使用Hash表和树位图的两级IPv6地址查找算法

王亚刚[1,2] 杜慧敏[2] 杨康平[2]

(西安电子科技大学计算机学院 西安 710071)[1] (西安邮电学院计算机科学与技术系 西安 710121)[2]

摘 要 为了提高IPv6地址查找效率,在分析IPv6路由前缀长度分布规律的基础上,提出了基于哈希表及树位图(Tree-bitmap)的两级IPv6地址查找算法。算法将长度为16,32,48和64比特的前缀分别存储在4个Hash表中,其余前缀的前16,32和48比特利用已有的Hash表存储,剩余的不足16比特的部分前缀利用树位图存储,并将树位图的入口地址保存在Hash表中。IP地址查找时在Hash表和树位图中进行两级查找。实验表明,该查找算法的平均内存访问次数为1～2,最坏情况下为7,适用于高速IPv6地址查找。

关键词 IPv6,地址查找,哈希表,树位图

中图法分类号 TP393.03 **文献标识码** A

Two-stage IPv6 Address Lookup Scheme Based on Hash Tables and Tree Bitmaps

WANG Ya-gang[1,2] DU Hui-min[2] YANG Kang-ping[2]

(School of Computer, Xidian University, Xi'an 710071, China)[1]

(Department of Computer Science and Technology, Xi'an Institute of Posts and Telecommunications, Xi'an 710121, China)[2]

Abstract IP address lookup is a key issue in modern high performance router design, especially with the evolution of IPv6. In order to improve the efficiency of IP address lookup, a novel IPv6 address lookup scheme based on Hash tables and tree bitmaps was proposed, with an analysis on the prefix length distribution of routing table state-of-the-art. In this scheme, four Hash tables were used to store the prefixes with the length of 16, 32, 48 and 64 bits respectively; the sub-prefixes with length of 16, 32, 48 bits of the other prefixes were stored in these four Hash tables too, their remaining part shorter than 16 bits was coded into tree-bitmap and indexed by a certain Hash table entry, thereby to form a two-stage address lookup architecture. The results show that the scheme can achieve an average memory access number of 1～2 per IPv6 address lookup and 7 for the worst case, and can be applied in the high performance IPv6 address lookup implementation.

Keywords IPv6, Routing lookup, Hash table, Tree bitmap

Internet的快速发展对Internet性能提出了更高的要求。IP路由器作为Internet的核心设备,其基本功能是从输入接口接收IP分组,并根据IP分组首部中的目标地址,从路由信息表(RIB,Routing Information Base)中查找出相应的输出接口信息,然后将IP分组转发到相应的输出接口,从而完成分组的转发功能。然而,随着路由信息表中路由前缀(Routing Prefix)数目的快速增长,IP地址的查找效率问题日益突出,成为限制路由器性能提升的关键因素之一。为了解决路由信息表中路由前缀数目快速增长的问题,无分类域间路由(CIDR,Classless Inter-Domain Routing)的IP地址编制方案[1]被正式采用。CIDR可以进行路由表项的聚合,从而减少路由表项的数目。然而,随着CIDR的引入,IP地址查找从一个精确匹配的问题转变成为一个最优匹配的问题,必须采用最长前缀匹配(LPM,Longest Prefix Match)算法解决,从而大大增加了IP地址查找的复杂性。

另一方面,随着IPv6[2]的引入,IP地址的长度从IPv4的32比特增加到IPv6的128比特,查找效率问题更加突出。因此,必须结合IPv6的特点及其前缀的分布规律,研究适合IPv6的地址查找算法。

本文在分析IPv6路由前缀分布规律的基础上,提出了基于前缀长度的二分查找和树位图查找的两级查找方法,该方法利用4个Hash表,分别存储长度为16,32,48和64比特的前缀,通过对该4个Hash表的查找,可以找到最佳匹配前缀(BMP,Best Match Prefix),或者将地址查找限定在一个步长为4、深度为4的多比特树中,并使用Tree bitmap算法进行查找。实验结果表明,基于Hash表和Tree bitmap的两级IPv6地址查找算法中,大部分前缀匹配可以在第一级的Hash表查找中完成,算法的平均查找次数为1～2,最坏情况下的查找次数为7。

本文第1节介绍了IPv6查找的相关研究;第2节在分析IPv6路由前缀分布规律的基础上,介绍了基于Hash表和树位图的两级IP地址查找算法;第3节给出了相关的实验数据

到稿日期:2009-10-20 返修日期:2009-12-25 本文受国家自然科学基金(60976020)资助。

王亚刚(1972－),男,博士生,讲师,主要研究方向为计算机网络、SoC设计与验证等,E-mail:wangyg@xupt.edu.cn;杜慧敏(1966－),女,博士,教授,硕士生导师,主要研究方向为SoC形式化设计与验证;杨康平(1984－),男,硕士生,主要研究方向为SoC设计与验证。

及其分析；最后总结了本文并给出了进一步的研究方向。

1 相关研究

目前 IPv6 地址查找的算法主要借鉴了 IPv4 的地址查找方法，包括基于 Trie 树的算法、基于范围的查找(Search on Prefix Ranges)、基于前缀长度的二分查找(Binary Search on Prefix Lengths)以及基于 TCAM 的算法等[3,4]。表 1 给出了几种典型查找算法的性能比较(其中，W 表示前缀长度，N 表示路由前缀的数目，k 表示多比特树的步长)。

表 1 各种算法的性能比较

Algorithm	Performance		
	Lookup	Storage	Update
Binary Trie	O(W)	O(NW)	O(W)
k-bits Trie	O(W/k)	$O(2^kNW/k)$	$O(W/k+2^k)$
Binary Search on Prefix Lengths	$O(Log_2W)$	$O(NLog_2W)$	$O(Log_2W)$
k-way Range Search	$O(log_kN)$	O(N)	O(N)
TCAM	O(1)	O(N)	O(W)

Waldvogel 等提出了一种按前缀长度进行二分查找的算法[5]，该算法将最长前缀匹配按前缀长度分解成一系列的精确匹配，并将前缀按长度分别存储在不同的 Hash 表中，地址查找时，按前缀长度对所有的 Hash 表进行二分查找。该算法查找的平均次数为 $O(\log_2 W)$，存储空间复杂度为 $O(N\log_2 W)$。该算法具有良好的可扩展性，对于 IPv6 的最大查找次数为 7。但该算法为了进行二分查找并避免回溯，引入了大量的标记(Marker)，约占前缀总数的 25%，因此增加了存储的复杂度。

Eatherton 提出的 Tree bitmap 算法具有高速查找、存储代价小的特点，并支持增量更新[6,7]，是一种改进的多比特树(Multi-bit Trie)算法。Tree bitmap 算法的设计主要具有以下特点：(1)在 Tree bitmap 算法中，要求同一个父节点的所有子树节点(亲兄弟)必须连续存储，父节点中使用一个子树指针(Child Node Pointer)，所有的子树节点可以通过该子树指针和一个偏移量相加来获得；(2)每一个树节点中包括两个位图，其中内部前缀位图(Internal Prefix Bitmap)用来描述该多比特节点中所包含的前缀信息，扩展路径位图(Extending Path Bitmap)用来描述该节点的所有子树节点信息；(3)Tree bitmap 算法中树节点的大小必须尽量小，从而保证较少的存储访问次数；(4)同一个节点中有效前缀所对应的下一跳信息也必须连续存放，其起始地址保存在下一跳指针(Next hop Pointer)中。

如图 1 所示，(a)为原始的前缀信息，(b)为相应的步长为 4 的 Tree bitmap 存储数据结构。每个树节点(Trie node)中包含了指向连续存放的子树节点的指针以及扩展路径位图，也包含了指向连续存放的下一跳信息的指针和本节点所描述的内部前缀位图。例如，在根节点 Root node 中，内部前缀位图为 000011000000010，该位图中的 3 个“1”分别代表了 P_1，P_2，P_3，其对应的下一跳信息 Ptr1_1，Ptr1_2，Ptr1_3 则连续存放，并且将其起始地址保存在 Root node 的 Next Hop Pointer 字段中。扩展路径位图 0101010010010000 描述了该 Root node 的子树扩展信息，5 个“1”分别对应于 5 个子树节点 $Node_2$，$Node_3$，$Node_4$，$Node_5$ 和 $Node_6$，这些子树节点连续存放，其起始地址保存在 Root node 的 Child Node Pointer 字段中。假设前缀长度为 W(对于 IPv4，$W=32$；对于 IPv6，$W=128$)，前缀数目为 N，k 为多比特树的步长，则该算法的查找复杂度为 $O(W/k)$，存储复杂度为 $O((2^K * N * W)/k)$。

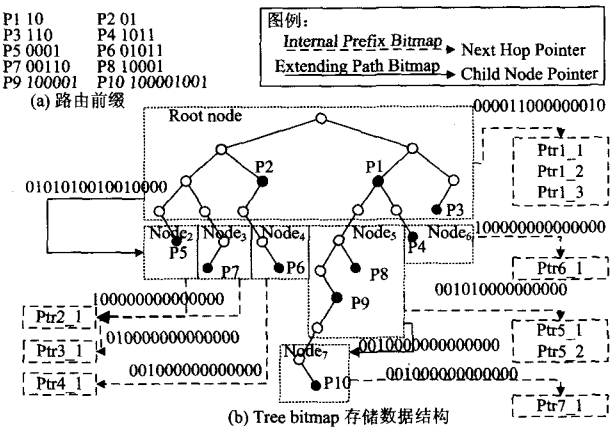

图 1 Tree bitmap 及其存储结构

也能看出，基于前缀长度的二分查找中会引入较多的标记(Marker)，增加了存储代价，而且需要使用较多的 Hash 表。Tree Bitmap 算法存储代价小，支持增量更新，但在步长 k 较小(例如 $k=4$)时，多比特树的深度较大，因此平均查找次数较大。本文提出的算法根据 IPv6 前缀的分布规律，结合了上述两种算法的优点，使用 4 个 Hash 表分别存储长度为 16，32，48 和 64 比特的前缀，并同时作为多比特树位图的索引。因此，该算法既可以在 4 个 Hash 表中快速完成大部分的地址查找，同时对于长度不能被 16 整除的前缀，也可以大大降低多比特树的深度，减少平均查找次数。

2 两级 IP 地址查找算法

2.1 定义

为了讨论方便，特作如下定义。

定义 1(前缀 $P(p,len)$) 表示长度为 len 的前缀 P，p 为该前缀的二进制比特串表示(由于 IPv6 地址长度为 128 位，为了方便，文中使用 IPv6 的地址记法)。

定义 2(前缀覆盖，Prefix Overlap) 对于前缀 $P_1(p_1, l_1)$，$P_2(p_2, l_2)$，如果 $l_2>l_1$，且 p_2 的二进制比特串的前 l_1 个比特与 p_1 的二进制比特串相同，则称 P_1 覆盖 P_2。例如，对于前缀 P_1(110010,6)，P_2(11001001,8)，因为 P_2 表示的前缀中前 6 个比特与 P_1 的前缀比特相同，因此称 P_1 覆盖 P_2。

定义 3(子前缀 $P_s(P,l_s)$) 设 $P(p,len)$表示长度为 len 的前缀 p，则将 p 中前 l_s 个比特表示的前缀称为 P 的长度为 l_s 的子前缀。例如，对于前缀 P(11001001,8)而言，$P_s(P,3)$ 表示的前缀为 110，$P_s(P,6)$表示的前缀为 110010。

定义 4(前缀区间 Zone(S,L)) 表示长度为 $S+1, S+2, \cdots, S+L$ 的前缀集合。

定义 5(Hash(L)) 表示存储前缀长度为 L 比特的 Hash 表，例如 Hash(16)，Hash(32)，Hash(48)，Hash(64)分别表示存储前缀长度为 16，32，48 及 64 比特前缀的 Hash 表。

定义 6(虚拟前缀 VP，Virtual Prefix) 是指存在于 Hash 表中的某个前缀表项，但该前缀表项并不代表某个真正 Hash 的路由前缀，而是某个较长前缀的子前缀。

2.2 IPv6 前缀分布规律

随着 IPv6 的发展，IPv6 路由表项数目逐渐增加，并呈现

出一定的分布规律[8]。本文以 potaroo[9]上的核心路由器中的 BGP 路由表前缀为例，分析该路由表中 IPv6 路由前缀长度的分布规律。图 2 描述了该路由表中表项前缀长度的分布规律，可以看出：

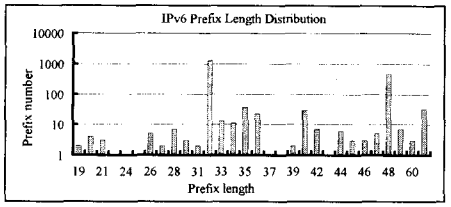

图 2　IPv6 前缀长度分布规律

(1)IPv6 路由前缀的长度介于 16 比特到 64 比特之间。

(2)长度为 32,48 和 64 比特的前缀表项占前缀总数的 90.5%。该特性表明，使用 Hash(16)，Hash(32)，Hash(48)和 Hash(64)这 4 个 Hash 表可以表示大部分的前缀。

(3)路由前缀之间普遍存在着覆盖关系，即大部分较长的前缀，其前 16,32 和 48 比特的子前缀都以较大的概率作为实际的路由前缀出现在路由表中。以 potaroo[9] 中的路由表为实验数据，并以 Hash(32)中引入的虚拟前缀的数目为例，从表 2 的统计数据可知，前缀长度 $32<L<48$ 比特的前缀表项共 142 项，其长度为 32 比特的子前缀数目为 72，引入新的 32 比特的虚拟前缀的数目为 28，比例为 28/142=19.7%。同样，对于前缀长度 $32<L\leqslant48$ 比特的前缀(共 600 项)，引入新的 32 比特的虚拟前缀的数目为 53，比例为 53/600=8.83%；对于前缀长度 $32<L\leqslant64$ 比特的前缀(共 643 项)，引入新的 32 的虚拟前缀的数目为 56，比例为 56/643=8.71%。可以看出，长度较长的前缀中包含的长度为 16,32,48 比特的子前缀，已经以较大的几率(大于 80%)出现在实际的路由前缀中。该特性表明，实际路由表中的前缀覆盖关系非常普遍，长度不等于 16,32,48 和 64 比特的前缀所引入的虚拟前缀的数目也较少。

表 2　前缀覆盖关系统计

前缀长度	#前缀	#32 比特子前缀	#32 比特 VP	VP 比例
$32<L<48$	142	72	28	19.7%
$32<L\leqslant48$	600	154	53	8.83%
$32<L\leqslant64$	643	165	56	8.71%

2.3　基于 Hash 表和树位图的两级查找算法总体描述

基于 Hash 表和树位图的两级 IPv6 地址查找算法的提出，主要基于如下的事实：

(1)由于 IPv6 路由前缀长度小于 64 比特且大量的前缀分布在 32,48 及 64 比特长度上(约为 90%)，因此使用 Hash(16)，Hash(32)，Hash(48)和 Hash(64)这 4 个 Hash 表可以表示大部分的前缀并完成大部分的地址查找任务。

(2)对于长度不能整除 16 的前缀 $P(p,l)$而言，由于前缀覆盖关系的普遍性，可以借助 Hash 表中已有的前缀表项来表示 P 中能被 16 整除的部分，而不会引入过多的存储开销；另外，使用树位图表示的前缀长度最大不超过 15 比特，可以大大降低多比特树的深度。例如表 3 中的前缀 P_4，其前缀长度为 35，而其长度为 32 的子前缀 2001:1668::/32 已经作为实际的路由表项出现在 Hash(32)中，因此可以利用 Hash(32)中 P_1 的表项来描述 P_4 前 32 比特的信息。对于 P_4 最后的 3 个比特则按照 Tree bitmap 方式进行存储，并将该 Tree bitmap 的根节点地址保存在该 Hash(32)的 P_1 表项中。

表 3　初始的路由表项

路由前缀	前缀值(IPv6 表示法)	前缀长度(比特)
P_1	2001：1668::	32
P_2	2001：12f8：0001::	48
P_3	2001：1650::	32
P_4	2001：1668：2000::	35
P_5	2001：0200：0900::	40
P_6	2001：0252：ffca：2669::	64
P_7	2001：0254：8000::	33
P_8	2001：02a0::	32
P_9	2001：0350::	32

本文提出的算法考虑到 IPv6 路由表的分布规律，结合了 Hash 查找及 Tree bitmap 查找的特点，对前缀信息的存储采用 Hash 表及 Tree bitmap 的两级结构，如图 3 所示。使用 4 个 Hash 表分别存储长度为 16,32,48 及 64 比特的前缀(包括虚拟前缀)，如果前缀 $P(p,len)$的长度 len 不能被 16 整除，那么首先将该前缀的子前缀 $P_s(P,16)$，$P_s(P,32)$，$P_s(P,48)$分别插入到 Hash(16)，Hash(32)和 Hash(48)表中，再将该前缀最后的 n 个 bit($n=len \bmod 16, 0<n<16$)形成的前缀按步长为 4 的 Tree bitmap 方式存储，并将该树位图节点的入口地址存储在 Hash 表的对应表项中。利用这种策略，与按前缀长度二分查找的算法[5]相比，可以避免引入过多的标记表项，减少 Hash 表的数量；与传统的 Tree bitmap 算法相比，可以降低多比特树的深度，提高查找效率。

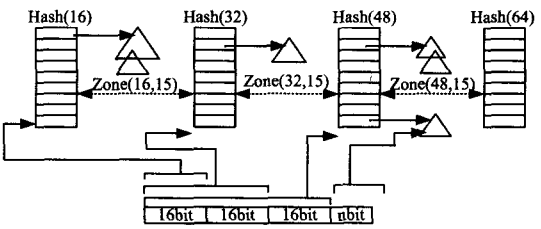

图 3　前缀的两级存储结构

2.4　Hash 表的组织和树位图的构造

本算法中使用了 4 个不同的 Hash 表，即 Hash(16)，Hash(32)，Hash(48)和 Hash(64)分别存储前缀长度为 16，32,48 及 64 比特的前缀。Hash(L)中表项数据结构的定义如图 4(a)所示，其中的 Prefix 字段和 Nexthop 字段分别存储路由前缀和下一跳信息；C_Zone 表示所有被该前缀所覆盖的且长度为 $L+1$ 到 $L+15$的前缀数目；C_Total 表示所有被该前缀所覆盖的且长度为 $L+16$ 的前缀总数(包括虚拟前缀)；Bitmap_Entry 则描述了所有被该前缀所覆盖的且长度为 $L+1$ 到 $L+15$ 的前缀形成的 Tree-bitmap 根节点地址。

Tree bitmap 算法中的树节点采用图 4(b)的数据结构。本算法中假设多比特树的步长为 4，则每个位图节点中都保存了 15 比特的内部前缀位图和 16 比特的扩展路径位图以及相应的下一跳信息的起始地址和子树节点的起始地址。

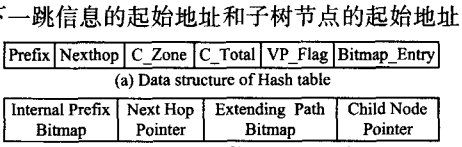

图 4　Hash 表及 Tree bitmap 的数据结构

2.5　路由表创建

路由表初始化为空，包括 4 个 Hash 表及其各个字段。

对每一个新建的路由表项，逐一进行插入处理，最终形成完整的转发表。对于前缀 $P(p,len)$ 的插入过程分为两种情况：

(1)如果前缀 $P(p,len)$ 的长度 len 可以被 16 整除，则依次将该前缀 P 的子前缀 $P_s(P,16)$，$P_s(P,32)$，$P_s(P,48)$ 和 $P_s(P,64)$ 分别存储在 Hash(16)，Hash(32)，Hash(48) 和 Hash(64)中；如果该前缀在 Hash 表中作为虚拟前缀 VP 出现，则将该表项的虚拟前缀标志 VP_Flag 置 1；

(2)如果前缀 $P(p,len)$ 的长度 len 不能被 16 整除，那么首先将该前缀的子前缀 $P_s(P,16)$，$P_s(P,32)$，$P_s(P,48)$ 分别插入到 Hash(16)，Hash(32) 和 Hash(48)表中(如果该前缀在 Hash 表中作为虚拟前缀 VP 出现，则将该表项的虚拟前缀标志 VP_Flag 置 1)，再将该前缀最后的 n 个 bit($n=len \bmod 16, 0<n<16$)形成的前缀按步长为 4 的 Tree bitmap 方式存储，并将该树位图节点的入口地址存储在 Hash 表的对应表项中。

以表 3 所示的路由表项为例，前缀 P_1(2001:1668::,32)插入时，(1)将 P_1 的前 16 个比特插入到 Hash(16)中，由于 P_1 的前 16 个比特表示的前缀 2001::/16 并不是实际的前缀表项，因此将该表项的虚拟前缀标志置为 1，同时由于 P_1 被该虚拟前缀表项所覆盖，且 P_1 的长度等于 $L+16$(此时 $L=16$)，因此该虚拟前缀表项的 C_Total 加 1；(2)将 P_1 的前 32 个比特插入到 Hash(32)中，由于该 32 比特前缀 2001:1668::/32 是实际的前缀表项，因此该表项的虚拟前缀标志置为 0。

进行 P_4 表项的插入时，同样地，(1)首先将 P_4 的前 16 个比特插入到 Hash(16)中，由于该 16 比特的前缀已经存在于 Hash(16)中，而且 P_4 的前 32 个比特必然被该 16 比特的前缀所覆盖，因此该虚拟前缀表项的 C_Total 加 1；(2)将 P_4 的前 32 个比特插入到 Hash(32)中，由于 P_4 被该前缀表项所覆盖，且 P_4 的前缀长度为 35，介于 $L+1$ 到 $L+15$ 之间(此时 $L=32$)，因此 P_4 的 C_Zone 加 1；(3)将 P_4 的最后 3 个比特采用步长为 4 的 Tree bitmap 存储，并将该树位图的根节点入口地址存储在 Hash(32)中相应表项的位图入口字段中。

重复上述操作，表 3 的路由表项插入完成后，形成的存储结构如图 5 所示。

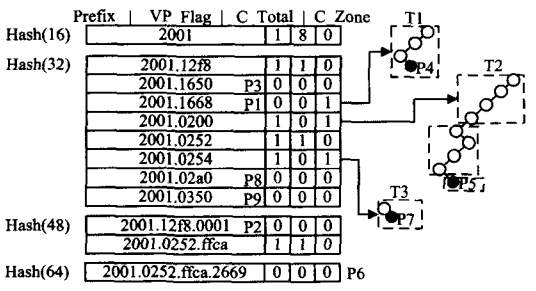

图 5　两级存储结构

2.6　查找过程

IP 地址查找分为两个阶段：第一阶段进行 Hash 表的查找。考虑到前缀的分布规律，以长度为 32 比特的 Hash(32)表为入口，利用二分法进行 4 个 Hash 表的查找，则可以直接获得最佳匹配前缀。或者将 IPv6 地址查找限定在一个起始长度为 S(其中 $S=16,32$ 或 48)，长度为 $L=15$ 的区间 Zone(S,L)中；第二阶段在区间 Zone(S,L)中按树位图方式进行地址查找。

第一阶段：在 Hash 表中进行二分查找。以 Hash(32)表为入口，利用二分法进行 4 个 Hash 表的查找。对于 Hash(L)的查找按照如下方法进行：

(1)如果 Hash(L)中匹配成功且匹配的表项中 C_Total＞0，则表明可能有更长的匹配前缀存在于 Hash$(L+16)$中，此时将待查找的 IP 地址在 Hash$(L+16)$中做试探性匹配测试。如果匹配成功，则跳转到 Hash$(L+16)$中进行查找，否则表明没有比 $L+15$ 更长的前缀可以和该 IP 地址匹配，因此应该执行(2)。

(2)如果 Hash(L)中匹配成功且匹配的表项中 C_Total＝0，则最长匹配的前缀长度应该小于等于 $L+15$。

A. 如果 C_Zone＞0，则表明在区域 Zone$(L,15)$中可能有更长的前缀匹配跳转到该表项所描述的 Tree bitmap 根节点中进行第二阶段的查找；

B. 如果 C_Zone＝0，则表明没有长度大于 L 的前缀可以匹配，则 Hash(L)中的匹配项为最长匹配前缀 BMP，返回该匹配表项中的下一跳信息，并完成查找。

(3)如果 Hash(L)中没有匹配项，则跳转到 Hash$(L-16)$中进行查找。

第一阶段的查找可以直接获得下一跳信息，或者将 IPv6 地址查找限定在一个起始长度为 S($S=16,32$ 或 48)、区间长度为 $L<16$ 的区域 Zone(S,L)中。以 Hash(32)为入口，对 Hash(16)，Hash(32)，Hash(48)及 Hash(64)进行二分查找的最大查找次数为 3。另外，由于路由前缀特有的分布规律，大部分的 IP 查找可以在第一阶段完成。

第二阶段：在区间 Zone(S,L)中按位图方式进行地址查找。对于第二阶段的查找，基本的查找方法与传统的 Tree bitmap 算法相同。由于该 Tree-bitmap 表示的前缀最长为 15 比特，因此在步长为 4 的 Tree bitmap 算法中，最大查找次数为 4。

以 IPv6 地址 2001：0254：8120::1101 的查找为例，首先将该 IP 地址的前 32 比特在 Hash(32)中进行 Hash 查找，结果匹配成功，则将该匹配设置为当前的最长匹配前缀；因为该匹配项的 C_Total＝0，所以最长匹配前缀的长度应该小于 48；由于该匹配项的 C_Zone＞0，表示可能有长度大于 32 且长度小于 48 的匹配前缀，因此以该表项中 Bitmap_Entry 为入口，在多比特树 T_3 上进行 Tree bitmap 的查找，可以找到最长匹配前缀为 P_7。

以 IPv6 地址 2001：0120::010A 为例，首先将该 IP 地址的前 32 比特在 Hash(32)中进行 Hash 查找，结果匹配不成功，则表明没有长度大于等于 32 比特的前缀与该 IP 地址匹配，因此跳转到 Hash(16)中进行查找；使用该 IP 地址的前 16 比特在 Hash(16)中进行查找，结果匹配成功，因为该匹配项的 C_Zone＝0 且 VP_Flag＝1，所以该最长的匹配前缀是一个虚拟前缀。至此，可以看出在路由表中没有匹配的前缀表项。

2.7　路由表的更新

路由表项 $P(p,len)$ 的更新主要包括 3 种操作：插入、删除和修改。对于前缀插入操作，可以按照路由表的创建算法进行插入。对于前缀修改和删除操作，则类似于地址查找的过程，首先利用查找算法，找到该前缀表项，然后执行相应的修改和删除操作，可以分为以下两种情况：

(1)当前缀 $P(p,len)$ 的长度为 16，32，48 或者 64 比特

(下转第 80 页)

2002/168
[13] Thomas T, Lal A K. Group Signature Scheme Using Braid Groups[DB/OL]. http://arXiv.org/cs.CR/0602063
[14] Zou S H, Zeng J W, Quan J J. Designated Verifier Signature Scheme Based on Braid Groups[DB/OL]. http://eprint.iacr.org/2006/329
[15] Verma G K. Blind Signature Schemes over Braid Groups[DB/OL]. http://eprint.iacr.org/2008/027
[16] Verma G K. A Proxy Signature Scheme over Braid Groups[DB/OL]. http://eprint.iacr.org/2008/160
[17] Zhang L L, Zeng J W. Proxy Signature Based on Braid Group [J]. Journal of Mathematical Study, 2008, 41(1): 56-64
[18] Lal S, Verma V. Some Proxy Signature and Designated Verifier Signature Schemes over Braid Groups[DB/OL]. http://arXiv.org/cs.CR/09043422
[19] Chaum D, Van E H. Group Signatures[A]// Proceedings of Europcrypt'91, Lecture Notes in Computer Science[C]. Springer-Verlag, 1991, 547: 257-265
[20] 刘文远,宋高效. 高效可撤销成员的不可链接的群盲签名方案[J]. 计算机科学,2008,35(11):60-62
[21] 全俊杰,曾吉文,邹时华. 基于 MSP 秘密共享的(t,n)门限群签名方案[J]. 数学研究,2008,41(1):65-71
[22] 于宝证,徐枞巍. 对一类群签名方案的伪造攻击[J]. 电子与信息学报,2009,31(1):246-249
[23] 禹勇,许春香,周敏,等. 对两个提名代理签名方案的密码学分析[J]. 电子与信息学报,2009,31(5):1218-1220
[24] 蔡永泉,刘岩. 一种基于身份信息无可信中心无随机预言的群签名方案[J]. 电子学报,2009,37(4A):87-91

(上接第 39 页)

时,则在 Hash(16),Hash(32),Hash(48)及 Hash(64)中,修改前缀 P 的子前缀 $P_s(P,16)$,$P_s(P,32)$,$P_s(P,48)$,$P_s(P,64)$表项的 C_Total 及 C_Zone。如果 VP_Flag=1 且 C_Total=0,则将该表项置为无效。

(2)如果 *len* 不能被 16 整除,则首先在 Hash(16),Hash(32)及 Hash(48)中,修改前缀 P 的子前缀 $P_s(P,16)$,$P_s(P,32)$,$P_s(P,48)$表项的 C_Total 及 C_Zone。如果 VP_Flag=1 且 C_Total=0,则将该表项置为无效,然后通过树位图的入口地址,在其对应的树位图中执行相应的修改和删除操作。

3 实验及性能分析

3.1 虚拟前缀数量

在基于 Hash 表和树位图的两级 IPv6 地址查找算法中,由于前缀之间大量存在覆盖关系,较长前缀可以借助 Hash 表中的前缀表项来表示该前缀 P 中能被 16 整除的部分,因此会在 Hash 表中引入一定数量的虚拟前缀 VP。实验中采用的路由表信息来自网络 potaroo[9]上的统计信息,其中长度为 16,32,48,64 比特的实际前缀总数为 1734,引入虚拟前缀之后,长度为 16,32,48 及 64 比特的前缀总数为 1841,增加的前缀数目为 107 个,所占的比例为 6.17%。同时,由于前缀之间的覆盖关系密切,相对于总前缀数目而言,引入的虚拟前缀的比例为 107/1917,约为 5%左右,如图 6 所示。

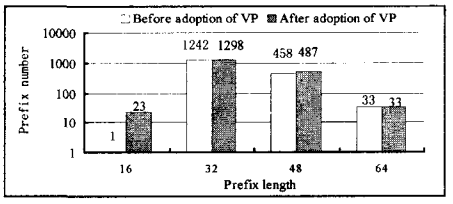

图 6 算法引入的虚拟前缀数目

3.2 两级地址查找的效率

为了评估本文 IP 地址查找算法的效率,本文使用随机和非随机两种策略产生 IPv6 测试地址,生成脚本采用了 ipv6gen[10]。表 4 中列出了采用 4 种 IP 测试数据所得到的地址查找次数和平均查找次数。可以看出,本文算法的平均查找次数介于 1～2 之间。尤其是当 IPv6 地址的分布规律与路由表分布规律类似时,算法的平均查找性能接近于 1。

表 4 平均查找次数

路由前缀数目	IP 地址数目	总查找次数	平均查找次数
1917	1917(随机)	3836	2.001
1917	19021(随机)	36369	1.912
1917	33124(随机)	61250	1.849
1917	43737(非随机)	55065	1.259

结束语 本文将按前缀长度二分查找与树位图查找的方法结合起来,将 IPv6 的地址查找分成两个阶段,第一个阶段直接使用 Hash 查找,第二阶段使用树位图查找。本文算法的主要优点为:1)由于采用两级查找,增加了查找算法的灵活性,两级算法可以相对独立,便于算法的升级;2)相对于按前缀二分查找算法来说,本算法引入的虚拟前缀数量较少,可以节约存储空间,提高了平均查找性能;相对于 Tree bitmap 算法而言,大大降低了树的高度,提高了查找效率;3)本算法平均查找次数为 1～2 次,最差查找次数为 7,查找效率较高。进一步的研究将采用并行的处理方法,同时对各个 Hash 表进行查找,从而进一步提高地址查找的效率。

参 考 文 献

[1] Fuller V, Li T, Yu J, et al. Classless Inter-domain Routing(CIDR): an address assignment and aggregation strategy (RFC 1519)[EB/OL]. ftp://ds.internic.net/rfc/rfc1519.txt, 1993
[2] Deering S, Hinden R. Rfc1883: Internet protocol, version 6 (ipv6) specification [EB/OL]. ftp://ds.internic.net/rfc/rfc1883.txt, 1995
[3] Sánchez M, Biersack E W, Dabbous W. Survey and taxonomy of IP address lookup algorithms[J]. IEEE Network, 2001, 15(2): 8-23
[4] Li Y K, Pao D. Address lookup algorithms for IPv6[J]. IEE Proceedings-Communications, 2006, 153(6): 909-918
[5] Waldvogel M, Varghese G, Turner J, et al. Scalable high speed IP routing lookups[C]// Proceedings of the ACM SIGCOMM'97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. ACM, 1997
[6] Eatherton W. Hardware-based internet protocol prefix lookups [D]. St. Louis: Washington University, 1999
[7] Eatherton W, Varghese G, Dittia Z. Tree bitmap: hardware/software IP lookups with incremental updates[J]. ACM SIGCOMM Computer Communication Review, 2004, 34(2): 97-122
[8] Huston G. Analyzing the Internet's BGP routing table[J]. The Internet Protocol Journal, 2001, 4(1): 2-15
[9] AS2-IPv6 BGP Table Statistics[EB/OL]. http://bgp.potaroo.net/v6/as2.0/index.html. 2009-06-18
[10] ipv6gen-IPv6 prefix generator[EB/OL]. http://techie.dev-null.cz/ipv6/ipv6gen. 2009-06-18