Efficient Gray-Code-Based Range Encoding Schemes for Packet Classification in TCAM

Yeim-Kuan Chang, Cheng-Chien Su, Yung-Chieh Lin, and Sun-Yuan Hsieh

***Abstract*—An efficient ternary content addressable memory (TCAM) encoding scheme using a binary reflected Gray code (BRGC) and the concept of elementary intervals is presented for efficiently storing arbitrary ranges in TCAM. The proposed *layered BRGC* range encoding scheme (L-BRGC) groups ranges into BRGC range sets in which each range can be encoded into a single ternary vector. The results of experiments performed on real-life and synthesized rule tables show that L-BRGC consumes less TCAM than all the existing range encoding schemes for all rule tables, except that the direct conversion scheme (EIGC) using elementary intervals and BRGC codes performs best for a small real-life ACL rule table.**

***Index Terms*—Elementary intervals, Gray code, packet classification, ternary content addressable memory (TCAM).**

I. Introduction

PACKET classification in Internet routers is needed in a variety of Internet applications, such as quality of service, security, and multimedia communications. Packet classification performs searches over a set of filters (i.e., rules) using multiple fields of the packet as the search key. Filters define a flow or a set of flows by specifying the match conditions on the packet header fields. Typically, five packet header fields are used in filters: source and destination IP address prefixes, source and destination transport port ranges, and a single value or wildcard protocol numbers. Routers resolve the flow for a given packet by searching the set of filters for the subset of matching filters against the 5-field values in the packet header.

Packet classification can be implemented in either software or hardware [6], [9]. Software solutions have the disadvantages of nondeterministic run times and large memory requirements. It is a challenge to have a software implementation that meets the gigabit search speed requirement. In this paper, the focus is restricted to hardware architectures using ternary content addressable memory (TCAM). TCAM-based packet classification is currently a popular hardware solution because: 1) industry vendors are providing cheaper and faster TCAM products; 2) TCAM architecture is easy to understand and simple to manage in updating TCAM entries; and 3) TCAM's performance is deterministic (i.e., it takes the same number of cycles to complete a search). Despite these advantages, however, TCAMs suffer from four primary deficiencies: 1) high cost per bit as compared to other memory technologies; 2) high power consumption; 3) limited scalability to long input keys; and 4) inefficiency in storing ranges. The cost-to-density-ratio of TCAM has improved dramatically in recent years. The circuit designs that reduce the matchline voltage swing, switching activity, and active matchline capacitance [16] and rule table partitioning schemes are feasible techniques to reduce TCAM power consumption. Thus, our focus is on designing range encoding schemes for efficiently storing ranges in TCAM.

The direct range-to-prefix conversion is a traditional database-independent scheme. This scheme individually converts each range into multiple prefixes. In the worst case, a 16-bit range may be converted to 30 prefixes, and thus a single rule consisting of two 16-bit range fields may require 900 2-D rules in prefix format by cross-producting the prefixes converted from the two range fields. SRGE [2] and DIRPE [11] are two other database-independent encoding schemes. The primary advantage of database-independent schemes is their fast update operations because updating one rule does not affect the others. However, they suffer from large TCAM consumption.

The database-dependent encoding schemes, which reduce TCAM consumption by exploiting the dependency among rules, are hence discussed. The encoding process of one field is independent of the other fields. Each field value is individually converted into one or more *field ternary vectors*. The field ternary vectors of all field values in a rule are cross-producted to obtain one or more *rule ternary vectors* that are finally stored in the TCAM. The field values in input packet headers must be translated into intermediate results, which in turn will be used as search keys in TCAM. The address-to-intermediate result translation can be implemented by a precomputed direct map in fast SRAM or by specially designed hardware. The proposed *layered binary reflected Gray code* range encoding scheme (L-BRGC) groups ranges into BRGC range sets that can be encoded by a minimal number of ternary vectors. An incremental layer-based range insertion scheme is likewise developed so that each range can be converted into a single ternary vector.

The rest of this paper is organized as follows. Problem statement and preliminaries are given in Section II, and related work is discussed in Section III. The proposed scheme and performance results are given in Sections IV and V, respectively. The paper concludes in Section VI.

II. Problem Statement and Preliminaries

Before we introduce the terminology we use throughout the paper, the formal problem statement is given first as follows.

Manuscript received October 21, 2010; revised March 15, 2012; accepted September 05, 2012; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor P. Crowley. Date of publication October 22, 2012; date of current version August 14, 2013.

The authors are with the Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan 701, Taiwan (e-mail: ykchang@mail.ncku.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2012.2220566

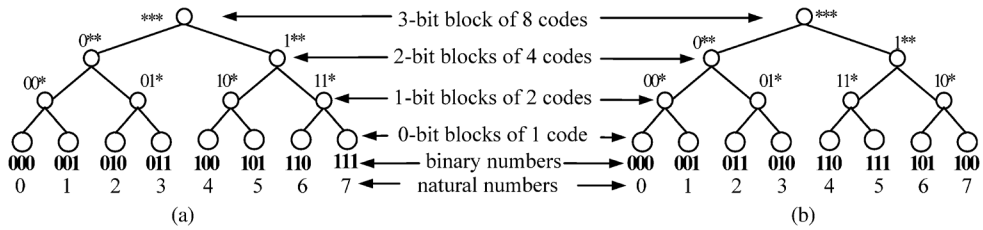


Fig. 1. (a) Buddy and (b) binary reflected Gray codes in the 3-bit address space.

Given: A set of integer ranges R defined on the interval $[0 \ldots N-1]$ and an integer n.

Find: A function $\boldsymbol{k}(r)$ that maps range $r \in R$ to a set of ternary vectors of length n and a function $\boldsymbol{q}$ that maps each integer in $[0 \ldots N-1]$ to an n-bit vector, such that $\boldsymbol{q}(i)$ matches $\boldsymbol{k}(r)$ for any range r in R that includes i (and does not match ranges that do not include i).

Example 1: Given ranges $A = [1,5]$ and $B = [3,6]$ on interval [0...7], two solutions may be the following:

1) $\boldsymbol{k}(A) = \{001, 01^*, 10^*\}$, $\boldsymbol{k}(B) = \{011, 10^*, 110\}$, $\boldsymbol{q}(0) = 000$, $\boldsymbol{q}(1) = 001$, $\boldsymbol{q}(2) = 010$, $\boldsymbol{q}(3) = 011$, $\boldsymbol{q}(4) = 100$, $\boldsymbol{q}(5) = 101$, $\boldsymbol{q}(6) = 110$, $\boldsymbol{q}(7) = 111$;
2) $\boldsymbol{k}(A) = \{1^*\}$, $\boldsymbol{k}(B) = \{^*1\}$, $\boldsymbol{q}(i) = 00$ for $i \in \{0,7\}$, $\boldsymbol{q}(i) = 10$ for $i \in \{1,2\}$, $\boldsymbol{q}(i) = 11$ for $i \in \{3,4,5\}$, and $\boldsymbol{q}(i) = 01$ for $i \in \{6\}$.

The first solution uses three 3-bit ternary vectors for each range. Hence, 18 TCAM bits are required. The second one uses one 2-bit ternary vector for each range, and thus 4 TCAM bits are required. Obviously, the second solution is better. In this paper, we focus on finding functions $\boldsymbol{k}(r)$ and $\boldsymbol{q}(i)$ such that the number of TCAM bits required is minimal.

A W-bit prefix of length $W-h$, $b_{W-1} \ldots b_h * \ldots *$ or simply $b_{W-1} \ldots b_h *$, covers addresses $b_{W-1} \ldots b_h 0 \ldots 0$ to $b_{W-1} \ldots b_h 1 \ldots 1$, where $b_j = 0$ or 1 for $W-1 \geq j \geq h$ and $h*$'s (wildcards or "don't care" bits) are appended. If we allow wildcards to be at arbitrary positions of bit patterns, we call them *ternary vectors* to be distinguishable from prefixes. A W-bit range $R = [L,U]$ covers addresses from L to U. For five-dimensional filters, $W = 16$ for source and destination port range fields, $W = 32$ (128) for source and destination IPv4 (IPv6) address prefix fields, and $W = 8$ for the protocol field. Some definitions for two ranges $A = [u,v]$ *and* $B = [x,y]$ are initially given as follows.

Definition 1:

1) A and B are disjoint if $v < x$ or $y < u$, i.e., $|A \cap B| = 0$. One special case is that A and B are adjacent if $x = v+1$ or $u = y+1$.
2) A and B are nested if ($x \leq u$ and $v \leq y$) or ($u \leq x$ and $y \leq v$), i.e., one is contained within the other *or* $|A \cap B| = \min\{|A|, |B|\}$. One special case is that A is weak-nested by B if ($u = x$ and $v \leq y$) or ($x \leq u$ and $v = y$).
3) A and B are intersecting if ($u < x \leq v < y$) or ($x < u \leq y < v$), i.e., they are neither disjoint nor nested, *or* $0 < |A \cap B| < \min\{|A|, |B|\}$.

Let A_n be a sequence of n-bit binary vectors for $n \geq 1$. The reverse or reflected sequence of A is denoted as A^{r}. We define bA_{n-1} as the sequence of n-bit binary vectors after prepending a bit b to all binary vectors in A_{n-1}. For example, if $A_2 = (00, 01, 11, 10)$, $(A_2)^{\mathrm{r}} = (10, 11, 01, 00)$, and $0A_2 = (\underline{0}00, \underline{0}01, \underline{0}11, \underline{0}10)$.

A. Buddy Code (BC)

The *Buddy code* (BC) B_n is defined recursively as $B_1 = (0,1)$ and $B_k = (0B_{k-1}, 1B_{k-1})$ for $k = 2$ to n. B_n follows the sequence of natural numbers 0 to $2^n - 1$. Given $0 \leq h \leq n$, B_n can be divided into 2^{n-h} blocks (called *h-bit block*) of consecutive 2^h codes such that all the 2^h codes in a block can be merged into a prefix. The prefix $b_{n-1} \ldots b_h *$ of length $n-h$ corresponds to the h-bit block of codes $b_{n-1} \ldots b_h 0 \ldots 0$ to $b_{n-1} \ldots b_h 1 \ldots 1$. Each prefix $A = b_{n-1} \ldots b_{h+1} b_h *$ has only one Buddy prefix $B = b_{n-1} \cdots b_{h+1} \overline{b_h} *$ such that A and B can be combined into $b_{n-1} \ldots b_{h+1} *$. Fig. 1(a) shows a 3-bit Buddy code example.

B. BRGC

The BRGC G_n, for $1 \leq i \leq n$, is defined recursively as $G_1 = (0,1)$ and $G_k = (0G_{k-1}, 1(G_{k-1})^r)$ for $k = 2$ to n. For example, $G_2 = (00, 01, 11, 10)$ and $G_3 = (000, 001, 011, 010, 110, 111, 101, 100)$ as shown in Fig. 1(b). Gray codes in G_3 follow the order of the sequence of binary numbers: 000, 001, 011, 010, 110, 111, 101, 100. In general, we can convert a BRGC code to a binary number directly as follows.

BC to BRGC ($b2g$) Conversion:: Let $b_{n-1} \ldots b_0$ and $g_{n-1} \ldots g_0$ be the bitmaps of a BC code k and its corresponding BRGC code $b2g(k)$, respectively. The conversion is done by setting $g_{n-1} = b_{n-1}$ and $g_i = b_{i+1}$ XOR b_i, for $i = 0$ to $n-2$.

Clearly, a BC range $[L,U]$ covers consecutive addresses L to U because $L \leq U$. However, this may cause confusion when we write $[L,U]$ as a BRGC range. Therefore, the BRGC range $[L,U]$ consists of addresses $b2g(L), b2g(L+1), \ldots, b2g(U)$ written as $b2g([L,U])$. For example, the BC range [2, 4] consists of binary numbers 010, 011, and 100, while the BRGC range [2, 4] consists of binary numbers 011, 010, and 110 (i.e., $b2g(2)$, $b2g(3)$, and $b2g(4)$).

Similar to BC, G_n can be divided into 2^{n-h} h-bit blocks of 2^h BRGC codes that can be merged into ternary vectors. BRGC is better than BC in that any two neighboring h-bit blocks can be combined into a single ternary vector. For instance, the BC range [2, 5] can be merged into two ternary vectors 01^* and 10^*, while BRGC is better because the BRGC range [2, 5] corresponds to binary numbers 011, 010, 110, and 111 that can be merged into $^*1^*$.

C. Elementary Interval (EI)

Let G be a set of original ranges in which the *default range* covers the whole address space. The set of EIs [5], constructed from G, is $E = \{E[i] | E[i] = [l[i], u[i]]$ for $i = 0$ to $|E|-1\}$, where $|E|$ is the number of EIs in E. E must satisfy four conditions: 1) $l[0] = 0$ and $u[k-1] = 2^W - 1$; 2) $u[i] = l[i+1] - 1$

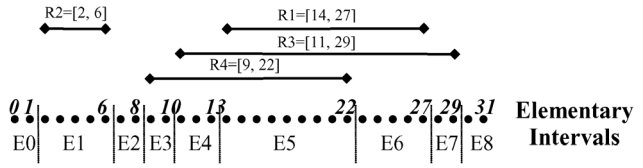


Fig. 2. Example for elementary intervals.

for $i = 0$ to $k - 2$; 3) all the addresses in $E[i]$ are covered by the same subset of G, denoted by $G[i]$; and 4) $G[i] \neq G[i+1]$. For example, from the four 5-bit ranges in Fig. 2, $E = \{[0,1],[2,6],[7,8],[9,10],[11,13],[14,22],[23,27],[28,29],[30,31]\}$. Six *valid* EIs are covered by at least one range other than the default range. The intervals covered only by the default range [0, 31] are *default* EIs.

III. Related Work

A range encoding scheme is either database-independent or -dependent based on whether the process of encoding a range is independent of other ranges in the database. Database-independent schemes are usually fast for update, but they tend to use more TCAM than the database-dependent ones.

A. Basic Elementary Interval Encoding Scheme (BEIE)

Based on the concept of elementary intervals, each elementary interval is given a unique identifier (code), and each range can then be encoded by the identifiers of the EIs covered by the range. The ranges encoded by the identifiers of the EIs are called *primitive ranges* [14]. For example, the BC code assignment can be done by $E_i = i$ for $i = 0$ to 8 in Fig. 2. Thus, $\mathrm{R1}$ can be encoded as the primitive range [5, 6] because $\mathrm{R1}$ covers EI_5 and EI_6. Similarly, $\mathrm{R2}, \mathrm{R3}$, and $\mathrm{R4}$ are encoded as [1, 1], [4, 7], and [3, 5], respectively. Since there are nine EIs, 4 bits are sufficient. By using direct range-to-prefix conversion, $\mathrm{R1}$ is converted into two prefixes 0101 and 0110; $\mathrm{R2}$ is converted to 0001; $\mathrm{R3}$ is converted to 01^{**}; and $\mathrm{R4}$ is converted to 0011 and 010^{*}. Totally, six 4-bit prefixes are needed. In comparison, the original direct range-to-prefix conversion generates 17 5-bit prefixes for the same set of ranges.

B. Parallel Packet Classification Encoding Scheme

The parallel packet classification (PPC) encoding scheme [14] is also based on the concept of elementary intervals. The only difference between PPC and BEIE is that PPC only encodes the valid elementary intervals. PPC divides the primitive ranges into *layers*. Depending on encoding style, code assignments in one layer may be: I) independent of; II) partially dependent on; or III) completely dependent on code assignment in other layers.

The *bitmap intersection* scheme [10] that uses one layer per range is the predecessor of PPC. Each EI is associated with an n-bit identifier as follows. The ith bit is set to 1 if the EI is covered by the range in the ith layer; otherwise, it is set to 0. The range in layer i is assigned an n-bit ternary vector called *match condition* whose ith bit is set to 1, and other bits are set to *.

PPC style-I improves bitmap intersection by allowing more than one disjoint range in a layer. Consequently, the number of layers required will be equal to the maximum number of ranges that all overlap one another (i.e., all cover a common address). The layer consisting of L ranges needs $\lceil \log(L+1) \rceil$ bits. PPC style-II reduces the number of bits needed for each layer by inspecting the code dependencies among layers. Two ranges at the same layer can be assigned a common identifier if one is nested by a range at another layer and the other is not nested by the same range. PPC style-III further reduces the number of bits by grouping many layers into a larger one. However, a primitive range may be represented by more than one match condition.

Recently, a scheme called LIC [3] similar to PPC style-I is proposed to split ranges between multiple layers containing mutually disjoint ranges. Its goal is to find a way to encode the ranges in all layers such that the code size is minimal.

C. Direct Range-to-Prefix Conversion

The most straightforward database-independent scheme is the direct range-to-prefix conversion based on Buddy code. Each range is converted into a number of prefixes. An efficient algorithm can be found in [4]. In the worst case, range $[1, 2^W - 2]$ in the W-bit address space is split into $2W - 2$ prefixes.

D. Ternary Vector Compaction (Boolean Expression)

Although the technique of combining prefixes into ternary vectors is useful in some cases, it is impossible to combine two prefixes of the same length into a single ternary vector when they come from the set of prefixes converted from a range by the direct range-to-prefix conversion. For example, the 4-bit range [1, 14] is converted into 0001, 001^{*}, 01^{**}, 10^{**}, 110^{*}, 1110. None of the pairs, 0001/1110, $001^{*}/110^{*}$, and $01^{**}/10^{**}$, can be combined into a single ternary vector. However, we can view range [1, 14] as 14 singleton addresses and combine them into four ternary vectors, $1^{**}0$, $^{**}01$, $^{*}01^{*}$, and 01^{**}, in which the addresses 0101 and 1010 are used twice, solved by the Boolean expression minimization technique. In the Boolean expression minimization, each bit of address space is mapped onto a unique Boolean variable. A range is expressed as a sum of minterms, each representing an address in the range. A Karnaugh map (K-map) technique can then be used to find the minimum sum of products. For example, based on K-map, the minimized Boolean expression of the 4-bit range [1, 14] is $v_1\overline{v_4}+\overline{v_3}v_4+\overline{v_2}v_3+\overline{v_1}v_2$, which is equivalent to $1^{**}0 +^{**} 01 +^{*} 01^{*} + 01^{**}$. For BRGC range [1, 14] that consists of all 4-bit addresses except 0000 and 1000 (i.e., $b2g([1,14])$), the minimized Boolean expression of addresses $b2g([1,14])$ is $v_2 + v_3 + v_4$, which is equivalent to $^{*}1^{**} +^{**} 1^{*} +^{***} 1$.

Notice that Boolean expression minimization is NP-complete. Espresso-II [1], [13] is a fast heuristic algorithm that can be used in practice. The linear time algorithm [17] can also be used.

E. Direct Conversion Using EIs and BRGC (EIGC)

As described in Section II, BRGC has the advantage over the BC in that an h-bit block of BRGC codes can be combined with one of its two neighboring h-bit blocks into an $(h+1)$-bit block (a ternary vector). Thus, a database-dependent scheme called EIGC based on elementary intervals and BRGC can be developed. For example, the EIs constructed from $\mathrm{R1}, \mathrm{R2}, \mathrm{R3}$, and $\mathrm{R4}$ in Fig. 2 are numbered with codes 0–8 in the BRGC sequence from left to right. Since there are nine codes, a 4-bit code space is needed. As a result, $\mathrm{R1}, \mathrm{R2}$ and $\mathrm{R3}$ are encoded

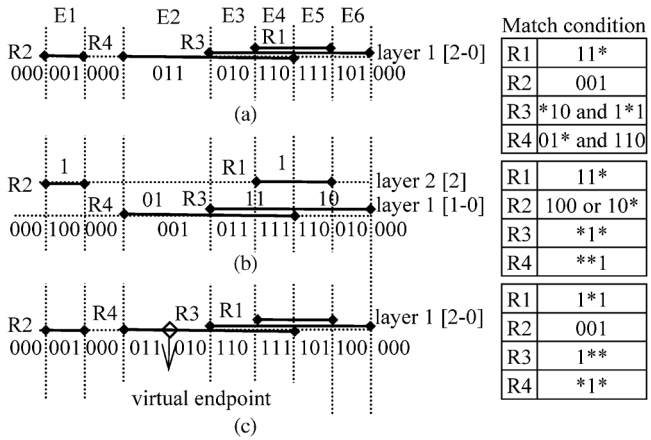


Fig. 3. PPC and BRGC encoding shemes. (a) RFC-like encoding + BRGC; (b) PPC style-II and III + BRGC; (c) PPC style-III + BRGC + virtual endpoint.

by 01*1, 0001, and 01**, respectively, and R4 is encoded by 0010 and 011*.

One minor improvement to reduce the code space is that all the default EIs can be given the same code. Assume there are n valid EIs, denoted by $VE_1, \ldots, VE_n$ from left to right. We simply assign $\lceil \log_2(n+1) \rceil$-bit code, $b2g(i)$, to E_i for $i = 1$ to n. Any unused code can be assigned to the default EIs. Consider the same example in Fig. 2. Three default EIs—E0, E2, and E8—are assigned the code 0. The code assignments for the six valid EIs are $VE_1 = \mathrm{E1} = 001$, $VE_2 = \mathrm{E3} = 011$, $VI_3 = \mathrm{E4} = 010$, $VI_4 = \mathrm{E5} = 110$, $VI_5 = \mathrm{E6} = 111$, and $VI_6 = \mathrm{E7} = 101$. Three bits are thus sufficient for the code space. As a result, range R1 is represented as 11*. Similarly, R2 is represented as 001, R3 as *10 and 1*1, and R4 as 01* and 110, as shown in Fig. 3(a). EIGC may encode each range into more than one ternary vector.

F. Database-Independent Pre-Encoding (DIRPE)

Another database-independent pre-encoding (DIRPE) is recently proposed in [11]. The basic DIRPE scheme uses $2^W - 1$ ternary values to encode each W-bit range into a single ternary vector. However, the number of required ternary values $(2^W - 1)$ is extraordinarily larger than W. To reduce the number of required ternary values, a hierarchical scheme that divides the address space into chunks was proposed. One drawback is that more than one ternary vector may be needed to represent a range. In general, if the number of chunks is set to l, the worst-case number of ternary vectors needed for a range is $2l - 1$.

G. Hybrid Scheme

The hybrid schemes divide ranges into two groups. Each group uses a different encoding scheme. If the ranges are already in the format of prefix, they need not be encoded and are put into one group. Some encoding schemes can be applied to the other ranges. One way to handle these two groups is to use two separate TCAM search engines, which leave no interference between these two groups. However, an additional operation is needed to process the results generated from these two TCAM search engines. Another way to handle these two groups is to use only one TCAM search engine. However, wider TCAM entries will have to be used to store the concatenated match conditions of both groups. Likewise, the search keys will also become wider. One example is the range mapping scheme proposed by Liu [12]. In Liu's scheme, each frequently occurring range in one group is encoded by one extra bit based on the bitmap intersection scheme [10], and the infrequently occurring ranges in the other group are encoded by the direct range-to-prefix conversion scheme. Another example used in the Dynamic Range Encoding Scheme (DRES) proposed recently can be found in [7].

H. Semantically Equivalent Rule Set

These schemes [8], [15], [20] reduce the number of TCAM entries by identifying semantically equivalent rule sets. A number of effective techniques such as trimming rules, expanding rules, merging rules, and adding rules are developed. However, these techniques achieve higher compression ratios only in the cases when rules share the same actions.

IV. Proposed Range Encoding Scheme

In this section, a scheme called *layered BRGC range encoding scheme* (L-BRGC) is proposed. L-BRGC groups the ranges in a range set into layers such that the maximal intersecting range sets found in each layer can be converted to BRGC range sets with or without adding virtual endpoints, where virtual endpoint, maximal intersecting range set, and BRGC range set will be defined later. Then, each BRGC range set can be easily encoded based on the proposed BRGC code assignment so that every range can be represented by only one ternary vector. If all BRGC range sets contain only one range, L-BRGC is similar to PPC [14]. For easy understanding of the proposed L-BRGC, how to insert a range is briefly outlined as follows.

1) Insert the range into layer i (starting from the first layer).
2) Construct all the maximal intersecting range sets in layer i, and for each maximal intersecting range set, S, the following operations are performed.
   A) If S is a BRGC range set, go to step C; else go to step B.
   B) Add a number of virtual endpoints into the elementary intervals of S to make S a BRGC range set. If this operation fails, go to step 3; else go to step C.
   C) Perform BRGC code assignments for every range in S such that each range can be represented by a single ternary vector. If the assignment process fails, go to step 3.
3) Increment i by one. If layer i exists, go to step 2. Otherwise, create a new layer and insert the range in it.

L-BRGC reduces the complexity of the prefix or ternary vector encoding process by changing the size of the original address space to the code size of EI identifiers, $\lceil \log_2 |\mathrm{E}| \rceil$ bits, where $|\mathrm{E}|$ is the number of elementary intervals constructed from the set of original ranges. L-BRGC represents each range by only one ternary vector. Overlapped ranges can be put in the same layer as long as they can all be represented by unique ternary vectors. A systematic approach will subsequently be presented. The idea is illustrated by using the ranges in Fig. 2, and the results are shown in Fig. 3(b). Layer 1 contains ranges R3 and R4, which divide the address space into three valid elementary intervals that can be encoded with 2-bit BRGC codes 01, 11, and 10, respectively. Layer 2 contains ranges R1 and R2 and needs only one bit based on PPC style-II. As a

result, only three bits for the code space are needed. The code assignments of all the seven valid elementary intervals are the same as PPC style-II, but determining the match conditions of the ranges needs a little elaboration. R3 is encoded by 1^* in layer 1, and thus its overall match condition is $\underline{*}1^*$ by setting bit 2 to $*$, where bit 2 is underlined. Similarly, R4 is represented as $^{**}1$. By using the reasoning in PPC style-II, R1 and R2 can be assigned the same code "1" in layer 2. As a result, R1 and R2 can be distinguished by bit 1. Thus, R1 is represented as 11^*, and R2 can be represented as 100 or 10^* by giving the unused code 101 to R2.

Fig. 3(c) shows another way of assigning BRGC identifiers to the valid elementary intervals covered by the four ranges in Fig. 2. All the ranges are put into a single layer. Range R3, which covers four elementary intervals, is taken as an example. Any of the 2-bit blocks of BRGC codes can be assigned to these four intervals. Fig. 3(c) shows that codes 110, 111, 101, 100 are assigned to E3, E4, E5, and E6, respectively. As a result, R3 and R1 can be represented by 1^{**} and 1^*1, respectively. Since R4 covers three elementary intervals—E2, E3, and E4—it is not possible to combine the codes of these three elementary intervals into one ternary vector by any existing encoding scheme. Two codes can be assigned to E2 and make R4 consist of four codes. This is done by assigning the 1-bit block of codes 011 and 010 to E2 because it is the neighboring block of 110 and 111 assigned to E3 and E4. As a result, R4 can be represented by a single ternary vector $^*1^*$. Finally, the elementary interval E1 covered by R2 is assigned code 001, and default intervals are assigned code 000. Since two codes are assigned to E2, we say that a *virtual endpoint* is thus inserted in E2.

Based on the BRGC assignments done in Fig. 3(c), the four ranges are divided into two disjoint range sets: One consists of R2, and the other consists of R1, R3, and R4. These two range sets are denoted by *intersecting range sets*, the formal definition of which is as follows.

Definition 2: An intersecting range set is either a single range or a set of ranges in which any range must intersect at least one of the other ranges. Let S be an intersecting range set found in a set of ranges G. S is a maximal intersecting range set in G if there does not exist any other intersecting range set T in G such that $S \subset T$.

Definition 3: Let MIRS_1 and MIRS_2 be two maximal intersecting range sets found in a range set G. MIRS_1 and MIRS_2 are said to be disjoint if any range in MIRS_1 is disjoint from any range in MIRS_2. MIRS_1 is said to be nested by MIRS_2 if all the ranges in MIRS_1 are completely contained in one of the elementary intervals constructed from MIRS_2.

Consider the ranges R1, R2, R3, and R4 in Fig. 2 and two additional ranges $\mathrm{R5} = [14, 17]$ and $\mathrm{R6} = [16, 19]$. Ranges R5 and R6 are covered by R1, R3, and R4 simultaneously. In addition to the maximal intersecting range set $\mathrm{MIRS}_1 = \{\mathrm{R1}, \mathrm{R3}, \mathrm{R4}\}$, the second maximal intersecting range set $\mathrm{MIRS}_2 = \{\mathrm{R5}, \mathrm{R6}\}$ is formed because neither R5 nor R6 intersects with any range in MIRS_1. R2 is disjoint from all other ranges, and thus $\mathrm{MIRS}_3 = \{\mathrm{R2}\}$ is the third maximal intersecting range set in G. Notice that any two maximal intersecting range sets found in a range set can be either disjoint or nested, and they cannot be intersecting.

We denote the *left* and *right* EIs of A by $\mathrm{l}(A)$ and $\mathrm{r}(A)$, respectively. We say that range A is *half-cut* by range B in G if

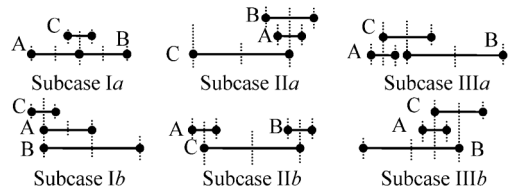


Fig. 4. Proof of Lemma 1.

$|A \cap B| = |A|/2$, and thus $|A| \leq |B|$. Two ranges A and B in G are said to be *half-cutting* if *one* is *half-cut* by the other. Specifically, we say that A is *right half-cut* by B if the addresses covered by $A \cap B$ are contained in the right half of B and A is *left half-cut* by B if the addresses covered by $A \cap B$ are contained in the left half of B.

Definition 4: A BRGC range set is defined to be the intersecting range set that satisfies the following constraints: 1) The number of elementary intervals contained in each range must be a power of two. 2) For any two ranges A and B that are intersected in the set, they must be half-cutting.

The first constraint is required for a range to be represented by one ternary vector because the only possible case to merge n single-value addresses into a ternary vector is that n is a power of two. The second constraint implies that we can find a way to assign codes to the EIs covered by two half-cutting ranges A and B such that both A and B can be represented by on one ternary vector, as explained by the following example. Consider two half-cutting ranges A and B that contain 2^a and 2^b EIs, respectively. Let $a \geq b$, and thus A half-cuts B. Range A can be assigned a a-bit block of BRGC codes. Let the rightmost $(b-1)$-bit block of BRGC codes inside A be block_i and the right neighboring $(b-1)$-bit block of block_i outside of A be block_{i+1}. We can assign block_i and block_{i+1} to range B. Thus, both ranges can be represented with one only ternary vector individually.

Subsequently, we shall first give some useful properties of BRGC range sets and present the proposed layered BRGC range encoding scheme later.

Lemma 1: Let ranges A, B, and C belong to a BRGC range set. If both A and B intersect C, ranges A and B must not be intersected.

Proof: If A and B are intersected, then the following inequality must be true:

$$0 < |A \cap B| < \min\{|A|, |B|\}. \tag{1}$$

Now, there are three cases to prove: I) $|C| < \min\{|A|, |B|\}$; II) $|C| > \max\{|A|, |B|\}$; and III) $\min\{|A|, |B|\} \leq |C| \leq \max\{|A|, |B|\}$. The proof can be referred to Fig. 4. In case I), $|A \cap C| = |B \cap C| = |C|/2$ implies two subcases: Ia) A and B are adjacent; and Ib) A and B are weak-nested, which lead to $|A \cap B| = 0$ and $|A \cap B| = \min\{|A|, |B|\}$, respectively. This is a contradiction to inequality (1). In case II), two possible subcases are: IIa) both A and B are left half-cut or right half-cut by C; and IIb) one is left half-cut by C and the other is right half-cut by C. In subcase IIa), A and B are nested, and thus $|A \cap B| = \min\{|A|, |B|\}$; in subcase IIb), A and B are disjoint, and thus $|A \cap B| = 0$. This is a contradiction.

In case III), without loss of generality, we assume $|A| \leq |C| \leq |B|$. Since A and C are intersected, one may have two subcases: A is left half-cut by C and right half-cut by C. Now,

we consider the former, while the latter can be discussed symmetrically. Since $|C| \leq |B|$, we may have two subcases: IIIa) C is left half-cut by B; and IIIb) C is right half-cut by B. Subcase IIIa) leads to $|A \cap B| = 0$, and subcase IIIb) leads to $|A \cap B| = \min\{|A|, |B|\}$, which contradict inequality (1). Thus, A and B are not intersected for all these three cases. □

Based on Lemma 1, if a range in a BRGC range set is half-cut by two different ranges, then these two ranges must be adjacent or weak-nested. Subsequently, we temporarily focus on the special condition in which any two ranges in a BRGC range set are not adjacent.

Lemma 2: Let n ranges A_1 to A_n belong to a BRGC range set in which any two ranges are neither adjacent nor weak-nested. If A_i and A_{i+1} are half-cutting for $i = 1, \ldots, n-1$, then A_n must not be half-cut by any of the ranges $A_1, \ldots, A_{n-3}$, and A_{n-2}.

Proof: Assume range A_t is a range that contains the largest number of elementary intervals among all ranges. As a result, both ranges A_{t-1} and A_{t+1} are half-cut by A_t and in turn A_{t-2} and A_{t+2} are half-covered by A_{t-1} and A_{t+1}, respectively, and so on. Assume the number of elementary intervals contained in A_i is 2^{di}. Since no ranges are adjacent or weak-nested, either A_{t-1} and A_{t+1} may have the same number of elementary intervals as A_t, but not both. Also, we must have $|A_{t+k}| > |A_{t+k+1}|$ for $k = 1, \ldots, n-t-1$, i.e., $d_{t+1} > d_{t+2} > \ldots > d_n$. Otherwise, if $|A_{t+k}| \leq |A_{t+k+1}|$, then A_{t+k} will be half-cut by both A_{t+k-1} and A_{t+k+1}, which implies that A_{t+k-1} and A_{t+k+1} are adjacent or weak-nested (by Lemma 1), which contradicts the assumption. Similarly, we can prove that $|A_{t-k}| > |A_{t-k-1}|$ for $k = 1, \ldots, i-2$. Hence, $d_{i-1} > d_{i-2} > \ldots > d_1$.

Without loss of generality, we assume that $d_n \geq d_k$ and $d_n < d_{k+1}$ for $k \in \{1, \ldots, t-1\}$. If A_n and A_j for $j = k+1, \ldots n-2$ are half-cutting, then A_n is half-cut by A_j because $d_n \leq d_j$. This would imply that A_n is half-cut by both A_{n-1} and $A_{j.}$. Then, by Lemma 1, A_{n-1} and A_j are adjacent or weak-nested, which contradicts the assumption. Therefore, A_n is not half-cut by any range in $\{A_{k+1}, A_{k+2}, \ldots, A_{n-3}, A_{n-2}\}$. Next, we show that A_n and any range in $\{A_1, A_2, \ldots, A_{k-1}, A_k\}$ are also not half-cutting. Suppose, by contradiction, that A_n and any range $A_j \in \{A_1, A_2, \ldots, A_{k-1}, A_k\}$ are half-cutting. Then, A_j must be half-cut by A_n because $d_j \leq d_n$. However, since A_j is also half-cut by A_{j+1}, then according to Lemma 1, this would lead to a contradiction. Thus, the lemma follows. □

Theorem 1: Each range in a BRGC range set G can be represented by a unique ternary vector provided that any two ranges in G are neither adjacent nor weak-nested.

Proof: The theorem is proved by construction as follows. We first select the maximal range R_1 that contains the maximal number of elementary intervals among all the ranges in the BRGC range set. Tie is broken randomly. We can easily assign a block of BRGC codes to R_1. Based on Lemmas 1 and 2, we can find a set of ranges $R_2, R_3, \ldots$, and R_n such that R_i half-cuts R_{i+1} for $i = 1$ to $n-1$, i.e., $d_1 \geq d_2 > d_3 > \ldots > d_n$, where $|R_i|$ is 2^{di}. As a result, R_{i+1} can be assigned a d_{i+1}-bit block of BRGC codes after R_i is given the needed d_i-bit block of BRGC codes. Thus, the theorem follows. □

Theorem 2: Each range in a BRGC range set G can be represented by a unique ternary vector, provided that any two ranges in G are not adjacent.

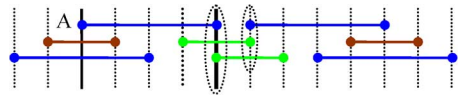


Fig. 5. Proof of Theorem 3.

Proof: We have the following two cases: 1) Any two ranges are not aligned; and 2) there exist two weak-nested ranges A and B in G. By Theorem 1, case 1 immediately holds.

For case 2, the proof is by induction on the number n of ranges in G. Without loss of generality, we assume A is left weak-nested by B. Two base cases for $n = 2$ clearly hold because A consisting of only one EI can be assigned code 00 and B consisting of two EIs can be assigned codes 00 and 01. Assume that the result holds when $n = k$. Now, we consider the case where $n = k+1$. We can temporarily remove range A from G. If one of the A's rightmost and leftmost EI is not shared with other ranges, this EI will also be removed from the BRGC range set. Removing an EI from G may invalidate the two constraints of BRGC range set for some ranges. However, we can add this EI back by using a virtual point so that all the ranges (excluding A) in G still satisfy the two constraints. By the induction hypothesis, all the ranges except A can be represented by unique ternary vectors. Now, if we add A back into G, all ranges will not violate the constraints of the BRGC range set. Moreover, since $|A| = |B|/2^k$ for $k \geq 1$ and A and B share the same rightmost and leftmost EI, it can also be assigned a block of BRGC codes that is the rightmost or leftmost subblock of the BRGC codes assigned to B. By the property of BRGC, the codes assigned to A can be represented by a unique ternary vector. Hence, the theorem follows. □

Theorem 3: It is not always possible that every range in a BRGC range set can be represented by a unique ternary vector, provided that if two ranges in the set are adjacent.

Proof: We prove it by contradiction. The example in Fig. 5 shows that both ranges A and B cannot be given a block of BRGC codes simultaneously. Thus, the theorem follows. □

Definition 5: A BRGC range set is called perfect BRGC range set if it covers $2^d - 1$ valid elementary intervals that need $2^d - 1$ distinct codes.

If a perfect BRGC range set is the only range set in a layer, a d-bit code space is sufficient for the layer, and the unused code is assigned to the default elementary intervals. As a result, all 2^d d-bit codes are used, and no code is wasted. We show the perfect BRGC range sets consisting of two, four, and eight ranges in Fig. 6. For example, Fig. 6(a) shows that the two ranges cover three valid elementary intervals that are assigned with the BRGC codes 01, 11, and 10. The code 00 is assigned to the two default elementary intervals. As a result, these two ranges can be represented with two 2-bit match conditions, 1^* and ^{*1}. The perfect BRGC range sets of four or more ranges can be constructed recursively. For example, the BRGC range set in Fig. 6(b) is constructed from Fig. 6(a) by adding two ranges shown in red. A BRGC range sets in Fig. 6(c) is constructed from 6(b) by adding four dark blue ranges that half-cut the black ranges at the black endpoints. In general, we can construct a perfect BRGC range set that consists of 2^n ranges covering $2^{n+1} - 1$ valid elementary intervals, and thus an $(n+1)$-bit code space is needed. We denote these perfect BRGC range sets that contain 2^n ranges by $\mathrm{Perfect}(2^n)$. If $\mathrm{Perfect}(2^n)$ is organized into layers by PPC

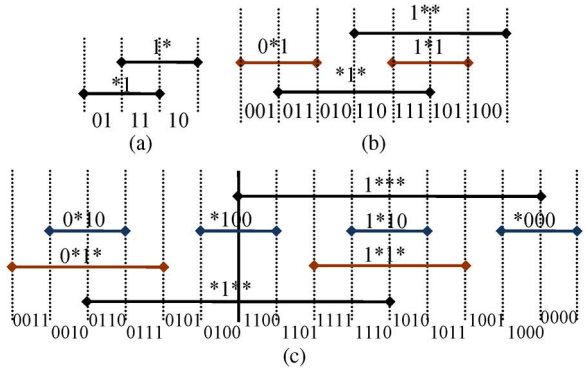


Fig. 6. Perfect BRGC range sets of 2^n ranges, $\mathrm{Perfect}(2n)$. (a) $\mathrm{Perfect}(2^1)$; (b) $\mathrm{Perfect}(2^2)$ (code 000 is unused); (c) $\mathrm{Perfect}(2^3)$ (code 0001 is unused).

1. ***E*** *is the elementary interval (EI) array built from the intersecting range set I.*
2. *R.i and R.j are the indexes of the left-most and right-most EIs covered by range R.*
3. *The maximal range R_{max} in G is the range that covers the maximum # of EIs and its left-most EI will be aligned to two consecutive (c–1)-bit blocks of d-bit code space, where $2^c = R_{max}.j - R_{max}.i + 1$. Thus, the exact mapping of |E| elementary intervals onto d-bit code space is as follows:*

EI	1	...	$R_{max}.i$	...	h	...	$R_{max}.j$	...	$\lvert E\rvert-1$
code	$(-R_{max}.i+1)\%2^d$	...	0	...	$(h-R_{max}.i)\%2^d$	...	$R_{max}.j-R_{max}.i$	...	$-R_{max}.i+\lvert E\rvert-1$

```
Encode_a_BRGC_set(G, E) {
//================== Find the maximal range R_max ==============
01 max = 0;
02 for (each range R in G)
03    if (R.j – R.i + 1 > max){max = R.j – R.i + 1; R_max = R; c = log2(max); }
04 d = ⌈log2(|E|+1)⌉; // d-bit code space is needed for |E| EIs along with the
       default EI.
//==maximal range R_max alignment: assume R_max is aligned to code 0 at its left =
05 for (h = 0 to |E|– 1){ E[h].code = b2g((h – R_max.i) mod 2^d); }
06 for (each range R in G)
07    if (codes E[R.i].code to E[R.j].code cannot be merged into a ternary
       vector) return False;
08 return True;
}
```

Fig. 7. Code assignment for a single BRGC range set.

style-I, $n+1$ layers are needed. Notice that any two ranges in $\mathrm{Perfect}(2^n)$ are not adjacent and weak-nested. There are many other possible perfect BRGC range groups when two ranges in the set are adjacent or weak-nested.

Checking whether or not an intersecting range set I is a BRGC range set is implemented in function $\mathrm{Is\_BRGC\_set}(\mathrm{I}, E, \mathrm{count})$ by examining the two constraints in Definition 3. Parameter E is the EI array constructed from I, and parameter count outputs the number of ranges that do not satisfy the first constraint. There are at most $2n+1$ EIs constructed for the endpoints of n ranges in I, and thus $O(n^2)$ time is needed for constructing the EI array. Checking constraint 1 for all n ranges takes $O(n)$ time. Checking constraint 2 for all $O(n^2)$ pairs of ranges in I takes $O(n^2)$ time. As a result, $\mathrm{Is\_BRGC\_set}(\mathrm{I}, E, \mathrm{count})$ can be performed in $O(n^2)$ time.

Fig. 7 shows the function $\mathrm{Encode\_a\_BRGC\_set}(G, E)$ that encodes the ranges in a BRGC range set G. As in lines 2 and 3, the *maximal range* $R_{\max}$ in G covering the maximum number of EIs is first located. Line 4 computes the code size needed for G. In line 5, a d-bit BRGC block is assigned to all the EIs contained in G by aligning the leftmost EI of range $R_{\max}$ to code 0. Since it is not guaranteed that all the ranges can be represented by single ternary vectors stated in Theorem 3, lines 6 and 7 perform the check. Since there are $O(n)$ EIs for n ranges, the time complexity of $\mathrm{Encode\_a\_BRGC\_set}(G, E)$ is $O(n)$.

1. *m: max # of virtual endpoints to be added.*
2. *k : the # of virtual endpoints not yet inserted.*
3. *D[1..m]: i^{th} virtual endpoint is inserted in the D[i]th elementary interval, i.e., E[D[i]].*
4. *success (set to false initially): global flag to indicate if a BRGC range set is found.*

```
Virtual_endpoint_insert(G, D, m, k)
{
01 if (k = 0){
02    for (i = 1 to m)
03       Add a virtual endpoint in the D[i]th element of EI array E, i.e., E[D[i]];
04    if (Is_BRGC_set(G, E, count) = True and
          Encode_a_BRGC_set(G, E) = True) success = True;
05    return;
06 }
07 for (j = 0 to |E| – 1) {
08    if (success is True) break;
09    D[k] = j;
10    Virtual_endpoint_insert(G, D, m, k – 1);
11}
}
```

Fig. 8. Virtual endpoint insertion.

Subsequently, a *virtual endpoint insertion* algorithm is developed to convert a non-BRGC range set to a BRGC range set by inserting some virtual endpoints. For example, Fig. 3(c) shows that ranges R1, R3, and R4 do not form a BRGC range set because the number of elementary intervals contained in R4 is not a power of 2. By adding one virtual endpoint into elementary interval E2 in Fig. 3(c), R4 will cover four elementary intervals assigned four BRGC codes and thus can be represented by the ternary vector *1*.

Fig. 8 shows the function $\mathrm{Virtual\_endpoint\_insert}(G, D, m, k)$ that implements the virtual endpoint insertion process recursively. This function, initially called $\mathrm{Virtual\_endpoint\_insert}(G, D, m, k)$, tries to add k virtual endpoints to the EI array E. A global variable *success* is used, if set to true, to stop the recursive virtual endpoint insertion process when a range set is successfully confirmed to be a BRGC range set after some virtual endpoints are inserted. Virtual endpoints can be added in any of the $|E|$ elementary intervals. The fourth parameter k is decremented by one every time the function is called recursively, as shown in line 11. When k reaches zero in line 1, the process of adding k virtual endpoints is completed. Functions $\mathrm{Is\_BRGC\_set}(G, E, count)$ and $\mathrm{Encode\_a\_BRGC\_set}(G, E)$ are then used to check if the given range set is a BRGC range set. The worst-case time complexity of the $\mathrm{Virtual\_endpoint\_insert}(G, D, m, k)$ is $O(n^{k+2})$ because each endpoint can be inserted into any of the $O(n)$ elementary intervals, and $\mathrm{Is\_BRGC\_set}$ takes $O(n^2)$ time.

Inserting too many virtual endpoints (i.e., a large k) is very time-consuming. It is also difficult to determine the minimum number of virtual endpoints needed to be inserted in order to convert a non-BRGC range set to a BRGC range set. Therefore, the following virtual endpoint assignment scheme is used with caution. For a range set, if there are count ranges that do not satisfy constrain 1, $\mathrm{count}+s$ virtual endpoints are inserted, where count is returned from $\mathrm{Is\_BRGC\_set}(G, E, count)$ and s is a predetermined value. In this paper, we set s to 1. We start executing $\mathrm{Virtual\_endpoint\_insert}(G, D, count+s, k)$ by setting $k = \mathrm{count}+s$. If the range set G can be converted to a BRGC range set, then the process stops. Otherwise, this process is repeated by decrementing k by one in next round until $k = 0$.

```
1. I = {R1...Rn}: the intersecting range set;
2. layers: # of layers.
Incremental_layered_insertion(I)
{
01 Insert R0 into layer 0; layers = 1;
02 for (i = 1; i < |I|; i++) { // insert one range at a time
03   for (j = 0; j <layers; j++) {
04     Temporarily insert range Ri into layer j;
05     Compute all the maximal intersecting range sets, G1 to Gn, in layer j;
06     if (all G1 to Gn can be successfully encoded by the proposed BRGC
       schemes based on the functions shown in Figs. 7 and 8) {
07     Make the insertion of Ri into layer j permanent;
08     break; // break inner loop j
09   }
10 } // end for loop j
11 if (j = layers) {
12   Create a new layer and insert Ri in it; layers++;
13   assign code 0 to EI covered by the only range and 1 to default EI; }
14 } // end for loop i
15 return layers; // return # of layers constructed
}
```

Fig. 9. Incrementally insert a range into layers.

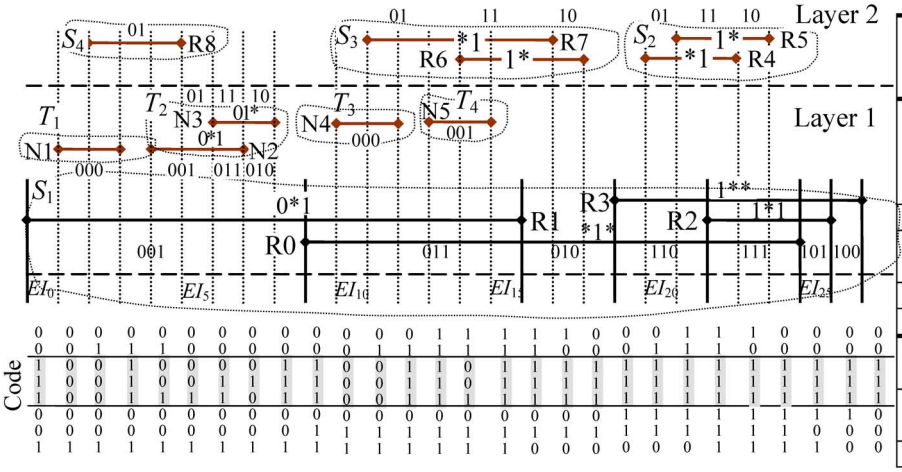
To deal with the situation when an intersecting range set is not a BRGC range set or when converting a non-BRGC range set to a BRGC range set by adding virtual endpoints fails, an incremental range insertion scheme to assign codes in a layered fashion is proposed. This scheme adds the ranges of an intersecting range set I into layers, one at a time, as shown in the self-explanatory function $\mathrm{Incremental\_layered\_insertion}(I)$ of Fig. 9. After the successful incremental insertion process, the codes assigned to all maximal intersecting range sets in each layer can be combined into the complete ternary vectors for all the ranges. Before the details of the code-combining process are given, we first show an example in Fig. 10. Two layers are assumed to be generated. Layer 1 contains five BRGC sets in which S_1 nests the other four BRGC sets, and layer 2 contains three disjoint BRGC sets, S_2, S_3, and S_4. Fig. 10 also shows the BRGC codes assigned to these eight BRGC range sets by the code assignment schemes, which will be subsequently described.

The code assignment for the BRGC sets in a layer is described here. Two BRGC sets in the same layer can be disjoint or nested. For example, Fig. 11 illustrates that sets T_1 to T_4 in layer 1 are nested by S_1, and sets S_2, S_3, and S_4 in layer 2 are disjoint. The method of assigning codes to disjoint BRGC range sets is first described. As aforementioned, when assigning codes to many disjoint BRGC range sets, the total number of bits required cannot be determined until the number of codes each BRGC range set needs is determined. The code assignments for the disjoint BRGC range sets in the same layer can be performed independently or dependently, as in PPC encoding schemes [14]. It can be recalled that the main step in Fig. 7 in assigning codes to a BRGC range set is the alignment of the maximal range $R_{\max}$ of size 2^c to two consecutive unused $(c-1)$-bit blocks of BRGC codes. There is only one default EI for all these BRGC range sets in a layer, and thus we only have to arbitrarily find an unused code for the default EI at the last step.

To minimize the code size needed for a group of disjoint BRGC range sets, $\mathrm{S} = \{S_1 \ldots S_n\}$, in a layer, the code assignment is implemented in function $\mathrm{Encode\_disjoint\_BRGC\_sets(S)}$ of Fig. 11. E_i is the EI array of S_i and the number of elementary intervals covered by S_i is $|E_i|$. Each elementary interval of S_i may also nest a number of disjoint BRGC range sets. In line 1 of Fig. 11, function $\mathrm{Encode\_a\_nesting\_BRGC\_set}(S_i, E_i)$ shown in Fig. 12 is first executed to combine the codes assigned to the BRGC range sets nested by S_i and to obtain the number of extra $_g$i bits needed in S_i, where $g_i = S_i.\max\_\mathrm{nested\_bits}$. In line 2, all n BRGC range sets in S are sorted according to their sizes $|E_i|^*2^{gi}$ in the nonincreasing order. Code assignment starts from the largest set S_1 and ends in the smallest set S_n. In lines 3 and 4, a d-bit block of codes $(d = \lceil log_2|E_1| \rceil + g_1)$ is allocated initially. In lines 5–12, a loop is started to assign codes to S_i by finding a series of unused codes for elementary intervals of S_i so that the maximal range $R_{i,\max}$ in S_i can be aligned to a $(c_i - 1)$-bit block of d-bit BRGC codes, where $R_{i,\max}$ contains 2^{ci} elementary intervals. Initially, we allocate a d-bit code space (i.e., a d-bit BRGC block) and assign $|E_1|^*2^{g1}$ codes to S_1. Next, from the unused codes (called an *address hole*) in the allocated d-bit code space, we try to search a series of $|E_2|^*2^{d2}$ unused codes so that the maximal range of S_2 is exactly aligned to a $(c_2 - 1)$-bit block. If the search process fails, the size of the allocated d-bit block will have to be doubled to accommodate S_2 (i.e., one more bit is used for the code space) and prepend a bit 0 to the codes already assigned to S_1. The codes are continuously assigned this way until all n BRGC range sets are exhausted. In general, when assigning codes to S_i, the holes inside the d-bit code block already allocated are sought for a series of $|E_i|^*2^{gi}$ unused codes that can be aligned to the maximal range of S_i. Finally, an unused code should be found for the default elementary interval. If there is no unused code available, one more bit has to be used for the code space of the layer.

For example, consider the two disjoint BRGC sets T_1 and T_2 in Fig. 10. Since T_2 is larger than T_1, codes 01, 11, and 10 are first assigned to the three elementary intervals of T_2. T_1 is then assigned the only remaining code 00. However, one more code is needed for the default interval. Therefore, the code space has to be increased by one bit, which results in appending all the four 2-bit codes with a bit 0 and arbitrarily assigning one of the remaining codes—100, 101, 110, or 111—to the default elementary interval.

The method of assigning codes to the disjoint BRGC range sets nested by another BRGC range set is subsequently discussed. If two BRGC range sets X and Y are contained in two distinct elementary intervals E_x and E_y of another BRGC range set Z, respectively, X and Y must be assigned the same code size because the match condition of any range in Z must be represented by a single ternary vector. The resulting code size required for range sets X, Y, and Z will be $\max(L_X, L_Y) + L_Z$, where L_G denotes the code size of a BRGC range set $G \in \{X, Y, Z\}$. Let the match condition of a range R_Z computed only with the ranges in Z be $mat(R_Z)$. The resulting match condition of R_Z computed with all the ranges in X, Y, and Z will be $mat(R_Z)$ prepended with $\max(L_X, L_Y)$ "don't care" bits. Let the match condition of a range R_X computed only with the ranges in X be $\mathrm{mat}(R_X)$. The resulting match condition of R_X computed with all the ranges in X, Y, and Z will be $\mathrm{mat}(R_X)$ appended with the code assigned to the elementary interval E_x. For example, the two disjoint BRGC range sets, T_3 and T_4, can be assigned the same codes as that of T_1 and T_2. When the code assignment process in Fig. 11 is executed for the two disjoint range sets T_3 and T_4, ranges N4 and N5 get the codes 00 and



BRGC Set		Match condition Layer 2	Layer 1	
S_1	R0	**	***	*1*
	R1	**	***	0*1
	R2	**	***	1*1
	R3	**	***	1**
T_1	N1	**	000	001
T_2	N2	**	0*1	001
	N3	**	01*	001
T_3	N4	**	000	011
T_4	N5	**	001	011
S_2	R4	*1	***	11*
	R5	1*	***	11*
S_3	R6	1*	***	01*
	R7	*1	***	01*
S_4	R8	01	***	001

Fig. 10. Two layers of BRGC sets constructed from an intersecting range set of 14 ranges.

```
1. S = {S1…Sn} is a group of n disjoint BRGC range sets and the maximal range in
   Si is Ri,max of size2^gi.
2. The EI array of Si is Ei.
3. gi = Si.max_nested_bits records the maximum # of bits needed for the BRGC
   range sets nested by Si and the total number of codes required for Si is |Ei|* 2^gi.
Encode_disjoint_BRGC_sets(S)
{
01 for (i = 1 to n) Encode_a_nesting_BRGC_set(Si, Ei);
02 Sort S1…Sn according to their sizes |Ei|*2^gi in a non-increasing order;
03 d = ⌈log2|E1|⌉+g1; //the number bits needed for the code space
04 Allocate an array C of size 2^d and reset all its 2^d elements;
05 for (i = 1 to n) {
06   Assign a series of unused codes for Ei in the d-bit code space such that
     the maximal range Ri,max in Si can be aligned to a (ci–1)-bit block of
     BRGC codes)
07   if (the above assignment process fails) {
08     d++; // increment d by one bit to double the code space
09     Double the size of array C while keeping the old values in C
10     Prepend one more bit 0 to the codes already assigned to the ranges in
       sets S1 to Si–1;
11     Assign a series of unused codes for Ei in the d-bit code space such that
       the maximal range Ri,max in Si can be aligned to a (ci–1)-bit block of
       BRGC codes
12   }
13 }
14 return d;
}
```

Fig. 11. Code assignment for disjoint BRGC range sets.

```
1. E is the EI array of a BRGC range set G.
2. Each elementary interval E[h] for h = 0 to |E| contains a group of disjoint BRGC
   range sets, E[h].Gnest .
Encode_a_nesting_BRGC_set(G, E){
01 G.max_nested_bits = 0;
02 for (h = 0 to |E|– 1){
03   if (E[h].Gnest ≠ ϕ) dh = Encode_disjoint_BRGC_Sets(E[h].Gnest);
04   if (dh > G.max_nested_bits) G.max_nested_bits = dh;
05 }
// Merging with the codes assigned to the BRGC range sets covered by all EIs
06 for (h = 0 to |E|– 1){
07   if (E[h].Gnest ≠ ϕ){
08     Append (G.max_nested_bits – dh)-bit 0 to all the codes assigned to the
       IEs in E[h].Gnest
09     Prepend E[h].code to all the codes assigned to the IEs in Gnest;
10     select one unused G.max_nested_bits-bit code and append it to
       E[h].code;
11   } else { append G.max_nested_bits-bit 0 to E[h].code; }
12 }
}
```

Fig. 12. Code assignment for nested BRGC range sets.

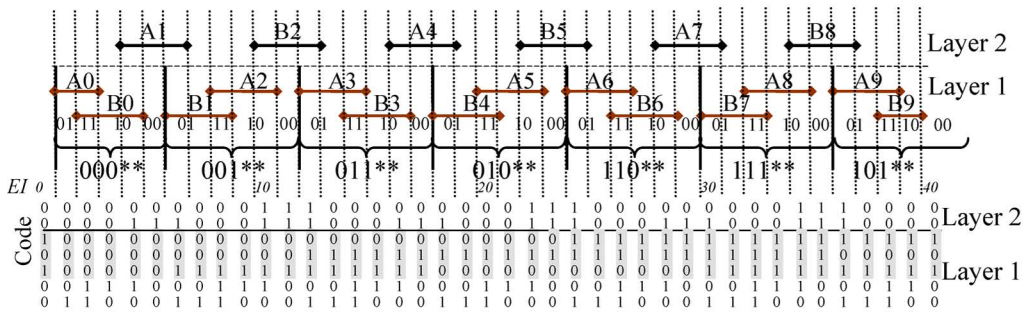
01, respectively, and the default elementary interval gets either 10 or 11. To make T_3 and T_4 have the same code size as T_1 and T_2, the codes assigned to N4 and N5 are prepended with a bit 0, and thus their codes become 000 and 001. The default elementary interval will get any one of the remaining unused codes.

Encode_a_nesting_BRGC_set(G, E) shown in Fig. 12 is a recursive function that implements the code assignment for a BRGC range set nesting some other disjoint BRGC sets. If an elementary interval contains a group of disjoint BRGC range sets, function Encode_disjoint_BRGC_sets(S) of Fig. 11 is used recursively to complete the required task. The main purpose is to obtain the maximum number of nested bits called $G.\max\_\text{nested\_bits}$ for all BRGC range sets nested by G. Lines 2–5 compute $G.\max\_\text{nested\_bits}$ needed among all the EIs in G. Lines 6–12 perform the following tasks for each $E[h]$. Since all the EIs contained in G can share the code space of the same size ($G.\max\_\text{nested\_bits}$ bits), all the codes assigned to the BRGC range sets (denoted by G_{nest}) nested by $E[h]$ are appended with $G.\max\_\text{nested\_bits}$—$s$ 0's and prepended with $E[h]$.code. We then select one unused code ($G.\max\_\text{nested\_bits}$-bit) as the code for default EI inside $E[h]$ and assign it to $E[h]$.code. For example, the BRGC range sets nested by S_1 are T_1, T_2, T_3, and T_4. The maximum number of nested bits $G.\max\_\text{nested\_bits}$ will be 3 bits. The final results of the match conditions of all ranges are shown in Fig. 10.

V. Performance Evaluation

In this section, the performance of the proposed BRGC-based encoding schemes is evaluated and compared to existing ones. It has been assumed that the search key translation process (intermediate result computation) is implemented by a fast SRAM that stores 65 536 precomputed search keys for all possible 65 536 16-bit source or destination port values. We first give the performance analysis and compare it with PPC style-I to show that L-BRGC is superior, especially when the number of ranges to be encoded is scaled to a large number. Then, we give the experimental results based on real-world rule sets.

The main difference between L-BRGC and PPC style-I is that L-BRGC puts disjoint BRGC range sets or the ones nested by



Ternary vectors.

Layer 2			Layer 1		
A1	01	00***	A0	**	000*1
B2	11	0*1**	B0	**	0001*
A4	01	01***	B1	**	001*1
B5	11	*10**	A2	**	0011*
A7	01	11***	A3	**	011*1
B8	11	1*1**	B3	**	0111*
			B4	**	010*1
			A5	**	0101*
			A6	**	110*1
			B6	**	1101*
			B7	**	111*1
			A8	**	1111*
			A9	**	101*1
			B9	**	1011*

Fig. 13. L-BRGC encoding example for the ranges equally distributed among two layers.

TABLE I
NUMBERS OF BITS NEEDED FOR A PERFECT BRGC RANGE SET ENCODED BY L-BRGC AND PPC STYLE-I

Perfect BRGC range set	*L-BRGC*	*PPC style-I*
2^1 ranges	2	2
2^2 ranges	3	4
2^3 ranges	4	6
2^4 ranges	5	10
2^5 ranges	6	14
2^6 ranges	7	20

one of the elementary intervals of these disjoint BRGC range sets into the same layer. As proposed in Section IV, every time a range (say R) is to be inserted, L-BRGC tries to find an existing layer such that R can be combined with one or more existing BRGC range sets to form a larger BRGC range set by adding some virtual endpoints if needed. If all the ranges are disjoint (i.e., only one layer is needed in PPC style-I), each BRGC range set can only contain one range, L-BRGC is degenerated into PPC style-I. However, overlapping and intersecting ranges do exist in range fields of the real-world rule sets and the number of layers needed in terms of PPC style-I is a small constant. Thus, it is highly possible that overlapping and intersecting ranges can form BRGC range sets that can be put in the same layer and the number of bits will be reduced compared to PPC style-I. Subsequently, we shall compare the numbers of bits needed for L-BRGC and PPC style-I by assuming there are N ranges that need L layers encoded by PPC style-I.

We first consider the best case scenario for L-BRGC in which all the ranges can be grouped as the only perfect BRGC range set, e.g., $\mathrm{Perfect}(2^n)$, as described in Section IV. For a $\mathrm{Perfect}(2^{L-1})$ range set, only L bits are needed for L-BRGC. If the ranges in $\mathrm{Perfect}(2^{L-1})$ are organized by PPC style-I, L layers are required and thus at least $L-1+\log(2^{L-1}-L+1) \approx 2(L-1)$ bits are needed for PPC style-I. Since the order in which ranges are inserted influences how to select a layer to put the ranges to be inserted in PPC style-I, we compute the numbers of bits needed in $\mathrm{Perfect}(2^n)$ for $n = 1$ to 6 and compare them to PPC style-I in Table I. We can see that when $n \geq 4$, L-BRGC needs at most half of the number of bits needed in PPC style-I.

Second, we consider another best scenario for L-BRGC in which N ranges can be grouped into $\lceil N/2^L \rceil$ disjoint $\mathrm{Perfect}(2^L)$ range sets. Therefore, each $\mathrm{Perfect}(2^L)$ set takes $2^{L+1}-1$ codes, and L-BRGC needs $\mathrm{LBRGC}_{\min}(N,L)$ bits computed as follows:

$$\mathrm{LBRGC}_{\min}(N,L) = \left\lceil \log\left(\frac{N}{2^L} \times (2^{L+1}-1)+1\right)\right\rceil \cong \log N + 1.$$

Third, we show the worst-case scenario for both L-BRGC and PPC style-I. As shown in [14], the number of bits denoted by $\mathrm{PPC}_{\mathrm{I,max}}(N,L)$ for PPC style-I to encode a set of N ranges is maximized when all ranges are distributed equally among L layers. $\mathrm{PPC}_{\mathrm{I,max}}(N,L)$ can be computed as follows:

$$\mathrm{PPC}_{\mathrm{I,max}}(N,L) = L\left\lceil \log\left(\frac{N}{L}+1\right)\right\rceil.$$

For L-BRGC, we shall show that L-BRGC only needs no more than a half number of bits needed in PPC style-I for the ranges that equally distributed among layers. Consider $L = 2$ first. The most complicated condition is that all ranges are intersecting in a chain fashion as shown in Fig. 13 for $N = 20$.

Fig. 13 shows the range distribution encoded by L-BRGC. For L-BRGC, the code assignment to all 41 elementary intervals and the final ternary vectors given to all ranges are clearly shown in Fig. 13. Two thirds of the ranges (14 ranges) form seven disjoint $\mathrm{Perfect}(2^1)$ range sets that can be put in layer 1. These disjoint $\mathrm{Perfect}(2^1)$ range sets can be assigned codes easily based on the proposed encoding scheme. Totally, 4×7 codes are required, and thus a 5-bit code space is needed in layer 1, where these seven 2-bit code subspaces are assigned based on binary reflected Gray code. For the six remaining ranges that are put in layer 2, we can use the PPC style-II to reduce the size of the code space needed. Two neighboring ranges (e.g., A1 and B2) are given different codes, and thus two codes (01 and 11) plus the code (00) for default elementary intervals are needed. Thus, 7-bit code space is needed for L-BRGC. If the ranges in Fig. 13 are organized by PPC style-I, two layers are also needed. Each layer takes 10 ranges. As a result, 8-bit code space is needed. This example can be generalized to N ranges, and the number of bits needed for L-BRGC denoted by $\mathrm{L-BRGC}_{\max}(N,2)$ is computed as follows:

$$\mathrm{LBRGC}_{\max}(N,2) = \left\lceil \log\left(\frac{N}{3} \times 4 + 1\right)\right\rceil + 2.$$

Now, consider the case when $L > 2$. Each layer contains N/L ranges. We take two layers at a time and encode them by

TABLE II
PERFORMANCE OF REAL-LIFE RULE TABLES

Schemes	All rules							Non-prefix rules						
	# of Rules	Entry sizes (bits)		# of TCAM Entries	EF	TCAM size (Kb)	SRAM size (KB)	# of Rules	Entry sizes (bits)		# of TCAM Entries	EF	TCAM size (Kb)	SRAM size (KB)
		src	dst						src	dst				
DC	269 (*fw1*)	16	16	914	3.40	28.56	0	27 (*fw1*)	16	16	672	24.89	21.00	0
SRGE		16	16	902	3.35	28.19	0		16	16	660	24.44	20.63	0
EIDC		5	6	1,811	6.73	19.45	88		3	3	84	3.11	0.49	48
EIGC		5	6	1,760	6.54	18.91	88		3	3	84	3.11	0.49	48
DIRPE		34	29	434	1.61	26.70	0		34	29	192	7.11	11.81	0
Bitmap		13	43	269	1.00	14.71	448		4	4	27	1.00	0.21	64
PPC		7	8	269	1.00	3.94	120		3	3	27	1.00	0.16	48
L-BRGC		4	6	269	1.00	2.63	80		3	3	27	1.00	0.16	48
DC	752 (*acl1*)	16	16	1,834	2.44	57.31	0	165 (*acl1*)	16	16	1,247	7.56	38.97	0
SRGE		16	16	1,834	2.44	57.31	0		16	16	1,247	7.56	38.97	0
EIDC		1	7	1,940	2.58	15.16	64		1	6	558	3.38	3.81	56
EIGC		1	7	1,878	2.50	14.67	64		1	6	505	3.06	3.45	56
DIRPE		1	29	1,418	1.89	41.54	0		1	29	831	5.04	24.35	0
Bitmap		1	140	752	1.00	103.55	1128		1	66	165	1.00	10.80	536
PPC		1	77	752	1.00	57.28	624		1	66	165	1.00	10.80	536
L-BRGC		1	70	752	1.00	52.14	568		1	62	165	1.00	10.15	504
DC	1550 (*ipc1*)	16	16	2,180	1.41	68.13	0	152 (*ipc1*)	16	16	782	5.14	24.44	0
SRGE		16	16	2,145	1.38	67.03	0		16	16	747	4.91	23.34	0
EIDC		6	7	21,638	13.96	274.70	104		3	3	161	1.06	0.94	48
EIGC		6	7	20,296	13.09	257.66	104		3	3	159	1.05	0.93	48
DIRPE		34	29	1,791	1.16	110.19	0		34	29	393	2.59	24.18	0
Bitmap		34	54	1,550	1.00	133.20	704		5	5	152	1.00	1.48	80
PPC		9	13	1,550	1.00	33.30	176		4	5	152	1.00	1.34	72
L-BRGC		6	10	1,550	1.00	24.22	128		3	3	152	1.00	0.89	48

the same encoding process described above. As a result, the total number of bits needed in the code space is

$$\mathrm{LBRGC}_{\max}(N,L) = \left\lfloor \frac{L}{2} \right\rfloor \times \left(\left\lceil \log \left(\frac{\frac{N}{L}}{3} \times 4 + 1 \right) \right\rceil + 2 \right) + \left(\left\lceil \frac{L}{2} \right\rceil - \left\lfloor \frac{L}{2} \right\rfloor \right) \times \left\lceil \log \left(\frac{N}{L} + 1 \right) \right\rceil.$$

Assume all ranges are equally distributed over L layers and L is a small constant. PPC style-I needs $L \times \log N$ bits while L-BRGC only needs $L/2 \times \log N$ bits. Asymptotically, L-BRGC needs a half of the number of bits needed in PPC style-I. Notice that it is possible to encode the ranges equally distributed over L layers more efficiently by considering three or more layers at a time, instead of only two layers.

Now, we show the experimental results based on three real-life rule tables and some large synthesized rule tables generated by ClassBench [18] that were used in the experiments. Real-life rule tables that reflect the current status of the rule tables used today consist of at most a few hundreds of unique ranges in the source and destination port fields. Only the source and destination ports of these rule tables were extracted and evaluated in the experiments based on the storage requirements in SRAM and TCAM. The results were given in terms of expansion factors, defined to be the ratio of the number of expanded rules by some range encoding scheme to the number of original rules.

The evaluated schemes include the direct range-to-prefix conversion (DC), the direct range-to-ternary-vector conversion using Gray code (SRGE) [2], the elementary interval-based scheme using Buddy code (EIDC), the proposed elementary interval-based scheme using binary reflected Gray code (EIGC), the database-independent range pre-encoding (DIRPE) [11], the bitmap intersection scheme (Bitmap) [10], PPC style-II or III that performs the best (PPC) [14], and the proposed L-BRGC scheme. As suggested in [11], DIRPE uses the strides of 2, 2, 3, 3, 3, 3 and 2, 2, 2, 2, 2, 3, 3 for the source and destination port fields, respectively. Thus, TCAM entry widths of source and destination port fields are 34 and 29 bits, respectively.

Table II shows the performance results for three real-life rule tables. Since many original rules called *prefix rules* whose port values in both their source and destination fields are already prefixes, they can be stored in TCAM directly without the need of any encoding scheme. The performance on the nonprefix rules has a primary impact on the overall performance. Therefore, experiments on the nonprefix rules were also conducted. The TCAM size in kilobits (kb) is calculated by (number of TCAM Entries $\times$ sum of source and destination port entry sizes)/1024. The SRAM size in kilobytes (kB) is calculated by $(65\,536 \times \text{sum of source and destination port entrysizes})/(8 \times 1024)$. From the results of the Bitmap scheme, the entry sizes indicate the number of distinct ranges in the source and destination address fields. For example, the entry size of Bitmap scheme is one for table $acl1$ because $acl1$ has only one distinct range value in the source address field, which is 0–65 535. The performance difference between DC and SRGE and between EIDC and EIGC is small. L-BRGC outperforms the other schemes for tables $fw1$ and $ipc1$. However, for $acl1$, EIGC performs the best. After a detailed analysis on the range field values in $acl1$, we found that the source address field contains only the wildcard field value (0–65535), and the destination address fields of the nonprefix rules consist of two completely nested range sets. The proposed L-BRGC scheme performs poorly for the completely

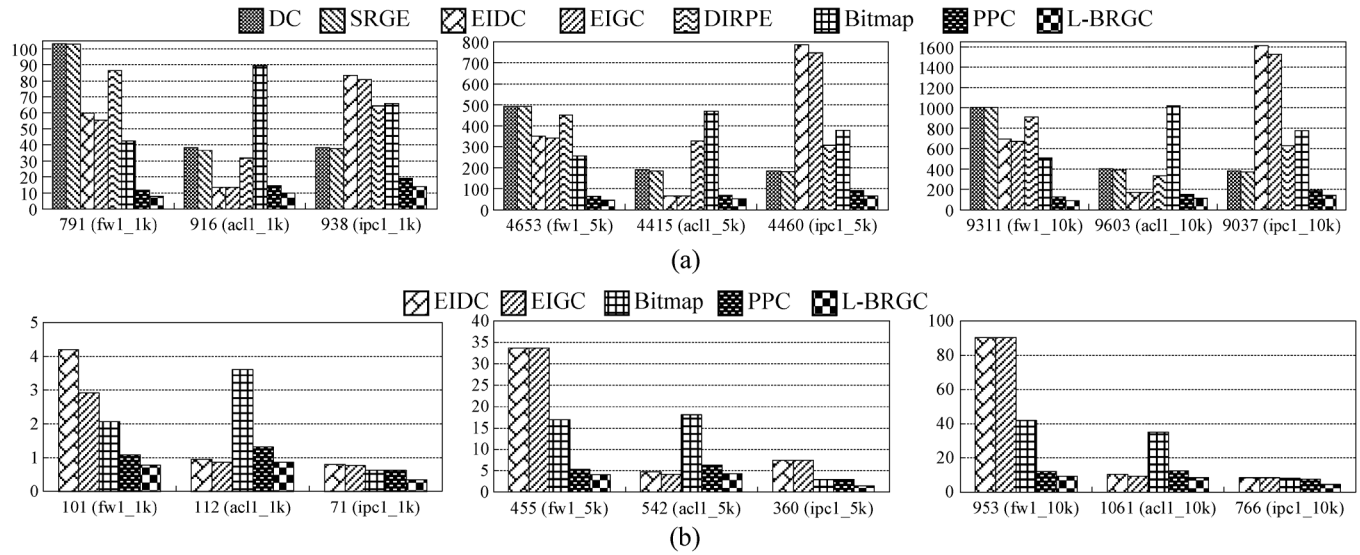


Fig. 14. TCAM cost (kb) comparisons for synthesized rule tables. (a) All rules. (b) Nonprefix rules.

nested range sets. Therefore, L-BRGC needs almost as many bits as the bitmap scheme for each TCAM entry. It turns out that the TCAM requirements for L-BRGC and Bitmap are equally larger than those for other schemes. On the other hand, EIGC is better than SRGE in that the ranges in EIGC encoded with the elementary intervals are converted to a less number of prefixes than their original ranges in SRGE. Thus, EIGC consumes less TCAM than other schemes. Also, if all rules are considered for table $ipc1$, EIDC and EIGC consume the most TCAM because their expansion factor is so large, which is around 13.

The same experiments were also conducted for the synthesized rule tables of 1000, 5000, and 10 000 rules generated by *ClassBench* [18]. However, because space is limited, only the detailed results for 5000 rules are shown in Table III; the TCAM requirements are summarized in Fig. 14, where the results of DC, SRGC, and DIRPE are removed for nonprefix rules to show the differences between PPC and L-BRGC. It is apparent that PPC and the proposed L-BRGC scheme consume less TCAM than other schemes for all ipc, fw, and acl tables. Likewise, L-BRGC needs only 50%–77% of the TCAM required in PPC. In addition, L-BRGC needs at most 128 kB of SRAM when the direct IP address to key translation array is used. The size of SRAM needed for all database dependent schemes does not increase much as the size of rule tables increases.

The above experimental results demonstrate a general idea of how much TCAM a range encoding scheme requires. However, in practice, there may not exist a TCAM that is as wide as the rule entry size needed by a large rule table for some range encoding scheme. Therefore, the hybrid schemes such as *Liu's scheme* [12] and *DRES* [7] are proposed to efficiently utilize the extra bits that are left unused if the original range-to-prefix conversion is used. In the experiments, we use DRES as the basic scheme to study performance impacts of the proposed L-BRGC, PPC, and *bitmap* schemes. In other words, from the original range sets, we select some ranges that are heavier than other ranges, where range A is said to be heavier than another range B if the number of prefixes converted from A is larger than that converted from B based on the direct range-to-prefix conversion. The number of selected ranges are then encoded by scheme L-BRGC, PPC, or *bitmap*. If there are k extra bits, only k heavy ranges can be selected for being encoded by *bitmap*. Since L-BRGC and PPC have a higher encoding efficiency than *bitmap*, they can select more than k heavy ranges. Also, as usual, the direct range-to-prefix conversion is used for the ranges that are not selected. As a result, a fixed number of 32 bits is needed for two port ranges when the direct range-to-prefix conversion is used. The range selection algorithm proposed in [7] is implemented to obtain better encoding gain. We use three different sizes of extra bits, 8, 24, and 40 bits.

The performance results of TCAM entry sizes, number of TCAM entries, TCAM sizes, and expansion factor are reported for the real-life tables and synthesized tables of 5000 rules. We first consider the results of $acl1$ tables shown in Table IV. The source port field needs no extra bit because only one source port field value of "*" exists in acl tables. For the real-life table $(acl1)$, the number of nonprefix destination port field values is much greater than 40. Therefore, all the extra bits are used for encoding the destination port field that needs 24, 40, and 56 bits when 8, 24, and 40 extra bits are available, respectively. As shown in Table IV, L-BRGC has better performance gains in terms of TCAM usage and *expansion factor* (EF) than *bitmap* and PPC. For the large synthesized table $(acl1\_5k)$, only 7 extra bits (i.e., 23 bits in total) are sufficient to encode all the nonprefix ranges of destination port field by L-BRGC. However, 12 and 33 bits are sufficient for schemes *bitmap* and PPC, respectively. For $fw1$ and $ipc1$ tables, eight extra bits are sufficient for all three schemes to obtain the expansion factor of 1.0. In other words, more extra bits do not obtain better performance for $fw1$ and $ipc1$. Thus, the results for 24 and 40 extra bits are not shown in Tables V and VI. For IPC tables, L-BRGC performs a little better than PPC and *bitmap*, while all these three schemes have the same performance for *Firewall* tables. In summary, the performance improvement of L-BRGC over *bitmap* and PPC in terms of TCAM usage increases as the number of nonprefix source or destination port field values increases.

Now, we shall show the preprocessing time and incremental update times of the proposed L-BRGC that are compared to the existing schemes in Tables VII and VIII. We use the three real-world rule sets for computing the preprocessing times and synthesized rule tables of 5000 rules for incremental insertion

TABLE III
PERFORMANCE OF 5000 RULE TABLES

Schemes	All rules							Non-prefix rules						
	# of Rules	Entry sizes (bits)		# of TCAM Entries	EF	TCAM size (Kb)	SRAM size (KB)	# of Rules	Entry sizes (bits)		# of TCAM Entries	EF	TCAM size (Kb)	SRAM size (KB)
		src	dst						src	dst				
DC	4653 (*fw1*)	16	16	15,778	3.39	493.06	0	455 (*fw1*)	16	16	11,580	25.45	361.88	0
SRGE		16	16	15,766	3.39	492.69	0		16	16	11,568	25.42	361.50	0
EIDC		5	6	32,686	7.02	351.12	88		5	6	3,123	6.86	33.55	88
EIGC		5	6	31,723	6.82	340.77	88		5	6	3,123	6.86	33.55	88
DIRPE		34	29	7,342	1.58	451.71	0		34	29	3,144	6.91	193.43	0
Bitmap		13	43	4,653	1.00	254.46	448		12	26	455	1.00	16.88	304
PPC		6	8	4,653	1.00	63.62	112		6	6	455	1.00	5.33	96
L-BRGC		4	6	4,653	1.00	45.44	80		4	5	455	1.00	4.00	72
DC	4415 (*acl1*)	16	16	6,148	1.39	192.13	0	542 (*acl1*)	16	16	2,275	4.20	71.09	0
SRGE		16	16	5,905	1.34	184.53	0		16	16	2,032	3.75	63.50	0
EIDC		1	7	9,664	2.19	75.50	64		1	6	792	1.46	5.41	56
EIGC		1	7	9,542	2.16	74.55	64		1	6	688	1.27	4.70	56
DIRPE		1	29	5,327	1.21	156.06	0		1	29	1,454	2.68	42.60	0
Bitmap		1	108	4,415	1.00	469.96	872		1	33	542	1.00	18.00	272
PPC		1	15	4,415	1.00	68.98	128		1	11	542	1.00	6.35	96
L-BRGC		1	11	4,415	1.00	51.74	96		1	7	542	1.00	4.23	64
DC	4460 (*ipc1*)	16	16	5,916	1.33	184.88	0	360 (*ipc1*)	16	16	1,816	5.04	56.75	0
SRGE		16	16	5,830	1.31	182.19	0		16	16	1,730	4.81	54.06	0
EIDC		6	7	62,023	13.91	787.40	104		3	3	1,268	3.52	7.43	48
EIGC		6	7	58,882	13.20	747.53	104		3	3	1,268	3.52	7.43	48
DIRPE		34	29	5,008	1.12	308.11	0		34	29	908	2.52	55.86	0
Bitmap		34	53	4,460	1.00	378.93	696		4	4	360	1.00	2.81	64
PPC		9	12	4,460	1.00	91.46	168		4	4	360	1.00	2.81	64
L-BRGC		5	10	4,460	1.00	65.33	120		2	2	360	1.00	1.41	32

TABLE IV
PERFORMANCE OF $acl1$ RULE TABLES

Scheme	# of Rules	**8** Extra Bits					**24** Extra Bits					**40** Extra Bits				
		Entry Size (bits)		# of TCAM entries	TCAM Size (Kb)	EF	Entry Size (bits)		# of TCAM entries	TCAM Size (Kb)	EF	Entry Size (bits)		# of TCAM entries	TCAM Size (Kb)	EF
		src	dst				src	dst				src	dst			
Bitmap	752 (*acl1*)	16	24	1,560	60.9	2.07	16	40	1,255	68.6	1.67	16	56	1015	71.4	1.35
PPC		16	24	1,560	60.9	2.07	16	40	1,255	68.6	1.67	16	56	1015	71.4	1.35
L-BRGC		16	24	1,479	57.8	1.96	16	40	1,113	60.9	1.48	16	56	937	65.9	1.25
Bitmap	4415 (*acl_5k*)	16	24	4,947	193.2	1.12	16	40	4,497	245.9	1.01	16	49	4,415	280.3	1.00
PPC		16	24	4,718	184.3	1.06	16	27	4,415	185.4	1.00	16	27	4,415	185.4	1.00
L-BRGC		16	23	4,415	168.2	1.00	16	23	4,415	168.2	1.00	16	23	4,415	168.2	1.00

TABLE V
PERFORMANCE OF $ipc1$ RULE TABLES

Scheme	# of Rules	8 Extra Bits				
		Entry Sizes (bits)		# of TCAM entries	TCAM Size (Kb)	EF
		src	dst			
Bitmap	1550 (*ipc1*)	19	20	1,550	59.03	1.00
PPC		19	20	1,550	59.03	1.00
L-BRGC		18	19	1,550	56.01	1.00
Bitmap	4460 (*ipc1_5k*)	19	19	4,460	165.51	1.00
PPC		19	19	4,460	165.51	1.00
L-BRGC		18	18	4,460	156.80	1.00

TABLE VI
PERFORMANCE OF $fw1$ RULE TABLES

Scheme	# of Rules	**8** Extra Bits				
		Entry Sizes (bits)		# of TCAM entries	TCAM Size (Kb)	EF
		src	dst			
Bitmap	269 (*fw1*)	18	18	269	9.46	1.00
PPC		18	18	269	9.46	1.00
L-BRGC		18	18	269	9.46	1.00
Bitmap	4653 (*fw1_5k*)	18	18	4,653	163.58	1.00
PPC		18	18	4,653	163.58	1.00
L-BRGC		18	18	4,653	163.58	1.00

times. The testing program is written in C running on a PC with an Intel i3-540 CPU (3.06 GHz). We can see that L-BRGC takes a longer time to preprocess and insert ranges because BRGC range sets have to be calculated for reducing the required code space. L-BRGC's preprocessing and update performance is acceptable because the rule table changes do not occur frequently. Also, the CoPTUA scheme proposed in [19] that does not lock TCAM during the updating process can be used to alleviate the impact of slow encoding process.

VI. CONCLUSION

In this paper, binary reflected Gray codes and the concept of elementary intervals are used to design a memory-efficient

TABLE VII
PREPROCESSING TIME (MILLISECONDS)

	fw1	acl1	ipc1
# of Rules	**269**	**752**	**1550**
DC	0.44	1.38	2.43
SRGE	0.46	1.43	2.52
EIDC	0.11	0.22	1.31
EIGC	0.13	0.27	1.08
DIRPE	0.18	0.53	1.26
Bitmap	0.3	2.19	8.34
PPC	0.3	0.19	0.16
L-BRGC	0.6	1.92	2.01

TABLE VIII
INCREMENTAL INSERTION TIME (MILLISECONDS)

	fw1	acl1	ipc1
# of Rules	**5000**	**5000**	**5000**
DC	8.47	6.32	7.54
SRGE	8.58	6.67	7.86
EIDC	2.13	1.14	4.00
EIGC	2.29	1.53	3.52
DIRPE	3.41	3.43	3.75
Bitmap	123.2	124.6	118.9
PPC	0.76	0.69	0.55
L-BRGC	15.02	11.72	10.65

TCAM encoding scheme. From the performance results experimented on real-life rule tables, the proposed *L-BRGC* scheme gives the best performance for $fw1$ and $ipc1$ tables, and the proposed *EIGC* scheme performs the best for the $acl1$ table. From the performance results of experiments on larger synthesized rule tables, the proposed L-BRGC scheme performs the best for all $fw1$, $acl1$, and $ipc1$ tables. From the experimental results based on DRES scheme, the proposed L-BRGC scheme also performs better than *PPC* and *bitmap* schemes.

REFERENCES

[1] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Norwell, MA: Kluwer, 1984.

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

[3] A. Bremler-Barr, D. Hay, and D. Hendler, "Layered interval codes for TCAM-based classification," in *Proc. IEEE INFOCOM*, 2009, pp. 1305–1313.

[4] A. L. Buchsbaum, G. S. Fowler, B. Krishnamurthy, K.-P. Vo, and J. Wang, "Fast prefix matching of bounded strings," *J. Exp. Algor.*, vol. 8, pp. 1–17, 2003, Article No. 1.3.

[5] Y.-K. Chang and Y.-C. Lin, "Dynamic segment trees for ranges and prefixes," *IEEE Trans. Comput.*, vol. 56, no. 6, pp. 769–784, Jun. 2007.

[6] H. J. Chao, "Next generation routers," *Proc. IEEE*, vol. 90, no. 9, pp. 1518–1558, Sep. 2002.

[7] 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.

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

[9] P. Gupta and N. McKeown, "Packet classification on multiple fields," *Comput. Commun. Rev.*, vol. 29, no. 4, pp. 147–160, 1999.

[10] T. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," *Comput. Commun. Rev.*, vol. 28, no. 4, pp. 203–214, 1998.

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

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

[13] H. Liu, "Routing table compaction in ternary CAM," *IEEE Micro*, vol. 22, no. 1, pp. 58–64, Jan.–Feb. 2002.

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

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

[16] N. Mohan and M. Sachdev, "Low power dual matchline ternary content addressable memory," in *Proc. IEEE ISCAS*, 2004, vol. 2, pp. 633–636.

[17] B. Schieber, D. Geist, and A. Zaks, "Computing the minimum DNF representation of boolean functions defined by intervals," *Discrete Appl. Math.*, vol. 149, no. 1–3, pp. 154–173, Aug. 2005.

[18] D. Taylor and J. Turner, "ClassBench: A packet classification benchmark," in *Proc. IEEE INFOCOM*, 2005, vol. 3, pp. 2068–2079.

[19] Z. Wang, H. Che, M. Kumar, and S. K. Das, "CoPTUA: Consistent policy table update algorithm for TCAM without locking," *IEEE Trans. Comput.*, vol. 53, no. 12, pp. 1602–1614, Dec. 2004.

[20] F. Yu and R. H. Katz, "Efficient multi-match packet classification with TCAM," in *Proc. IEEE Symp. High Perform. Interconnects*, 2004, pp. 28–34.

Yeim-Kuan Chang received the Ph.D. degree in computer science from Texas A&M University, College Station, in 1995.

He is currently a Professor with the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan. His research interests include Internet router design, computer architecture, and multiprocessor systems.

Cheng-Chien Su received the M.S. and Ph.D. degrees in computer science and information engineering from National Cheng Kung University, Taiwan, in 2005 and 2011, respectively.

His research interests include high-speed packet processing in hardware and deep packet inspection architectures.

Yung-Chieh Lin received the M.S. degree in computer science and information engineering from National Cheng Kung University, Taiwan, in 2005, and is currently pursuing the Ph.D. degree in computer science and information engineering at National Cheng Kung University.

His current research interests include high-speed networks and high-performance Internet router design.

Sun-Yuan Hsieh received the Ph.D. degree in computer science from National Taiwan University, Taipei, Taiwan, in 1998.

From August 2000 to January 2002, he was an Assistant Professor with the Department of Computer Science and Information Engineering, National Chi Nan University, Taiwan. In February 2002, he joined the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, where he is now a Distinguished Professor. His current research interests include design and analysis of algorithms, fault-tolerant computing, bioinformatics, parallel and distributed computing, and algorithmic graph theory.