基于 ASIC 实现的高速可扩展并行 IP 路由查找算法

谭明锋, 龚正虎
(国防科技大学计算机学院, 湖南长沙 410073)

摘 要: 本文提出的IP路由查找算法基于 ASIC 实现, 用多个Hash函数对不同长度的前缀进行映射并保存在不同的组相联存储器中, 运用组相联存储器的特性很好地解决了Hash碰撞, 并极大地减少了空间耗费. 查找时并行查找所有存储器以进行最长前缀匹配, 可在一次访存时间内完成查表, 而路由更新平均只需数次访存. 该算法在使用 10ns 的存储器件时已可满足 OC2768 接口的线速转发要求, 而且具有良好的可扩展性和并行性, 可满足更大容量的路由表和更高速度网络单元的线速转发要求.

关键词: 专用集成电路 (ASIC); IP 路由查找; 可扩展性; 并行性; OC768 接口; 线速转发
中图分类号: TP393 文献标识码: A 文章编号: 0372-2112 (2005) 02-0209-05

High Speed IP Lookup Algorithm with Scalability and Parallelism Based on ASIC Implementation

TAN Ming-feng, GONG Zheng-hu
(Institute of Computer science, National University of Defense Technology, Changsha, Hunan 410073, China)

Abstract: This paper proposes a high performance IP routing lookup algorithm based on ASIC implementation. It keeps prefixes of different length in different group associated CAMs, and uses different Hash functions to map the prefixes into the corresponding groups of the CAMs. By this means it reduces the Hash collisions and the memory usage. This scheme can finish the lookup within 1 memory access time and need only few memory accesses for each update in average. With 10ns memory, this scheme can fully match the link speed of OC2768. For its good scalability and parallelism, it can be extended to adapt larger forwarding tables and faster forwarding requirements.

Key words: application specific integrated circuit (ASIC); IP routing lookup; scalability; parallelism; OC768 interface; link-speed forwarding

1 引言

Internet 的网络规模的高速增长对 IP 路由查找算法处理大容量路由表的适应性提出了更高要求; 而 1997 年后 Internet 的网络流量和带宽以及骨干网路由器接口速率约每 6 个月增长一倍[1], 骨干路由器每秒所需转发的报文数随之剧增. 因此作为路由器转发能力的关键因素之一, IP 路由查找算法的优劣直接影响了当前和未来 Internet 网络的整体性能. 本文提出的基于 ASIC 实现的高速可扩展并行 IP 路由查找算法, 突破了现有算法以存储空间和路由更新性能为代价换取路由查找性能的一般模式, 充分运用了 Amdahl 定律, 利用 IP 转发表的分布特征和现有器件的功能特性, 降低了对存储空间的要求, 使算法的数据结构可被放入更小并且更快的存储器中. 不但能够做到高速查找而且能够高速更新. 该算法具有良好的可扩展性和并行性, 能够适应大容量路由表, 且易于并行和流水实现.

2 现有算法简析

IP 路由查找的经典算法是基于二叉 Trie 树的软件算法, 如 Radix Trie[2], Patricia 算法[3]以及 Sklower[4]对 Patricia 算法的改进算法等, 但此类算法在最差情况下至少需要访存 32 次. 事实上对于所有基于树的算法, 树高决定了访存次数, 从而决定了算法的时间性能. 因此, 一些算法采用了路径压缩[3]、leaf-pushing[5]、前缀扩展[5]以及多分支等优化技术来降低树高. 例如多分支 Trie 树算法的典型例子 LC-Trie 树算法[6]、受控前缀扩展算法[5]等. 但这些算法更新时间复杂度高, 耗费的空间大, 并且树的特点决定了这些算法仍然需要多次访存.

利用 TCAM (Ternary CAM) 的硬件算法[7]将较长的前缀保存在低地址表项中, 搜索时 TCAM 并行匹配所有保存的路由项, 并选取地址最低的匹配表项以进行最长前缀匹配. TCAM 具有查找速度快的优点, 但是它也具有功耗较大、更新复杂, 单位 bit 昂贵、容量较小等不足[11].

24-8-DIR[8]算法是另一种典型的硬件算法, 它实际上是硬件实现的两级多分支前缀扩展算法: 第一级有 2^{24} 个分支, 若有二级节点, 则该一级节点有 2^8 个二级子节点. 该算法主要基于前缀长度的分布: 99.93% 前缀的长度都小于等于 24[9]. 因此它将长度小于 24 的所有前缀全部展开为 24 位前缀, 然后用前缀或 IP 地址的高 24 位作为存储器地址访问路由表, 一般只需要一次访存即可找到目标路由, 所以是 / 以存

收稿日期: 2003-06-30; 修回日期: 2004-09-08
基金项目: 国家重点基础研究发展计划 (973 项目) / 新一代互联网路由与交换理论 0 (No. 2003CB314802)

储器速度进行路由查找0的算法. 但该算法每次更新都需要大量访存, 而且需要大量的存储空间, 因此难以将其数据结构放到更小但更快的存储器中. 而如果使用 SRAM 作为存储器来提高速度, 成本很高.

一些利用 Cache 的算法努力让更多的访存落到 Cache 中以减少搜索的时间, 如 Lulea 算法[10] 以及 Chiueh[11] 和 Degermark[12] 所提出的算法等. 根据 Ruiz2Sanchez[13] 对纯软件算法和基于 Cache 的算法的模拟, 基于 Cache 的算法在空间耗费和搜索时间两方面都相对具有较好的性能. 但由于 Lulea 算法使用了压缩技术和 leaf pushing 技术, 导致更新时要求重建整个数据结构. 并且这些在进行路由搜索时需要解压缩, 所以虽然它们的访存的次数少, 但是平均每次路由查找需执行约 200 条指令.

通过对已有算法和实际需求的分析可知, 理想的 IP 路由查表算法应满足下列要求: (1) 高速查找; (2) 内存需求小; (3) 更新时间短; (4) 能灵活实现; (5) 能处理真实的大容量路由表; (6) 预处理时间短. 而目前已有的算法通常都着重考虑了算法的搜索性能, 对更新性能和空间性能上的考虑相对较少. 尤其是为了提高搜索性能, 所使用的优化技术越来越复杂, 导致路由更新相当困难甚至要求重构整个路由表, 而且实现复杂性也比较高.

3 算法描述

算法的设计目标是: (1) 空间耗费少, 数据结构能被放入更小更快的存储器; (2) 每次查表在一次访存时间内完成; (3) 每次更新平均在数次访存时间内完成; (4) 易于扩展, 能处理真实的大容量路由表; (5) 易于流水和并行实现.

为描述方便, 定义前缀长度为 PLen, 并称 IP 地址的最高位为第 1 位, 最低位位第 32 位. 此外定义 Hash 函数 $h_{x,y}(IP)$, 该函数的结果是 IP 地址的第 x 位到第 y 位. 而 Hash 函数 $f_{x,y}(IP)$ 的映射结果是未被 $h_{x,y}(IP)$ 使用的位. 当这两组函数作用在前缀上时, 忽略超出前缀长度的位.

311 基本思想

首先换一个角度考察 24- 8 DIR 算法. 实际上该算法首先进行了前缀扩展, 然后用 Hash 函数 $h_{1,24}$ 截取了前 24 位作为地址去访存. 该函数的值域为 $\{i \mid 0 \leq i \leq 2^{24}-1\}$, 共需 16M 项的存储空间. 但目前转发表的容量一般小于 200K, 所以这种映射过于稀疏, 空间浪费严重. 本文提出的算法采用了不同 Hash 策略, 使用更高效的映射减小值域, 从而避免稀疏映射, 提高空间利用率.

Amdahl 定律告诉我们: / Make the common case fast0. 我们知道, 转发表中绝大部分前缀的长度都介于 8 和 24 之间, 因此首先考虑对其的处理. 首先将长度为 8~ 24 的前缀分别放入 17 个的存储器, 并且为每个存储器关联一个 Hash 函数. 查找时这 17 个 Hash 函数为目标 IP 地址并行计算各个存储器中对应的地址, 然后并行访问各个存储器中对应的单元, 最后通过仲裁器选取其中最长的前缀作为最终结果.

算法使用组相联 CAM 来减少 Hash 碰撞. 例如使用 4 路组相联 CAM 保存长度为 24 前缀和下一跳信息. 因此 CAM 中的每个项需要保存如下信息: (1) $f_{x,y}$(Prefix), 这些位将用于对组中的各项进行精确匹配; (2) 15 位的下一跳索引. 转发表中的下一跳数目通常较少, 15 已足够对其编码.

如图 1 所示, CAM24 只保存 24 位的前缀. IP 地址到达时, 17 个 Hash 函数为 17 个 CAM 并行地进行地址计算. 例如在 CAM 24 通过 $h_{x24,y24}(IP)$ 计算得到组地址, 然后通过第一个仲裁器把这 4 个项中保存的 4 个 $f_{x24,y24}$(Prefix) 与 $f_{x24,y24}(IP)$ 进行比较, 发现第 3 个项完全匹配, 因此获得其下一跳信息 NHC. 如果其在它 CAM 中也找到了匹配的前缀, 则通过下一个仲裁器选择最长前缀匹配.

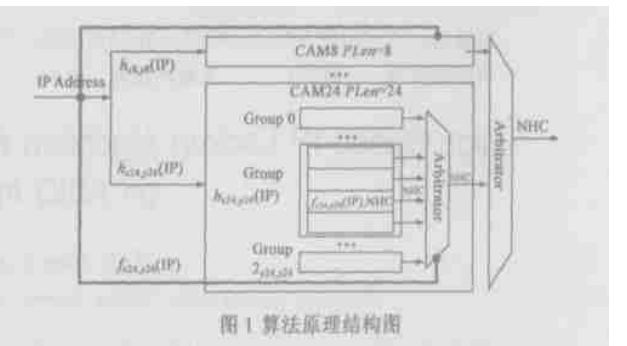

图 1 算法原理结构图

312 例外前缀处理

上述过程中, 有两种例外的路由项未被处理: 其一是少量长度小于 8 和大于 24 的前缀; 其二是在添加新的路由项时, 相应 CAM 组内所有项已被全部占满而出现的溢出前缀. 通过实验我们发现适当的 Hash 函数能够使两种例外前缀的总数少于 1K.

如图 2 所示, 算法引入一个小 TCAM 来处理这些/ 例外0 前缀, 小的 TCAM 速度快、价格低、功耗小, 而且它的更新算法可以根据/ 小0这一特点进行优化. 在添加新路由时时, 只有在少数情况下才会对 TCAM 进行操作; 在搜索时, 17 个 CAM 和 TCAM 并行搜索, 对搜索性能无影响. 综合两方面的因素, 引入 TCAM 对于算法的整体性能影响很小.

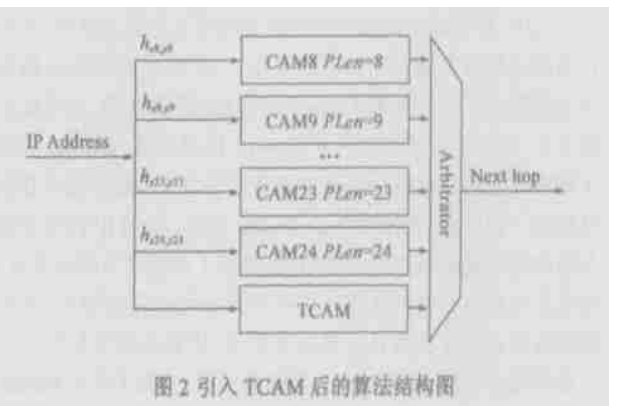

图 2 引入 TCAM 后的算法结构图

313 Hash 函数及其他算法参数的选取

Hash 函数决定了空间性能. 24- 8 DIR 算法需要大量空间的原因就在于它使用的 hash 函数是 $h_{1,24}$. 而本文的算法用不同的 Hash 函数直接提取若干位作为不同 CAM 的访存地址. 但是仅对于长度为 PLen 的前缀集, 就共有 $\sum_{i=1}^{PLen} P_{PLen}^{i}$ 个可选

Hash 函数. 如 PLen= 15 时, 可选映射的数目高达 3380, 558, 065, 因此我们只能根据路由表前缀分布的一般特征来选取.

优秀的 Hash 函数能够把前缀尽量均匀地映射到 CAM 内的各个组上. 而理论上好的 Hash 函数其结果的熵也越大, 因此定量判定 Hash 函数的好坏的方法是比较它们的熵. 例如有两个 Hash 函数都提取两个位作为其结果, 那么其结果只能是 0, 1, 2 和 3. 令3 r, s4 表示有 s 个前缀被映射到值 r 上, 并假设这两个 Hash 函数对 16 个前缀的 Hash 结果分别是 {3 0, 4431, 3432, 543 3, 44} 和 {30, 243 1, 1432, 943 3, 44}, 从直观上我们可以判定第一个 Hash 函数要好于第二个. 而第一个 Hash 函数的熵是: $\frac{4}{16}\log_2\frac{16}{4}+\frac{3}{16}\log_2\frac{16}{3}+\frac{5}{16}\log_2\frac{16}{5}+\frac{4}{16}\log_2\frac{16}{4}$ U 1. 98, 第二个的熵是 1. 87, 这个结果和直观判断吻合.

表 1 显示了三个 Hash 函数的熵. 从表中中可知, Hash 函数应选取靠近前缀长度的位作为其结果, 实际上这是因为不同的前缀, 越靠近前缀长度的那些位差异性越大. 表 2 给出了些函数的 Hash 结果, 可以看到两个表中的结果相符. 本文中所使用的 12 个随机选取的路由表中, 1~ 4 号来自 mae2west 的路由表[14], 5~ 8 号来自 aads 的路由表[14], 9~ 12 号为 Cernet 的 Global 路由表[15].

表 1　Hash 函数的熵, Hash 对象: 真实路由表中长为 24 的前缀

$h_{x,y}$	Hash 函数的熵, PLen= 24		
	路由表 1, 共 28527 个前缀, 其中有 15160 个 24 位前缀	路由表 5, 共 30836 个前缀, 其中有 15874 个 24 位前缀	路由表 9, 共 112077 个前缀, 其中有 62666 个 24 位前缀
$h_{1,16}$	4. 8183	4. 79817	5. 00457
$h_{5,20}$	9. 02572	9. 05862	9. 29941
$h_{9,24}$	11. 3117	11. 3425	11. 7535

表 2　多个 Hash 函数的映射结果

$h_{x,y}$	Hash 映射结果, PLen= 24 这里(x, y) 指共有 y 个 Hash 桶高度为 x		
	路由表 1	路由表 5	路由表 9
$h_{1,16}$	54 种不同的桶高, 最大桶高 103	52 种不同的桶高高度, 最大桶高 86	121 种不同的桶高, 最大桶高 186
$h_{5,20}$	(1, 3851) (2, 1463) (3, 663) (4, 393) (5, 196) (6, 129) (7, 84) (8, 67) (9, 29) (10, 33) (11, 16) (12, 17) (13, 12) (14, 14) (15, 7) (16, 30) (18, 2)	(1, 3995) (2, 1554) (3, 699) (4, 418) (5, 210) (6, 131) (7, 91) (8, 66) (9, 29) (10, 35) (11, 17) (12, 19) (13, 11) (14, 13) (15, 10) (16, 29) (18, 2)	(1, 6683) (2, 3493) (3, 2080) (4, 1605) (5, 1059) (6, 771) (7, 574) (8, 493) (9, 365) (10, 270) (11, 164) (12, 117) (13, 120) (14, 132) (15, 106) (16, 181) (17, 22) (18, 20) (19, 9) (20, 6) (21, 4) (22, 4) (23, 3) (24, 1) (25, 2) (27, 1)
$h_{9,24}$	(1, 12112) (2, 1361) (3, 106) (4, 2)	(1, 12507) (2, 1482) (3, 133) (4, 1)	(1, 24618) (2, 11849) (3, 3601) (4, 725) (5, 117) (6, 8) (7, 2)

表 3　算法 Hash 函数和其他参数的选取

PLen	Hash 函数及其他参数				
	前缀数目	Hash 函数	桶高分布	CAM 组相联数	CAM 大小(字节)
8	17	$h_{1,8}$	(1, 17)	1	256@3
9	6	$h_{2,9}$	(1, 6)	1	256@3
10	7	$h_{3,10}$	(1, 7)	1	256@3
11	12	$h_{4,11}$	(1, 10) (2, 1)	1	256@3
12	34	$h_{5,12}$	(1, 30) (2, 2)	1	256@3
13	85	$h_{16,13}$	(1, 63) (2, 8) (3, 2)	1	256@3
14	231	$h_{7,14}$	(1, 103) (2, 41) (3, 14) (4, 1)	2	2@256@3
15	414	$h_{7,15}$	(1, 190) (2, 82) (3, 16) (4, 3)	2	2@512@3
16	7225	$h_{4,16}$	(1, 3880) (2, 1381) (3, 189) (4, 4)	4	4@8192@3
17	1414	$h_{7,17}$	(1, 750) (2, 264) (3, 37) (4, 5) (5, 1)	4	4@2048@3
18	2573	$h_{7,18}$	(1, 1459) (2, 420) (3, 80) (4, 6) (5, 2)	4	4@4096@3
19	7517	$h_{7,19}$	(1, 3174) (2, 1410) (3, 382) (4, 78) (5, 13)	4	4@8192@3
20	7174	$h_{8,20}$	(1, 2895) (2, 1316) (3, 386) (4, 94) (5, 19) (6, 3)	4	4@8192@3
21	5109	$h_{9,21}$	(1, 2814) (2, 890) (3, 150) (4, 15) (5, 1)	4	4@8192@3
22	7758	$h_{10,22}$	(1, 3026) (2, 1462) (3, 443) (4, 106) (5, 11)	4	4@8192@3
23	9410	$h_{10,23}$	(1, 5434) (2, 1534) (3, 259) (4, 29) (5, 3)	4	4@16384@3
24	62666	$h_{9,24}$	(1, 24618) (2, 11849) (3, 3601) (4, 725) (5, 117) (6, 8) (7, 2)	4	4@65536@3
小结	长度小于 8 和大于 24 的前缀数目		: 380		
	溢出前缀数目		: 293		
	TCAM 中需保存的前缀数目		: 380+ 293= 673		
	TCAM 容量		: 1K 项		
	17 个 CAM 总的大小		: 1557504 字节		

一个无碰撞的Hash 函数可将大小为 N 的转发表映射到地址宽度为 $\log_2 N$, 大小为 N 的存储器中. 但事实上这样的理想函数难以获得, 因此需要为算法确定适当的参数. 如表 3 所示, 9 号路由表共有 62666 个长度为 24 位的前缀, 因此 CAM24 的地址宽度位$\lceil \log_2(62666) \rceil = 16$, 所以 CAM24 的Hash 函数是 $h_{9,24}$, 此时仅有 7 种 Hash 桶高度. 算法使用 4 组相联 CAM 作为 CAM24, 则有(5- 4) * 117+ (6- 4) * 8+ (7- 4) * 2= 139 个从 CAM24 中溢出的前缀. 所有从各个 CAM 中溢出的前缀和短于 8 位、长于 24 位的前缀都被保存在 TCAM 中. 对于 24 位长的前缀, 它们的第 1 位到第 8 位以及下一跳信息被保存在 CAM24 中, 因此只需要 8+ 15= 23 位即可. 所以 CAM24 共需要(2^{16}组* 4 项/ 组* 3 字节/ 项) 的容量.

用类似的方法求出其他 CAM 所需的算法参数, 并最终可确定 TCAM 的容量. 从表中可知, 17 个 CAM 仅仅需要 1557504 字节, 而所需的 TCAM 的容量少于 1k 项. 对于典型的 128k 大小的真实转发表, 表 3 中的算法参数已足够满足其需要. 而对于更大的转发表, 容易利用算法的可扩展性满足其需求.

3.4 路由更新

添加一条前缀为 Prefix 的路由:

(1) 按前缀长度选择相应 CAM, 用相应 Hash 函数获得 CAM 地址 h(Prefix).

(2) 如果 CAM 地址所指向的组中还有空闲项, 则将 f(Prefix) 和下一跳信息保存在空闲项中并结束添加.

(3) 否则将路由项保存到 TCAM 中.

删除路由的过程类似:

(1) 按前缀长度选择相应 CAM, 用相应 Hash 函数获得 CAM 地址 h(Prefix). (2) 同时从相应的 CAM 和 TCAM 中删除相应路由项, 只会有一个删除成功.

表 4 TCAM 中的路由项数目

路由表	1	2	3	4	5	6	7	8	9	10	11	12
TCAM 中路由项数目/ 路由项总数	28/ 28527	28/ 27884	32/ 29025	32/ 30421	420/ 30836	317/ 31509	251/ 32174	249/ 30013	673/ 112077	692/ 112447	679/ 112260	671/ 112201

4 实验结果及其分析

用表 3 中的参数对前述的 12 个路由表进行测试的结果如表 4 所示. 从表中可以看出, 算法使用表 3 中的参数能有效地处理真实的路由表. 而算法的平均更新时间可由下式得到:

平均更新时间= CAM 访存时间+ TCAM 更新时间 @(1- CAM 更新命中率) (1)

这里假设 TCAM 更新时采用时间复杂度最大的顺序移动算法, 则每次对 TCAM 的更新平均需要移动 TCAM 中前缀数目的一半, 而(1- CAM 更新命中率) 可以使用 TCAM 中的前缀数目与总的前缀数目之比来近似, 则式(1) 变为:

平均更新时间U CAM 访存时间+ (TCAM 访存时间 @TCAM 中前缀数目^2)/ (2@前缀总数) (2)

根据上式, 在 12 个路由表中, 平均更新时间最大的是 5 号路由表:

平均更新时间U CAM 访存时间+ 2.86TCAM 访存时间 (3)

由此可以看出算法的平均更新时间与搜索时间相差不大. 需要指出的是, 如果使用更为高效的 TCAM 更新算法, 将会使算法的平均更新更新时间极大减小.

现在 10ns、4ns 甚至更高速的 CAM 和 TCAM 已很常见, 以 10ns 的器件为例, 算法每秒可完成 100M 次查找, 已能完全满足 OC768 接口的线速查表要求(OC768 接口每秒最多可收到 104.2M 个 48 字节的最短报文). 而且由于算法更新速度很快, 对于路由抖动等有非常好的适应性.

5 算法并行性和可扩展性

算法建议采用 ASIC(专用集成电路) 实现. 原因在于业界所使用的 CAM 结构比算法所需的 CAM 结构复杂, 直接使用现有器件不能获得最好的时空性能和性价比; 其次, 算法硬件实际上是纯粹的高速存储器件, 结构简单, 无需处理器参与, 利于集成, 并且可以将更多的晶体管用于各个 CAM 和 TCAM.

该算法对于路由表的增长有很强的适应性. 如图 3 左图所示, 可以把多个算法 ASIC 并联在一起, 每个 ASIC 中保存整个转发表的一部分, 从而减少对每个 ASIC 的容量要求. 在进行路由更新和搜索时, 可以根据前缀地址的前若干位进行片选, 决定要操作的 ASIC.

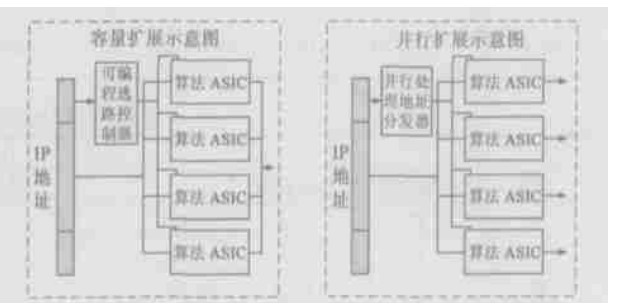

图 3 算法可扩展性和并行性

此外, 算法还有很好的并行性, 当报文的到达速度高于查表速度时, 可采用图 3 右图中的并行结构. 与扩展性结构不同, 这里每个 ASIC 中都保存完整的转发表. 查表时由分发器将目标 IP 地址流水地分配到各个 ASIC 上进行查表, 查到的结果流水送出. 由于查表时间固定, 因此报文具有保序性. 这两种模式可以组合出大容量、高并行性的 IP 路由查找引擎. 可以根据路由器的实际情况, 如同在个人电脑上插拔 RAM 内存一样进行灵活的配置. 通过这种容量扩展和并行扩展, 可以解决未来更高速率和更大容量设备的线速转发要求.

6 结论

当前绝大部分的网络流量仍然是 IPv4 报文, 因此对于高性能 IPv4 路由查找算法的研究仍然很重要. 我们的研究组一直都关注着 IP 路由查找和 IP 报文分类问题的研究[16-18]. 通过研究和探索我们认为, 在 IP 路由查找这个对速度要求极高的问题上, 简单但直接的方法可能比复杂的优化更为有效. 在

此思想的指导下, 我们运用 Andah1 定律提出了基于ASIC实现的高速可扩展并行 IP 路由查找算法. 该算法利用了路由表的统计和分布特征, 使用更小但更快的存储器, 使算法在搜索和更新的时空性能以及性价比与现有的硬件算法相比有很大提高, 而且具有很好的可扩展性和并行性, 可满足现在和未来更高速率和更大容量设备的线速转发要求.

参考文献:

[1] L G Roberts. Beyond Moore . s law: Internet growth trends[J]. IEEE Computer Internet Watch, 2000, 33(1): 117- 119.

[2] D E Knuth. 计算机程序设计艺术第 3 卷: 排序和查找(第二版)[M]. 苏运霖, 译. 北京: 国防工业出版社, 2003. 458- 478.

[3] D R Morrison. PATRICIA2pratical algorithm to retrieve information coded in alphanumeric[J]. Journal of ACM, 1968, 15(4): 514- 534.

[4] K Sklower. A tree2based packet routing table for berkeley unix[A]. Proceedings of 1991 Winter USENIX Conference[C]. Dallas TX USA: USENIX, 1991. 93- 99.

[5] V Srinivasan, G Varghese. Fast IP lookups using controlled prefix ex2 pansion[J]. ACM Transactions on Computer Systems, 1999, 17(1): 1- 40.

[6] Nilsson, G Karlsson. IP address lookup using LC2tries[J]. IEEE Journal on Selected Areas in Communications, 1999, 17(6): 1083- 1092.

[7] 徐恪, 吴建平. 基于 TCAM 的高速路由查找[J]. 小型微型计算机系统增刊(CERNET 2001 学术年会), 2001.

[8] Pankaj Gupta, Steven Lin, Nick Mckeown. Routing lookups in hardware at memory access speeds[A]. Roch Guerin. Proceedings of the IEEE Infocom 98(San Francisco USA)[C]. New York USA: IEEE Xplore Press, 1998. 1240- 1247.

[9] IPMA 组织. Mae2west mae2east 和 aads 等路由器的路由表前缀长度统计[DB/OL]. http://www.merit.edu/ipma/routing. table, 2003 - 05.

[10] M DegerMark, et al. Small forwarding tables for fast routing lookups [A]. Christophe Diot. Proceedings of ACM Sigcomm 97(Cannes France)[C]. New York USA: ACM Press, 1997. 3- 14.

[11] T C Chiueh, P Pradhan. High performance IP routing table lookup using CPU caching[A]. Bharat Doshi. Proc of IEEE Infocom99(New York USA)[C]. Piscataway: IEEE Xplore Press, 1999. 1421- 1428.

[12] Mikael Degermark, Andrej Brodnik, Svante Carlsson, Stephen Pink. Small forwarding tables for fast routing lookups[A]. Christophe Diot. Proceedings of ACM Sigcomm 97(Cannes France)[C]. New York, USA: ACM Press, 1997. 3- 14.

[13] M A Ruiz2Sanchez, E W Biersack, W Dabbous. Survey and taxonomy of IP address lookup algorithms[J]. IEEE Network, 2001, 15: 8- 23.

[14] IPMA 组织. Mae2west mae2east 和 aads 等路由器的路由表映象[DB/OL]. ftp://ftp.merit.edu/statistics/ipma/data, 2003- 05.

[15] 中国 Cernet 中心. Cernet 中心路由器的 Global 路由表映象[DB/OL]. http://bgpview. 6test. edu. cn/bgp - view/download. shtml, 2003- 05.

[16] 彭元喜, 唐玉华, 龚正虎. 基于压缩 NH 表的高速IP路由查找算法的研究[J]. 电子学报, 2002, 30(2): 196- 200.
PENG Yuan2xi, TANG Yu2hua, GONG Zheng2hu. The study on high2 speed algorithms of IP routing lookups based on the compressed next Hop table[J]. ACTA ELECTRONICA SINICA, 2002, 30(2): 196- 200.

[17] 彭元喜, 龚正虎. 基于 LSOT 的高速 IP 路由查找算法[J]. 计算机学报, 2002, 25(1): 106- 111.

[18] 彭元喜, 龚正虎, 刘耀. 基于短前缀长度分割的高速二维分类算法[J]. 计算机研究与发展, 2002, 39(9): 1038- 1042.

作者简介:

谭明锋　男, 1976 年出生于云南, 1998 年毕业于国防科技大学计算机系, 获学士学位, 2001 年毕业国防科技大学计算机学院, 获工学硕士学位, 现为国防科技大学计算机学院 2001 级博士研究生. 主要研究方向为下一代网络体系结构及超高速网络交换与路由及报文分类研究. E2mail: mftan@263. net.

龚正虎　男, 1945 年出生于湖南, 国防科技大学计算机学院教授, 博士生导师, 主要研究方向为网络管理、网络信息安全技术、下一代网络体系结构、超高速网络交换与路由研究等.