doi:10.3969/j.issn.1000-436x.2013.06.004

# 基于前缀区间集合的 IPv6 路由查找算法

崔宇，田志宏，张宏莉，方滨兴
(哈尔滨工业大学 网络与信息安全技术研究中心，黑龙江 哈尔滨 150001)

**摘 要**：对 IPv6 相关的通用型与特定型路由算法进行了分析，重点研究了以 BSR 为基础的 IPv6 路由算法在查找和更新时的不平衡问题，提出了基于前缀区间集合的 IPv6 路由算法。通过对路由前缀（*N*）进行范围（*K*）、集合（*M*）划分以及更新节点自修复提高查询速度、降低不平衡性的影响，具有 $O(\log 2N/K)$和 $O(\log 2N/K+2M)$的查询与更新时间复杂度，空间复杂度为 $O(K+2N)$。实验表明，该算法具有良好的查询性能，降低了更新不平衡性的影响。
**关键词**：路由；IPv6；前缀区间集合；自修复
**中图分类号**：TP393.08 **文献标识码**：A **文章编号**：1000-436X(2013)06-0029-09

# Binary search on range of IPv6 prefix sets

CUI Yu, TIAN Zhi-hong, ZHANG Hong-li, FANG Bin-xing
(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)

**Abstract:** The general and special types of IPv6 routing algorithms were studied. Focusing on the imbalance problems in lookup and update process in routing algorithm, a novel method called BSRPS (binary search on range of prefix sets) was presented. By range-partition (*K*) and set-partition (*M*) on routing table (*N*), and self-recovery after updating, this method enhanced the lookup speed and reduced the impact of imbalance in updating. Time complexity in lookup and update process is $O(\log 2N/K)$ and $O(\log 2N/K+2M)$, and space complexity is $O(K+2N)$ where *N* is the size of routing table. Experiment results show that this method is efficient in lookup and reduces the impact of imbalance after updating effectively.
**Key words:** route; IPv6; range of prefix sets; self-recovery

## 1 引言

近年来，IPv4 地址日渐枯竭，IPv6 普及过程逐步加快，IPv6 相关技术和服务支持得到了快速的发展，导致 IPv6 分配的地址范围和网络流量迅猛增加，因此需要更加高效的 IPv6 路由算法为数据分组的快速转发提供支撑。

与 IPv4 相同，IPv6 路由方式也为最长前缀匹配，因此可继承 IPv4 的部分路由算法，但在算法性能方面，两者有以下区别：1) IPv6 地址长度 128 bit，IPv4 为 32 bit，使得按位比较的相关算法（如 Binary Trie、Multi-bit）的查找树深度增大，造成最坏时间、空间复杂度大量增加；2) IPv4 路由规模基本稳定而 IPv6 处于发展阶段，路由条数始终保持增长趋势，根据文献[1]对 BGP 路由表的统计，路由条数从 2004 年的 500 条增加到当前约 8 000 条，规模和分布存在较大不确定性，这对以前缀分布特点为基础的相关算法影响较大，同时，由于更新相对频繁，以前缀值为基础的相关算法形成的查找树中不平衡现象会加强，影响查找和更新效率；3) IPv4 和 IPv6 前缀长度体现了一定的集中性，目前，IPv6 前缀绝大部分为 32 和 48，而文献[2]也预测未来前缀长度

**收稿日期**：2012-08-02；**修回日期**：2012-12-19
**基金项目**：国家重点基础研究发展计划（“973”计划）基金资助项目（2011CB302605）；国家高技术研究发展计划（“863”计划）基金资助项目（2011AA010705, 2012AA012506, 2012AA012502）；国家自然科学基金资助项目（61202457）；国家科技支撑计划基金资助项目（2012BAH37B01）
**Foundation Items:** The National Basic Research Program of China (973 Program) (2011CB302605); The National High Technology Research and Development Program of China (863 Program) (2011AA010705, 2012AA012506, 2012AA012502); The National Natural Science Foundation of China (61202457); The National Key Technology R&D Program of China (2012BAH37B01)

集中在32、47、48和64，这与IPv4分布相异，导致长度分布敏感的算法性能下降；4) IPv6前缀分布较为集中，从文献[1]的数据可知，高16 bit为0x2001的前缀占30.5%，分布极大不平衡，这对前缀位敏感的算法不利，比如若干集中分布的前缀会出现在Patricia[3]树的同一子树下，增加局部查找深度，这主要是前期地址分配不平衡和地域发展不均匀所致，而随着IPv6的普及，不平衡现象将会减少。

上述分析表明，目前，IPv6前缀存在分布集中和更新频繁2个主要特点，因此良好的IPv6路由算法不仅在理论上应具有较好的平均时间和空间复杂度，还应在集中或稀疏区域具有较好的局部性能，同时也适应路由前缀更新频繁的特点。目前，IPv6使用的软路由算法可分为2类：其一是以IPv4为基础的通用型算法；其二是针对IPv6特点实现的特定型路由算法。

以IPv4为基础的通用型算法主要分为基于前缀位、前缀长度和前缀值3种。

基于前缀位的路由查找算法主要以二进制树（binary trie）为基础，具有$O(W)$相关的查找与更新复杂度（$W$为地址长度）。该方法存在3个问题，其一是树中存在大量不包含前缀的空节点，浪费大量内存；其二是空节点的存在增加了大量无用的比较次数；其三是可能导致已经找到最长匹配前缀却继续向叶子节点查找的情况，增加无效比较的次数。为了去除树中空节点，Patricia方法对二进制树中的连续空节点进行了压缩，解决了前2个问题。文献[4]中的多分枝Trie树(multibits trie)通过将$k$ bit整合到一个节点中，实现了$O(W/k)$的查找复杂度，降低了查找树的深度。文献[5]提出的优先树(priority trie)对二进制树结构进行了翻转，可优先处理长度较长的前缀，同时去掉了树中的空节点，部分解决了二进制树的3个问题，查找和更新时间复杂度介于[$O(\log N)$, $O(W)$]之间（$N$为路由数目），是近些年出现的较新颖的算法。

Waldvogel在文献[6]中提出了基于前缀长度的二分查找方法。该方法具有$O(\log W)$的查找复杂度，但需要大量的初始化计算和复杂的更新过程，同时也会产生大量的Marker，增加计算复杂度和内存空间。文献[7]通过复杂算法降低了Marker数量，进而降低整个算法的内存占用。该类方法以散列为基础，必然会产生散列冲突，影响算法效率和实际效果。

基于值的路由策略主要有二分查找树(BST)和按前缀范围进行二分查找(BSR)2种。在BST中，前缀按值顺序排列，递归选取子序列的中间位置进行比较，直到子序列前缀数为1停止，查找的时间复杂度为$O(\log N)$。BST方法不存在空节点，因此最大程度地节省了空间，具有$O(N)$的空间复杂度。但BST方法因前缀层次问题，在构建时会出现不平衡现象，增加了查找深度。文献[8]提出的DPT算法通过对前缀进行leaf-pushing扩展形成了平衡二叉树，从而避免不平衡现象。文献[9]以最内层节点为树节点，同时用数组维护前缀的包含关系，提出了只使用不相交前缀进行二分查找的方法，避免初始条件下不平衡的发生。

BSR[10]方式用[$s$, $f$]表示一个前缀，其中，$s$表示前缀起始值，补0扩展，$f$表示结束值，补1扩展，两者差值表示前缀的范围。BSR将点集进行排序，形成有序点集$\{s_1, f_1, s_2, s_3, f_3, f_2, \cdots\}$并对其进行初始化计算以存储每个节点对应的最优前缀(BMP, best match prefix)，对其进行二分查找，时间复杂度为$O(\log 2N)$。BSR算法由于用2个点表示前缀，因此最多需要前缀数量二倍的节点。同时，由于需要序列进行重新计算，因此更新的时间复杂度最坏为$O(2N)$。为了提高该方法的更新复杂度，文献[11]提出了Multi-Range-Tree方法将查找和更新的时间复杂度提高到$O(\log_k 2N)$，但存在频繁更新导致不平衡的可能，同时空间复杂度增加到了$O(kN\log_k 2N)$。文献[12]也提出了查找和更新复杂度为$O(\log N)$的算法，通过维护$N$+1棵树来维护前缀间、前缀与前缀间隔的包含关系，实现快速地更新，但实现难度较大。与文献[11]类似，两者的核心均是保存前缀的包含关系。文献[13]提出的RBMRTs方法将前缀进行分层，不被其他前缀包含的前缀是第一层前缀，只包含于某个第一层前缀A的前缀形成A的第二层前缀，以此类推。同层前缀之间不存在包含关系，因此可以减少更新时的计算量，实现了约为$O(\log 2N)$的查找和更新复杂度。

针对IPv6特点实现的特定路由算法通过对路由表分布特性的分析，利用各种通用算法的特点，将多种通用算法组合成新算法。如文献[14]提出的TSB方法就是将二叉树、段表和路由桶技术结合的IPv6路由算法。其利用IPv6前缀分布特点形成的第一层二叉树结构，只需要较少节点，同时结合段表和桶路由技术，可较好地提高算法性能并降低内存占用。该算法第二层路由桶和段表结构切换时需

要进行数据结构的整体切换，会产生一定延时。TSB 是多种算法组合，会在特定局部含有某算法的缺陷，但整体上，TSB 组合各算法的优点降低单一算法在特定条件下的缺点。随着 IPv6 发展，TSB 第一层二叉树节点数目会不断增加，降低查找性能，但这种增加不会在短时间内频繁进行。

由于 IPv6 地址长度较大，且分布集中，因此理论上基于前缀值的具有 $O(\log N)$时间复杂度的相关路由算法比 $O(W)$相关的前缀位算法有更好的性能。所以，本文对 BSR 相关路由算法进行了研究，分析其查找和更新过程中存在的问题，结合 IPv6 前缀特点，提出了结合段表和 BSR 2 种通用算法并以 RBMRTs 为基础的基于前缀区间集合的 IPv6 路由算法 BSRPS (binary search on range of prefix set)。

## 2　前缀值相关算法中查询与更新的不平衡性

BSR 相关路由算法在 IPv4 中具有较好的性能，但在 IPv6 环境中存在的前缀分布与更新不平衡性会严重影响该类方法的性能。本节对该类算法存在的 2 种不平衡性进行举例和理论分析，指出其在 IPv6 环境中存在的严重问题，提出了解决该问题的基本思路。

### 2.1　查询不平衡性

BSR 中，前缀按值排序形成$\{s_1, f_1, s_2, f_2, \cdots, s_m, f_m\}$前缀值序列（假设不存在包含关系），查找时通过二分查找确定位置，进而与某一路由条目进行对应。表 1 为一前缀实例，地址长度为 5($W$=5)，前缀 $P_1$~$P_7$ 集中分布于地址空间[2, 15]上，$P_8$ 相对孤立。以这种前缀集合形成的 BSR 和 Patricia 的基本形式如图 1 和图 2 所示。

**表 1　　前缀实例**

| 节点名 | 前缀 | 长度 | 范围 |
|---|---|---|---|
| $P_1$ | 0001 | 4 | [2, 3] |
| $P_2$ | 0010 | 4 | [4, 5] |
| $P_3$ | 0011 | 4 | [6, 7] |
| $P_4$ | 0100 | 4 | [8, 9] |
| $P_5$ | 0101 | 4 | [10, 11] |
| $P_6$ | 0110 | 4 | [12, 13] |
| $P_7$ | 0111 | 4 | [14, 15] |
| $P_8$ | 11 | 2 | [24, 31] |

可以看到，BSR 方法中，集中区域的前缀具有较浅的深度，而 Patricia 相反，集中区域的前缀深度较大。如前缀 $P_5$，在前者中深度为 3，后者为 5。反之，对于处于稀疏区域的前缀，如 $P_8$，BSR 中深度为 4，而 Patricia 中为 2，差别较大。

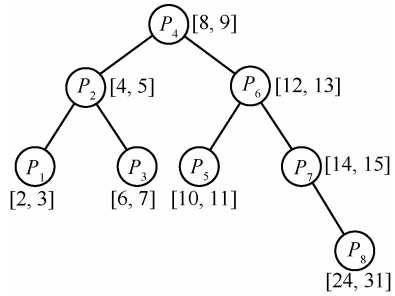



图 1　BSR 查找树

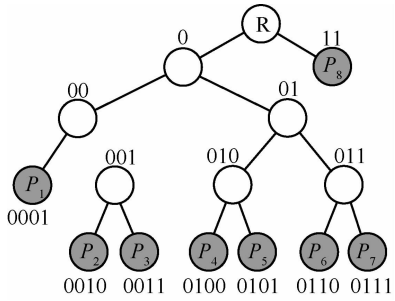



图 2　Patricia 查找树

理论上，考虑一种极限情况，假设在某一集中区域中存在拥有共同前缀 $a$ 的 $2^{m-1}$ 个连续等长前缀，其范围为$\{[a\times 2^m + 0, a\times 2^m + 1], [a\times 2^m + 2, a\times 2^m + 3], \cdots, [a\times 2^m + 2^m - 2, a\times 2^m + 2^m - 1]\}$，以及 1 个前缀为 $b$($b$!=$a$ 且与 $a$ 等长)的前缀 $Px$。则在 BSR 中，$Px$ 必然处于查找树的最左端或最右端，因此其查找深度为 $\log(1+2^{m-1})$，约为 $m-1$；而在 Patricia 中由于 $b$!=$a$，$Px$ 的查找深度为 2。对于连续的 $2^{m-1}$ 个前缀，查找深度分别为 $m-1$ 和 $1+m$，差别较小。可见，基于前缀区间的相关查找算法对集中区域的前缀有较好的查找性能，而基于前缀位的相关算法(如 Patricia、Multi-bit)对稀疏区域前缀的查找性能较好。

不同于 IPv4，IPv6 前缀分布的不均匀会严重增加 BSR 算法的不平衡性。首先，目前的 IPv6 中含有大量 0x2001 起始的前缀，分布集中，这对稀疏区域前缀的查找影响较大；其次，稀疏区域中存在的一些特殊前缀，如 2002::/16（6to4 隧道）约占整体流量 40%，会严重影响路由的整体性能。因此，为了避免这种情况，本文通过结合前缀位路由算法的思想，对前缀进行一次范围划分以减轻稀疏区域前缀查询的不平衡性，将在第 3 节中详细介绍。

### 2.2　更新不平衡性

BSR 相关路由算法在不断更新后可能会产生极端不平衡情况，比如图 1 中删除前缀 $P_8$[24, 31]并逐次插入 $P_8$[16, 17]、$P_9$[18, 19]、…、$P_{15}$[30, 31]后形成的查找树如图 3 所示。

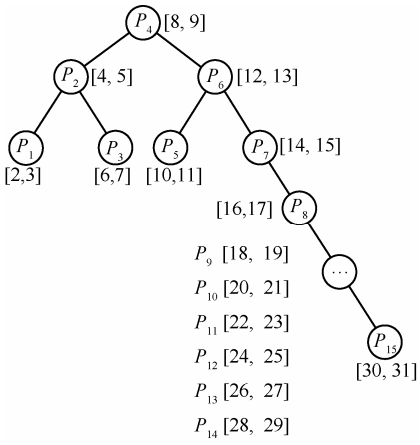



图 3　更新不平衡性举例

从图 3 可见，按照 BSR 基本更新方法插入前缀 $P_8$~$P_{15}$ 会产生众多单分支节点。若不进行平衡旋转，则必然会增加查找深度。在最坏情况下，整个查找树会出现每个节点均为单分支的情况，查询时间复杂度降为 $O(N)$。

由于 IPv6 前缀更新较为频繁，因此查找树的形态会对 BSR 的查询效率产生极大影响，必须有效控制最坏情况的出现，降低其对查询效率产生的影响。为此，本文借鉴了 Muliti-bit 方法中对若干前缀位进行合并的思想，将前缀区间进行第二次划分，形成前缀区间集合，达到有效地降低查找树深度的目的。并且，在更新后配合节点自修复算法以避免大量空置集合的出现，降低内存占用量。

## 3　基于前缀区间集合的 IPv6 路由查找算法

BSRPS 算法以 RBMRTs 为基础，针对 IPv6 前缀长度、分布范围等特点，通过地址范围划分和前缀集合划分对 RBMRTs 基础结构进行改造，构造 BSRPS 树，进行查找与更新。更新时，对更新过程中涉及到的节点使用自修复算法以避免前缀集合划分引入的空位，降低不断更新对查找树平衡性的影响。本节首先介绍 BSRPS 算法经二次划分后形成的多棵多层查找树的结构，其次分别介绍查找和更新算法。

### 3.1　BSRPS 算法划分方法与框架

BSRPS 以 RBMRTs 为基础，后者将前缀进行分层，形成单棵、多层二分查找树。图 4 显示了表 1 和表 2 所示前缀形成的查找树。

**表 2**　　**前缀补充实例**

| 节点名 | 前缀 | 长度 | 范围 |
|---|---|---|---|
| $P_9$ | 00001 | 5 | [1, 1] |
| $P_{10}$ | 00010 | 5 | [2, 2] |
| $P_{11}$ | 00011 | 5 | [3, 3] |
| $P_{12}$ | 1100 | 4 | [26, 27] |
| $P_{13}$ | 1101 | 4 | [24, 25] |
| $P_{14}$ | 111 | 3 | [28, 31] |
| $P_{15}$ | 11101 | 5 | [29, 29] |

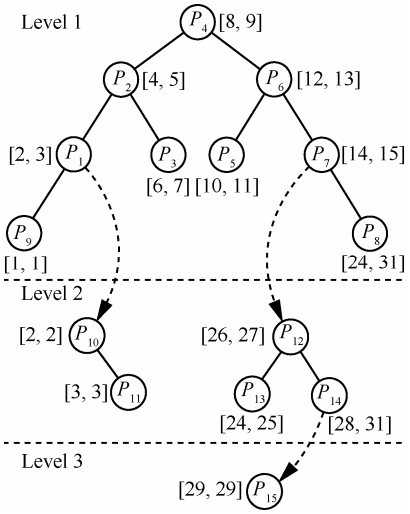



图 4　RBMRTs 算法框架

图 4 中，RBMRTs 将前缀集合分成若干层，同层前缀不存在包含关系，层间前缀为包含关系，称包含于某一前缀的若干前缀形成的树为该前缀的包含子树，如 $P_{10}$、$P_{11}$ 形成了 $P_1$ 的包含子树。该结构有以下 2 点优势：1) 同层无包含关系可省去 BSR 中一部分预计算和更新时间；2) 查询和更新算法也相对简单。查询时，首先在 Level1 中进行查询，如果不存在则返回空，若前缀节点存在包含子树则进入下一层查询，否则返回该前缀。更新过程与查询类似，但增加了层间移动、替换等操作。

BSRPS 设计中，首先用地址范围划分将 Level1 进行分割，形成多棵第一层树，如图 4 中若按地址范围四等分可形成{$P_9$, $P_1$, $P_2$, $P_3$}、{$P_4$, $P_5$, $P_6$, $P_7$}和{$P_8$} 3 棵第一层查找树。之后将每棵查找树的若

干节点组合为一个众节点，形成 BSRPS 查找树。

### 3.1.1　地址范围划分

RBMRTs 算法基于 BSR，因此对集中区域前缀的查询能力较好。为了提高稀疏区域前缀性能，需通过地址范围划分分离稀疏区的 IPv6 前缀，避免其查询时间被平均化，提高对稀疏区前缀的查询效率。为此，本文采用对地址空间进行分段的方法，将地址空间分成 $K$ 段，占用地址的高 log$K$ bit，形成多棵第一层树。

对于 $K$ 的选择有以下考虑：1) 空置率，在前缀覆盖范围一定的情况下，$K$ 增大，每段范围缩小，空置率可能会增加，内存浪费增大；2) 分散率，$K$ 越大，第一层树越多，分散越广，查找性能越高；3) 被覆盖率，若 $K$ 很大，log$K$ 也会增大，假设存在某个前缀 $Px$，其长度 $Lx$ 小于 Log$K$，那么则要进行位扩展到 log$K$ bit，从而增加 $2^{Lx-\log K}$ 个前缀，内存占用增加，也增加了更新的复杂度。良好的路由算法应具有低空置率、高分散率和零被覆盖率。对于 IPv6，目前最短前缀长度为 16，除去最高 3 bit 的全局地址标识，可以用高 13 位分段，$K$=$2^{13}$，既可以避免被覆盖现象，又可以最大程度上划分子树，分散率最高。

根据这种划分，Potaroo 路由表最终形成了如表 3 所示的前缀分布情况。其中，Num 表示经划分后，查找树中节点数目为[1, 10]、[11,100]、[101, 1 000]或[1 001, ∞]的查找树类型；*Count.*表示该类型查找树的数目，比如节点个数在[101, 1 000]范围内的查找树有 17 棵；*Percent* 表示该类查找树包含的前缀占前缀总数的百分比，可以看出 100+类型包含了最多的前缀；*Acc-Rate* 表示极限情况下各个段中查找深度在划分前后的理论比值，体现了极限情况下查找的加速情况。

**表 3　　前缀分布关系**

| Num | *Count.* | *Percent* | *Acc-Rate* |
|---|---|---|---|
| 1+ | 13 | 0.36% | 9.87 |
| 10+ | 6 | 4.83% | 1.89 |
| 100+ | 17 | 63.02% | 1.48 |
| 1 000+ | 1 | 30.43% | 1.06 |

可以看到，100+类型的前缀占整体数目的 63.02%，且理论上可加速到 1.48 倍，因此这种分段方法可有效提升整体的性能。同时，根据笔者对 IPv6 网络中骨干路由器统计，稀疏区域 1+类型包含的以“2002::”开头的前缀(6to4 隧道)占有了约 40%的流量，经过地址范围划分，该类流量理论上可提升到 9.87 倍，提升效果明显。可见，对地址范围的有效分段可以较大提高 IPv6 路由转发的性能，是十分必要和有效的。

### 3.1.2　前缀集合划分

前缀集合划分借鉴了 Multi-bits 前缀位集合方法，在前缀按值升序排序基础上，将 $M$ 个连续前缀划分为一个前缀集合，称为众节点，并用[$Ss$, $Se$]表示该众节点的范围。假设图 4 所示前缀经地址划分后属于同一查找树，图 5 则表示经过集合划分（假设 $M$=3）后的情况。

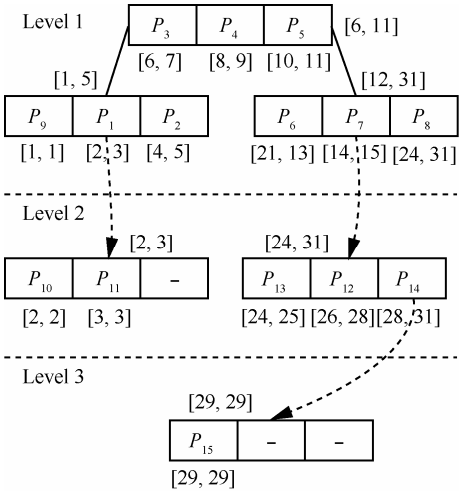



图 5　集合划分框架

前缀集合划分直观上反映出 3 个特点：1) 每个众节点含有多个前缀；2) 查找树深度降低；3) 众节点存在空闲位置。查找树深度降低是前缀合并的必然结果。原始情况下查找树深度为 log$N$，集合划分后，查找树深度变为 log($N$/$M$)，总层数减少了 log$M$。这对于查找失败选择默认路由的情况是有利的，可减少 log$M$ 次比较。

众节点会存在空闲位置，如 Level3 中 $P_{15}$ 所在众节点，其根本原因是原始情况下查找树包含的前缀数不能被 $M$ 整除，因此必然在查找树最右众节点出现空闲现象。由于空闲情况只在最右出现，且目前路由表不会形成大量的包含子树，因此不会产生浪费大量内存的情况。而在更新过程中，插入和删除可能会形成空闲位置，需要对更新节点进行自修复以避免不断更新产生大量的空闲众节点，将在 3.3 节详细说明。

## 3.2　BSRPS 查找

从图 5 可知，BSRPS 虽然形成了众节点，但查

找树基本框架未变，因此可根据每个众节点的范围进行二分查找。算法首先需要定位到某一众节点，假设 $N_i$ 表示某棵查找树，则定位某一众节点的时间复杂度为 $\log(2N_i/M)$。之后仍用二分查找定位该众节点中的前缀，时间复杂度为 $\log M$。单棵查找树总时间复杂度为 $\log(2N_i/M)+\log M=\log 2N_i$，与原始算法一致。因此，对于单棵查找树，BSRPS 并没有提升理论上的查询复杂度，主要是降低更新不平衡对查找性能的影响，但对于查找失败进行默认路由的情况，查询复杂度降为 $\log(2N_i/M)$，性能有一定提升。在加入地址范围划分后，平均时间复杂度有所降低，为 $\log(2N_i/K)$。下面给出查找算法，其中，bs-set(*pt*, *address*)对一个众节点中的 $M$ 个前缀进行二分查找，*Prefix-set* 表示第一层查找树。

**算法 1** BSRPS 主查找算法

```
1)  procedure Lookup(address)
2)  k ← address.[3-16bits]
3)  pt ←Prefix-set[k]
4)  return RL (pt, address)
5) end procedure
```

**算法 2** 子树内递归查找算法

```
1) procedure RL (pt, address)
2)   while(pt)
3)       if address<pt.left then
4)           pt ←pt.lset
5)       else if address<pt.right then
6)           pt ←pt.rset
7)       else break
8)   end while
9)   if !pt then
10)      return no match
11)  end if
12)  if pre ← bs-set(pt, address) then
13)      if pre.child then
14)          pre_c = RL (pre.child, address)
15)      end if
16)      return (pre_c?pre_c:pre)
17)  end if
18)  return no match
19) end procedure
```

### 3.3 更新与自修复

BSRPS 的更新过程与 RBMRTs 类似，包括同层内添加删除前缀和层间前缀子树的替换与扩展。但是，由于 BSRPS 采用了众节点模式，在比较和移动时略有不同，如图 6 显示了添加前缀 $P_{\text{new}}$ 时的 3 类情况（删除过程类似）：1) 添加的前缀只与一个众节点相关，如图 6(a)~图 6(d)所示；2) 与任何众节点均无关，如图 6(e)所示；3) 与多个众节点相关，如图 6(f)所示。

图 6(a)中，$P_{\text{new}}$ 覆盖了整个众节点，用只包含 $P_{\text{new}}$ 的新众节点进行替换，同时将包含 $P_1$~$P_m$ 的原始众节点扩展为 $P_{\text{new}}$ 包含子树；图 6(b)表示 $P_{\text{new}}$ 包含了原始众节点的部分前缀，此时将 $P_2$、$P_3$ 形成新的众节点并扩展为 $P_{\text{new}}$ 的包含子树，并用 $P_{\text{new}}$ 替换两者原始位置；图 6(c)中，$P_{\text{new}}$ 属于 $P_3$ 的子前缀，进入 $P_3$ 的包含子树更新；图 6(d)表示 $P_{\text{new}}$ 位于该众节点中两个连续前缀中间，此时需要挪动众节点中的其他前缀以重新形成有序数列。若该众节点是满节点，则在插入 $P_{\text{new}}$ 时必须要移出 $P_1$ 或 $P_m$，此时 $P_1$ 或 $P_m$ 需以该众节为根节点进行如图 6(e)的插入过程，否则挪动单侧前缀即可形成新有序数列；图 6(e)表示 $P_{\text{new}}$ 不包含于任何众节点，若 $P_{\text{reset}B}$ 未满，则填入 $P_{\text{reset}B}$ 末尾，否则新建一众节点并链到 $P_{\text{reset}B}$ 的右孩子处。反之亦然。

当 $P_{\text{new}}$ 涉及多个众节点时，更新操作相对复杂，如图 6(f)显示了向左子树扩展多个众节点的情况。可以看到 $P_{\text{new}}$ 覆盖了 $P_{\text{reset}A}$ 的前面部分、$P_{\text{reset}B}$ 的后面部分，且对于所有大于 $P_{\text{reset}B}.P_1$ 小于 $P_{\text{reset}A}.P_3$ 的前缀

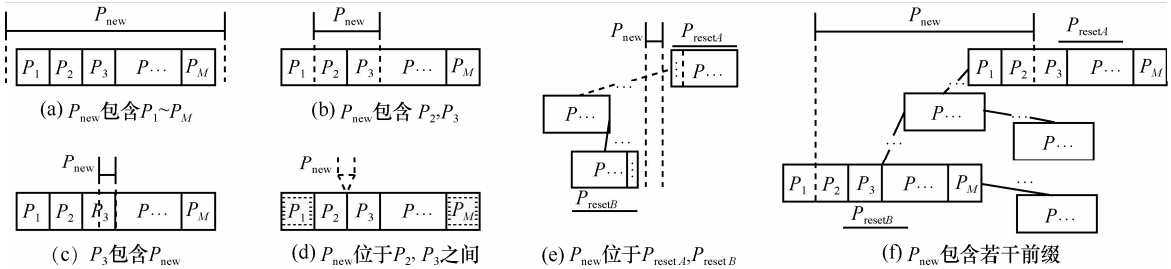



图 6 BSRPS 更新情况（$P_{\text{reset}}$ 表示众节点）

（斜杠条纹显示），这些需要被替换的众节点组成 $P_{new}$ 的包含子树（与 RBMRTs 相同，需要对包含子树进行平衡旋转，防止出现大量单分支众节点）。而 $P_{new}$ 则被加入 $P_{resetA}$ 或 $P_{resetB}$，更新原有众节点。

上述方法可正确更新，但存在以下 2 个问题：1) 缺少机制保证被更新的众节点是满的，这会导致该众节点可能只剩下极少的有效前缀，如图 6(a)、图 6(f)分别会使一个和 2 个众节点包含的前缀数目大量减少，众节点的空置率偏高，浪费大量内存；2) 当出现大量不满众节点时，必然会增加查找树的深度，影响查询和更新效率。为此，本文提出了更新节点的自修复算法以避免出现上述情况，保证最坏的查询复杂度为 log(2*N*/*KM*)。

节点自修复算法的核心思想是当众节点自身在更新后不满时，不断从小于或大于该众节点的区域中取最接近前缀以填补该众节点的空白，直到该众节点满。该算法主要包括 4 个操作：重排序、定位、移动和旋转。下面以图 7 为例说明这一过程。

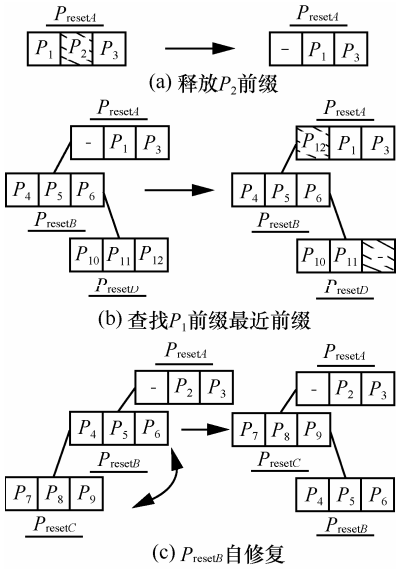



图 7　自修复算法举例

图 7(a)中假设 $P_{resetA}$ 中前缀 $P_2$ 被替代或被删除后，$P_{resetA}$ 进入自修复过程（更新过程涉及的所有众节点均需进行自修复，以保证除叶子节点外的所有众节点均为满众节点）。首先假设 $P_{resetA}$ 优先选择从左子树取最接近的前缀，则 $P_{resetA}$ 需要对自身进行重排序留前部的空位，如图 7(a)所示；之后查找左孩子的最右子树以定位小于该众节点的最近前缀，如图 7(b)所示，将 $P_{12}$ 挪动到 $P_{resetA}$ 的空闲处。此时，$P_{resetD}$ 虽被更新但由于其为叶子节点因此不需要进行自修复。图 6(c)假设 $P_{resetB}$ 的右子树 $P_{resetD}$ 中的前缀已经全被修复到 $P_{resetA}$ 的第一个位置上，因而必须从 $P_{resetB}$ 的末尾选择前缀移动到 $P_{resetA}$ 的前部。由于 $P_{resetB}$ 不是叶子节点（含有左子树），因此也要执行自修复。此时，$P_{resetB}$ 有 2 个可用策略：1) 从其左子树里不断移动最接近的前缀补位；2) 若其左孩子非叶子节点，则可首先进行旋转，再移动 $P_6$ 到 $P_{resetA}$ 中，如图 7(c)的右侧部分，既可以降低更新的时间复杂度，还可平衡查找树，降低查找深度。

前缀集合划分和更新节点的自修复并不能避免不平衡性的出现，但可以减少最坏情况下查找和更新的复杂度。同时，在配合使用 AVL 算法时，查找树节点总数从 *N* 降为 *N*/*M*，可节省大量平衡开销。

## 4　性能分析与实验

本节通过理论分析和实际测试对 BSRPS 算法的性能进行了评估。首先在内存使用上，该算法首先进行了区域划分，将 Level1 层查找树分成了 *K* 份，因此需要 *O*(*K*)的空间存储树根指针；同时，由于一般情况下只有最右众节点可能存在不满情况，因此空间复杂度为 *O*(*K*+2*N*)。但在经过不断更新后，由于叶子节点无法进行自我修复，会出现叶子众节点中只含有极少前缀的情况，浪费存储空间。在这种情况下，对于二叉树而言，假设其含有的非叶子众节点个数为 *x*，那么叶子众节点数目至多 *x*+1，由于 *Mx*+1×(*x*+1)=2*N*，可推导出 *x*=(2*N*–1)/(*M*+1)，由此查找树占用的内存为 *M*×(2*x*+1)≈2*M*(2*N*–1)/(*M*+1)，因此算法最坏空间复杂度为 *O*(*K*+4*N*)。

在时间复杂度上，对于每一层查找树而言，该算法拥有 log2$N_i$ 的查询速度。但是，前缀中存在若干包含子树，实际查询效率必然会低于该值，且查询效率随层数增多而下降。目前前缀最多包含 5 层，且每层的个数不定，不利于理论分析，因此通过实验进一步分析其对查找性能的影响。在更新方面，与查询过程相同，首先定位新前缀位置，进而向下替换或者更新，其最多需要对 2 个众节点进行自修复（如图 6(f)所示情况），因此更新复杂度为 *O*(log2$N_i$+2*M*)。

为了测试算法实际性能，笔者在一台 2.6 GHz 主频、16 GB 内存的服务器上对该算法进行了对比测试。对比算法采用 Patricia、BSR 和 RBMRTs 3 种，实验数据采用了文献[1,15]提供的 Potaroo、Tokyo 和 Atlanta 路由表。表 4 列出了 3 个 BGP 路

**表 4** **3 种路由表及内存占用**

| Name | Count. | Patricia/kB | BSR/kB | RBMRTs/kB | BSRPS/kB | Max-L | L1 | SEG |
|---|---|---|---|---|---|---|---|---|
| Potaroo | 7 843 | 604 | 429 | 502 | 499 | 5 | 5 570 | 28 |
| Tokyo | 8 019 | 618 | 449 | 513 | 508 | 5 | 6 129 | 28 |
| Atlanta | 8 140 | 626 | 456 | 521 | 515 | 5 | 6 136 | 29 |

由表的相关参数。

表 4 第 2 列显示了 Potaroo、Tokyo 和 Atlanta 3 个路由表中路由条数，均在 8 000 左右。第 3~6 列显示了 4 种路由算法在不同路由表下的内存占用量，其中，BSR 在理论上不存在内存的浪费因此在实际中的占用量也最少，而 BSRPS 方法理论上的空间复杂度大于 RBMRTs，但实际中却相反，这主要是由于 BSRPS 方法中，众节点对若干前缀进行了集合，因此对于一个 $M$ 个前缀的众节点只需 2 个指针指向左右子树，而 RBMRTs 方法则需要 $2M$ 个指针，增加了大量的内存。表 4 还显示了各个路由表中最深层均为 5 层、且大部分前缀处于 Level1 层，因此不会对算法总体性能产生巨大影响。最后一列 SEG 表示（按照 $K$=8 192 计算）地址划分将路由表分成了约 28 个段（即分成 28 棵第一层树），分配较不平均，有效分段只占 3.4%。随着 IPv6 的发展，当高 16 bit 地址使用明显增加时，有效分段比例也会有明显提高。

为了测试算法的查询和更新速度，本文选择 $M$=5，$K$=8 192。理论上，$M$ 大小不影响查找速度，但 $M$ 较大时会影响算法的内存占用量。同时，$M$ 的大小也与算法的更新时间复杂度相关，在当前路由条数在 8 000 左右的情况下，log$N$ 约为 13，选择 $M$=5 可不过分突出众节点重排序的时间开销。在以上条件下，表 5 显示了 4 种路由算法在查找 Potaroo 路由表时，不同分布前缀的查询能力。其中，1 000+ 表示地址范围划分后前缀个数大于 1 000 的查找树（与表 3 相同），表中数值表示该算法对于该类查找树各层部分前缀的平均查找时间。

**表 5** **Potaroo 查找/μs**

| 前缀个数 | Patricia | BSR | RBMRTs | BSRPS |
|---|---|---|---|---|
| 1 000+ | 0.391 | 0.124 | 0.119 | 0.110 |
| 100+ | 0.293 | 0.135 | 0.106 | 0.097 |
| 10+ | 0.277 | 0.141 | 0.101 | 0.077 |
| 1+ | 0.081 | 0.128 | 0.092 | 0.032 |

表 5 中，Patricia 对集中区和稀疏区前缀的查询速度有较大差异，稀疏区的查询速度明显优于集中区，差距在 4 倍以上。BSR 基本保持平均，对集中区的查询速度优于 Patricia，而稀疏区慢于前者。RBMRTs 方法与 BSR 方法类似，性能相差不多。BSRPS 算法由于进行了地址划分，生成了多棵 Level1 层查找树因此与 Patricia 具有相似的性能趋势，均对稀疏区域的前缀有较好的查询性能。而与 BSR 和 RBMRTs 相比，在集中区，BSRPS 与前两者性能相近，这是由于集中区形成的 Level1 层查找树中节点总数并没有成指数级的降低，查找树深度降低不明显，从而导致性能提升不高。在稀疏区，由于 BSRPS 地址范围划分使得稀疏区前缀形成的独立查找树只具有少量的节点，查找深度大量减少，因此性能有很大提高。Tokyo 和 Atlanta 路由表的测试结果与 Potaroo 类似。

在更新测试中，本文针对 Potaroo 前缀集合首先对图 6 所示的 6 种情况进行了更新测试。测试选取集中区域（1 000+，D）和稀疏区域（10+，F）的若干第一层子树节点进行了插入测试。

**表 6** **Potaroo 更新性能/μs**

| Case | Patricia | RBMRTs | BSRPS |
|---|---|---|---|
| D-a | 1.268 | 1.185 | 1.389 |
| D-b | 1.211 | 1.107 | 1.211 |
| D-c | 1.178 | 1.083 | 1.012 |
| D-d | 1.210 | 1.146 | 1.262 |
| D-e | 1.216 | 1.077 | 0.906 |
| D-f | 1.063 | 1.218 | 1.367 |
| F-a | 0.932 | 1.151 | 0.936 |
| F-b | 0.915 | 0.911 | 0.894 |
| F-c | 0.892 | 1.083 | 0.821 |
| F-d | 0.885 | 1.097 | 0.835 |
| F-e | 0.901 | 0.882 | 0.812 |
| F-f | 0.863 | 1.153 | 0.945 |

表 6 中，BSRPS 方法在稀疏区域（F）的更新性能与 Patricia 相当，且由于查找性能提高和较少的自修复过程，整体更新性能优于 RBMRTs 算法。

在集中区域（D），c/e 2 种更新情况不需要复杂的更新流程，时间少于 RBMRTs；在 a/b/d/f 4 种情况下，由于需要建立子树或更新节点自修复等相对复杂的更新流程，BSRPS 算法较 RBMRTs 增加了至多 20%的更新时间。其中，a/b/f 3 种情况表示 $P_{new}$ 前缀包含多个原有路由前缀的情况，d 情况表示 $P_{new}$ 前缀处于原有 2 个前缀之间。通常路由更新时并不能进行查找操作，因此当上述 4 种情况频率较高时，会增加分组丢失概率，降低用户体验。相反，若更新的情况主要是 c/e 2 种情况，如在新启用的地址空间中分配 IPv6 的前缀时，更新性能相比 RBMRTs 会有所提高。

为了验证 BSRPS 在极端情况下的性能，本文针对集中区（Du）与稀疏区（Fu）子树第一层进行了连续插入 64 个递增前缀的极端环境测试，并获得了这种情况对于查找速度的影响（Du-L，Fu-L），测试结果如表 7 所示。可以看到，BSRPS 算法更新时间较长，但不超过 RBMRTs 的 115%。在经过极端插入后，BSRPS 算法在集中区和稀疏区均具有较好的查询速度，查询时间仅为 RBMRTs 的 30%左右，Patricia 的 40%左右。主要原因是连续插入的 64 个递增前缀会在 RBMRTs 查找树下形成深度为 64 的局部单链子树，BSRPS 形成深度为 64/$M$ 的局部单链子树，查找性能有线性级的提高。理论上，Patricia 深度增加最少，为对数级。

**表 7　　Potaroo 极限情况/μs**

| Case | Patricia | RBMRTs | BSRPS |
|---|---|---|---|
| Du | 2.356 | 2.157 | 2.414 |
| Fu | 1.331 | 1.809 | 1.465 |
| Du-L | 0.409 | 0.441 | 0.141 |
| Fu-L | 0.264 | 0.400 | 0.116 |

## 5　结束语

高效的 IPv6 路由算法是提高网络传输性能的重要因素。本文提出的 BSRPS 算法通过地址范围划分、连续前缀集合和更新节点自修复算法在牺牲一定更新时间的条件下有效地提升了平均和极端情况下的查询速度，同时也降低了内存的占用量，具有较好的性能。测试结果表明，查找性能在平均情况下性能比 RBMRTs 提升 1.5 倍，极端情况下提升 3 倍以上。

本文的研究仍存在一定不足。首先，$M$ 取值没有给出规范的公式，这需要与内存和更新时间联合评估；其次，本算法未从根本上消除 RBMRTs 算法频繁更新产生的不平衡性，只是降低不平衡性对查询的影响。当使用 AVL 进行平衡时，由于降低了节点总数，因此可提高 AVL 的性能；再次，集中区域 a/b/d/f 4 种情况下更新时间有所降低，增加更新过程导致分组丢失的概率，需进一步研究；最后，地址划分的有效性较低，有效值仅为 3%，对集中区域的性能提升效果不佳，可采用文献[14]中将 2001::/16 前缀进行单独划分等方式进一步优化。

## 参考文献：


[1] http://bgp.potaroo.net[EB/OL]. 2010.

[2] SUN Q, HUANG X H, ZHOU X J. A dynamic binary hash scheme for IPv6 lookup[A]. Proceedings of IEEE GLOBECOM[C]. New Orleans, USA, 2008. 1-5.

[3] RUIZ-SANCHEX M A, BIERSACK E W, DABBOUS W. Survey and taxonomy of IP address lookup algorithms[J]. IEEE Network, 2001, 15(2):8-23.

[4] SAHNI S, KIM K S. Efficient construction of multibit tries for IP address lookup[J]. IEEE/ACM Trans Networking, 2003, 11(4): 650-662.

[5] LIM H, YIM C H. SWARTZLANDER E. Priority tries for IP address lookup[J]. IEEE Transactions on Computers, 2010, 6(59): 784-794.

[6] WALDVOGEL M, VARGHESE G, TURNER J. Scalable high speed IP routing lookups[A]. Proceeding of ACM SIGCOMM[C]. Cannes, France, 1997. 25-36.

[7] Kim K S, SAHNI S. IP lookup by binary search on prefix length[A]. ISCC '03 Proceedings, of the Eighth IEEE International Symposium on Computers and Communications[C]. Antalya, Turkey, 2003.77-82.

[8] LIM H, KIM W, LEE B. Binary search in a balanced tree for IP address lookup[A]. IEEE Workshop High Performance Switching and Routing[C]. HongKong, China, 2005. 490-494.

[9] LIM H, KIM H, YIM C. IP address lookup for internet routers using balanced binary search with prefix vector[J]. IEEE Transactions on Communication, 2009, 57(3):618-821.

[10] LAMPSON B, SRINIVASAN V, VARGHESE G. IP lookups using multiway and multicolumn search[J]. IEEE/ACM Trans Networking, 1999, 7(3):324-334.

[11] WARKHEDE P, SURI S, VARGHESE G. Multiway range trees: scalable IP lookup with fast updates[J]. Computer Networks, 2004, 44: 289-303.

[12] SAHNI S, KIM K S. An $O(\log N)$ dynamic router-table design[J]. IEEE Transactions on Computers, 2004, 53(3):351-363.

[13] ZHONG P F. An IPv6 address lookup algorithm based on recursive balanced multi-way range trees with efficient search and update[A]. Computer Science and Service System (CSSS)[C]. Paris, France, 2011. 2059-2063.

[14] 李振强, 郑东去, 马严. TSB:一种多阶段 IPv6 路由表查找算法[J]. 电子学报, 2007, 35(10):1859-1864.
LI Z Q, ZHENG D Q, MA Y. TSB:a multi-stage algorithm for IPv6 routing table lookup[J]. Chinese Journal of Electronics, 2007, 35(10): 1859-1864.

[15] http://www.routeviews.org[EB/OL]. 2012.


（下转第 48 页）

[26] KUO C C, CHUI L C, CHUNG B L. The comparison of algorithms in change-points problem[J]. Journal of Applied Science and Engineering, 2012, 15(1):11-19.

[27] BERTRAND P R, FHIMA M, GUILIN A. Fast change point analysis on the Hurst index of piecewise fractional brownian motion[J]. Journée de Statistiques 2011 (JDS 2011), 2011,(3):1-6.

[28] HUAT N K, MIDI H. Change point detection with robust control chart[J]. Mathematical Problems in Engineering, 2011. 2011:1-20.

[29] SHENG H, CHEN Y Q, QIU T. On the robustness of Hurst estimators[J]. IET Signal Process, 2011, 5(2):209-225.

[30] Stilian stoev's new web page[EB/OL]. http://www.stat.lsa.umich.edu/~sstoev, 2011.

[31] PARK J, PARK C. Robust estimation of the Hurst parameter and selection of an onset scaling[J]. Statistica Sinica, 2009, 19(4):1531- 1555.

**作者简介：**

**郭通**（1984-），男，江西南昌人，国家数字交换系统工程技术研究中心博士生，主要研究方向为宽带信息网络、高速网络业务管控等。

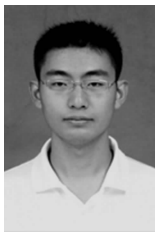
**兰巨龙**（1962-），男，河北张北人，博士，国家数字交换系统工程技术研究中心教授、博士生导师，主要研究方向为宽带信息网络、高速路由器核心技术等。

**黄万伟**（1979-），男，江苏盐城人，博士，国家数字交换系统工程技术研究中心讲师，主要研究方向为实时任务调度、可重构计算。

**张震**（1985-），男，山东济宁人，国家数字交换系统工程技术研究中心博士生，主要研究方向为宽带信息网络、高速网络业务识别等。

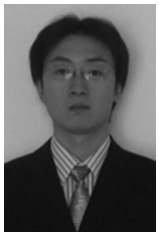
（上接第 37 页）

**作者简介：**



**崔宇**（1985-），男，黑龙江哈尔滨人，哈尔滨工业大学博士生，主要研究方向为IPv6、网络安全。

**张宏莉**（1973-），女，吉林榆树人，哈尔滨工业大学教授、博士生导师，主要研究方向为计算机网络信息安全、并行处理。



**田志宏**（1978-），男，黑龙江哈尔滨人，博士，哈尔滨工业大学副研究员，主要研究方向为网络安全主动防御、入侵取证。

**方滨兴**（1960-），男，江西万年人，哈尔滨工业大学教授、博士生导师，主要研究方向为计算机体系结构、信息安全和计算机网络。