

# 多级索引路由查找算法及其实现

## A Multi-level Index Lookup Algorithm and Its Implementation

闫新成/YAN Xin-cheng

(中兴通讯股份有限公司 中心研究院, 江苏 南京 210012)  
(Central Academy of ZTE Corporation, Nanjing 210012, China)

中图分类号: TP393 文献标识码: A  
文章编号: 1009-6868 (2006) 02-0045-05

**摘要:** 路由查找是IP网络传输中或者基于IP构建的通信分组网中的重要组成部分。多分枝trie树查找算法是一种快速高效的路由查找算法,但同时也带来了巨大的内存开销。一种改进的多分枝trie树查找算法,即多级索引路由查找算法,将原有算法中的二级索引扩展为多级索引,并引入了标志位连续存储的方式,在基本不影响查找效率的前提下,极大地减小了路由索引表的内存开销。

**关键词:** 路由; 查找算法; trie树; 多级索引

**Abstract:** Routing lookup is an important technology for IP transmission network and IP-based packet switching network. Although the multi-branch trie tree lookup algorithm is a fast and efficient routing lookup algorithm, it also makes large memory consuming. A multi-level index lookup algorithm, i.e., an improved multi-branch trie tree lookup algorithm, is proposed. It expands the original two-level index to multi-level index, and introduces a continuous flag bit storage method that is able to extremely reduce memory overhead of routing index table while not obviously affecting the lookup efficiency.

**Key words:** route; lookup algorithm; trie tree; multi-level index

IP网络传输中或者以IP构建的通信分组网中,路由查找是极其重要的部分。本文介绍一种路由快速查找的软件算法,并针对该算法内存开销较大的问题提出改进方案及实现方法。多分枝trie树查找算法<sup>[1]</sup>是一种快速高效的路由查

找算法,通过IP地址的某一段比特位依次查找一个trie树结构的索引表,找到对应的路由。该算法虽然查找效率极高,但内存开销较大,一种改进算法是加入索引压缩机制,称为二层多分枝压缩trie树算法<sup>[2]</sup>。压缩后的索引在一定程度上减少了内存开销,但是其内存开销仍很大,同时由于索引项数与添加的路由掩码长度等特征有关,使得人们不能预知索引表的内存大小,很难适用于内存池集管理方式。本文首先介绍该算法的原理及其实现,其后给出改进后的算法,最后给出性能比较。

### 1 路由查找的最佳匹配准则

下面先给出几个路由相关的称谓,同时给出它们的数学表达:

(1)如果IP地址 $a$ 在路由 $A$ 的索引范围之内,则称路由 $A$ 是地址 $a$ 的路由,即路由 $A$ 匹配地址 $a$ 。

数学表达为:

if  $a \& A_{pfx\_len} = A_{addr}$ , then  $a \in A$

其中 $A_{addr}$ 、 $A_{pfx\_len}$ 分别表示路由 $A$ 的网络前缀和掩码; $a \in A$ 表示地址 $a$ 属于路由 $A$ ,即 $A$ 是地址 $a$ 的路由。默认路由是任意IP地址的路由。

(2)如果路由 $A$ 的索引范围包含路由 $B$ 的索引范围,则称路由 $A$ 为路由 $B$ 的次匹配路由。

数学表达为:

if  $A_{pfx\_len} > B_{pfx\_len}$  and  $B_{addr} \& A_{pfx\_len} = A_{addr}$ , then  $B \subset A$

其中 $B \subset A$ 表示 $B$ 是 $A$ 的一个子集,即 $A$ 包含 $B$ 。任何路由都是默认路由的子集。

(3)如果路由 $A$ 在所有 $a$ 的路由中掩码最长,则称路由 $A$ 是地址 $a$ 的最优匹配路由,即最长掩码路由,也是对地址 $a$ 进行有效转发的那条路由。当用地址 $a$ 进行路由查找时,结果应该是 $A$ 。

数学表达为:

if  $A_{pfx\_len} = \max(X_{pfx\_len}), X \in R_a = \{a \in A\}$ , then  $a \in A$

其中 $R_a$ 为一张路由表中所有 $a$ 的路由集合,如果 $a \in A$ ,则一定有 $A \in R_a$ ;  $a \in A$ 表示路由 $A$ 是地址 $a$ 的最优路由。

(4)如果路由 $A$ 在路由 $B$ 的所有次匹配路由中掩码最长,则称路由 $A$ 是路由 $B$ 的最长掩码次匹配路由。

数学表达为:

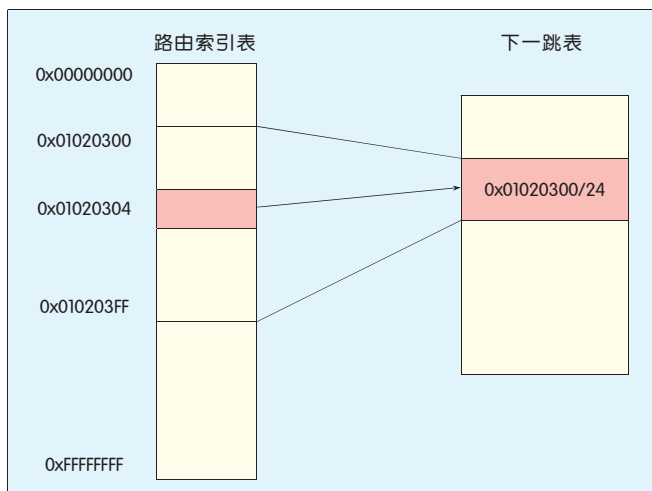
if  $A_{pfx\_len} = \max(X_{pfx\_len}), X \in \{B \subset A\}$ , then  $B \subset A$

其中 $B \subset A$ 表示路由 $A$ 是 $B$ 的最长掩码次匹配路由。

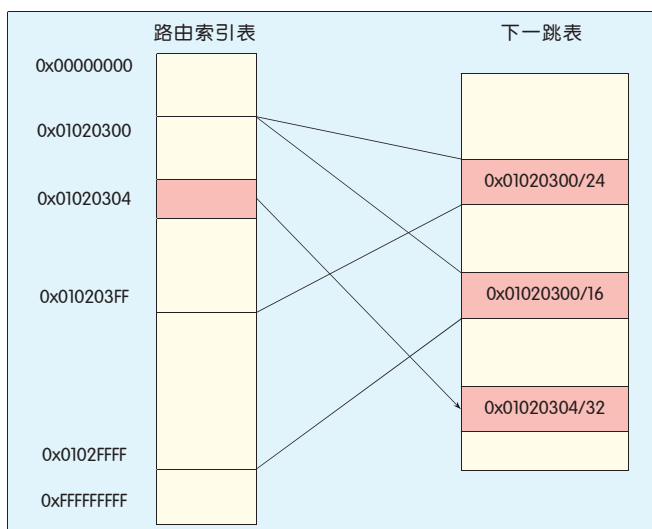
下面给出路由查找的最佳匹配准则,也称最长掩码匹配准则:查找一个地址对应的路由时,如果路由中存在多个路由满足条件,选用掩码最长的那条路由。

### 2 分段索引表的原理

分段索引表是根据地址的映射空间直接定位路由表,用索引来表达路由的前缀地址空间,从而建立地址与索引



▲图1 全IP地址映射索引表



▲图2 索引表的路由最长匹配示意图

一对多的对应关系。如果不考虑内存的问题,我们可以建立一个索引数组来表达IPv4的全地址空间,我们将这个索引表称为“全IP地址映射索引表”,如图1。整个路由表分为路由索引表和下一跳表两部分,左侧的路由索引表负责路由的快速查找,而右侧下一跳表存储路由的下一跳转发信息,也就是通常所说的路由表。索引表由4 294 967 296个索引表项组成,依次表示IPv4地址的从0x00000000到0xFFFFFFFF间的所有地址,索引表的每一项用于存储其对应的下一跳表的指针。通过IP地址查找路由时,只需将IP地址作为索引表的数组下标,取出对应的路由表项。例如,如果要查找IP地址1.2.3.4所属的路由,只需取出索引表的第0x01020304项,通过下一跳表,该索引表项便指向对应的路由0x01020300/24。每一个索引表项都唯一对应一项路由表(如果没有对应的路由,则指针为“空”),但一条路由往往对应多个索引表项,图1中路由0x01020300/24对应着从

0x01020300到0x010203FF的256项索引。

在全IP地址映射索引表中,路由对应的索引项数取决于路由掩码长度,如果该路由掩码长度为 $px\_len$ ,它将对应 $2^{32-px\_len}$ 项索引表。

当存在一个索引对应多条路由时,即这些路由的索引范围出现重叠,索引将选择掩码最长的那条路由与之对应,这便是索引表的路由最长掩码匹配实现。即索引机制中的最优匹配准则体现在路由的增删中,不影响路由的查找,这一点有别于其他一些路由查找算法(如radix树)。图2中,路由表中有3条路由由0x01020300/24、0x01020300/16和0x01020304/32都是地址0x01020304的路由,但由于第三条路由的掩码最长,最终将选用第三条路由。

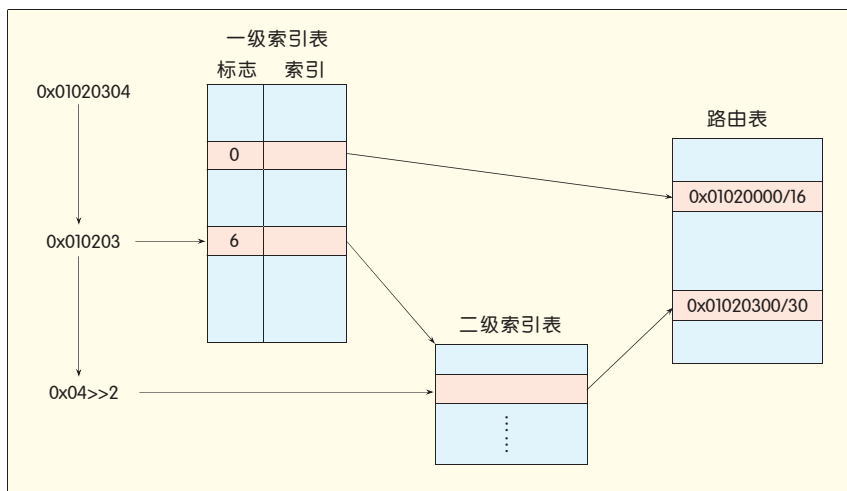
## 3 二级索引表

### 3.1 二级索引表的结构

全IP地址映射索引表无疑具有很高的查找效率(这种索引表的路由查找非常精确,是“定位”而非“查找”),但问题是索引表的内存消耗过大:索引表拥有4G个表项,每个索引表项存储4字节的下一跳表指针,那么整个索引表将占用16 GB的内存,这在目前几乎是不可能实现的。实际使用的路由表采用的是分级索引的方式,即通过多级索引表链接定位一条路由,这种算法来源于Gupta的多分枝trie树查找算法,其后Huang等人引入索引压缩机制,这种索引机制通常被称作二级索引。

二级索引表由两级索引组成(如图3所示)。第一级索引步长固定,其索引表项由两部分组成:指针和标志。指针采用复用的形式,一级索引表既可能直接指向路由表,也可能指向二级索引表,再由二级索引表指向路由表;标志用于表示二级索引表的步长,如果标志为0,表示索引表直接指向路由表。第二级索引采用索引压缩机制,根据路由掩码长度选择适当的步长(每级索引表所能表达的IP地址的长度称作该级索引表的步长),如果某级索引表步长为 $n$ ,则该级索引表便具有 $2^n$ 个索引表项。

为了描述简单,我们采用24-8的索引形式来说明,即一级索引表步长为24,二级索引表步长最大为8。仍以查找地址0x01020304为例,因为一级索引表的步长为24,所以取出地址的前3个字节0x010203作为一级索引表的数组下标;对应的索引表的标志为6,表示一级索引表的指针表项并不直接指向路由表,而是指向一个步长为6的二级索引表;由最低的1个字节去定位它在二级索引表中的偏移,因为该二级索引表的步长为6,所以偏移量为最后1个字节的高6比特,即 $0x04 \gg 2$ ,该位置存储的便是对应路由的指针。二级索引表总步长为30,则该二级索引指向的路由掩码最长为30,也就是说索引表的总步长不能小于路由掩码。如果路由的掩码长度小于一级索引的步长,便不需要添加二级索



▲图3 二级索引表路由查找示意图

引表,这时对应的一级索引表中标志为0,索引项直接指向路由表。

### 3.2 二级索引的内存开销

下面我们来讨论一下二级索引表需要占用的内存大小。以20-12的二级索引表为例,如果路由表最大支持1K条路由,则整个索引表最大耗用内存的情形是每条路由都有一个一一对应的最大的二级索引表指向它,则索引表需要的索引项数为: $2^{20}+2^{12} \times 1K=5M$ ,而前面说到的全IP地址映射索引表需要4G个索引表项,已然是天壤之别。显然分级索引极大的节省了内存空间,这是因为:

- 路由的掩码长度并不总是32;
- 程序支持的路由条目数是有限的。

也就是说,分级索引表之所以节省内存空间,主要在于它不必映射全部的IP地址空间,而只是按路由表中的路由映射部分IP地址空间,这样在一定程度上,便将地址的样本空间与路由样本空间相关联。如果用二级索引表来表达一个IPv4的全地址空间,需要每个一级索引都挂有一个最大的二级索引表,总计需要 $2^{20}+2^{12} \times 2^{20}=4\ 097M$ 个索引表项。所以分级索引之所以节省内存空间,主要在于它对IP地址的部分映射。

分级索引表沿用了通过IP地址直接定位路由表的形式,所不同的是定位次数,全IP地址映射索引表只需一次定位,二级索引最大需要两次定位,但这仍旧是直接定位的方式,相比而言,查找效率并未明显降低。

但是二级索引表仍旧不完全适合我们的系统,因为首先这种索引结构还是需要较大的内存空间:为了表达一条32位掩码的地址路由,需要申请 $2^{12} \times 4=16K$ 字节的内存空间作为第二级索引表,这就造成了极大的内存浪费;其次,第二级索引由于采用了压缩机制,虽然一定程度节省了内存,但也使得索引表的大小不可预知,很难适应内存池集

技术的内存管理方式。

## 4 多级索引表

### 4.1 多级索引表结构

多级索引是一种多层多分枝trie树,仍采用通过地址分段定位的方式进行路由表查找,但针对二级索引存在的内存问题做了如下改进:

- 采用多层索引表串联的形式定位路由表。根据上一节的分析,这种算法更加细化了IP地址空间,从而可以有效的减小索引表的内存开销。
- 将索引项由4字节的指针改为2字节的数组标号。

• 二级索引表中标志字段虽然申明为1个字节,但由于字节对齐的缘故,耗用了4个字节。如果将标志位连续存储,可以减小标志字段的实际内存开销。

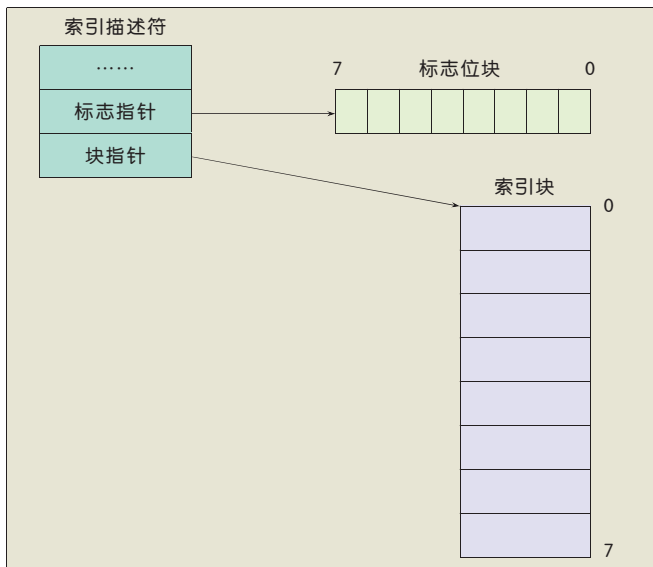
• 不采用索引压缩技术,即索引步长固定。如果索引步长较小,索引压缩技术节省内存开销的效果便不是那么明显了,相反还会使算法复杂化。同时固定步长的索引块不受内存管理方式的限制。

多级索引表是一种多分枝的trie树,由多级索引构成,同一级索引的步长相等,但不同级之间的索引步长不要求相等。IPv4路由表的各级索引表步长和为32。多级索引表的表项,我们称之为“索引项”,是一个16比特的序号。该序号是复用的,表示路由表的数组下标或者下一级索引的索引描述符表的数组下标。这里路由表为狭义路由表,指用来存储下一跳等信息结构体数组。由索引表项组成的数组称为“索引块”,与二级索引算法中的索引表相对应,用以表达路由前缀特定的几个比特位。不同的是多级索引中每一级的步长都是固定的(二级索引算法中第二级索引的项数是不定的),所以每级索引块的大小是固定的,这使得每个索引块耗用的内存空间是预知的,同时降低了一定的代码复杂度。

由于索引项的序号是复用的,所以需要1比特的标志位来区分索引项的含义。如果这个标志作为索引表项的一个成员存在,那么它将至少占用1个字节(还需要考虑字节对齐问题)。

为了节省空间,我们为每个索引块关联一个标志位块,如图4,标志位块的每一个比特位与索引块的每一项在逻辑上一一对应,用以区分索引项表征的内容。如果索引块有8项,对应的标志位块的大小便为1字节,即8比特。

为了加强标志位块与索引块的关联,将它们的起始地址放到同一个结构体中,称该结构体为“索引描述符”。如图4,索引描述符的两个指针成员:标志指针和块指针分别



▲图4 标志位与索引的对应关系

指向标志位块和索引块,而标志位块中的每一位与索引块中的每一项一一对应,例如,索引块中的第5项中索引的含义由标志位块中第5比特决定。引入索引描述符后,便可以通过索引描述符间接定位索引块及其对应的标志位块。标志位块是一个用字节形式存储的数组,如果已知地址在索引块中的偏移,即数组下标,将下标除以8,即得到对应的标志位块的下标,而下标对8取余,便可得对应的比特。

每级索引都可能包括多个描述符,这些描述符以数组的形式存在,称之为“索引描述符表”。每一级索引都有一个唯一的索引描述符表,用以存储本级的所有描述符。因为可以通过索引描述符来定位索引块,所以当要连接下一级索引块时,可以记录索引块所对应的描述符在索引描述符表中的下标,如图5。多级索引算法通过描述符表与索引块、标志位块相互链接定位一条路由:

- 通过索引描述符表定位索引块及其对应的标志位块；
- 通过地址和索引表对应的比特位计算在该级索引块中的偏移,取出序号,根据对应标志位是否置位判决是否为路由表序号；
- 如果索引中存储的是次级索引表的序号,那么就以此序号作为次级索引描述符表的数组下标,取出相应的索引描述符,重复以上步骤。

相比图3,显然多级索引表更为复杂,不但增加了索引级数,而且在各级索引表的连接中增加了描述符节点,该节点的引入主要由于将索引表(块)的每一项由指针改为索引的缘故。

图5中需要注意的是:

- 为了结构与代码实现上的统一，第一级索引块和第一级标志位块也用索引描述符关联。第一级索引描述符表只有一项。

- 因为最后一项索引块存储的一定是路由表索引, 所以不需要标志位数组, 但仍用索引描述符表将索引块关联, 只是将描述符表的标志位块指针置为“空”。

- 索引描述符中存储的是索引块和标志位块的指针,而非数组实例。这样可以使索引描述符的使用更为灵活,因为其结构不受索引表步长的影响,各级索引表的描述符可以使用相同的结构,从而使得索引表的级数和步长可以定制。

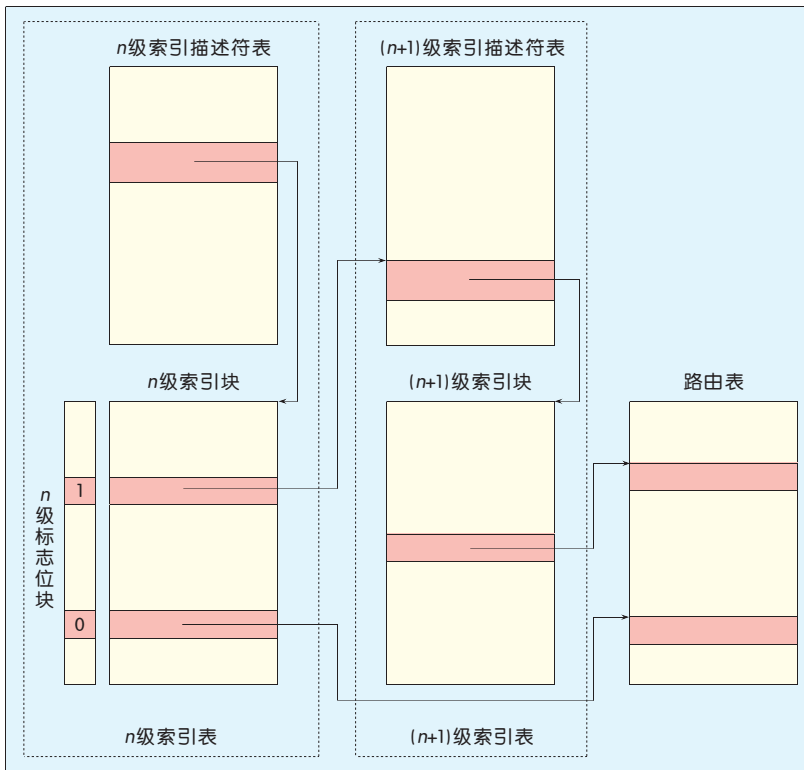
## 4.2 多级索引的开销

首先我们给出多级索引的最大内存开销公式:

$$L = 2^{x_0} + 2^{x_1} M \cdots \cdots 2^{x_n} M \quad (1)$$

$$x_0+x_1+\dots+x_n=32 \quad (x_i < 32, i < n < 32) \quad (2)$$

其中 $L$ 为索引表内存开销, $M$ 为路由条目数,路由级数为 $n+1$ ,各级索引的步长依次为 $x_0, x_1, \dots, x_n$ 。第一级索引表的大小与路由容量无关,所以第一级索引表的内存开销只与第一级索引步长 $x_0$ 有关,为 $2^{x_0}$ 项索引;而第二级、第三级……的内存开销与路由容量相关,最差的情形下,各级索引表(第一级索引表除外)的个数为路由表容量 $M$ ,所以 $L$ 表示的是索引表最大所需要的索引表的索引项数。使内存开销 $L$ 最小的最大内存开销问题便近似为求取以上公式的条



### ▲图5 多级索引关联



▼表1 多级索引最大内存开销

索引级数及索引长度	不同路由表容量下的最大内存开销/MB						
	1K	2K	4K	8K	16K	32K	64K
20,6,6	2.3	2.7	3.1	4.2	6.4	10.8	19.5
18,7,7	1.0	1.5	2.5	4.5	8.5	16.5	32.5
20,4,4,4	2.1	2.3	2.5	3.0	5.1	6.1	8.3
17,5,5,5	0.5	0.7	1.0	2.0	3.8	7.4	14.5
16,4,4,4,4	0.3	0.5	0.8	1.5	2.9	5.6	11.1

▼表2 多级索引最大内存开销

索引级数及索引长度	不同路由表容量下的最大内存开销/MB						
	1K	2K	4K	8K	16K	32K	64K
16,16	257	513	1025	2049	/	/	/
20,12	24	40	72	136	264	520	1032
24,8	33	34	36	40	48	64	96

▼表3 索引表查找周期数

索引级数及索引长度	不同路由条目数下的最佳匹配所需的时钟周期数					平均周期	平均时间/ $\mu$ s
	100	200	400	800	1600		
20,12	298	203	220	248	235	241	1.20
20,6,6	299	215	245	253	248	252	1.26
17,5,5,5	292	262	240	234	280	262	1.31
16,4,4,4,4	302	251	262	284	267	273	1.36

件极值。求得:

$$x_0 = \frac{32 + n \log M}{n+1} \quad (3)$$

$$x_0 = \frac{32 - \log M}{n+1} \quad (i=1, 2, \dots, n) \quad (4)$$

$M$ 表示以2为底数的路由表容量。将式(3)和式(4)变形一下,得到:

$$x_0 - x_i = \log M \quad (5)$$

不难看出,式(1)、式(2)所表达的含义为:如果路由表容量为 $M$ 时,当第一级索引步长比其余各级索引步长长 $\log M$ 时,索引表的最大内存开销最小。

根据公式(1),表1列出几种典型的索引配置在不同路由条目数下的内存开销。

表1中第一列表示索引级数及索引长度,如(20,6,6)表示索引表级数为3,各级索引表步长依次为20、6和6;第一行表示路由表容量,即路由表所能支持的路由最大条目数;表中数据表示在不同索引长度和索引表容量下索引表所需的最大内存开销,单位为兆字节。而二级索引表所对应的最大内存开销见表2,在相同的路由表容量下,二级索引表的最大内存开销要比多级索引表的大很多,显然多级索引表在最大内存开销方面的性能要远优于二级索引表。表2中“/”表示开销过大,无统计意义。

另外,从表1中可以看出,通常索引级数越大,最大内存开销便越小,但并不是说索引级数越大越好,因为伴随着索引级数的增大,路由查找的效率会有一定程度的降低,如表3所示,表中显示的是不同索引表(第一列)在不同路由条目数下路由最佳匹配所需的时钟周期数,测试环境为ARM公司主频为200 MHz的处理器。可以看出索引级数的增大,一定程度上会导致查找效率的降低,所以并非索引级数越长越好,而是要兼顾内存开销和查找效率,极端的,当索引级数为32时,多分枝trie树便退化成了二进制trie树。有关多分枝trie树和二进制trie树的查找性能可以查看文献[3]。

## 5 结束语

多级索引相比二级索引而言,索引级数较多,步长较短,因而可以很大程度地减少路由前缀地址在索引表中的映射空间,从而可以有效地节省内存。因为索引采用了索引序号复用的形式,相比指针节省了一半的索引存储空间。又因为引入了标志位块,有效地避免了存储上由于按字节存储、字节对齐等原因造成的实际占用内存远大于变量需求空间的问题。

因为多级索引的步长较小,而不必采用索引压缩机制,从而对内存的管理方式的依赖性较小,同时路由索引的级数和索引步长可以根据路由表容量灵活设定。

## 6 参考文献

- [1] Gupta P, Lin S, McKeown N. Routing Lookups in Hardware at Memory Access Speeds[C]// Proceedings of INFOCOM. Mar 29-Apr 2, 1998, San Francisco, CA, USA. Piscataway, NJ, USA: IEEE, 1998: 1240-1247.
- [2] Huang Nenfu, Zhao Shiming. A Novel IP-routing Lookup Scheme and Hardware Architecture for Multigigabit Switching Routers[J]. IEEE Journal on Selected Areas in Communications, 1999, 17(6): 1093-1104.
- [3] Srinivasan V. Fast and Efficient Internet Lookups[D]. Washington DC, USA: Washington University, 1999.

收稿日期:2005-12-27

### 作者简介



闫新成,东南大学信号与信息系统硕士研究生毕业,中兴通讯股份有限公司中心研究院南京研究所软件研发工程师,主要从事3G IP网络的研究和产品开发。