# $O(\log n)$ Dynamic Router-Tables for Prefixes and Ranges

Haibin Lu and Sartaj Sahni, *Fellow*, *IEEE*

**Abstract**—Two versions of the Internet (IP) router-table problem are considered. In the first, the router table consists of $n$ pairs of tuples of the form $(p, a)$, where $p$ is an address prefix and $a$ is the next-hop information. In this version of the router-table problem, we are to perform the following operations: insert a new tuple, delete an existing tuple, and find the tuple with longest matching-prefix for a given destination address. We show that each of these three operations may be performed in $O(\log n)$ time in the worst case using a priority-search tree. In the second version of the router-table problem considered by us, each tuple in the table has the form $(r, a)$, where $r$ is a range of destination addresses matched by the tuple. The set of tuples in the table is conflict-free. For this version of the router-table problem, we develop a data structure that employs priority-search trees as well as red-black trees. This data structure permits us to perform each of the operations insert, delete, and find the tuple with most-specific matching-range for a given destination address in $O(\log n)$ time each in the worst case. The insert and delete operations preserve the conflict-free property of the set of tuples. Experimental results are also presented.

**Index Terms**—Packet routing, dynamic router-tables, longest-prefix matching, most-specific-range matching, conflict-free ranges.

## 1 Introduction

In the general Internet **packet classification** problem, we use a rule table to classify incoming packets. Each entry in the rule table is a pair of the form (rule, action). For each incoming packet, we are to determine the best rule that matches the packet-header fields. Once this is done, the action corresponding to this rule is performed. In one-dimensional packet classification, each rule is either a prefix or a range and the packet-header field used for the classification is the destination address of the packet. In this paper, we are concerned solely with the one-dimensional packet classification problem, which is very closely related to the packet forwarding problem (each action is the next-hop to which the packet is to be sent). Therefore, we refer to the rule table as the router table and to the actions as next-hop information. Our focus in this paper is **dynamic** router-tables, that is, tables into/from which rules are inserted/deleted concurrent with packet classification.

When each rule is a prefix, we refer to the router table as a **prefix router-table**. The length of a prefix is limited by the length $W$ of the destination address ($W = 32$ for IPv4 destination addresses and $W = 128$ for IPv6). In prefix router-tables, the best rule that matches a destination address is the longest prefix that matches this address. Hence, in these tables, we use, what is called **longest-prefix matching** to classify (or lookup) packets. In a **range router-table**, each rule is a range of destination addresses. Several criteria have been proposed to select the best rule that matches a given destination address—first matching-rule in table, highest-priority rule that matches the address, and so on.

In this paper, we show, in Section 4, how priority-search trees may be used to represent dynamic prefix-router-tables. The resulting structure, which is conceptually simpler than the CRBT (collection of red-black trees) structure of [1], permits lookup, insert, and delete in $O(\log n)$ time each in the worst case. Notice that, although a simple balanced search tree for the intervals defined by a set of prefixes permits longest-prefix matching in $O(\log n)$ time, updates require $O(n)$ time. For example, when $W = 5$, the ranges corresponding to prefix set $\{*, 0010*, 1000*, 1010*\}$ are $\{[0, 31], [4, 5], [16, 17], [20, 21]\}$; the (basic) intervals obtained by decomposing the range $[0, 2^W - 1]$ into the natural disjoint ranges defined by the end points of the prefix ranges are

$$\{[0, 3], [4, 5], [6, 15], [16, 17], [18, 19], [20, 21], [22, 31]\};$$

four of these basic intervals correspond to prefix *, so, removing prefix * requires removing these four basic intervals. In general, a prefix may be decomposed into $O(n)$ basic intervals. So, a straightforward solution using a balanced tree structure does not work well for updates.

For range router-tables, we consider the case when the best matching range is the most-specific matching range (this is the range analog of longest-matching prefix). Although much of the research in the router-table area has focused on static prefix-tables, our focus here is dynamic prefix and range-tables. We are motivated to study such tables for the following reasons: First, in a prefix-table, aggregation of prefixes is limited to pairs of prefixes that have the same length and match contiguous addresses. In a range-table, we may aggregate prefixes and ranges that match contiguous addresses regardless of the lengths of the prefixes and ranges being aggregated. So,

- *H. Lu is with the Univeristy of Missouri-Columbia, 107 Engr. Bldg. West, Columbia, MO 65211-2060. E-mail: luhaibin@missouri.edu.*
- *S. Sahni is with the Computer and Information Sciences and Engineering Department, University of Florida, Gainesville, FL 32611. E-mail: sahni@cise.ufl.edu.*

*Manuscript received 12 Feb. 2004; accepted 11 May 2004.*
*For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0045-0204.*

Authorized licensed use limited to: UNIVERSIDADE FEDERAL FLUMINENSE. Downloaded on June 02,2021 at 00:45:32 UTC from IEEE Xplore. Restrictions apply.

range aggregation is expected to result in router tables that have fewer rules. Second, with the move to QoS services, router-table rules include ranges for port numbers (for example). Although ternary content addressable memories (TCAMs), the most popular hardware solution for prefix tables, can handle prefixes naturally, they are unable to handle ranges directly. Rather, ranges are decomposed into prefixes. Since each range takes up to $2W-2$ prefixes to represent, decomposing ranges into prefixes may result in a large increase in router-table size. Multidimensional classifiers typically have one or more fields that are ranges. Since data structures for multidimensional classifiers are built on top of data structures for one-dimensional classifiers, it is necessary to develop good data structures for one-dimensional range router-tables (as we do in this paper). Third, dynamic tables that permit high-speed inserts and deletes are essential in QoS (Quality of Service) and VAS (Value Added Service) applications [2].

In Section 5, we show that dynamic range-router-tables that employ most-specific range matching and in which no two ranges intersect (Definition 2) may be efficiently represented using two priority-search trees. Using this two-priority-search-tree representation, lookup, insert, and delete can be done in $O(\log n)$ time each in the worst case.

The general case of nonconflicting ranges (Definition 5) is considered in Section 6. In a nonconflicting range set, two ranges may intersect (partially overlap, but one range is not completely contained in another). Although range intersection may be an infrequent occurrence in IP packet forwarding, it is a frequent occurrence for range fields of single and multidimensional QoS classifiers. In multidimensional packet classification, several filters may intersect. For intersecting multidimensional filters, Hari et al. [3] introduced the notion of *filter conflict* and used resolve filters to make filter sets conflict free under the most-specific-matching rule. Our definition of conflict-free range is a natural extension to ranges of the definition of conflict free given by Hari et al. [3]. Since an efficient solution to one-dimensional packet classification is essential if we are to have an efficient solution for multidimensional packet classification, our work with respect to intersecting ranges may be considered a stepping stone to an efficient solution for multidimensional packet classification. In Section 6, we augment the data structure of Section 5 with several red-black trees to obtain a range-router-table representation for nonconflicting ranges that permits lookup, insert, and delete in $O(\log n)$ time each in the worst case.

Section 2 lists related work and Section 3 introduces the terminology we use. Experimental results are reported in Section 7.

## 2 Related Work

Ruiz-Sanchez et al. [4] review data structures for static prefix router-tables and Sahni et al. [5] review data structures for both static and dynamic prefix router-table design. Several trie-based data structures for prefix router-tables have been proposed [6], [7], [8], [9], [10], [11], [12]. Structures such as that of [6] perform each of the dynamic router-table operations (lookup, insert, delete) in $O(W)$ time. Others (e.g., [7], [8], [9], [10], [11], [12]) attempt to optimize lookup time and memory requirement through an expensive preprocessing step. These structures, while providing very fast lookup capability, have a prohibitive insert/delete time (insert/delete may involve a rebuild of the entire structure) and, so, they are suitable only for static router-tables (i.e., tables into/from which no inserts and deletes take place).

Waldvogel et al. [13] have proposed a scheme that performs a binary search on hash tables organized by prefix length. This binary search scheme has an expected complexity of $O(\log W)$ for lookup. Waldvogel et al.'s scheme is very similar to the *k*-ary search-on-length scheme developed by Berg et al. [14] and the binary search-on-length schemes developed by Willard [15]. Berg et al. [14] used a variant of stratified trees [16] for one-dimensional point location in a set of $n$ disjoint ranges. Willard [15] modified stratified trees and proposed the y-fast trie data structure to search a set of disjoint ranges. By decomposing filter ranges that are not disjoint into disjoint ranges, the schemes of [14], [15] may be used for longest-prefix matching in router tables. The asymptotic complexity using the schemes of [14], [15] is the same as that of Waldvogel et al.'s scheme [13]. An alternative adaptation of binary search to longest-prefix matching is developed in [17]. Using this adaptation, a lookup in a table that has $n$ prefixes takes $O(W + \log n)$ time. Because the schemes of [13] and [17] use expensive precomputation, they are not suited for dynamic router-tables.

Suri et al. [18] have proposed a B-tree data structure for dynamic router tables. Using their structure, we may find the longest matching prefix in $O(\log n)$ time. However, inserts/deletes take $O(W \log n)$ time. Even though the structure of Suri et al. [18] takes more time to find a longest matching-prefix than do structures optimized for static router-tables, the structure of Suri et al. [18] has a significantly more favorable ratio between lookup and update times, making it more suitable for high-update applications.

Sahni and Kim [1] developed a data structure, called a collection of red-black trees (CRBT), that supports the three operations of a dynamic prefix-router table in $O(\log n)$ time each. In [19], Sahni and Kim show that their CRBT structure is easily modified to extend the biased-skip-list structure of Ergun et al. [20] so as to obtain a biased-skip-list structure for dynamic prefix-router-tables. Using this modified biased skip-list structure, lookup, insert, and delete can each be done in $O(\log n)$ expected time. Like the original biased-skip list structure of [20], the modified structure of [19] adapts so as to perform lookups faster for bursty access patterns than for nonbursty patterns. The CRBT structure may also be adapted to obtain a collection of splay trees structure [19] which performs the three dynamic prefix-router-table operations in $O(\log n)$ amortized time and which adapts to provide faster lookups for bursty traffic.

Cheung and McCanne [21] develop "a model for table-driven route lookup and cast the table design problem as an optimization problem within this model." Their model accounts for the memory hierarchy of modern computers and they optimize average performance rather than worst-case performance.

Authorized licensed use limited to: UNIVERSIDADE FEDERAL FLUMINENSE. Downloaded on June 02,2021 at 00:45:32 UTC from IEEE Xplore. Restrictions apply.

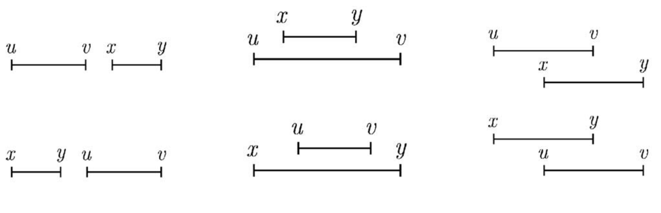


Fig. 1. Relationships between pairs of ranges. (a) Disjoint. (b) Nested. (c) Intersect.

Hardware solutions that involve the use of content addressable memory [22] as well as solutions that involve modifications to the Internet Protocol (i.e., the addition of information to each packet) have also been proposed [23], [24], [25].

Gupta and McKeown [26] have developed two data structures for dynamic range-router-tables—heap on trie (HOT) and binary search tree on trie (BOT). Both of these are for the case when the best-matching rule is the highest-priority rule that matches the given destination address. The HOT takes $O(W)$ time for a lookup and $O(W \log n)$ time for an insert or delete. The BOT structure takes $O(W \log n)$ time for a lookup and $O(W)$ time for an insert/delete.

## 3 Preliminaries

### 3.1 Prefixes and Longest-Prefix Matching

The prefix 1101* (the prefix is a binary prefix) matches all destination addresses that begin with 1101 and 10010* matches all destination addresses that begin with 10010. For example, when $W = 5$, 1101* matches the addresses $\{11010, 11011\} = \{26, 27\}$ and, when $W = 6$, 1101* matches $\{110100, 110101, 110110, 110111\} = \{52, 53, 54, 55\}$. Suppose that a router table includes the prefixes $P1 = 101*$, $P2 = 10010*$, $P3 = 01*$, $P4 = 1*$, and $P5 = 1010*$. The destination address $d = 1010100$ is matched by the prefixes $P1$, $P4$, and $P5$. Since $|P1| = 3$ (the length of a prefix is number of bits in the prefix), $|P4| = 1$ and $|P5| = 4$, $P5$ is the longest prefix that matches $d$. In **longest-prefix routing**, the next hop for a packet destined for $d$ is given by the longest prefix that matches $d$.

### 3.2 Ranges

**Definition 1.** *A* **range** *$r = [u, v]$ is a pair of addresses $u$ and $v$, $u \le v$. The range $r$ represents the addresses $\{u, u+1, \ldots, v\}$. $start(r) = u$ is the start point of the range and $finish(r) = v$ is the finish point of the range. The range $r$* **covers** *or* **matches** *all addresses $d$ such that $u \le d \le v$. $isRange(q)$ is a predicate that is true iff $q$ is a range.*

The start point of the range $r = [3, 9]$ is 3 and its finish point is 9. This range covers or matches the addresses $\{3, 4, 5, 6, 7, 8, 9\}$. In IPv4, $u$ and $v$ are up to 32 bits long and, in IPv6, $u$ and $v$ may be up to 128 bits long. The IPv4 prefix $P = 0*$ corresponds to the range $[0, 2^{31} - 1]$. The range $[3, 9]$ does not correspond to any single IPv4 prefix. We may draw the range $r = [u, v] = \{u, u+1, ..., v\}$ as a horizontal line that begins at $u$ and ends at $v$. Fig. 1 shows ranges drawn in this fashion.

Notice that every prefix of a prefix router-table may be represented as a range. For example, when $W = 6$, the prefix $P = 1101*$ matches addresses in the range $[52, 55]$. So, we say $P = 1101* = [52, 55]$, $start(P) = 52$, and $finish(P) = 55$.

Since a range represents a set of (contiguous) points, we may use standard set operations and relations such as $\cap$ and $\subset$ when dealing with ranges. So, for example, $[2, 6] \cap [4, 8] = [4, 6]$. Note that some operations between ranges may not yield a range. For example, $[2, 6] \cup [8, 10] = \{2, 3, 4, 5, 6, 8, 9, 10\}$ is not a range.

**Definition 2.** *Let $r = [u, v]$ and $s = [x, y]$ be two ranges.*

1. *The predicate $isDisjoint(r, s)$ is true iff $r$ and $s$ are disjoint.*

$$isDisjoint(r, s) \Longleftrightarrow r \cap s = \emptyset \Longleftrightarrow v < x \vee y < u.$$

2. *The predicate $isNested(r, s)$ is true iff one of the ranges is contained within the other.*

$$\begin{aligned} isNested(r, s) &\Longleftrightarrow r \cap s = r \vee r \cap s = s \\ &\Longleftrightarrow r \subseteq s \vee s \subseteq r \\ &\Longleftrightarrow x \le u \le v \le y \vee u \le x \le y \le v. \end{aligned}$$

3. *The predicate $isIntersect(r, s)$ is true iff $r$ and $s$ have a nonempty intersection that is different from both $r$ and $s$.*

$$\begin{aligned} isIntersect(r, s) &\Longleftrightarrow r \cap s \ne \emptyset \wedge r \cap s \ne r \wedge r \cap s \ne s \\ &\Longleftrightarrow u < x \le v < y \vee x < u \le y < v. \end{aligned}$$

*Notice that $r \cap s = [x, v]$ when $u < x \le v < y$ and $r \cap s = [u, y]$ when $x < u \le y < v$.*

$[2, 4]$ and $[6, 9]$ are disjoint; $[2, 4]$ and $[3, 4]$ are nested; $[2, 4]$ and $[2, 2]$ are nested; $[2, 8]$ and $[4, 6]$ are nested; $[2, 4]$ and $[4, 6]$ intersect; and $[3, 8]$ and $[2, 4]$ intersect. $[4, 4]$ is the overlap of $[2, 4]$ and $[4, 6]$; and $[3, 8] \cap [2, 4] = [3, 4]$.

**Lemma 1.** *Let $r$ and $s$ be two ranges. Exactly one of the following is true: $isDisjoint(r, s)$, $isNested(r, s)$, $isIntersect(r, s)$.*

**Proof.** Straightforward. □

### 3.3 Most-Specific-Range Routing and Conflict-Free Ranges

**Definition 3.** *The range $r$ is* **more specific** *than the range $s$ iff $r \subset s$.*

$[2, 4]$ is more specific than $[1, 6]$ and $[5, 9]$ is more specific than $[5, 12]$. Since $[2, 4]$ and $[8, 14]$ are disjoint, neither is more specific than the other. Also, since $[4, 14]$ and $[6, 20]$ intersect, neither is more specific than the other.

**Definition 4.** *Let $R$ be a range set. We define $ranges(d, R)$ (or simply $ranges(d)$ when $R$ is implicit) as the subset of ranges of $R$ that match/cover the destination address $d$. We define $msr(d, R)$ (or $msr(d)$) as the most specific range of $R$ that matches $d$. That is, $msr(d)$ is the most specific range in $ranges(d)$. $msr([u, v], R) = msr(u, v, R) = r$ iff $msr(d, R) = r$, $u \le d \le v$. When $R$ is implicit, we write $msr(u, v)$ and $msr([u, v])$ in place of $msr(u, v, R)$ and $msr([u, v], R)$. $msr(d)$ may not exist. In*

Authorized licensed use limited to: UNIVERSIDADE FEDERAL FLUMINENSE. Downloaded on June 02,2021 at 00:45:32 UTC from IEEE Xplore. Restrictions apply.

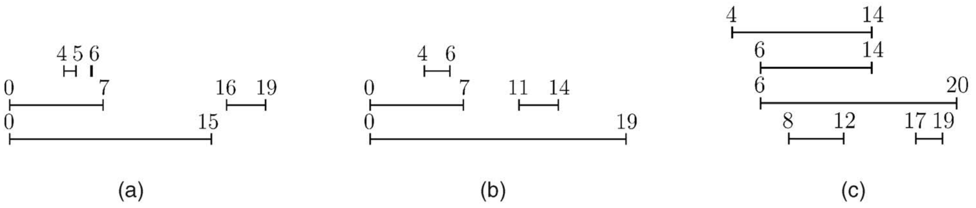


Fig. 2. Sample set: (a) Prefixes ($W = 5$) $\{0*, 00*, 0010*, 00110, 100*\}$. (b) Nonintersecting ranges. Range $[4, 6]$ can't be represented by single prefix, neither can $[0, 19]$, $[11, 14]$. (c) Conflict-free ranges. Ranges $[4, 14]$ and $[6, 20]$ intersect.

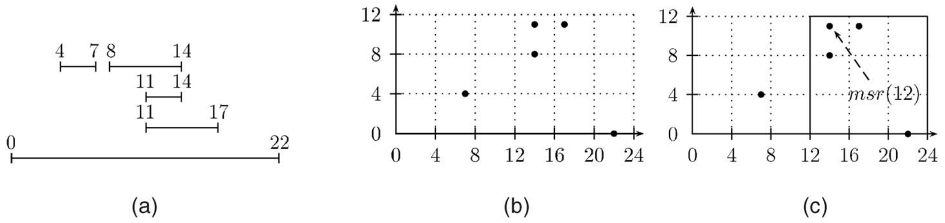


Fig. 3. (a) An example range set $R$. (b) Its mapping $map1(R)$ into points in 2D. The $x$ value is the range finish point and the $y$ value is the range start point. (c) $ranges(12)$ includes all the points inside the solid rectangle since range $[st, fn]$ such that $st = y \leq 12 \leq fn = x$ matches 12. $msr(12) = [11, 14]$ has the largest $y$ value among those having the smallest $x$ value. $minXinRectangle(2^W 12 + (2^W - 1 - 12), \infty, 12)$ performed on $PST1$ yields $msr(12)$.

**most-specific-range routing**, *the next hop for packets destined for d is given by the next-hop information associated with* $msr(d)$.

When $R = \{[2, 4], [1, 6]\}$, $ranges(3) = \{[2, 4], [1, 6]\}$, $msr(3) = [2, 4]$, $msr(1) = [1, 6]$, $msr(7) = \emptyset$, and $msr(5, 6) = [1, 6]$. When

$$R = \{[4, 14], [6, 20], [6, 14], [8, 12], [17, 19]\},$$

shown in Fig. 2c, $msr(4, 5) = [4, 14]$, $msr(6, 7) = [6, 14]$, $msr(8, 12) = [8, 12]$, $msr(13, 14) = [6, 14]$, and

$$msr(15, 20) = [6, 20].$$

**Definition 5.** *The range set R has a* **conflict** *iff there exists a destination address d for which* $ranges(d) \neq \emptyset \wedge msr(d) = \emptyset$. *R is* **conflict free** *iff it has no conflict. The predicate* $isConflictFree(R)$ *is true iff R is a conflict-free range set.*

$isConflictFree(\{[2, 8], [4, 12], [4, 8]\})\}$ is true while $isConflictFree(\{[2, 8], [4, 12]\})\}$ is false.

Searching for $msr(d)$ in a conflict-free range set using a priority search tree is based on Lemma 2.

**Lemma 2.** *Let R be a conflict-free range set and let d be a destination address. If* $ranges(d) \neq \emptyset$, *then* $start(msr(d)) = a = maxStart(ranges(d)) = \max\{start(r) | r \in ranges(d)\}$ *and*

$$finish(msr(d)) = b = minFinish(ranges(d)) \\ = \min\{finish(r) | r \in ranges(d)\}.$$

**Proof.** Since $R$ is conflict free and $ranges(d) \neq \emptyset$, $msr(d) \neq \emptyset$. Assume that $msr(d) = s$. If $s \neq [a, b]$, then $start(s) < a$ or $finish(s) > b$. Assume that $start(s) < a$ (the case $finish(s) > b$ is similar). Let $t \in ranges(d)$ be such that $start(t) = a$. Now, $isIntersect(s, t) \vee t \subset s$. Hence, $s \neq msr(d)$. □

## 3.4 Priority Search Trees and Ranges

A priority-search tree (PST) [27] is a data structure that is used to represent a set of tuples of the form $(key1, key2, data)$, where $key1 \geq 0$, $key2 \geq 0$, and no two tuples have the same $key1$ value. The data structure is simultaneously a min-tree on $key2$ (i.e., the $key2$ value in each node of the tree is $\leq$ the $key2$ value in each descendent node) and a search tree on $key1$. There are two common PST representations [27]:

1. In a **radix priority-search tree** (RPST), the underlying tree is a binary radix tree on $key1$.
2. In a **red-black priority-search tree** (RBPST), the underlying tree is a red-black tree.

McCreight [27] has suggested a PST representation of a collection of ranges with distinct finish points. This representation uses the following mapping of a range $r$ into a PST tuple:

$$(key1, key2, data) = (finish(r), start(r), data), \tag{1}$$

where $data$ is any information (e.g., next hop) associated with the range. Each range $r$ is, therefore mapped to a point $map1(r) = (x, y) = (key1, key2) = (finish(r), start(r))$ in two-dimensional space. Fig. 3 shows a set of ranges and the equivalent set of two-dimensional points $(x, y)$.

McCreight [27] has observed that, when the mapping of (1) is used to obtain a point set $P = map1(R)$ from a range set $R$, then $ranges(d)$ is given by the points that lie in the rectangle (including points on the boundary) defined by $x_{left} = d$, $x_{right} = \infty$, $y_{top} = d$, and $y_{bottom} = 0$. These points are obtained using the method

$$enumerateRectangle(x_{left}, x_{right}, y_{top}) \\ = enumerateRectangle(d, \infty, d)$$

of a PST ($y_{bottom}$ is implicit and is always 0).

When an RPST is used to represent the point set $P$, the complexity of

Authorized licensed use limited to: UNIVERSIDADE FEDERAL FLUMINENSE. Downloaded on June 02,2021 at 00:45:32 UTC from IEEE Xplore. Restrictions apply.

$$enumerateRectangle(x_{left}, x_{right}, y_{top})$$

is $O(\log maxX + s)$, where $maxX$ is the largest $x$ value in $P$ and $s$ is the number of points in the query rectangle. When the point set is represented as an RBPST, this complexity becomes $O(\log n + s)$, where $n = |P|$. A point $(x, y)$ (and, hence, a range $[y, x]$) may be inserted into or deleted from an RPST (RBPST) in $O(\log maxX)$ ($O(\log n)$) time [27]. We shall employ these RPST (RBPST) algorithms here without further description. These algorithms are described in detail in [27].

### 3.5 Prefix Router-Table versus Range Router-Table

The management of dynamic prefix router-tables is simplified by the fact that inserting and/or deleting from a set of prefixes leaves behind a set of prefixes. When dealing with nonintersecting ranges, each insertion must verify that the new range doesn't intersect any of the existing ranges. This makes the case of nonintersecting ranges slightly harder to handle. Note, however, that when a range is deleted from a set of nonintersecting ranges, the range set that remains is nonintersecting. The case of conflict-free ranges is the hardest to handle. Both the insertion and deletion of a range from a set of conflict-free ranges may cause the set of remaining ranges to have conflicts. Further, checking for conflicts is harder than checking for intersections. In the following sections, we examine these three cases of filter sets in increasing order of difficulty—prefixes, nonintersecting ranges, conflict-free ranges.

## 4 Prefixes

Let $R$ be a set of ranges such that each range represents a prefix. It is well-known (see [1], for example) that no two ranges of $R$ intersect. Therefore, $R$ is conflict free. For simplicity, assume that $R$ includes the range that corresponds to the prefix * (prefix * is the default prefix, its length is zero, and it matches all destination addresses). With this assumption, $msr(d)$ is defined for every $d$. From Lemma 2, it follows that $msr(d)$ is the range $[maxStart(ranges(d)), minFinish(ranges(d))]$. To find this range easily, we first transform $P = map1(R)$ into a point set $transform1(P)$ so that no two points of $transform1(P)$ have the same $x$-value. Then, we represent $transform1(P)$ as a PST.

**Definition 6.** *Let $W$ be the (maximum) number of bits in a destination address ($W = 32$ in IPv4). Let $(x, y) \in P$. $transform1(x, y) = (x', y') = (2^W x + (2^W - 1 - y), y)$ and $transform1(P) = \{transform1(x, y) \mid (x, y) \in P\}$.*

We see that $0 \leq x' < 2^{2W}$ for every $(x', y') \in transform1(P)$ and that no two points in $transform1(P)$ have the same $x'$-value. Let $PST1(P)$ be the PST for $transform1(P)$. The operation

$$enumerateRectangle(2^W d + (2^W - 1 - d), \infty, d)$$

performed on $PST1$ yields $ranges(d)$ since $(y \leq d \leq x) \Leftrightarrow ((y' \leq d) \wedge (2^W d + (2^W - 1 - d) \leq x'))$. To find $msr(d)$, we employ the

$$minXinRectangle(x'_{left}, x'_{right}, y'_{top})$$

operation, which determines the point in the defined rectangle that has the least $x'$-value.

$$minXinRectangle(2^W d + (2^W - 1 - d), \infty, d)$$

performed on $PST1$ yields $msr(d)$ ($y'_{bottom}$ is implicit and is always 0). The point $(x', y')$ returned corresponds to the point $(x, y)$ that has the largest $y$-value among the points having the least $x$-value in the rectangle defined by $x_{left} = d$, $x_{right} = \infty$, $y_{top} = d$, and $y_{bottom} = 0$. From the definition of $map1$, such an $(x, y)$ corresponds to the range $[maxStart(ranges(d)), minFinish(ranges(d))]$.

To insert the prefix whose range is $[u, v]$, we insert $transform1(map1([u, v]))$ into $PST1$. In case this prefix is already in $PST1$, we simply update the next-hop information for this prefix. To delete the prefix whose range is $[u, v]$, we delete $transform1(map1([u, v]))$ from $PST1$. When deleting a prefix, we must take care not to delete the prefix *. Requests to delete this prefix should simply result in setting the next-hop associated with this prefix to $\emptyset$.

Since $minXinRectangle$, insert, and delete each take $O(\log n)$ time when $PST1$ is a RBPST, $PST1$ provides a router-table representation in which longest-prefix matching, prefix insertion, and prefix deletion can be done in $O(\log n)$ time each when an RBPST is used.

## 5 Nonintersecting Ranges

Let $R$ be a set of nonintersecting ranges (note that $R$ may contain nested as well as disjoint ranges). Clearly, $R$ is conflict free. For simplicity, assume that $R$ includes the range $z$ that matches all destination addresses ($z = [0, 2^{32} - 1]$ in the case of IPv4). With this assumption, $msr(d)$ is defined for every $d$. We may use $PST1(transform1(map1(R)))$ to find $msr(d)$ as described in Section 4.

Insertion of a range $r$ is to be permitted only if $r$ does not intersect any of the ranges of $R$. Once we have verified this, we can insert $r$ into $PST1$ as described in Section 4. Range intersection may be verified by noting that there are two cases for range intersection (Definition 2.3). When inserting $r = [u, v]$, we need to determine if $\exists s = [x, y] \in R[u < x \leq v < y \vee x < u \leq y < v]$. We see that $\exists s = [x, y] \in R[x < u \leq y < v]$ iff $map1(R)$ has at least one point in the rectangle defined by $x_{left} = u$, $x_{right} = v - 1$, and $y_{top} = u - 1$ (recall that $y_{bottom} = 0$ by default). Hence, $\exists s = [x, y] \in R[x < u \leq y < v]$ iff

$$\begin{aligned} minXinRectangle(&2^W u + (2^W - 1 - (u - 1)), \\ &2^W (v - 1) + (2^W - 1), u - 1) \end{aligned}$$

exists in PST1.

To verify $\exists s = [x, y] \in R[u < x \leq v < y]$, map the ranges of $R$ into two-dimensional points using the mapping, $map2(r) == (start(r), 2^W - 1 - finish(r))$. Call the resulting set of mapped points $map2(R)$. We see that $\exists s = [x, y] \in R[u < x \leq v < y]$ iff $map2(R)$ has at least one point in the rectangle defined by $x_{left} = u + 1$, $x_{right} = v$, and $y_{top} = (2^W - 1) - (v + 1)$. To verify this, we maintain a second PST, $PST2$ of points in $transform2(map2(R))$, where $transform2(x, y) = (2^W x + y, y)$ Hence, $\exists s = [x, y] \in R[u < x \leq v < y]$ iff

Authorized licensed use limited to: UNIVERSIDADE FEDERAL FLUMINENSE. Downloaded on June 02,2021 at 00:45:32 UTC from IEEE Xplore. Restrictions apply.

$$minXinRectangle(2^W(u+1), 2^W v + (2^W - 1) - (v+1), (2^W - 1) - (v+1))$$

exists.

To delete a range $r$, we must delete $r$ from both $PST1$ and $PST2$. Deletion of a range from a PST is similar to deletion of a prefix as discussed in Section 4.

The complexity of the operations to find $msr(d)$, insert a range, and delete a range is the same as that for these operations for the case when $R$ is a set of ranges that correspond to prefixes.

## 6 Conflict-Free Ranges

In this section, we extend the two-PST data structure of Section 5 to the general case when $R$ is an arbitrary conflict-free range set. Once again, we assume that $R$ includes the range $z$ that matches all destination addresses. $PST1$ and $PST2$ are defined for the range set $R$ as in Sections 4 and 5. Section 6.1 shows how to determine $msr(d)$, Section 6.2 introduces the resolving subset for two intersecting ranges. Section 6.3 gives the algorithm for inserting a range. Before inserting a new range $r$ into a conflict-free range set $R$, we need to determine whether or not $R \cup \{r\}$ is conflict-free. Section 6.4 gives the algorithm for deleting a range. Before deleting an existing range $r$ from a conflict-free range set $R$, we need to determine whether or not $R - \{r\}$ is conflict-free. To efficiently compute the functions $maxP$ and $minP$ (Definition 9) that are used to verify the conflict-free property prior to inserting/deleting a range, we employ the notion of a normalized range set. Normalized range sets are introduced in Section 6.5 The methods to compute $maxP$ and $minP$ are given in Section 6.6 and the method to update the normalized range set is developed in Section 6.7.

### 6.1 Determine $msr(d)$

Since $R$ is conflict free, $msr(d)$ is determined by Lemma 2. Hence, $msrd(d)$ may be obtained by performing the operation

$$minXinRectangle(2^W d + (2^W - 1 - d), \infty, d)$$

on PST1.

### 6.2 Projections and Resolving Subsets

**Definition 7.** *Let $R = \{r_1, \dots, r_n\}$ be a set of $n$ ranges. The* **projection**, $\Pi(R)$, *of $R$ is*

$$\Pi(R) = \cup_{i=1}^{n} r_i.$$

*That is, $\Pi(R)$ comprises all addresses that are covered by at least one range of $R$.*

For $A = \{[2,5],[3,6],[8,9]\}$, $\Pi(A) = \{2,3,4,5,6,8,9\}$ and, for $B = \{[4,8],[7,9]\}$, $\Pi(B) = \{4,5,6,7,8,9\}$. $\Pi(A)$ is not a range. However, $\Pi(B)$ is the range $[4,9]$. Note that $\Pi(R)$ is a range iff $d \in \Pi(R)$ for every $d$, $u \le d \le v$, where $u = \min\{d | d \in \Pi(R)\}$ and $v = \max\{d | d \in \Pi(R)\}$.

**Definition 8.** *Let $r$ and $s$ be two intersecting ranges of the range set $R$. The subset $Q \subset R$ is a* **resolving subset** *within $R$ for these two ranges iff $Q$ is conflict free and $\Pi(Q) = r \cap s$. Two ranges of a range set are in* **conflict** *iff they intersect and have no resolving subset. Two ranges are* **conflict free** *iff they are not in conflict.*

**Lemma 3.** *A range set is conflict free iff it has no pair of ranges that are in conflict.*

**Proof.** See the Appendix (which can be found on the Computer Society Digital Library at http://computer.org/tc/archives.htm). □

### 6.3 Insert a Range $r = [u, v]$

When inserting a range $r = [u,v] \notin R$, we must insert $transform(map1(r))$ into $PST1$ and insert $transform2(map2(r))$ into $PST2$. Additionally, we must verify that $R \cup \{r\}$ is conflict free. This verification is done using Lemma 4.

**Lemma 4.** *Let $R$ be a conflict-free range set. Let $A = R \cup \{r\}$, where $r = [u,v] \notin R$.*

$$isConflictFree(A) \Longleftrightarrow maxY(u,v,R) \le maxP(u,v,R) \wedge minX(u,v,R) \ge minP(u,v,R),$$

*where $maxY \le maxP$ ($minX \ge minP$) is true whenever $maxY$ ($minX$) does not exist and is false when $maxY$ ($minX$) exists but $maxP$ ($minP$) does not.*

**Proof.** See the Appendix (which can be found on the Computer Society Digital Library at http://computer.org/tc/archives.htm). □

*$maxP$, $minP$, $maxY$,* and *$minX$* are defined below.

**Definition 9.** *$maxP(u,v,R) = \max\{finish(\Pi(A)) | A \subseteq R \wedge isRange(\Pi(A)) \wedge start(\Pi(A)) = u \wedge finish(\Pi(A)) \le v\}$ is the maximum finish point of a possible projection that is a range that starts at $u$ and finishes by $v$.*

$$minP(u,v,R) = \min\{start(\Pi(A)) | A \subseteq R \wedge isRange(\Pi(A)) \wedge finish(\Pi(A)) = v \wedge start(\Pi(A)) \ge u\}$$

*is the minimum start point of a possible projection that is a range that finishes at $v$ and starts by $u$.*

*When*

$$\nexists A \subseteq R[isRange(\Pi(A)) \wedge start(\Pi(A)) = u \wedge finish(\Pi(A)) \le v],$$

*we say that $maxP(u,v,R)$ does not exist. Similarly, $minP(u,v,R)$ may not exist. At times, we use $maxP$ and $minP$ as abbreviations for $maxP(u,v,R)$ and $minP(u,v,R)$, respectively.*

*$maxY(u,v,R) = \max\{y | [x,y] \in R \wedge x < u \le y < v\}$ and $minX(u,v,R) = \min\{x | [x,y] \in R \wedge u < x \le v < y\}$. Note that $maxY$ and $minX$ may not exist.*

Fig. 4 gives a high-level description of our algorithm to insert a range into $R$. Step 1 is done by searching for $transform1(map1(r))$ in $PST1$. For Step 2, we note that

Authorized licensed use limited to: UNIVERSIDADE FEDERAL FLUMINENSE. Downloaded on June 02,2021 at 00:45:32 UTC from IEEE Xplore. Restrictions apply.

**Step 1:** If $r = [u, v] \in R$, update the next-hop information associated with $r \in R$ and terminate.

**Step 2:** Compute $maxP(u, v, R)$, $minP(u, v, R)$, $maxY(u, v, R)$ and $minX(u, v, R)$.

**Step 3:** If $maxY(u, v, R) \leq maxP(u, v, R) \wedge minX(u, v, R) \geq minP(u, v, R)$, $R \cup \{r\}$ is conflict free; otherwise, it is not. In the former case, insert $transform1(map1(r))$ into $PST1$ and $transform2(map2(r))$ into $PST2$. In the latter case, the insert operation fails.

Fig. 4. Insert $r = [u, v]$ into the conflict-free range set $R$.

**Step 1:** If $r = z$, change the next-hop information for $z$ to $\emptyset$ and terminate.

**Step 2:** Delete $transform1(map1(r))$ from $PST1$ and $transform2(map2(r))$ from $PST2$ to get the PSTs for $A = R - \{r\}$. If $PST1$ did not have $transform1(map1(r))$, $r \notin R$; terminate.

**Step 3:** Determine whether or not $A$ has a subset whose projection equals $r = [u, v]$.

**Step 4:** If $A$ has such a subset, conclude $isConflictFree(A)$ and terminate.

**Step 5:** Determine whether $A$ has a range that contains $r = [u, v]$. If not, conclude $isConflictFree(A)$ and terminate.

**Step 6:** Determine $m$ and $n$ as defined in Lemma 5 as follows.
$m = start(maxXinRectangle(0, 2^W u + (2^W - 1) - v, (2^W - 1) - v)$ (use PST2)
$n = finish(minXinRectangle(2^W v + (2^W - 1 - u), \infty, u)$ (use PST1)

**Step 7:** Determine whether $[m, n] \in A$. If so, conclude $isConflictFree(A)$. Otherwise, conclude $\neg isConflictFree(A)$. In the latter case reinsert $transform1(map1(r))$ into $PST1$ and $transform2(map2(r))$ into $PST2$ and disallow the deletion of $r$.

Fig. 5. Delete the range $r = [u, v]$ from the conflict-free range set $R$.

$$maxY(u, v, R) = maxXinRectangle(2^W u + (2^W - 1 - (u - 1)), 2^W(v - 1) + (2^W - 1), u - 1)$$
$$minX(u, v, R) = minXinRectangle(2^W(u + 1), 2^W v + (2^W - 1) - (v + 1), (2^W - 1) - (v + 1)),$$

where, for $maxY$, we use $PST1$ and, for $minX$, we use $PST2$. Section 6.6 describes the computation of $maxP$ and $minP$. The point insertions of Step 3 are done using the standard insert algorithm for a PST [27].

## 6.4 Delete a Range $r = [u, v]$

Suppose we are to delete the range $r = [u, v]$. This deletion is to be permitted iff $r \neq z$ ($z$ is the default range that matches all destination addresses) and $A = R - \{r\}$ is conflict free. Its correctness follows from Lemma 5.

**Lemma 5.** *Let $R$ be a conflict-free range set. Let $A = R - \{r\}$, for some $r \in R$.*

1. $\exists B \subseteq A[\Pi(B) = r] \Longrightarrow isConflictFree(A)$.
2. $$\nexists B \subseteq A[\Pi(B) = r] \Longrightarrow [isConflictFree(A) \Longleftrightarrow ((\nexists s \in A[r \subset s]) \vee ([m, n] \in A))],$$
   *where* $m = \max\{start(s) | s \in A \wedge r \subseteq s\}$ *and* $n = \min\{finish(s) | s \in A \wedge r \subseteq s\}$.

**Proof.** See the Appendix (which can be found on the Computer Society Digital Library at http://computer.org/tc/archives.htm). □

Assume $r = [7, 9]$, $m = 6$, and $n = 10$. $R - \{r\}$ contains the ranges $[5, 10]$ and $[6, 12]$. But, range $[m, n] = [6, 10] = [5, 10] \cap [6, 12]$ is not in $R - \{r\}$. So, $R - \{r\}$ is not conflict free.

Fig. 5 gives a high-level description of our algorithm to delete $r$. Step 2 employs the standard PST algorithm to delete a point [27]. For Step 3, we note that $A$ has a subset whose projection equals $r = [u, v]$ iff $maxP(u, v, A) = v$. In Section 6.6, we show how $maxP(u, v, A)$ may be computed efficiently. For Step 5, we note that $r = [u, v] \subseteq s = [x, y]$ iff $x \leq u \wedge y \geq v$. So, $A$ has such a range iff

$$minXinRectangle(2^W v + (2^W - 1 - u), \infty, u)$$

exists in $PST1$.

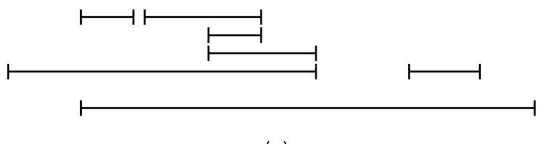
In Step 6, we assume that $maxXinRectangle$ and $minXinRectangle$ return the range of $R$ that corresponds to the desired point in the rectangle. To determine whether $[m, n] \in A$ (Step 7), we search for the point $(2^W n + (2^W - 1 - m), m)$ in $PST1$ using the standard PST search algorithm [27]. The reinsertion into $PST1$ and $PST2$, if necessary, is done using the standard $PST$ insert algorithm [27].

## 6.5 Normalized Ranges for Computing $maxP$ and $minP$

**Definition 10 (Normalized Ranges).** *The range set $R$ is* **normalized** *iff one of the following is true:*

1. $|R| \leq 1$.
2. $|R| > 1$ *and, for every $r \in R$ and every $s \in R$, $r \neq s$, one of the following is true.*

Authorized licensed use limited to: UNIVERSIDADE FEDERAL FLUMINENSE. Downloaded on June 02,2021 at 00:45:32 UTC from IEEE Xplore. Restrictions apply.


(a)

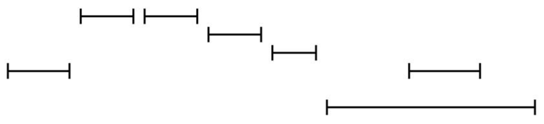

(b)

Fig. 6. Unnormalized and normalized range sets.

a. $isDisjoint(r, s)$.
b.

$$isNested(r, s) \wedge start(r) \neq start(s) \wedge \\ finish(r) \neq finish(s).$$

*That is, $r$ and $s$ are nested and do not have a common end-point.*

Fig. 6a shows a range set that is not normalized (it contains ranges that intersect as well as nested ranges that have common end-points). Fig. 6b shows a normalized range set.

**Definition 11.** *An ordered sequence of ranges $(r_1, \ldots, r_n)$ is a* **chain** *iff $\forall i < n[start(r_{i+1}) = finish(r_i) + 1]$. A range set $R$ is a chain iff its ranges can be ordered so as to form a chain. $isChain(R)$ is a predicate that is true iff $R$ is a chain.*

The range sequence $([2,4],[5,7],[8,12])$ is a chain while $([5,8],[12,14])$ and $([5,8],[2,4])$ are not. The range sets $\{[5,8],[2,4]\}$ and $\{[2,4],[8,12],[5,7]\}$ are chains while $\{[2,4],[8,12]\}$ and $\{[2,4],[5,7],[8,12],[9,10]\}$ are not. Note that, when $R$ is a chain,

$$\Pi(R) = [minStart(R), maxFinish(R)].$$

**Lemma 6.** *Let $N$ be a normalized range set.*

1. *$N$ may be uniquely partitioned into a set of longest chains $CP(N) = \{C_1, \ldots, C_k\}$, $N = \cup_{i=1}^{k} C_i$. By longest chains, we mean that no two chains of $CP$ may be combined into a single chain. $CP(N)$ is called a* **canonical partitioning**.
2. *For all $i$ and $j$, $1 \leq i < j \leq k$, $\Pi(C_i)$ and $\Pi(C_j)$ are either disjoint or $C_i$ is properly contained within a range of $C_j$ or $C_j$ is properly contained within a range of $C_i$. A chain $C_i$ is properly contained within the range $r$ iff $\Pi(C_i) \subset r$ and $C_i$ and $r$ share no end point.*

**Proof.** See the Appendix (which can be found on the Computer Society Digital Library at http://computer.org/tc/archives.htm). □

Fig. 7 shows a normalized range set and its canonical partitioning into three chains.

Next, we state a chopping rule that we use to transform every conflict-free range set $R$ into a normalized range set $norm(R)$.

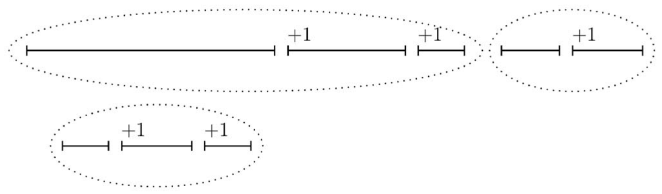


Fig. 7. Partitioning a normalized range set into chains.

**Definition 12 (Chopping Rule).** *Let $r = [u, v] \in R$, where $R$ is a range set. $chop(r, R)$ (or, more simply, $chop(r)$ when $R$ is implicit) is as defined below.*

1. *If neither $maxP(u, v-1, R)$ nor $minP(u+1, v, R)$ exists, $chop(r) = r$.*
2. *If only $maxP(u, v-1, R)$ exists,*
$$chop(r) = [maxP(u, v-1, R) + 1, v].$$
3. *If only $minP(u+1, v, R)$ exists,*
$$chop(r) = [u, minP(u+1, v, R) - 1].$$
4. *If both $maxP(u, v-1, R)$ and $minP(u+1, v, R)$ exist and*
$$maxP(u, v-1, R) + 1 \leq minP(u+1, v, R) - 1, \\ chop(r) = [maxP(u, v-1, R) + 1, \\ minP(u+1, v, R) - 1].$$
5. *If both $maxP(u, v-1, R)$ and $minP(u+1, v, R)$ exist and*
$$maxP(u, v-1, R) + 1 > minP(u+1, v, R) - 1,$$
*$chop(r) = \emptyset$, where $\emptyset$ denotes the null range. The null range neither intersects nor is contained in any other range.*

*Define $norm(R) = \{chop(r) \mid r \in R \wedge chop(r) \neq \emptyset\}$.*

**Lemma 7.** *Let $R$ be a conflict-free range set. For every $r' \in norm(R)$, there is a unique $r \in R$ such that $chop(r) = r'$.*

**Proof.** See the Appendix (which can be found on the Computer Society Digital Library at http://computer.org/tc/archives.htm). □

For $r' \in norm(R)$, define $full(r') = chop^{-1}(r') = r$, where $r$ is the unique range in $R$ for which $chop(r) = r'$. Notice that $full(chop(r)) = r$ except when $chop(r) = \emptyset$.

## 6.6 Computing $maxP$ and $minP$

Although $maxP$ and $minP$ are relatively difficult to compute using data structures such as $PST1$ and $PST2$ that directly represent $R$, they may be computed efficiently using data structures for $norm(R)$. In this section, we show how to compute $maxP$ from $norm(R)$. The computation of $minP$ is similar.

Authorized licensed use limited to: UNIVERSIDADE FEDERAL FLUMINENSE. Downloaded on June 02,2021 at 00:45:32 UTC from IEEE Xplore. Restrictions apply.

```
Step 1: Find $r' \in norm(R)$ such that $start(r') = u$.
        If (no such $r'$) $\vee$ $start(full(r')) \neq u$ $\vee$ $finish(full(r')) > v$, $maxP$ does not exist; terminate.

Step 2: $maxP = finish(r')$;
        while $(\exists s' \in norm(R)[(start(s') = maxP + 1) \wedge (full(s') \subseteq [u, v])])$
            $maxP = finish(s')$;
```

Fig. 8. Simple algorithm to compute $maxP(u, v, R)$, where $[u, v]$ is a range and $isConflictFree(R)$.

### 6.6.1 A Simple Algorithm to Compute $maxP$

Fig. 8 is a high-level description of a simple, though not efficient, algorithm to compute $maxP(u, v, R)$. Step 1 determines whether or not there is a range that is nested inside $[u, v]$ and starts at $u$. If there is no such range, $maxP$ does not exist. Step 2 extends $maxP$ as far as possible by following a chain.

**Theorem 1.** *Fig. 8 correctly computes* $maxP(u, v, R)$.

**Proof.** See the Appendix (which can be found on the Computer Society Digital Library at http://computer.org/tc/archives.htm). □

### 6.6.2 An Efficient Algorithm to Compute $maxP$

The algorithm of Fig. 8 takes time $O(length(C_i))$, where $length(C_i)$ is the number of ranges in the chain $C_i \in CP(norm(R))$ that contains $r'$. We can reduce this time to $O(\log length(C_i))$ by representing each chain of $CP(norm(R))$ as a red-black tree (actually, any balanced search tree structure that permits efficient join and split operations may be used). The number of red-black trees we use equals the number of chains in $CP(norm(R))$.

Let $D = (t'_1, \ldots, t'_q)$ be a chain in $CP(norm(R))$. The red-black tree, $RBT(D)$, for $D$ has one node for each of the ranges $t'_i$. The key value for the node for $t'_i$ is $start(t'_i)$ (equivalently, $finish(t'_i)$ may be used as the search tree key). Each node of $RBT(D)$ has the following four values (in addition to having a $t'_i$ and other values necessary for efficient implementation): $minStartLeft$, $minStartRight$, $maxFinishLeft$, and $maxFinishRight$. For a node $p$ that has an empty left subtree, $minStartLeft = 2^W - 1$ and $maxFinishLeft = 0$. Similarly, when $p$ has an empty right subtree, $minStartRight = 2^W - 1$ and $maxFinishRight = 0$. Otherwise,

$$minStartLeft = \min\{start(full(r'))|r' \in leftSubtree(p)\}$$
$$minStartRight = \min\{start(full(r'))|r' \in rightSubtree(p)\}$$
$$maxFinishLeft = \max\{finish(full(r'))|r' \in leftSubtree(p)\}$$
$$maxFinishRight = \max\{finish(full(r'))|r' \in rightSubtree(p)\}.$$

The collection of red-black trees representing $norm(R)$ is augmented by an additional red-black tree $endPointsTree(norm(R))$ that represents the end points of the ranges in $norm(R)$. With each point $x$ in $endPointsTree$, we store a pointer to the node in $RBT(D)$ that represents $s'$.

To implement Step 1 of Fig. 8, we search $endPointsTree$ for the point $u$. If $u \notin endPointsTree$, then $\nexists r' \in norm(R)[start(r') = u]$. If $u \in endPointsTree$, then we use the pointer in the node for $u$ to get to the root of the $RBT$ that has $r'$. A search in this $RBT$ for $u$ locates $r'$. We may now perform the remaining checks of Step 1 using the data associated with $r'$.

Suppose that $maxP$ exists. At the start of Step 2, we are positioned at the $RBT$ node that represents $r'$. This is node 0 of Fig. 9. We need to find $s' \in norm(R)$ with least $s'$ such that $start(s') > finish(r') \wedge full(s') \not\subseteq [u, v]$. If there is no such $s'$, then

$$maxP = \max\{finish(root.range), root.maxFinishRight\}.$$

If such an $s'$ exists, $maxP = start(s') - 1$.

$s'$ may be found in $O(height(RBT))$ time using a simple search process. We illustrate this process using the tree of Fig. 9. We begin at node 0. If $[minStartRight, maxFinishRight] \subseteq [u, v]$, then $s'$ is not in the right subtree of node 0. Since node 0 is a right child, $s'$ is not in its parent. So, we back up to node 1 (in general, we back up to the nearest ancestor whose left subtree contains the current node). Let $t'_1$ be the range in node 1. $s' = t'$ iff $t' \not\subseteq [u, v]$. If $s' \neq t'$, we perform the test $[minStartRight, maxFinishRight] \subseteq [u, v]$ at node 1 to determine whether or not $s'$ is in the right subtree of node 1. If the test is true, we back up to node 2. Otherwise, $s'$ is in the right subtree of node 1. When the right subtree (if any) that contains $s'$ is identified, we make a

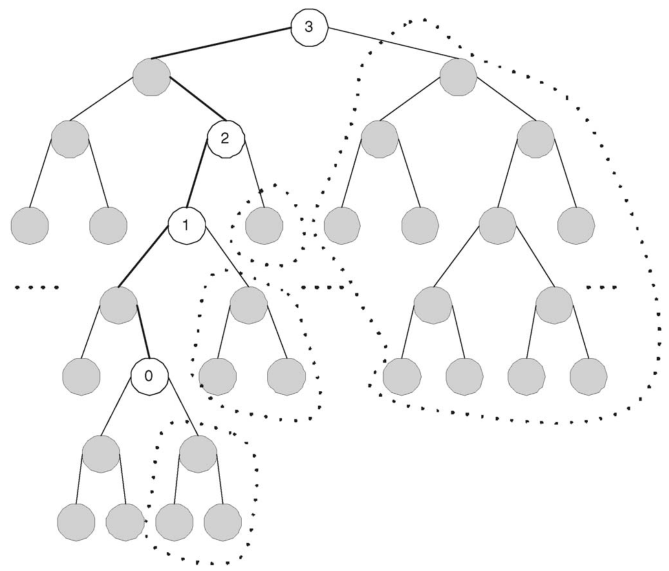


Fig. 9. An example $RBT$.

Authorized licensed use limited to: UNIVERSIDADE FEDERAL FLUMINENSE. Downloaded on June 02,2021 at 00:45:32 UTC from IEEE Xplore. Restrictions apply.

```
$downwardPass(currentNode)$
    // $currentNode$ is the root of a subtree all of whose ranges start at the right of $u$
    // This subtree contains $s'$. Return $maxP$.
    while (true) {
        if ($[currentNode.minStartLeft, currentNode.maxFinishRight] \subseteq [u, v]$)
            // $s'$ not in left subtree
            if ($currentNode.range \subseteq [u, v]$)
                // $s' \notin currentNode$. $s'$ must be in right subtree.
                $currentNode = currentNode.rightChild$;
            else return ($start(currentNode.range) - 1$);
        else // $s'$ is in left subtree
            $currentNode = currentNode.leftChild$;
    }
```

Fig 10. Find $s'$ (and, hence, $maxP$) in a subtree known to contain $s'$.

downward pass in this subtree to locate $s'$. Fig. 10 describes this downward pass.

## 6.7 Update $norm(R)$

Now that we have augmented $PST1$ and $PST2$ with a collection of $RBT$s and an $endPointsTree$, whenever we insert a range $r = [u, v]$ into $R$ or delete a range $r = [u, v]$ from $R$, we must update not only $PST1$ and $PST2$ as described in Section 6.3, but also the $RBT$ collection and $endPointsTree$.

### 6.7.1 Update $norm(R)$ after Inserting $r = [u, v]$

The $norm(R)$ update algorithm following inserting $r = [u, v]$ is based on Lemmas 8 and 9. Lemma 8 tells us that, when a range $r$ is inserted into the conflict-free range set $R$, the $chop()$ value may change only for the smallest enclosing range $s \in R$ of $r$.

**Lemma 8.** *Let $R$ be a conflict-free range set. Let $r \notin R$ be such that $R \cup \{r\}$ is conflict free.*

1. $$chop(r, R \cup \{r\}) = \emptyset \Longrightarrow \forall t \in R[chop(t, R) = chop(t, R \cup \{r\})].$$

2. *Let $s$ be the smallest range of $R$ that contains $r$. Assume that $s$ exists and that $chop(r, R \cup \{r\}) \neq \emptyset$.*

   a. $\forall t \in R - \{s\}[chop(t, R) = chop(t, R \cup \{r\})]$.

   b. $$chop(s, R) \neq chop(s, R \cup \{r\}) \Longrightarrow (x' = u' \wedge y' = v') \vee (x' = u' \wedge y' > v) \vee (x' < u \wedge y' = v'),$$

   *where $r = [u, v]$,*

   $$chop(r, R \cup \{r\}) = chop(r, R) = [u', v'],$$

   *and $chop(s, R) = [x', y']$.*

**Proof.** See the Appendix (which can be found on the Computer Society Digital Library at http://computer.org/tc/archives.htm). □

Lemma 9 provides a method to compute $chop(s, R \cup \{r\})$ for the smallest enclosing range $s \in R$ of $r \notin R$.

**Lemma 9.** *Let $R$, $r = [u, v]$, $s = [x, y]$, $x'$, $y'$, $u'$, and $v'$ be as in Lemma 8. Assume that $s$ exists and $chop(s) \neq \emptyset$.*

1. $$isDisjoint(r, chop(s, R)) \vee x' < u \leq v < y' \Longrightarrow chop(s, R \cup \{r\}) = chop(s, R).$$

2. $x' = u' \wedge y' = v' \Longrightarrow chop(s, R \cup \{r\}) = \emptyset$.
3. *Suppose $x' = u' \wedge y' > v$. If $maxP(v' + 1, y', R)$ doesn't exist, then $chop(s, R \cup \{r\}) = [v + 1, y']$. If it exists,*

   $$chop(s, R \cup \{r\}) = [maxP(v' + 1, y', R) + 1, y'].$$

4. *Suppose $x' < u' \wedge y' = v'$. If $minP(x', u' - 1, R)$ doesn't exist, then $chop(s, R \cup \{r\}) = [x', u - 1]$. If it exists,*

   $$chop(s, R \cup \{r\}) = [x', minP(x', u' - 1, R) - 1].$$

**Proof.** See the Appendix (which can be found on the Computer Society Digital Library at http://computer.org/tc/archives.htm). □

To update $RBT$ collection and $endPointsTree$, we first compute $chop(r, R \cup \{r\}) = chop(r, R) = [u', v']$ by first computing $minP(u + 1, v)$ and $maxP(u, v - 1)$ as described in Section 6.6. $[u', v']$ is now easily obtained from the chopping rule. Since $z \in R$ and $r \neq z$ ($z$ is prefix *), such an $s$ must exist. Then, $chop(s, R \cup \{r\})$ can be computed using Lemma 9.

Note that the insertion of $r$ may combine two chains of $CP(norm(R))$. In this case, we use the $join$ operation of red-black trees to combine the $RBT$s corresponding to these two chains.

### 6.7.2 Update $norm(R)$ after Deleting $r = [u, v]$

The $norm(R)$ update algorithm following deleting $r = [u, v]$ is based on Lemma 10. Lemma 10 tells us that the only $s \in R \cup \{r\}$, whose $chop()$ value may change as a result of the deletion of $r$, is the smallest enclosing range of $r$. This lemma also provides a way to compute $chop(s, R - \{r\})$.

Authorized licensed use limited to: UNIVERSIDADE FEDERAL FLUMINENSE. Downloaded on June 02,2021 at 00:45:32 UTC from IEEE Xplore. Restrictions apply.

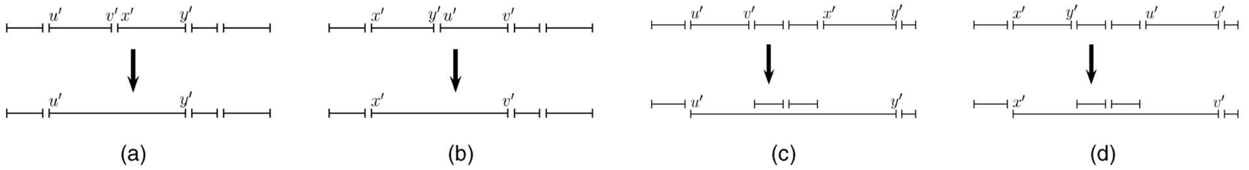


Fig. 11. Cases when $s'$ and $r'$ are in the same chain of $CP(norm(R))$.

**Lemma 10.** *Let $R$ be a conflict-free range set. Let $r = [u, v] \in R$ be such that $R - \{r\}$ is conflict free.*

1.
$$chop(r, R) = \emptyset \Longrightarrow$$
$$\forall t \in R - \{r\}[chop(t, R) = chop(t, R - \{r\})].$$

2. *Let $s = [x, y]$ be the smallest range of $R - \{r\}$ that contains $r$. Assume that $s$ exists and that $chop(r, R) = [u', v']$.*

   a. $\forall t \in R - \{r, s\}[chop(t, R) = chop(t, R - \{r\})]$.
   b. $chop(s, R) = \emptyset \Longrightarrow chop(s, R - \{r\}) = [u', v']$.
   c.
$$chop(s, R) = [x', y'] \Longrightarrow$$
$$chop(s, R - \{r\}) = [\min\{x', u'\}, \max\{y', v'\}].$$

**Proof.** See the Appendix (which can be found on the Computer Society Digital Library at http://computer.org/tc/archives.htm). □

When $chop(r, R) = \emptyset$, no changes are to be made to the $RBT$s and $endPointsTree$ (Lemma 10.1). So, assume that $chop(r, R) \neq \emptyset$. We first find $s$, the smallest range that contains $r$ (see Lemma 10.2). Note that, since $z \in R$ and $r \neq z$, $s$ exists. One may verify that $s$ is one of the ranges given by the following two operations:

$$minXinRectangle(2^W v + (2^W - 1 - u), \infty, u),$$
$$maxXinRectangle(0, 2^W u + 2^W - 1 - v, 2^W - 1 - v),$$

where the first operation is done in $PST1$ and the second in $PST2$ (both operations are done after $transform1(map1([u, v]))$ has been deleted from $PST1$ and $transform2(map2([u, v]))$ has been deleted from $PST2$). The ranges returned by these two operations may be compared to determine which is $s$.

Once we have identified $s$, Lemma 10.2 is used to determine $chop(s, R - \{r\})$. Assume that $chop(s, R) \neq \emptyset$. Let $chop(r, R) = r' = [u', v']$ and $chop(s, R) = s' = [x', y']$. When $s'$ and $r'$ are in different $RBT$s (this is the case when $r' \subset s'$), $chop(s, R) = chop(s, R - \{r\})$ and the $RBT$ that contains $r'$ may need to be split into two $RBT$s. When $s'$ and $r'$ are in the same $RBT$, they are in the same chain of $CP(norm(R))$. If $s'$ and $r'$ are adjacent ranges of this chain, we may simply remove the $RBT$ node for $r'$ and update that for $s'$ to reflect its new start or finish point (only one may change). When $r'$ and $s'$ are not adjacent ranges, the nodes for these two ranges are removed from the $RBT$ (this may split the $RBT$ into up to two $RBT$s) and $chop(s, R - \{r\})$ is inserted. Fig. 11 shows the different cases.

## 6.8 Complexity

The portions of the search, insert, and delete algorithms that deal only with $PST1$ and $PST2$ have the same asymptotic complexity as their counterparts for the case of nonintersecting ranges (Section 5). The portions that deal with the $RBT$s and $endPointsTree$ require a constant number of search, insert, delete, join, and split operations on these structures. Since each of these operations takes $O(\log n)$ time on a red-black tree and since we can update the values $minStartLeft$, $minStartRight$, and so on, that are stored in the $RBT$ nodes in the same asymptotic time as taken by an insert/delete/join/split, the overall complexity of our proposed data structure is $O(\log n)$ for each operation when $RBPST$s are used for $PST1$ and $PST2$.

## 7 Experimental Results

### 7.1 Prefixes

We programmed our red-black priority-search tree algorithm for prefixes (Section 4) in C++ and compared its performance to that of the ACRBT of [19]. The ACRBT is the best performing $O(\log n)$ data structure reported in [19] for dynamic prefix-tables. For test data, we used six IPv4 prefix databases obtained from [28]. The databases Paix1, Pb1, MaeWest, and Aads were obtained on 22 November 2001, while Pb2 and Paix2 were obtained 13 September 2000. As can be seen in Table 1, the ACRBT structure takes almost three times as much memory as is taken by the PST structure. Further, the memory requirement of the PST structure can be reduced to about 50 percent that of our current implementation. This reduction requires an $n$-node implementation of a priority-search tree, as described in [27], rather than our current implementation, which uses $2n - 1$ nodes, as in [29].

For the database Paix2, the optimal height 2 variable stride trie that results when the controlled prefix expansion scheme of [10] is used requires 2.5MB [12]; when the trie height is 3, 1.1MB is needed. Using the $n$-node priority-search tree implementation of [27], our priority-search scheme would take about 2.4MB. This is very competitive with the height 2 optimal variable stride trie (2OVST [10]),

TABLE 1
Database Size and Memory Usage of PST and ACRBT [19]

| Database | | Paix1 | Pb1 | MaeWest | Aads | Pb2 | Paix2 |
|---|---|---|---|---|---|---|---|
| Num of Prefixes | | 16172 | 22225 | 28889 | 31827 | 35303 | 85988 |
| Memory (KB) | PST | 884 | 1215 | 1579 | 1740 | 1930 | 4702 |
| | ACRBT | 2417 | 3331 | 4327 | 4769 | 5305 | 12851 |

Authorized licensed use limited to: UNIVERSIDADE FEDERAL FLUMINENSE. Downloaded on June 02,2021 at 00:45:32 UTC from IEEE Xplore. Restrictions apply.

TABLE 2
Prefix Times on a 700 MHz Pentium III PC with 256K L2 Cache

| Database | | | Paix1 | Pb1 | MaeWest | Aads | Pb2 | Paix2 |
|---|---|---|---|---|---|---|---|---|
| Search ($\mu$sec) | PST | Mean | 1.39 | 1.54 | 1.61 | 1.65 | 1.70 | 1.97 |
| | | Std | 0.27 | 0.22 | 0.17 | 0.14 | 0.00 | 0.04 |
| | ACRBT | Mean | 1.36 | 1.44 | 1.44 | 1.49 | 1.54 | 1.80 |
| | | Std | 0.25 | 0.18 | 0.13 | 0.14 | 0.14 | 0.06 |
| Insert ($\mu$sec) | PST | Mean | 2.41 | 2.63 | 2.60 | 2.83 | 2.80 | 3.07 |
| | | Std | 0.87 | 0.30 | 0.53 | 0.43 | 0.40 | 0.14 |
| | ACRBT | Mean | 11.97 | 12.63 | 13.48 | 13.62 | 13.77 | 14.93 |
| | | Std | 0.95 | 0.67 | 0.24 | 0.48 | 0.35 | 0.18 |
| Delete ($\mu$sec) | PST | Mean | 2.32 | 2.38 | 2.49 | 2.45 | 2.55 | 2.91 |
| | | Std | 0.82 | 0.61 | 0.52 | 0.47 | 0.00 | 0.17 |
| | ACRBT | Mean | 11.69 | 12.55 | 12.95 | 13.01 | 13.40 | 14.10 |
| | | Std | 0.87 | 0.63 | 0.54 | 0.44 | 0.48 | 0.16 |

TABLE 3
Nonintersecting Ranges, PIII 700MHz with 256K L2 Cache

| Database | | Paix1 | Pb1 | MaeWest | Aads | Pb2 | Paix2 |
|---|---|---|---|---|---|---|---|
| Insert ($\mu$sec) | Mean | 6.68 | 6.69 | 7.17 | 6.93 | 7.27 | 8.22 |
| | Std | 0.93 | 0.53 | 0.43 | 0.46 | 0.43 | 0.18 |
| Delete ($\mu$sec) | Mean | 5.56 | 6.01 | 5.92 | 6.36 | 6.12 | 7.30 |
| | Std | 0.43 | 0.69 | 0.49 | 0.46 | 0.35 | 0.29 |

and a little more that twice the memory needed for the 3OVST [10].

To obtain the mean time to find the longest matching-prefix (i.e., to perform a search), we started with a PST or ACRBT that contained all prefixes of a prefix database. Next, a random permutation of the set of start points of the ranges corresponding to the prefixes was obtained. This permutation determined the order in which we searched for the longest matching-prefix for each of these start points. The time required to determine all of these longest-matching prefixes was measured and averaged over the number of start points (equal to the number of prefixes). The experiment was repeated 20 times and the mean and standard deviation of the 20 mean times computed. Table 2 gives the mean time required to find the longest matching-prefix on a 700MHz Pentium III PC that has a 256KB L2 cache. The standard deviation in the mean time is also given in this table. On our PC, finding the longest matching-prefix takes about 2 to 10 percent less time using an ACRBT than a PST. Extrapolating from the times reported in [1], [19], [30], we anticipate that a search in a PST takes about three times as much time as a search in a 2OVST.

To obtain the mean time to insert a prefix, we started with a random permutation of the prefixes in a database, inserted the first 67 percent of the prefixes into an initially empty data structure, measured the time to insert the remaining 33 percent, and computed the mean insert time by dividing by the number of prefixes in 33 percent of the database. This experiment was repeated 20 times and the mean of the mean as well as the standard deviation in the mean computed. These latter two quantities are given in Table 2 for our PC. As can be seen, insertions into a PST take about 20 percent of the time to insert into an ACRBT!

The mean and standard deviation data reported in Table 2 for the delete operation were obtained in a similar fashion by starting with a data structure that had 100 percent of the prefixes in the database and measuring the time to delete a randomly selected 33 percent of these prefixes. Deletion from a PST takes about 20 percent of the time required to delete from an ACRBT.

Extrapolating from the times reported in [1], [19], [30], we anticipate that an insert in a 2OVST takes about 75 times as much time as an insert in our PST structure and that the corresponding ratio for a delete is about 300!

Fig. 12 histograms the search, insert, and delete time data of Table 2. Experimental results for a 500MHz UltraSparc-IIe workstation and 1.4GHz Pentium 4 PC may be found in [30]. The comparative performance between the PST and ACRBT is the same as for the Pentium III PC.

## 7.2 Nonintersecting Ranges

Since no two prefixes may intersect, we may use prefix databases to benchmark our data structure (Section 5) for nonintersecting ranges. The search time for our six IPv4 prefix databases is the same using the data structure for nonintersecting ranges as it is when the data structure for prefixes is used. However, the memory requirement is doubled since we now have two priority search trees, $PST1$ and $PST2$. Table 3 gives the the mean times and standard deviations for insert and delete. The run times are for our 700MHz Pentium III PC. The insert and delete experiments were modeled after those conducted for the case of prefix databases. Since the insert algorithm for the case of nonintersecting ranges requires us to first verify nonintersection with existing ranges and then insert into two

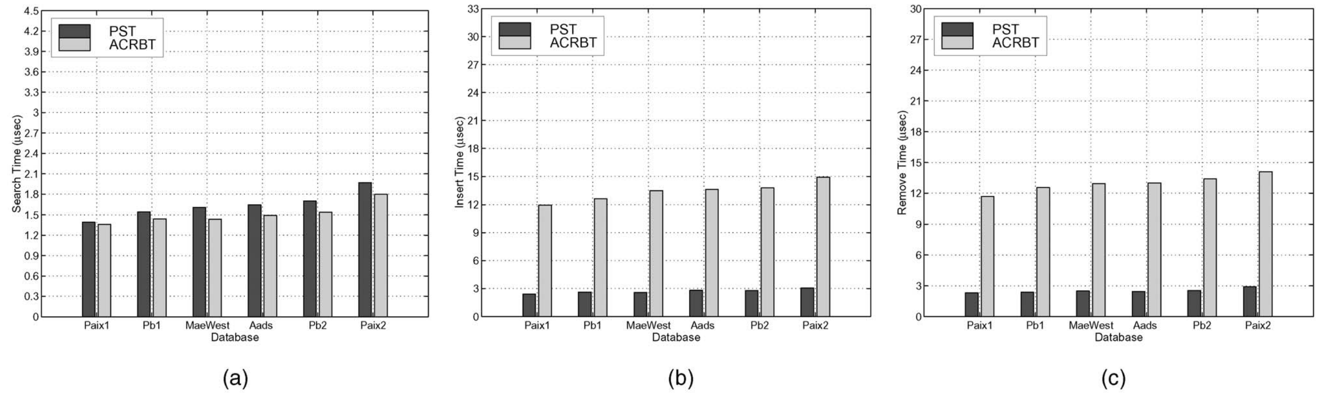


Fig. 12. Times on a 700MHz Pentium III PC with 256K L2 cache.

Authorized licensed use limited to: UNIVERSIDADE FEDERAL FLUMINENSE. Downloaded on June 02,2021 at 00:45:32 UTC from IEEE Xplore. Restrictions apply.

```
**Algorithm** $randomConflictFreeRanges(n)$ {
    $u = 0;\ v = 2^W - 1;$
    $Z = \phi;\ i = 0;$
    **while**$(i < n)$ {
        Use $PST1$ to list ranges $[x, y] \in R$ such that $x < u \leq y < v$ ($A$ is the set of such ranges);
        $InsertSetA = \{[u, y] \mid [x, y] \in A\};$
        Sort $InsertSetA$ according to the finish points of ranges;
        Use $PST2$ to list ranges $[x, y] \in R$ such that $u < x \leq v < y$ ($B$ is the set of such ranges);
        $InsertSetB = \{[x, v] \mid [x, y] \in B\};$
        Sort $InsertSetB$ according to the start points of ranges;
        Append $InsertSetA$ to $Z$; $i = i + |InsertSetA|;$
        Append $InsertSetB$ to $Z$; $i = i + |InsertSetB|;$
        Append $\{[u, v]\}$ to $Z$; $i = i + 1;$
        Generate a random range $[u, v]$;
    }
    return $Z$;
}
```

Fig. 13. Generate random conflict-free ranges.

priority search trees, an insert is expected to take more than twice the insert time for the case of prefixes. This expectation is born out in our experiments. Although the delete operation for nonintersecting ranges does about twice the work this operation does for prefixes, our measured delete times for nonintersecting ranges are more than twice that for prefixes. We believe this apparent anomaly is due to a disproportionate increase in the number of cache misses resulting from the fact that the nonintersecting-range data structure uses twice as much memory as used by the data structure for prefixes.

## 7.3 Conflict-Free Ranges

Fig. 13 gives the algorithm used by us to generate a random sequence $Z$ of $n$ conflict-free ranges. When the ranges in the sequence $Z$ are inserted in sequence order, every insert succeeds (the proof of this is omitted). The sequence $Z$ is used by us to measure insert times. For deletion, 33 percent of the ranges are removed in the reverse of the insert order.

Table 4 gives the memory required as well as the mean times and standard deviations for the case of conflict-free ranges.

It should be noted that the memory required by our data structure for conflict-free ranges is a little more than twice that required by the structure for nonintersecting ranges. Further, the lookup time is about 30 percent more than required by comparable-sized nonintersecting-range tables, the insert time for conflict-free ranges is about 2.5 times that for nonintersecting ranges, and the remove time for conflict-free ranges is about three times that for nonintersecting ranges. The insert and delete times for conflict-free ranges are, respectively, 6.7 and 7.5 times as much as they are for prefix tables represented as priority-search trees.

TABLE 4
Conflict-Free Ranges, PIII 700MHz with 256K L2 Cache

| Num of Ranges in R | | 30000 | 50000 | 80000 |
|---|---|---|---|---|
| Num of Ranges in norm(R) | Mean | 29688 | 48868 | 76472 |
| | Std | 18.03 | 42.90 | 60.05 |
| Memory Usage (KB) | Mean | 6240 | 9979 | 15219 |
| | Std | 7.06 | 10.91 | 11.19 |
| Search ($\mu$sec) | Mean | 1.98 | 2.34 | 2.69 |
| | Std | 0.07 | 0.09 | 0.06 |
| Insert ($\mu$sec) | Mean | 18.45 | 19.65 | 20.76 |
| | Std | 0.51 | 0.27 | 0.27 |
| Delete ($\mu$sec) | Mean | 19.3 | 20.49 | 21.60 |
| | Std | 0.41 | 0.13 | 0.29 |

## 8 Conclusion

We have developed data structures for dynamic router tables. Our data structures permit one to search, insert, and delete in $O(\log n)$ time each in the worst case. Although $O(\log n)$ time data structures for prefix tables were known prior to our work [1], [19], our data structure is more memory efficient than the data structures of [1], [19]. Further, our data structure is significantly superior on the insert and delete operations, while being competitive on the search operation.

Although the OVST structure of [10] isn't designed to support insert and delete operations, one may still compare the OVST and PST structures. The memory required by a 2OVST and a PST for the Paix2 database are about the same; however, a PST takes about twice the memory needed by a 3OVST. For the search operation, the PST is significantly slower than a 2OVST, taking about 3.2 times as much time. So, if one has a static IPv4 table (i.e., a table that doesn't permit inserts and deletes), the 2OVST is the way to go. The insert operation takes about 75 times as much time using a 2OVST as it does when a PST used and the delete operation takes about 300 times as much time. While these ratios will decrease as we go from a 2OVST to a 3OVST, a 4OVST, and so on, the OVST search time will correspondingly increase, making the PST more attractive in a high update environment (for example, dynamic multifield classification as used in firewalls).

With IPv6 databases, the memory required by a 2OVST and 3OVST could be quite prohibitive. We expect that the relative benefits of the PST structure would be enhanced when IPv6 is in use.

Authorized licensed use limited to: UNIVERSIDADE FEDERAL FLUMINENSE. Downloaded on June 02,2021 at 00:45:32 UTC from IEEE Xplore. Restrictions apply.

For nonintersecting ranges and conflict-free ranges, our data structures are the first to permit $O(\log n)$ search, insert, and delete.

Even though our data structures require the use of multiple trees, only one tree is needed for the lookup operation. The remaining trees are accessed only during an update. *The run times reported in this paper are for a 700MHz PC and are not indicative of the native speed of the proposed data structures on contemporary PCs dedicated to the packet classification task.* The structures are expected to be much faster on (say) a 3GHz PC with no background tasks running. *Further, much faster search and update times can be expected from a hardware implementation of the proposed structures and associated algorithms* for comparative purposes.

## ACKNOWLEDGMENTS

This work was supported, in part, by the US National Science Foundation under grant CCR-9912395.

## REFERENCES

[1] S. Sahni and K. Kim, "$O(\log n)$ Dynamic Packet Routing," *Proc. IEEE Symp. Computers and Comm.*, pp. 443-448, 2002.

[2] C. Macian and R. Finthammer, "An Evaluation of the Key Design Criteria to Achieve High Update Rates in Packet Classifiers," *IEEE Network*, vol. 15, no. 6, pp. 24-29, Nov./Dec. 2001.

[3] A. Hari, S. Suri, and G. Parulkar, "Detecting and Resolving Packet Filter Conflicts," *Proc. IEEE INFOCOM*, 2000.

[4] M. Ruiz-Sanchez, E. Biersack, and W. Dabbous, "Survey and Taxonomy of IP Address Lookup Algorithms," *IEEE Network*, vol. 15, no. 2, pp. 8-23, Mar./Apr. 2001.

[5] S. Sahni, K. Kim, and H. Lu, "Data Structures for One-Dimensional Packet Classification Using Most-Specific-Rule Matching," *Proc. Int'l Symp. Parallel Architectures, Algorithms, and Networks (ISPAN)*, May 2002.

[6] K. Sklower, "A Tree-Based Routing Table for Berkeley Unix," technical report, Univ. of California, Berkeley, 1993.

[7] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, "Small Forwarding Tables for Fast Routing Lookups," *Proc. ACM SIGCOMM*, pp. 3-14, 1997.

[8] W. Doeringer, G. Karjoth, and M. Nassehi, "Routing on Longest-Matching Prefixes," *IEEE/ACM Trans. Networking*, vol. 4, no. 1, pp. 86-97, 1996.

[9] S. Nilsson and G. Karlsson, "Fast Address Look-Up for Internet Routers," *IEEE Broadband Comm.*, 1998.

[10] V. Srinivasan and G. Varghese, "Faster IP Lookups Using Controlled Prefix Expansion," *ACM Trans. Computer Systems*, pp. 1-40, Feb. 1999.

[11] S. Sahni and K. Kim, "Efficient Construction of Fixed-Stride Multibit Tries for IP Lookup," *Proc. Eighth IEEE Workshop Future Trends of Distributed Computing Systems*, 2001.

[12] S. Sahni and K. Kim, "Efficient construction of Variable-Stride Multibit Tries for IP Lookup," *Proc. IEEE Symp. Applications and the Internet (SAINT)*, 2002.

[13] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable High Speed IP Routing Lookups," *Proc. ACM SIGCOMM*, pp. 25-36, 1997.

[14] M. Berg, M. Kreveld, and J. Snoeyink, "Two- and Three-Dimensional Point Location in Rectangular Subdivisions," *J. Algorithms*, vol. 18, no. 2, pp. 256-277, 1995.

[15] D.E. Willard, "Log-Logarithmic Worst-Case Range Queries Are Possible in Space $\theta(n)$ ," *Information Processing Letters*, vol. 17, pp. 81-84, 1983.

[16] P.V. Emde Boas, R. Kass, and E. Zijlstra, "Design and Implementation of an Efficient Priority Queue," *Math. Systems Theory*, vol. 10, pp. 99-127, 1977.

[17] B. Lampson, V. Srinivasan, and G. Varghese, "IP Lookup Using Multi-Way and Multicolumn Search," *Proc. IEEE INFOCOM*, 1998.

[18] S. Suri, G. Varghese, and P. Warkhede, "Multiway Range Trees: Scalable IP Lookup with Fast Updates," *Proc. GLOBECOM*, 2001.

[19] S. Sahni and K. Kim, "Efficient Dynamic Lookup for Bursty Access Patterns," http://www.cise.ufl.edu/~sahni, 2003.

[20] F. Ergun, S. Mittra, S. Sahinalp, J. Sharp, and R. Sinha, "A Dynamic Lookup Scheme for Bursty Access Patterns," *Proc. IEEE INFOCOM*, 2001.

[21] G. Cheung, S. McCanne, "Optimal Routing Table Design for IP Address Lookups under Memory Constraints," *Proc. IEEE INFOCOM*, 1999.

[22] A. McAuley and P. Francis, "Fast Routing Table Lookups Using Cams," *Proc. IEEE INFOCOM*, pp. 1382-1391, 1993.

[23] G. Chandranmenon and G. Varghese, "Trading Packet Headers for Packet Processing," *IEEE Trans. Networking*, 1996.

[24] P. Newman, G. Minshall, and L. Huston, "IP Switching and Gigabit Routers," *IEEE Comm. Magazine*, Jan. 1997.

[25] A. Bremler-Barr, Y. Afek, and S. Har-Peled, "Routing with a Clue," *Proc. ACM SIGCOMM*, pp. 203-214, 1999.

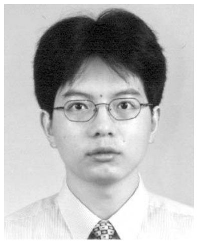
[26] P. Gupta and N. McKeown, "Dynamic Algorithms with Worst-Case Performance for Packet Classification," *IFIP Networking*, 2000.

[27] E. McCreight, "Priority Search Trees," *SIAM J. Computing*, vol. 14, no. 1, pp. 257-276, 1985.

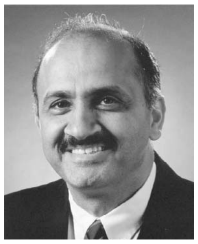
[28] Merit, "IPMA Statistics," http://nic.merit.edu/ipma, 2000, 2001.

[29] K. Melhorn, *Data Structures and Algorithms 3: Multi-Dimensional Searching and Computational Geometry*. New York: Springer Verlag, 1984.

[30] H. Lu and S. Sahni, "$O(\log n)$ Dynamic Router-Tables for Prefixes and Ranges," http://www.cise.ufl.edu/~sahni, 2003.



**Haibin Lu** received the BE and ME degrees in electronic engineering from Tsinghua University, Beijing, China, in 1997 and 1999 and the PhD degree in computer engineering from the University of Florida in 2003. He joined the faculty of the Department of Computer Science, University of Missouri-Columbia, as an assistant professor in 2003. His primary research focus lies in algorithmic aspects of computer network and multimedia communication.



**Sartaj Sahni** received the BTech (electrical engineering) degree from the Indian Institute of Technology, Kanpur, and the MS and PhD degrees in computer science from Cornell University. He is a distinguished professor and chair of the Computer and Information Sciences and Engineering Department at the University of Florida. He is also a member of the European Academy of Sciences, a fellow of the IEEE, ACM, AAAS, and Minnesota Supercomputer Institute, and a distinguished alumnus of the Indian Institute of Technology, Kanpur. In 1997, he was awarded the IEEE Computer Society Taylor L. Booth Education Award "for contributions to Computer Science and Engineering education in the areas of data structures, algorithms, and parallel algorithms" and, in 2003, he was awarded the IEEE Computer Society W. Wallace McDowell Award "for contributions to the theory of NP-hard and NP-complete problems." He was awarded the 2003 ACM Karl Karlstrom Outstanding Educator Award for "outstanding contributions to computing education through inspired teaching, development of courses and curricula for distance education, contributions to professional societies, and authoring significant textbooks in several areas including discrete mathematics, data structures, algorithms, and parallel and distributed computing." He has published more than 250 research papers and written 15 texts. His research publications are on the design and analysis of efficient algorithms, parallel computing, interconnection networks, design automation, and medical algorithms. He is a co-editor-in-chief of the *Journal of Parallel and Distributed Computing*, a managing editor of the *International Journal of Foundations of Computer Science*, and a member of the editorial boards of *Computer Systems: Science and Engineering*, *International Journal of High Performance Computing and Networking*, *International Journal of Distributed Sensor Networks*, and *Parallel Processing Letters*. He has served as program committee chair, general chair, and been a keynote speaker at many conferences. Dr. Sahni has served on several US National Science Foundation and US National Institutes of Health panels and he has been involved as an external evaluator of several computer science and engineering departments.

Authorized licensed use limited to: UNIVERSIDADE FEDERAL FLUMINENSE. Downloaded on June 02,2021 at 00:45:32 UTC from IEEE Xplore. Restrictions apply.