A 2-Level TCAM Architecture for Ranges

Yeim-Kuan Chang

Abstract—As the demand for high-quality Internet increases, emerging network applications are spurring the need for faster, feature-rich, and cost-effective routers. Multifield packet classification in routers has been a computation-intensive data path function for software implementation. Therefore, solutions for packet classification based on hardware design, such as Ternary Content Addressable Memory (TCAM), are necessary to sustain gigabit line processing rate. Traditionally, TCAMs have been designed for storing prefixes. However, multifield packet classification usually involves two fields of arbitrary ranges that are TCP/IP layer 4 source and destination ports. Storing ranges in TCAMs relies on decomposing each individual range into multiple prefixes, which leads to range-to-prefix blowout. To reduce the total number of prefixes needed to represent all ranges, this paper proposes a 2-level TCAM architecture and two range-to-prefix conversion schemes. In the first proposed scheme, designed for disjoint ranges, the maximum number of entries needed in TCAM is $2m-1$ for m disjoint ranges. In the second proposed scheme, designed for contiguous ranges, only m TCAM entries are needed. In a general case of n arbitrary ranges, all ranges can first be converted into disjoint ranges or contiguous ranges and then the proposed algorithms can be applied. As a result, only $4n-3$ TCAM entries are needed for the disjoint ranges and only $2n+1$ TCAM entries are needed for contiguous ranges. This paper also proposes insertion and deletion algorithms to accommodate incremental changes to the range sets. The experiments made show that the proposed range-to-prefix conversion schemes perform better than the existing schemes in terms of the number of required TCAM entries and execution time for range update operations.

Index Terms—TCAM, ranges, disjoint ranges, contiguous ranges.

1 Introduction

Firewalls, intrusion detectors, or routers with differentiated Quality of Service use packet classification to prevent unauthorized accesses to network resources or to provide differentiated services. Packet classification is a hardware or software mechanism that uses multiple fields in the packet header to classify packets according to specified filtering rules. Multidimensional packet classification allowing range matches is difficult to implement at high speed with a large filtering rule set.

Ternary Content Addressable Memories (TCAMs) [8], [14], [19] are a well-known hardware solution for IP lookups and packet classification. TCAMs are fully associative memories in which each cell takes one of three logic states: "0," "1," and "*" (don't care). Each TCAM entry consists of multiple cells. TCAM entries are suitable for storing prefix fields in the rule tables for packet classification. TCAM is designed in such a way that all of the entries are compared in parallel against multiple fields of the incoming packets. If multiple matched entries are found, the highest priority entry in TCAM is typically returned as the result. Thus, a matched entry, if it exists, can be found in a single TCAM access cycle. In contrast, conventional network processor-based software and ASIC-based hardware designs [1], [2], [3], [4] that use various data structures may require multiple memory accesses for a single lookup. Also, the update process in TCAMs is generally simpler [8].

Despite these advantages, TCAMs have three disadvantages: 1) high TCAM manufacturing cost due to low chip density, 2) high power consumption due to the parallel execution of TCAM entries, and 3) low TCAM utilization due to the inefficient method of storing range fields. One common example of ranges is the source and destination port fields in rule tables for packet classification. These disadvantages may slow down the process for Internet vendors to adopt TCAMs in packet forwarding devices. The cost-to-density-ratio of TCAM has been dramatically improved in recent years. A lower power consumption for TCAMs can be achieved by means of circuit designs that reduce the matchline voltage swing, the switching activity, or the active matchline capacitance [14]. The partitioning techniques proposed in [10], [11] can also reduce the TCAM power consumption, however, but only for prefix fields.

The last problem to be solved is how to store ranges in TCAMs efficiently. One simple solution is to convert each range into multiple prefixes and then store them in TCAMs as usual. However, this simple solution may result in the range-to-prefix blowout. For example, the range [1, 14] in the 4-bit address space is converted into six prefixes that are 0001, 001^*, 01^{**}, 10^{**}, 110^*, and 1110. In the worst case, a W-bit range will be expanded into $2(W-1)$ prefixes. If more than one range is specified in a rule, the number of the expanded prefixes is multiplied. For example, the standard 5-tuple rule tables use two range fields that are the 16-bit source and destination port numbers. The direct range-to-prefix conversion could result in 900 expansions for a single rule. This range-to-prefix blowout translates into the TCAM, which is 900 times larger than the original rule table. This huge size of TCAM is prohibitive because of the excessive power consumption. Therefore, researchers have proposed two approaches to solving the problem of range-to-prefix blowout. The first

• *The author is with the Department of Computer Science and Information Engineering, National Cheng Kung University, No. 1, Ta-Hsueh Road, Tainan, Taiwan, ROC. E-mail: ykchang@mail.ncku.edu.tw.*

Manuscript received 7 Apr. 2005; revised 10 Nov. 2005; accepted 3 May 2006; published online 20 Oct. 2006.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0100-0405.

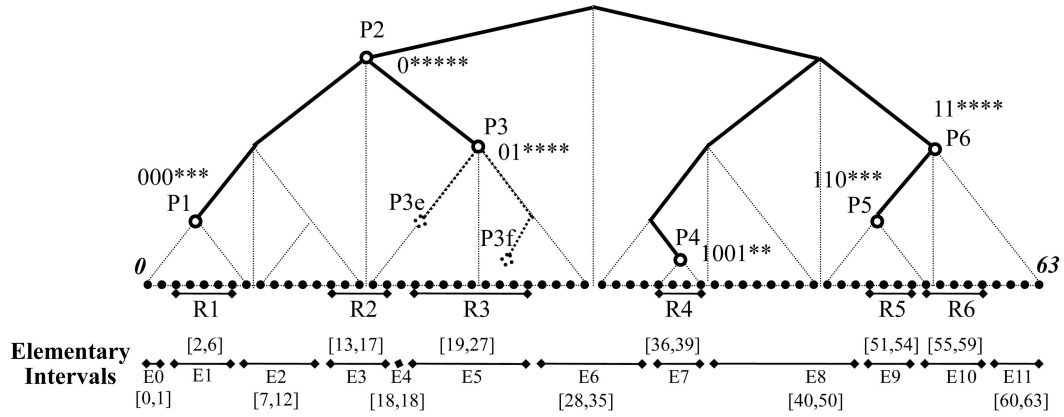

Fig. 1. The range set $G = \{R1, R2, R3, R4, R5, R6\}$ and $EI = \{E0, \ldots, E11\}$.

approach is to design a new TCAM-like circuit that can store a range with only one entry, while the second approach (2) is to invent a new mechanism that converts a range into one or two prefixes.

The authors of [6] designed a hardware that uses the first approach. We refer to this hardware design as range CAM (RCAM) since the circuit of each RCAM entry contains two endpoint addresses and proper comparing logic. A similar design that augments the traditional TCAM with the range check circuits was proposed in [7]. The circuit of RCAMs is much more complex than that of TCAMs. Currently, the real implementations of RCAMs from hardware vendors are not yet available. Therefore, it is desirable to have efficient storagewise schemes to store ranges in TCAMs.

An efficient range-to-prefix conversion scheme based on the elementary intervals is proposed in [1]. If there are m elementary intervals from a set of n ranges, this scheme needs $2m$ prefixes. The worst-case number of prefixes needed is $4n + 2$ for a set of n ranges because m is at most $2n + 1$. Another range-to-prefix conversion scheme is proposed in [13] for solving the point intersection problem. This scheme uses the concept of the *longest common prefix* (LCP) of two addresses to build prefixes. Each disjoint range is converted into two prefixes. The two converted prefixes with the range endpoints are stored in two separate 2-level TCAM-RAM architectures. Thus, this scheme requires $2n$ TCAM entries (prefixes) for a set of n disjoint ranges. Also, this scheme needs two TCAM lookups to complete a search and two TCAM operations for a range insertion or deletion.

In this paper, we propose two range-to-prefix conversion schemes. Our algorithms explore the relationship between disjoint ranges and obtain better results than the schemes proposed in [1] and [13]. We use only one 2-level TCAM-RAM architecture, instead of two as proposed in [13]. Our TCAM-RAM architecture is augmented with a 5-tuple rule structure consisting of two endpoint addresses and three rule IDs. After TCAM returns the matched result, an extra operation of checking the two endpoint addresses is performed to obtain the final matched rule ID. The proposed architecture is suitable for software implementation.

The first proposed range-to-prefix conversion scheme is designed for disjoint ranges. Each disjoint range can be represented by the longest prefix that completely covers it. However, some ranges must be split into two ranges in order to satisfy the proposed 5-tuple rule structure. As a result, this scheme needs at most $2m - 1$ prefixes for m disjoint ranges. The second proposed range-to-prefix conversion scheme is designed for contiguous ranges. No range splitting is needed. At most m prefixes are needed for m contiguous ranges. In general, the first scheme is suitable when the degree of range overlapping is low (that is, only a few ranges overlap the others). The second scheme is suitable when the degree of range overlapping is high.

For a set of n arbitrary ranges, we use a two-step approach. If Step 1 converts the ranges into disjoint ranges (at most $2n - 1$), then Step 2 applies the first proposed scheme. As a result, this two-step method generates at most $4n - 3$ prefixes needed in TCAM. If Step 1 converts the original ranges into contiguous disjoint ranges (elementary intervals, at most $2n + 1$), then Step 2 applies the second proposed scheme. As a result, this two-step method generates at most $2n + 1$ TCAM entries.

Incremental insertion and deletion algorithms are also proposed in this paper. Specifically, an optimized insertion algorithm is developed for the first proposed scheme to reduce the number of TCAM insertion and deletion operations. As a result, the execution time for inserting or deleting a prefix in/from TCAM can be reduced.

The rest of the paper is organized as follows: Section 2 illustrates the notations and definitions needed in this paper. Section 3 describes the proposed 2-level TCAM architecture. The proposed algorithms are divided into three subsections, i.e., disjoint ranges, contiguous ranges, and arbitrary ranges. Section 4 presents the evaluation of the performance results and Section 5 contains the conclusion.

2 Notations and Definitions

To describe the proposed schemes clearly, we use the binary trie to represent the prefixes and use the range set G in Fig. 1 to explain the notations and definitions needed in the paper.

Notations.

W: The maximum number of bits in the address space of ranges. A W-bit address A must satisfy the condition of $0 \leq A \leq 2^W - 1$.

Range: A series of consecutive addresses. A W-bit range $R = [L, U]$ satisfies $0 \le L \le 2^W - 1$ and $0 \le U \le 2^W - 1$, where $L \le U$. If $L = U$, we call R the *singleton* range.

R_{min} and R_{max}: The lowest and highest addresses of the range R. Specifically, $R_{min} = L$ and $R_{max} = U$ when $R = [L, U]$.

$b_{W-1} \dots b_h * \dots *$: *Ternary format* of prefixes. It represents a prefix of length $W - h$ and $b_j = 0$ or 1 for $W - 1 \ge j \ge h$ and $b_j = *$ for $j < h$. For simplicity, a single don't care bit is used to denote a series of don't care bits. Thus, prefix 1^* denotes 1^{****} in a 5-bit address space.

R_{ext}: *Extended prefix* of range R. R_{ext} is the longest prefix that covers R. If prefixes are illustrated in the binary trie, R_{ext} is the lowest common ancestor of addresses R_{min} and R_{max}. To be specific, assuming $R_{min} = a_{W-1} \dots a_1 a_0$ and $R_{max} = b_{W-1} \dots b_1 b_0$, $R_{ext} = c_{W-1} \dots c_k *$, where $c_i = a_i = b_i$ for $W - 1 \ge i \ge k$ and $a_{k-1} \ne b_{k-1}$. Consider range $R1$ in Fig. 1. $R1_{min} = a_5 \dots a_1 a_0 = 000010_2$ and $R1_{max} = b_5 \dots b_1 b_0 = 000110_2$. Since $a_i = b_i$ for $i = 5$ to 3 and $a_2 \ne b_2$, $\mathrm{R1_{ext}} = 000^{***}$.

P_{mid-} *and* P_{mid+}: *Middle addresses of a prefix of length less than* W. $P_{mid-} = d$ and $P_{mid+} = d + 1$, where $d = \lfloor (P_{min} + P_{max})/2 \rfloor$. For example, in Fig. 1, $P2_{mid-} = 15$ and $P2_{mid+} = 16$.

G_{ext}: The longest prefix that covers all ranges in range set G. In Fig. 1, $G_{ext} = ^{******}$.

Prefix (or prefix range): *A prefix R is a range that satisfies $R = R_{ext}$*. In Fig. 1, R4 is a prefix. We use $P\{0\}$ and $P\{1\}$ to represent the prefixes of the left and right children of a prefix P in the binary trie, respectively. For example, $P2\{1\}$ represents $P3$ in Fig. 1.

Disjoint, nested, and intersecting: Let $R1 = [L1, U1]$ and $R2 = [L2, U2]$ be two ranges.

1. $R1$ and $R2$ are *disjoint* if either $U1 < L2$ or $U2 < L1$. $R1$ is said to be smaller than $R2$ (denoted by $R1 < R2$) if $U1 < L2$.
2. $R1$ is *nested* by (*covered by, or contained in*) $R2$ if $L2 \le L1 \le U1 \le U2$ (denoted by $R1 \subset R2$). Also, $R1$ is said to be more *specific* than $R2$ if $R1 \subset R2$.
3. $R1$ and $R2$ are *intersecting* (*overlapping*) if they are neither disjoint nor nested. Specifically, $R1$ and $R2$ are *intersecting* if either $L1 < L2 \le U1 < U2$ or $L2 < L1 \le U2 < U1$. In this case, we also say that $R1$ intersects $R2$ or $R2$ intersects $R1$.

Successive and *contiguous* ranges: Two disjoint ranges are successive if no other range exists between them. Two successive ranges, $[L1, U1]$ and $[L2, U2]$, are contiguous if $L2 = U1 + 1$. For example, in Fig. 1, $R1$ and $R2$ are successive and $R5$ and $R6$ are contiguous.

Elementary intervals, valid and default intervals: The *default prefix*, *default range*, or *default rule* denoted by Rdf covers the entire address space. For example, in the one-dimensional packet classification, the default prefix is $Rdf = [0, 2^W - 1]$, which is usually the final result when no other prefix matches the destination address. The set of *elementary intervals*, constructed from the endpoints of a range set G, is $EI = \{E[i] \mid E[i] = [L[i], U[i]] \text{ for } i = 0 \text{ to } k - 1\}$, where k is the number of elementary intervals in EI. EI must satisfy the following four conditions:

1. $L[0] = 0$ and $U[k-1] = 2^W - 1$,
2. $U[i] = L[i+1] - 1$ for $i = 0$ to $k - 2$,
3. all the addresses in $E[i]$ are covered by the same subset of G, denoted by $G[i]$, and
4. $G[i] \ne G[i+1]$.

For example, EI computed from the six 6-bit ranges in Fig. 1 is $EI = \{[0,1], [2,6], [7,12], [13,17], [18,18], [19,27], [28,35], [36,39], [40,50], [51,54], [55,59], [60,63]\}$. The intervals that are covered by at least one original range in G are called *valid* intervals, e.g., [2, 6], [13, 17], etc. The other intervals are *default* intervals covered only by the default range [0, 63].

We only solve the one-dimensional packet classification (PC) problem in this paper. The multidimensional PC problem can be extended [1].

Definition 1. *Given a packet with destination address p and a set of n rules $G = \{Rule[i] = (R[i], C[i], A[i]) | R[i] = [L[i], U[i]]$ is a W-bit range, $C[i]$ is the priority, and $A[i]$ is the action of $Rule[i]$ for $i = 0 \ \dots \ n - 1\}$, the one-dimensional PC problem is to find the rule $Rule[i] \in G$ such that the range associated with rule $R[i]$ contains address p and $C[i]$ is the highest. As a result, action $A[i]$ can be taken for the packet.*

There exists a default rule (Rdf, Cdf, Adf) where $Rdf = [0, 2^W - 1]$, Cdf is set to be lower than the priority of any other rule, and Adf could be any action such as "Accept" or "Reject." For simplicity, $C[i]$ and $A[i]$ in $Rule[i]$ are ignored and, thus, $Rule[i]$ and $R[i]$ are used interchangeably when no confusion is incurred.

As stated earlier, storing ranges in TCAM can be solved by decomposing each individual range into multiple prefixes. Fig. 2 shows the direct range-to-prefix conversion algorithm *Direct_Convert* written in C-like pseudocode [5]. The inner *for* loop determines if R_{min} is a valid starting address of a prefix of size 2^i. For example, if $R_{min} = 4$, R_{min} is the staring address of the prefix 0001^{**} in the 6-bit address space. R_{max} is used to determine i since all the addresses R_{min} to $R_{min} + 2^i - 1$ of the prefix must be contained in range $[R_{min}, R_{max}]$. After the prefix of size 2^i is determined, its lower and upper addresses are stored in arrays $L[]$ and $U[]$, respectively. The outer *while* loop continues the process of finding the next prefix until R_{max} is reached. For example, when we run *Direct_convert* with

```
Direct_convert(R_min, R_max, L[], U[])
{
  int d = 0, i;
  while (R_min < R_max) {
    for (i = 0; i < W; i++)
      if ((R_min is not divisable by 2^(i+1) OR (R_min + 2^(i+1) − 1) > R_max) break;
    L[d] = R_min;   U[d] = R_min + 2^i − 1; d = d + 1;
    R_min = R_min + 2^i;
  }
}
```

Fig. 2. Range $[R_{min}, R_{max}]$ is cut into d prefixes at $s = d - 1$ cutting points. Arrays $L[0 \dots d-1]$ and $U[0 \dots d-1]$ store the lower and upper addresses of the converted prefixes. We call $U[0] \dots U[s-1]$ the lower cutting addresses of the conversion.

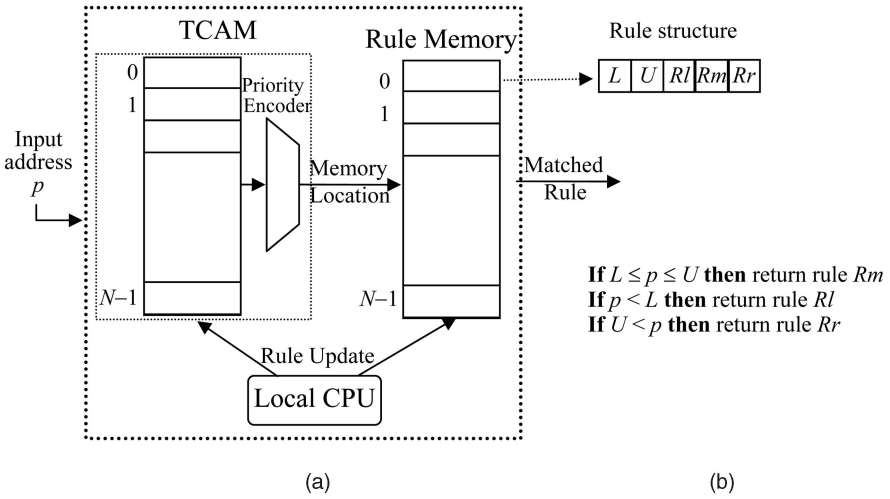

Fig. 3. The proposed 2-level TCAM architecture for the 1D PC problem.

range R_3 in Fig. 1, the inner for loop determines the first prefix [19, 19] because $R_{min} = 19$ is not divisible by 2^1. The next two prefixes, [20, 23] and [24, 27], can be obtained because addresses 20 and 24 are divisible by 2^2. In the worst case, the range $[1, 2^W - 2]$ is split into $2W - 2$ prefixes. For a set of m ranges, the worst-case number of prefixes generated by algorithm *Direct_convert* is $O(mW)$.

Example 1. Assume there are m ranges selected from the W-bit range set

$$D = \{R[i] | R[i] = [L[i], U[i]], \text{where } L[i] = t \times 2^V + 1 \text{ and } U[i] = (t+1) \times 2^V - 2 \text{ for } t = 0 \ldots 2^{W-V} - 1\}.$$

Each range in D can be decomposed into $2 \times (V - 1)$ prefixes using the algorithm in Fig. 2. Since these m ranges are disjoint, $(2V - 2) \times m$ prefixes are needed. Assume $V = O(W)$. Thus, $O(Wm)$ prefix entries are needed to store these m ranges in TCAM.

Before going into the details of the proposed algorithms, we first demonstrate why the proposed algorithm can solve the special case described in Example 1. By using the concept of extended prefixes, range R[i] in Example 1 can be stored as its extended prefix $P[i] = R[i]_{ext}$. Since all the extended prefixes $P[1]$ to $P[m]$ are disjoint, only m prefixes are needed. As we shall explain later, we use an auxiliary data structure, called *rule structure*, for each extended prefix $P[i]$ to store the two endpoints $L[i]$ and $U[i]$. Thus, if the input address matches $P[i]$ and the input address is between $L[i]$ and $U[i]$ (inclusively), then the matched range is $R[i]$. Otherwise, the matched rule is the default range. Notice that, in general, some of the extended prefixes may be nested by other ranges. Thus, the one-dimensional PC problem becomes complicated and will be solved completely in the latter part of this paper.

3 Proposed Method

In this section, we shall propose a 2-level TCAM architecture, as shown in Fig. 3, to solve the one-dimensional PC problem with range fields. The first level of the proposed architecture is the traditional TCAM and the second level is the rule structure memory, which can be implemented with fast memory such as SRAM. As stated in Definition 1, each rule is associated with a range. By using one of the two proposed schemes, each range can be converted to one or two prefixes which can then be stored in TCAM. In addition, the rule structures in SRAM are used to store other range related information. When a packet containing the destination address p arrives, the following operations are performed: Address p is compared with all TCAM entries in parallel. The rule memory index of the matched entry with the highest priority is returned. Then, the information stored in the rule structure of the matched rule memory is compared with p. The final matched rule is selected from the three possible rules, namely, Rl, Rm, and Rr, as shown in Fig. 3b. As we can see, the rule structure stores two addresses, L and U, which can only distinguish three segments in the associated prefix. Therefore, at most three ranges can be associated with the three segments. The local CPU is responsible for updating the TCAM and rule structure memory.

The two proposed range-to-prefix conversion schemes are based on the concepts of disjoint ranges and contiguous ranges. Basically, for each range R, if the extended prefix R_{ext} of R can be associated with a *correct rule structure*, then it is directly stored in TCAM. Otherwise, R is split into two prefixes which are associated with their correct rule structures. By "correct rule structure," we mean that the search process can find the correct classification result based on the proposed 2-level TCAM architecture.

Example 2. Consider the ranges in Fig. 1. The extended prefixes of all ranges except R_3 can be associated with the correct rule structures. For example, $P2 = R2_{ext}$ can be associated with the rule structure (13, 17, Rdf, $R2$, Rdf). The addresses of $P3 = R3_{ext}$ are covered by four address segments, namely, [16, 17], [18, 18], [19, 27], and [28, 31]. Thus, no correct rule structure can be assigned to $P3$. Range $R3$ must be split into two ranges, $R3e = [19, 23]$ and $\text{R3f} = [24, 27]$. Their extended prefixes $P3e = 010^{***}$ and $P3f = 0110^{**}$ can be associated with the correct rule structures, (18, 18, $R2$, Rdf, $R3$) and (24, 27, Rdf, $R3$, Rdf), respectively. If the input address is 17, the

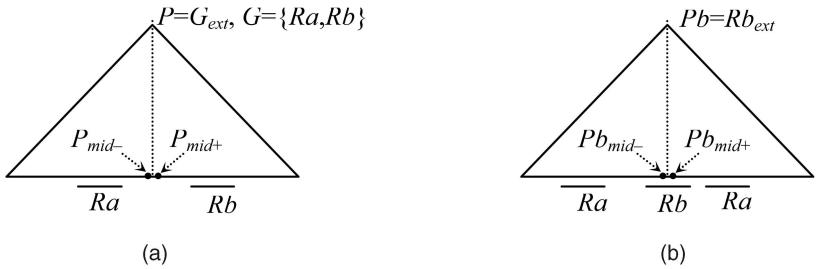

Fig. 4. Pictorial examples for the properties of prefixes extended from disjoint ranges.

longest prefix match is $R3e$. Thus, the matched range is $R2$ after checking the rule structure of $R3e$.

3.1 Disjoint Ranges

In this section, a systematic mechanism based on the range split algorithm is developed to store the disjoint ranges in the proposed 2-level TCAM architecture. For each disjoint range, we use the following properties to construct one or two prefixes (TCAM entries) and the associated rule structures:

Properties of the prefixes extended from disjoint ranges: For two disjoint ranges Ra and Rb, assuming $Pa = Ra_{ext}$, $Pb = Rb_{ext}$, $G = \{Ra, Rb\}$, $P = G_{ext}$, we have

1. $Pa \neq Pb$.
2. *Pa and Pb are disjoint if $Ra_{max} \leq P_{mid-}$ and $P_{mid+} \leq Rb_{min}$.*
3. *Pb covers Pa if Pb covers Ra.*

We omit the complete proofs and only illustrate these properties for nonsingleton ranges by using the examples shown in Fig. 4. By definition, if $Pa_{ext} = Pb_{ext}$, both Ra and Rb must contain Pa_{mid-} and Pa_{mid+}. This is a contradiction because Ra and Rb are disjoint. Thus, we have the first property, $Pa \neq Pb$. We know that $P = G_{ext}$ covers both Ra and Rb. If $Ra_{max} \leq P_{mid-}$ and $P_{mid+} \leq Rb_{min}$, as shown in Fig. 4a, Ra must be contained in $P\{0\}$, which is the prefix of the left child of P in the binary trie. Similarly, Rb is contained in $P\{1\}$. Since $P\{0\}$ and $P\{1\}$ are disjoint, we have the second property that Pa and Pb are disjoint. The condition that Pb covers Ra implies $G_{ext} = Rb_{ext}$. Thus, the third property holds. Fig. 4b shows that Pb covers Ra. We can see that Rb must contain addresses Pb_{mid-} and Pb_{mid+} that are contained in $P\{0\}$ and $P\{1\}$, respectively. Therefore, Pb must cover Pa.

Notice that the highest priority TCAM entry is the longest prefix that matches the input address p. According to the third property described above, if address p matches $Pa = Ra_{ext}$, then the matched range can be found by inspecting the rule structure associated with Pa because Pa is longer than Pb. However, if the longest prefix returned by TCAM is Pb, then the input address p must not match Pa. Therefore, when we consider building the rule structure for range Rb, all the ranges (such as Ra) covered by Pb can be ignored. As a result, we are able to associate Pb with a correct rule structure in a recursive manner and ensure that the rule selection process returns correct results based on the proposed 2-level TCAM architecture.

Recall that we propose using the rule structure with two endpoints, as shown in Fig. 3b, to select the correct ranges. Two endpoints can only distinguish at most three segments in a line. Four *fundamental cases*, as shown in Fig. 5, illustrate four different situations that can be solved by the proposed rule structure without a range split. Case 1 illustrates the simplest case in which no other range intersects $P = R_{ext}$. Obviously, the rule structure of P can be set to (R_{min}, R_{max}, Rdf, R, Rdf), where Rdf denotes the default range. Cases 2 and 3 illustrate that only one range $R0$ intersects prefix $P = R_{ext}$. In case 2, P is associated with rule structure ($R0_{max} + 1$, $R_{min} - 1$, $R0$, Rdf, R). However, in case 3, P is associated with the rule structure (R_{min}, R_{max}, $R0$, R, Rdf). Notice that the mirrored versions of cases 2 and 3 are constructed similarly. In case 4, prefix $P = R_{ext}$ is completely covered by three ranges, $R0$, R, and $R1$, and is associated with the rule structure (R_{min}, R_{max}, $R0$, R, $R1$).

The proposed rule structure does not work for other cases, such as cases 5 and 6 in Fig. 5, which require range splits to fit to the fundamental cases. The following lemma determines whether or not a range needs to be split and how to split the range for the proposed 2-level TCAM architecture.

Lemma 1. *The range R will be split by a range Rc iff one of the following conditions is true, assuming Rc and R are disjoint and $Pc = Rc_{ext}$ covers $P = R_{ext}$:*

1. *$Rc_{min} < P_{min} \leq Rc_{max}$ and $R_{min} \neq Rc_{max} + 1$ and $R_{max} \neq P_{max}$.*
2. *$Rc_{min} \leq P_{max} < Rc_{max}$ and $R_{max} \neq Rc_{min} - 1$ and $R_{min} \neq P_{min}$.*

When range R is split, it is split into two ranges, $Re = [R_{min}, P_{mid-}]$ and $Rf = [P_{mid+}, R_{max}]$.

We omit the proof and only illustrate the idea of this lemma. The above two conditions are shown as cases 5 and 6 in Fig. 5. Take case 5 in Fig. 5e as an example. The addresses covered by $P = R_{ext}$ are divided into four segments. Hence, the proposed rule structure with two endpoints fails to represent the rule matching conditions for prefix P. Therefore, we split R into two ranges, Re and Rf. Since $Re_{max} = (Re_{ext})_{max}$ and $Rf_{min} = (Rf_{ext})_{min}$, no further split is needed for Re and Rf based on the same lemma. Re_{ext} and Rf_{ext} will be associated with case 2 rule structure ($Rc_{max} + 1$, $R_{min} - 1$, Rc, Rdf, R) and case 1 rule structure (P_{mid+}, R_{max}, Rdf, R, Rdf), respectively.

As shown in Example 2, $R3$ is split into two ranges, $R3e = [19, 23]$ and $R3f = [24, 27]$, because $P3$ satisfies case 5. After the split, $P3f = R3f_{ext}$ is a prefix range that fits case 1.

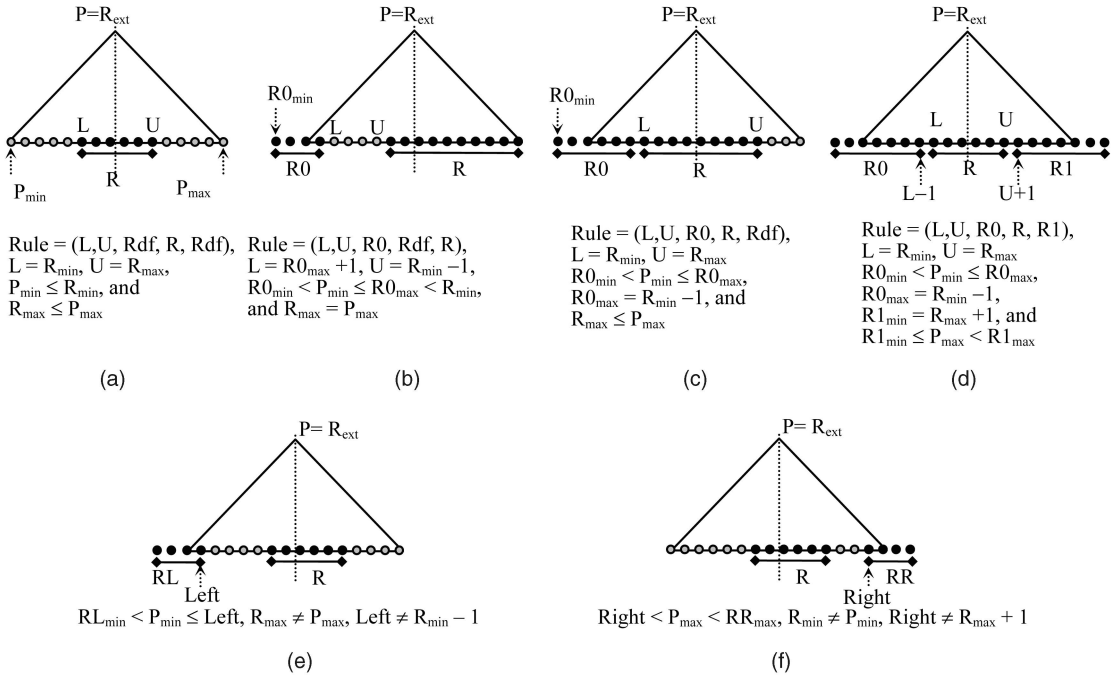

Fig. 5. Four fundamental cases for selecting correct rule structures and two cases that do not fit these fundamental cases. (a) Case 1. (b) Case 2. (c) Case 3. (d) Case 4. (e) Case 5. (f) Case 6.

Therefore, $P3f$ is associated with (24, 27, Rdf, $R3$, Rdf). $P3e = R3e_{ext}$ fits case 2 and is associated with (18,18,$R2$,Rdf,$R3$). The other ranges do not need to be split. As a result, the proposed scheme generates seven prefixes from the range set G. If the ranges in G are converted individually, 14 prefixes will be generated.

3.1.1 Range Split (SPLIT Algorithm)

To demonstrate how to split the ranges, we store the disjoint ranges in the binary trie. In Fig. 1, we have seen the binary trie built from the extended prefixes of the six ranges in G. Based on the results in Lemma 1, we propose the algorithm $SPLIT(Root, NL, NR)$ in Fig. 6 that splits the disjoint ranges stored in the binary trie recursively. The first parameter, Root, is set to the pointer to the root of the binary trie initially. Parameters NL and NR are the pointers pointing to the predecessor and successor of the subtrie rooted at Root. For example, the predecessor and successor of node $P3$ in Fig. 1 are node $P2$ and the root of the trie, respectively. Initially, NL and NR are set to NULL. The nodes in the binary trie are classified into two types, i.e., *prefix nodes* and *nonprefix nodes*. Prefix nodes store the range associated with it. For example, nodes marked with bold circles in Fig. 1 are prefix nodes. If the current *Root* is a prefix node, we use *Lemma 1* (lines 10-11) to check if it should be split into two ranges. If the range R associated with *Root* needs to be split, it is split into $Re = [R_{min}, P_{mid-}]$ and $Rf = [P_{mid+}, R_{max}]$. Two new prefix nodes generated from Re and Rf are inserted in the binary trie for further processing. The *Root* node is then reset as a nonprefix node. If the range associated with *Root* need not be split, then *Root* is associated with the correct rule structure. Since the range associated with *Root* must satisfy one of the four fundamental cases, the correct rule structure can be assigned accordingly, as shown in Fig. 5. Finally, two recursive calls (lines 19-20) are made into the two children of *Root*. Since each range is only split at most once and there is at least one range that is not split, we have the following result:

Lemma 2. *For a set of m disjoint ranges, the total number of ranges generated by the SPLIT algorithm is at most $2m - 1$.*

3.1.2 Range Update

Now, we consider how to insert and delete a range. We first propose the insertion and deletion algorithms that incrementally update the related prefixes and their rule structures without restarting the *SPLIT* algorithm again. The number of prefixes after inserting or deleting a range is

```
00 SPLIT(Root, NL, NR)
{
01   If (Root ≠ NULL) {
02     If (Root is a prefix node) {
03       Left = –∞ and Right = ∞;
04       If ( NL is a prefix node)
05           Left = the upper address of the range associated with node NL;
06       If ( NR is a prefix node)
07           Right = the lower address of the range associated with node NR;
08       P = the prefix associated with the prefix node Root;
09       R = the range associated with the prefix node Root;
10       If (   (P_min≤Left and R_min≠Left+1 and P_max ≠R_max)   |
11           (Right≤P_max and R_max≠Right–1 and P_min≠R_min)   ){
12           Split R into Re = [R_min, P_mid–] and Rf = [P_mid+, R_max];
13           Insert Re and Rf in the binary trie;
14           Reset Root as a non-prefix node;
15       }
16       If (Root is a prefix node)
17           Associate Root with the correct rule structure, using NL and NR;
18     }
19     SPLIT(Root.LeftChild,  NL,  Root);
20     SPLIT(Root.RightChild, Root,  NR);
21   }
}
```

Fig. 6. Range split procedure for a set of disjoint ranges.

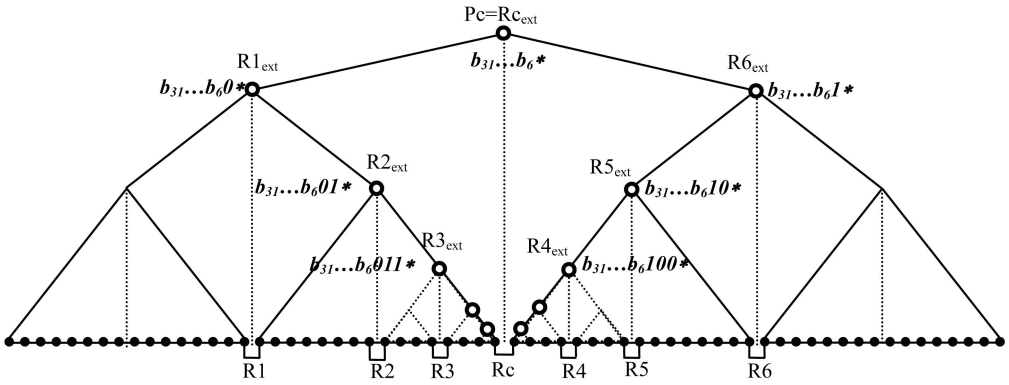

Fig. 7. The ranges that may be split by a 2-address range Rc.

kept below $2m$, where m is the number of the original ranges. In the second part of this section, we consider how to reduce the number of TCAM memory update operations and propose the optimized insertion algorithm. Although the constraint that one original range is only split once is relaxed, the number of prefixes is still kept below $2m$. With the optimized insertion algorithm, the TCAM update performance is much improved.

Insertion. When range Rs is to be inserted, the following steps are required for the proposed 2-level TCAM architecture: 1) Determine if Rs needs to be split into two ranges, Re and Rf, 2) search the ranges that need to be split by Rs or Re and Rf, and 3) change the rule structures of other ranges that depend on Rs or Re and Rf.

In the first step, we look for the ranges RL and RR that intersect Rs_{ext}. Recall that the ranges are stored in the binary trie. Thus, we first insert the prefix Rs_{ext} in the binary trie. Then, RL and RR can be obtained by traversing the trie from the root to the node Rs_{ext}. If one of the two conditions in Lemma 1 is satisfied, Rs is split into Re and Rf. Since Re and Rf will not be split again, they can be associated with the correct rule structures according to the four fundamental cases.

In the second step, we determined what other existing ranges need to be split because of the insertion of Rs, or Re and Rf if Rs is split. Subsequently, we only show the necessary actions when inserting Rs since the necessary actions are the same when inserting Re and Rf. We look for the candidate range R for splitting such that R is covered by Rs_{ext} and Rs intersects R_{ext}. The process of finding the candidate ranges R will be presented later in this section. Briefly, we traverse the binary trie from Rs_{ext} to Rs_{max} and from Rs_{ext} to Rs_{min}. If any traversed node is a prefix node and its associated range R satisfies one of the two conditions in Lemma 1, R must be split into two ranges.

Notice that the ranges split by Rs may be the ones causing the other ranges to be split earlier. Therefore, when these kinds of ranges are split, those ranges split by them can be merged back together without affecting the correctness of the proposed algorithms. However, there may be other ranges that will be affected by these merged ranges. This chain effect may have a negative impact on TCAM performance. Therefore, in this paper, we do not perform this kind of merge operations.

The last step of the insertion algorithm looks for the ranges, the associated rule structures of which depend on Rs. The process of determining which existing ranges depend on Rs is the same as Step 2. After we know which existing ranges depend on Rs, we can modify their rule structures according to the four fundamental cases in Fig. 5.

To facilitate the insertion process, we shall develop a mechanism to calculate the possible candidate ranges that may be split by Rs. Before we show the results, we explain our idea using the following example:

Example 3. Fig. 7 shows a range set $G = \{R1, R2, R3, R4, R5, R6\}$ and the range Rc to be inserted in a 32-bit address space. The extended prefix $Pc = Rc_{ext}$ is of length 26. Based on Lemma 1, every range in G should be split by Rc. However, if we apply the *SPLIT* algorithm to the range set G before Rc is inserted, ranges $R2$ and $R5$ are already split by $R1$ and $R6$, respectively.

To generalize the above example, we have the following result: Let prefix Pc of length $h (0 \le h < W-3)$ be $b_{W-1} \ldots b_{W-h} *$ and range Rc be $[Pc_{mid-}, Pc_{mid+}]$. *The possible candidate ranges that may be split by Rc are the ones whose extended prefixes are* $b_{W-1} \ldots b_{W-h} 1\{0\}^k *$ *and* $b_{W-1} \ldots b_{W-h} 0\{1\}^k *$ *for* $k = 0$ *to* $W-h-4$.

Let R be a candidate range and consider $b_{W-1} \ldots b_{W-h} 1\{0\}^k *$ only. It is worth showing why $P = R_{ext}$ cannot be $b_{W-1} \ldots b_{W-h} 1\{0\}^k *$ for $k = W-h-3$ to $W-h-1$. Consider $P = b_{W-1} \ldots b_{W-h} 1\{0\}^{W-h-1}$ or $b_{W-1} \ldots b_{W-h} 1\{0\}^{W-h-2} *$. P must cover the address Pc_{mid+} and make P not disjoint from Rc. Therefore, range R, whose extended prefix is $b_{W-1} \ldots b_{W-h} 1\{0\}^{W-h-1}$ or $b_{W-1} \ldots b_{W-h} 1\{0\}^{W-h-2} *$, does not exist. Now, consider $P = b_{W-1} \ldots b_{W-h} 1\{0\}^{W-h-3} **$. The only condition that makes P disjoint from Rc is $R_{min} = Pc_{mid+} + 1$. This condition does not satisfy the second term of the first condition in Lemma 1. Therefore, R will not be split by Rc. Prefix Rc_{ext} must be shorter than $W-3$ to have existing ranges split by Rc.

To calculate the maximum number $N(Rc)$ of ranges that may be split by $Rc = [Pc_{mid-}, Pc_{mid+}]$, we have to consider the existing ranges that may already be split by other exiting range before Rc is inserted, as shown in Example 3. Assuming prefix Rc_{ext} is of length h, $N(Rc)$ is $2 \times \lceil (W-h-3)/2 \rceil$. $N(Rc)$ is at most $W-2$ if we assume $h = 0$ and W is an even number.

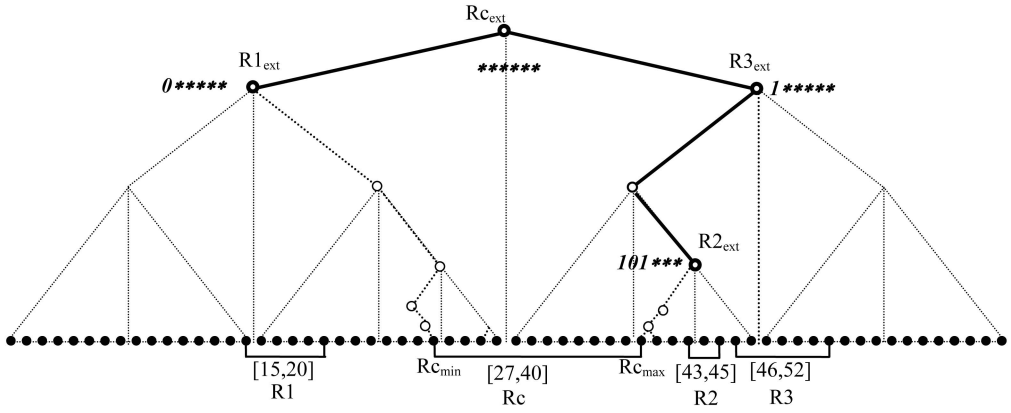

Fig. 8. The ranges that may be split by an arbitrary range Rc.

Now, we consider Rc to be an arbitrary range, instead of a 2-address range. Assume Rc_{ext} is of length h. We have to consider many 2-address ranges, like $[Pc_{mid-}, Pc_{mid+}]$, contained in Rc which intersect the extended prefixes of other ranges. These 2-address ranges are just $[U[i], U[i]+1]$ for $i = 0$ to $s-1$, where $U[i]$ are lower cutting addresses of range Rc from the direct range-to-prefix conversion as shown in Fig. 2. The number $N(Rc)$ of ranges that may be split by the arbitrary range Rc is at most $2 \times \lceil (W-h-3)/2 \rceil$. $N(Rc)$ is at most $W-2$ when $h = 0$ and W is an even number. Based on the above results, the following conclusion is made:

The candidate ranges that may be split by Rc are the ones whose extended prefixes are associated with the prefix nodes on the two paths from node Pc to node Rc_{min} and from Pc to node Rc_{max} in the binary trie. Also, these extended prefixes are shorter than $W-3$.

Example 4. Fig. 8 shows a range set $G = \{R1, R2, R3\}$ and the range Rc to be inserted in a 6-bit address space. The binary trie built from G is shown in bold lines. The prefix nodes in the binary trie are marked as bold circles. Based on Lemma 1, every range in G should be split by Rc. The extended prefixes of the ranges in G are just the prefixes associated with the prefix nodes on the two paths from node Rc_{ext} to node Rc_{min} and from node Rc_{ext} to node Rc_{max}.

Deletion. When a range $Rdel$ is to be deleted, we have to 1) delete the range $Rdel$ or the two contiguous ranges split from $Rdel$ and 2) modify the rule structures of the extended prefixes that depend on $Rdel$.

In the first step, if an existing $Rdel$ is found, it is deleted directly. Otherwise, we look for two contiguous ranges, $Re = [Rdel_{min}, Pdel_{mid-}]$ and $Rf = [Pdel_{mid+}, Rdel_{max}]$, where $Pdel = Rdel_{ext}$. If both Re and Rf exist, they are deleted directly.

Notice that we can find the pairs of contiguous ranges that were split by $Rdel$ or by Re and Rf split from $Rdel$. All of these pairs of contiguous ranges may be merged back to their original ranges after $Rdel$ is deleted. However, merging these ranges may cause the other ranges to be split in order to maintain the correct rule structures for the proposed 2-level TCAM architecture. Merging and splitting ranges incur very costly TCAM update operations. Therefore, we do not propose to merge these contiguous ranges split by $Rdel$ back to their original ranges since the constraint that a range is split at most once is not violated.

The second step is to find the ranges, the extended prefixes of which depend on $Rdel$. Then, the rule structures of these ranges are modified to remove the dependency on $Rdel$. Let R be a range that depends on $Rdel$. R may or may not be split from $Rdel$. $Rdel$ must intersect R_{ext}. Therefore, R_{ext} must be associated with the rule structure of case 2, 3, or 4 before $Rdel$ is deleted. When $Rdel$ is deleted, R_{ext} should be changed to case 3 if R_{ext} was associated with case 4 rule structure or R_{ext} should be changed to case 1 if R_{ext} was associated with case 2 or 3 rule structure. We formulate the above result for deletion as follows: *When a range is deleted from a set of disjoint range G, at most two prefixes are deleted from TCAM.*

Optimization. Reducing the number of TCAM memory updates is important [8] because TCAM updates slow down the TCAM search performance. Also, updating TCAM is much slower than updating the rule structures stored in the rule memory of the proposed 2-level TCAM architecture. Therefore, we propose the optimized insertion algorithm to reduce the number of TCAM updates while keeping the number of prefixes below $2m$ for m disjoint ranges.

When one range is inserted, one or two (if the range is split) prefixes have to be inserted in TCAM. In addition, we have to determine what other existing ranges have to be split by the range to be inserted. A split operation on an existing range R results in one prefix, (R_{ext}), being deleted and two prefixes, Re_{ext} and Rf_{ext}, being inserted, where Re and Rf are the ranges split from R. Based on the properties of extended prefixes in Section 3.1, prefixes R_{ext}, Re_{ext}, and Rf_{ext} must be different. Therefore, when an existing range is split, it incurs three TCAM update operations. We know that the maximum number of ranges that may be split by the newly inserted range is on the order of $O(W)$. As a result, splitting existing ranges by the newly inserted range has a very negative impact on TCAM performance. One way to avoid splitting existing ranges is to split the range the newly inserted range itself between addresses $U[i]$ and $U[i]+1$, where $U[i]$ for $i = 0$ to $s-1$ are the lower cutting addresses of the direct range-to-prefix conversion scheme in Fig. 2. Therefore, we propose the following optimized insertion algorithm:

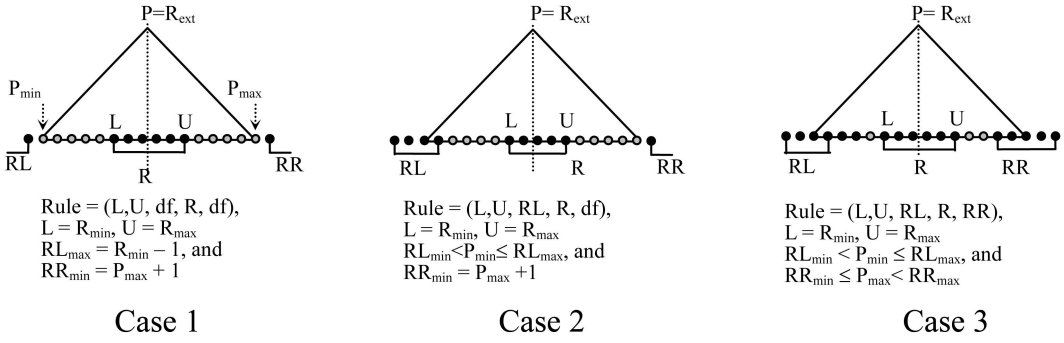

Fig. 9. Three fundamental cases for contiguous ranges.

Optimized Insertion. When inserting range Rs, three steps are needed. First, we determine if Rs needs to be split into two ranges, Re and Rf, as in the nonoptimized insertion. Second, we search the existing ranges that should be split by the 2-address range $[U[i], U[i]+1]$ for $i = 0$ to $s-1$ contained in Rs (or in Re and Rf if Rs is split). If there is any existing range that will be split by $[U[i], U[i]+1]$, we split Rs (Re and Rf) between $U[i]$ and $U[i]+1$. Third, we set the ranges split from Rs with the correct rule structures.

Example 5. As shown in Fig. 8, the optimized insertion does not split all ranges in G when range Rc is inserted. Instead, range Rc itself is split into three ranges [27, 31], [32, 39], and [40, 40]. Splitting Rc between addresses 31 and 32 prevents ranges $R1$ and $R3$ from being split. Similarly, splitting Rc between addresses 39 and 40 prevents range $R2$ from being split. As a result, only three prefixes extended from [27, 31], [32, 39], and [40, 40] need to be inserted in TCAM instead of deleting three prefixes and inserting six prefixes for unoptimized insertion.

It is easy to see that the optimized insertion algorithm generates at most $O(W)$ prefixes when a disjoint range is inserted. Based on the optimized insertion, we have to modify the deletion algorithm. If an original range is to be deleted, all ranges split from it must be deleted. Therefore, at most $O(W)$ ranges split from the original range need to be deleted in order to complete the process of the deletion. Thus, we have the following result:

Lemma 3. *At most $2m-1$ prefixes are needed in the proposed 2-level TCAM after some insertions and deletions, where m is the number of original disjoint ranges.*

Proof. We have shown that the SPLIT algorithm generates at most $2m-1$ ranges in a set of m disjoint ranges because each disjoint range can be split into two ranges only once. Although the optimized insertion algorithm allows a range to be split more than once, it will not generate more ranges because each cut on the newly inserted range is traded for not splitting one or more other ranges. Thus, the lemma follows. □

3.2 Contiguous Ranges (Elementary Intervals)

In this section, we propose a different approach that converts the set of ranges into contiguous disjoint ranges (also called elementary intervals). This approach is suitable for both disjoint and overlapping ranges. All the default and valid elementary intervals are used when constructing the extended prefixes. Thus, the whole address space is covered by all extended prefixes. We shall show that no range split is needed when associating all extended prefixes with the correct rule structures. Since each elementary interval is also a disjoint range, we use the elementary interval and disjoint range interchangeably in this section.

As defined, all elementary intervals are contiguous. The addresses contained in R_{ext} must be completely covered by R, by the elementary intervals contained in R_{ext}, and by the two elementary intervals (say, RL and RR) that intersect R_{ext} if they exist. Thus, when we consider how to assign a correct rule structure to R_{ext}, we only need to consider R, RL, and RR. The addresses in R_{ext} that should match R_{ext} will match the longest prefixes extended from one of the elementary intervals contained in R_{ext}. Fig. 9 shows three fundamental cases that can be used to assign the correct rule structure to R_{ext}. The extended prefixes of all elementary intervals must satisfy one of these three fundamental cases without any range split.

In Fig. 10, we show the algorithm $Build\_EI(Root, NL, NR)$ that associates the extended prefixes of all elementary intervals with the correct rule structures. As in algorithm *SPLIT*, all elementary intervals are first stored in the binary trie by using their extended prefixes. Only the nodes corresponding to these extended prefixes are marked as prefix nodes in the binary trie. The rule structures associated with the extended prefixes are obtained based on the three fundamental cases in Fig. 9. For example, when calling $Build\_EI(Root, NL, NR)$ to assign the rule structure to prefix $P5$ extended from $E9 = [51, 54]$ in Fig. 1, Root, NL, and NR are set to nodes 110^{***} ($P5$), 1^{*****}, and 11^{****} ($P6$), respectively. As a result, $P5$ satisfies the fundamental case 2 and can thus be associated with (51, 54, Rdf, $R5$, $R6$). Based on the above result, we conclude the following: *Given m contiguous ranges, algorithm $Build\_EI$ generates at most m prefixes for the proposed 2-level TCAM architecture.*

Now, we consider the range update operations. We assume that the newly inserted range Rs is disjoint from all valid intervals. Thus, Rs must be nested by a default interval denoted by Rd. Also, the range $Rdel$ to be deleted is one of the existing valid elementary intervals.

Insertion. In general, Rs and Rd can be converted into three elementary intervals, i.e., Rl, Rs, and Rr. If $Rs_{min} = Rd_{min}$, Rl is empty. Similarly, if $Rs_{max} = Rd_{max}$,

```
Build_EI(Root, NL, NR) // EI stands for Elementary Interval
{
01   If (Root ≠ NULL) {
02      If (Root is a prefix node)
03         Assign Root with the correct rule structure, using the ranges associated
           nodes NL and NR, based on the three fundamental cases in Figure 9;
04      Build_EI(Root.LeftChild, NL, Root);
05      Build_EI(Root.RightChild, Root, NR);
      }
}
```

Fig. 10. Construction algorithm for a set of contiguous disjoint ranges.

Rr is empty. Rd_{ext} may be the same as Rl_{ext}, Rs_{ext}, or Rr_{ext}. Let $G = \{Rl, Rs, Rr\}$. If there exists a range R in G such that $R_{ext} = Rd_{ext}$, R must contain both of the addresses, Pd_{mid-} and Pd_{mid+}, where $Pd = Rd_{ext}$. Therefore, at most two TCAM entries with the correct rules structures for Rl_{ext} and Rr_{ext} are created and inserted in TCAM. Also, the rule structure of Rd_{ext} needs to be changed to that of R_{ext}. On the contrary, when either $R_{max} = Pd_{mid-}$ or $R_{min} = Pd_{mid+}$, it is not possible that $R_{ext} = Rd_{ext}$, where $R \in G$. Thus, inserting Rs involves at most four TCAM update operations (one deletion and three insertions).

After determining which elementary interval needs to be deleted or inserted, the process of changing the rule structure associated with the prefix extended from the elementary interval becomes easy. Assume we have to delete the extended prefix Rd_{ext} and insert three extended prefixes, Rl_{ext}, Rs_{ext}, and Rr_{ext}. Since Rd is a default interval, we only need to reset the node Rd_{ext} in the binary trie as the nonprefix node. No rule structure of any prefix needs to be modified. To insert, Rl_{ext}, Rs_{ext}, and Rr_{ext}, we perform the following actions: We only show the process of inserting Rs_{ext} because it is similar for Rl_{ext} and Rr_{ext}. We first traverse the binary trie from root to node Rs_{ext} in order to obtain the predecessor NL and successor NR of the subtrie rooted at Rs_{ext}. Then, we set node Rs_{ext} to be the prefix node and use NL and NR to set the rule structure of Rs_{ext} according to one of the three fundamental cases. Next, we traverse the two paths of the binary trie from Rs_{ext} to Rs_{max} and from Rs_{ext} to Rs_{min} to find any prefix node P and change the rule structure associated with P according to the three fundamental cases as well.

Deletion. The process of deleting an existing range is similar to the insertion process described above. The only difference is that two or three adjacent elementary intervals may be associated with the default range simultaneously after deletion and thus need to be merged into one. Obviously, if two elementary intervals are merged, at most three TCAM update operations (one insertion and two deletions) are needed. If three elementary intervals are merged, at most four TCAM update operations (three deletions and one insertion) are needed.

Optimized deletion. As stated earlier, deleting an existing disjoint range involves many other insertions and deletions which are costly TCAM operations. Here, we propose an optimized deletion algorithm to reduce the overhead of updating TCAM. When deleting an existing disjoint range $Rdel$, we only update the rule structure of the TCAM entry corresponding to $Rdel$ instead of really deleting $Rdel_{ext}$ from TCAM and updating the rule structures of other TCAM entries, depending on $Rdel$. Assume that the rule structure of $Rdel$ is (L, U, Ra, $Rdel$, Rb) before deletion. Since the value of $Rdel$ in (L, U, Ra, $Rdel$, Rb) is in fact the ID of the range $Rdel$, we only need to change the related routing information of $Rdel$ to that of the default range. Thus, no TCAM memory movement is needed. The only overhead is to maintain an unnecessary TCAM entry for $Rdel_{ext}$ along with the associated rule structure.

This approach may violate the constraint that the number of elementary intervals must not exceed $2m$ for a set of m disjoint ranges. What we do to remedy this problem is to perform the merging operations if the constraint is violated, or periodically, or when the TCAM load is light.

3.3 Arbitrary Ranges

In this section, we consider the arbitrary ranges. We use a 2-step approach. The first step is to convert the arbitrary ranges into disjoint ranges or contiguous ranges. The second step is to apply the proposed *SPLIT* algorithm for disjoint ranges or the *Build_EI* algorithm for contiguous ranges. For disjoint ranges, we only consider valid intervals. For n arbitrary ranges, the number of valid intervals is at most $2n - 1$. The proposed *SPLIT* algorithm, along with its insertion and deletion algorithms, can then be applied to store these disjoint ranges in the proposed 2-level TCAM architecture. Thus, we have the following result:

Theorem 1. *For a set of n original arbitrary ranges, at most $4 \times n - 3$ prefixes are needed in the proposed 2-level TCAM based on disjoint ranges.*

Proof. A set of n original ranges can be converted into at most $m = 2n - 1$ valid intervals. Each of these intervals is covered by at least one original range. Based on the proposed insertion and deletion algorithms in Section 3.1, the total number of prefixes after some insertions and deletions will be at most $2m - 1$. Thus, the theorem follows. □

In Section 3.2, we consider the elementary intervals that are valid intervals or default intervals. The proposed *Build_EI* algorithm, along with its insertion and deletion algorithms, can be applied. Thus, we have the following result:

Theorem 2. *For a set of n original arbitrary ranges, at most $2n + 1$ prefixes are needed in the proposed 2-level TCAM based on the elementary intervals.*

Proof. Based on the proposed insertion and deletion algorithms for contiguous ranges in Section 3.2, the theorem is true because there are at most $2n+1$ elementary intervals for a set of n arbitrary ranges. □

Now, we consider how to perform the update operations. The update operations for the elementary intervals have been solved fairly well by the data structures, such as the segment tree [9]. The update complexity of the segment tree is $O(log\, k)$ for a tree of k intervals. The lookup operations can also have the complexity of $O(log\, k)$. If we choose the segment tree to organize the elementary intervals in the proposed 2-level TCAM architecture, the $O(log\, k)$ lookup complexity can be easily achieved. Whether disjoint ranges or contiguous ranges are considered, the same segment tree data structure can be used.

In general, the operations for a lookup in the proposed 2-level TCAM architecture work as follows: One TCAM memory read is needed to locate the matched prefix and its associated rule structure. A simple rule selection process is then performed to find the targeted interval. After finding the targeted interval, $O(log\, k)$ memory reads from the segment tree are needed to get the original range with the highest priority that covers the targeted interval. Therefore, the total complexity of the lookups is $O(log\, k)$ in the proposed 2-level TCAM architecture. However, the complexity of lookup operations in the TCAM-based architecture must be $O(1)$, a constant time. In other words, using the segment tree to solve the problem of the elementary intervals may compromise the constant time performance of TCAM memories. Therefore, to achieve constant time for the lookup operations, we decide to use a linear array instead of a segment tree to maintain what the original ranges covering an elementary interval are. The worst-case complexity of the update process will become $O(k)$ for a set of k intervals. However, the lookup complexity of $O(1)$ can be achieved. How to balance the times taken for the lookup and update processes will be left as future research.

Since the update operations on SRAM are much faster than those in TCAM, we focus on the update operations in TCAM when an original range Rs is inserted or an original range $Rdel$ is deleted. For simplicity, we only describe the update algorithms for elementary intervals because similar algorithms can be developed for disjoint ranges.

Insertion.

1. The first step locates any existing elementary interval (EI) that is completely covered by Rs. We assume there is a *range list* associated with each interval that records the original ranges covering the interval. Let Rh be the highest priority range associated with EI. If EI is the valid interval and the priority of Rs is higher than Rh or EI is the default interval, we set Rh to Rs. If EI is the valid interval and the priority of Rs is not higher than Rh, Rs is put in the range list of EI. Since the extended prefix EI_{ext} already exists in TCAM, no TCAM insertion is needed.
2. The second step locates any existing elementary interval EI that intersects Rs. There are at most two such intervals. We first divide EI into two intervals, Ea and Eb. Assume that Ea is contained in Rs, but Eb is not. We find the correct rule structures for Ea_{ext} and Eb_{ext}. If $EI_{ext} = Ea_{ext}$, we replace the rule structure of EI_{ext} with that of Ea_{ext}. Also, we add the prefix Eb_{ext} in TCAM using the proposed insertion algorithm in Section 3.2. The operations are similar if $EI_{ext} = Eb_{ext}$. However, if $EI_{ext} \neq Ea_{ext} \neq Eb_{ext}$, we delete EI_{ext} from TCAM and insert the prefixes Ea_{ext} and Eb_{ext} in TCAM. In summary, at most six TCAM update operations are needed for inserting a range.
3. The third step locates any existing interval EI that completely covers Rs if the above two steps are not satisfied. We have described how to process this case in Section 3.2. Since Rs may be divided into at most three intervals, at most four TCAM operations are needed (one deletion and three insertions).

Deletion. As with insertion, we assume that each elementary interval EI is associated with the highest priority range Rh and a range list. Assume there are k contiguous intervals ($E[1]$ to $E[k]$) that are covered by $Rdel$. $E[i]$ ($i = 1$ to k) must not be the default interval because $E[i]$ is at least covered by $Rdel$. For each $E[i]$, the following actions are performed: If $Rdel = Rh$, the highest priority range is removed from the range list of $E[i]$ and replaces Rh. If Rh is the only range covering $E[i]$ (i.e., the range list is empty), $E[i]$ becomes the default interval. By the definition of elementary intervals, the sets of original ranges that cover $E[i]$ and $E[i+1]$ must be different before and after Rdel is deleted. Thus, $E[i]$ and $E[i+1]$ for $i = 1$ to $k-1$ cannot be merged into one interval. However, it is possible that, after deletion, the set of ranges covering $E[1]$ (or $E[k]$) is the same as that of its neighboring interval. Therefore, $E[1]$ (or $E[k]$) needs to be merged with its neighboring interval. As a result, at most six TCAM operations are needed. Notice that the optimized deletion proposed in Section 3.2 can be used here directly.

4 Performance Evaluation

In this section, we first summarize the worst-case performance for all the schemes we studied. They are the two proposed algorithms (denoted by *DISJ* and *CONT*), the direct range-to-prefix conversion in Fig. 2 (denoted by *DIRECT*), the trie node conversion method proposed in [1] (denoted by *TrieNode*), and the range conversion scheme proposed for solving the point intersection problem (denoted by *LCP*) [13] in Table 1. All schemes except DIRECT have an extra delay for accessing the information in SRAM. This delay is not included in the experiments. In columns 4, 5, 8, and 9, we define the number of TCAM operations to be the number of prefixes inserted in or deleted from TCAM. We assume that the times taken for inserting and deleting a prefix into/from TCAM are the same without differentiating the lower-level TCAM operations such as TCAM write and disable operations. The number of TCAM write and disable operations will be calculated based on the CAO_OPT TCAM update algorithm proposed in [8] when we evaluate the performance of inserting/deleting a prefix into/from TCAM.

TABLE 1
The Worst-Case Performance for All Range-to-Prefix Conversion Schemes

Range type / Algorithm	Disjoint Range Set				Arbitrary Range Set			
	No. of TCAM lookups per search	No. of TCAM entries for n ranges	No. of TCAM operations per insertion	No. of TCAM operations per deletion	No. of TCAM lookups per search	No. of TCAM entries for n ranges	No. of TCAM operations per insertion	No. of TCAM operations per deletion
DIRECT	1	O(Wn)	O(W)	O(W)	1	O(Wn)	O(W)	O(W)
TrieNode	-	4n + 2	-	-	-	4n + 2	-	-
LCP	2	2n	2	2	4	4n	4W	4
DISJ	1	2n – 1	W – 2	2	1	4n – 3	O(kW)	O(k)
$DISJ_{opt}$			W – 2	W – 2			O(kW)	O(kW)
CONT	1	2n + 1	4	6	1	2n + 1	6	6
$CONT_{opt}$				0				0

Since the TrieNode scheme [1] does not provide performance results for insertion and deletion, we mark them as "-". When inserting or deleting an arbitrary range R, we assume k default intervals are covered by R.

TABLE 2
Performance versus the Ratios of the Number of Default Intervals to that of Elementary Intervals ($|\mathrm{E}|/|\mathrm{default}|$)

|default| / |E|	50.2%	44.7%	38%	29%	17%
DISJ / |E|	76%	88%	100%	113%	135%
LCP / |E|	200%	200%	200%	200%	200%
TrieNode / |E|	200%	200%	200%	200%	200%
DIRECT / |E|	373%	429%	504%	599%	693%

As shown in Table 1, CONT performs better than other schemes in all cases, except that its worst-case performance of inserting a disjoint range is worse than LCP. In LCP, a disjoint range is represented by two prefixes. Thus, two prefixes are always inserted/deleted into/from TCAM when inserting/deleting a disjoint range. Since the two prefixes converted from a disjoint range are stored in two separate TCAMs, two TCAM lookups are needed per search. As a result, LCP is the worst in terms of search speed among all schemes. All schemes perform better than DIRECT in terms of the number of TCAM entries required. Given that the power consumption of a TCAM is linearly proportional to the number of searched entries, we can use this number as a measure of the TCAM power consumption.

DIRECT is better than DISJ only in the process of inserting or deleting arbitrary ranges. No TCAM operation is needed for the optimized deletion algorithm in $\mathrm{CONT}_{\mathrm{opt}}$ because the operations of merging contiguous intervals are performed periodically or when the TCAM load is light. $\mathrm{DISJ}_{\mathrm{opt}}$ is different from DISJ in that the range to be inserted may be split many times to avoid splitting other existing ranges. Since a disjoint range R may be split into at most $W-2$ subranges in $\mathrm{DISJ}_{\mathrm{opt}}$, at most $W-2$ extended prefixes need to be removed for deleting R. We assume that, in the worst case, k default intervals are covered by the arbitrary range R that is to be inserted. Therefore, for DISJ, inserting R is equivalent to inserting k disjoint ranges plus a constant overhead for the operations dealing with the intervals partially covered by R. As a result, $O(kW)$ TCAM operations are needed, which is the major disadvantage of DISJ. In the last part of our experiments, we shall show that DISJ has a better average performance than other schemes when most of the ranges in an arbitrary range set are disjoint.

To demonstrate the performance difference between DISJ and CONT, we generate range sets such that the ratios of the number of default intervals to that of elementary intervals vary. The number of elementary intervals is kept in the range of 200 to 1,000. Table 2 shows the normalized number of prefixes required in various schemes to that in CONT. The breakpoint between DISJ and CONT is at about 38 percent. When the ratio is larger than 38 percent, the number of required prefixes for DISJ is less than that for CONT. Otherwise, CONT needs a smaller number of prefixes. The number of prefixes required for DIRECT is the largest among all schemes.

Now, we show the experimental results based on ClassBench [12]. The range sets we experimented with are the destination ports extracted from the rule tables generated by ClassBench. Notice that, when the range sets generated from ClassBench become large, they will lose the characteristics of the real rule tables. In other words, a large number of ranges intersect with one another. We only show the results for the destination ports of Firewall tables from ClassBench since similar results for IP Chain and Access Control List are obtained.

To inspect the advantages of DISJ, we perform the same experiments with the range sets in which most of the ranges are disjoint. The following assumptions are made: First, 95 percent of the ranges are mutually disjoint. Second, we randomly generate the remaining 5 percent of the ranges that intersect those mutually disjoint ranges. Third, if a range's start address is less than 1,024, we make it a singleton range by forcing the finish address of the range to be equal to the start address. This setting follows the tradition that the number 1,023 or less is usually a well-known port number for a specific application.

Table 3 shows the numbers of required TCAM entries that can be used as a measure of power consumption. CONT performs best with Firewall rule tables, as shown in Table 3a, because all the elementary intervals are valid

TABLE 3
Numbers of Required TCAM Entries: (a) Destination Port Ranges of Firewall Rule Tables and (b) Randomly Generated Range Sets

# of ranges	200	400	600	800	1000	2000	3000	4000	5000
DIRECT	1770	3291	4493	5782	6912	11674	15886	19572	22994
TrieNode	798	1592	2380	3178	3942	7758	11402	15102	18530
LCP	790	1569	2335	3102	3828	7351	10537	13634	16403
CONT	400	797	1191	1590	1972	3880	5702	7552	9266
DISJ	644	1277	1907	2552	3127	5378	7755	10065	12163

(a)

# of ranges	200	400	600	800	1000	2000	3000	4000	5000
DIRECT	1322	2266	3080	3917	4596	5810	7686	9369	11060
TrieNode	714	1430	2138	2828	3538	8000	11998	15996	19996
LCP	400	798	1189	1576	1972	4043	5886	7710	9509
CONT	358	716	1070	1415	1770	4001	6000	7999	9999
DISJ	277	590	875	1205	1478	2435	3707	4795	6046

(b)

TABLE 4
Numbers of TCAM Writes per Range Insertion:
(a) Destination Port Ranges of Firewall Rule Tables and (b) Randomly Generated Range Sets

# of ranges	200	400	600	800	1000	2000	3000	4000	5000
DIRECT	8.70	8.55	8.00	8.20	8.11	6.32	5.76	5.41	4.76
CONT	3.75	3.96	4.08	4.29	3.69	3.88	3.64	3.48	3.28
DISJ	4.09	3.64	3.67	3.50	3.47	3.30	2.99	2.84	2.54
$DISJ_{opt}$	4.09	3.64	3.67	3.50	3.47	3.30	2.99	2.84	2.54

(a)

# of ranges	200	400	600	800	1000	2000	3000	4000	5000
DIRECT	5.80	4.58	4.83	4.29	4.07	2.94	2.75	2.38	2.27
CONT	3.80	3.59	3.78	3.93	3.69	3.71	3.71	3.44	3.56
DISJ	2.57	1.87	2.09	1.87	1.72	1.65	1.71	1.60	1.66
$DISJ_{opt}$	2.42	1.86	2.10	1.83	1.67	1.64	1.66	1.59	1.62

(b)

intervals and completely cover the entire address space. The number of prefixes needed for TrieNode and LCP is almost double that of CONT. As expected, DIRECT performs much worse than other schemes. Table 3b shows the results for the range sets generated by us. When the number of original ranges is less than or equal to 1,000, a lot of elementary intervals are default intervals which are not used in DISJ. Thus, DISJ outperforms CONT.

Subsequently, we conduct experiments to evaluate the execution time for range update operations. The update operations are based the CAO_OPT TCAM update algorithm [8]. The numbers of TCAM write and disable operations are calculated and used as the update performance metrics. The TCAM disable operation actually disables the TCAM entry that does not need to be active when performing searches in TCAM. We know that a disjoint range R is split into two smaller ranges Re and Rf. In the worst case, we need one TCAM deletion to delete the extended prefix R_{ext} from TCAM and two TCAM insertions to insert the two extended prefixes Re_{ext} and Rf_{ext} into TCAM. In the implementation of the CAO_OPT algorithm, we use a simple batch-like optimization to improve the update performance of the two proposed schemes. We do not treat these three operations independently. We will check whether or not the deleted TCAM entry for R_{ext} can be used to hold one of the newly generated prefixes, say Re_{ext} if $R_{ext} = Re_{ext}$. If yes, only one TCAM write is required to delete R_{ext} and insert Re_{ext}. No TCAM disable operation that is used to disable a TCAM entry corresponding to R_{ext} is needed. Afterward, we use the normal CAO_OPT insertion algorithm to insert Rf_{ext}. When inserting Rf_{ext}, many other TCAM movements (writes) may be needed to satisfy the constraint of chain ancestor ordering required for CAO_OPT. This optimization technique can be used in both DISJ and CONT. The schemes TrieNode and LCP are not included for comparison because no detailed update algorithms were given for arbitrary ranges in [1] and [13].

Table 4 shows the average number of TCAM writes needed for inserting a range. The experiment environment is set up as follows: We first use 90 percent of the original ranges to construct the prefixes according to the specified scheme. Then, we insert the remaining 10 percent of the original ranges one at a time by using the insertion algorithm of the specified scheme. For example, the performance of $DISJ_{opt}$ is obtained by inserting ranges using the proposed optimized insertion algorithm. For the proposed schemes, if an old TCAM entry is deleted, it will

TABLE 5
Numbers of TCAM Write Operations per Range Deletion:
(a) Destination Port Ranges of Firewall Rule Tables and (b) Randomly Generated Range Sets

# of ranges	200	400	600	800	1000	2000	3000	4000	5000
DIRECT	9.85	9.28	8.92	8.58	9.03	7.06	6.22	6.04	5.66
CONT	4.88	4.64	4.74	4.41	4.72	4.40	4.12	4.20	3.70
DISJ	4.89	4.99	4.53	4.60	4.04	3.66	3.50	3.19	3.11
$DISJ_{opt}$	4.07	4.31	4.02	3.96	3.72	3.45	3.19	2.95	2.88

(a)

# of ranges	200	400	600	800	1000	2000	3000	4000	5000
DIRECT	6.10	4.43	4.80	3.96	3.64	2.87	2.57	2.26	2.16
CONT	4.15	4.14	3.81	4.17	3.69	3.63	3.58	3.22	3.34
DISJ	2.67	2.11	2.39	2.02	1.82	1.83	1.69	1.55	1.47
$DISJ_{opt}$	3.10	2.35	2.38	2.07	1.93	1.87	1.68	1.67	1.60

(b)

TABLE 6
Numbers of TCAM Disable Operations per Range Deletion:
(a) Destination Port Ranges of Firewall Rule Tables and (b) Randomly Generated Range Sets

# of ranges	200	400	600	800	1000	2000	3000	4000	5000
DIRECT	7.75	7.03	6.58	6.24	6.32	4.90	4.38	4.09	3.78
CONT	2.00	2.00	2.00	1.96	1.94	1.91	1.79	1.79	1.71
DISJ	3.05	3.03	2.97	2.96	2.68	2.45	2.37	2.24	2.15
$DISJ_{opt}$	3.05	2.93	2.88	2.78	2.62	2.48	2.30	2.18	2.12

(a)

# of ranges	200	400	600	800	1000	2000	3000	4000	5000
DIRECT	6.30	4.50	4.85	4.00	3.65	2.87	2.57	2.26	2.16
CONT	2.00	2.00	2.00	2.00	1.99	2.00	2.00	2.00	2.00
DISJ	2.40	1.98	2.05	1.83	1.67	1.64	1.51	1.40	1.38
$DISJ_{opt}$	2.50	2.00	2.07	1.85	1.70	1.66	1.52	1.41	1.40

(b)

hold the newly inserted prefix. Thus, no TCAM disable operation is needed. CONT performs worse than DISJ and $\mathrm{DISJ}_{\mathrm{opt}}$ for both Firewall rule tables and randomly generated range sets for the following reasons: The number of the extended prefixes constructed in DISJ is different from that in CONT. Thus, the chain ancestor ordering structures (implemented as the binary tries) of DISJ and CONT are different. When we delete a range in CONT, we may have to do at most twice of the following operations: Delete one interval EI and insert two intervals Ea and Eb split from EI. For the ranges from Firewall rule tables, it is likely that the intervals EI, Ea, and Eb are the same as that in DISJ. As shown in Table 4a, DISJ still performs better than CONT. We conjecture that chain ancestor ordering structure favors DISJ because the length of the ancestor chain in DISJ is shorter than that in CONT. Since intervals in CONT are split, the extended prefixes of these split subintervals have less chance to enclose other prefixes. Thus, the length of a chain ancestor becomes shorter for DISJ. As a result, based on the CAO_OPT algorithm, DISJ needs a smaller number of TCAM writes (TCAM entry movements) than CONT. For randomly generated ranges, in addition to the reason explained above, the range to be inserted for DISJ has a large chance of being disjoint from other existing ranges. Thus, DISJ takes a smaller number of TCAM writes than CONT.

For the results from the Firewall rule tables, both DISJ and $\mathrm{DISJ}_{\mathrm{opt}}$ have the same performance for the following reasons: Refer to the second step of the Insertion algorithm in Section 3.3. Let EI be an interval that intersects Rs, the original range to be inserted. EI is divided into two ranges, namely, Ea and Eb, by Rs. For DISJ, we need to delete EI_{ext} from TCAM and insert Ea_{ext} and Eb_{ext} into TCAM. Since Ea and Eb are contiguous subintervals split from EI, it is not possible that there exists an interval that is not split by EI but by Ea or Eb. In other words, when inserting Ea and Eb, no existing interval will be split. Also, the extended prefixes of all intervals completely covered by Rs are already in TCAM. Therefore, both DISJ and $\mathrm{DISJ}_{\mathrm{opt}}$ have the same performance. Table 4b shows the results for the randomly generated ranges. Since most of the randomly generated ranges are disjoint, $\mathrm{DISJ}_{\mathrm{opt}}$ performs a little better than DISJ. Also notice that DIRECT performs better than CONT when the number of ranges is 2,000 or more. This is because, when the number of ranges increases, the addresses contained in each disjoint range decrease. As a result, the average number of prefixes converted from DIRECT decreases.

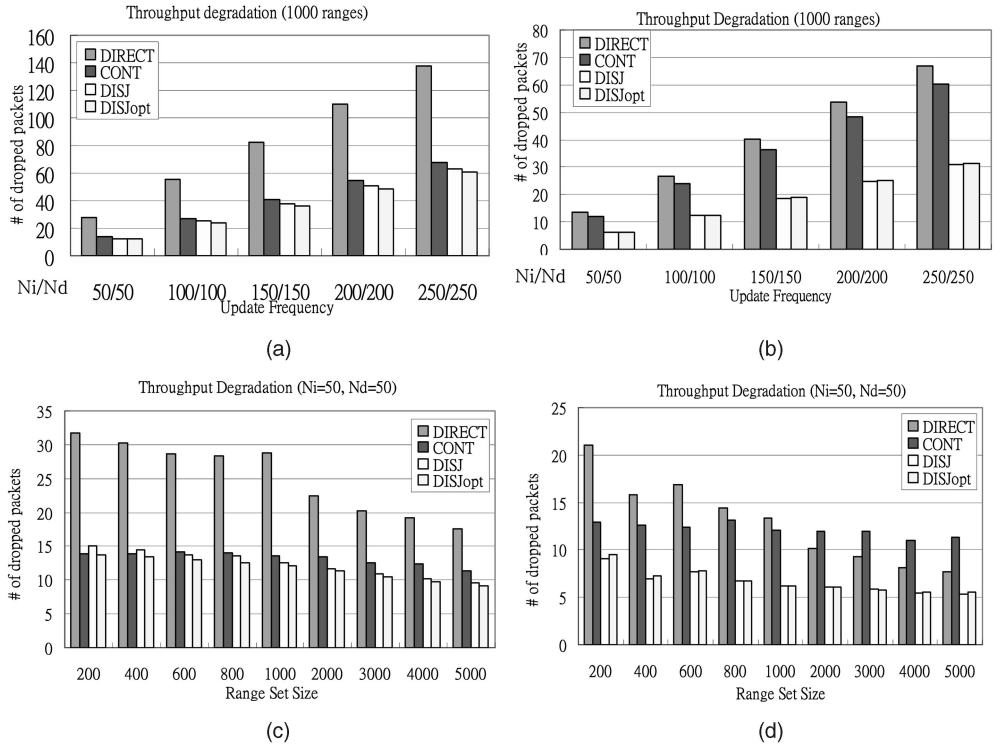

Fig. 11. Throughput degradation comparisons. (a) Destination port ranges of Firewall rule tables. (b) Randomly generated range sets. (c) Destination port ranges of Firewall rule tables. (d) Randomly generated range sets.

Tables 5 and 6 show the average numbers of TCAM write operations and TCAM disable operations needed for deleting a range, respectively. Deleting ranges not only deletes prefixes from TCAM but also inserts prefixes in TCAM sometimes. For DISJ, we construct all the prefixes and randomly delete 10 percent of the ranges. If an insertion is needed when deleting a range, we use the unoptimized insertion algorithm. For DISJ_{opt}, we first construct the needed prefixes by using 90 percent of the ranges and then use optimized insertion algorithm to insert the remaining 10 percent of the ranges. Then, we randomly delete 10 percent of the ranges. If an insertion is needed when deleting a range, we use the optimized insertion algorithm.

Based on the results in Table 5a, DISJ_{opt} performs the best for the following reason: As described earlier, it is possible that, when a range is deleted, two contiguous intervals (say $R1$ and $R2$) may need to be deleted and one interval (say R) that is the union of these two contiguous intervals may need to be inserted. If we treat the deletions of the extended prefixes $R1_{ext}$ and $R2_{ext}$ and the insertion of the extended prefix R_{ext} independently, many existing intervals may be split by R. As a result, a lot of TCAM write operations are required. Based on the optimized insertion algorithm for DISJ_{opt}, we can avoid these costly TCAM operations by splitting R itself. Obviously, the best way is to split R into $R1$ and $R2$ because no other existing range will be split by $R1$ and $R2$. Thus, by combining the deletions of $R1_{ext}$ and $R2_{ext}$ and the insertion of R_{ext}, it is likely that no TCAM write is needed. This is why DISJ_{opt} performs the best. There is no much difference between CONT and DISJ when the number of ranges is less than 1,000. For the results in Table 5b, DISJ performs better than DISJ_{opt} because it is most likely that the range R to be deleted is disjoint from the other existing ranges and DISJ_{opt} may delete more than two prefixes when deleting R. Also notice that DIRECT performs better than CONT when the number of ranges is 1,000 or more because the ranges are small in the randomly generated range set, as explained above.

Table 6 shows the average number of TCAM disable operations needed for deleting a range. Generally, CONT needs a smaller number of TCAM disable operations than DISJ and DISJ_{opt}. The number of TCAM disable operations required for CONT is roughly two for the following reason: As explained in the Deletion algorithm for CONT in Section 3.3, if $E[i]$ to $E[k]$ are the consecutive intervals covered by the range to be deleted, it is likely that $E[1]$ or $E[k]$ needs to be merged with its neighboring interval. In total, four prefixes are deleted and two prefixes are inserted. Since deletion and insertion can be combined, deleting a range only needs two disable operations. DISJ and DISJ_{opt} perform almost the same.

To quantify the performance impact caused by the update operations, we calculate the execution times taken for TCAM to insert or delete a range. We make the following TCAM hardware assumptions, which are similar to the assumptions in [17] and [18]:

1. The TCAM search process is locked during the update process,
2. the TCAM memory width is 64 bits,
3. the TCAM has a sustained search rate of 1 million lookups per second,
4. the local CPU has a clock rate of 100 MHz (10 nsec per clock cycle) with a 64-bit bus, and

5. each TCAM write takes three clock cycles and each TCAM disable takes one clock cycle.

Based on these assumptions, if inserting a range takes n TCAM writes, then $T_i = 3n$ clock cycles are needed to complete the range insertion. If deleting a range takes n TCAM writes and m TCAM disables, then $T_d = 3n + m$ clock cycles are needed to complete the range deletion. We assume that the range update frequency is N_i range insertions and N_d range deletions per second. Thus, the time taken for TCAM to process these updates is $T = N_i \times T_i + N_d \times T_d$. T_i and T_d can be obtained from Tables 4, 5, and 6. We calculate the throughput degradation that is defined to the number of dropped packets due to the TCAM lock during the TCAM update process. Fig. 11 shows the results. In general, DIRECT performs the worst, and DISJ and DISJ_{opt} perform the best. The difference between DISJ and DISJ_{opt} is not significant. One exception is that, when the size of the randomly generated range set is 2,000 or larger, DIRECT performs better than CONT.

5 Conclusion

In this paper, we proposed a novel two-level architecture for storing ranges in TCAM. Only a small overhead of rule structure memory is needed for each TCAM entry. Depending on whether or not the default rule is included, two range-to-prefix conversion algorithms are implemented. The analysis and extensive experiments show that the performances using the proposed schemes are better than those using other existing schemes.

Acknowledgments

This work was supported in part by the National Science Council, Republic of China, under Grant NSC-93-2213-E-006-085.

References

[1] A. Feldmann and S. Muthukrishnan, "Tradeoffs for Packet Classification," *Proc. IEEE INFOCOM,* Mar. 2000.
[2] P. Gupta and N. McKeown, "Algorithms for Packet Classification," *IEEE Network,* vol. 15, no. 2, pp. 324-334, 2001.
[3] T. Woo, "A Modular Approach to Packet Classification: Algorithms and Results," *Proc. IEEE INFOCOM,* 2000.
[4] F. Baboescu and G. Varghese, "Scalable Packet Classification," *Proc. ACM SIGCOMM,* Aug. 2001.
[5] A.L. Buchsbaum, G.S. Fowler, B. Krishnamurthy, K.-P. Vo, and J. Wang, "Fast Prefix Matching of Bounded Strings," *ACM J. Experimental Algorithmics,* vol. 8, Jan. 2003.
[6] S.H. Huang, L.E. Lee, and M.L. Huang, "A Special Purpose Content Addressable Memory for Network Packet Classifier," *Proc. 2004 Symp. Digital Life and Internet Technologies,* 2004.
[7] E. Spitznagel, D. Taylor, and J. Turner, "Packet Classification Using Extended TCAMs," *Proc. IEEE Int'l Conf. Network Protocols (ICNP),* 2003.
[8] D. Shah and P. Gupta, "Fast Updates on Ternary-CAMs for Packet Lookups and Classification," *Proc. Hot-Interconnects VIII,* Aug. 2000, also in *IEEE Micro,* vol. 21, no. 1, pp. 36-47, Jan./Feb. 2002.
[9] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications,* second ed. Springer-Verlag, 2000.
[10] F. Zane, G. Narlikar, and A. Basu, "CoolCAMs: Power-Efficient TCAMs for Forwarding Engines," *Proc. IEEE INFOCOM,* Mar. 2003.
[11] Y.-K. Chang, "Power-Efficient TCAM Partitioning for IP Lookups with Incremental Updates," *Lecture Notes in Computer Science,* vol. 3391, pp. 531-540, Jan. 2005.
[12] D.E. Taylor and J.S. Turner, "ClassBench: A Packet Classification Benchmark," *Proc. IEEE INFOCOM,* 2005.
[13] R. Panigrahy and S. Sharma, "Sorting and Searching Using Ternary CAMs," *IEEE Micro,* vol. 23, no. 1, pp. 44-53, Jan./Feb. 2003.
[14] N. Mohan and M. Sachdev, "Low Power Dual Matchline Ternary Content Addressable Memory," *Proc. IEEE Symp. Circuits and Systems,* pp. 633-636, May 2004.
[15] T. Lakshman and D. Stiliadis, "High-Speed Policy-Based Packet Forwarding Using Efficient Multi-Dimensional Range Matching," *Proc. ACM SIGCOMM,* 1998.
[16] N. Yazdani and P.S. Min, "Fast and Scalable Schemes for the IP Address Lookup Problem," *Proc. IEEE High Performance Switching and Routing,* 2000.
[17] Z. Wang, H. Che, M. Kumar, and S.K. Das, "CoPTUA: Consistent Policy Table Update Algorithm for TCAM without Locking," *IEEE Trans. Computers,* vol. 53, no. 12, pp. 1602-1614, Dec. 2004.
[18] M. Adiletta, M.R. Bluth, D. Bernstein, G. Wolrich, and H. Wilkinson, "The Next Generation of Intel IXP Network Processors," *Intel Technology J.,* vol. 6, no. 3, pp 6-18, 2002.
[19] V.C. Ravikumar, R.N. Mahapatra, and L.N. Bhuyan, "EaseCAM: An Energy and Storage Efficient TCAM-Based Router Architecture for IP Lookup," *IEEE Trans. Computers,* pp. 521-533, May 2005.

Yeim-Kuan Chang received the MSc degree in computer science from the University of Houston at Clear Lake in 1990 and the PhD degree in computer science from Texas A&M University, College Station, Texas, in 1995. He is currently an assistant professor in the Department of Computer Science and Information Engineering at National Cheng Kung University, Tainan, Taiwan, Republic of China. His research interests include computer architecture, multiprocessor systems, Internet router design, and computer networks.