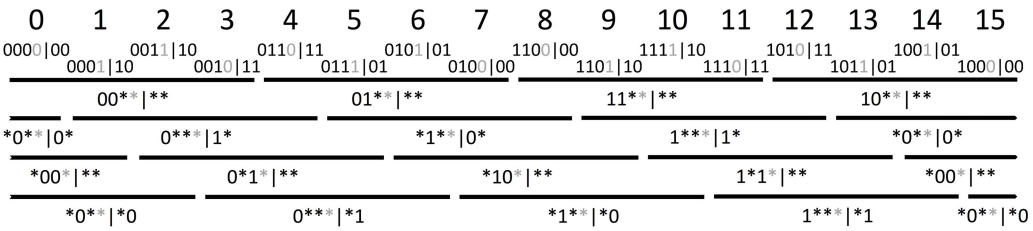

Fig. 6. Encoding for all sub-ranges of length 4 and values in range $[0, 16)$. Left bits are the ternary BRGC encoding. Right bits are the extra bits for nontrivial layers. The bits in gray can be removed as explained in Section III-B.

Proof: Assume by contradiction that R starts before $\texttt{cover}(R)$ starts or ends after $\texttt{cover}(R)$ ends. If R starts before $\texttt{cover}(R)$, $x < \lfloor x/h \rfloor \cdot h$. So $x/h < \lfloor x/h \rfloor$, which is of course impossible. Also, if R ends before $\texttt{cover}(R)$ ends, $x + h > (\lfloor x/h \rfloor + 2) \cdot h$. This implies that $x/h > \lfloor x/h \rfloor + 1$ which is also impossible, and thus a contradiction. ∎

Note that the existence of the cover range is a unique property of the binary-reflected Gray code. The cover range $\texttt{cover}(R)$ helps us distinguish R from other ranges in the same layer. For range $R = [x, x + h) \in L_h^i$ of length $h = 2^k$ ($k \in \mathbb{N}_0$), $\texttt{tcode}(I)$ starts with the $1 + w - log_2(h)$ most significant bits of the ternary BRGC representation of R, if R is trivial, or of $\texttt{cover}(R)$, if R is nontrivial. Then, $h-2$ extra bits are concatenated one after the other where the i^{th} bit is either * if $I \notin L_h^i$ or $\left\lfloor \frac{x-i}{h} \right\rfloor \bmod 2$ otherwise (namely, all the extra bits except one are *).

Our main result is that RENÉ's encoding function, $\texttt{tcode}$, is an *admissible encoding function* for ranges of any length $h = 2^k$ ($k \in \mathbb{N}_0$). The total length of the admissible encoding produced by $\texttt{tcode}$ for a single value or range is hence $w - \log(h) + h - 1$.

Before proving this result (Theorem 2), we introduce the following two technical lemmas:

Lemma 2: If a range R is nontrivial, then no other range from the same layer is fully contained in $\texttt{cover}(R)$.

Proof: Assume $R = [x, x + h)$, where $h = 2^k$, and that there exists another range from the same layer, R', that is also fully contained in $\texttt{cover}(R)$. By the definition of the layers, R and R' are both of length h and are not overlapping. By Definition 6, $\texttt{cover}(R)$ is of length 2^{k+1}, implying that the union of R and R' is equal to $\texttt{cover}(R)$, but since both ranges are fully contained in the cover, the union is exactly the cover. This, in turn, implies that $x \bmod h = \lfloor x/h \rfloor$. Choosing $i = 2 \lfloor x/h \rfloor$ yields that $R = [i \cdot 2^k, (i + 2) \cdot 2^k)$ and thus, by Theorem 1, R is a trivial range, in contrast to the assumption. ∎

Based on this property of cover ranges, we can completely distinguish between ranges in the same layer using the extra bits we added to the ternary BRGC encoding:

Lemma 3: Let $R = [x, x + h)$, where $h = 2^k$, be a range in L_h^i. For every value v in $\texttt{cover}(R)$, if $v \in R$, v has the same bit value as R, and if $v \notin R$, then v has the opposite bit value.

Proof: Assume that a value $v \in R$, has a bit value that is different than the bit value of R. v is in R so $v - x \leqslant h$, and thus $\left\lfloor \frac{v-i}{h} \right\rfloor = \left\lfloor \frac{x-i}{h} \right\rfloor$, meaning that the bit value of v must be equal to the bit value of R.

To prove the other direction, assume that a value $v \notin R$ has the same bit value as R. Let $R_{\text{before}} = [x - h, x)$ be the range that precedes R in L_h^i and $R_{\text{after}} = [x + h, x + 2h)$ be the range that succeeds R in L_h^i. The bit value of R_{before} and R_{after} must be different than the bit value of R as they are both adjacent to R. Since $v \in \texttt{cover}(R)$ but not in R, and since the length of $\texttt{cover}(R)$ is at most $2h$, v must be either in R_{before} or in R_{after}, and thus it must have the opposite bit value than R. ∎

We now turn to the main theorem.

Theorem 2: The function $\texttt{tcode}$ is an admissible encoding function for ranges of length $h = 2^k$.

Proof: Assume that there exist a value v and a range R for which $v \in R$ but $\texttt{tcode}(v) \not\approx \texttt{tcode}(R)$. v is in R so the ternary BRGC of R (in case R is trivial) or of $\texttt{cover}(R)$ (in case R is nontrivial) must match the BRGC encoding of v ternary-wise. Thus, some extra bit does not match. Since for trivial ranges all extra bits are *, R must be nontrivial. For nontrivial ranges, only one extra bit in $\texttt{tcode}(R)$ is not a '*'. However, this bit must be equal to the corresponding bit in $\texttt{tcode}(v)$ by Lemma 3, which is a contradiction to the assumption that $\texttt{tcode}(v) \not\approx \texttt{tcode}(R)$.

To prove the opposite direction, assume that there exist a value v and a range R for which $\texttt{tcode}(v) \approx \texttt{tcode}(R)$ but $v \notin R$. The BRGC encoding of v must match the ternary BRGC encoding of R (in case R is trivial) or $\texttt{cover}(R)$ (if R is nontrivial). If R is trivial and there is a match then $v \in R$, as all extra bits in $\texttt{tcode}(R)$ are '*'. Thus, R must be nontrivial, and v must be inside $\texttt{cover}(R)$. However by Lemma 3, if $v \in \texttt{cover}(R)$ and has the same bit value as R for the layer R belongs to, then v must be in R. ∎

Figure 6 shows the encoding of all sub-ranges of length 4 in range $[0, 16)$. Note that the first and third layers do not require extra bits, so these are both set to * in their encoding. In other layers, the corresponding extra bit alternates between ranges in the same layer. For example, the range $[1, 4]$, which cannot be encoded solely using a ternary BRGC codeword, is encoded as `0***1*`, where the fifth bit is the extra bit that corresponds to the second layer. Only points in $[1, 4]$ match this encoding.

D. Encoding Multiple Range Lengths

Given RENÉ's encoding function for ranges of some maximal length h_{max} we can encode, without using more bits, *all ranges whose lengths are smaller than* h_{max} as well. We define a logical conjunction operation, denoted by $\sqcap$, to encode the intersection of two ranges. The truth table

TABLE I
THE TRUTH TABLE OF A TERNARY LOGICAL CONJUNCTION, DENOTED BY THE $\sqcap$ OPERATOR

a_i	b_i	$a_i \sqcap b_i$
$\star$	$\star$	$\star$
0	$\star$	0
$\star$	0	0
0	0	0
1	$\star$	1

a_i	b_i	$a_i \sqcap b_i$
$\star$	1	1
1	1	1
0	1	$\perp$
1	0	$\perp$

of such a conjunction is given in Table I. $\perp$ means an undefined output, and we later make sure to never get such an output when using this operation. For two ternary words, $a = a_0, \ldots, a_m$ and $b = b_0, \ldots, b_m$, the conjunction $c = a \sqcap b$ is the ternary word where $c_i = a_i \sqcap b_i$. If at least one of the symbols c_i is $\perp$, then c is also marked as $\perp$ and is not defined. Note that the conjunction operation is independent of the specific encoding function.

The essence of the conjunction operation is captured in the following two lemmas:

Lemma 4: For any value $v \in \mathcal{U}$ and any two ranges $R_1, R_2 \subseteq \mathcal{U}$, if **tcode** is an admissible encoding function for R_1 and R_2, then $\textbf{tcode}(v) \approx \textbf{tcode}(R_1) \sqcap \textbf{tcode}(R_2)$ if and only if $v \in R_1 \cap R_2$.

Proof: Assume $\textbf{tcode}(v) \approx \textbf{tcode}(R_1) \sqcap \textbf{tcode}(R_2)$ and $v \notin R_1 \cap R_2$. Without loss of generality, assume $v \notin R_1$. Since $v \notin R_1$ and **tcode** is an admissible encoding function, there exists some i for which $\textbf{tcode}(v)_i \neq \star$, $\textbf{tcode}(R_1)_i \neq \star$, and $\textbf{tcode}(v)_i \neq \textbf{tcode}(R_1)_i$. Without loss of generality, let $\textbf{tcode}(v)_i = 0$, so $\textbf{tcode}(R_1)_i = 1$ and therefore $\textbf{tcode}(R_1)_i \sqcap \textbf{tcode}(R_2)_i$ is either 1 or $\perp$. Thus, by definition, $\textbf{tcode}(v)_i \not\approx \textbf{tcode}(R_1)_i \sqcap \textbf{tcode}(R_2)_i$ implying $\textbf{tcode}(v) \not\approx \textbf{tcode}(R_1) \sqcap \textbf{tcode}(R_2)$, which is a contradiction.

To prove the other direction, assume that $v \in R_1 \cap R_2$. Since $v \in R_1$ and $v \in R_2$ and **tcode** is admissible, $\textbf{tcode}(v)_i \approx \textbf{tcode}(R_1)_i$ and $\textbf{tcode}(v)_i \approx \textbf{tcode}(R_2)_i$, for any i. The admissibility of **tcode** also implies that $\textbf{tcode}(v)_i$ is either 0 or 1. Assume without loss of generality that for some i it is 0. Then, $\textbf{tcode}(R_1)_i$ and $\textbf{tcode}(R_2)_i$ are either 0 or $\star$. Hence, $\textbf{tcode}(R_1)_i \sqcap \textbf{tcode}(R_2)_i$ is either 0 or $\star$, and therefore, $\textbf{tcode}(v)_i \approx \textbf{tcode}(R_1)_i \sqcap \textbf{tcode}(R_2)_i$. Since this is true for any i, it implies that $\textbf{tcode}(v) \approx \textbf{tcode}(R_1) \sqcap \textbf{tcode}(R_2)$, and the claim follows. ∎

Lemma 5: If **tcode** is an admissible encoding function for R_1 and R_2, and the result of $\textbf{tcode}(R_1) \sqcap \textbf{tcode}(R_2)$ is $\perp$ then $R_1 \cap R_2 = \emptyset$.

Proof: Assume $\textbf{tcode}(R_1) \sqcap \textbf{tcode}(R_2) = \perp$ and to the contrary, that $R_1 \cap R_2 \neq \emptyset$. Then, there exists some i for which, without loss of generality, $\textbf{tcode}(R_1)_i = 0$ and $\textbf{tcode}(R_2)_i = 1$, and some value $v \in R_1 \cap R_2$. From the admissibility of **tcode**, if $v \in R_1$, then $\textbf{tcode}(v) = 0$, and thus $v \notin R_2$, and if $v \in R_2$, then $\textbf{tcode}(v) = 1$, and thus $v \notin R_1$, which is a contradiction. ∎

Note that the other direction of Lemma 5 is not necessarily true: the conjunction of codes of two disjoint ranges may not be $\perp$.

We assume that there is a value $h_{max} = 2^{k_{max}}$, which is the maximum length we should consider. Note that any range $[x, x+h)$ of length $h < h_{max}$ (h is not necessarily a power of 2 anymore) can be written as the intersection of two ranges of length h_{max} as follows:

$$[x, x+h) = [x+h-h_{max}, x+h) \cap [x, x+h_{max}).$$

Using the conjunction operator and Lemma 4 we can construct the code for ranges of any length $h \leqslant h_{max}$, with $h_{max} - 2$ extra-bits:

$$\textbf{tcode}([x, x+h)) = \textbf{tcode}([x+h-h_{max}, x+h)) \sqcap \textbf{tcode}([x, x+h_{max})).$$

We also know from Lemma 5 that $\textbf{tcode}([x, x+h))$ is not $\perp$ as the intersection is never empty.

The encoding function $\textbf{tcode}(v)$ for some value v when using any range lengths up to h_{max} is shown in Algorithm 1. When encoding a range R of length $h \leqslant h_{max}$ that is centered at some point v, Algorithm 2 is used to obtain $\textbf{tcode}(R)$.

The length of the resulting encoding of a value $v \in \mathcal{U}$ or a range $R \subseteq \mathcal{U}$ is therefore $w - \log_2(h_{max}) + h_{max} - 1$.

Algorithm 1 Encoding Function for a Value v

```
function tcode(v, h_max)
                    ▷ v - value, h_max - maximal range length
  word ← BRGC(p) ≫ (log2(h_max) − 1)        ▷ Bitwise shift
  for i ← 0 to (h_max − 1) do
    if layer is skipped then
      continue                  ▷ Optional - encode less layers
    end if
    if layer is nontrivial (i ≠ 0 and i ≠ h_max/2) then
      b ← ⌊(v − i)/h_max⌋ mod 2
      word ← (word ≪ 1)|b                    ▷ Bitwise OR
    end if
  end for
  return word
end function
```

E. Running Time Analysis

Computing **tcode** for either a value or a range is simple: results only depend on the value or range themselves, and the maximal range length h_{max}. The running time of both functions, for a value and a range, is linear with h_{max}, and does not depend in the number of encoded ranges: $O(h_{max})$ when encoding a value and $O(h_{max} + h)$ when encoding a range of length $h \leqslant h_{max}$.

F. Lower Bound on the Number of Bits per Range

As previously recalled, Lakshminarayanan *et al.* [14] introduced the worst-case necessary condition of $2^w - 1$ bits to encode a w-bit range. We use this observation to introduce a lower bound for the number of bits required to encode ranges, when their lengths are limited by some upper bound h_{max}:

Theorem 3: In order to represent any ranges up to length h_{max} without row expansion, in a field of width w bits, at least $max(h_{max} - 1, w)$ bits are necessary.

Algorithm 2 Encoding Function for a Range $[s,t]$

function `tcode` $([s,t], h_{max})$
  ▷ $[s,t]$ - range, h_{max} - maximal range length
  if $t-s+1 \neq h_{max}$ **then**
    ▷ Encode range as an intersection
    $R_1 = [s, s+h_{max}-1]$
    $R_2 = [t-h_{max}+1, t]$
    $\Gamma \leftarrow \{R_1, R_2\}$
  else ▷ Encode range directly
    $\Gamma \leftarrow \{[s,t]\}$
  end if
  $result \leftarrow 0$
  $count \leftarrow 0$
  for $[x,y] \in \Gamma$ **do**
    $mask \leftarrow 0$
    for $i \leftarrow x+1$ to y **do**
      $mask \leftarrow mask | (BRGC(i-1) \oplus BRGC(i))$
      ▷ bitwise OR and XOR
    end for
    $word \leftarrow BRGC(x) \gg (\log_2(h_{max}) - 1)$
    $mask \leftarrow mask \gg (\log_2(h_{max}) - 1)$
    for $i \leftarrow 0$ to $(h_{max}-1)$ **do**
      if layer is skipped **then**
        continue ▷ Optional - encode less layers
      end if
      if layer is nontrivial ($i \neq 0$ and $i \neq \frac{h_{max}}{2}$) **then**
        if $x \bmod h_{max} \neq i$ **then** ▷ Irrelevant layer
          $mask \leftarrow (mask \ll 1)|1$ ▷ Put a '*'
          $word \leftarrow word \ll 1$
        else ▷ $[x,y]$ is in this layer
          $mask \leftarrow mask \ll 1$
          $b \leftarrow \left\lfloor \frac{x-i}{h_{max}} \right\rfloor \bmod 2$
          $word \leftarrow (word \ll 1)|b$
        end if
      end if
    end for
    if $count > 0$ **then**
      $result \leftarrow result \sqcap (word, mask)$
    else
      $result \leftarrow (word, mask)$
    end if
    $count \leftarrow count + 1$
  end for
  return $result$
end function

Proof: The maximal range length h_{max} is given as some fixed value. We show that the theorem is correct for any $w \geqslant \log_2(h_{max})$, as a field with less bits than that cannot have ranges of length h_{max}. For $w = log_2(h_{max})$, the range field starts at 0 and ends at $h_{max}-1$ and is of size of exactly h_{max}. According to the condition in [14, Th. 1], to encode all ranges in this field, the worst-case length of the ternary representation is at least $2^w - 1 = h_{max} - 1$ bits.

The proof of [14, Th. 1] stems from the fact that for a range of size 2^w there are $2^w - 1$ contained sub-ranges that are also contained in each other (and thus are overlapping), and a single bit per range is necessary to distinguish between them. When using a range field with more bits (i.e. larger w), we do not reduce the number of possibly overlapping range. Thus, the number of required bits cannot be lower than $h_{max} - 1$ (note that this lower bound is not necessarily tight).

In any case, and specifically when $w > h_{max} - 1$, at least w bits are necessary to represent singular values (ranges of length 1). ■

IV. RENÉ FOR PACKET CLASSIFICATION

Range encoding on TCAM has been used for packet classification for long time. Row expansion significantly limited its usage when multiple header fields are ranges, leading vendors and administrators to avoid such situations as much as possible. However, next generation SDN applications, such as load balancers, security tools, and quality of service, rely on sophisticated packet classification that is performed on the datapath itself (i.e. the switch) [9]–[11], [60]. Most of these solutions require range based matching on multiple header fields. We summarize several examples for such fields and metadata information that can benefit when using RENÉ:

- **TCP/UDP Port Fields:** In real-life datasets, short ranges (up to length 64) sometimes consist more than 60% of the unique ranges [15]. Thus, if a network administrator uses mainly short ranges for TCP/UDP port fields, or even for only one of these fields, RENÉ may suit their needs.
- **Network ToS (or DSCP):** In both the deprecated ToS field and the new DSCP field the precedence is set using an increasing value, and to specify one or more precedence classes, either an exact value or a short range should be used.
- **Packet Size:** Packet size (e.g. IP total length field) can be a useful piece of information for packet classification. When classifying according to this property, a categorization can be done in order to reduce range lengths. As usually one does not classify packets according to a specific length, but rather according to categories (small, medium, large, etc.), short ranges can be used to represent multiple categories. For example, a recent attack named *Tsunami SYN Flood Attack* can be identified based on the size of packets (about 1000 bytes or more) [61].
- **Timestamp and Counters:** Recent works suggest adding packet's metadata such as hit counters and timestamps (or time deltas) to classification data path, for example in OpenFlow switches [60]. It is likely that classification on such fields would be based on ranges and not on exact values, and thus RENÉ may be used.
- **IP Spoofing Detection:** In order to protect against IP spoofing and attacks that use this technique (e.g., DDoS), it was suggested to inspect the IP TTL value and conclude about possible spoofed packets [62], [63]. The detection is based on the fact that the TTL value does not change dramatically over short time for the same host or subnet, and these values can be found using `ping` and other tools. Thus, if a packet with IP from a known subnet comes with a TTL value that is too far from the expected value, it is classified as spoofed and dropped.

Since the TTL value is not compared to an exact value, but rather to a short range, RENÉ can be used in order to implement IP spoofing detection on classification hardware with TCAM.

- **AS Numbering:** BGP routers and SDX [64] sometimes make classification decisions based on autonomous systems (ASes) numbers. Large ISPs and content providers usually hold multiple, consecutive AS numbers [65], which form one or more short ranges. For example, Comcast has multiple such short ranges 7015-7016, 33489-33491, 33650-33668 (in addition to five more non-consecutive AS numbers). Grouping AS numbers to ranges can reduce the total number of classification rules, as long as no row expansion is induced. RENÉ fits this goal as the ranges are short and it induces no row expansion.

A. Evaluation and Experiments

1) Experiment on an OpenFlow Switch: We implemented RENÉ and a sample SDN application that uses it for packet classification over the Ryu SDN controller [66]. We use a NoviFlow NoviKit 250 hardware switch that supports OpenFlow 1.3 [3] and has an internal TCAM. Our code is available at `https://github.com/yotamhc/rene`.

Classification uses OpenFlow table pipeline in the following way: First table, given a destination TCP port for a packet, writes its translation into RENÉ's encoding to the metadata field (using the OpenFlow's Write-Metadata instruction). This table is precomputed on the controller and contains up to 64K entries - an entry for each port number. Then, second table matches the packet according to the metadata only (original port information is not necessary at this stage), and forwards it accordingly.

On the same switch, when a packet is classified based on only its TCP port, without table pipeline, the total round-trip time to and from the switch, using a 1Gbps copper link, is 157μs. Using our table pipeline, such round-trip takes 161μs. Thus, latency increases by only 2.5%, which is a negligible factor.

2) Quantitative Evaluation: To evaluate the quality of RENÉ's encoding function **`tcode`** we compare it with best prior-art encoding techniques that can provide no row expansion: DIRPE [14], a database-independent encoding scheme and LIC [20], a database-dependent encoding scheme. We do not compare RENÉ to SRGE [15], for example, as it requires row expansion. We evaluate the database dependent scheme LIC both in its worst case, where all ranges are to be represented, and using a commercial classification dataset with 257 range rules. Since our goal is no expansion of TCAM entries, we compare the amount of TCAM bits required for a single range field, such that no expansion is induced whatsoever. Using the classification database, LIC performs worse than RENÉ on ranges up to length 32. When the dataset contains much higher number of ranges, LIC always performs worse than RENÉ.

Figure 7 shows the bit requirement of each encoding technique, given the maximal length of encoded ranges, assuming a 16-bits range field. In addition, it shows the lower bound

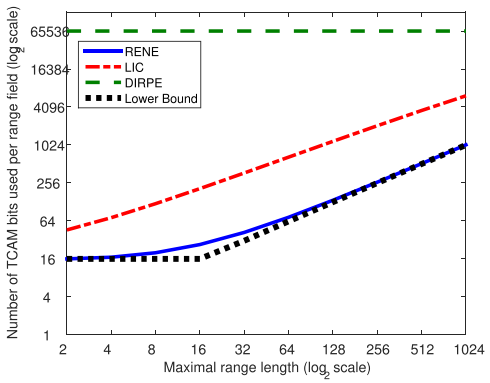

Fig. 7. Analysis of the number of TCAM bits required for a 16-bits range field when representing all ranges of up to a given length.

of $max(h_{max} - 1, w)$ bits (see Theorem 3), as a black, dotted line. Evidently, RENÉ (blue, solid line) is much closer to the lower bound than all other techniques. Moreover, the bit requirements for database-dependent techniques such as LIC [20] are higher by an order of magnitude, when *all* ranges up to a certain length should be encoded. The database-independent technique DIRPE [14] always requires $2^w - 1$ bits as it does not use the additional information on the maximal range length.

V. RENÉ for Nearest Neighbor Search

TCAM is a powerful device with high parallelism that can also be used for tasks outside of the networking fields. Just as TCAM has broken the performance limits of packet classification and IP lookup in networking, it can also be used to break such computational limits in problems from other fields, serving as a coprocessor for the CPU, similarly to a GPU or FPGA. RENÉ can be used to implement on such TCAM applications that use the nearest-neighbor problem or its variants. We show several such variants in this section, and by experiments and simulations we show that RENÉ can improve their performance by orders of magnitudes.

Multidimensional nearest neighbor search (NN) lies at the core of many computer science applications. The formal definition of the problem in the space of integers is provided in Section II-C.

The NN problem and its variants are utilized in a wide range of applications, such as spatial search, object recognition, image matching, image segmentation, classification and detection, to name a few [26]–[28].

In this section we present super high-speed algorithms for the NN problem using TCAM as a coprocessor, and our encoding scheme RENÉ. The proposed algorithms solve the ANN problem with ℓ_∞-normed distance using *a single TCAM lookup* and linear space.

The r–Near-Neighbor Report Problem is a simpler problem that we will use as a building block in our algorithms. It only searches for a neighbor close enough to the query point, or discovers that there is no such neighbor at all. It is formally defined in Definition 2.

Using RENÉ's encoding function **`tcode`** , a single ternary match can determine whether a given d-dimensional point is

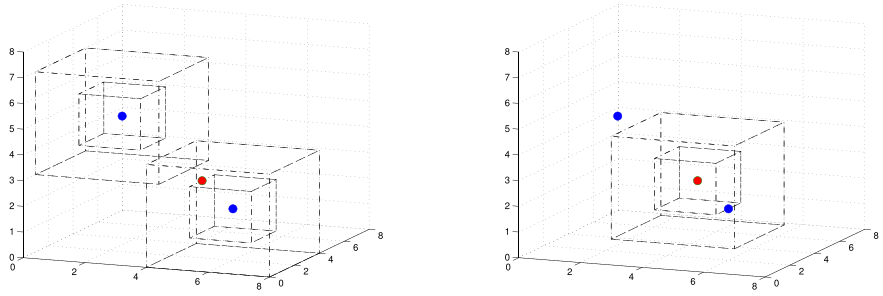

Fig. 8. Illustration of the two alternative algorithms for Nearest Neighbor Search using TCAM. Left: Encoding nested cubes around each point in the database. A query is a point in this encoding, and the result is the smallest cube encompassing the query point. Right: Encoding cubes around the query points and the data is encoded as points. A query is a sequence of nested cubes in increasing edge length. The result is the first data point that matches a cube.

Algorithm 3 Encoding Function for a d-Dimensional Point

function TCODE($p[], d, h_{max}$)
  $word \leftarrow \varepsilon$
  for $i \leftarrow 1$ to d **do**
    $word \leftarrow word +$ **tcode**$(p[i], h_{max})$
  end for
  return $word$
end function

Algorithm 4 Encoding Function for a d-Dimensional Cube

function TCODE($p[], d, h, h_{max}$)
  $word \leftarrow \varepsilon$
  for $i \leftarrow 1$ to d **do**
    $word \leftarrow word +$ **tcode**$([p[i] - \lfloor h/2 \rfloor, p[i] + \lfloor h/2 \rfloor], h_{max})$
  end for
  return $word$
end function

inside a given d-dimensional cube: To encode a d-dimensional point, or a d-dimensional cube, each coordinate is encoded using the **tcode** function, and the codewords of all d coordinates are concatenated into a single ternary word. The encoding functions for a d-dimensional point and for a d-dimensional cube are shown in Algorithm 3 and in Algorithm 4, respectively.

A. Approximate Nearest-Neighbor Search

Our APPROXIMATE NEAREST-NEIGHBOR SEARCH algorithms solve in fact multiple instances of the r-NEAR NEIGHBOR REPORT PROBLEM for increasing values of r. In ℓ_∞, the value of r defines a cube around each data point p such that for all query points q inside that cube, p is a valid solution of the r-NEAR NEIGHBOR REPORT PROBLEM with q, and for all query points outside that cube p is not a valid solution.

Our *time-efficient method* solves the APPROXIMATE NEAREST-NEIGHBOR SEARCH in ***a single TCAM lookup***. Given a set $\mathcal{H}$ of edge lengths, let $h_{max} = \max_h \mathcal{H}$. For each point $p \in S$ and $h \in \mathcal{H}$ we store a TCAM entry representing a d-dimensional cube centered at p, with edge length h. Entries are sorted by the value of h: the smaller h is, the higher the priority of the entry.

Given a query point $q \in [0, w)^d$, we use **tcode** to build a d-dimensional point representation for maximal edge length of h_{max}, and use a *single TCAM lookup* to find the smallest cube that contains the point q. The TCAM returns the highest priority entry that matches, which is the entry of the cube that is centered at some point p, has the shortest edge length, and contains q. An example is shown in Figure 8 (left). Note that in general, p is not necessarily the exact nearest neighbor of q (as there may be more than one such cube with the same edge length h). However, the distance (under ℓ_∞) of q from its exact nearest neighbor is strictly more than $\left\lfloor \frac{1}{2} \max_{h' \in \mathcal{H}} \{h' < h\} \right\rfloor$. As we will show later, by carefully choosing the edge length set, we can obtain a $c = 1 + \varepsilon$ approximation factor, where the size of $\mathcal{H}$ is inversely proportional to ε.

In our *memory-efficient method*, the data points and query points switch roles: we store in the TCAM a single entry for each data point. The order of the entries does not matter. Upon a query q, we construct a sequence of $|\mathcal{H}|$ cubes centered in p with edge lengths in $\mathcal{H}$. Then, we perform TCAM lookups with cubes of increasing edge length values until a match is found. As in the previous method, if a point was matched with a query of edge length h, then it is a solution of the $h/2$-NEAR NEIGHBOR REPORT PROBLEM.

1) Analysis of Approximation: Let $\mathcal{H} = \{h_1, h_2, \ldots, h_{max}\}$ such that $h_i < h_j$ for each $i < j$. Matching a data point p corresponding to a cube with edge length h_i implies that $D(p, q) \leq \left\lfloor \frac{h_i}{2} \right\rfloor$ (where D is defined under ℓ_∞). Since h_i is the first edge length to be matched, $D(s^*, q) \geq \left\lfloor \frac{h_{i-1}}{2} \right\rfloor + 1$ (s^* is the exact nearest neighbor of q). This implies that under ℓ_∞, both methods solve the c-APPROXIMATE-NEAREST-NEIGHBOR PROBLEM for $c = \max_{h_i \in \mathcal{H}} \frac{\lfloor h_i/2 \rfloor}{\lfloor h_{i-1}/2 \rfloor + 1}$, where h_i is the i^{th} smallest element in $\mathcal{H}$ and $h_1 = 1 \in \mathcal{H}$.[3]

In order to get the exact nearest neighbor in ℓ_∞, one can choose $\mathcal{H}$ to be the set of odd numbers. Reducing the size of $\mathcal{H}$ reduces the number of required entries, but decreases the quality of the results. For example, to get a c-approximate solution, $\mathcal{H}$ can consist of all even values up to $2/(c-1)$, along with the values of a geometric series starting at $2/(c-1)$, with the parameter c.

[3]To get a bounded approximation ratio, 1 must be added to $\mathcal{H}$.

When distances are defined under ℓ_p norms, for finite values of $p \geq 1$, the approximation ratio is at most $c \cdot \sqrt[p]{d}$ in ℓ_p, where c is the approximation ratio in ℓ_∞.

2) Database Update: Our algorithms allow efficient hot updates in the lookup database (the set S). Deletion of data points is trivial (simply delete all corresponding entries from the TCAM). When using the *time-efficient method*, efficient addition of new points is possible by keeping some empty TCAM entries between entries of different edge length, by adding entries for the corresponding cubes in these empty slots. Also, one can track deletion for more empty slots. Nevertheless, this further increases space requirement.

When using the *memory-efficient method*, the situation is simpler: since the order of entries is not important, point addition or deletion requires a single TCAM entry update.

B. Exact Nearest-Neighbor Search in ℓ_p

Our algorithms achieve $\sqrt[p]{d}$ approximate solution under ℓ_p norm. We suggest the following extension to find (exactly) the nearest neighbor under ℓ_p: For each data point $s \in S$ and for each edge length $h \in \mathcal{H}$, we *precompute* the *neighborhood set* $\mathcal{N}(s,h) = \{s' \in S \mid D_p(s,s') \leq h\sqrt[p]{d}\}$, where D_p is the distance between the two points under the p-norm. The neighborhood sets are stored in memory. Precomputing these sets is possible since datasets are relatively static and the neighborhood sets do not depend on the query points.

Since for every two points, the distance in ℓ_p is at most $\sqrt[p]{d}$ the distance in ℓ_∞, we immediately conclude that if the algorithms described in Section V-A return a data point s for query point q with some distance $h \in \mathcal{H}$, then the exact nearest neighbor in ℓ_p of q is in $\mathcal{N}(s,h)$.

While this method requires additional computations following the TCAM lookup, in most datasets the number of points in $\mathcal{N}(s,h)$ would be very small. In our experiments (see Section V-E) $\mathcal{N}(s,h)$ contained only s itself for lower values of h in most cases and was small even for higher values of h. Thus, the time required to find the exact nearest neighbor is still much shorter than that required for brute-force over all points in the database.

The precomputed neighborhood sets can also be used to find k-nearest neighbors instead of only one. However, the number of neighbors in these sets might be smaller than k, so one TCAM lookup might not suffice. To find the set of k exact nearest neighbors, the lookup process should continue until k or more neighbors are found, and also until no more neighbors are found in cubes whose edge length is equal to that of previous neighbors. This process is formally described in Algorithm 5, assuming a multi-match technique such as the one presented in [14] is used.

C. Algorithms for the Partial Match Problem

The PARTIAL MATCH PROBLEM is defined as follows:

Definition 7: Given a set of data points $S = \{p_i\}_{i=1}^n$, $p_i \in \mathbb{Z}^d$, a query point $q \in \mathbb{Z}^d$, and a subset of the dimensions $D_q \subseteq \{1,\ldots,d\}$ of size $d_q < d$, THE PARTIAL MATCH PROBLEM is to find the point $s^* = \arg\min_{s' \in S} D(s',q)|_{D_q}$, where $D(a,b)|_{D_q}$ is the distance between a and b under some norm in the d_q-dimensional space. Namely, for a p-norm,

$$D(a,b)|_{D_q} = \left(\sum_{i \in D_q} |a_i - b_i|^p \right)^{1/p}.$$

This problem is useful when some features in the vector are not important for a specific query or user, and in traditional computing models it is known to be *more difficult* [67] than the nearest neighbor problem, where all relevant dimensions are given a-priori. For example, LSH (and its extension to TCAMs, TLSH [7]) cannot be used to solve this problem. However, our solution for the NN problem can be used instantly to solve the partial match problem.

Under the maximum norm ℓ_∞, a PARTIAL MATCH solution is to replace, *in the queries*, all the bits corresponding to coordinates in irrelevant dimensions with * bits. We replace coordinates in queries and not for data point, as the relevant dimensions are selected per query. This technique works both for our *time-efficient* and *memory-efficient methods*.

For ℓ_p, our solution results in $\sqrt[p]{d_q}$ approximation, where d_q is the dimension of the specific query. The extensions to EXACT NEAREST NEIGHBOR SEARCH and k–NEAREST NEIGHBORS SEARCH, as described in Section V-B, work also for this problem. The neighborhood sets are precomputed on the d-dimensional space, but queries and distance computations after queries are done on the specific d_q dimensional space. The results are still correct as distances in the d_q-dimensional space are bounded by distances in the d-dimensional space.

Algorithm 5 Exact k-NEAREST NEIGHBORS SEARCH Algorithm in ℓ_p

```
function FIND-EXACT-KNN(q, S, k)
    N = ∅                                   ▷ Candidate neighbors set
    h_last = −1
    repeat
        (s, h) ← TCAMLOOKUP(q, S)
        ▷ returns the datapoint and correponding edge length
        if |N| < k or h_last = −1 or h = h_last then
            N ← N ∪ 𝒩(s, h)
            h_last ← h
        end if
    until |N| ⩾ k and h > h_last
    R ← arg min^k_{s'∈N} D_p(s', q)          ▷ k min-distance points
    return R
end function
```

D. Geometric Clustering on TCAM

Another closely related problem that could benefit from using TCAM with RENÉ is high-dimensional geometric clustering. The k-MEANS CLUSTERING problem, for example, is usually solved as a sequence of nearest-neighbor search problems, each of these consists of a database with k d-dimensional points [68].

Algorithm 6 shows how the traditional k-MEANS CLUSTERING algorithm can be implemented on TCAM using our

Algorithm 6 k-MEANS CLUSTERING Algorithm on TCAM

function FIND-K-MEANS(S, k)
  $changed \leftarrow false$
  $t \leftarrow 1$
  Randomly select $D \leftarrow \{d_1, \ldots, d_k\} \subseteq S$ ($|D| = k$)
  for $i \leftarrow 1$ to k **do**
    $C_0^i \leftarrow \emptyset$
    $C_1^i \leftarrow \emptyset$
  end for
  repeat
    Clear TCAM
    for $h \leftarrow 1$ to $\mathcal{H}$ **do**
      for $i \leftarrow 1$ to k **do**
        Add **`tcode`** (d_i, h) to end of TCAM
      end for
    end for
    for each $s \in S$ **do**
      $d_i \leftarrow$ query TCAM with **`tcode`** (s)
      if $s \notin C_{t-1}^i$ **then**
        $changed \leftarrow true$
        $C_t^i \leftarrow C_t^i \cup \{s\}$
      end if
    end for
    if $changed$ **then**
      for $i \leftarrow 1$ to k **do**
        $d_i \leftarrow$ center of C_t^i
        $C_{t+1}^i \leftarrow \emptyset$
      end for
      $t \leftarrow t + 1$
      $changed \leftarrow false$
    end if
  until $changed = false$
  return $D = \{d_1, \ldots, d_k\}$
end function

encoding function **`tcode`**, under ℓ_∞ norm. As k is usually much smaller than the number of points, this solution requires relatively low TCAM space. Still, a standard TCAM can support up to thousands of clusters using this algorithm.

E. Evaluation and Experimental Results

We evaluate our nearest-neighbor algorithms using an image similarity search application (using GIST [69] descriptors), on a real-life image dataset [70]. We then compare the results and performance with prior-art solutions. Our evaluation is based both on experiments with real-life TCAM devices and simulations. Each image in the dataset was encoded as a GIST vector in $\mathbb{R}^{512}$, downsampled to $\mathbb{R}^{40}$ and quantified to $\{0, \ldots, 255\}^{40}$ before performing search. Images were randomly partitioned to a dataset of $21,019$ images a query set of $1,000$ images.

1) Experiment With a TCAM Device: Since there is no evaluation board for such devices, we used a commodity network switch (Quanta T1048-LB9) that contains a TCAM for our experiment (similarly to [7]). This switch has 48 1 Gbps ports, each handling at most $1.5M$ packets per second.

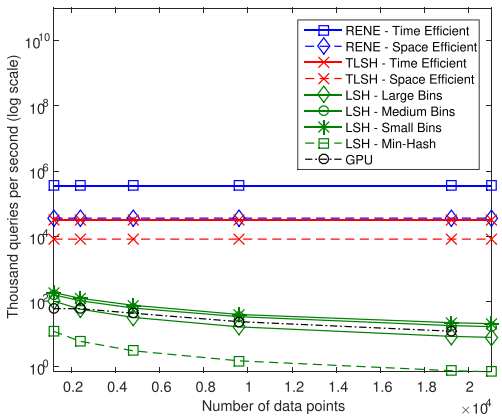

Fig. 9. Throughput comparison of the various algorithms for solving the nearest neighbor problem, as a function of the size of the search database. Throughput of TCAM-based algorithms is simulated based on 360 MHz TCAM throughput. We denote TLSH with one TCAM lookup per r-Near Neighbor Report Problem instance as *time-efficient*, and TLSH with $\log(1/\varepsilon)$ TCAM lookups per instance as *space-efficient*.

TCAM is used for packet classification for OpenFlow 1.3. Using the OpenFlow interface to the switch, we mapped each entry produced by our algorithms to a set of header fields. A commercial traffic generator injected manually crafted packets that contain the queries in their headers, where each query is broken into header fields in the same way TCAM entries are stored.

We verified correctness by counting the number of matches of each TCAM entry. Using one ingress port the switch easily achieved a throughput of $1.5M$ queries per second, which is the upper bound of the link between the traffic generator and the switch (but not of the TCAM). Using 24 ingress ports we achieved throughput of $35.69M$ queries per second (almost $1.5M \times 24$). Hence, the bottleneck was not in the TCAM: If we had more ports we could have reached much higher throughput, as contemporary TCAM devices are capable of query throughput of $360M$ to 1.6 billion queries per second [38], [39].

2) Simulation Results: We compared our results to the results of brute-force exact nearest neighbor (using MATLAB or on GPU [37]), locality sensitive hashing [33], [51] (using implementation from [71]), and TLSH [7]. LSH approximated results in ℓ_2 were comparable to our approximated results in ℓ_2 only when LSH used the most complex hash functions, or when used very large bins. Both options mean longer computation time due to either more complex hash computation or much more distance computations.

Figure 9 presents a comparison of the throughput (queries per second) of CPU implementations of LSH [71], GPU implementation [37], TLSH [7] simulation, and RENÉ simulation. Each line in the figure presents the throughput of a single algorithm/implementation, as a function of the number of data points in the dataset.[4]

For the TCAM simulations we used a software simulation with unbounded memory limits for the TCAM, 360MHz throughput, with 50 cycles latency per query.

[4]LSH implementations were ran on an Intel Core i7 2600 3.4GHz CPU. We used the same dataset and queries for our algorithms, LSH and TLSH. TCAM algorithms used 10 different cube sizes. GPU throughput is as reported in [37] for the closest lower values of n and d.

The required TCAM space for our *space efficient* method is $8M$ bits and for our *time efficient* method with 10 different cube sizes is $80M$ bits, with 440 bits wide entries. These requirements are available in most modern TCAM devices. TLSH requires much higher TCAM capacity and much wider TCAM entries.

3) Geometric Clustering Analysis: In this section we analyze the running time of Algorithm 6. We define i to be the number of iterations until convergence, T_c as the time for completing a single TCAM cycle, T_ℓ as the number of cycles required for TCAM to complete a request,[5] and T_{clear} as the time to clear the whole TCAM. Given the set of points S, the set of edge lengths $\mathcal{H}$ and the number of clusters k, the running time of the algorithm is:

$$\Theta(i \cdot (T_c \cdot (k|\mathcal{H}| + T_\ell \cdot (1 + |S|)) + T_{clear}))$$

4) Comparison to TLSH: As recalled, Shinde *et al.* [7] were the first to suggest using TCAM for nearest neighbor search. They use a ternary variant of the Locality Sensitive Hashing, called TLSH, to provide a probabilistic solution for the nearest neighbor problem. The main advantages of our algorithms over TLSH are that our time-efficient algorithm solves *multiple* instances of the r-NEAR-NEIGHBOR REPORT PROBLEM in a single TCAM lookup, while TLSH requires $|\mathcal{H}|$ lookups (hence the factor of 10 difference in the results presented in Figure 9), and that the TCAM space requirements, and specifically and more importantly TCAM entry width requirement, are lower in at least one order of magnitude than those of TLSH. Furthermore, our algorithms provide deterministic results and are not subject to probabilistic errors, and they allow database hot updates.

VI. CONCLUSION

While the problem of range encoding on TCAM has been deeply investigated over recent years, the proven theoretical limits on the number of bits one must use have diverted researchers to use row expansion. However, row expansion causes exponential increment in the number of TCAM entries. New applications such as SDN implementations for load balancing, security tools, and NFV frameworks use more than a few range fields. Thus, row expansion makes solutions that use it impractical.

In this paper we introduce the sub-problem of *short range encoding*, and we show that the theoretical limits on the number of required bits can be lowered in this situation. We present RENÉ: An encoding scheme for short ranges, and show that it is closer to the lower bound than any other technique. We then present multiple applications that may benefit from such short range encoding, in the area of packet classification. Furthermore, we propose to use TCAM as a co-processor for solving problems outside of the networking field, such as the nearest neighbor problem and its variants, which so far has been known to take very long time to compute. We show that using TCAM, one could solve such problems in much higher rates than previously suggested solutions, and outperform known lower bounds in traditional memory models.

[5]TCAM works in a pipeline, where a request is processed over multiple cycles, but at each cycle a new request may begin.

REFERENCES

[1] W. Jiang, Q. Wang, and V. K. Prasanna, "Beyond TCAMs: An SRAM-based parallel multi-pipeline architecture for terabit IP lookup," in *Proc. INFOCOM*, 2008, pp. 1786–1794.

[2] V. C. Ravikumar and R. Mahapatra, "TCAM architecture for IP lookup using prefix properties," *IEEE Micro*, vol. 24, no. 2, pp. 60–69, Mar./Apr. 2004.

[3] *OpenFlow Switch Specification Version 1.3.2*, Open Netw. Found., Menlo Park, CA, USA, Apr. 2013.

[4] A. Bremler-Barr, D. Hay, and Y. Koral, "CompactDFA: Scalable pattern matching using longest prefix match solutions," *IEEE/ACM Trans. Netw.*, vol. 22, no. 2, pp. 415–428, Apr. 2014.

[5] A. Goel and P. Gupta, "Small subset queries and bloom filters using ternary associative memories, with applications," in *Proc. SIGMETRICS*, 2010, pp. 143–154.

[6] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "DREAM: Dynamic resource allocation for software-defined measurement," in *Proc. SIGCOMM*, 2014, pp. 419–430.

[7] R. Shinde, A. Goel, P. Gupta, and D. Dutta, "Similarity search and locality sensitive hashing using ternary content addressable memories," in *Proc. SIGMOD*, 2010, pp. 375–386.

[8] ETSI. (Oct. 2012). *Network Function Virtualization*. [Online]. Available: http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[9] R. Wang, D. Butnariu, and J. Rexford, "OpenFlow-based server load balancing gone wild," in *Proc. Hot-ICE*, 2011, p. 12.

[10] Radware. (2014). *DefenseFlow—SDN Applications and DDoS Attack Defense*. [Online]. Available: http://www.radware.com/Products/DefenseFlow/

[11] M. S. Seddiki *et al.*, "FlowQoS: QoS for the rest of us," in *Proc. HotSDN*, 2014, pp. 207–208.

[12] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," in *Proc. SIGCOMM*, 1998, pp. 191–202.

[13] E. Spitznagel, D. Taylor, and J. Turner, "Packet classification using extended TCAMs," in *Proc. ICNP*, 2003, pp. 120–131.

[14] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for advanced packet classification with ternary CAMs," in *Proc. SIGCOMM*, 2005, pp. 193–204.

[15] A. Bremler-Barr and D. Hendler, "Space-efficient TCAM-based classification using Gray coding," in *Proc. INFOCOM*, 2007, pp. 1388–1396.

[16] Y.-K. Chang and C.-C. Su, "Efficient TCAM encoding schemes for packet classification using Gray code," in *Proc. GLOBECOM*, 2007, pp. 1834–1839.

[17] H. Che, Z. Wang, K. Zheng, and B. Liu, "DRES: Dynamic range encoding scheme for TCAM coprocessors," *IEEE Trans. Comput.*, vol. 57, no. 7, pp. 902–915, Jul. 2008.

[18] H. Liu, "Efficient mapping of range classifier into ternary-CAM," in *Proc. 10th Symp. High Perform. Interconnects*, 2002, pp. 95–100.

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

[20] A. Bremler-Barr, D. Hay, and D. Hendler, "Layered interval codes for TCAM-based classification," *Comput. Netw.*, vol. 56, no. 13, pp. 3023–3039, Sep. 2012.

[21] O. Rottenstreich, I. Keslassy, A. Hassidim, H. Kaplan, and E. Porat, "On finding an optimal TCAM encoding scheme for packet classification," in *Proc. INFOCOM*, 2013, pp. 2049–2057.

[22] Q. Dong, S. Banerjee, J. Wang, D. Agrawal, and A. Shukla, "Packet classifiers in ternary CAMs can be smaller," in *Proc. SIGMETRICS*, 2006, pp. 311–322.

[23] C. R. Meiners, A. X. Liu, and E. Torng, "TCAM Razor: A systematic approach towards minimizing packet classifiers in TCAMs," in *Proc. ICNP*, Oct. 2007, pp. 266–275.

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

[25] J. S. Beis and D. G. Lowe, "Shape indexing using approximate nearest-neighbour search in high-dimensional spaces," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 1997, pp. 1000–1006.

[26] M. Brown and D. G. Lowe, "Recognising panoramas," in *Proc. ICCV*, vol. 3. 2003, p. 1218.

[27] L. Liang, C. Liu, Y.-Q. Xu, B. Guo, and H.-Y. Shum, "Real-time texture synthesis by patch-based sampling," *ACM Trans. Graph.*, vol. 20, no. 3, pp. 127–150, 2001.

[28] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *Proc. CVPR*, 2007, pp. 1–8.

[29] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. 7th IEEE Int. Conf. Comput. Vis.*, vol. 2. Sep. 1999, pp. 1150–1157.
[30] H. Samet, *Foundations of Multidimensional and Metric Data Structures*. San Mateo, CA, USA: Morgan Kaufmann, 2006.
[31] J. L. Bentley, "Multidimensional divide-and-conquer," *Commun. ACM*, vol. 23, no. 4, pp. 214–229, 1980.
[32] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," in *Proc. ACM Sigmod Rec.*, 1990, vol. 19. no. 2, pp. 322–331.
[33] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Commun. ACM*, vol. 51, no. 1, pp. 117–122, Jan. 2008.
[34] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *Proc. VLDB*, 1998, pp. 194–205.
[35] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *Proc. VISAPP*, 2009, pp. 331–340.
[36] F. Gray, "Pulse code communication," U.S. Patent 2 632 058 A, Mar. 17, 1953.
[37] V. Garcia, É. Debreuve, F. Nielsen, and M. Barlaud, "K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching," in *Proc. ICIP*, 2010, pp. 3757–3760.
[38] Renesas Electronics America Inc. *20 Mbit QUAD-Search Content Addressable Memory*. Accessed: Jul. 11, 2016. [Online]. Available: http://www.renesas.com/products/memory/TCAM/index.jsp
[39] C. Inc. (2014). *NEURON Search Processors*. [Online]. Available: http://www.cavium.com/processor_NEURON-Search.html
[40] I. Corp. (2011). *Intel Xeon Processor E7-4870*. [Online]. Available: http://ark.intel.com/products/53579/
[41] Nvidia. (Nov. 2014). *Tesla K80 GPU Accelerator*. [Online]. Available: http://international.download.nvidia.com/pdf/kepler/BD-07317-001_v04.pdf
[42] O. Rottenstreich and I. Keslassy, "Worst-case TCAM rule expansion," in *Proc. INFOCOM*, 2010, pp. 1–5.
[43] O. Rottenstreich and I. Keslassy, "On the code length of TCAM coding schemes," in *Proc. ISIT*, 2010, pp. 1908–1912.
[44] K. Kogan, S. I. Nikolenko, O. Rottenstreich, W. Culhane, and P. Eugster, "Exploiting order independence for scalable and expressive packet classification," *IEEE/ACM Trans. Netw.*, vol. 24, no. 2, pp. 1251–1264, Apr. 2016.
[45] K. Kogan, S. I. Nikolenko, P. Eugster, A. Shalimov, and O. Rottenstreich, "Efficient FIB representations on distributed platforms," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3309–3322, Dec. 2017.
[46] K. Kogan, S. Nikolenko, O. Rottenstreich, W. Culhane, and P. Eugster, "SAX-PAC (scalable and expressive packet classification)," in *Proc. SIGCOMM*, 2014, pp. 15–26.
[47] C. R. Meiners, A. X. Liu, and E. Torng, "Bit Weaving: A non-prefix approach to compressing packet classifiers in TCAMs," in *Proc. ICNP*, 2009, pp. 93–102.
[48] K. Zheng, H. Che, Z. Wang, B. Liu, and X. Zhang, "DPPC-RE: TCAM-based distributed parallel packet classification with range encoding," *IEEE Trans. Comput.*, vol. 55, no. 8, pp. 947–961, Aug. 2006.
[49] C. R. Meiners, A. X. Liu, E. Torng, and J. Patel, "Split: Optimizing space, power, and throughput for TCAM-based classification," in *Proc. ANCS*, 2011, pp. 200–210.
[50] Y. Afek, A. Bremler-Barr, and L. Schiff, "Recursive design of hardware priority queues," in *Proc. SPAA*, 2013, pp. 23–32.
[51] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. STOC*, 1998, pp. 604–613.
[52] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.
[53] S. Dasgupta and Y. Freund, "Random projection trees and low dimensional manifolds," in *Proc. STOC*, 2008, pp. 537–546.
[54] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proc. VLDB*, 1999, pp. 518–529.
[55] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe LSH: Efficient indexing for high-dimensional similarity search," in *Proc. VLDB*, 2007, pp. 950–961.
[56] Y. Tao, K. Yi, C. Sheng, and P. Kalnis, "Quality and efficiency in high dimensional nearest neighbor search," in *Proc. SIGMOD*, 2009, pp. 563–576.
[57] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using GPU," in *Proc. IEEE CVPR*, Jun. 2008, pp. 1–6, doi: 10.1109/CVPRW.2008.4563100.
[58] S. Liang, C. Wang, Y. Liu, and L. Jian, "CUKNN: A parallel implementation of K-nearest neighbor on CUDA-enabled GPU," in *Proc. YC-ICT*, 2009, pp. 415–418.
[59] J. Pan and D. Manocha, "Fast GPU-based locality sensitive hashing for K-nearest neighbor computation," in *Proc. GIS*, 2011, pp. 211–220.
[60] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.
[61] Radware. (Oct. 2014). *Tsunami SYN Flood Attack—A New Trend in DDoS Attacks?* [Online]. Available: http://blog.radware.com/security/2014/10/tsunami-syn-flood-attack/
[62] G. Pazi, A. Bremler-Bar, R. Rivlin, and D. Touitou, "Protecting against distributed denial of service attacks," U.S. Patent 2003 0110274 A1. Jun. 12, 2003. [Online]. Available: http://www.google.com/patents/US20030110274
[63] C. Jin, H. Wang, and K. G. Shin, "Hop-count filtering: An effective defense against spoofed DDoS traffic," in *Proc. CCS*, 2003, pp. 30–41.
[64] A. Gupta *et al.*, "SDX: A software defined internet exchange," in *Proc. SIGCOMM*, 2014, pp. 551–562.
[65] *AS Names—CIDR Report*. Accessed: Jan. 29, 2018. [Online]. Available: http://www.cidr-report.org/as2.0/autnums.html
[66] (2014). *Ryu SDN Controller*. [Online]. Available: http://osrg.github.io/ryu/
[67] A. Borodin, R. Ostrovsky, and Y. Rabani, "Lower bounds for high dimensional nearest neighbor search and related problems," in *Proc. STOC*, 1999, pp. 312–321.
[68] S. P. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. IT-28, no. 2, pp. 129–137, Mar. 1982.
[69] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *Int. J. Comput. Vis.*, vol. 42, no. 3, pp. 145–175, 2001.
[70] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman, "LabelMe: A database and Web-based tool for image annotation," *Int. J. Comput. Vis.*, vol. 77, nos. 1–3, pp. 157–173, 2008.
[71] M. Aly, M. Munich, and P. Perona, "Indexing in large scale image collections: Scaling properties and benchmark," in *Proc. WACV*, 2011, pp. 418–425.

Anat Bremler-Barr received the Ph.D. degree (Hons.) in computer science from Tel Aviv University, Tel Aviv, Israel. In 2001, she co-founded and was the Chief Scientist of Riverhead Networks, Inc. (acquired by Cisco Systems in 2004), which provided systems to protect from denial-of-service attacks. She then joined the Interdisciplinary Center Herzliya, Herzliya, Israel, in 2004, where she co-founded (with Prof. D. Hay) the DEEPNESS Laboratory (funded by an ERC starting grant) that focuses on designing deep packet inspection for next-generation network devices. She is currently an Associate Professor with the School of Computer Science, Interdisciplinary Center Herzliya. Her research interests include computer networks and network security.

Yotam Harchol received the Ph.D. degree from the Hebrew University of Jerusalem, Israel, in 2017. Before joining the University of California at Berkeley (UC Berkeley), Berkeley, CA, USA, he was a Post-Doctoral Researcher with VMware Research. He is currently a Post-Doctoral Scholar (with Prof. S. Shenker) with the Department of Electrical Engineering and Computer Science, UC Berkeley. His research interests include software-defined networking, network security, and high-performance algorithms for network middleboxes. He was the recipient of the Intel Award for Graduate Students in 2010, the Hammer Fellowship for Master Students in 2009, and the Chais Scholarship for Social Leadership in 2007.

David Hay received the B.A. degree *(summa cum laude)* and the Ph.D. degree in computer science from the Technion Israel Institute of Technology, Haifa, Israel, in 2001 and 2007, respectively. In addition, he was with: the IBM Haifa Research Laboratories, Haifa; Cisco Systems, San Jose, CA, USA; the Electronic Department, Politecnico di Torino, Turin, Italy; and the Electrical Engineering Department with Columbia University, New York, NY, USA. In 2010, he co-founded (with Prof. A. Bremler-Barr) the DEEPNESS laboratory, focusing on deep packet inspection in next-generation network devices. He is currently an Associate Professor with The Rachel and Selim Benin School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem, Israel. His research interests include computer networks in particular, network algorithmics, packet classification, deep packet inspection, network survivability, and software-defined networking.

Yacov Hel-Or received the Ph.D. degree in computer science from the The Hebrew University of Jerusalem. He was a Visiting Scientist with Google Inc. and a Research Scientist with Amazon from 2016 to 2018. Prior to this, he held post-doctoral positions with the Weizmann Institute of Science and the NASA Ames Research Center, California. He is currently a Faculty Member with the School of Computer Science, Interdisciplinary Center Herzliya, Israel. His main research interests include computer vision, image processing, and computer graphics.