$O(\log n)$ Dynamic Router-Tables For Prefixes And Ranges [*]

Haibin Lu & Sartaj Sahni
{halu, sahni}@cise.ufl.edu
Department of Computer and Information Science and Engineering
University of Florida, Gainesville, FL 32611

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 also are presented.

Keywords: 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.

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

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 requires $O(n)$ 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, 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 non-conflicting ranges (Definition 5)is considered in Section 6. In a non-conflicting 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 non-conflicting 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 [6–12]. Structures such as that of [6] perform each of the dynamic router-table operations (lookup, insert, delete) in $O(W)$ time. Others (e.g., [7–12]) 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'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'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. The number of cache misses is $O(\log n)$ for each operation. When W bits fit in $O(1)$ words (as is the case for IPv4 and IPv6 prefixes) logical operations on W-bit vectors can be done in $O(1)$ time each. In this case, the scheme of [18] takes $O(\log W * \log n)$ time for an insert and $O(W + \log n) = O(W)$ time for a delete. 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 non-bursty 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.

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 [23–25].

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, $u \leq v$. *The range* r *represents the addresses* $\{u, u+1, ..., v\}$. $\boldsymbol{start(r)} = u$ *is the start point of the range and* $\boldsymbol{finish(r)} = v$ *is the finish point of the range. The range* r **covers** *or* **matches** *all addresses* d *such that* $u \leq d \leq v$. $\boldsymbol{isRange(q)}$ *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

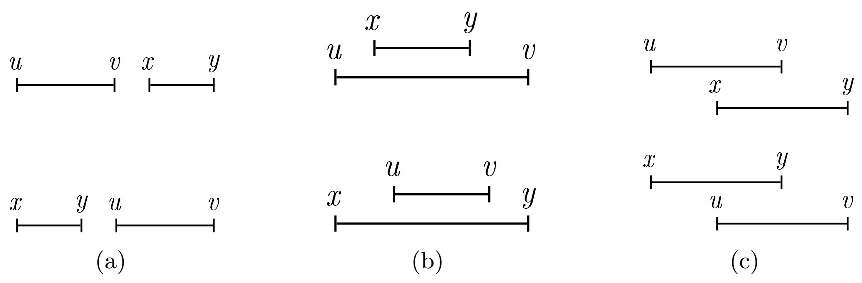


Figure 1: Relationships between pairs of ranges. (a). Disjoint. (b). Nested. (c). Intersect

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. Figure 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.*

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

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

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

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

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

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

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

[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* $\boldsymbol{ranges(d, R)}$ *(or simply $ranges(d)$ when R is implicit) as the subset of ranges of R that match/cover the destination address d. We define* $\boldsymbol{msr(d, R)}$ *(or $msr(d)$) as the most specific range of R that matches d. That is, $msr(d)$ is the most specific range in $ranges(d)$.* $\boldsymbol{msr([u, v], R)} = \boldsymbol{msr(u, v, R)} = r$ *iff $msr(d, R) = r$, $u \leq d \leq v$. 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* **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 Figure 2(c), $msr(4, 5) = [4, 14]$, $msr(6, 7) = [6, 14]$, $msr(8, 12) = [8, 12]$, $msr(13, 14) = [6, 14]$, 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* $\boldsymbol{isConflictFree(R)}$ *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.

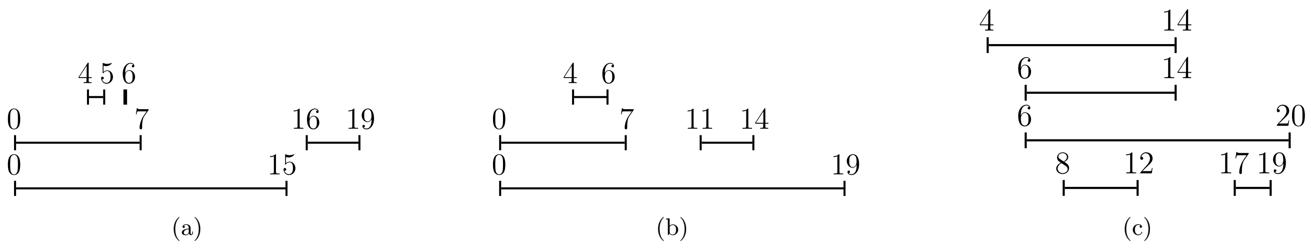


Figure 2: Sample set: (a). Prefixes ($W = 5$) {0*, 00*, 0010*, 00110, 100*}. (b). Non-intersecting ranges. Range [4, 6] can't be represented by single prefix, neither can [0, 19], [11, 14]. (c). Conflict-free ranges. Ranges [4, 14] and [6, 20] intersect.

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 $\boldsymbol{map1(r)} = (x, y) = (key1, key2) = (finish(r), start(r))$ in 2-dimensional space. Figure 3 shows a set of ranges and the equivalent set of 2-dimensional points (x, y).

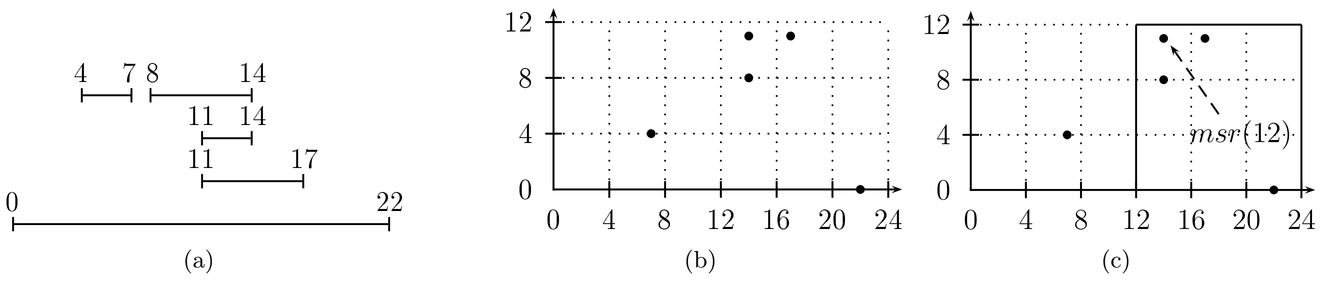


Figure 3: (a). An example range set R. (b). Its mapping $map1(R)$ into points in 2D. The x value is the range finish point and the y value is the range start point. (c). $ranges(12)$ includes all the points inside the solid rectangle since range $[st, fn]$ such that $st = y \leq 12 \leq fn = x$ matches 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 $msr(12)$.

McCreight [27] has observed that when the mapping of Equation 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

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

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 vs. 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 non-intersecting ranges, each insertion must verify that the new range doesn't intersect any of the existing ranges. This makes the case of non-intersecting ranges slightly harder to handle. Note however, that when a range is deleted from a set of non-intersecting ranges, the range set that remains is non-intersecting. 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, non-intersecting 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 $(x, y) \in P$.* $\boldsymbol{transform1(x,y)}$ $= (x', y') = (2^W x + (2^W - 1 - y), y)$ *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 $\boldsymbol{PST1(P)}$ 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(c)). 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 $minXinRectangle(2^W u + (2^W - 1 - (u - 1)), 2^W(v - 1) + (2^W - 1), u - 1)$ exists in PST1.

To verify $\exists s = [x, y] \in R[u < x \leq v < y]$, map the ranges of R into 2-dimensional points using the mapping, $\boldsymbol{map2(r)} = (start(r), 2^W - 1 - finish(r))$. 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, $\boldsymbol{PST2}$ of points in $transform2(map2(R))$, where $\boldsymbol{transform2(x, y)} = (2^W x + y, y)$ Hence, $\exists s = [x, y] \in R[u < x \leq v < y]$ iff $minXinRectangle(2^W(u+1), 2^W v+(2^W-1)-(v+1), (2^W-1)-(v+1))$ 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 are 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 of 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 of 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 $R = \{r_1, ..., r_n\}$ 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 Appendix. ■

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 $r = [u, v] \notin R$.*

$$isConflictFree(A) \Longleftrightarrow maxY(u, v, R) \leq maxP(u, v, R) \land minX(u, v, R) \geq minP(u, v, R)$$

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

Proof See Appendix. ■

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

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

$\boldsymbol{minP(u, v, R)} = \min\{start(\Pi(A)) | A \subseteq R \land isRange(\Pi(A)) \land finish(\Pi(A)) = v \land start(\Pi(A)) \geq u\}$ *is the minimum start point of a possible projection that is a range that finishes at v and starts by u.*

When $\nexists A \subseteq R[isRange(\Pi(A)) \land start(\Pi(A)) = u \land finish(\Pi(A)) \leq v]$, 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.

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

Figure 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

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

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.

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

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

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 Appendix. ■

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.

Figure 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 $PST1$.

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.

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

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.*

   (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.*

Figure 6(a) shows a range set that is not normalized (it contains ranges that intersect as well as nested ranges that have common end-points). Figure 6(b) shows a normalized range set. Regardless of which of these two range sets is used, every destination d has the same most-specific range after the

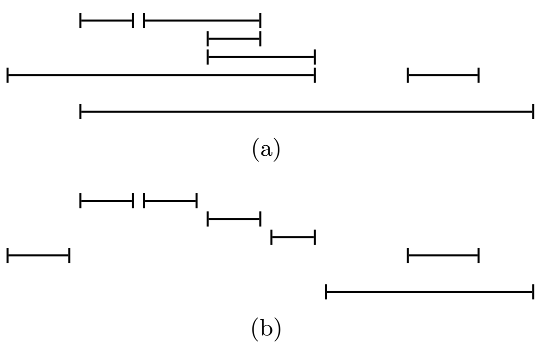


Figure 6: Unnormalized and normalized range sets

correspondence is established between ranges in the original range set and those in the normalized range set ($chop(msr(d, R)) = msr(d, norm(R))$, Lemma 21).

Definition 11 *An ordered sequence of ranges $(r_1, ..., r_n)$ 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 $CP(N) = \{C_1, ..., C_k\}$, $N = \cup_{i=1}^{k} C_i$. 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 Appendix. ■

Figure 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 an equivalent normalized range set $norm(R)$. By equivalent, we mean that for every destination d, the most-specific matching-range is the same in R as it is in $norm(R)$.

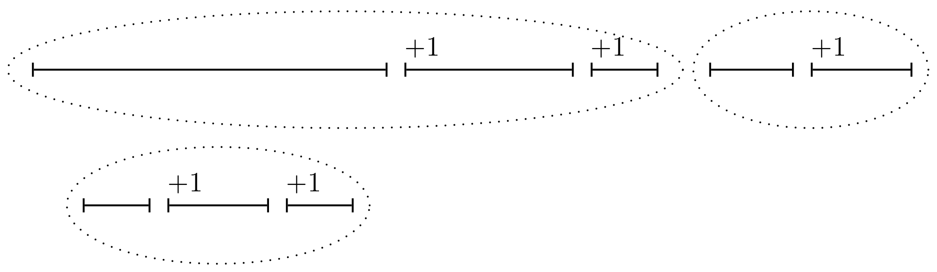


Figure 7: Partitioning a normalized range set into chains

Definition 12 [Chopping Rule] *Let $r = [u, v] \in R$, where R is a range set.* $\boldsymbol{chop(r, R)}$ *(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 $maxP(u, v-1, R) + 1 \le minP(u+1, v, R) - 1$, $chop(r) = [maxP(u, v-1, R) + 1, minP(u+1, v, R) - 1]$.*
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 $\boldsymbol{norm(R)} = \{chop(r) \mid r \in R \wedge chop(r) \neq \emptyset\}$.

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 Appendix. ■

For $r' \in norm(R)$, define $\boldsymbol{full(r')} = chop^{-1}(r') = r$, 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.

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

Figure 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*

Figure 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 *Figure 8 correctly computes* $maxP(u, v, R)$.

Proof See Appendix. ∎

6.6.2 An Efficient Algorithm to Compute *maxP*

The algorithm of Figure 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 $D = (t'_1, ..., t'_q)$ 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,

$$\begin{aligned} minStartLeft &= \min\{start(full(r'))|r' \in leftSubtree(p)\} \\ minStartRight &= \min\{start(full(r'))|r' \in rightSubtree(p)\} \\ maxFinishLeft &= \max\{finish(full(r'))|r' \in leftSubtree(p)\} \end{aligned}$$

$$maxFinishRight \quad = \quad \max\{finish(full(r'))|r' \in rightSubtree(p)\}$$

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'. Alternatively, we may use a PST, $PST3$, for the range set $chains = \{[start(C_i), finish(C_i)]|C_i \in CP(norm(R))\}$. The points in $PST3$ are $map1(chains)$; with each point in $PST3$, we keep a pointer to the root of the RBT for that chain. Note that since range end-points are distinct in $chains$, we do not need to use $transform1$ as used in $PST1$. To find an end point d, we first find the smallest chain that covers d by performing the operation $minXinRectangle(d, \infty, d)$ in $PST3$. Next, we follow the pointer associated with this chain to get to the corresponding RBT. Finally, a search of this RBT gets us to the RBT node for the s' with the given end point. In the sequel, we assume that $endPointsTree$, rather than $PST3$, is used. A parallel discussion is possible for the case when $PST3$ is used.

To implement Step 1 of Figure 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 Figure 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 Figure 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 downward pass in this subtree to locate s'. Figure 10 describes this downward pass.

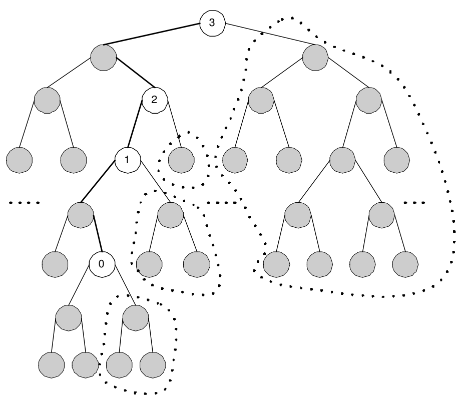


Figure 9: An example *RBT*

```
$downwardPass(currentNode)$
    // $currentNode$ 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
            $currentNode = currentNode.leftChild$;
    }
```

Figure 10: Find s' (and hence $maxP$) in a subtree known to contain s'

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

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 Appendix. ■

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 Appendix. ■

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 $chop(s, R - \{r\})$.

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 Appendix. ■

When $chop(r, R) = \emptyset$, no changes are to be made to the RBTs 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 RBTs (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 RBTs. 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 RBTs) and $chop(s, R - \{r\})$ inserted. Figure 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).

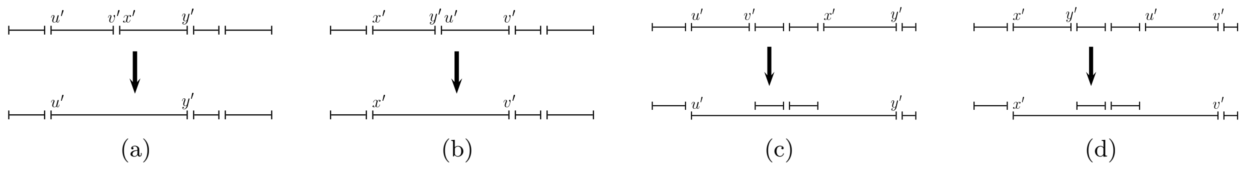


Figure 11: Cases when s' and r' are in the same chain of $CP(norm(R))$

The portions that deal with the RBTs 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 number of prefixes in each of these databases as well as the memory requirements for each database of prefixes using our data structure (PST) of Section 4 as well as the ACRBT structure of [19] are shown in Table 1. The databases Paix1, Pb1, MaeWest and Aads were obtained on Nov 22, 2001, while Pb2 and Paix2 were obtained Sept. 13, 2000. Figure 12 is a plot of the data of Table 1. As can be seen, 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% 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].

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

Table 1: Memory usage

For the database Paix2, the optimal height 2 variable stride trie that results when the controlled

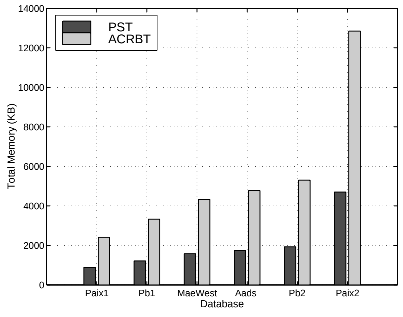


Figure 12: Memory usage

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]), 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 Sun Blade 100 Workstation that has a 500MHz UltraSPARC-Iie processor and has a 256KB L2 cache. The standard deviation in the mean time is also given in this table. On our Sun workstation, finding the longest matching-prefix takes about 10% to 14% less time using an ACRBT than a PST. Extrapolating from the times reported in [1, 19], we anticipate that a search in a PST takes about 3.2 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% of the prefixes into an initially empty data structure, measured the time to insert the remaining 33%, and computed the mean insert time by dividing by the number of prefixes in 33% 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 Sun workstation. As can be seen, insertions into a PST take between 18% and 22% the time to insert into an ACRBT!

Database			Paix1	Pb1	MaeWest	Aads	Pb2	Paix2
Search (μsec)	PST	Mean	2.88	3.06	3.25	3.31	3.43	4.06
		Std	0.36	0.18	0.17	0.16	0.09	0.05
	ACRBT	Mean	2.60	2.77	2.87	2.87	3.09	3.51
		Std	0.25	0.16	0.16	0.12	0.13	0.04
Insert (μsec)	PST	Mean	3.90	4.45	4.83	5.18	5.14	6.04
		Std	0.57	0.63	0.51	0.48	0.19	0.20
	ACRBT	Mean	21.15	23.42	24.77	25.36	25.54	28.07
		Std	1.11	0.66	0.38	0.29	0.19	0.18
Delete (μsec)	PST	Mean	4.36	4.45	4.73	4.71	5.06	5.48
		Std	0.91	0.63	0.53	0.00	0.19	0.16
	ACRBT	Mean	21.24	22.68	23.16	23.71	24.56	25.64
		Std	0.95	0.55	0.49	0.35	0.26	0.21

Table 2: Prefix times on a 500MHz Sun Blade 100 Workstation

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% of the prefixes in the database and measuring the time to delete a randomly selected 33% of these prefixes. Deletion from a PST takes about 20% the time required to delete from an ACRBT.

Extrapolating from the times reported in [1,19], 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!

Tables 3 and 4 give the corresponding times a PST takes about 3.2 times as much time as a search in an on a 700MHz Pentium III PC and a 1.4GHz Pentium 4 PC, respectively. Both computers have a 256KB L2 cache. The run times on our 700MHz Pentium III are about one-half the times on our Sun workstation. Surprisingly, when going from the 700MHz Pentium III to the 1.4GHz Pentium 4, the measured time to find the longest matching-prefix decreased by only about 5% for PST. More surprisingly, the corresponding times for ACRBT actually increased. The net result of the slight decrease in time for PST and the increase for ACRBT is that, on our Pentium 4 PC, the PST is faster than the ACRBT on all three operations–find longest matching-prefix, insert, and delete. This somewhat surprising behavior is due to architectural differences (e.g., differences in width and size of L1 cache lines) between the Pentium III and 4 processors.

Figures 13, 14 and 15 histogram the search, insert, and delete time data of the preceding tables.

Database			Paix1	Pb1	MaeWest	Aads	Pb2	Paix2
Search (μsec)	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 (μsec)	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 (μsec)	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

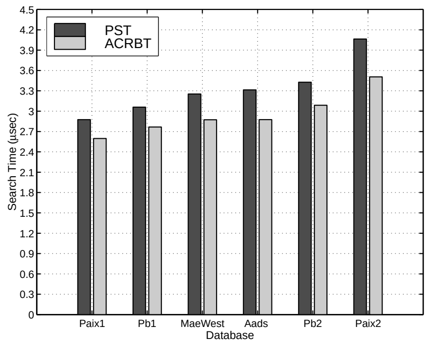
Table 3: Prefix times on a 700MHz Pentium III PC

Database			Paix1	Pb1	MaeWest	Aads	Pb2	Paix2
Search (μsec)	PST	Mean	1.30	1.44	1.51	1.52	1.63	1.92
		Std	0.19	0.18	0.17	0.13	0.13	0.06
	ACRBT	Mean	1.48	1.69	1.83	1.87	1.87	2.24
		Std	0.31	0.20	0.16	0.07	0.14	0.05
Insert (μsec)	PST	Mean	1.76	1.96	2.18	2.17	2.38	2.65
		Std	0.41	0.69	0.00	0.44	0.35	0.18
	ACRBT	Mean	11.22	11.81	12.41	12.91	12.92	13.94
		Std	0.41	0.60	0.41	0.44	0.26	0.18
Delete (μsec)	PST	Mean	1.76	1.69	1.92	1.93	2.00	2.22
		Std	0.41	0.60	0.38	0.21	0.42	0.17
	ACRBT	Mean	9.46	10.39	10.54	10.42	10.92	11.64
		Std	0.57	0.63	0.38	0.21	0.42	0.16

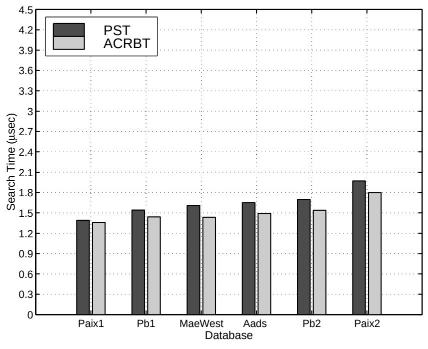
Table 4: Prefix times on a 1.4GHz Pentium 4 PC

7.2 Nonintersecting Ranges

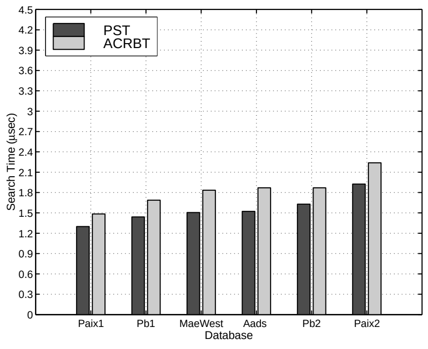
Sine no two prefixes may intersect, we may use prefix databases to benchmark our data structure (Section 5) for nonintersecting ranges. The search time for our six IPv4 prefix databases is same using the data structure for nonintersecting ranges as it is when the data structure for prefixes is used. However, the memory requirement is doubled since we now have two priority search trees, $PST1$ and $PST2$. Table 5 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 modelled 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 priority search trees, an insert is expected to take more than twice the insert time for the case of prefixes. This expectation is



(a)

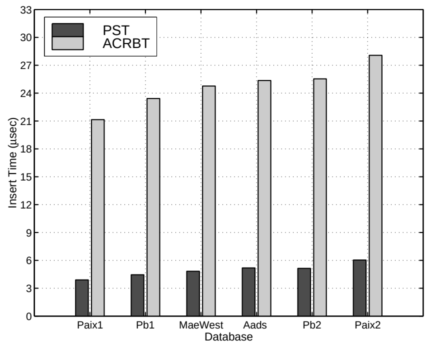


(b)

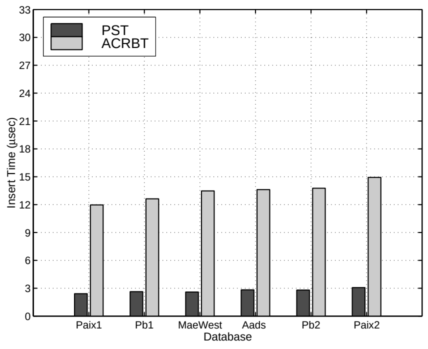


(c)

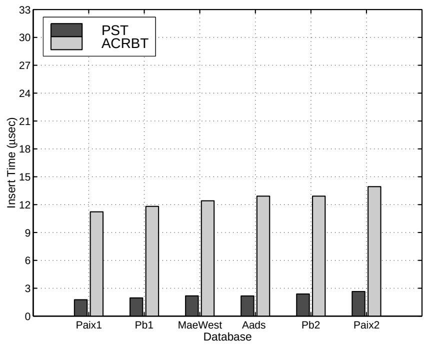
Figure 13: Longest matching-prefix (a) Sun, (b) 700MHz, (c) 1.4GHz



(a)

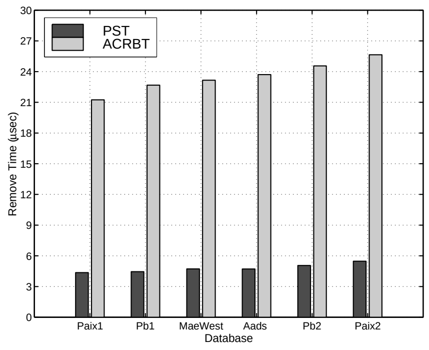


(b)

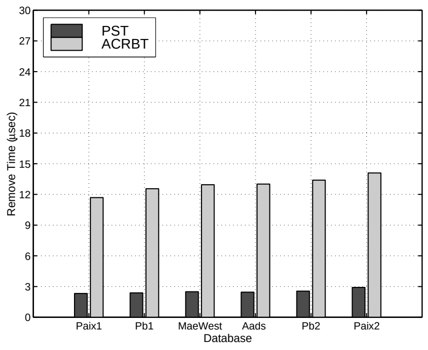


(c)

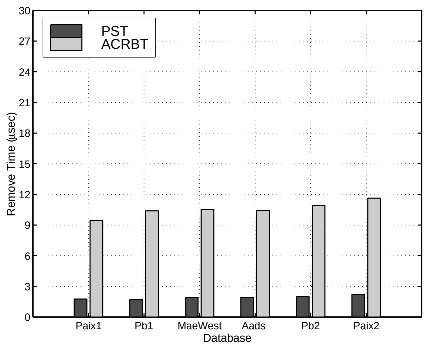
Figure 14: Insert a prefix (a) Sun, (b) 700MHz, (c) 1.4GHz



(a)



(b)



(c)

Figure 15: Delete a prefix (a) Sun, (b) 700MHz, (c) 1.4GHz

borne 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

```
**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 $Z$;
$\}$
```

Figure 16: Generate random conflict-free ranges

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.

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

Table 5: Nonintersecting Ranges. PIII 700MHz with 256K L2 cache

7.3 Conflict-free Ranges

Figure 16 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% of the ranges are removed in the reverse of the insert order.

Table 6 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 non-intersecting ranges. Further, the lookup time is about 30% more than required by comparable-sized nonintersecting-range tables; the insert time for conflict-free ranges is about 2.5 times that for non-intersecting 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.

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 (μsec)	Mean	1.98	2.34	2.69
	Std	0.07	0.09	0.06
Insert (μsec)	Mean	18.45	19.65	20.76
	Std	0.51	0.27	0.27
Delete (μsec)	Mean	19.3	20.49	21.60
	Std	0.41	0.13	0.29

Table 6: Conflict-free Ranges. PIII 700MHz with 256K L2 cache

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 multi-field 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.

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.

References

[1] S. Sahni and K. Kim. $o(\log n)$ dynamic packet routing. In *IEEE Symposium on Computers and Communications*, pages 443–448, 2002.

[2] C. Macian and R. Finthammer. An evaluation fo the key design criteria to achieve high update rates in packet classifiers. *IEEE Network*, pages 24–29, Nov./Dec. 2001.

[3] A. Hari, S. Suri, and G. Parulkar. Detecting and resolving packet filter conflicts. In *IEEE INFOCOM*, 2000.

[4] M. Ruiz-Sanchez, E. Biersack, and W. Dabbous. Survey and taxonomy of ip address lookup algorithms. *IEEE Network*, 15(2):8–23, March/April 2001.

[5] S. Sahni, K. Kim, and H. Lu. Data structures for one-dimensional packet classification using most-specific-rule matching. In *International Symposium on Parallel Architectures, Algorithms, and Networks (ISPAN)*, May 2002.

[6] K. Sklower. A tree-based routing table for berkeley unix. Technical report, University of California - Berkeley, 1993.

[7] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink. Small forwarding tables for fast routing lookups. In *ACM SIGCOMM*, pages 3–14, 1997.

[8] W. Doeringer, G. Karjoth, and M. Nassehi. Routing on longest-matching prefixes. *IEEE/ACM Transactions on Networking*, 4(1):86–97, 1996.

[9] S. Nilsson and G. Karlsson. Fast address look-up for internet routers. *IEEE Broadband Communications*, 1998.

[10] V. Srinivasan and G. Varghese. Faster ip lookups using controlled prefix expansion. *ACM Transactions on Computer Systems*, pages 1–40, Feb 1999.

[11] S. Sahni and K. Kim. Efficient construction of fixed-stride multibit tries for ip lookup. In *Proceedings 8th IEEE Workshop on Future Trends of Distributed Computing Systems*, 2001.

[12] S. Sahni and K. Kim. Efficient construction of variable-stride multibit tries for ip lookup. In *Proceedings IEEE Symposium on Applications and the Internet (SAINT)*, 2002.

[13] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner. Scalable high speed ip routing lookups. In *ACM SIGCOMM*, pages 25–36, 1997.

[14] M. Berg, M. Kreveld, and J. Snoeyink. Two- and three-dimensional point location in rectangular subdivisions. *Journal of Algorithms*, 18(2):256–277, 1995.

[15] D. E. Willard. Log-logarithmic worst-case range queries are possible in space $\theta(n)$. *Information Processing Letters*, 17:81–84, 1983.

[16] P.V. Emde Boas, R. Kass, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10:99–127, 1977.

[17] B. Lampson, V. Srinivasan, and G. Varghese. Ip lookup using multi-way and multicolumn search. In *IEEE INFOCOM*, 1998.

[18] S. Suri, G. Varghese, and P. Warkhede. Multiway range trees: Scalable ip lookup with fast updates. In *GLOBECOM*, 2001.

[19] S. Sahni and K. Kim. Efficient dynamic lookup for bursty access patterns. In *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. In *IEEE INFOCOM*, 2001.

[21] G. Cheung and S. McCanne. Optimal routing table design for ip address lookups under memory constraints. In *IEEE INFOCOM*, 1999.

[22] A. McAuley and P. Francis. Fast routing table lookups using cams. In *IEEE INFOCOM*, pages 1382–1391, 1993.

[23] G. Chandranmenon and G. Varghese. Trading packet headers for packet processing. *IEEE Transactions on Networking*, 1996.

[24] P. Newman, G. Minshall, and L. Huston. Ip switching and gigabit routers. *IEEE Communications Magazine*, Jan. 1997.

[25] A. Bremler-Barr, Y. Afek, and S. Har-Peled. Routing with a clue. In *ACM SIGCOMM*, pages 203–214, 1999.

[26] P. Gupta and N. McKeown. Dynamic algorithms with worst-case performance for packet classification. In *IFIP Networking*, 2000.

[27] E. McCreight. Priority search trees. *SIAM Jr. on Computing*, 14(1):257–276, 1985.

[28] Merit. Ipma statistics. In *http://nic.merit.edu/ipma*, 2000, 2001.

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

APPENDIX

Proof of Lemma 3. **Proof** It is sufficient to prove that a range set R is not conflict free iff it has at least one pair of ranges that are in conflict.

From Definition 5 it follows that if the range set R is not conflict free, there is a destination address d for which $ranges(d) \neq \emptyset \wedge msr(d) = \emptyset$. Since $ranges(d) \neq \emptyset$, there is at least one range that contains d. Let $r \in R$ be the smallest range that contains d (r may not be $msr(d)$). Because $msr(d) = \emptyset$, there exists another range $s \in R$ such that $d \in s$ and $isIntersect(r, s)$ is true. Since $r \in R$ is the smallest range that contains d, there is no range $t \in R$ such that $d \in t \wedge t \subseteq r \cap s$. Therefore, there is no resolving subset for r and s. The ranges r and s are in conflict.

If there is a pair of ranges r and s that are in conflict, r and s intersect and have no resolving subset. Either there is no subset $Q \subset R$ such that $\Pi(Q) = r \cap s$, or Q is not conflict-free for all $Q \subset R \wedge$

$\Pi(Q) = r \cap s$. For the former case, let $Q = \{t | t \in R \wedge t \subseteq r \cap s\}$. Since $\Pi(Q) \neq r \cap s$, there is a destination address d such that $d \in r \cap s \wedge d \notin \Pi(Q)$. Therefore, $ranges(d) \neq \emptyset \wedge msr(d) = \emptyset$. For the latter case, choose the largest Q, i.e., $\{t | t \in R \wedge t \subseteq r \cap s\}$. Since Q is not conflict free, from Definition 5, it follows that there is a destination address d for which $ranges(d, Q) \neq \emptyset \wedge msr(d, Q) = \emptyset$. Since $msr(d, Q) = msr(d, R)$, $msr(d, R) = \emptyset$. Since $ranges(d, Q) \subseteq ranges(d, R)$, $ranges(d, R) \neq \emptyset$. ■

Lemma 11 *Let R be a conflict-free range set. Let r be an arbitrary range. Let A be the subset of R that comprises all ranges of R that are contained in r. A is conflict free.*

Proof Since R is conflict free, every pair (s, t) of intersecting ranges in A has a resolving subset B in R. From Definition 8, it follows that every range in B is contained in $s \cap t$. Hence, $B \subset A$. Therefore, every pair of intersecting ranges of A has a resolving subset in A. So, A is conflict free. ■

Lemma 12(1) tells us that we can always find a subset $B \subseteq R$ such that $A \cup B$ is conflict-free and $\Pi(A) = \Pi(A \cup B)$. $r = \Pi(A)$ may not be in R, but $R \cup \{r\}$ is conflict free (Lemma 12(3)) because whenever a range s intersects a range r, there is a resolve subset in R for r and s (Lemma 12(2)).

Lemma 12 *Let R be a conflict-free range set. Let A, $A \subseteq R$ be such that $\Pi(A) = r = [u, v]$.*

1. $\exists B \subseteq R[isConflictFree(A \cup B) \wedge \Pi(A) = \Pi(A \cup B)]$
2. *Let $s \in R$ be such that $isIntersect(r, s)$.* $\exists B \subseteq R[\Pi(B) = r \cap s]$
3. $R \cup \{r\}$ *is conflict free.*

Proof

1. Follows from Lemma 11, i.e., let $A \cup B$ comprise all ranges of R that are contained in $[u, v]$.
2. When $r \in R$, (2) follows from the definition of a conflict-free range set. So, assume $r \notin R$. Let C comprise all ranges of A contained in s. If s intersects no range of A, $\Pi(C) = r \cap s$. If s intersects at least one range of A, then let $t \in A$ be an intersecting range with maximum overlap. Since R is conflict free, $\exists D \subseteq R[\Pi(D) = t \cap s]$. We see that $\Pi(C \cup D) = r \cap s$.
3. From parts (1) and (2) of this lemma, it follows that there is a resolving subset in $R \cup \{r\}$ for every $s \in R$ that intersects with r. Hence, $R \cup \{r\}$ is conflict free.

■

Proof of Lemma 4. **Proof** ($\Longrightarrow$) Assume that A is conflict free. When neither $maxY$ nor $minX$ exist (this happens iff no range of R intersects $r = [u, v]$), $maxY \leq maxP \wedge minX \geq minP$. When, $maxY$ exists, $s = [x, maxY] \in A \wedge x < u \leq maxY < v$. (Note that $isIntersect(r, s)$). Since A is conflict free, A has a (resolving) subset B for which $\Pi(B) = r \cap s = [u, maxY]$. Therefore, $maxY \leq maxP$. Similarly, when $minX$ exists, $minX \geq minP$.

($\Longleftarrow$) Assume that $maxY(u, v, R) \leq maxP(u, v, R) \wedge minX(u, v, R) \geq minP(u, v, R)$. When neither $maxY$ nor $minX$ exist, no range of R intersects r. When, $maxY$ exists, $\exists s = [x, y] \in A[x < u \leq y < v]$. Consider any such $s = [x, y]$. Since $maxY \leq maxP$ and $maxY$ exists, $maxP$ exists. Hence, from Lemma 11, $\exists B \subseteq R[isConflictFree(B) \wedge \Pi(B) = [u, maxP]]$. When $y = maxP$, B is a resolving subset for s and r in A. When $y < maxP$, $isIntersect(s, [u, maxP])$. Since $R \cup \{[u, maxP]\}$ is conflict free (Lemma 12(3)), $R \cup \{[u, maxP]\}$ (and so also R and A) has a resolving subset for s and $[u, maxP]$. This resolving subset is also a resolving subset for s and r. When $minX$ exists, $\exists s = [x, y] \in A[u < x \leq v < y]$. In a manner analogous to the proof for the case $maxY$ exists, we may show that A has a resolving subset for r and each such s. Hence, in all cases, intersecting ranges of A have a resolving subset. So, A is conflict free. ■

Lemma 13, which is used in the proof of Lemma 5, states when A has no subset whose projection equals r and when no range in A contains r, A has no subset whose projection contains r.

Lemma 13 *Let R be a conflict-free range set. Let $A = R - \{r\}$ for some $r \in R$.*

$$((\nexists B \subseteq A[\Pi(B) = r]) \wedge (\nexists s \in A[r \subseteq s])) \Longrightarrow (\nexists C \subseteq A[r \subseteq \Pi(C)])$$

Proof Assume

$$\nexists B \subseteq A[\Pi(B) = r] \tag{2}$$

and

$$\nexists s \in A[r \subseteq s] \tag{3}$$

We need to show that $\nexists C \subseteq A[r \subseteq \Pi(C)]$.

Suppose that there is a C such that $C \subseteq A \wedge r \subseteq \Pi(C)$. From $C \subseteq A$ and Equation 3, it follows that

$$\forall t \in C[isDisjoint(r, t) \vee isIntersect(r, t) \vee t \subset r] \tag{4}$$

If $\nexists t \in C[isIntersect(r,t)]$, then from Equation 4, we get $\forall t \in C[isDisjoint(r,t) \vee t \subset r]$. From this and $r \subseteq \Pi(C)$, it follows that all destination addresses d, $d \in r$, are covered by ranges of C that are contained in r. Therefore, $\exists B \subseteq C \subseteq A[\Pi(B) = r]$. This contradicts Equation 2.

Next, suppose $\exists t \in C[isIntersect(r,t)]$. Let D be the union of the resolving subsets for all of these t and r in R. Clearly, all ranges in D are contained in r. Further, let E be the subset of all ranges in C that are contained in r. Since every point in r is either in E or in a range that intersects r, $((D \cup E) \subseteq A) \wedge (\Pi(D \cup E) = r)$. This contradicts Equation 2. ■

Proof of Lemma 5. **Proof** For (1), we note that by replacing r by B in every resolving subset for intersecting ranges in R, we get resolving subsets that do not include r. Hence all of these resolving subsets are present in A. So, A is conflict free.

For (2), assume that $\nexists B \subseteq A[\Pi(B) = r]$.

($\Longrightarrow$) Assume that A is conflict free. We need to prove

$$\nexists s \in A[r \subset s] \vee [m,n] \in A \tag{5}$$

We do this by contradiction. So, assume

$$\exists s \in A[r \subset s] \wedge [m,n] \notin A \tag{6}$$

Since $\exists s \in A[r \subset s]$, m and n are well defined. Equation 6 implies that A has a range $[m,y]$, $y > n$ as well as a range $[x,n]$, $x < m$. Further, $isIntersect([m,y],[x,n])$ and $r \subseteq [m,y] \cap [x,n] = [m,n]$. Let B be the subset of R comprised of all ranges contained in $[m,n]$. From Lemma 11, it follows that B is conflict free. However, no subset of $C = B - \{r\}$ has projection equal to r. Further, from the definitions of m and n, no range of C can contain r. From Lemma 13, it follows that no subset of C has a projection that contains r. In particular, C has no subset whose projection is $[m,n]$. Therefore, A, has no subset whose projection is $[m,n]$. So, A has no resolving subset for $[m,y]$ and $[x,n]$. Therefore, A is not conflict free, a contradiction.

($\Longleftarrow$) If no range of A contains r, then r is not part of the resolving subset for any pair of intersecting ranges of R. This, together with the fact that R is conflict free, implies that A is conflict free. If $[m,n] \in A$, we can use $[m,n]$ in place of r in any resolving subset for intersecting ranges of R. Therefore, A has a resolving subset for every pair of intersecting ranges. So, A is conflict free. ■

Lemma 14 *Let N be a normalized range set.*

$$A \subseteq N \wedge \Pi(A) = [u,v] \Longrightarrow \exists B \subseteq N[isChain(B) \wedge \Pi(B) = [u,v]]$$

Proof Let B be the subset of A obtained by removing from A all ranges that are nested within at least one other range of A. Clearly, $\Pi(B) = \Pi(A) = [u, v]$. Since N is normalized and $B \subseteq N$, B is also normalized. From Definition 10 and the fact that B has no pair of nested ranges, it follows that all ranges of B are disjoint. For disjoint ranges to have a projection that is a range, the disjoint ranges must form a chain. ■

Proof of Lemma 6. **Proof**

1. Let A be the subset of N obtained by removing from N all ranges that are nested within at least one other range of N. From the proof of Lemma 14, A is normalized and all ranges of A are disjoint. Clearly, A can be uniquely partitioned into a set of longest chains $CP(A)$. Since every range in $N - A$ is nested within at least one other range of A and no two ranges in N have a common end-point, no range in $N - A$ can be assigned to any chain in $CP(A)$ to form a different chain. Therefore, $CP(A) \subseteq CP(N)$. Repeating the above procedure for $N - A$, we get $CP(B) \subseteq CP(N)$, where B is the subset of $N - A$ obtained by removing from $N - A$ all ranges that are nested within at least one other range of $N - A$. Clearly, $CP(B) \cap CP(A) = \emptyset$. Repeating the same procedure, we get $CP(C)$, $CP(D)$ and so on until there is no range left to partition. It is easy to see that $CP(N) = CP(A) \cup CP(B) \cup CP(C) \cup ...$ is a unique partitioning of N into a set of longest chains.

2. Direct consequence of the definition of a normalized set and that of a chain.

■

Lemma 15 establishes a relationship between s and $chop(r)$ when s is contained in r.

Lemma 15 *Let R be a conflict-free range set.*

$$\forall r \in R\ \forall s \in R \quad [s \subset r \Longrightarrow$$
$$(\ s \subset chop(r) \wedge start(s) \neq start(chop(r)) \wedge finish(s) \neq finish(chop(r))\ )$$
$$\vee\ isDisjoint(s, chop(r))]$$

Proof The lemma is trivially true when $chop(r) = \emptyset$ ($isDisjoint(s, \emptyset)$ is true). So, assume that $chop(r) = r'$. For the lemma to be false, either $isIntersect(s, r')$ or ($r' \subseteq s$ or s and r' have a common end point).

If $isIntersect(s, r')$, then either $start(r') < start(s) \leq finish(r') < finish(s)$ or $start(s) < start(r') \leq finish(s) < finish(r')$. Assume the former (the latter case is similar). From the chopping rule, it follows that $\exists A \subseteq R[\Pi(A) = [finish(r') + 1, finish(r)]]$. Therefore, $A \cup \{s\} \subseteq R \wedge \Pi(A \cup \{s\}) =$

$[start(s), finish(r)]$. From this, $start(r) \le start(r') < start(s)$, and the chopping rule, we get $finish(chop(r)) < start(s)$. But, $start(s) \le finish(r')$, a contradiction.

So, $r' \subseteq s$ or s and r' have a common end point. First consider the case $r' \subseteq s \subset r$. Suppose that $start(s) \ne start(r)$ (the case $finish(s) \ne finish(r)$ is similar). Since $r' = chop(r)$, $\exists A \subseteq R[\Pi(A) = [finish(r') + 1, finish(r)]]$. Therefore, $\Pi(A \cup \{s\}) = [start(s), finish(r)]$ and $start(r) < start(s) \le start(r')$. From the chopping rule, it follows that $finish(chop(r)) < start(s) \le start(r') \le finish(r')$, a contradiction. Therefore, $s \subset r'$. If $start(s) = start(r')$, $maxP(start(r), finish(r) - 1) \ge finish(s)$. So, $start(r') > finish(s)$, which contradicts $s \subset r'$. The case $finish(s) = finish(r')$ is similar. ■

Lemma 16 establishes a relationship among $chop(r)$, $chop(s)$, and $r \cap s$ when r and s intersect.

Lemma 16 *Let r and s be two intersecting ranges of a conflict-free range set R.*

$$isDisjoint(chop(r), r \cap s) \wedge isDisjoint(chop(s), r \cap s) \wedge isDisjoint(chop(r), chop(s))$$

Proof Without loss of generality, we may assume that $start(r) < start(s) \le finish(r) < finish(s)$. Since R is conflict free, $\exists A[A \subset R \wedge \Pi(A) = r \cap s]$. Therefore, $finish(chop(r)) < start(s)$ and $start(chop(s)) > finish(r)$. This proves the lemma. ■

Proof of Lemma 7. **Proof** Let r' be any range in $norm(R)$. Clearly, for every $r' \in norm(R)$, there is at least one $r \in R$ such that $chop(r) = r'$. Suppose two different ranges r and s of R have $r' = chop(r) = chop(s)$.

If $isIntersect(r, s)$, then from Lemma 16 we get $isDisjoint(chop(r), chop(s))$. So, $chop(r) \ne chop(s)$.

If $isNested(r, s)$, then from Lemma 15 it follows that $s \subset chop(r) \vee isDisjoint(s, chop(r))$ when $s \subset r$ and $r \subset chop(s) \vee isDisjoint(r, chop(s))$ when $r \subset s$. Consider the former case (the latter case is similar). $s \subset chop(r)$ implies $chop(s) \ne chop(r)$. $isDisjoint(s, chop(r))$ also implies $chop(s) \ne chop(r)$.

The final case is $isDisjoint(r, s)$. In this case, clearly, $chop(s) \ne chop(r)$. ■

Lemma 17 *For every conflict-free range set R, $norm(R)$ is a normalized conflict-free range set.*

Proof We shall show that $norm(R)$ is normalized. Since a normalized range set has no intersecting ranges, every normalized range set is conflict free.

If $|norm(R)| \le 1$, $norm(R)$ is normalized. So, assume that $|norm(R)| > 1$. Let r' and s' be two different ranges in $norm(R)$. We need to show that r' and s' satisfy property 2(a) or 2(b) of Definition 10. Let $r = [u, v] = full(r')$ and $s = full(s')$. There are three possible cases for r and s, they either intersect, are nested, or are disjoint (Lemma 1).

Case 1: $isIntersect(r, s)$. From Lemma 16, it follows that r' and s' are disjoint.

Case 2: $isNested(r, s)$. Either $s \subset r$ or $r \subset s$. Assume the former (the latter case is similar). From Lemma 15, we get

$$[s \subset chop(r) \wedge start(s) \neq start(chop(r)) \wedge finish(s) \neq finish(chop(r))] \vee isDisjoint(s, chop(r))$$

$s \subset chop(r) \wedge start(s) \neq start(chop(r)) \wedge finish(s) \neq finish(chop(r))$ implies that s' and r' are nested and do not have a common end-point. $isDisjoint(s, chop(r))$ implies that s' and r' are disjoint.

Case 3. $isDisjoint(r, s)$. Clearly, $isDisjoint(r', s')$. ■

Lemma 18 tells us that $full(r')$ is the $msr(r', R)$ if there is no $s' \in norm(R)$ that is nested inside r'.

Lemma 18 *Let $r' \in norm(R)$, where R is a conflict-free range set.*

$$\nexists s' \in norm(R)[s' \subset r'] \Longrightarrow r = full(r') = msr(r', R)$$

Proof Assume that $\nexists s' \in norm(R)[s' \subset r']$. If $\exists d \in r'[r \neq msr(d, R)]$, then $\exists s \subset r[d \in s]$. From Lemma 15, it follows that $s \subset r' \vee s \cap r' = \emptyset$. Since $d \in s \wedge d \in r'$, $s \cap r' \neq \emptyset$. Hence, $s \subset r'$. From Lemma 11, it follows that $A = \{t | t \in R \wedge t \subseteq s\}$ is conflict free. Since $s \in A$, $A \neq \emptyset$. From the chopping rule it follows that $norm(A) \neq \emptyset$. For any $t' \in norm(A) \subset norm(R)$, $t' \subseteq full(t') \subseteq s \subset r'$. This violates the assumption of this lemma. Therefore, $\nexists d \in r'[r \neq msr(d, R)]$. So, $r = msr(r', R)$. ■

Lemma 19 *Let R be a conflict-free range set, let x be the start point of some range in R, and let y be the finish point of some range in R.*

1. *Let $s \in R$ be such that $start(s) = x$ and $finish(s) = \min\{finish(t) | t \in R \wedge start(t) = x\}$*
   - *(a) $chop(s) \neq \emptyset$.*
   - *(b) $start(chop(s)) = x$.*
   - *(c) $chop(s)$ is the only range in $norm(R)$ that starts at x.*
2. *Let $s \in R$ be such that $finish(s) = y$ and $start(s) = \max\{start(t) | t \in R \wedge finish(t) = y\}$*
   - *(a) $chop(s) \neq \emptyset$.*
   - *(b) $finish(chop(s)) = y$.*
   - *(c) $chop(s)$ is the only range in $norm(R)$ that finishes at y.*

Proof We prove 1(a) - (c). 2(a) - (c) are similar. Since $maxP(start(s), finish(s) - 1, R)$ does not exist, case 5 of the chopping rule does not apply and $chop(s) \neq \emptyset$. One of the cases 1 and 3 applies. In both of these cases, $start(chop(s)) = x$. For 1(c), we note that $norm(R)$ is normalized (Lemma 17) and the definition of a normalized set (Definition 10) implies that no two ranges of $norm(R)$ share an end point. In particular, $norm(R)$ can have only one range that has x as an end point. ■

Lemma 20 *Let $r' \in norm(R)$, where R is a conflict-free range set.*

$$start(full(r')) \neq start(r') \Longrightarrow \nexists s \in R[start(s) = start(r')]$$

Proof Suppose that $start(full(r')) \neq start(r')$ and $\exists s \in R[start(s) = start(r')]$. From Lemma 19(1a and 1b), it follows that $\exists t \in R[start(t) = start(r') \wedge chop(t) \neq \emptyset \wedge start(chop(t)) = start(r')]$. Therefore, $norm(R)$ has at least two ranges (r' and $chop(t)$) that start at $start(r')$. This contradicts Lemma 19(1c). ■

Definition 13 *Let $r' \in norm(R)$. If $start(r') = start(full(r'))$, $start(r')$ is a* **real** *starting point. Otherwise, $start(r')$ is a* **fake** *start point.*

Lemma 21 *Let R be a conflict-free range set. Let $r \in R$ be such that $r = msr(u, v, R)$ for some range $[u, v]$. $r' = chop(r) = msr(u, v, norm(R))$.*

Proof From the definition of msr, it follows that there is no $s \in R$ such that $s \subset r \wedge s \cap [u, v] \neq \emptyset$. Therefore, $maxP(start(r), finish(r) - 1, R) < u$ if $maxP(start(r), finish(r) - 1, R)$ exists, and $minP(start(r)+1, finish(r), R) > v$ if $minP(start(r)+1, finish(r), R)$ exists. From the Chopping Rule, $[u, v] \subseteq chop(r)$. Further, from Lemmas 15 and 16, it follows that $norm(R)$ contains no $s' \subset chop(r)$. So, $r' = msr(u, v, norm(R))$. ■

Lemma 22 *Let R be a conflict-free range set that has a subset whose projection equals $[x, y]$. Let $A \subseteq R$ comprise all $r \in R$ such that $r \subseteq [x, y]$.*

1. $\exists B \subseteq norm(R)[\Pi(B) = [x, y]]$
2. $C = \{full(r') | r' \in norm(R) \wedge r' \subseteq [x, y]\} \subseteq A$

Proof

1. From Lemma 11, it follows that A is conflict free. Further, since R has a subset whose projection equals $[x, y]$, $\Pi(A) = [x, y]$. From Lemma 21, it follows that every $d \in [x, y]$ has a most-specific range in $norm(A)$. Therefore, $\Pi(norm(A)) = [x, y]$. From the definition of the chopping rule and that of A, we see that $\forall r \in A[chop(r, A) = chop(r, R)]$. So, $norm(A) \subseteq norm(R)$.

2. First, assume that $[x, y] \in R$. Suppose there is a range $r' \in norm(R)$ such that $r' \subseteq [x, y]$ and $r = full(r') \notin A$. There are three cases for r.

   Case 1: $isDisjoint(r, [x, y])$. In this case, $isDisjoint(r', [x, y])$ and so $r' \not\subseteq [x, y]$.

   Case 2: $isIntersect(r, [x, y])$. From Lemma 16, we get $isDisjoint(chop(r), [x, y])$. So $r' \not\subseteq [x, y]$.

   Case 3: $[x, y] \subset r$. From Lemma 15 and $r' \subseteq [x, y]$, we get $isDisjoint([x, y], chop(r))$. So $r' \not\subseteq [x, y]$.

   When $[x, y] \notin R$, let $R' = R \cup \{[x, y]\}$, $C' = \{full(r') | r' \in norm(R') \wedge r' \subseteq [x, y]\}$ and $A' = A \cup \{[x, y]\}$. Using the lemma case we have already proved, we get $C' \subseteq A'$. Since $chop([x, y], R') = \emptyset$ and $chop(s, R) = chop(s, R')$ for every $s \in R$, $norm(R') = norm(R)$. Therefore, $C = C'$. So, $C \subseteq A'$. Finally, since $[x, y] \notin C$, $C \subseteq A$.

■

Proof of Theorem 1. **Proof** First consider Step 1. From Lemma 19(1b), it follows that

$$\nexists r' \in norm(R)[start(r') = u] \Longrightarrow \nexists r \in R[start(r) = u]$$

Therefore, $\nexists r' \in norm(R)[start(r') = u] \Longrightarrow \nexists maxP$. From Lemma 20, it follows that $start(full(r')) \neq start(r') = u \Longrightarrow \nexists s \in R[start(s) = start(r') = u]$. So, $start(full(r')) \neq u \Longrightarrow \nexists maxP$. Finally, $u = start(r') = start(full(r))$ implies $finish(full(r')) = \min\{finish(t) | t \in R \wedge start(t) = u\}$ (Lemma 19(1)). So, $finish(full(r')) > v$ implies $\nexists s \in R[start(s) = u \wedge finish(s) \leq v]$. Hence, $start(r') = u \wedge finish(full(r')) > v \Longrightarrow \nexists maxP$. Further, when $\exists r' \in norm(R)[start(r') = u \wedge finish(full(r')) \leq v]$, $maxP$ exists and $maxP \geq finish(full(r')) \geq finish(r')$. Therefore, Step 1 correctly identifies the case when $maxP$ doesn't exist.

We get to Step 2 only when $maxP$ exists. From the definition of $maxP$, $\exists A \subseteq R[\Pi(A) = [u, maxP]]$. From this and Lemma 22(1), we get $\exists B \subseteq norm(R)[\Pi(B) = [u, maxP]]$. Now, from Lemma 14, we get $\exists D \subseteq norm(R)[isChain(D) \wedge \Pi(D) = [u, maxP]]$. From Lemma 6, it follows that D is a sub-chain of the unique chain $C_i \in CP(norm(R))$ that includes r'. Let r', s'_1, s'_2, ..., s'_q be the tail of C_i. It follows that $maxP$ is either $finish(r')$ or $finish(s'_j)$ for some j in the range $[1, q]$. Let j be the least integer such that $full(s'_j) \not\subseteq [u, v]$. If such a j does not exist, then $maxP = finish(s'_q)$ as $norm(R)$ has no subset whose projection equals $[u, x]$ for any $x > finish(s'_q)$. So, assume that j exists. From Lemma 22(2), it follows that $maxP < finish(s'_j)$. Hence, Step 2 correctly determines $maxP$. ■

Proof of Lemma 8. **Proof** For (1), note that $chop(r, R \cup \{r\}) = \emptyset \implies \exists A \subseteq R[\Pi(A) = r]$. Therefore, the addition of r to R does not affect any of the $maxP$ and $minP$ values.

For (2a), suppose there are two different ranges g and h in R such that $chop(g, R) \neq chop(g, R \cup \{r\})$ and $chop(h, R) \neq chop(h, R \cup \{r\})$. From the chopping rule, it follows that

$$r \subset g \wedge r \subset h \tag{7}$$

Therefore, $\neg isDisjoint(g, h)$. From this and Lemma 1, we get $isIntersect(g, h) \vee isNested(g, h)$. Equation 7 and $isIntersect(g, h)$ imply $r \subseteq g \cap h$. From this and Lemma 16, we get $isDisjoint(r, chop(g, R)) \wedge isDisjoint(r, chop(h, R))$. Therefore, $chop(g, R) = chop(g, R \cup \{r\})$ and $chop(h, R) = chop(h, R \cup \{r\})$, a contradiction. So, $\neg isIntersect(g, h)$.

If $isNested(g, h)$, we may assume, without loss of generality, that $g \subset h$. This and Equation 7 yield $r \subset g \subset h$. Therefore, $maxP(x, y - 1, R) = maxP(x, y - 1, R \cup \{r\})$ and $minP(x + 1, y, R) = minP(x + 1, y, R \cup \{r\})$, where $h = [x, y]$. So, $chop(h, R) = chop(h, R \cup \{r\})$, a contradiction.

Hence, there can be at most one range of R whose $chop()$ value changes as a result of the addition of r. The preceding proof for the case $isNested(g, h)$ also establishes that the $chop()$ value may change only for the range s, that is for the smallest enclosing range of r (i.e., smallest $s \in R[r \subset s]$).

For (2b), assume that $chop(s, R) \neq chop(s, R \cup \{r\})$. This implies that $chop(s, R) \neq \emptyset$ and so x' and y' are well defined. (Note that from part (1), we get $chop(r, R) \neq \emptyset$.) We consider each of the three cases for the relationship between r and $chop(s, R)$ (Lemma 1).

Case 1: $isDisjoint(r, chop(s, R))$. This case cannot arise, because then $chop(s, R) = chop(s, R \cup \{r\})$.

Case 2: $isIntersect(r, chop(s, R))$. Now, either $x' < u \leq y' < v$ or $u < x' \leq v < y'$. Consider the former case. Since $r \subset s = [x, y]$, $v \leq y$. $minP(x + 1, y, R) = y' + 1$ since $chop(s, R) = [x', y']$. When $v = y$, $minP(u + 1, v, R) = minP(x + 1, y, R)$ since $u \leq y'$. Since $start(r) = u$, r cannot contribute to $minP(u + 1, v, R \cup \{r\})$. So $minP(u + 1, v, R \cup \{r\}) = minP(u + 1, v, R)$. Therefore, when $v = y$, $minP(u + 1, v, R \cup \{r\}) = y' + 1$. So, $v' = y'$. Therefore, $x' < u \wedge y' = v'$. Consider the case $v < y$. From the chopping rule, it follows that $\exists A \subseteq R \subset R \cup \{r\}[\Pi(A) = [y' + 1, y]]$. From this, Lemma 12(2), and the fact that $R \cup \{r\}$ is conflict free, we conclude $\exists B \in R \cup \{r\}[\Pi(B) = r \cap [y' + 1, y] = [y' + 1, v]]$. From this and $minP(x + 1, y, R) = y' + 1$, we get $minP(u + 1, v, R \cup \{r\}) = y' + 1$. So, $v' = y'$. Once again, $x' < u \wedge y' = v'$. Using a similar argument, we may show that when $u < x' \leq v < y'$, $x' = u' \wedge y' > v$.

Case 3: $isNested(r, chop(s, R))$. So, either $r \subseteq chop(s, R)$ or $chop(s, R) \subset r$. First, consider all possibilities for $r \subseteq chop(s, R)$. The case $x' < u \leq v < y'$ cannot arise, because this implies $chop(s, R) = chop(s, R \cup \{r\})$. When $x' = u \leq v < y'$, $u' = x'$ since $maxP(u, v-1, R)$ does not exist (otherwise $maxP(x, y-1, R)$, if it exists, would be larger than $x' - 1$). So, $x' = u' \wedge y' > v$. When $x' < u \leq v = y'$, $v' = y'$ since $minP(u+1, v, R)$ does not exist (otherwise $minP(x+1, y, R)$, if it exists, would be smaller than $y' + 1$). So, $x' < u \wedge y' = v'$. The final case is when $x' = u \leq v = y'$. Now, $u' = x' \wedge y' = v'$.

Using an argument similar to that used in part (2a), we may show that when $chop(s, R) \subset r$, $x' = u' \wedge y' = v'$. ■

Proof of Lemma 9. **Proof** (1) follows from the proof of Lemma 8(2b). For (2), from the proof cases of Lemma 8(2b) that have $x' = u' \wedge y' = v'$, it follows that case 5 of the chopping rule applies for s in $R \cup \{r\}$. So, $chop(s, R \cup \{r\}) = \emptyset$.

For (3), $finish(chop(s, R \cup \{r\})) = y'$ follows from the proof of Lemma 8(2b). Also, we observe that $maxP(x, y-1, R \cup \{r\}) \geq v$. So, (3b) can be false only when $maxP(x, y-1, R \cup \{r\}) > v$ and either (a) $maxP(v'+1, y', R)$ doesn't exist or (b) $maxP(v'+1, y', R) < maxP(x, y-1, R \cup \{r\})$. For (a), $\exists [c, d] \in R[x \leq c \leq v' \wedge v < d < y']$. For (b), $\exists [c, d] \in R[x \leq c \leq v' \wedge v < maxP(v'+1, y', R) < d < y']$. In both cases, $c \leq u$ implies that $r = [u, v] \subset [c, d] \subset s$. This contradicts the assumption that s is the smallest enclosing range of r. Also, in both cases, $c > u$ implies $isIntersect(r, [c, d])$. So, $R \cup \{r\}$ has a subset whose projection is $[c, v]$. Therefore, $finish(chop(u, v, R \cup \{r\})) < c \leq v'$, a contradiction.

The proof for (4) is similar to that for (3). ■

Proof of Lemma 10. **Proof** For (1), note that $chop(r, R) = \emptyset \implies \exists A \subseteq R - \{r\}[\Pi(A) = r]$. Therefore, the removal of r from R does not affect any of the $maxP$ and $minP$ values.

For (2a) note that by substituting $R - \{r\}$ for R in Lemma 8(2a), we get $\forall t \in R - \{r, s\}[chop(t, R - \{r\}) = chop(t, R)]$. (2b) and (2c) follow from Lemma 9. ■