DRES: Dynamic Range Encoding Scheme for TCAM Coprocessors

Hao Che, *Senior Member*, *IEEE*, Zhijun Wang, Kai Zheng, *Member*, *IEEE*, and Bin Liu, *Member*, *IEEE*

Abstract—One of the most critical resource management issues in the use of ternary content-addressable memory (TCAM) for packet classification/filtering is how to effectively support filtering rules with ranges, known as range matching. In this paper, the Dynamic Range Encoding Scheme (DRES) is proposed to significantly improve the TCAM storage efficiency for range matching. Unlike the existing range encoding schemes requiring additional hardware support, DRES uses the TCAM coprocessor itself to assist range encoding. Hence, DRES can be readily programmed in a network processor using a TCAM coprocessor for packet classification. A salient feature of DRES is its ability to allow a subset of ranges to be encoded and, hence, to have full control over the range code size. This advantage allows DRES to exploit the TCAM structure to maximize the TCAM storage efficiency. DRES is a comprehensive solution, including a dynamic range selection algorithm, a search key encoding scheme, a range encoding scheme, and a dynamic encoded range update algorithm. Although the dynamic range selection algorithm running in the software allows optimal selection of ranges to be encoded to fully utilize the TCAM storage, the dynamic encoded range update algorithm allows the TCAM database to be updated lock free without interrupting the TCAM database lookup process. DRES is evaluated based on real-world databases and the results show that DRES can reduce the TCAM storage expansion ratio from 6.20 to 1.23. The performance analysis of DRES based on a probabilistic model demonstrates that DRES significantly improves the TCAM storage efficiency for a wide spectrum of range distributions.

Index Terms—Packet classification, range matching, ternary CAM, network processor.

1 Introduction

Packet classification has been recognized as a critical data path function for high-speed packet forwarding in a router. To keep up with multigigabit line rates, a high-performance router needs to be able to classify a packet in a few tens of nanoseconds. In the last few years, significant research efforts have been made to design fast packet classification algorithms for both Longest Prefix Matching (LPM) and general policy/firewall filtering (PF) [2], [6], [9], [10], [20], [21], [22], [24]. However, most of these algorithmic approaches cannot provide deterministic lookup performance matching multigigabit line rates.

An alternative approach, which has been gaining popularity, is the use of a *ternary content-addressable memory* (TCAM) coprocessor for fast packet classification. In general, a TCAM coprocessor works as a look aside processor for packet classification on behalf of a network processing unit (NPU) or network processor. When a packet is to be classified, an NPU generates a search key based on the information extracted from the packet header and passes it to the TCAM coprocessor for classification. A TCAM coprocessor finds a matched rule in a small constant number of clock cycles, offering the highest possible lookup/matching performance [8]. Indeed, packet processing at a line rate of 10 gigabits per second (Gbps) using an integrated NPU and TCAM coprocessor solution has been reported [1].

However, despite its fast lookup performance, the TCAM-based solution has its own shortcomings, including high power consumption, large footprint, and high cost. These shortcomings directly contribute to a critical issue for packet classification using TCAM, namely, supporting rules with ranges, or range matching. The difficulty lies in the fact that multiple TCAM entries have to be allocated to represent a range field. A rule that involves multiple range fields will cause a multiplicative expansion of the rule expressed in TCAM. Our statistical analysis of real-world rule databases shows that the TCAM storage efficiency can be as low as 16 percent due to the existence of a significant number of rules with port ranges. The work in [2], [9], [13], [19] also reported that today's real-world PF tables involve significant amounts of rules with ranges. Clearly, the reduced TCAM memory efficiency due to range matching makes TCAM power consumption, footprint, and cost even more serious concerns.

A general approach to deal with range matching is to do a range preprocessing/encoding by mapping ranges to a short sequence of encoded bits, known as bitmapping. The idea is to use a bit to represent a range in a field. Hence, each rule can be translated to a sequence of encoded bits, known as *rule encoding*. Accordingly, a search key based on the information extracted from the packet header is preprocessed to generate an encoded search key, called

- *H. Che is with the Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, TX 76019. E-mail: hche@cse.uta.edu.*
- *Z. Wang is with the Department of Computing, Hong Kong Polytechnic University, Hong Kong. E-mail: cszjwang@comp.polyu.edu.hk.*
- *K. Zheng is with the System Research Group, IBM China Resarch Lab, Beijing, P.R. China. E-mail: zhengkai@cn.ibm.com.*
- *B. Liu is with the Department of Computer Science and Technology, Tsinghua University, Beijing 10084, P.R. China. E-mail: liub@tsinghua.edu.cn.*

Manuscript received 17 Mar. 2006; revised 19 Feb. 2007; accepted 5 Oct. 2007; published online 17 Oct. 2007.
Recommended for acceptance by M. Gokhale.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0104-0306.
Digital Object Identifier no. 10.1109/TC.2007.70838.

search key encoding. Then, the encoded search key is matched against all the encoded rules to find the best matched rule. Although the rule encoding can be done in the software, the search key encoding is performed on a per-packet basis and must be done in the hardware at wire speed. In addition to algorithms for rule encoding and search key encoding, a comprehensive range encoding scheme also needs to provide two other important algorithms. First, it needs to provide a *range selection* algorithm that selects the ranges to be encoded to maximize the TCAM storage efficiency. Second, it needs to provide a *database update* algorithm to minimize its impact on the rule matching process.

The bit-map-based range encoding algorithm was originally proposed by Lakshman and Stiliadis [14]. The application of the bit-map-based range encoding for packet classification using a TCAM has also been reported [16], [17], [18]. In particular, the rule encoding schemes proposed by van Lunteren and Engbersen [17], [18] are the most effective schemes as they can encode N nonoverlapping ranges using only $log_2(N+1)$ bits.

All of the existing range encoding schemes are top-down approaches. That is, they strive to design the most efficient rule encoding algorithms by assuming the existence of the needed hardware to support search key encoding at high speed. For example, the search key encoding approaches proposed in [14], [17], [18] assume the existence of multiple processors and/or multiple memories for parallel search key encoding. Some other schemes simply do not address search key encoding issues (for example, see [15]). Another example of the top-down approach is the work by Spitznagel et al. [19], which addresses TCAM power and range matching issues by designing a new TCAM architecture.

The research based on a top-down approach is of great importance because it provides insights, directions, or even solutions on how the next-generation TCAM-based packet classifiers should be designed. What is missing, however, is an approach from the bottom up, that is, designing range-encoding schemes subject to hardware constraints of the existing TCAM-based packet classification solutions. A bottom-up approach may not offer the highest encoding gain, but it ensures that, when applied to an existing TCAM-based packet classifier, it will work with only software upgrades. A bottom-up approach is of paramount importance for any system vendors who use a third-party TCAM coprocessor in their system design. These vendors are badly in need of a range-encoding scheme, which not only significantly improves TCAM storage efficiency but also requires only software upgrades, leaving the existing hardware unchanged. As NPUs and TCAM coprocessors are fully integrated for major NPU vendor solutions, which are widely used by system vendors, including the Intel IXP2xxx series [11] and AMCC nP7xxx series [7], [12], there is an increasingly pressing need to develop efficient bottom-up range encoding schemes that can be immediately programmed in any NPUs using a TCAM coprocessor for packet classification. Unfortunately, to the best of our knowledge, no such solution exists today.

In this paper, we propose a Dynamic Range Encoding Scheme (DRES). DRES is a bottom-up solution, which can be readily programmed in any NPU using a TCAM coprocessor for packet classification. Statistical analyses on both real-world PF samples and a statistical model demonstrate that DRES can significantly improve the TCAM storage efficiency in support of range matching. Moreover, DRES brings the following novelties to range encoding designs. First, unlike most existing top-down schemes, which mainly focus on the design of efficient rule encoding algorithms, DRES is a comprehensive solution that provides all four algorithms needed for its immediate implementation, including a rule encoding algorithm, a search key encoding algorithm, a range selection algorithm, and a database update algorithm. Second, in DRES, an encoded rule structure is designed which allows any subset of ranges in any subset of rule fields to be encoded. This allows DRES to have full control over the encoded rule size and the number of rule fields to be encoded, making it possible for DRES to exploit various design trade-offs and adapt to the changing rule database structure or rule pattern to achieve the maximum encoding gain. Third, the DRES database update algorithm allows uninterrupted search key encoding and rule matching without TCAM locking for database updating. Finally, DRES is a general range encoding framework that allows any existing range encoding algorithms to be incorporated for range encoding. Of course, different range encoding algorithms will lead to different design complexities of the other three algorithms. In this paper, DRES is designed in the context of a simple bit-map intersection algorithm.

Furthermore, we note that, although primarily designed as a bottom-up solution, most of the ideas developed in DRES can be leveraged in the design of top-down approaches as well. For example, a variation of DRES with parallel search key encoding is successfully adopted in the design of a distributed TCAM-based packet classifier [25] that matches a 40 Gbps line rate.

The rest of this paper is organized as follows: Section 2 describes how the rules are expressed in a TCAM. The detailed description of DRES is given in Sections 3 and Section 4. Section 3 describes in detail how rules and search keys are encoded and how ranges are selected for encoding. Section 4 details the encoded range update procedure. Section 5 evaluates the performance of DRES based on real-world databases. Section 6 analyzes the performance of the DRES based on a statistical model. Finally, Section 7 concludes this paper.

2 Rule Implementation in TCAM

Basically, there are two types of TCAM coprocessors in terms of the number of interfaces that they support, that is, a single interface for NPU and two interfaces, one for NPU and the other for CPU. For a TCAM coprocessor with a single interface, the TCAM database update process must share the same interface with the lookup process, while, for a TCAM coprocessor with two interfaces, the TCAM update process may use either interface. In terms of performance, it is better to use the CPU interface for TCAM database update if it is available. However, due to the possible simultaneous rule matching for lookup and rule modification for rule updating, TCAM database updating through the CPU interface may cause erroneous rule matching. This makes it harder to design

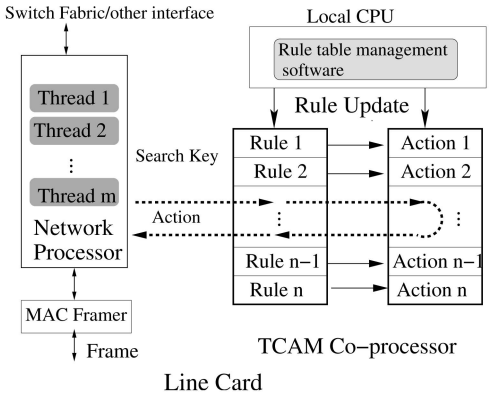


Fig. 1. A network processor and its TCAM coprocessor.

TCAM database update algorithms for cases when the CPU interface is used for updating. For this reason, DRES focuses on the TCAM coprocessors with two interfaces using the CPU interface for TCAM database updating. The proposed lock-free updating algorithms can be easily adapted to one interface case.

Fig. 1 gives a logic diagram showing how a TCAM coprocessor works with an NPU. A TCAM coprocessor includes two pieces of memory in general, that is, a TCAM and an associated memory (for example, an SRAM). The TCAM is organized in slots. The slot size is generally configurable, allowing different tables in TCAM to be configured at different slot sizes, for example, 64, 72, 128, 192, or 256 bits. A bit in each slot can take one of these three values: 0, 1, or "don't care," denoted as "$*$". The rules in a PF table are placed in the TCAM and each rule entry (as will be defined shortly) maps to a memory address in the associated memory in which the corresponding action code is kept. A rule takes one or multiple slots. When a packet comes, the NPU generates a search key based on the packet header information and passes it to the TCAM coprocessor to be classified via an NPU-TCAM coprocessor interface. A local CPU is in charge of rule table update through a separate CPU-TCAM coprocessor interface.

We use an example to guide the discussion throughout the rest of this paper. Consider a typical five-tuple 104-bit (for IPv4)[1] PF rule composed of the following five fields: {*source IP address, destination IP address, source port, destination port, and protocol number*}. Each rule is associated with an *action*. Fig. 2 gives eight such example rules, $L_1, \ldots, L_8$, and their corresponding actions. A port number can be an exact number or a range and each distinct port number for a given port field is called a unique port. For example, {$>$ 1,023}, {2,047}, {256-512}, and {$<$ 1,024} are four unique source ports.

Fig. 3 depicts what it looks like when rules L_1 and L_2 are placed in a TCAM with a 64-bit slot size. L_1 does not have a range in any of its fields and, hence, it takes two slots, with 24 free bits left in the second slot. Each such rule in the TCAM which takes the minimum number of slots is defined as a *rule entry*. L_2 has a range {256-512} in its destination port field. This range cannot be directly expressed in a TCAM and must be partitioned into two subranges, {256-511} and {512}, as shown

1. For IPv6, the procedure is similar; only the rule length varies.

	Source IP	Destination IP	Source Port	Destination Port	Protocol	Action
L_1	1.1.x.x	2.2.2.x	x.x	x.x	6	AF
L_2	x.x.x.x	2.2.x.x	x.x	256–512	6	BF
L_3	1.2.x.x	x.x.x.x	>1023	768–2047	17	Discard
L_4	3.1.x.x	2.2.2.x	x.x	6000–6064	4	EF
L_5	4.4.4.x	2.2.x.x	>1023	>1023	x	Accepted
L_6	5.5.x.x	3.3.x.x	2047	512–1536	x	BF
L_7	6.x.x.x	7.7.x.x	256–512	256–512	6	BF
L_8	8.x.x.x	9.x.x.x	<1024	2047	x	BF

BF: best effort forwarding AF: assurred forwarding EF: expedited forwarding

Fig. 2. An example of five-tuple rules. "x" represents a wild card byte.

in Fig. 3. Hence, L_2 takes four slots (slots 3, 4, 5, and 6) or two rule entries in the TCAM. Note that the action codes corresponding to the two rule entries must be identical and equal to the one corresponding to L_2.

To facilitate further discussion, a range is said to be *exactly implemented* in a TCAM if it is expressed in a TCAM without being encoded. A rule with an exactly implemented range therefore may take multiple TCAM rule entries, as in the case for L_2. As another example, six rule entries are needed to express range {$>$ 1,023} in a TCAM if the range is exactly implemented for its two port fields, as shown in Fig. 4. Hence, L_5 takes $6 \times 6 = 36$ rule entries if it is exactly implemented, resulting in a multiplicative rule expansion in TCAM. Ranges that cannot be exactly implemented using one rule entry in a TCAM are called *noncompact* ranges. Other ranges that can be exactly implemented in one rule entry in a TCAM are called *compact* ranges. For example, range {$<$ 1,024} in L_8 is a compact range and it can be exactly implemented as 000000********* in TCAM. To be exactly implemented, a noncompact range must be decomposed into a set of compact subranges, as in the case for $L2$. In what follows, we simply refer to a noncompact range as a range and refer to a subrange as either a compact or noncompact subrange.

3 Range Encoding

This section details how rules and search keys are encoded and how ranges are selected for encoding.

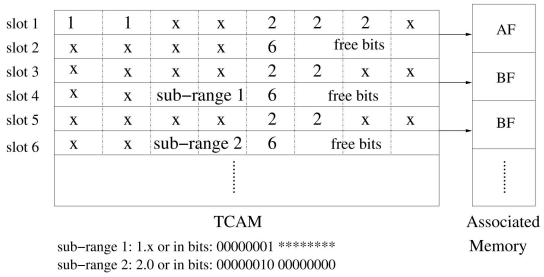


Fig. 3. Rules in a TCAM. The range {256-512} is split into two subranges, {256-511} and {512}, and implemented as subranges 1 and 2. "$*$" represents a "don't-care" bit and "x"="********" represents a wild card byte. The other numbers represent the actual byte values.

0000 01** **** ****	0000 1*** **** ****
0001 **** **** ****	001* **** **** ****
01** **** **** ****	1*** **** **** ****

Fig. 4. Range {> 1,023} is expressed in terms of six subranges in a TCAM.

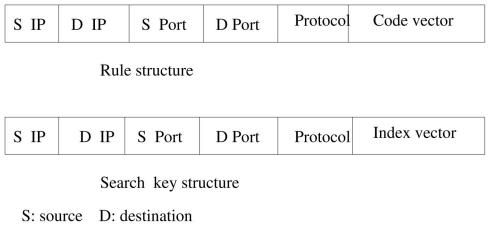


Fig. 5. Rule structures without and with range encoding and the encoded search key structure.

3.1 Structures of Encoded Rule and Encoded Search Key

Instead of replacing a rule field altogether by a sequence of code bits (for example, see [16], [18]; collectively called the *complete encoding approach* in this paper), we design a *hybrid encoding approach* for DRES. The hybrid encoding approach retains all of the fields in a rule and appends a sequence of code bits of length c_t, called the *code vector*, to the rule to form an encoded rule. Accordingly, an encoded search key is formed by appending a sequence of code bits of length c_t, called the *index vector*, to the original search key. The encoded rule and search key structures are shown in Fig. 5.

Code vectors and index vectors are calculated using a rule encoding algorithm and search key encoding algorithm, respectively. Due to the slotted TCAM structure, there will usually be some free bits left in each rule entry. For example, 24 free bits are left for the example given in Section 2. One may simply use these free bits to encode the ranges for free. However, if there is no free bit or the number of free bits is too small, DRES allows extra slots to be allocated for each rule entry as long as the overall TCAM storage efficiency is improved by encoding ranges using these slots. Since the number of ranges that can be encoded and the encoding gain are determined by c_t and rule database structure, a dynamic range selection algorithm must be designed to maintain high encoding gain in the presence of the changing rule database structure.

In our hybrid encoding approach, if a rule has no encoded range in any of its fields, the code vector is wild carded and the rule itself remains unchanged. This ensures that the index vector in a search key will always match the code vector in the encoded rule. If there is an encoded range in any field in the rule, that field is wild carded and the corresponding code vector is encoded based on the encoding rules to be described in Section 3.4. In this case, the matching between that range field and the corresponding field in the search key is replaced by the matching between the code vector and the index vector.

On the surface, it appears to be advantageous to adopt the CE approach rather than a hybrid encoding approach because, for the CE approach, the encoding gain can be improved by not only encoding all of the ranges but also eliminating the rule fields to reduce the rule length. Let us look at the example given in Section 2. If both 16-bit source and destination port fields are encoded and eliminated, there will be $24 + 16 \times 2 = 56$ free bits left for range encoding, which is 36 bits more than when the hybrid encoding approach is used. This, however, may not be true in practice because, to eliminate a rule field altogether, one must encode *all* of the unique values for that field in the rule database, whether they are range values or not. This means that one has no control over the code vector length c_t, which is rule database dependent. For example, if $c_t > 56$, at least one extra slot will have be added to each rule entry to accommodate the code vector. This, however, may lead to negative encoding gains, as we shall see in Section 5. Moreover, we also note that the ability to decide which ranges in which rule fields should be encoded allows DRES to have full control not only over the encoded rule size but also over both time and space complexities for search key encoding, as discussed in the following section.

3.2 TCAM-Based Search Key Encoding Process

Assume that m_k ranges from the kth rule field (for $k = 1, 2, \ldots, K$) in a rule database are selected for encoding. Then, K search key fields matching against the corresponding K *range tables* must be done to generate an index vector and, hence, an encoded search key. We refer to this process as the search key encoding process. The search key encoding must be performed at wire speed by the NPU on a per-packet basis. As a result, most top-down approaches make the assumption that parallel search key encoding can be done either by a set of processing cores [16], [17] or a set of on-chip or off-chip memories [19]. However, not all of the NPUs on the market today have multiple processing cores or multiple memories or memory interfaces available to implement such encoding functions. To ensure the applicability of DRES to all of the existing NPU-TCAM coprocessor solutions, we propose using the TCAM coprocessor itself for *sequential* search key encoding. In what follows, we discuss how such a TCAM-based solution works and also the related performance trade-offs.

As shown in Fig. 6, $K + 1$ tables are allocated in TCAM, including one encoded rule table and K range tables. In the encoded rule table, a code vector is appended to each rule to form an encoded rule. Each rule in the rule table maps to an action in the associated memory. Likewise, each range in a range table maps to an intermediate index vector in the associated memory. Note that each range in a range table must be represented by multiple TCAM entries (not shown in Fig. 6), and the corresponding intermediate index vector must be duplicated for every entry belonging to the same range.

Each search key encoding involves K TCAM range table matches. The kth search key field is matched against the kth range table, causing an intermediate index vector to be returned. Upon receiving a returned intermediate index vector, the NPU updates the index vector (initially set to NULL) by performing an OR operation between the index vector and the returned intermediate index vector. This process continues until all of the K range table matches are

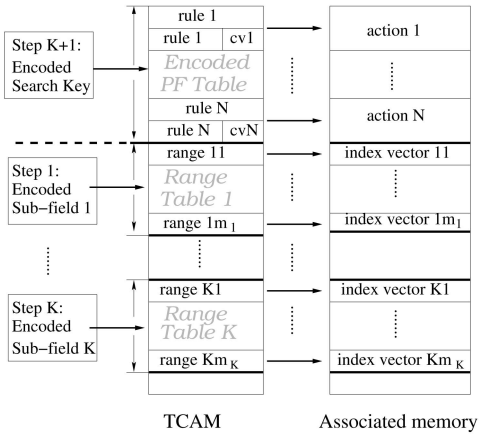


Fig. 6. The encoding process. $K+1$ TCAM searches for PF table matching in a TCAM coprocessor. "cv" stands for code vector.

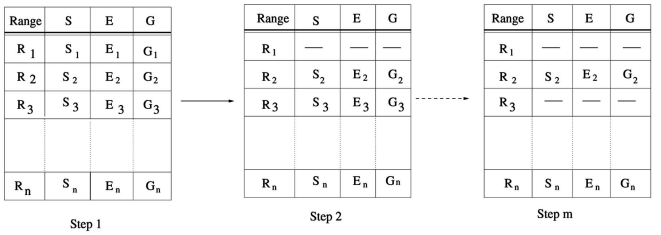


Fig. 7. Dynamic range selection procedure. The tables are range encoding gain tables in each step.

performed. Next, the encoded search key is formed by appending the final index vector to the original search key. This encoded search key is used to match against the encoded rule table. In summary, a rule table lookup with range encoding requires K range table lookups for search key encoding, plus one encoded rule table lookup.

Using TCAM for search key encoding provides an immediate solution applicable to any NPU using a TCAM coprocessor for packet classification. However, heavy use of the TCAM coprocessor for search key encoding may result in reduced packet classification performance due to TCAM access contention. Here, we quantify the performance impact of using TCAM for sequential search key encoding. Consider that a TCAM provided by Ayama [5] runs at 133 MHz, that is, 133 million lookups per second. For wire-speed forwarding at a 10 Gbps line rate, up to 31.3 million packets need to be classified in 1 second in the worst case. Thus, each packet is allowed to have $133/31.3 = 4.28$ TCAM lookups. If a rule entry takes s slots, a search key matching against a PF table requires s lookups. For a five-tuple rule, two lookups are needed, assuming that the slot size is 64 bits. Furthermore, assume that the size of any field with encoded range is not larger than the size of a slot, that is, 64 bits. Then, each range table matching for search key encoding requires one TCAM lookup. Now, if both the source and destination port fields have ranges to be encoded, that is, $K = 2$, each PF table matching requires four TCAM lookups. In this case, wire-speed forwarding at a 10 Gbps line rate can be achieved. For an NPU supporting a 2.5 Gbps line rate, each packet is allowed to perform 17 lookups. DRES achieves storage efficiency by reducing TCAM lookup throughput in a factor of 2. In this case, all five fields can be encoded if necessary. Encoding more fields leads to heavier TCAM access contention but better TCAM storage efficiency. The ability of DRES to allow any subset of rule fields to be selected for encoding makes it possible for DRES to fully exploit this trade-off.

Finally, we note that an alternative approach is to use an SRAM or DRAM for sequential range encoding (all of the existing NPUs have at least one memory interface for external memory access). To achieve deterministic one-memory-access performance, a possible solution is to perform direct range table indexing, as suggested in [16], that is, allow the rule-field-size-worth memory address space to be allocated for a range table. Each entry contains an intermediate index vector corresponding to the range to which the memory address of the entry falls. However, this approach quickly becomes infeasible because the memory occupancy for a range table grows exponentially with the size of the rule field to be encoded. Moreover, in the worst case, a single range update in a range table may cause change in all of the range table entries, making database update a great challenge. To date, there is simply no efficient SRAM/DRAM memory database update algorithm available which allows uninterrupted range table matching.

3.3 Dynamic Range Selection Algorithm

As we shall explain in more detail in this section, for simplicity, we use the bit-map scheme in [14], [16] to encode ranges, that is, each unique range is mapped to a unique bit. This scheme is referred to as the bit-map intersection scheme [18]. This scheme allows us to design an optimal dynamic range selection algorithm. The algorithm is run in the software in the control plane.

Fig. 7 gives a general range selection procedure for selecting m ranges for encoding out of n ranges. S_i is the number of subranges needed to exactly implement range R_i $(i = 1, 2, \ldots, n)$ in a TCAM. E_i is the number of rule entries to implement all of the rules that contain range R_i. G_i is the *encoding gain* for R_i, defined as the number of rule entries that can be eliminated if R_i is encoded. To select m ranges to be encoded, m steps are required, each selecting one range. In the first step, the values of E and G for all of the ranges are calculated, assuming that no range is selected for encoding. Then, the range with the maximum G is selected as the first range for encoding. Suppose that R_1 is selected. In the second step, E and G for all of the ranges, except for R_1, are updated, assuming that all of the rules containing R_1 have R_1 encoded.

Then, the range with the maximum G is chosen to be the second encoded range. This procedure continues until m ranges are selected. The computational complexity for this algorithm is $O(nm)$.

For example, in the PF table, as shown in Fig. 2, there are a total of $n = 7$ ranges in both the source and destination port fields. Ranges R_1-R_5 come from the destination port field. They are $R_1 = \{256 - 512\}$, $R_2 = \{768 - 2,047\}$, $R_3 = \{6,000 - 6,064\}$, $R_4 = \{> 1,023\}$, and $R_5 = \{512 - 1,536\}$. The rest of the two ranges come from the source port field, that is, $R_6 = \{> 1,023\}$ and $R_7 = \{256\text{-}512\}$. Note that, although R_1 and R_7 have the same range value, they are counted as two different ranges because they come from

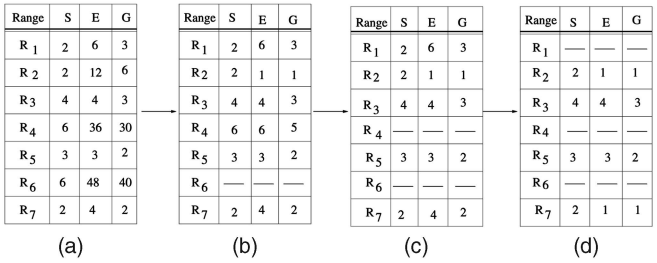


Fig. 8. An example of range encoding gain tables. (a) Initial gain table. (b) After R_6 is chosen. (c) After R_6 and R_4 are chosen. (d) After R_6, R_4, and R_1 are chosen.

different fields. The same is true for R_4 and R_6. $\{< 1,024\}$ in L_8 is a compact range and, hence, is not counted as a range here. Suppose that $m = 3$. Then, the range encoding gain tables are given in Fig. 8.

Initially, E and G, as shown in Fig. 8a, are calculated, without any ranges being encoded. For instance, the encoding gain with respect to R_6 is calculated as follows: As stated in the previous section, six subranges are needed to express R_6 (also R_4) in a TCAM. Since L_5 has R_6 in its source port and R_4 in its destination port, 36 rule entries are needed to exactly implement L_5. Similarly, 12 rule entries are needed to exactly implement L_3 because two subranges are needed to express R_2. Thus, a total of 48 entries are needed to exactly implement both L_3 and L_5, both having R_6 in their source port fields. If R_6 is encoded, the six subranges that represent R_6 in the TCAM are reduced to one encoded bit in an encoded rule. Then, the numbers of rule entries that are required to express L_5 and L_3 are reduced to six and two, respectively. Hence, the gain for encoding R_6 is 40, which is the largest in column G in Fig. 8a. According to the range selection procedure, R_6 is selected to be encoded at this point. Then, the values of E and G for all of the ranges, except for R_6, are updated, assuming that R_6 is encoded. The results are shown in Fig. 8b. Now, R_4 is selected to be encoded since it has the largest G value. Consequently, the values for E and G are updated again, assuming that R_6 and R_4 are encoded. The results are shown in Fig. 8c. Since both R_1 and R_3 have the largest G value among the three, one of them is randomly selected (here, R_1 is selected). Then, the encoding gain table is shown in Fig. 8d.

When the rules with ranges are added or deleted, the encoding gain for some or all of the ranges may be changed. Hence, the above range selection procedure needs to be executed in the software upon each rule update involving range changes. However, since a single rule update involving range changes cannot drastically change the ranks of the existing ranges in terms of encoding gains, most of the executions of the selection procedure are not expected to lead to the encoded range updating.

3.4 Code Vector and Index Vector Encoding Algorithms

DRES is a comprehensive range encoding framework in the sense that it can incorporate any range encoding algorithms in its design. In this paper, the bit-map range encoding algorithms developed in [14], [16], [17], [18] are fully leveraged for code vector and index vector encoding. The most efficient BM algorithm is the $\mathrm{P^2C}$ algorithm proposed in [18], which allows N ranges to be encoded by using only $log_2(N+1)$ bits in the best case, that is, when the ranges do not overlap with one another. The algorithm requires N bits to encode N ranges in the worst case, that is, when any range overlaps with any other ranges. It can be shown that the dynamic range selection problem for the $\mathrm{P^2C}$ algorithm can be translated in polynomial time to a weighted knapsack problem [24], which is NP-complete. Hence, it involves the design of a heuristic range selection algorithm. Moreover, it is much easier to design a lock-free encoded range update algorithm for the BM intersection algorithm than that for $\mathrm{P^2C}$. For these reasons, in this paper, we simply adopt the BM intersection algorithm for DRES. However, the range encoding gain of DRES using the $\mathrm{P^2C}$ algorithm is analytically evaluated in Section 6. Note that, when incorporating either encoding scheme in DRES, it is used in the context of the hybrid encoding approach, that is, encoding a subset of ranges. Hence, in what follows, we simply refer to these schemes as the hybrid encoding algorithm with Bit-Map (BM) Intersection and the hybrid encoding algorithm with $\mathrm{P^2C}$, respectively.

In BM, each bit in a code vector is assigned to a specific encoded range, which can come from any field in a rule. The code vector for a particular range R_i has value 1 in its assigned $(b_i)\mathrm{th}$ bit and "don't care," that is, "$*$" in all other bits. For example, let us encode seven ranges R_i coming from either the destination or source port field of the eight rules L_i, where $i = 1, 2, \ldots, 8$. Suppose that the code vector has 8 bits and the $i\mathrm{th}$ bit (assuming that $b_i = i$) is assigned to R_i. Then, the code vector for R_i has 1 in the $i\mathrm{th}$ bit and $*$ in all other bits. This way, the code vector for R_1 is 1*******.

If a rule has more than one field with encoded ranges, then the code vector for the rule has 1s in the corresponding bits, which are assigned to these encoded ranges, and has "$*$" for all of the other bits. For example, L_3 has R_6 in its source port and R_2 in its destination port. Hence, the code vector for L_3 is *1***1***. If a rule has no encoded ranges, the corresponding code vector is simply wild carded.

Now, let us discuss how we can encode the index vector. As stated in Section 3.2, ranges from different fields must be encoded using different range tables. The index vector encoding method is the same for each range table. In what follows, we first show by example how the index vector is encoded, given that a set of ranges from a single rule field is to be encoded. Let us encode five ranges R_i $(i = 1, 2, \ldots, 5)$, which are taken from the destination port field. Among these ranges, R_1 overlaps with R_5, R_2 overlaps with R_4 and R_5, R_3 is a subrange of R_4, and R_4 overlaps with R_5. The encoded index vectors are given in Table 1. Note that $R_{r_1,r_2,\ldots,r_n}$ represents the common subrange among $R_{r_1}, R_{r_2}, \ldots, R_{r_n}$.

Similarly to the code vector, the $(b_i)\mathrm{th}$ bit in the index vector is assigned to range R_i. The encoding rules used to generate the index vectors can then be stated as follows:

1. For R_i, the $(b_i)\mathrm{th}$ bit in the index vector must be set to 1.
2. If R_i is a subrange of R_j, its index vector must have its $(b_j)\mathrm{th}$ bit set to 1.
3. $R_{r_1,r_2,\ldots,r_n}$ for n overlapping ranges. $R_{r_1}, R_{r_2}, \ldots, R_{r_n}$ needs to be expressed as a separate range if it is a

TABLE 1
Index Vector Encoding for the Destination Port Field

Range	Weight	Index Vector
R_0	0	00000000
R_1	1	10000000
R_2	1	01000000
R_3	2	00110000
R_4	1	00010000
R_5	1	00001000
R_{15}	2	10001000
R_{24}	2	01010000
R_{25}	2	01001000
R_{45}	2	00011000
R_{245}	3	01011000

new range other than any existing encoded ranges. The corresponding index vector must have its $(b_{r_1})\text{th}, (b_{r_2})\text{th}, \ldots, (b_{r_n})\text{th}$ bits set to 1.

4. All other bits in the index vector must be set to 0s.
5. The weight or match priority for a range is equal to the number of 1s in the corresponding index vector.

When the field value in a search key matches a common subrange of n encoded ranges, it means that this field belongs to all n ranges. The greater the number of 1s in the index vector, the more ranges are being matched and the higher the match priority that this subrange must have. Hence, the match priority for a (sub)range can be simply set as the number of 1s in the corresponding index vector. In this paper, we consider an order-based TCAM. The rules in an order-based TCAM are arranged in an ordered list. When multiple rules/ranges are matched, the one in the lowest memory location is selected.

The index vectors in the range Table 1 are encoded as follows: 1) Assign a bit b_i (here, $b_i = i$) to each range R_i $(i = 1, 2, \ldots, 5)$ and set the corresponding bit to 1. 2) Set the bit in the index vector for each range at the bit location corresponding to a superrange of that range, if any. For example, since R_3 has R_4 as its superrange, the fourth bit in the index vector for R_3 is set to 1. 3) Add all of the common subranges to the range table and the corresponding index vectors are set, following rule 3 above. For example, R_{15} is a new subrange of R_1 and R_5 and, hence, it is added to the range table and the first and fifth bits in the corresponding index vector are set. On the other hand, the common range of R_3 and R_4 is the same as R_4 and, hence, it does not need to be encoded again.

Noncompact range R_0, with all wild card bits, is added to every range table and it has the lowest match priority. The index vector for this range is NULL. If the field of a search key does not match any range in a range table, R_0 will be matched. All 0s in the index vector means that no encoded range is matched for this field value. After all five ranges in the destination port field are encoded, the last three bits are not assigned to any ranges yet. These bits can be used to encode ranges from other fields. Let us encode R_6 and R_7 from the source port field. The resulting range table is shown in Table 2.

Based on the above range tables, we now can give a concrete example for search key encoding. Assume that the NPU generates a five-tuple search key $sk = \{1.2.3.4, 5.6.7.8, 1025, 1028, 17\}$ from the IP header of a received packet. Initially, the index vector iv is set to NULL. First, the NPU passes source port number 1,025 to the TCAM coprocessor to match against Table 2. Since port number 1,025 falls into R_6, an intermediate index vector $iv_1 = \{00000100\}$ is returned. Second, the NPU updates the index vector by performing an OR operation, that is, $iv = iv|iv_1 = \{00000100\}$. Third, the NPU passes destination port number 1,028 to the TCAM coprocessor to match against Table 1. Port value 1,028 matches R_2, R_4, R_5, R_{24}, R_{25}, R_{45}, and R_{245}. As a result, the index vector $iv_2 = \{01011000\}$ for R_{245} is returned, because it has the highest weight value. Fourth, the NPU updates the index vector again, that is, $iv = iv|iv_2 = \{01011100\}$. Bit value 1 at the second, fourth, fifth, and sixth bits in iv indicates that the destination port number of the search key falls in the ranges R_2, R_4, and R_5 and the source port number falls into R_6. Finally, an encoded search key is formed by simply appending iv to sk, that is, $\text{encoded search key} = \{sk, iv\}$, which is used to match against the encoded rule table in the TCAM.

TABLE 2
Index Vector Encoding for Source Port Field

Range	Weight	Index Vector
R_0	0	00000000
R_6	1	00000100
R_7	1	00000010

4 Encoded Range Update Process

In a PF table, new rules may be added and old rules may be deleted from time to time. Hence, some popular ranges may eventually become unpopular and vice versa. The dynamic range selection algorithm selects ranges with the largest encoding gains for encoding. If any of the newly selected ranges are different from the existing ones, the encoded ranges which are not selected must be unencoded to release the assigned bits to encode the newly selected ranges. The process for updating encoded ranges and the corresponding rules is called *encoded range update process*.

In this section, we propose a *lock-free* encoded range update algorithm, which allows the encoded range update and the search key encoding/PF table lookup processes to occur simultaneously without impacting the lookup performance. Although the encoded range update process is carried out through a CPU-TCAM coprocessor interface, the search key encoding and PF table lookup processes are performed through an NPU-TCAM coprocessor interface, as shown in Fig. 1. The basic idea is to maintain consistent and error-free rule and range tables throughout the update process, thus eliminating the need for locking the tables. The *lock-free* idea was first proposed in [23] to allow a lock-free rule update for PF in a TCAM. We extended the idea to allow an encoded range update, which involves both range and PF table updates.

Generally speaking, updating a TCAM database without TCAM locking may generate two possible types of incorrect TCAM lookups, that is, erroneous and inconsistent lookups. An erroneous lookup may occur if a TCAM rule gets a match while the rule or its corresponding action is partially

updated. Here, rules and actions are used in a generic sense and may represent ranges and index vectors, respectively. Inconsistent lookup means that a search key does not match the best matched rule. An inconsistent lookup may occur when a match takes place in the middle of a database update process and there is no guarantee of table consistency until the process finishes.

In general, each TCAM slot has a valid bit field associated with it, which allows a rule entry to be activated or deactivated/deleted by simply setting or resetting the valid bit for that rule entry. As explained in [23], the key to avoiding erroneous lookups is to avoid directly overwriting rule fields and/or the corresponding action when that rule entry is active, that is, it may be matched. Instead, any write operations for a rule/action over an existing rule/action must be decomposed into a *write process* including three operations: 1) inactivate the rule, 2) write the rule/action, and 3) activate the rule again. Writing a rule/action into an empty rule entry only requires the last two operations. Likewise, any operations to move a rule-action pair to a new TCAM-associated memory location must be decomposed into a *move process* including 1) using a write process to write the pair to the new location and 2) inactivating the rule at the old location. For DRES, it is assumed that the write and move processes are used and, hence, no erroneous lookups may occur. In what follows, we simply use $write$ and $move$ to stand for write and move processes, respectively.

Inconsistent lookups will not occur if, for each rule move, a search key always matches a rule that would be matched before the rule move. For each rule addition or deletion, a match always results in a matched rule that is the same as that which would be matched either right before or after the rule addition or deletion [23]. Any TCAM database update algorithm that ensures consistent and error-free rule matching does not require TCAM locking for database updating. In what follows, a lock-free update algorithm is described in detail.

In DRES, all of the empty range/rule entries in a range/rule table are kept either at the top or at the bottom of the table. At least one empty range/rule entry is assumed to be available to serve the purpose for consistent encoded range updates. For each table in an order-based TCAM, we assume that entries at the top in a table have higher match priorities than the ones at the bottom.

In DRES, a free bit in the index/code vector is reserved and used for encoded range updates. The encoded range update process can be decomposed into two phases: 1) use the free bit to encode a newly selected range and 2) unencode an encoded range to release the free bit. These two phases are explained separately in the following sections.

4.1 Encoding a Newly Selected Range

For a newly selected range to be encoded, the range that appeared in any rule in the original encoded rule table in the TCAM is exactly implemented. In this case, a consistent rule table is maintained if the range table update is ahead of the rule table update. This is because, when a search key, whether it is encoded or not, matches against an exactly implemented rule, it is nothing more than a search key matching without encoding at all. Hence, in our algorithm, the range table is updated first, followed by the rule table update.

Note that only the range table associated with the field to which the newly selected range belongs needs to be updated. The free bit in the index vector is assigned to encode this range. Then, the newly selected range and all possible common subranges generated by this and the existing encoded ranges are added to the table. If this newly selected range is a superrange of an existing encoded range, the corresponding bit in the index vector for that existing range must be set to 1.

Similarly to the PF table update [23], there are two steps for the range table update. The first step is to consistently move the ranges and their index vectors from top (bottom) to bottom (top) while leaving the entries for the newly selected range and corresponding subranges empty. To ensure table consistency upon each move, the ranges are moved according to their original order. This order of moves maintains the original priority relationships and, thus, the table consistency. If a range is a subrange of the newly selected range, the corresponding bit in the index vector is changed to 1 after it is moved to a new location. This change has no effect on the search key matching since the corresponding bit in the code vector in an encoded rule is wild carded.

The second step is to write the newly selected range, the associated subranges, and their index vectors to the preallocated locations in decreasing priority order (that is, the ranges with higher match priorities are added before the ranges with lower priorities are added).

After finishing the range table update, the following simpler process is used to update the rule table. The rules are moved from top (bottom) to bottom (top), with their relative orders unchanged. If a rule involves the newly encoded range, then, at the new location, the rule is moved and the range is encoded by adding the corresponding bit in its code vector and wild carding the field of the rule in which the range appears. Hence, multiple exactly implemented rule entries belonging to the same rule are reduced to one rule entry after the rule is moved to the new location.

Fig. 9 gives a simple example demonstrating how we can update a rule with a newly encoded range. Assume that L_1 and L_2 are implemented in a TCAM. L_1 has a higher match priority than L_2. The update process must keep the relative match priority for L_1 and L_2 unchanged at all times. L_2 has R_1 in its destination port field. The exact implementation of L_2 with respect to R_1 requires two rule entries, corresponding to two subranges {256-511} and {512}, represented by 2-byte values 1x and 20, respectively, as shown in Fig. 9a. Assume that the first bit of the code vector is assigned to encode R_1 and the range table is already updated. Then, the rule update for L_2 involves two steps: 1) write L_2 to a new location with the newly selected range encoded and 2) delete the rule entries at its old locations. Fig. 9b gives the configuration after Step 1 completes, that is, L_2 is written and activated at slots 7 and 8. Finally, L_1 is moved to slots 5 and 6. During the update process, L_1 always has a valid copy above L_2, which maintains the original priority relationship and provides a consistent table.

4.2 Releasing Encoded Ranges

When the newly selected range is encoded, some rule entries are released. If no free bit is left in the index and

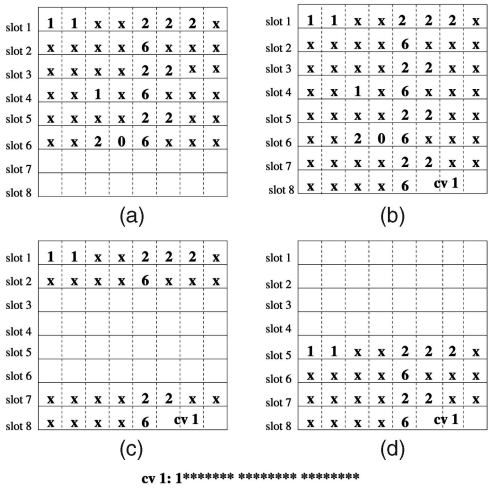


Fig. 9. Updating process in a rule table. (a) L_1 is at slots 1 and 2. L_2 is exactly implemented by two rule entries: One is at slots 3 and 4 and the other is at slots 5 and 6. (b) Write L_2 to a new location with the encoded destination field. (c) Delete the two exactly implemented rule entries at slots 3-6. (d) Move L_1.

code vector, the encoded range with the least encoding gain is unencoded to release a free bit. To unencode a range, the corresponding field in a rule with this encoded range needs to be exactly implemented, which increases the number of rule entries in the table. However, the increased number of rule entries must be less than the reduced number of rule entries by encoding a newly selected range. Otherwise, the encoded range update will not happen in the first place. Hence, only one empty rule entry in the rule table is required to do an encoded range update.

To release an encoded range, the rule table is updated first, followed by the range table update. For the rule table update, the update process changes the encoded range into an exactly implemented range in all of the rule entries having this encoded range. It does this through consistent rule moves and by changing the corresponding bit in the code vector to $*$s. For the range table update, both the encoded range and the derived subranges need to be deleted. To ensure range table consistency, these ranges must be deleted in increasing match priority order. In addition, if another encoded range is the subrange of the range to be unencoded, the corresponding bit in the index vector must be reset. The consistent move process for the range table update is similar to that for adding an encoded range.

4.3 Encoded Range Update Delay

The proposed encoded range update algorithm does not require locking TCAM tables during the update process. However, to ensure consistent and error-free lookups, the algorithm requires a relatively large number of write and delete operations, resulting in a longer update delay. This raises the concern as to whether the update delay would be too large such that the update process cannot keep up with the update requests. We resolve this issue by giving a rough estimation.

We only consider the rule table update delay for doing the encoded range update. The update delay of the range table is neglected because the size of a range table is, in general, much smaller than the size of a rule table. Assume that there are N_{er} rule entries in the rule table. All of the rule entries in the table are moved once for adding a newly encoded range and once for releasing an encoded range. Hence, the number of rule entry writes and deletes is $2N_{er}$ for each encoded range update. Assume that one rule entry write and delete cost 100 ns. Then, for a rule table with 100,000 rule entries, the encoded range update delay is 0.02 second and, for a table with one million rule entries, the update delay is 0.2 second This update delay is negligible given that the range popularity distribution changes much slower than the rule update rate, which occurs on the order of once every few seconds to once every few days.

TABLE 3
Range Statistics of Four Real-World PF Databases

Database	1	2	3	4
Rules	279	183	264	1550
TCAM entries	949	553	1638	2180
Storage Expansion Ratio	3.40	3.02	6.20	1.41
Rule with range (%)	9.3	7.7	54.9	9.8
Rule with range in source port (%)	7.8	6.6	43.2	1.4
Rule with range in destination port (%)	7.8	6.6	20.1	8.5
Unique source ports	13	9	28	34
Unique destination ports	43	40	49	54
Unique ranges in source port	2	2	3	3
Unique ranges in destination port	2	2	4	4

5 Performance Evaluation Based on Real-World Databases

In this section, the performance of DRES is evaluated and compared with Liu's algorithm [16], called the CE algorithm, based on four real-world five-tuple PF databases. The range statistics of the four real-world databases are given in Table 3. The number of rules varies from 183 to 1,550. The *storage expansion ratio*, defined as the number of rule entries divided by the number of rules in a TCAM, is from 1.41 to 6.25. The percentage of rules with ranges spans a wide range, that is, from 7.7 percent to 54.6 percent. The number of unique ports, including both exact port numbers and port ranges, is from 9 to 34 for the source port subfield and from 40 to 54 for the destination port subfield. Some ranges, such as $\{> 1{,}024\}$, can be exactly implemented without taking multiple slots. These ranges do not need to be encoded and are considered exact port numbers. The number of unique ranges found in any of these rule tables is small. For example, for both databases 3 and 4, the maximum number of unique ranges is 4 for the destination port and the maximum number of ranges in both port subfields is 7. The total number of unique ranges found in all four databases is 10.

Table 4 shows the range frequency and the number of subranges to exactly implement the range in TCAM for all four databases. FSP, FDP, TF, and NSUB represent the range frequency in the source port, in the destination port, in both ports, and the number of subranges to exactly implement the range, respectively.

In Table 4, one notes that RN_1 is the most popular range, making up about 88.3 percent of all the ranges. Both databases 1 and 2 have RN_1 and RN_4 in both source and destination ports. Database 3 has RN_1, RN_5, and RN_6 in its

TABLE 4
Range Frequency Distribution

Name	Range	FSP	FDP	TF	NSUB
RN_1	>1023	149	193	342	6
RN_2	109-110	10	11	21	2
RN_3	1645-1646	5	8	13	2
RN_4	33434-33600	2	2	4	6
RN_5	514-1023	2	0	2	8
RN_6	>1024	1	0	1	15
RN_7	33435-33524	0	1	1	7
RN_8	6660-6669	0	1	1	3
RN_9	6000-6099	0	1	1	4
RN_{10}	20-2511	0	1	1	12

source port and RN_1, RN_7, RN_8, and RN_9 in its destination port. Database 4 has RN_1, RN_2, and RN_3 in both the source and destination ports and RN_{10} in the destination port.

Now, we apply DRES to all four databases for a 64-bit slot TCAM, which is widely used in today's TCAM coprocessors. As stated in Section 3, the five-tuple rule has 104 bits. Each rule entry takes two slots in the TCAM and each rule entry has 24 free bits, which is much larger than 7, the maximum number of unique ranges found in the four databases. Hence, no extra slot is needed for range encoding.

In DRES, to encode ranges in both source and destination port fields, two range tables are needed. We assume that all of the range tables are stored in the TCAM and will be counted as part of the storage cost for DRES. In each range table, a range is exactly implemented, with each subrange taking one slot (that is, half a rule entry). If some ranges in the range table overlap with one another and generate a new subrange, that new range must be included in the range table. For the four databases, there is only one such range, which appears in the destination range table for database 4, that is, subrange {1,024-2,511}, which is the common range of RN_1 and RN_{10} and takes five slots. For the four databases, the sizes of the range table in the source (destination) port are 12(12), 12(12), 29(20), and 22(10) (in slots), respectively. In practice, due to the possible encoded range updates, a range table must be configured to be much larger than the maximum size 29. Let us assume that 60 slots are allocated for each range table, doubling the maximum size found in the four databases.

Note that, in DRES, as soon as either a range table is fully utilized or all of the free bits are exhausted, DRES can stop encoding more ranges to avoid overflowing the range table. In contrast, for CE, all of the unique port values for an encoded field must be encoded, each consuming a separate bit in the code vector. Hence, the size of the code vector must be larger than the sum of the number of unique port values for each encoded field. Unlike DRES, which has full control over the code vector size, the CE algorithm fails if the number of port values is more than what the code vector can accommodate. Therefore, in practice, a code vector with a sufficiently large number of bits must be configured to ensure that the scheme does not fail. The CE algorithm allows a total number of 40 bits (16 bits in the port field plus 24 free bits) to be used by the code vector to encode one port field. If the number of unique port values of that port field exceeds 40, an extra slot must be configured for each rule entry.

TABLE 5
TCAM Storage Expansion Ratio with and without Encoding

Database set		1	2	3	4
Without encoding		3.40	3.02	6.20	1.41
Encoding Source port	DRES	1.50	1.49	2.07	1.4
	CE	1.39	1.33	1.95	1.38
Encoding Destination port	DRES	1.50	1.49	3.00	1.05
	CE	2.09	1.99	4.33	1.54
Encoding Both ports	DRES	1.22	1.33	1.23	1.04
	CE	1.5	1.5	1.5	1.5

From Table 3, we see that, although the number of unique port values in the source port field is less than 40 in all of the databases, the numbers of unique port values in databases 3 and 4 reach 28 and 34, respectively, suggesting that an extra slot should be configured to ensure that there are no overflows of the code vector space. In the performance evaluation of CE, however, we assume that no extra slot is allocated for encoding the source port field. However, an extra slot is assumed to be allocated for encoding the destination port field and both the source and destination port fields, simply because the number of unique port values for the destination port field reaches 40 or more for all of the databases. Note that, for CE, the TCAM storage overhead due to the range tables is not counted, which is, in general, very large because all of the unique source and destination ports must be included in the range tables.

Table 5 shows the TCAM storage expansion ratio after range encoding. For ease of comparison, the TCAM storage expansion ratio without range encoding, as listed in Table 3, is also listed in Table 5. After encoding ranges in both the source and destination fields in DRES, each rule takes one TCAM rule entry. The rule expansion only comes from the range table (30 rule entries). If only the source (destination) port range is encoded, the number of TCAM entries for the encoded rules of four databases are 389(389), 243(243), 516(762), and 2,124(1,595), respectively. We observe that DRES achieves better encoding gains than CE when ranges in either both port fields or the destination port field only are encoded. CE achieves better encoding gain when the source port field only is encoded. This, however, is based on the assumption that, in CE, no extra slots are needed for the source port field encoding. The relatively reduced encoding gain in DRES is due to the added overhead of a range table in the TCAM. Note that, due to the lack of control of the code vector size, CE may lead to negative encoding gains, as in the case for the encoding of the destination port field and both port fields for database 4. We also note that the TCAM storage expansion ratio for CE is lower bounded at 1.5 when one extra slot has to be added to each rule entry. This is simply because each rule entry takes two slots and adding one extra slot expands the TCAM by 50 percent, even if all of the ranges are encoded, as in the case in Table 5 when both ports are encoded. In contrast, DRES can reduce the TCAM storage expansion ratio to close to 1, especially when the rule table size is relatively large compared with the range table sizes, as in the case for database 4.

In summary, DRES can significantly improve the overall TCAM storage efficiency for range matching. The key to

achieving this performance gain is to use a hybrid encoding scheme rather than the CE scheme.

6 Analytical Performance Evaluation

The performance analysis in the previous section is based on existing real-world databases, which may not capture the worst-case scenarios that can occur in the future as the Internet becomes more and more policy-based. In this section, we use a probabilistic model to analyze the performance of DRES in a wide range of parameters. This probabilistic model allows us to perform a rather general analysis of DRES, without having to resort to simulation, which is often tedious and can only sample a limited number of parameter settings.

In our model, each rule has K fields and each field can be a range or an exact number. To avoid modeling unnecessary details, we use compact ranges instead of both exact numbers and compact ranges. For example, source port numbers 80 and 23 and the source port range {< 1,024} are instances of the compact range for the source port field.

The following parameters are defined in the model:

- N_γ: The total number of rules in the PF table.
- N_k: The total number of unique noncompact ranges in field k.
- p_k: The probability that noncompact ranges occur in field k.
- $p_{k,j}$: The probability that the jth unique range occurs in field k.
- $p^\gamma_{k,j}$: The probability that the jth unique noncompact range occurs in field k.
- $m_{k,j}$: The number of subranges to exactly implement the jth range in field k in TCAM.
- $\omega(p_{k,j}, m_{k,j})$: The total number of TCAM rule entries required to implement all the rules in a TCAM.
- η: The TCAM storage expansion ratio.

Without loss of generality, the compact range in field k is specified as $j = 0$. In other words, $p_{k,0}$ is the probability that the compact range appears in field k and $m_{k,0} = 1$ $(k = 1, 2, \ldots, K)$. Then, $p_{k,j}$ for the kth field can be expressed as

$$p_{k,j} = \begin{cases} p_k p^\gamma_{k,j} & j = 1, \ldots, N_k \\ 1 - p_k & j = 0. \end{cases} \quad (1)$$

For simplicity, we assume that the range distributions for different fields are independent of each other. Then, we have

$$\omega(\{p_{k,j}, m_{k,j}\}) = N_\gamma \prod_{k=1}^{K} \left(\sum_{j=0}^{N_k} m_{k,j} p_{k,j} \right). \quad (2)$$

Note that this expression applies to a PF table with or without range encoding. The only difference is that the $m_{k,j}$ value changes to 1 after the jth range in the field k is encoded, while $p_{k,j}$ remains unchanged. The TCAM storage expansion ratio η can be expressed as

$$\eta = \omega(\{p_{k,j_k}, m_{k,j_k}\})/N_\gamma. \quad (3)$$

Note that η is a function of $p_{k,j}$, which is further dependent on p_k and $p^\gamma_{k,j}$, as indicated in (1). As shown in Table 3, p_k can take a wide spectrum of values, ranging from 1.4 percent to as high as 43.2 percent. In our numerical studies, p_k is set to 20 percent and 30 percent in two different cases. On the other hand, $p^\gamma_{k,j}$ is highly concentrated on a few popular ranges. The statistics in Table 4 indicates that {> 1,023} is the most popular one, appearing with a frequency of about 88.3 percent. The second popular one is {109-110}, which appears with a frequency of about 6 percent. This popularity distribution closely follows a Zipf-like distribution, with $z = 3$ [3]:

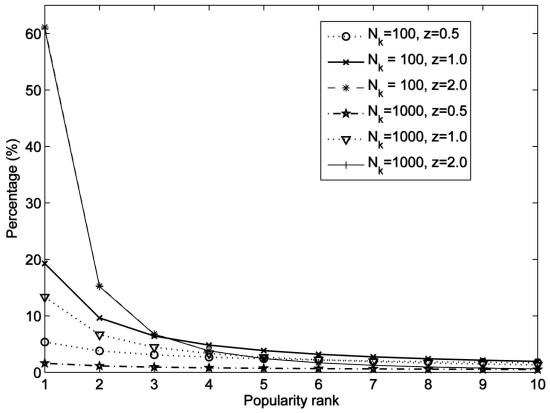


Fig. 10. The Zipf distribution for the 10 most popular ranges.

$$p^\gamma_{k,j} = c/\xi^z(j), \qquad j = 1, \ldots, N_k, \quad (4)$$

where z is the Zipf coefficient, $\xi(j)$ is the rank of the jth range, and c is a normalization factor. In this distribution, $p^\gamma_{k,j}$ is proportional to the popularity rank of range j. This distribution is used in our model to characterize $p^\gamma_{k,j}$. Note that DRES becomes more efficient as z gets larger because the popular ranges are concentrated on fewer ranges as z increases.

If the number of unique noncompact ranges is smaller than the number of ranges that can be encoded, the performance analysis of DRES is trivial. Hence, we consider the situation when N_k is large, for example, 10^2 to 10^4. Moreover, we consider a wide range of z values, for example, from 0.5 to 3.0. To allow one to get a feel of exactly what these parameters mean in terms of numbers, Fig. 10 plots the frequencies for the top 10 popular noncompact ranges at $z = 0.5, 1.0$, and 2.0 and $N_k = 100$ and $1,000$. First, one notes that, for $z = 2.0$, the curves for $N_k = 100$ and $1,000$ are almost identical and cannot be distinguished from each other in the plot. This is due to the fact that, as z gets larger, the few topmost popular ranges become more and more dominant. As a result, N_k value has less effect on the relative frequencies of the popular ranges. Second, as z goes as low as 0.5, the frequencies for even the most popular range constitute less than 7 percent of the total range appearance frequencies at both $N_k = 100$ and $1,000$ and the frequencies reduce very slowly as the range popularity rank drops. Therefore, we believe that the parameter range $z = 0.5$ to 3 should be wide enough to cover most of the worst-case scenarios that may occur.

We again consider a PF table with 104-bit five-tuple rules and a TCAM slot size of 64 bits. Assume that only the two ports may have ranges and the range distributions are the

same for both ports. As one bit is reserved for encoded range updating, 23 out of the 24 free bits are available for range encoding, with 12 bits for the source port ranges and 11 for the destination port ranges. $m_{k,j}$ is set to 6 for all of the ranges, which is the average value found in the four real-world databases.

Apparently, as the number of unique port numbers becomes large, the CE scheme becomes unviable due to the need to accommodate large range tables in TCAM and potentially oversized encoded rules. Hence, in this study, we do not compare a hybrid encoding scheme (used in DRES) with any CE scheme. Instead, in this study, we compare two hybrid encoding schemes that may be adopted by DRES, that is, bit-mapping (BM) and $\mathrm{P^2C}$. Note that, in the previous section, the performance of $\mathrm{P^2C}$ is not studied because DRES using BM has already given the best possible performance in terms of TCAM storage efficiency (one rule per TCAM rule entry). In addition, note that, although a comprehensive solution for DRES using BM has been described in this paper, whether or not a comprehensive solution for DRES using $\mathrm{P^2C}$ can be developed is yet to be studied. The key challenges to developing such a solution include the design of a dynamic range selection heuristic and a lock-free dynamic encoded range update algorithm. Since the study in this section is only concerned with the TCAM storage efficiency, we simply assume that a comprehensive solution for DRES using $\mathrm{P^2C}$ already exists.

With 23 bits used for range encoding, BM can encode 23 ranges. The number of encoded ranges using $\mathrm{P^2C}$ is dependent on the range structure. In the best case, when no range overlaps with any other range, a total of $2^{12}-1=4,095$ ranges in the source port and $2^{11}-1=2,047$ ranges in the destination port can be encoded. Such a large number is usually large enough to encode all of the ranges that may appear in a PF table. In the worst case, when any range can overlap with any other range, a total of 23 ranges can be encoded, which is the same as in BM. In the average case, assume that all of the selected ranges can be put into three layers such that any range from a given layer does not overlap with any other range from the same layer and each layer has the same number of ranges (for more information on the concept of layer, refer to [18]). Then, a total of $(2^4-1)\times 3=45$ ranges in the source port and $(2^4-1)\times 2+2^3-1=37$ ranges in the destination port can be encoded. In summary, in our numerical analysis, a total of 23 (12 from the source and 11 from the destination port fields) ranges are encoded for BM and a total of 82 ranges (45 from the source and 37 from the destination port fields) are encoded for $\mathrm{P^2C}$.

In our numerical analysis, we neglect the TCAM overhead for accommodating the two range tables. This approximation becomes more and more accurate as the rule table size gets larger. For example, assume that each encoded range generates one extra subrange that needs to be encoded as well. Then, a total of $(45+37)\times(1+1)\times 6=984$ slots or 492 rule entries are needed for all of the range tables. This constitutes only 4.9 percent of the total TCAM memory used to support a rule table with 10,000 rule entries. It further drops to 0.49 percent if a rule table with 100,000 rule entries is supported. As the TCAM resource is likely constrained only when the rule table size becomes moderately large, this approximation should be a good one in practice.

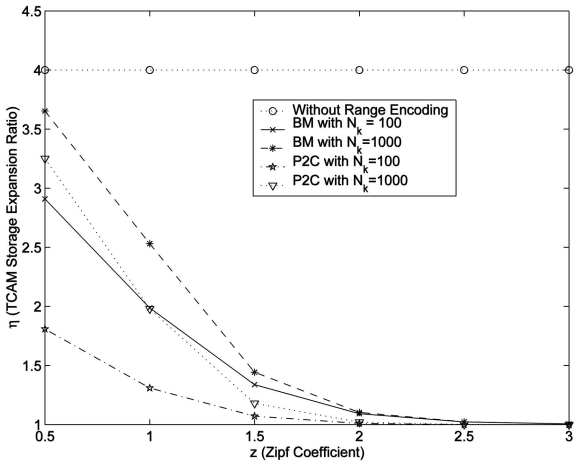


Fig. 11. The storage expansion ratio versus the Zipf coefficient.

6.1 Impact of the Zipf Coefficient

We now study the impact of the Zipf coefficient on the performance of DRES. p_k is set at 20 percent for this case study. Fig. 11 shows the TCAM storage expansion ratios η without range encoding, encoding by BM, and encoding by $\mathrm{P^2C}$ at $N_k=100$ and $1,000$, respectively. The storage efficiency is calculated by (3). Without range encoding, $\eta=4$. Note that, without range encoding, η is a function of p_k only and is independent of other parameters in our model. For BM, η is reduced to about 2.9 and 3.6 at $z=0.5$, saving 28 percent and 10 percent of TCAM resources at $N_k=100$ and $1,000$, respectively. About 50 percent and 40 percent of the TCAM resource are saved at $z=1.0$ and $N_k=100$ and $1,000$, respectively. η further reduces to less than 1.5 at $z=1.5$ and quickly converges to 1 as z further increases, independent of N_k. This is because, as z increases, the top 10 popular ranges become so dominant that the rest of the ranges do not contribute much to the TCAM expansion. The storage expansion using $\mathrm{P^2C}$ is reduced by up to 40 percent from that of BM. The results indicate that $\mathrm{P^2C}$ is much more efficient than BM in general.

6.2 Impact of the Number of Unique Ranges

We set $p_k=30\%$. Fig. 12 plots η without range encoding and with range encoding at $z=1.0$ and 1.5 as a function of N_k.

In Fig. 12, we can see that DRES significantly reduces η throughout the wide range of N_k, for example, from 10^2 to 10^4. Even at $z=1.0$ and $N_k=10^4$, the TCAM storage space is saved by 30 percent for BM and 50 percent for $\mathrm{P^2C}$. The TCAM saving increases fast and becomes less sensitive to N_k as z further increases. η becomes less than 1.7 for $z=1.5$. $\mathrm{P^2C}$ gives better performance than BM by further saving about 30 percent of the TCAM storage space throughout the parameter range. The above numerical results indicate that DRES can significantly improve the TCAM storage efficiency in a wide spectrum of parameter ranges.

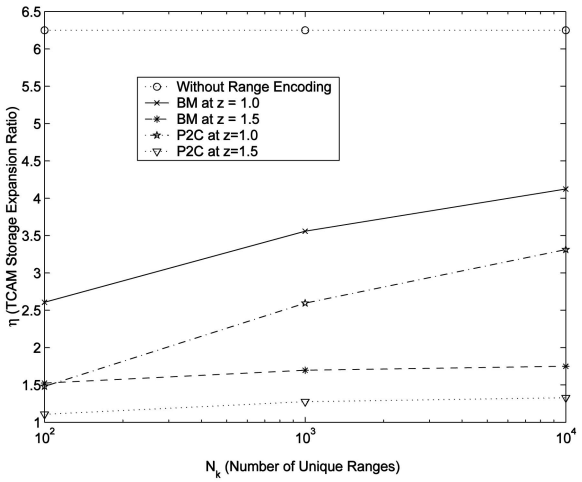


Fig. 12. The storage expansion ratio versus the number of unique ranges in each port field.

7 Conclusions and Future Work

In this paper, DRES is proposed to improve the TCAM storage efficiency in support of range matching. As a bottom-up approach, DRES is designed to solve the range matching issue for the existing network processors using a TCAM coprocessor for PF. DRES includes all of the necessary ingredients for its implementation in a network processor and its TCAM coprocessor with only a software upgrade.

A salient feature of DRES is its ability to have full control over the encoded rule size and to exploit the TCAM structure for maximizing the encoding gains. DRES provides a range selection algorithm to allow a dynamic selection of ranges for encoding so that the maximum encoding gain is maintained whenever the rules are updated. Moreover, DRES uses a lock-free encoded range update algorithm that allows encoded ranges to be updated without impacting the rule matching process. The performance on real-world databases shows that DRES can significantly reduce the TCAM storage expansion from 6.20 to as low as 1.23. Our statistical analysis on a probabilistic model demonstrates that DRES can significantly improve the TCAM storage efficiency in a wide spectrum of parameter ranges.

To allow for the overall simple design, DRES adopts the bit-map intersection encoding scheme, which is not the most effective range encoding scheme. This seems to be sufficient for today's PF databases, as we showed in Section 5. However, as demonstrated in Section 6, using the P^2C encoding scheme rather than the bit-map intersection encoding scheme in DRES can be much more effective in terms of the TCAM storage efficiency. Hence, our future work will focus on developing a comprehensive solution for DRES using P^2C for range encoding. The key challenges in achieving this goal include the design of an efficient dynamic range selection heuristic and a lock-free encoded-range update algorithm.

Acknowledgment

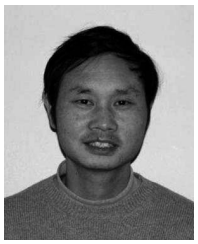
The authors would like to express their gratitude to Professor Jonathan Turner and Dr. David Taylor from Washington University, St. Louis, for kindly sharing their real-world databases and the related statistics. They also thank the anonymous reviewers and Professor Matthew Wright for their constructive comments, which greatly helped to improve the quality of this paper. This work was partially supported by the University Grants Committee (UGC) of Hong Kong under the CERG Grant PolyU 5293/06E and the China 973 program (2007CB310701).

References

[1] "AMCC Ships 10-Gbit/s Processor," *Light Reading*, Mar. 2002.
[2] F. Baboesu and G. Varghese, "Scalable Packet Classification," *Proc. ACM SIGCOMM '01*, pp. 97-108, 2001.
[3] L. Breslau, P. Cao, J. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-Like Distributions: Evidence and Implications," *Proc. IEEE INFOCOM '99*, pp. 126-134, 1999.
[4] H. Che, Y. Wang, and Z. Wang, "A Rule Grouping Technique for Weight-Based TCAM Coprocessors," *Proc. 11th Symp. High-Performance Interconnects*, pp. 32-37, Aug. 2003.
[5] Cypress Ayama 10K/20K NSE Series TCAM Products, http://www.cypress.com, 2008.
[6] A. Feldman and S. Muthukrishnan, "Tradeoffs for Packet Classification," *Proc. IEEE INFOCOM '00*, pp. 1293-1302, 2000.
[7] A. Gottlieb, "Optimizing Next-Generation Application-Dependent Packet Classification," http://www.ardeligroup.com/images/app_depend_pack_class.pdf, 2007.
[8] P. Gupta and N. McKeown, "Algorithms for Packet Classification," *IEEE Network*, pp. 24-32, 2001.
[9] P. Gupta and N. McKeown, "Packet Classification on Multiple Fields," *Proc. ACM SIGCOMM '99*, pp. 147-160, 1999.
[10] P. Gupta and N. McKeown, "Packet Classification Using Hierarchical Intelligent Cuttings," *Proc. Seventh Symp. High Performance Interconnects*, 1999.
[11] "IDT Introduces IP Coprocessors with Seamless QDR Interface to Intel's New Network Processors," http://www.idt.com/news/Feb02/02_26_02_1.html, Feb. 2002.
[12] "IDT Samples Industry's First Network Search Engines with a Fully Integrated Interface for AMCC Network Processors," http://www.idt.com/news/Mar03/03_31_03_1.html, 2007.
[13] M.E. Kounavis, A. Kumar, H. Vin, R. Yavatkar, and A.T. Campbell, "Directions in Packet Classification for Network Processors," *Proc. Second Workshop Network Processors*, 2003.
[14] T. Lakshman and D. Stiliadis, "High-Speed Policy-Based Packet Forwarding Using Efficient Multi-Dimensional Range Matching," *ACM SIGCOMM Computer Comm. Rev.*, vol. 28, no. 4, pp. 203-214, Oct. 1998.
[15] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for Advanced Packet Classification with Ternary CAMs," *Proc. ACM SIGCOMM*, 2005.
[16] H. Liu, "Efficient Mapping of Range Classifier into Ternary-CAM," *Proc. 10th Symp. High-Performance Interconnects*, pp. 95-100, Aug. 2002.
[17] J. van Lunteren and A.P.J. Engbersen, "Dynamic Multi-Field Packet Classification," *Proc. IEEE Global Telecomm. Conf.*, pp. 2215-2219, Nov. 2002.
[18] J. van Lunteren and A.P.J. Engbersen, "Fast and Scalable Packet Classification," *IEEE J. Selected Areas in Comm.*, vol. 21, no. 4, pp. 560-571, 2003.
[19] E. Spitznagel, D. Taylor, and J. Turner, "Packet Classification Using Extended TCAMs," *Proc. 11th IEEE Int'l Conf. Network Protocols*, pp. 120-131, Sept. 2003.
[20] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and Scalable Layer-Four Switching," *Proc. ACM SIGCOMM '98*, pp. 192-202, 1998.
[21] V. Srinivasan, S. Suri, and M. Waldvogel, "Packet Classification Using Tuple Space Search," *Proc. ACM SIGCOMM '99*, pp. 135-146, 1999.
[22] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable High-Speed Prefix Matching," *ACM Trans. Computer Systems*, vol. 19, no. 4, pp. 440-482, 2001.
[23] Z. Wang, H. Che, M. Kumar, and S. Das, "CoPTUA: Consistent Policy Table Update Algorithm for TCAM without Locking," *IEEE Trans. Computers*, vol. 53, no. 12, pp. 1602-1614, Dec. 2004.

[24] K. Zheng, C. Hu, H. Lu, and B. Liu, "A TCAM-Based Distributed Parallel IP Lookup Scheme and Performance Analysis," *IEEE/ACM Trans. Networking*, vol. 14, no. 4, pp. 863-875, Aug. 2006.
[25] K. Zheng, H. Che, Z. Wang, B. Liu, and X. Zhang, "DPPC-RE: TCAM-Based Distributed Parallel Packet Classification with Range Encoding," *IEEE Trans. Computers*, vol. 55, no. 8, pp. 947-961, Aug. 2006.

Hao Che received the BS degree from Nanjing University, China, in 1984, the MS degree in physics from the University of Texas at Arlington in 1994, and the PhD degree in electrical engineering from the University of Texas at Austin in 1998. From 1998 to 2000, he was an assistant professor of electrical engineering at Pennsylvania State University, University Park. From 2000 to 2002, he was a system architect with Santera Systems, Plano, Texas. Since September 2002, he has been an assistant professor of computer science and engineering in the Department of Science and Computer Engineering at the University of Texas at Arlington. His research interests include network architecture and design, network resource management, multiservice switching architecture, and network processor design. He is a senior member of the IEEE.



Zhijun Wang received the MS degree in electrical engineering from Pennsylvania State University, University Park, in 2001 and the PhD degree in computer science and engineering from the University of Texas at Arlington in 2005. He is currently an assistant professor in the Department of Computing at Hong Kong Polytechnic University. His research interests include high-speed network, network security, traffic control, data management in mobile networks, and peer-to-peer networks.

Kai Zheng received the BS degree in electronic engineering from Beijing University of Posts and Telecommunications in 2001 and the PhD degree in computer science from Tsinghua University in 2006. At that moment, his research interests included high-performance IP address lookup, packet classification, and pattern-matching-related network security issues. He joined the System Research Group at the IBM China Research Laboratory in July 2006. His current research interests include computer architecture-related topics such as high-performance deep packet inspection and emerging network applications on the mass multicore platforms. He is a member of the IEEE.

Bin Liu received the MS and PhD degrees in computer science and engineering from Northwestern Polytechnical University, Xi'an, China, in 1988 and 1993, respectively. From 1993 to 1995, he was a postdoctoral research fellow in the National Key Laboratory, Broadband Switching Technologies, Beijing University of Post and Telecommunications. In 1995, he joined the Department of Computer Science and Technology at Tsinghua University as an associate professor, where he mainly focuses on multimedia networking, including ATM switching technology and Internet infrastructure and where he has been a full professor since 1999. He is currently a cochair of the IEEE International Conference on Communications (ICC) 2008 and the ANI Symposium. He also serves on the TPC of the IEEE INFOCOM 2008. He is an associate editor for the *Security and Communication Networks Journal*. His research interests include high-performance switches/routers, network processors, traffic measurement and management, and high-speed network security. He is a coauthor of the book *High-Performance Switches and Routers* and is the holder of 16 patents in China. He is a member of the IEEE.