Computer Networks 164 (2019) 106898

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

A power-saving pre-classifier for TCAM-based IP lookup ☆

Wenjun Li [a,b,*], Dagang Li [a,c], Xinwei Liu [a], Ting Huang [a], Xianfeng Li [a], Wenxia Le [d], Hui Li [a,b]

[a] School of Electronic and Computer Engineering, Peking University, Shenzhen, China
[b] Peng Cheng Laboratory, Shenzhen, China
[c] PKU-HKUST ShenZhen-HongKong Institution, Shenzhen, China
[d] Network Energy Department, Huawei Technologies Co., Ltd, Shenzhen, China

ARTICLE INFO

Article history:
Received 5 November 2018
Revised 2 August 2019
Accepted 7 September 2019
Available online 9 September 2019

Keywords:
IP routing table lookup
TCAM
Range encoding
Power reduction
Memory efficient

ABSTRACT

Ternary Content Addressable Memory (TCAM) is widely used for designing high-throughput forwarding engines on most of today's high-end routers. Despite its capability for line-speed queries, it is very power hungry and space inefficient. By making use of a pre-classifier to activate TCAM blocks selectively, MEET-IP, a recently proposed TCAM based IP lookup scheme, significantly improves the utilization of TCAMs. However, it suffers from performance degradation because it uses a two-level pre-classifier. In this paper, we propose SplitIP, a memory and power efficient TCAM-based scheme for IP routing table lookup. We first transform the IP lookup problem to a point location problem through a routing table projection. Based on the projection, we propose a top-down splitting algorithm to separate routing table prefixes evenly into TCAM blocks. Finally, a simpler one-level classifier is constructed for fast pre-classification using improved range encoding techniques. The top-down prefix partitioning algorithm combined with the database independent encoding scheme provides an incremental update for SplitIP. Experimental results show that our design achieves more than 97% power reduction with a TCAM storage overhead of less than 3% on average.

1. Introduction

IP routing table lookup is one of the critical issues in designing high-performance routers. It is a challenging problem due to the following aspects: (1) the size of the routing table has become more than half a million and keeps growing [40]; (2) cloud computing and network applications are pushing the line rate of core routers to 400Gbps or even higher; (3) one IP address may match multiple prefixes when performing the Longest Prefix Matching (LPM); (4) the deployment of IPv6 will lead to larger routing tables with longer prefixes. With the exponential growth of the Internet, these challenges become even stronger than ever, therefore, efficient and scalable solutions for IP lookup are still under active investigation.

Numerous IP routing table lookup techniques have been proposed in the past twenty years [16,39]. They can be categorized broadly into two major approaches: software-based algorithmic solutions and hardware-based architectural solutions. Algorithmic solutions using ordinary memories and commercial processors have been well studied because they are cheap and flexible. However, despite extensive research efforts, most of them are memory and performance inefficient, falling short of the needs of high-speed networks. Thus, architectural solutions using special hardware, such as TCAM and FPGA, have become the de-facto route in the industry for designing high-throughput forwarding engines on high-end routers. Among the hardware used in major architectural solutions, Ternary Content Addressable Memory (TCAM) is regarded as a promising device and has been widely deployed in existing network devices.

TCAM is a fully associative memory that allows a "don't care" state to be stored in each memory cell in addition to 0s and 1s. This feature makes it particularly attractive for packet classification and routing table lookup that require longest prefix matches [2]. When a destination address is presented to the TCAM, each TCAM entry is searched in parallel and the longest matching address prefix is returned. Thus, a single TCAM access is sufficient to perform a routing table lookup. Moreover, TCAM-based table is much simpler to manage and update than those implemented

☆ The preliminary version of this paper titled "MEET-IP: Memory and Energy Efficient TCAM-based IP Lookup" was published in the proceedings of the 26th International Conference on Computer Communications and Networks (ICCCN), Vancouver, Canada, July, 2017 [56].

* Corresponding author at: School of Electronic and Computer Engineering, Peking University, Shenzhen, China.
E-mail addresses: wenjunli@pku.edu.cn (W. Li), dgli@pku.edu.cn (D. Li), lxw0724@pku.edu.cn (X. Liu), huangting53@pku.edu.cn (T. Huang), lixianfeng.sz@pku.edu.cn (X. Li), lewenxia@huawei.com (W. Le), lih64@pku.edu.cn (H. Li).

https://doi.org/10.1016/j.comnet.2019.106898
1389-1286/

using SRAM/DRAM [23,37,48]. Despite its high speed and ease of management, TCAM has its own limitations with respect to IP routing table lookup. First, TCAM has limited capacity and is not expected to increase dramatically in the near future. Second, TCAM chips consume large amount of power. For example, a typical 1Mb TCAM chip can consume up to 15-30 Watts of power, which is roughly 150 times more power per bit than SRAM [30]. Third, TCAM is very expensive and normally contributes a significant fraction to the total cost of network devices [10].

In recent years, leading TCAM vendors have provided block-based TCAM designs, where the TCAM device is partitioned into fix-sized blocks, and a subset of them can be activated for lookups when needed. This improved design provides a good substrate for potential power reduction. Many research efforts exploit this feature to optimize the power consumption of TCAM-based forwarding engines. Authors of CoolCAMs [13] proposed an architecture to partition the entire routing table into multiple sub-tables or buckets, where each bucket is laid out over one or more TCAM blocks. SmartPC [65], one of well-known works targeting blocked TCAM, introduced a pre-classifier with the idea of expanding and combining original classifier rules based on their IP address locations. However, these efforts achieve power saving at the expense of poor utilization of TCAM capacity (i.e., idle memory holes in the TCAM blocks), and the potential of power reduction is not fully exploited in many cases. In order to support incremental updates, the author of paper [61] proposed a new partitioning algorithm based on the prefix comparison, which can split the routing table into a fixed number of buckets as evenly as possible, but it may suffer from performance degradation because of its binary search based pre-classifier. Recently, GreenTCAM [59] achieves more energy reduction and solves the problem of poor TCAM utilization in SmartPC, yet it cannot be well applied to IP lookup. MEET-IP, the preliminary version of this paper, significantly improves the utilization of TCAMs using a top-down pre-classifier, but its two-level pre-classifier is rather complicated performance-wise. Therefore, although the 2-stage framework with pre-classifier is a promising solution for power-efficient TCAMs, as far as we know, existing approaches still lack of a smart pre-classifier which can make excellent balance among power consumption, memory utilization, search performance and update performance.

In this paper, we propose SplitIP, a 2-stage TCAM-based scheme for IP routing table lookup. In the first stage, each incoming IP address is classified by a one-level pre-classifier (or index table), which provides information on which TCAM block to be activated in the next stage; in the second stage, the TCAM block from the first stage is activated and searched for a match. We identified three main challenges when designing the pre-classifier of our schemes: (1) minimum TCAM holes when accommodating large number of table entries; (2) effective mapping of pre-classifier entries onto TCAM blocks; (3) quick index search mechanism during the pre-classification. To solve these problems, we first transform the LPM problem to a point location problem through routing prefixes projection. After then, we propose a global top-down partition algorithm to effectively separate the projected prefixes evenly into TCAM blocks. Finally, a range encoding based pre-classifier is used to quickly select the right TCAM blocks to activate. The simple top-down prefix projection makes SplitIP much easier to achieve efficient partitioning and to support fast incremental updates. Experimental results show that our design achieves more than 97% power reduction with a TCAM storage overhead of less than 3% on average.

The rest of the paper is organized as follows. Section 2 briefly summarizes the related work and challenges. Sections 3 and 4 present the technical details of our work. Section 5 provides our experimental results. Finally, Section 6 draws conclusions on this work.

2. Related work and challenges

A lot of software-based algorithmic approaches have been proposed for IP lookup in the past twenty years. In general, they can be categorized into trie-based algorithms [14,20,28,42,45–47,49,50,52,54], hash-based algorithms [11,17,24,44] and range-based algorithms [7,29,66]. However, most approaches still fall short of providing satisfactory lookup performance. Thus, architectural solutions using special hardware have been wildly studied in recent years. Based on their implementation platforms, we can categorize prior works into TCAM-based algorithms [12,22,23,25,26,38], FPGA based algorithms [17,27] and GPU-based algorithms [41,64]. Among them, TCAM has been widely used for IP routing table lookup and packet classification because of its ability to allow a “don’t care” state and process packet at line-speed. Moreover, TCAM-based solutions are much easier to manage than other hardware-based solutions.

Despite these advantages, this brutal force hardware solution is not only expensive and space inefficient, but also very power-hungry. Besides, TCAM also suffers from severe expansion problem when storing ranges. During the past decade, a lot methods and algorithms had been proposed to alleviate these problems, such as classifier minimization [3–6] and range encoding [1,15,18,21,31,35,58,62,63]. To reduce the power consumption of TCAMs, the latest TCAM devices from leading vendors come with a power saving mechanism where a subset of its TCAM blocks can be selectively activated. A TCAM block is a contiguous, fixed-size chunk of TCAM entries, much smaller than the size of the entire TCAM. Recent research efforts exploit this feature to construct a 2-stage power-efficient TCAMs and introduce a pre-classifier to activate TCAM blocks selectively [9,13,56,59,65]. In this paper, we propose a more efficient pre-classifier called SplitIP by combining block-based TCAM designs with range-based lookup algorithms in a novel way, so that TCAM solutions can achieve huge power reduction without heavy TCAM storage waste. Next, we give a more detailed review on several representative 2-stage power-efficient TCAMs.

2.1. 2-stage Power-efficient TCAMs

CoolCAMs [13] is one of the most representative 2-stage power-efficient TCAM-based routing table lookup algorithms. The key idea for the CoolCAMs architecture is to partition the entire routing table into multiple sub-tables or buckets, where each bucket is laid out over one or more TCAM blocks. Two different table-splitting schemes have been proposed, which are bit selection architecture and tire-based table partitioning. Bit selection architecture provides a straightforward technique for power reduction with simple extra hardware requirement. Based on observation that most prefixes in core routing tables are between 16 and 24 bits long, the bit selection architecture puts the special prefixes into a set of TCAM blocks to search on every lookup, and the remaining prefixes are partitioned into buckets which is selected by hashing for each lookup. However, this technique has some drawbacks such as terrible worst-case bounds and unsustainable assumptions about prefix distribution. To improve on these drawbacks, tire-based table partitioning algorithm has been proposed, using a prefix trie to get the ID of the right TCAM bucket. Two different partitioning schemes have been proposed: subtree-split and post-order split. During the partitioning step of the subtree-split algorithm, the prefix trie is traversed in post order looking for a covering node, whose count is at least half of the TCAM bucket size. The drawback of subtree-split algorithm is that the smallest and largest bucket sizes vary by as much as a factor of 2, leading to poor memory utilization with a lot of holes in the TCAM blocks. Assuming a TCAM block size of 4, Fig. 1 shows a partition example of subtree-split based

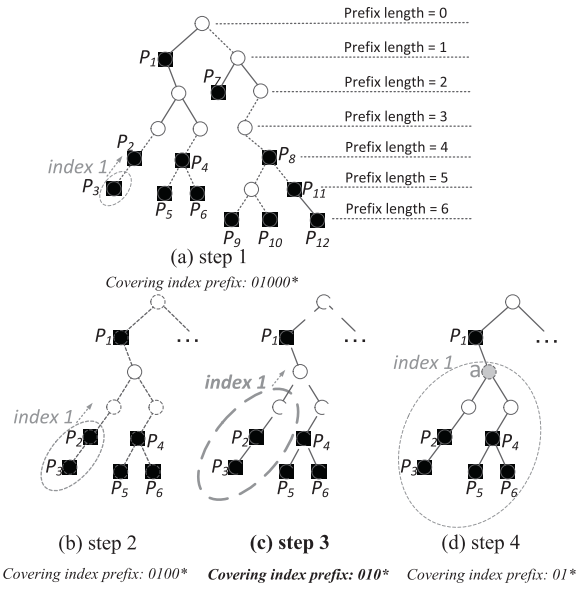


Fig. 1. A partition example of CoolCAMs (subtree-split algorithm).

CoolCAMs. It first selects the leftmost bottom prefix P_3 as the initial covering index prefix, and then tries to cover more prefixes by doubling the covering range (i.e., *index* 1) each time. When *index* 1 tries to cover prefix node *a* in step 4, three prefixes (i.e., P_4, P_5, P_6) should be included simultaneously, which will cause an overflow of a TCAM block. Thus, only two prefixes in step 3 (i.e., P_2, P_3) can be stored in the TCAM block, leaving the remaining two entries in idle. Post-order split algorithm eliminates the TCAM holes through even routing table partitioning, where each partitioned bucket is constructed from a collection of subtrees, rather than a single subtree as in the case of subtree-split algorithm. However, such an even partition comes at the extra cost of a larger number of entries in the index TCAM which are used to store indexes in the first stage pre-classifier (at most $W+1$ index entries for one TCAM block, where W is the maximum prefix length in the routing table). With the continuous growth of routing table size and gradual deployment of IPv6, the problem of index TCAM storage overhead will become more severe. MEET-IP, a recently proposed 2-stage power-efficient TCAMs for IP lookup, significantly improves the utilization of TCAMs by constructing a top-down pre-classifier for routing tables. In MEET-IP, each incoming address is presented to the index TCAM for retrieving the best matching prefix, followed by some extra comparisons in RAM. Thus, its two-level pre-classifier may not be favorable performance-wise.

SmartPC [65], another well-known 2-stage power-efficient TCAM scheme, introduces a pre-classifier for packet classification, where each incoming packet is forwarded according to a set of multi-field rules. The basic idea for the pre-classifier construction is to divide the whole source-destination dimensional space into some non-intersecting sub-spaces, each covering a subset of rules inside it. In SmartPC, each pre-classifier entry is constructed by expanding and combining original classifier rules based on their source-destination address locations. More exactly, it starts with a randomly selected free rule (not yet included in any pre-classifier entry), and the initial sub-space covered by this entry corresponds to the rectangle formed by the source and destination fields of the selected rule. Then, the sub-space continues its space expansion until the subset of rules completely covered by the current sub-space reaches the limit of the TCAM block size, and rules partly intersected with the current sub-space will be regarded as general rules to search for every lookup. When the sub-space expansion process completes, a pre-classifier entry can be generated for the sub-space, and a corresponding TCAM block is allocated for the contained subset of rules. Clearly, this random bottom-up way of generating pre-classifier lacks the global view on the relationship of rules, leading to a large number of general rules and TCAM holes. Recently, GreenTCAM [59] is proposed to solve the problem of poor utilization of TCAM capacity suffered in SmartPC. Based on the observation that most classifier rules contain at least one 'small' IP address field [55,57], it separates the original classifier table into several sub-tables and builds pre-classifier for each sub-table. GreenTCAM achieves a 93.6% energy reduction with a TCAM storage overhead of 5.6% on average, much better than that of SmartPC. However, its two-level pre-classifier may also lead to performance degradation. Besides, it cannot be well applied to IP lookup.

2.2. Challenges and motivation

According to the discussions above, the 2-stage framework is a promising approach for power-efficient TCAMs, where the first stage is a pre-classification and the second is a selective post-classification. With the help of pre-classification, only a few TCAM blocks containing the candidates will be activated in the second stage, rather than comparing against all TCAM entries blindly. Under this framework, the critical challenge is how to build an effective pre-classifier, such that the original table can be well accommodated into TCAM blocks as compact and efficient as possible. At the same time, since the pre-classifier itself also needs to be stored and searched, the pre-classifier itself should be small and well suited for fast pre-classification.

Currently, most existing pre-classifiers are built in a locally greedy heuristic way, and their bottom-up approach restricts their ability for global optimization. In this paper, we propose a top-down framework, which can process the prefixes from a global view to achieve a more effective pre-classifier. The whole framework includes global address projection, block splitting and range encoding. More details will be given in the following two sections.

3. SplitIP: a 2-stage top-down framework

In this section, we present SplitIP, a memory and power efficient scheme for TCAM based IP routing table lookup, which can also be applied to IPv6. We first transform the LPM problem into a point location problem through routing table projection, generating a set of elementary intervals from the address prefixes. Based on these elementary intervals, we propose a top-down splitting algorithm that can evenly separate routing prefixes into TCAM blocks with minimum memory holes and indexing TCAM entries, much less than in CoolCAMs and SmartPC. Finally, a simpler one-level pre-classifier based on range encoding techniques is constructed for the TCAM blocks, where each index range can be split into two sub-ranges, and adopts different encoding techniques adaptively to reduce edcoded TCAM entries. For the convenience of description and comparison, the example of a 7-bit routing table containing 12 address prefixes from Table 1 is used, which was also used in CoolCAMs.

3.1. LPM Problem → point location problem

A network address prefix is commonly indicated as a pair of *address/prefix length*, where only the most significant bits indicated by the prefix length are used for matching, and the rest least wildcard bits (*) are simply ignored. In essence, each address prefix corresponds to a range of values, where the lower bound and upper bound can be obtained by setting the wildcards to 0s and 1s respectively, as shown in the column of prefix ranges in Table 1.

Table 1
A sample routing table.

Routing table	Prefix database (7 bits in length)		Next Hop
	Prefix bit strings	Prefix ranges	
P_1	*0*******	[*0, 63*]	*hop1*
P_2	*0100****	[*32,39*]	*hop2*
P_3	*01000***	[*32, 35*]	*hop3*
P_4	*0110****	[*48, 55*]	*hop4*
P_5	*01100***	[*48, 51*]	*hop5*
P_6	*01101***	[*52, 55*]	*hop6*
P_7	*10******	[*64, 95*]	*hop7*
P_8	*1101****	[*104, 111*]	*hop8*
P_9	*110100**	[*104, 105*]	*hop9*
P_{10}	*110101**	[*106, 107*]	*hop10*
P_{11}	*11011***	[*108, 111*]	*hop11*
P_{12}	*110111**	[*110, 111*]	*hop12*

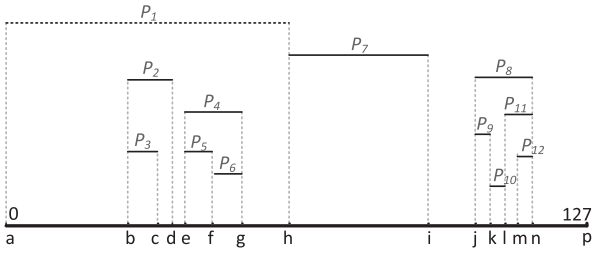


Fig. 2. Routing table projection from a geometric point of view.

The range of an address prefix may be contained by the range of another address prefix, if and only if the latter is a prefix of the former. For example, by projecting the address prefixes in Table 1 onto a number line, we find that P_2, P_3, P_4, P_5 and P_6 are all contained by P_1, as the prefix *0** of P_1 is a prefix of P_2, P_3, P_4, P_5 and P_6. Because of this property of the routing table, an IP address may match multiple prefixes. In this case, the longest prefix, which is most specific range containing the IP address, will be the best match. Algorithms finding the best match are thus called the Longest Prefix Matching (LPM) algorithms.

With the projection of address prefixes in Fig. 2, a set of endpoints (i.e., *a, b, c…, m, n, p*) can be identified to split the whole number line into a set of contiguous and disjoint intervals. These intervals are called *elementary intervals* which can be defined formally as follows:

Definition 1. (Elementary intervals): For W-bit address space of 0 to 2^W-1, it can be divided into a set of k elementary intervals $E = \{X_i \mid X_i = [l_i, r_i]$, for $i = 1$ to $k\}$, E satisfy the following:

(1) $l_1 = 0$ and $r_k = 2^W-1$,
(2) $r_i = l_{i+1} - 1$ for $i = 1$ to $k - 1$.

By introducing elementary intervals, the routing table lookup can be viewed as a problem of point location among a set of contiguous and non-overlapping elementary intervals. Therefore, given an incoming IP address, the first step is to decide the elementary interval where the IP address is located. Unlike the original LPM problem, there will only be one unique matching for this IP address in terms of elementary interval. The second step is to decide the best address prefix from all prefixes that cover the elementary interval obtained from the first step. For example, with an IP address of *0100001*, the elementary interval [*b, c*] in Fig. 2 is selected, then among the three prefixes P_1, P_2 and P_3 covering [*b, c*], the best matching prefix is P_3 as it is the most specific prefix for [*b, c*], and in turn the most specific prefix for IP address *0100001*. The correctness of this two-step algorithm comes from an important fact: all address values (points) in one elementary interval share the same set of address prefixes covering the whole interval. That is why we can turn the LPM problem into an equivalent point location problem. Fig. 3 and Table 2 show all elementary intervals and corresponding matching prefixes for routing table projection in Fig. 2.

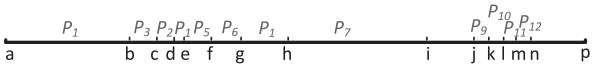


Fig. 3. LPM problem → point location problem.

Table 2
Elementary intervals and corresponding prefixes.

Elementary intervals	Corresponding matching prefixes	
	Covered prefixes	Best matching prefix
[*0, 31*]	P_1	P_1
[32, 35]	P_1, P_2, P_3	P_3
[*36, 39*]	P_1, P_2	P_2
[*40, 47*]	P_1	P_1
[*48, 51*]	P_1, P_4, P_5	P_5
[*52, 55*]	P_1, P_4, P_6	P_6
[*56, 63*]	P_1	P_1
[*64, 95*]	P_7	P_7
[*96, 103*]	*NULL*	*NULL*
[*104, 105*]	P_8, P_9	P_9
[*106, 107*]	P_8, P_{10}	P_{10}
[*108, 109*]	P_8, P_{11}	P_{11}
[*110, 111*]	P_8, P_{11}, P_{12}	P_{12}
[*112, 127*]	*NULL*	*NULL*

Based on this transformation, an elementary interval based splitting algorithm for routing tables can be developed, which can separate prefixes evenly into blocks without any TCAM holes (idle entries). Intuitively, one dimensional partition problem based on elementary intervals is often easier to handle than the original 32-level separating problem, as we will see in the next subsection.

3.2. Top-down splitting algorithm

One of the key issues in power-efficient routing table lookup is how to separate all routing prefixes into a set of TCAM blocks and build a pre-classifier for these blocks. In this subsection, we detail an elementary interval based top-down splitting algorithm, with each split interval containing at most *TBsize* prefixes, where *TBsize* is the TCAM block size (except possibly the last one subset).

The rationale behind our routing table partitioning algorithm is simple: by transforming addresses into a set of elementary intervals, we can simply generate index ranges for TCAM blocks by merging neighboring elementary intervals recursively. More specifically, we first select the leftmost elementary interval in the number line (not included in any existing index range) as the first index range. Then, this index range is expanded by moving its right end point to another right-side projection endpoint, with exactly one more elementary interval included or completely covered each time, until the subset of prefixes included by the current index range reaches the capacity limit of a TCAM block (i.e., one more included elementary interval should cause block overflow). At the same time, a corresponding TCAM block can be allocated for the subset of prefixes that are included. For all included prefixes in each TCAM block, they need to be stored in descending order of their priorities. These two steps will be repeated until all elementary intervals are processed. It is easy to see that each generated index range is a combination of several consecutive and non-overlapping smaller elementary intervals.

Take elementary intervals of Fig. 3 as an example to describe the above merging method intuitively (assuming the *TBsize* = 4). First, the left-most elementary interval [*a, b*] is selected, and the best matching prefix P_1 is included by the initial index range. In

Table 3
Index range table.

Index ranges	Included elementary intervals	TCAM block blocks
[0, 51]	[0, 31], [32, 35], [36, 39], [40, 47], [48, 51]	1^{st} block: {P_3, P_5, P_2, P_1}
[52, 105]	[52, 55], [56, 63], [64, 95], [96, 103], [104, 105]	2^{nd} block: {P_9, P_6, P_7 ,P_1}
[106, 127]	[106, 107], [108, 109], [110, 111], [112, 127]	3^{rd} block: {P_{10}, P_{12}, P_{11}}

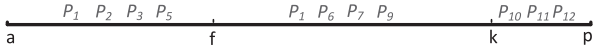


Fig. 4. Routing table partition with a TCAM block size of 4.

order to accommodate more prefixes, the index range [*a, b*] then moves its right end point from endpoint *b* to *c*, and so the index range becomes [*a, c*]. At this point, two elementary intervals are completely covered by the new index range containing two best matching prefixes (i.e., P_1 and P_3). Since the number of included prefixes is still smaller than the block size, the current index range continues to move its right end point to produce a larger new index range, which may cover more not-yet-included elementary intervals. When the index range moves its right end point to endpoint *f*, the new index range [*a, f*] will include five elementary intervals containing four unique best matching prefixes (i.e., P_1, P_2, P_3, P_5), reaching the capacity limit of a TCAM block. Thus, the first index range expansion process stops at endpoint *f*, and the index range [*a, f*] is generated with four best matching prefixes stored in its corresponding TCAM block.

After that, a new index range expansion process repeats the steps above by setting the index range to be the left-most not-yet-included elementary interval, until all remaining elementary intervals are processed. Table 3 and Fig. 4 show all index ranges and the elementary intervals they include, as well as the corresponding TCAM blocks.

As can be seen from Table 3, the routing table can be evenly partitioned into three TCAM blocks, with each block except the last one containing as many prefixes as a TCAM block can hold. This fine characteristic derives from the simple one-dimensional merging method which can include at most one prefix at a time, rather than an exponential order of prefixes as in CoolCAMs using subtree-split algorithm. Essentially, our splitting algorithm is derived from the global top-down partitioning method (i.e., routing table projecting), with more insights into the prefix relationship. As described above, all address values in an elementary interval share the same set of address prefixes covering each of them, therefore, at most one prefix can be included by adding an elementary interval each time, achieving a more precise control over the expansion.

It is not difficult to see that a prefix can be separated into multiple blocks, because it may span multiple index ranges (e.g., P_1 in Table 3), and such duplicated prefixes may introduce additional storage overhead. Fortunately, in real routing tables, the proportion of such duplicated prefixes is almost negligible, which can be verified from our following experimental results. Essentially, this feature comes from a well-known observation about prefix length distribution, that is, there are very few routes with prefixes longer than 24-bit [36,47,66]. For ease of evaluation, we can define this prefix replication factor as *blocking replication*, which can be obtained by the following formula: *(routing table size + #duplicated rules)/routing table size*. Thus, the blocking replication of our sample routing table is 13/12. More evaluation results for real routing tables can be seen in Section 5.3. The pseudo code for above routing table splitting and pre-classifier construction algorithm is shown in Fig. 5.

Algorithm 1: BuildPreClassifier()

Input: Elementary intervals & matching prefixes
Output: Index table & partitioned routing tables
1: **while** (unused elementary intervals != null)
2: covering range = the leftmost unused elementary interval
3: **while**(number of covering prefix < block size
and unused elementary intervals != null)
4: add leftmost unused elementary interval to covering range
5: **endwhile**
6: put covering prefixes in new TCAM block (descending order)
7: put covering range in pre-classifier/index table
8: **endwhile**

Fig. 5. Elementary interval based routing table splitting algorithm.

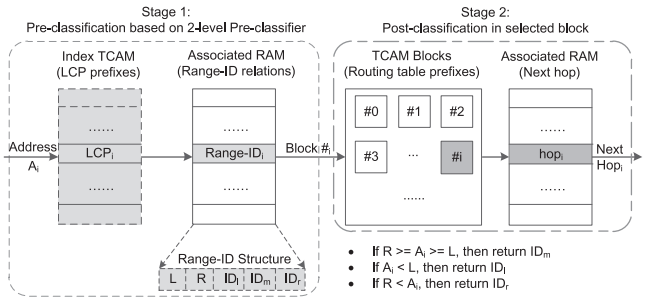


Fig. 6. The architecture of MEET-IP.

3.3. One-level pre-classifier based on encodings

A pre-classifier can now be constructed with the consecutive and non-overlapping index ranges generated from the above splitting algorithm to quickly classify incoming IPs into TCAM blocks for high-speed post-classification. For each incoming destination address, a pre-classifier should be able to return an index ID that explicitly indicates the right TCAM block to obtain the best matching result. The key issue is how to perform a high-speed lookup in the first pre-classification phase.

Based on the concept of *Longest Common Prefix* (*LCP*), a two-level TCAM pre-classifier is proposed in MEET-IP as illustrated in Fig. 6. In MEET-IP, each incoming address is presented to the index TCAM for retrieving the best matching *LCP*, followed by some extra comparisons in RAM. According to the ID obtained from pre-classification, a second TCAM lookup is performed on its corresponding TCAM block, where the longest matching prefix can be obtained with the next hop information in its associated RAM. More details can refer to [56,60].

Although MEET-IP produces just one index for each TCAM block, its complicated two-level pre-classifier may result in performance degradation. Based on the observation that the number of generated index ranges is much less than routing table size, we can simply encode each index range into an equivalent set of prefixes. Thus, we introduce a one-level pre-classifier based on the following described SplitCoding, which is a practical TCAM range encoding scheme. Fig. 7 shows the architecture of SplitIP.

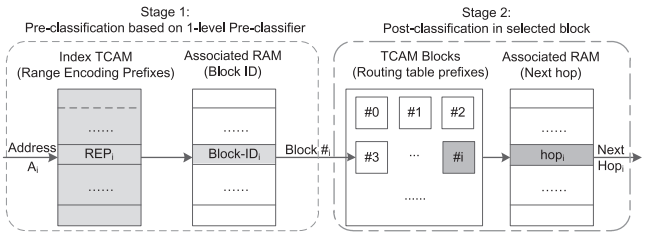
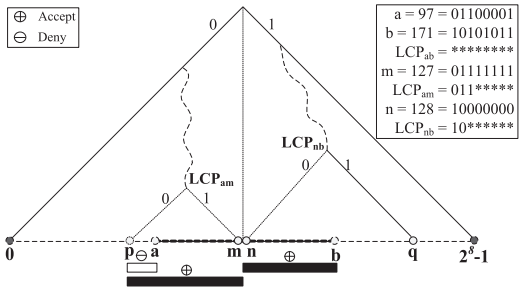


Fig. 7. The architecture of SplitIP.



(c) Proposed SplitCoding:
*{01100000 → deny, 011***** → accept, 100***** → accept, 10100*** → accept, 101010** → accept}*

Fig. 8. A sample of SplitCoding for 8-bit range R = [97, 171].

4. SplitCoding for fast pre-classification

In this section, we introduce a simple TCAM range encoding scheme called SplitCoding, which is not only better in terms of range expansion but also easier to understand and much more practical to implement. Instead of adopting complex recursive algorithms as in previous approaches, all encoded prefixes in SplitCoding can be easily obtained in linear time by counting 0/1-bit of the range endpoints. More background about internal, external and optimal range encodings can refer to [31–35,53]. In addition, we prove that the upper-bound on the TCAM worse-case range expansions in SplitIP is $K(W-log_2K+1)$, better than $K(W+1)$ in CoolCAMs, where K is the number of index ranges (i.e., #used TCAM blocks) and W is the bit width of addresses in SplitIP. For the convenience of description, we use the 8-bit range R_{ab} = [97, 171] as a discussion example in this section as shown in Fig. 8.

4.1. Notations

Definition 2. (Range, width, length): A range is a contiguous interval of integers [*a, b*], where each integer is of *W* bits wide and its two range endpoints satisfy $a \le b$. For a range $R_{ab} = [a, b]$, its range length $L = \log_2(b-a+1)$. For example, for an interval R_1 = [80, 1024] in IPv4 address space, we can say that R_1 is a 32-bit range of length 10.

Definition 3. (Longest Common Prefix: *LCP*): LCP_{ab} is the longest prefix that covers the range $R_{ab} = [a, b]$. If prefixes are illustrated in the binary trie, LCP_{ab} is the lowest common ancestor of *a* and *b*. As illustrate in Fig. 8, for the 8-bit range R_{am} = [97, 127] = [*01100001, 01111111*], its longest common prefix $LCP_{am} = R_{pm}$ = 011*****, where endpoint *P* is the leftmost value of LCP_{am}.

Definition 4. (Splitting endpoint): For rang $R_{ab} = [a, b]$, there are two *splitting endpoints* for R_{ab} (one if $a = b$), where the value of these endpoints are the middle values of LCP_{ab}. For the 8-bit range R_{ab} = [97, 171] = [*01100001, 10101011*], the two middle values of LCP_{ab} are 127 and 128, because LCP_{ab}= ********. Thus, the *splitting endpoints* for range R_{ab} are *m* and *n*, where *m* = 127 and *n* = 128.

Definition 5. (Extremal range, generalized extremal range): A *W*-bit range $R_{ab} = [a, b]$ is called *extremal range* if $a = 0$ or $b = 2^W-1$. More broadly, R_{ab} is called *generalized extremal range* if integer *a* is the leftmost value of LCP_{ab}, or integer *b* is the rightmost value of LCP_{ab}. For example, $R_{0(a-1)} = [0, a-1]$ is an *extremal range* in Fig. 8, while $R_{am} = [a, m]$ and $R_{nb} = [n, b]$ are *generalized extremal ranges*, because *m* and *n* are the rightmost and leftmost values for LCP_{am} and LCP_{nb}, respectively.

4.2. Related encodings

Internal encoding: In internal encoding scheme, the *W*-bit range can be completely divided into at most *2W−2* successive and contiguous sub-ranges, where each sub-range can be represented as a prefix with an action of *Accept* [53]. Take the 8-bit range R_{am} = [97, 127] shown in Fig. 8 as an example, it can be encoded into following five 8-bit prefixes: {*0111****, *01101***, *011001***, *0110001**, *01100001*} → *Accept*. A sample implementation of the internal encoding scheme can refer to the Fig. 2 shown in paper [59].

External encoding: By exploiting the characteristic of *order of the entries* in TCAMs (i.e., output the first matching entry), the external encoding scheme can reduce the worst-case number of encoded prefixes from *2W−2* to *W* [35]. For each range using external encoding scheme, its complementary ranges are encoded into prefixes with an action of *Deny*, following with an *Accept* prefix. Then, it choose the better one compared with the internal encoding scheme. Also take the range R_{am} shown in Fig. 8 as an example, it can be encoded into two prefixes: *01100000* → *Deny* and *011***** → *Accept*, better than that in internal encoding scheme.

4.3. Related lemmas

Before describing the key steps of SplitCoding, we first give two lemmas which have been stated and proved in previous paper [8].

Lemma 1. *The smallest number of ternary entries that span range [0, b] is equal to the number of 1-bits in (b+1). These entries can be simply derived in linear time by respectively replacing each 1-bit in (b+1) with 0-bit and padding all the following bits with '*'. More details can refer to[9].*

Take the *extremal range* [0, 96] in Fig. 8 as example. With internal encoding, it can be encoded into minimally three prefixes, as there are three 1-bits in *01100001*, the binary representation of *97*, and the three encoded prefixes are *00*******, *010***** and *01100000*.

Lemma 2. *For any a > 0, the smallest number of ternary entries that span range [a, 2^W-1] is equal to the number of 0-bits in (a−1). These entries can be simply derived in linear time by respectively replacing each 0-bit in (a−1) with 1-bit and padding all the following bits with '*'. More encoding details can refer to[9].*

Take the *extremal rangeR* = [172, 2^8-1] in Fig. 8 as example. Using internal encoding, it can be encoded into minimally three prefixes, becasue there are three 0-bits in *10101011*, the binary representation of 171, and the three encoded prefixes are *11******, *1011**** and *101011***.

4.4. Key encoding steps

We now introduce the three key steps of SplitCoding for the general range $R = [a, b]$, where $a < b$. The encoding details for the sample 8-bit range $R = [a, b]$ = [97, 171] from Fig. 8 are illustrated in Fig. 9. The pseudo code for SplitCoding is shown in Fig. 10.

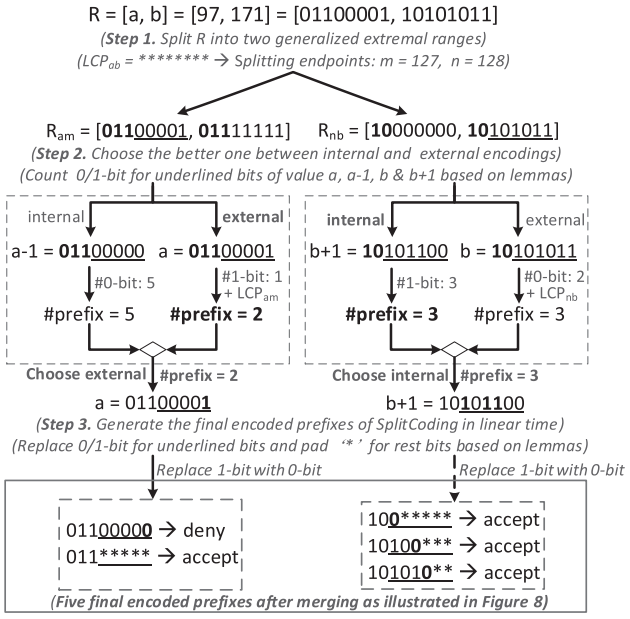


Fig. 9. Range encoding example using SplitCoding.

Algorithm 2: SplitCoding()

Input: Range $R = [a, b]$, a < b
Output: Encoded prefixes for R
1: compute LCP_R to generate *splitting endpoints* of R
2: split R into R_l and R_r from *splitting endpoints*
3: **for** left *generalized extremal range* R_l
3.1: Excluding the non-don't-care common bits in the LCP_{Rl}
3.2: compute the number of 0-bits in *a-1* (#0-bit)
3.3: compute the number of 1-bits in *a* (#1-bit)
3.4: **if** (#0-bit > #1-bit) {external encodings for R_l *based on* Lemma 1}
3.5: **else** {internal encodings for R_l *based on* Lemma 2}
4: **for** right *generalized extremal range* R_r
4.1: Excluding the super-prefix from the LCP_{Rr}
4.2: compute the number of 1-bits in *b+1* (#1-bit)
4.3: compute the number of 0-bits in *b* (#0-bit)
4.4: **if** (#1-bit > #0-bit) {external encodings for R_l *based on* Lemma 2}
4.5: **else** {internal encodings for R_r *based on* Lemma 1}
5: **Return** encoded prefixes in steps 3 & 4

Fig. 10. The pseudo code of SplitCoding.

Step 1 If R is a generalized extremal range, skip to Step 2. Otherwise, split R into two generalized extremal ranges.

Take the range $R = [a, b] = [97, 171]$ as an example, we first compute the *LCP* for R that is $LCP_{ab} = ********$. Then, we generate two *splitting endpoints* for R: $m = 01111111$ and $n = 10000000$. Finally, based on these two *splitting endpoints*, we can split R into two sub-ranges that are R_{am} and R_{nb} as shown in Fig. 8, where each of the split sub-ranges is a *generalized extremal range*.

Step 2 For each split *generalized extremal range*, choose between internal and external encoding by counting 0/1-bit of range endpoints (excluding the non-don't-care common bits in its corresponding *LCP*).

Take the left *generalized extremal range*$R_{am} = [97, 127]$ as an example. We can first encode the external part of R_{am} under the range of LCP_{am} – that is $R_{p(a-1)}$ as shown in Fig. 8. Although the range R_{am} and $R_{p(a-1)}$ are not *extremal ranges* as described in above Lemmas, it is easy to see that these Lemmas still hold for *generalized extremal ranges* by ignoring the common super-prefix bits from its *LCP*. Due to relevance and space limitation, detailed proofs will not be given in this paper. Thus, for the two *generalized extremal ranges* R_{am} and $R_{p(a-1)}$, the smallest number of encoded prefixes spanning these ranges can be obtained by simply counting the 0/1-bit of their range endpoints, ignoring the first three common bits from LCP_{am}: the lower five bits in integer value $(a-1)$ for R_{am} based on Lemma 2 and a for $R_{p(a-1)}$ based on Lemma 1. Since R_{am} can be encoded into 5 and 2 prefixes by using internal and external encoding scheme, the external encoding scheme will be used for R_{am}.

Step 3 Generate the final encoded prefixes in linear time by flipping each 1-bit (Lemma 1) or 0-bit (Lemma 2) of the range endpoints – excluding the common super-prefix – and padding the following bits with '*'.

After deciding on the encoding scheme for each *generalized extremal range*, we can then obtain the smallest number of prefixes that span the range with the methods in Lemmas 1 or 2 where applicable. Most importantly, these encoded prefixes can simply be generated in linear time. For example, by using external encoding scheme for the left *generalized extremal range*$R_{am} = [97, 127]$, we can first encode the external part of R_{am} under the range of LCP_{am} which is $R_{p(a-1)} = [96, 96]$. Based on Lemma 1, the smallest number of encoded prefixes that span range $R_{p(a-1)}$ is equal to the number of 1-bits in the lower five bits in integer value a that is *01100001*. Thus, the single encoded prefix for $R_{p(a-1)}$ can be generated by replacing the last 1-bit in a with 0-bit, that is *01100000*. Therefore, we can obtain two encoded prefixes for R_{am} with external encoding scheme as illustrated in Figs. 8 and 9.

4.5. Theoretical analysis of encoding complexity

Based on above descriptions of SplitCoding, any W-width range R can be encoded in a nearly constant time. From Fig. 9 and 10, we can see that in SplitCoding, all final encoded items of range $R = [a, b]$ can be obtained based on two endpoints of R: a and b. Thus, from the memory access point of view, SplitCoding can finish the coding for R in $O(1)$ memory access, regardless of the range width W. In contrast, optimal encoding adopts a dynamic-programming algorithm presented in [43] to generate encoded items recursively, where it needs to modify two candidate subsets in each recursive step. Thus, it may consume up to $O(W)$ memory accesses to encode range R, much more than that in SplitCoding.

4.6. Worst-case range expansion for pre-classifier

We next give some refined theorems for SplitCoding about upper-bound on range expansion. We then prove the worst-case range expansion for pre-classifier in SplitIP by using SplitCoding. Assume that the range $R = [a, b]$ $(a < b)$, where its width and length are W and L, respectively.

Theorem 1. *The range expansion g(R) of the extremal range R (i.e., $a = 0$ or $b = 2^W-1$) satisfies the following upper-bound:*

$$g(R) \le \left\lfloor \frac{W+2}{2} \right\rfloor = \left\lceil \frac{W+1}{2} \right\rceil$$

Proof. For the binary representation of b, it is not difficult to prove that the number of 1-bit in $b+1$ is at most one more than that in b. Based on Lemmas 1 and 2, we can see that the smallest number of ternary entries that span $[0, b]$ and $[b+1, 2^W-1]$ are equal to the number of 1-bits in $b+1$ and the number of 0-bits in b, respectively. Thus, the total number of encoded prefixes using internal and external encoding scheme is at most $W+2$ (i.e., $W+1$ bits and one external prefix). After choosing the better one, we can obtain the above upper-bound. The proof is similar for the *extremal range* $[a, 2^W-1]$.

Theorem 2. *The range expansion g'(R) of the generalized extremal range R satisfies the following upper-bound:*

$$g'(\mathrm{R}) \leq \left\lceil \frac{L+1}{2} \right\rceil$$

Proof. Obviously, the *generalized extremal range R* can be treated as an *extremal range* under the range of LCP_R, where the *sub-width* of LCP_R is equal to *L*, which is the length of *R*. Based on the proof of Theorem 1, we can easily prove the above upper-bound under the range of LCP_R.

Theorem 3. *The range expansion f(R) of the general range R satisfies the following upper-bound:*

$$f'(\mathrm{R}) \leq \mathrm{L}+1$$

Proof. It is easy to prove that the *L*-length range *R* can be split into two *generalized extremal ranges*, where their maximum sub-widths are *L* and *L−1* respectively. So:

$$f'(\mathrm{R}) \leq \left\lceil \frac{(L-1)+1}{2} \right\rceil + \left\lceil \frac{L+1}{2} \right\rceil = \mathrm{L}+1$$

Theorem 4. *For K W-bit consecutive and nonoverlapping ranges covering the whole range space, they can be encoded in at most K(W−log₂K+1) TCAM entries.*

Proof. Assume the length of range R_i is L_i ($i=1,2,\ldots,K$), we have:

$$2^W = 2^{L_1} + 2^{L_2} + \ldots + 2^{L_K}$$

Based on AM-GM inequality, we have:

$$2^W \geq K \cdot \sqrt[K]{2^{L_1+L_2+\cdots+L_K}}$$

$$L_1 + L_2 + \cdots + L_K \leq K(W - log_2 K)$$

Thus, the total range expansion of *K W*-bit consecutive and nonoverlapping ranges $f'_{total}(W)$ is:

$$f'_{total}(W) = f'_1(W) + f'_2(W) + \ldots + f'_K(W)$$

$$\leq (L_1+1) + (L_2+1) + \ldots + (L_K+1)$$

$$\leq K(W - log_2 K + 1)$$

So, by using SplitCoding, *K* index ranges generated by SplitIP can be encoded into at most *K(W−log₂K+1)* TCAM entries, with *W−log₂K+1* index entries for each TCAM block on average.

4.7. Route update

To support incremental updates, we need to make two changes on above basic SplitIP: (1) to support incremental updates of inserting, the first change needed is to reserve a small fraction of entries in each TCAM block during routing table partitioning; (2) to ensure the correctness of lookup after deleting, the second change needed is to reserve the prefixes completely covered by longer prefixes, even though it may never be matched in this block due to the LPM principle, such as P_4 and P_8 in Fig. 2. Next, we give more details about incremental updates.

The incremental update for prefix *P* can be completed in two steps. The first step is to determine which one or more index ranges are overlapped with *P*. The second step is to active the TCAM blocks indexed by these overlapped index ranges for update operations. Recall that the index ranges in pre-classifier are disjoint and consecutive. Thus, when updating a prefix *P*, we can perform the following steps. First we use two endpoint of *P* to locate two TCAM blocks (b_m and b_n, $m \leq n$) at which they are supposed to reside. Then we activate TCAM blocks from b_m to b_n to conduct update operations as follows. When inserting a new prefix P_i, some blocks may overflow after insertion, which should be handled before the insert command is issued. To solve this problem, we can move the leftmost or the rightmost prefixes to its neighboring blocks recursively, until no block may overflow after the insertion. For each modified block, we can recode its new index range with SplitCoding, because it is not only fast but also database independent. In contrast, the deletion of an old prefix requires only a delete command issued for these activated blocks to remove it out if it exists.

Table 4
Two core routing tables.

Table snapshot	Location	Date	Table size
rcc01 (linx_1)	*London*	*1/1/2012*	*396376*
oregon (linx_1)	*Oregon*	*17/12/2014*	*518231*

5. Experimental results

In this section, we present some experimental results of SplitIP using 32-bit BGP routing tables. We start with an overview of our experimental methodology. Since the primary metrics for evaluating the performance of power-efficient TCAM based solutions are power reductions and memory consumptions, we then evaluate power reductions and memory consumptions of SplitIP separately. Finally, we evaluate the effectiveness of SplitCoding. All experiments are run on a machine with AMD Radeon 5-2400G CPU@3.6GHz and 8G DRAM.

5.1. Experimental methodology

We evaluate the performance of SplitIP using two representative routing tables obtained from *RIPE NCC* [40] and *Oregon Route Views Project* [51] respectively, which are recently used for evaluation by *SAIL* [47] and *Poptrie* [14]. Details of these two routing tables are listed in Table 4. Since the TCAM block size is an important parameter for evaluations, we show results with different block sizes to demonstrate how the performance of our scheme is affected by block size.

As mentioned in above sections, the main component of TCAM power consumptions is proportional to the number of activated TCAM blocks. For the convenient of evaluation, we employ a simple linear power model to estimate power reductions, though real reductions may be slightly different. Suppose the routing table contains *N* prefixes and the TCAM block size is *B*, the number of activated TCAM blocks for default TCAM schemes without using pre-classifier can be defined as *X*, where $X = \lceil N/B \rceil$. In order to forward a destination address, the pre-classifier together with at most one specific TCAM block are needed to be activated. Thus, assuming that *M* pre-classifier entries are formed, at most $Y = \lceil M/B \rceil + 1$ TCAM blocks are activated for each routing lookup. Therefore, the percentage of *power reduction* with SplitIP is $(X-Y)/X$. We use these definitions to evaluate power reductions of SplitIP.

Note that the primary objective of our work is to achieve power reductions without sacrificing TCAM consumption. There are two reasons for extra TCAM overhead: additional memory for building pre-classifier and multiple entries for each duplicated prefix. Suppose *M* pre-classifier entries are generated and an amount of *Q* prefixes are stored in TCAM (include original prefixes and duplicated prefixes), we can then calculate the ratio of *storage overhead* as $(M+Q)/N$ and the ratio of *blocking replication* as Q/N. Besides, the *pre-classifier storage overhead* can be obtained by the following formula: $(M+N)/N$. We use these notations and definitions to evaluate memory consumptions of SplitIP.

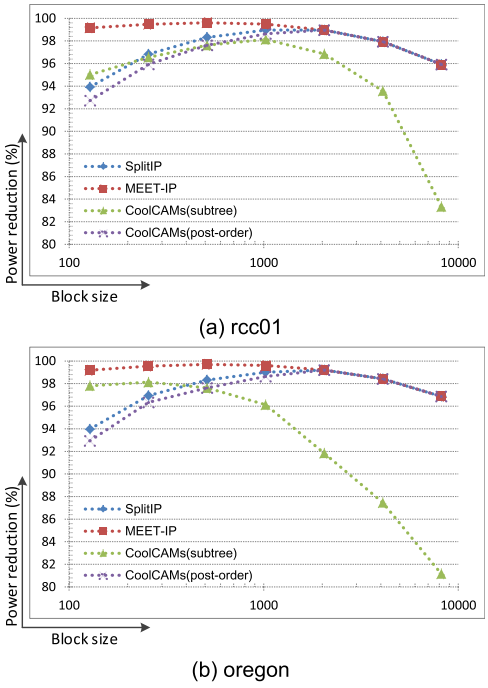


Fig. 11. The power reductions of SplitIP.

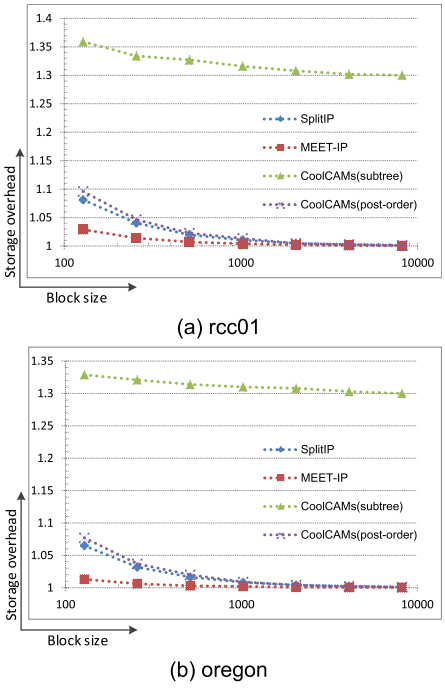


Fig. 12. The storage overhead of SplitIP.

5.2. Power reduction

The reductions in power consumption by using SplitIP are shown in Fig. 11, where x-axis represents block size and y-axis shows the percentage of power reduction. As shown in Fig. 11(a) and (b), SplitIP achieves huge power reductions similar to MEET-IP, ranging from 93.93% to 99.21%, with an average power reduction of 97.41%, better than that of 92.56% and 96.28% in CoolCAMs using two different splitting methods. Additionally, SplitIP is more suitable for incremental updates than CoolCAMs, because the merged intervals are consecutive in SplitIP, while those merged intervals in CoolCAMs are discrete. Fig. 11 also shows that, with the increase of block size, the power reductions show a trend from rise to decline, finally reaching the same value for different schemes. The rising trend in the first stage can be explained by previous definitions of power reduction, which is defined by $(X-Y)/X$, where $X = \lceil N/B \rceil$ and $Y = \lceil P/B \rceil + 1$. With the increase of block size, $\lceil P/B \rceil$ will be a fixed value (i.e., 1) for different schemes (block size larger than the number of pre-classifier entries), leading same power reductions for different schemes.

5.3. Storage overhead

The storage overheads by using SplitIP are shown in Fig. 12, where x-axis represents block size and y-axis shows the ratio of storage overhead. We can see from Fig. 12(a) and (b) that SplitIP achieves huge power reductions with negligible storage overhead as MEET-IP, ranges from 1.001 to 1.081, with an average storage overhead of 1.021, better than that of 1.31 and 1.028 in CoolCAMs using two different splitting methods. That is to say, only 2.1% extra TCAM overhead is needed for SplitIP. Another conclusion can be drawn from Fig. 12(a) and (b): storage overhead is declined with the increase of block size. To gain more insights about storage overhead in SplitIP, we next give more storage overhead details. This improvement, combined with the result in Fig. 11, justifies our claim that a careful combination of algorithmic approaches and block-based TCAM designs can achieve significant power reductions and avoid severe TCAM storage waste at the same time.

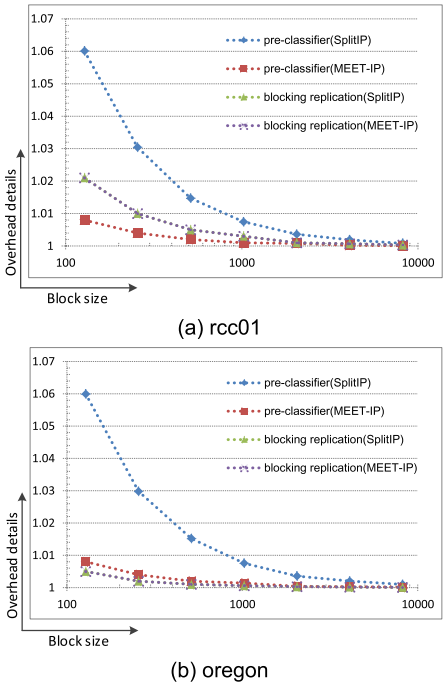


Fig. 13. More details about storage overhead of SplitIP.

As mentioned in above sections, two reasons may lead to extra storage overhead: pre-classifier and duplicated prefixes. Thus, to gain more insights about storage overhead of SplitIP and MEET-IP, we look into some details for these two reasons. The results are shown in Fig. 13, where x-axis represents block size and y-axis shows the ratio of storage overhead. We can see from Fig. 13 that SplitIP and MEET-IP have the same ratio of duplicated prefixes, but with different ratio of pre-classifier storage overhead: 1.017 and 1.003 on average. Thus, compared to MEET-IP, SplitIP can avoid additional endpoint comparisons by only introducing 1.4% additional pre-classifier entries.

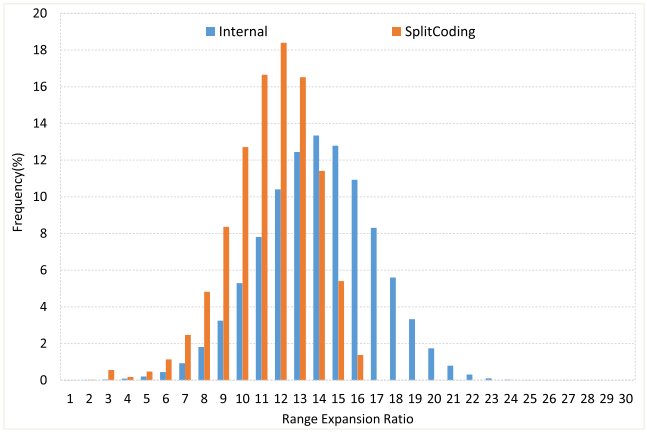


Fig. 14. Range expansion distribution over all possible ranges for width $W = 16$.

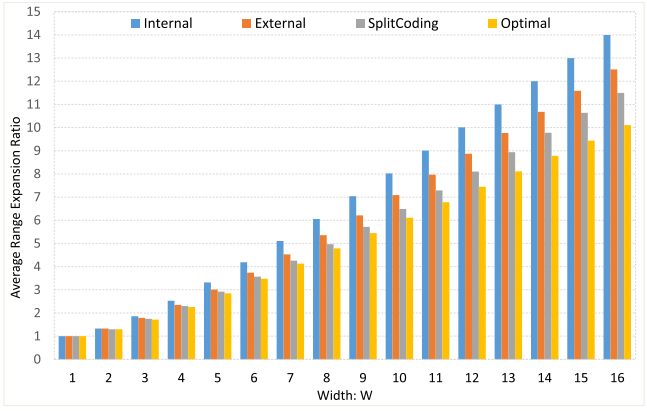


Fig. 15. Average range expansion ratio over all possibile ranges for diffierent width.

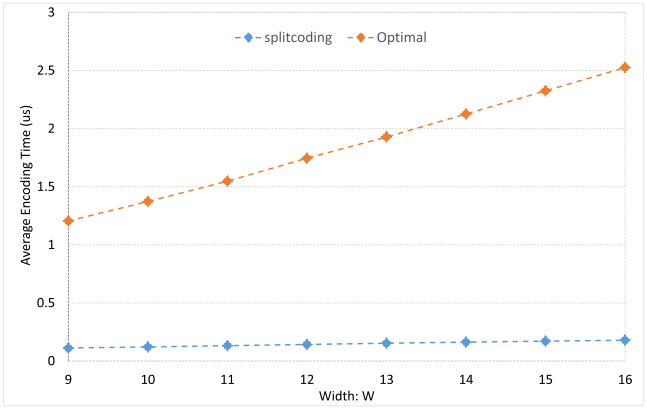


Fig. 16. Average range encoding time of SplitCoding compared to optimal encoding for all possibile ranges with diffierent width.

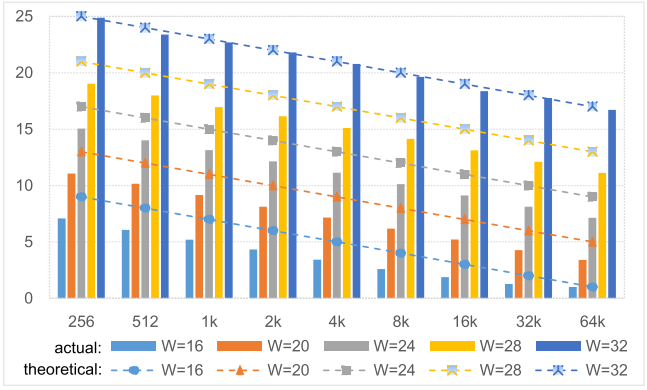


Fig. 17. Average range expansion ratio for different size of elementary intervals with different range width W.

As an extended work of MEET-IP, another main contribution of SplitIP is the proposed novel encoding scheme called SplitCoding, as well as some new theorems and proofs. Thus, we next give more evaluations to verify the effectiveness and prove the correctness of SplitCoding.

5.4. Effectiveness of SplitCoding

We evaluate the performance of SplitCoding with internal [53], external [32] and optimal [33] encoding scheme. Given the width W, we generate all possible ranges with the same probability as follows: $[0, 0]$, $[0, 1]$, $[0, 2]$,..., $[0, 2^W-1]$, $[1, 1]$, $[1, 2]$, $[1, 3]$,..., $[2^W-2, 2^W-1]$, $[2^W-1, 2^W-1]$. All source codes are publicly available in [19].

Fig. 14 shows the range expansion distribution over all possible ranges with width $W = 16$. The worst-case expansion of internal encoding approach is $2W-2 = 30$ (with negligible probability), while it is $W = 16$ in SplitCoding. Clearly, more ranges are encoded with fewer prefixes in SplitCoding. Fig. 15 presents the average range expansion ratio over all possible ranges for different width. SplitCoding achieves an average reduction of 20.85% and 7.95% in encoded TCAM entries compared to internal and external approach. Although optimal encoding achieves an average reduction of 8% in encoded TCAM entries compared to SplitCoding, it consumes more than 10 times on average to generate encoded items as illustrated in Fig. 16. Besides, regardless of the range width W, Fig. 16 shows that SplitCoding can always finish encoding in a nearly constant time. In contrast, the average encoding time of optimal encoding grows linearly with the range width W, which is consistent with our theoretical analysis of encoding complexity given in Section 4.4. Thus, SplitCoding achieves similar improvement in performance of compression as optimal encoding, but much faster and simpler than optimal encoding.

Fig. 17 shows the evaluation between the theoretical upper bound and the actual expansion ratio for randomly generated elementary intervals. As the size of elementary intervals increases from 256 to 64k and W varies from 16 to 32, the expansion decreases almost linearly at a fixed slope, which indicates that there is a good deal of room for the expansion reduction as well as shows the correctness of the new theoretical upper bound.

6. Conclusion

In this paper, we propose SplitIP, a memory and power efficient 2-stage scheme for TCAM based IP routing table lookup. In order to build a more effective pre-classifier, we introduce a top-down framework which has a better global view on relationships among address prefixes. After the projecting of address prefixes, we propose a global splitting algorithm for projected elementary intervals, which can separate prefixes evenly into blocks without any TCAM holes. Besides, for the purpose of fast pre-classification, we introduce a one-level pre-classifier based on database independent TCAM range encodings, where each index range can be firstly split into two sub-ranges and then adopt different encoding techniques adaptively to reduce the final encoded TCAM entries. Experimental results show that our design achieves more than 97% power reduction with an extra TCAM storage overhead of less than 3% on average.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

We gratefully acknowledge the detailed comments and constructive suggestions made by editors and anonymous reviewers for our original submission. We appreciate the help from Ori Rottenstreich in terms of the design of SplitCoding. Besides, we also want to appreciate the friendly advices from Gaogang Xie and Tong Yang in this work. Hui Li and Dagang Li are corresponding authors of this paper. This work is supported in part by the PCL Future Regional Network Facilities for Large-scale Experiments and Applications (PCL2018KP001), Shenzhen Peacock Innovation Program (KQJSCX20180323174744219), Key Areas R&D Program of Guangdong (2019B010137001), National Keystone R&D Program of China (2017YFB0803204), NSFC (61671001) and Shenzhen Research Programs (JCYJ20170306092030521).

References

[1] A. Bremler-Barr, D. Hendler, Space-efficient TCAM-based classification using gray coding, in: Proceedings of the IEEE INFOCOM, 2007.
[2] A.J. McAuley, P. Francis, Fast routing table lookup using CAMs, in: Proceedings of the IEEE INFOCOM, 1993.
[3] A.X. Liu, G. Gouda, Complete redundancy removal for packet classifiers in TCAMs, IEEE Trans. Parallel Distrib. Syst. 20 (4) (2010) 424–437.
[4] A.X. Liu, C.R. Meiners, E. Torng, Packet classification using binary content addressable memory, in: Proceedings of the IEEE INFOCOM, 2014.
[5] A.X. Liu, C.R. Meiners, E. Torng, TCAM Razor: A systematic approach towards minimizing packet classifiers in TCAMs, IEEE/ACM Trans. Netw. 18 (2) (2010) 490–500.
[6] A.X. Liu, C.R. Meiners, Y. Zhou, All-match based complete redundancy removal for packet classifiers in TCAMs, in: Proceedings of the IEEE INFOCOM, 2008.
[7] B. Lampson, V. Srinivasan, G. Varghese, IP lookups using multiway and multicolumn search, IEEE/ACM Trans. Netw. 7 (3) (1999) 324–334.
[8] B. Schieber, D. Geist, A. Zaks, Computing the minimum DNF representation of Boolean functions defined by intervals, Discrete Applied Mathematics 149 (1-3) (2005) 154–173.
[9] B. Vamanan, T. Vijaykumar, TreeCAM: decoupling updates and lookups in packet classification, in: Proceedings of the ACM CoNEXT, 2011.
[10] C.R. Meiners, A.X. Liu, E. Torng, Hardware Based Packet Classification for High Speed Internet Routers, Springer Science & Business Media, 2010.
[11] D. Sarang, K. Praveen, D. Taylor, Longest prefix matching using bloom filters, in: Proceedings of the ACM SIGCOMM, 2003.
[12] D. Shah, P. Gupta, Fast incremental updates on Ternary-CAMs for routing lookups and packet classification, in: Proceedings of the IEEE Hot Interconnects, 2000.
[13] F. Zane, G. Narlikar, A. Basu, CoolCAMs: power-efficient TCAMs for forwarding engines, in: Proceedings of the IEEE INFOCOM, 2003.
[14] H. Asai, Y. Ohara, Poptrie: a compressed trie with population count for fast and scalable software IP routing table lookup, in: Proceedings of the ACM SIGCOMM, 2015.
[15] H. Che, Z. Wang, K. Zheng, B. Liu, DRES: dynamic range encoding scheme for TCAM coprocessors, IEEE Trans. Comput. 57 (7) (2008) 902–915.
[16] H.J. Chao, B. Liu, High Performance Switches and Routers, John Wiley & Sons, 2007.
[17] H. Lim, K. Lim, N. Lee, K. Park, On adding bloom filters to longest prefix matching algorithms, IEEE Trans. Comput. 63 (2) (2014) 411–423.
[18] H. Liu, Efficient mapping of range classifier into ternary-CAM, in: Proceedings of the IEEE Hot Interconnects, 2002.
[19] http://www.wenjunli.com/SplitCoding.
[20] K. Huang, G. Xie, Y. Li, A.X. Liu, Offset addressing approach to memory-efficient IP address Lookup, in: Proceedings of the IEEE INFOCOM Mini-Conference, 2011.
[21] K. Lakshminarayanan, A. Rangarajan, S. Venkatachary, Algorithms for advanced packet classification with ternary CAMs, in: Proceedings of the ACM SIGCOMM, 2005.
[22] K. Zheng, C. Hu, H. Lu, B. Liu, A TCAM-based distributed parallel IP lookup scheme and performance analysis, IEEE/ACM Trans. Netw. 14 (4) (2006) 863–875.
[23] K. Zheng, C. Hu, H. Lu, B. Liu, An ultra high throughput and power efficient TCAM-based IP lookup engine, in: Proceedings of the IEEE INFOCOM, 2004.
[24] L. Luo, G. Xie, S. Uhlig, L. Mathy, K. Salamatian, Y. Xie, Longest prefix matching using bloom filters, in: Proceedings of the ACM CoNEXT, 2012.
[25] L. Luo, G. Xie, Y. Xie, L. Mathy, K. Salamatian, A Hybrid Hardware Architecture for High-speed IP Lookups and Fast Route Updates, IEEE/ACM Trans. Netw. 22 (3) (2014) 957–969.
[26] L. Luo, G. Xie, Y. Xie, L. Mathy, K. Salamatian, A hybrid IP lookup architecture with fast updates, in: Proceedings of the IEEE INFOCOM, 2012.
[27] M. Bando, Y. Lin, H. Chao, FlashTrie: beyond 100-Gb/s IP route lookup using hash-based prefix-compressed trie, IEEE/ACM Trans. Netw. 20 (4) (2012) 1262–1275.
[28] M. Degermark, A. Brodnik, S. Carlsson, S. Pink, Small forwarding tables for fast routing lookups, in: Proceedings of the ACM SIGCOMM, 1997.
[29] M. Waldvogel, G. Varghese, J. Turner, B. Plattner, Scalable high speed IP routing lookups, in: Proceedings of the ACM SIGCOMM, 1997.
[30] Micron Technology Inc, Harmony TCAM 1Mb and 2Mb, Datasheet, 2003.
[31] O. Rottenstreich, I. Keslassy, On the code length of TCAM coding schemes, in: Proceedings of the IEEE ISIT, 2010.
[32] O. Rottenstreich, I. Keslassy, Worst-case TCAM rule expansion, in: Proceedings of the IEEE INFOCOM Mini-Conference, 2010.
[33] O. Rottenstreich, I. Keslassy, A. Hassidim, H. Kaplan, E. Porat, On finding an optimal TCAM encoding scheme for packet classification, in: Proceedings of the IEEE INFOCOM, 2013.
[34] O. Rottenstreich, I. Keslassy, A. Hassidim, H. Kaplan, E. Porat, Optimal In/Out TCAM encodings of ranges, IEEE/ACM Trans. Netw. 24 (1) (2016) 555–568.
[35] O. Rottenstreich, R. Cohen, D. Raz, I. Keslassy, Exact worst-case TCAM rule expansion, IEEE Trans. Comput. 62 (6) (2013) 1127–1140.
[36] P. Gupta, S. Lin, N. McKeown, Routing lookups in hardware at memory access speeds, in: Proceedings of the IEEE INFOCOM, 1998.
[37] P. He, W. Zhang, H. Guan, K. Salamatian, G. Xie, Partial order theory for fast TCAM updates, IEEE/ACM Trans. Netw. 26 (1) (2018) 217–230.
[38] P. Rina, S. Samar, Reducing TCAM power consumption and increasing throughput, in: Proceedings of the IEEE Hot Interconnects, 2002.
[39] R. Miguel, B. Ernst, D. Walid, Survey and taxonomy of IP address lookup algorithms, IEEE Netw. 15 (2) (2001) 8–23.
[40] RIPE network coordination centre [on line]. Available: http://www.ripe.net.
[41] S. Han, K. Jang, K. Park, S. Moon, PacketShader: a GPU-accelerated software router, in: Proceedings of the ACM SIGCOMM, 2010.
[42] S. Nilsson, G. Karlsson, IP-address lookup using LC-tries, IEEE J. Sel. Areas Commun. 17 (6) (1999) 1083–1092.
[43] S. Suri, T. Sandholm, P. Warkhede, Compressing two-dimensional routing tables, Algorithmica 35 (4) (2003) 287–300.
[44] T. Yang, A.X. Liu, Y. Shen, Q. Fu, D. Li, X. Li, Fast openflow table Lookup with fast update, in: Proceedings of the IEEE INFOCOM, 2018.
[45] T. Yang, B. Yuan, S. Zhang, T. Zhang, R. Duan, Y. Wang, B. Liu, Approaching optimal compression with fast update for large scale routing tables, in: Proceedings of the IEEE/ACM IWQoS, 2012.
[46] T. Yang, G. Xie, A.X. Liu, Q. Fu, Y. Li, X. Li, L. Mathy, Constant IP lookup with FIB explosion, IEEE/ACM Trans. Netw. 26 (4) (2018) 1821–1836.
[47] T. Yang, G. Xie, Y. Li, Q. Fu, A.X. Liu, Q. Li, L. Mathy, Guarantee IP lookup performance with FIB explosion, in: Proceedings of the ACM SIGCOMM, 2014.
[48] T. Yang, R. Duan, J. Lu, S. Zhang, H. Dai, B. Liu, CLUE: achieving fast update over compressed table for parallel lookup with reduced dynamic redundancy, in: Proceedings of the IEEE ICDCS, 2012.
[49] T. Yang, T. Zhang, S. Zhang, B. Liu, Constructing optimal non-overlap routing tables, in: Proceedings of the IEEE ICC, 2012.
[50] T. Yang, Z. Mi, R. Duan, X. Guo, J. Lu, S. Zhang, X. Sun, B. Liu, An ultra-fast universal incremental update algorithm for Trie-based routing lookup, in: Proceedings of the IEEE ICNP, 2012.
[51] University of Oregon route views project [on line]. Available: http://www.routeviews.org.
[52] V. Srinivasan, G. Varghese, Faster IP lookups using controlled prefix expansion, ACM SIGMETRICS Perform. Eval. Rev. 26 (1) (1998) 1–10.
[53] V. Srinivasan, G. Varghese, S. Suri, M. Waldvogel, Fast and Scalable Layer Four Switching, in: Proceedings of the ACM SIGCOMM, 1998.
[54] W. Eatherton, G. Varghese, Z. Dittia, Tree bitmap: Hardware/software IP lookups with incremental updates, ACM SIGCOMM Comput. Commun. Rev. 34 (2) (2004) 97–122.
[55] W. Li, X. Li, HybridCuts: a scheme combining decomposition and cutting for packet classification, in: Proceedings of the IEEE Hot Interconnects, 2013.
[56] W. Li, X. Li, H. Li, MEET-IP: memory and energy efficient TCAM-based IP lookup, in: Proceedings of the International Conference on Computer Communications and Networks (ICCCN), 2017.
[57] W. Li, X. Li, H. Li, G. Xie, CutSplit: a decision-tree combining cutting and splitting for scalable packet classification, in: Proceedings of the IEEE INFOCOM, 2018.
[58] W. Li, X. Liu, W. Le, H. Li, H. Zhang, A practical range encoding scheme for TCAMs, in: Proceedings of the ACM SIGCOMM Posters and Demos, 2019.
[59] X. Li, Y. Lin, W. Li, GreenTCAM: a memory- and energy-efficient TCAM-based packet classification, in: Proceedings of the International Conference on Computing, Networking and Communications (ICNC), 2016.
[60] Y.K. Chang, A 2-level TCAM architecture for ranges, IEEE Trans. Comput. 55 (12) (2006) 1614–1629.
[61] Y.K. Chang, Power-efficient TCAM partitioning for IP lookups with incremental updates, in: Proceedings of the International Conference on Information Networking (ICOIN), 2005.
[62] Y.K. Chang, C.I. Lee, C.C. Su, Multi-field range encoding for packet classification in TCAM, in: Proceedings of the IEEE INFOCOM, 2011.

[63] Y.K. Chang, C.C. Su, Y.C. Lin, S.Y. Hsieh, Efficient gray-code-based range encoding schemes for packet classification in TCAM, IEEE/ACM Trans. Netw. 21 (4) (2013) 1201–1214.
[64] Y. Li, D. Zhang, A.X. Liu, J. Zheng, GAMT: a fast and scalable IP lookup engine for GPU-based software routers, in: Proceedings of the ACM/IEEE ANCS, 2013.
[65] Y. Ma, S. Banerjee, A smart pre-classifier to reduce power consumption of TCAMs for multi-dimensional packet classification, in: Proceedings of the ACM SIGCOMM, 2012.
[66] Z. Marko, R. Luigi, M. Miljenko, DXR: towards a billion routing lookups per second in software, ACM SIGCOMM Comput. Commun. Rev. 42 (5) (2012) 29–36.

Wenjun Li, Wenjun Li received his B.Sc. from University of Electronic Science and Technology of China, in 2011, and M.Sc. from Peking University, in 2014. From 2014 to 2015, he worked as a researcher in network research department, Huawei Technologies Co. Ltd. Now, he is a Ph.D. candidate in School of Electronics Engineering and Computer Science, Peking University. His research interest includes computer network architecture and high-performance network device.

Dagang Li received his Bachelor from Huazhong University of Science and Technology, Wuhan, China in 1998 and Ph.D. from Katholieke Universiteit Leuven (University of Leuven), Leuven, Belgium in 2010. He is currently an Assistant Professor with Peking University Shenzhen Graduate School. His research areas are data center networking and storage infrastructure, and big data processing systems.

Xinwei Liu received her B.Sc. from Huazhong University of Science and Technology in 2017. Now she is a graduate student in School of Electronics and Computer Engineering, Peking University. Her research interest includes computer network architecture and Software Defined Networking.

Ting Huang received her B.Sc. from Sun Yat-sen University of Data and Computer Science in 2017. Now she is a graduate student in School of Electronics and Computer Engineering, Peking University. Her research interest includes computer network architecture and Software Defined Networking.

Xianfeng Li received his B.Sc. from School of Computer and Control Engineering, Beijing Institute of Technology, in 1995, and Ph.D. degree in computer science from the National University of Singapore, in 2005. Now, he is associate professor in School of Electronic and Computer Engineering, Peking University. His research interest includes Software Defined Networking, network security and Internet of Things.

Wenxia Le is an enterprise network engineer and a sales manager in Network Energy Department, Huawei Technologies Co., Ltd. Her research interest includes energy efficient data center networking, low power devices in data communication network and Network Function Virtualization.

Hui Li received the B.Eng. and M.S. degrees from Tsinghua University, Beijing, China, in 1986 and 1989 respectively, and Ph.D. degree from The Chinese University of Hong Kong in 2000. He is now with the Shenzhen Key Lab of Information Theory & Future Network Architecture, Future Network PKU Lab of National Major Research Infrastructure, Shenzhen Engineering Lab of Converged Networking Technology, Huawei & PKU Jointly Engineering Lab of Future Network Based on SDN and the PKU Institute of Big Data Technology, Shenzhen Graduate School, Peking University. His research interests include distributed storage systems, network coding, large-scale switching and routing.