基于有限前缀扩展和多 Hash 函数的动态 IP 路由查找算法

谭明锋, 龚正虎, 高 蕾
(国防科学技术大学计算机学院, 湖南长沙 410073)

摘 要: 该算法根据 IP 路由表的分布特征将前缀有限扩展为三种长度, 并用算法所提出的最大熵判定法选取多个 Hash 函数, 将扩展后的前缀映射到三个 Hash 表的不同级别. 在查找过程中算法根据三个 Hash 表的命中率动态计算查找代价, 并据此调整对三个 Hash 表的搜索顺序. 算法支持增量更新, 适于软件实现和硬件流水实现. 实验表明, 对 128K 前缀的真实转发表算法仅约需 3.7M 字节, 平均每次查找仅需约 1.1 次访存, 而且路由更新时间较小.

关键词: 动态 IP 路由查找; 有限前缀扩展; 哈希; 最大熵判定法
中图分类号: TP393 文献标识码: A 文章编号: 0372-2112 (2005) 11-1992-08

A Dynamic IP Routing Lookup Algorithm Based on Limited Prefix Expansion and Multiple Hash Function Techniques

TAN Ming-feng, GONG Zheng-hu, GAO Lei
(*School of Computer Science, National University of Defense Technology, Changsha, Hunan 410073, China*)

Abstract: This algorithm expands all prefixes to three kinds of lengths according to the prefixes distribution of route tables. And several Hash functions are selected by using the maximum entropy criterion method proposed in this paper, then map the expanded prefixes to different levels of three Hash tables. In process of IP routing lookup, the algorithm dynamically calculates the searching cost using the hit rate of the three Hash tables, and then adjusts its searching sequence accordingly. The algorithm supports increasing update, and is easy to be implemented in software or pipelined hardware. The experiment demonstrates that it needs less than 3.7 M bytes for the real route tables of 128K prefixes, and averagely it needs only 1.1 memory accesses for each lookup and few memory accesses for each update.

Key words: dynamic IP routing lookup; limited prefix expansion; hash; maximum entropy criterion method

1 引言

Internet 的高速发展使路由器每秒所需转发的报文数量剧增, 而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到 Internet 的整体性能. 我们在[1]中提出了一种高性能 IP 路由查找硬件算法, 该算法利了多个存储体的并行性以提高查找速度, 所以不利于用软件实现.

因此本文提出了 LPE_MH (Limited Prefix Expansion and Multiple Hash) 算法, 利用路由表分布特征, 提出并运用最大熵判定法选择算法所用的 Hash 函数, 运用有别于传统树形结构的新思路: 将前缀有限扩展成为三种长度的前缀, 并保存到三个 Hash 表中, 搜索时动态计算查找代价并调整搜索顺序, 从而获得更高性能, 并能自动适应不同的报文流. 且该算法支持动态更新, 软硬件均易实现.

2 相关工作

IP 路由查找的经典算法是基于二叉 Trie 树的 Radix Trie[2]、Patricia 算法[3]以及 Sklower[4]对 Patricia 算法的改进算法等, 它们的树高决定了访存次数, 从而决定了时间性能. 因此, 路径压缩[3]、leaf pushing[5]等技术被用于降低树高, 但它们在路由表较大时基本失效, 所需平均访存数达 20 次以上[6]. 而 LC Trie 树算法[7]、受控前缀扩展算法[5]等则使用前缀扩展[5]以及多分支等优化技术来降低树高, 但这些算法更新时间复杂度高, 耗费空间大, 且树的特点决定了这些算法仍需多次访存.

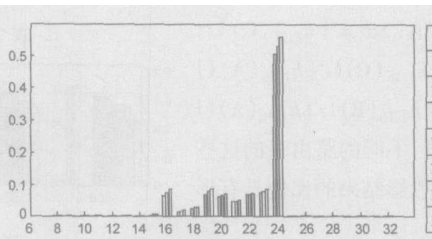
一些利用 Cache 的算法努力让更多的访存落到 Cache 中以提高速度, 如 Lulea 算法[8]以及 Chiueh 所提出的算法[9]等. 但由于使用了压缩技术和 leaf pushing 技术, 更新时需重建数据结构, 且在搜索时需要解压缩, 所以虽然它们的访存次数少, 但是每次路由查找平均约需执行 200 条指令.

24-8 DIR[10]算法实际是硬件实现的两级多分支前缀扩展算法. 由于长度大于 24 的前缀极少, 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 位作为地址访问存储器, 一般每次查找只需要一次访存. 但该算法需要大量的存储空间, 若使用 SRAM 作为存储器来提高速度, 成本很高, 且每次更新需要大量访存. 而利用

收稿日期: 2005-03-08; 修回日期: 2005-08-09
基金项目: 国家重点基础研究发展计划 (973 项目) (No. 2003CB314802); 国家自然科学基金重点项目 (No. 90104001)

TCAM (Ternary CAM)的硬件算法[11]将较长的前缀保存在低地址表项中, 搜索时并行匹配所有表项, 并选取地址最低的匹配项以实现最长前缀匹配. TCAM 查找速度快, 但有功耗大、更新复杂、单位 bit 昂贵、容量小等不足[11].

可以看到, 目前很多算法着重考虑了搜索性能, 而对更新性能、空间性能、可扩展性等考虑较少, 所使用的优化技术越来越复杂, 导致更新困难甚至要求重构整个路由表, 而且实现复杂性较高.



8	0.0323	16	8.54	24	55.9	32	0.0498
9	0.0108	17	1.99	25	0.269		
10	0.00717	18	3.06	26	0.263		
11	0.0143	19	8.45	27	0.308		
12	0.0502	20	7.11	28	0.237		
13	0.145	21	4.97	29	0.162		
14	0.259	22	7.05	30	0.110		
15	0.441	23	8.41	31	0.00892		

注: 左表数值为实验中得到的不同长度前缀所占比例的最大值 x%

图 1 不同长度前缀在不同路由表中所占比例的最小、平均和最大值, 横轴为前缀长度, 纵轴为比例

为解决上述问题, 我们提出了基于 ASIC 实现的硬件算法[1]和其它一些算法[12-14]. 该算法运用了 Amdahl 定律, 使所占比例最大的前缀被查找和更新的速度最快. 它使用了多个 Hash 函数, 并利用了组相联 CAM 的特性, 使其具有很高的时空性能. 对真实的 128K 转发表, 它仅需总容量 1.55MB 的 17 个 CAM 和 1 个容量 1K 项的 TCAM, 能在一次访存时间内获得查表结果, 并可在数次访存时间内完成更新, 可扩展性好. 但该算法在查找时需要并行搜索 18 个存储器件, 因此不利于软件实现.

3 路由表前缀特性相似性简析

通过研究我们发现不同路由表的很多分布特征具有相似性, 这里分析 3 种与本文相关的相似性. 我们以随机选取的 12 个真实 BGP 路由表为例进行分析. 其中 1-4 号取自 Maewest 路由器, 5-8 号取自 Aads 路由器[15], 前缀数目都约为 30K; 9-12 号取自 Cernet 的 Global BGP 路由表[16], 前缀数目约为 112K, 接近 128K. 需要说明的是, 用其它路由表也可以得到相同或类似的结论.

3.1 前缀长度比例分布相似性

图 1 列出不同长度的前缀在路由表中所占比例. 虽然不同路由表的前缀数差别很大, 但这些比例总是大致相同. 一些算法, 如 24-8 DIR 算法就利用了长度大于 24 的前缀非常少的这一特点, 而我们提出的基于 ASIC 实现的算法[1]则进一步利用了长度大于 24 和小于 8 的前缀非常少的这一特性.

3.2 选择 Hash 函数的最大熵判别法以及前缀 Hash 变换结果相似性

本节研究如何选择 LPE_MH 算法中所使用的多个 hash 函数. 在此定义前缀长度为 *PLen*, 并称 IP 地址的最高位为第 1 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.

优秀的 Hash 函数能均匀地映射, 这意味着其结果的熵也越大. 因此我们提出最大熵判别法来选定 Hash 函数. 例如两个 Hash 函数都提取两个位作为其结果, 令〈r, s〉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(〈0, 4〉〈1, 3〉〈2, 5〉〈3, 4〉)和(〈0, 2〉〈1, 1〉〈2, 9〉〈3, 4〉), 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: $\frac{4}{16}\log_2\frac{16}{4}+\frac{3}{16}\log_2\frac{16}{3}+\frac{5}{16}\log_2\frac{16}{5}+\frac{4}{16}\log_2\frac{16}{4}\approx 1.98$, 第二个的熵是 1.87, 该结果和直观判断吻合. 表 1 显示了三个 Hash 函数的熵和 Hash 的结果, 可以看到熵越大, Hash 的的效果越好.

从试验结果可知, 应选取靠近前缀长度的那些位作为 Hash 映射结果, 而且我们的实验表明, 不同长度的前缀经过扩展成为同一长度的前缀时该结论仍然成立. 此外可看到, 不同路由表中某种长度的前缀经过某个 $h_{x,y}$ 映射后的熵非常接近, 所以它们经过上述 Hash 变换后的均匀程度具有相似性.

表 1　多个 Hash 函数的映射结果及其熵, Hash 对象(路由表中长为 24 的前缀)

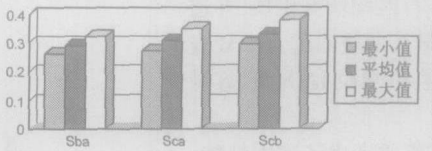
$h_{x,y}$	Hash 映射结果, *PLen*= 24　(x, y)指共有 y 个 Hash 桶高度为 x					
	路由表 1, 共 28527 个前缀, 其中有 15160 个 24 位前缀		路由表 5, 共 30836 个前缀, 其中有 15874 个 24 位前缀		路由表 9, 共 112077 个前缀, 其中有 62666 个 24 位前缀	
	〈桶高, 桶数目〉	熵	〈桶高, 桶数目〉	熵	〈桶高, 桶数目〉	熵
$h_{1,16}$	54 种桶高, 最大桶高 103	4.818	52 种桶高, 最大桶高 86	4.798	121 种桶高, 最大桶高 186	5.005
$h_{5,20}$	(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)	9.026	(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)	9.059	(1, 6683) (2, 3493) (3, 2080) (4, 1605) (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)	9.299
$h^{9,24}$	(1, 12112) (2, 1361) (3, 106) (4, 2)	11.31	(1, 12507) (2, 1482) (3, 133) (4, 1)	11.343	(1, 24618) (2, 11849) (3, 3601) (4, 725) (5, 117) (6, 8) (7, 2)	11.75

3.3 前缀展开后投影结果比例相似性

后文将利用本节的结论和实验结果, 并将解释这些投影变换的缘由. 首先将 *PLen* ≤16 的前缀展开为 *PLen*= 16 的前缀, 得到前缀集 A; 将 17≤*PLen* ≤21 的前缀展开为 *PLen*= 21 的前缀, 得到前缀集 *B*; 将 22≤*PLen*≤32 的前缀展开为 *PLen* = 24 的前缀, 得到前缀集 *C*(*PLen*> 24 的前缀按照 24 位看待). 然后进行两个投影: 将 A、B、C 三个集合用 $h_{1,16}$ 投影, 并得到像集大小| $h_{1,16}$(A) ∪ $h_{1,16}$(B) ∪ $h_{1,16}$(C)|, 以及 B 和 C 各自像集的大小| $h_{1,16}$(B)|和| $h_{1,16}$(C)|; 将 B、C 用 $h_{5,21}$ 投影, 得到像集大小| $h_{5,21}$(B) ∪ $h_{5,21}$(C)|, 以及 C 的像集大小

$|h_{5,21}(C)|$. 这里计算 3 个比值: $Scb = |h_{5,21}(C)|/|h_{5,21}(B) \cup h_{5,21}(C)|$、$Sca = |h_{1,16}(C)|/|h_{1,16}(A) \cup h_{1,16}(B) \cup h_{1,16}(C)|$ 和 $Sba = |h_{1,16}(B)|/|h_{1,16}(A) \cup h_{1,16}(B) \cup h_{1,16}(C)|$. 可以看到, 不同的路由表的这些比例基本上是类似的, 即前缀投影结果的比例具有形似性.

表 2 前缀投影结果比例相似性（数据来源同图 1）



比例因子	Sba	Sca	Scb
最小值	0.2641	0.2709	0.2935
平均值	0.2891	0.3029	0.3269
最大值	0.3207	0.3508	0.3814

4 LPE_MH 基本软件算法 LPE_MH_BS

4.1 LPE_MH_BS 算法思想

LPE_MH 算法吸取了 24_8DIR 算法和 CAT 算法的优点, 首先通过有限前缀扩展生成三种长度的前缀以利于软件搜索, 然后运用 CAT 算法中多 Hash 函数的思想, 降低存储空间耗费, 从而在软件实现时获得很好的时空性能, 而且也易于硬件实现.

LPE_MH_BS 算法将前缀按照其长度分为三组: 第一组 $PLen \leqslant 16$, 第二组 $17 \leqslant PLen \leqslant 21$, 第三组 $22 \leqslant PLen \leqslant 32$. 首先将这三组前缀分别展开为 16、21 和 24 位的前缀($PLen > 24$ 的前缀按照 24 位前缀处理), 分别保存到 A、B、C 三个 Hash 表中, 查找时按照 C、B、A 的顺序搜索以进行最长前缀匹配. 选择这三种长度的原因在于: 根据图 1 中所示的比例, 这三个长度可以使展开后生成的前缀数目较少, 每个表中前缀的数目不会非常多, 有利于查找和减少空间耗费; 而且 A、B、C 三个组中的前缀最多只会分别生成 256、16 和 4 个扩展后的前缀, 可以避免太大的更新时间耗费.

根据 2.2 节最大熵判别法, A 的 Hash 函数是 $h_{1,16}$. 而表 B 的第一级 B1 选择 $h_{5,21}$ 作为其 Hash 函数, 原因是: 对于 128K 大小的路由表, 根据图 1 中给出的比例, 第二组前缀展开后的数目大约为 100K, 而按最大熵判别法用 $h_{5,21}$ 作为其 Hash 函数时 B1 共有 128K 项, 理想情况下正好可将 100K 扩展后的前缀映射到 B1 的不同单元中, 而实际情况下能以较小的内存耗费减少 B1 的 Hash 冲突, 并增加对 B1 的命中率, 减少平均访存次数, 提高性能. 此时 B 的二级表 B2 的 Hash 函数为 $h_{1,4}$.

类似地, 表 C 的第一级 C1 选择 $h_{8,24}$ 作为其 Hash 函数. 而 C_2 的 Hash 函数为 $h_{4,7}$, 除了最大熵判别法外, 这里还考虑到 C2 和 C3 缓冲池的共享问题: C2 选择了 $h_{4,7}$, 则最多有 8 个扩展前 $PLen \leqslant 24$ 的前缀会映射到同一个 C2 单元上, 而 $PLen > 24$ 的前缀非常少, 假设 8 种 $PLen > 24$ 的前缀各有一个映射到同一个 C2 单元的话, 总共最多会有 16 个前缀会映射到同一个 C2 单元上, 每个 C3 结点中最多需要保存 16 个前缀, 而算法中 C 的第三级节点中将按照前缀长度顺序放置前缀, 所以每个三级节点需要 16 项单元. 而第二级使用了 $h_{4,7}$ 后, 每个 C2 节点中有 16 项单元. 而 C2 和 C3 单元的内容类似, 因此 C2 和 C3 可共用同一个缓冲池, 从而减少内存耗费.

当多个前缀映射到 B1 或 C1 的同一单元时, 它不保存路由信息, 而是标记并保存二级结点索引, 并用该索引和二级 Hash 函数来定位下一级单元. 对于 C2 也类似, 不同的是这些前缀将按长短顺序保存在 C3 的节点中. 这些二级和三级节点从相应缓冲池获取, 在删除路由后若为空则回收到缓冲池.

实际上, 虽然 B 有 2 级, C 有 3 级, 但是通过多个 Hash 函数的映射, 大部分路由都在第一级. 而且搜索时第一级的命中率远高于二、三级, 实验表明无论更新还是查找都很少进入 C3. 因此 C3 中的顺序查找对整体性能并没有影响, 因而能以较低的空间耗费获得较好的平均搜索和更新性能.

4.2 LPE_MH_BS 算法数据结构

根据前面的描述可得到图 2 所示的数据结构. 为有效利用空间, 我们对存储内容进行简单编码. 首先, 路由器中下一跳数目通常较少, 用 12 位可编码 2048 个下一跳, 已足够使用, 而真正的下一跳保存在 NHT (Next Hop Table) 中. 此外如

表 3 Hash 表不同级中的单元内容及其含义

Hash 表	长度编码	下一跳索引	比较位	附加比较位	意义
A	0	0	—	—	该单元空闲
	0	非零的 12 位下一跳索引	—	—	有 1 个长度为 1 的前缀映射到该单元
	1~15	12 位下一跳索引	—	—	有 1 个长度为 2~16 的前缀映射到该单元
*B*1	0	0	0	—	该单元空闲
	0	15 位二级索引	1	—	有多个前缀映射到该单元
	1~5	12 位下一跳索引	前缀 ⊢4 位	—	有 1 个长度为 17~21 的前缀映射到该单元
*B*2	0	0	—	—	该单元空闲
	1~5	12 位下一跳索引	—	—	有 1 个长度为 17~21 的前缀映射到该单元
*C*1	0	0	0	—	该单元空闲
	0	15 位 2 级索引	1	—	有 1 个长于 24 或多个前缀映射到该单元
	1~3	12 位下一跳索引	前缀 ⊢7 位	—	有 1 个长度为 22~24 的前缀映射到该单元
*C*2	0	0	0	0	该单元空闲
	0	15 位 3 级索引	1	0	有多个长度为 22~32 的前缀映射到该单元
	1~11	12 位下一跳索引	前缀 ⊢3 位	附加比较位	有 1 个长度为 22~32 的前缀映射到该单元
*C*3	0	0	0	0	该单元空闲
	1~11	12 位下一跳索引	前缀 ⊢3 位	附加比较位	有 1 个长度为 22~32 的前缀映射到该单元

图3 所示, 算法对不同 Hash 表单元中的内容进行了简单编码. 表 3 中是对这些数据结构内容的解释. 通过这种简单的编码可极大降低内存耗费.

4.3 LPE_MH_BS 算法描述

这里给出 LPE_MH_BS 的路由添加算法和查找算法的文字描述. 实际上添加时首先要搜索和定位要添加的位置, 然后进行路由添加; 而删除则与添加类似, 首先搜索和定位要删除的位置, 然后进行删除, 所不同的只是在 B2、C2 和 C3 的结点上删除路由后, 若该节点为空则进行回收.

另外, 在此说明访问 Hash 表各级单元方法: A、B1、C1 都直接用一级 Hash 函数和前缀获取单元索引. B2 和 C2 则从一级单元中获得第二级结点索引, 并用二级 Hash 函数和前缀获得结点内的单元索引. 而 C3 中路由按照长短顺序存储, 因此需要顺序比较查找获取单元索引: 首先从 C2 单元中获得 C3 结点索引, 并通过比较找到对应的单元及其索引. 获得单元索引后, 即可访问该单元.

LPE_MH_BS 算法路由添加算法描述

输入: 路由 r

1. 将 r 的前缀展开为相应长度的前缀集合 P, 并根据前缀长度得到要添加的 Hash 表 X
2. 循环, 对每个 pre$\in P$　/* 下列流程表 A 只到第一级, 表 B 只到第二级* /
   - 2.1 用 Hash 函数和 pre 得到 X 的第一级对应单元 entry1
   - 2.2 若 entry1 为空
     - 2.2.1 若前缀长度 $plen \leqslant 24$ 则直接存储路由信息
     - 2.2.2 若 $plen > 24$, 则分配一个 X 的二级结点 node2, 在 entry1 中记录 node2 的索引并标记 entry1 有二级子结点, 然后将该路由信息 node2 中对应的单元里
   - 2.3 若 entry1 中保存了一个路由 $r1$, 且 r 与 $r1$ 其中一个的前缀是另一个的子前缀, 则保留子前缀的路由. 若无此关系, 则分配一个 X 的二级结点 node2, 在 entry1 中记录 node2 的索引并标记 entry1 有二级子结点, 然后将 r 和 $r1$ 的路由信息分别保存在 node2 中对应的两个单元里
   - 2.4 若 entry1 标记为有二级结点, 则获取二级结点内对应 pre 的单元 entry2
     - 2.4.1 若 entry2 为空则直接存储路由信息
     - 2.4.2 若 entry2 保存了一个路由 r2, 且 r 与 r2 其中一个的前缀是另一个的子前缀, 则保留子前缀的路由. 若无此关系, 则分配一个 X 的三级结点 node3, 在 entry2 中记录 node3 的索引并标记 entry2 有二级子结点, 然后将 r 和 $r2$ 路由信息按照前缀由长到短的顺序保存在 node3 中的前两个单元里
     - 2.4.3 若 entry2 标记为有三级结点, 则将新的路由存储到该三级结点 node3 中, 并判断是否存在前缀与子前缀的关系并保留子前缀的路由, 最后将 node3 中所有保存的路由按照前缀长度从长到短排序

LPE_MH_BS 算法路由查找算法描述

输入: IP 地址 h 输出: 下一跳索引 nhIndex

1. 在 C 中查找

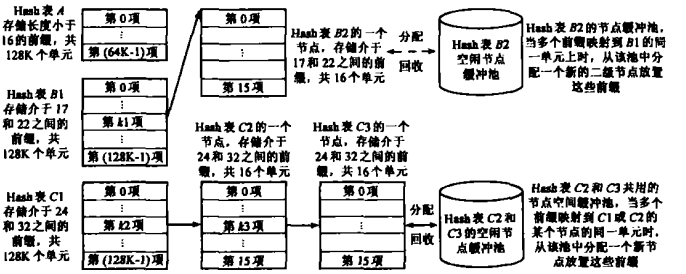


图 2 LPE_MH 算法数据结构及其关系图

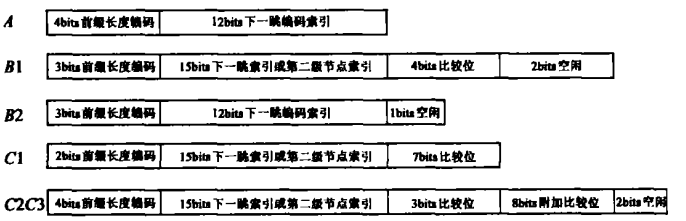


图 3 各个 Hash 表单元数据结构图

   - 1.1 用 $h_{8,24}(h)$ 获得对应的 C1 单元 entryC1
   - 1.2 若 entryC1 为空, 则转到 2
   - 1.3 若 entryC1 中只保存了一个路由项 r
     - 1.3.1 若 r 与 h 匹配, 则返回 entryC1 的 nhIndex
     - 1.3.2 否则转到 2
   - 1.4 若 entryC1 被标记为有多个路由
     - 1.4.1 用 entryC1 中保存的第 2 级索引和 $h_{4,7}(h)$ 获得第 2 级单元 entryC2
     - 1.4.2 若 entryC2 为空, 则转到 2
     - 1.4.3 若 entryC2 中只保存了一个路由项 r
       - 1.4.3.1 若 r 与 h 匹配, 则返回 entryC2 中所保存的 nhIndex
       - 1.4.3.2 否则转到 2
     - 1.4.4 若 entryC2 被标记为有多个路由, 则用 entryC2 中保存的第 3 级索引获得第 3 级节点 nodeC3, 然后用 h 顺序对 nodeC3 中每个单元所保存的路由 r 进行匹配, 若匹配则返回它的 nhIndex, 若一直到搜索完所有单元也未匹配, 则转到 2.
2. 在 B 中查找
   - 2.1 用 $h_{5,21}(h)$ 获得对应的 B1 单元 entryB1.
   - 2.2 若 entryB1 为空, 则转到 3.
   - 2.3 若 entryB1 中只保存了一个路由项 r
     - 2.3.1 若 r 与 h 匹配, 则返回 entryB1 中所保存的 nhIndex.
     - 2.3.2 否则转到 3.
   - 2.4 若 entryB1 被标记为有多个路由
     - 2.4.1 用 entryB1 中保存的第 2 级索引和 $h_{1,4}(h)$ 获得第 2 级单元 entryB2.
     - 2.4.2 若 entryB2 为空, 则转到 3.
3. 在 A 中查找
   - 3.1 用 $h_{1,16}(h)$ 获得对应的 A1 单元 entryA1.
   - 3.2 若 entryA1 为空, 则没有找到任何匹配, 返回 0.
   - 3.3 若 entryA1 中保存的路由 r 与 h 匹配, 则返回 entryA1 中所保存的 nhIndex.

4.4 LPE_MH_BS 算法模拟和实验结果分析

实验所使用是真实的路由表[15,16], 而报文流亦来自于

表 4 LPE_MH_BS 算法更新性能测试数据(其中路由表 1、5、9 同表 1;时间单位为访存数)

路由表	1	5	9	路由表	1	5	9
平均更新时间(读/写/读写)	2.01/2.55/4.56	2.02/2.51/4.53	1.97/2.79/4.76	B2 占用节点	2418	2421	17839
最大更新时间(读/写/读写)	32/256/256	32/256/256	32/256/256	C2 占用节点	2346	2850	26578
展开后前缀比 B1:B2	4.80:1	5.12:1	1.25:1	C3 占用节点	90	185	1655
展开后前缀比 C1:C2:C3	119.48:25.59:1	43.89:10.36:1	13.87:16.42:1	所需总空间	1.28MB	1.32MB	3.43MB

表 5 LPE_MH_BS 算法搜索性能测试数据(其中路由表 1、5、9 同表 1;时间单位为访存数)

其中 IP Trace1:75,974,921 报文头;IP Trace2:19,175,914 报文头;IP Trace3:18,523,793 报文头

搜索性能 P		IP Trace1			IP Trace2			IP Trace3		
路由表		1	5	9	1	5	9	1	5	9
命中率 x%	A+B+C	9.11	8.88	22.9	10.3	6.06	24.52	8.49	7.94	20.26
	A	8.22	8.03	19.82	9.75	5.5	19.23	7.69	7.22	16.52
	B	0.83	0.78	2.51	0.48	0.5	5.05	0.73	0.65	3.46
	C	0.058	0.068	0.589	0.064	0.0067	0.23	0.072	0.081	0.27
平均/最大搜索时间		3.09/9	3.40/13	3.44/17	3.06/8	3.06/13	3.33/15	3.07/8	3.08/13	3.35/17

Internet[17]. 由于涉及隐私,因此报文流做了某种不可逆变换,但保留了原报文流中的很多分布特征,因此实验数据具有一定可参考性. 所获得的性能数据如表 4 和表 5 所示.

从实验的结果可看到每次查找平均只需 3 次访存,最差时只需 17 次访存,所需空间少于 3.43MB. 不过按照原预期,前缀数目多的表命中率越高,因此 C 的命中率应高于 B,而 B 的要高于 A,此时平均搜索时间将较小. 但实验结果与该预期刚好相反,前缀长度越短,命中的概率越高,而且大多数报文都未能匹配. 我们推测主要有如下原因:(1)测试用路由表与 IP Trace 文件来自 Internet 不同的域,匹配度较低;(2)当路由表中存在缺省路由时,可能出现上述高 Miss 率,此时 Miss 报文都通过缺省路由转发. 此类情况主要在接入路由器、边缘路由器或汇聚路由器上发生,因此具有其现实的应用背景和意义.(3)长度越短的前缀在数轴上覆盖的范围也越大,命中可能性越大.

由此我们得到一个两难问题:按照 C、B、A 的顺序查找,实际上很大一部分是做了无用功. 而若简单地按照 A、B、C 的顺序查找,那么无论有没有匹配都要搜索更长的前缀,也有可能做无用功. 因此我们得到一个启示:如果一个报文不被任何前缀所匹配,那么越早知道这一信息越好,这样可以降低搜索所需时间. 所以我们对 LPE_MH_BS 算法进行改进,得到 LPE_MH_ES 算法.

5 LPE_MH_BS 的改进算法 LPE_MH_ES 及其模拟结果

上述实验结果表明不应预先假设 3 个 Hash 表的各自的命中率,应动态对其检测并计算查找,进而调整搜索顺序. LPE_MH_ES 算法的改进主要有两点:针对总体 Miss 率的优化和根据 Miss 率和命中率动态调整查找顺序的优化.

5.1 针对总体 Miss 率的优化

对于 *Miss* 的报文(这里指对非缺省路由 *Miss*),为其所作的所有查找工作都是徒劳的. 因此,一个关键性的思想是:尽早知道表中不存在对该报文头的匹配,这可以减少无用的查表时间耗费.

如图 4 所示,首先为 *A* 的每个单元增加 2 个字节. 当在 *B* 中更新一个路由 *r* 后,在 *A* 中用 $h_{1,16}$ 对 r 的前缀 *pre* 进行映射,获得 *A* 的第 $h_{1,16}(pre)$ 个单元 *entryA*,然后将 *entryA* 中的 *B* 引用计数器增 1(添加)或是减 1(删除). 而在 *C* 中更新一个路由 *r* 1 后,同样获得 *entryA* 1,然后将 *entryA* 1 中的 *C* 引用计数器增 1 或是减 1. 通过这种方式,在 *A* 中记录了 *B* 和 *C* 中是否存在目标地址的可能匹配路由. 当对 *A* 进行搜索时,如果发现 *A* 中记录了 *B* 或是 *C* 中存在可能的匹配项,那么则需要到 *B* 或是 *C* 中继续查找,否则搜索将在 *A* 停止. 通过这种方式,达到了"尽早知道表中不存在对该报文头的匹配"的目的,从而减少无用的后续搜索,获得性能的提高. 类似地为 *B*1 的每个单元增加 1 个字节的 *C* 引用计数器. 通过这样的方法,算法用较少的空间减少了大量无用的访存,获得性能的提高. 需要指出的是,实际上这就是 3.3 节中的前缀投影,而 3.3 节中所获得的数据将在后一节使用.

5.2 根据 Miss 率和命中率动态调整查找顺序优化

下面的通过分析给出进行动态调整的判定条件以及为达到此目的算法需要额外记录的信息. 首先假设 A 的命中率为 x,B 的为 y,C 的为 z,这里的命中率是指算法在 A、B、C 中获得最长前缀匹配的命中率. 并假设在 A、B、C 中查找的平均代价分别为 a、b、c,且假设 C 中的前缀对 A、B1 的投影比例为 Sca、Scb,B 中的前缀对 A 的投影比例为 Sba. 这里共有 3 个 Hash 表,所以共有 6 种查找顺序. 我们分别计算 6 种顺序下的平均查找性能以分析如何判定查找顺序. 这里用 CostHA、CostHA、CostHC 分别表示在 A、B、C 中命中时的平均代价,而 CostM 为对 A、B、C 都 Miss 时的代价. 而每种情况下都有 CostHA = CostM,因此总平均代价为:

$$\begin{aligned} Cost &= x*CostHA + y*CostHB + z*CostHC + [1-(x+y+z)]*CostM \\ &= CostHA + y*(CostHB-CostHA) + z*(CostHC-CostHA) \end{aligned} \quad (1)$$

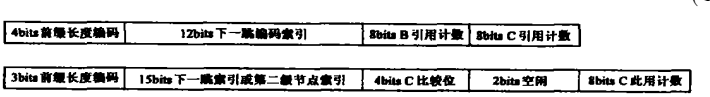


图 4 改进后 *A* 和 *B*1 各单元的内容

根据实验结果, 可得到 a、b、c 的近似值分别为: a= 1, b= 1.0367, c= 1.27, 并且可以取表 2 中的平均数 Sba= 0.2891, Sca = 0.3059, Scb= 0.3269 作为参数. 由此得到(1)中的各个系数, 如表 6 所示.

表 6　平均代价 Cost 的系数求取

顺序	CostHA= CostM	CostHB	CostHC	CostHB- CostHA	CostHC- CostHA
ABC	a+ Sba* b+ (Sba* Scb+ Sca) * c= 1.8082	A+ b+ Scb* c= 2.4519	a+ b+ c= 3.3067	0.6437	1.4985
ACB	a+ Sca* c+ Sba* b= 1.6882	A+ Sca* c+ b= 2.4252	a+ b+ c= 3.3067	0.7370	1.6185
BAC	b+ a+ Scb* Sca* c= 2.1637	b+ Scb* c= 1.4519	b+ a+ c= 3.3067	-0.7118	1.1430
BCA	b+ Scb* c+ a= 2.4519	b+ Scb* c= 1.4519	b+ c= 2.3067	-1	-0.1452
CAB	c+ a+ Sba* b= 2.5697	c+ a+ Sba* b= 2.5697	c= 1.27	0	-1.2997
CBA	c+ b+ a= 3.3067	c+ b= 2.3067	c= 1.27	-1	-2.0367

在此仅解释按照 ABC 的顺序查找且命中 A 的平均代价, 即 CostHA 的估算方法: 首先查找 A 并命中, 此时检查 A 中是否记录了 B 中是否有可能的匹配, 可能有的概率为 Sba, 然后决定到是否进入 B 中查找. 而到 C 中查找的概率是进入 B 后再到 C 的概率(Sba* Scb)和直接由 A 进入 C 的概率 Sca 之和(Sba* Scb+ Sca). 因此总的代价即为 a+ Sba* b+ (Sba* Scb + Sca) * c.

因此在查表的过程中, 通过统计 B 的命中率 y 和 C 的命中率 z 就可以利用式(1)求出上述各个顺序下的平均代价, 经过简单的排序后, 即可选取代价最小者作为下一次的查找顺序.

LPE_MH_ES 并不对每个报文都调整顺序, 而是每隔 N, 例如 N= 64K 个报文后, 重新计算一次. 这样做主要有 3 个原因: 1、可降低该计算对性能的影响; 2、上述的平均性能只是近似推导, 并不完全精确, 因此没有必要调整每个报文的搜索顺序, 只需要在较大尺度上进行调整即可; 3、每隔 N 个, 如 64K 个报文计算一次可以使算法对报文流的动态特征具有良好的适应性.

5.3　LPE_MH_ES 算法描述

改进后得到的 LPE_MH_ES 算法在 A、B、C 三个表内查找的方法和过程与 LPE_MH_BS 基本相同, 只是多了计算查找顺序的过程以及根据记录的信息跳过下一个 Hash 表或是中止查找的判断. 而更新的差别仅在于 C 更新完后, 要到 B 和 A 中修改标记, 而 B 更新完毕后需要到 A 中修改标记.

LPE_MH_ES 算法路由查找算法描述

输入: **IP** 地址 **addr** 输出: 下一跳索引 **nhIndex** 初始顺序: **ABC**

1　循环接收报文

1.1　当前顺序为 **X、Y、Z**, 所以从第一个表 **X** 开始查找.

1.2　在 **X** 中查找完成后, 根据 **X** 中记录的标记判断 **B** 或 **C** 中是否存在可能的匹配, 然后决定继续向 **Y** 查找, 或是跳过 **Y** 到 **Z** 中查找, 或是结束查找转到 1.3.

1.3　根据查找的命中结果更新 **B** 和 **C** 的命中计数器和总报文计数器.

1.4　若报文计数器值已达到 **N**(如 64**K**)

1.4.1　重新计算顺序和设置新的查找顺序

1.4.2　所有计数器清零.

5.4　LPE_MH_ES 算法实验结果

实验得到的算法性能如表 7 和表 8 所示, 实际上, ES 算法所需的空间仅比 BS 算法多 256KB. 表 9 是 LPE_MH_ES 算法与其它几个算法的定性比较.

表 7　LPE_MH_ES 算法更新性能测试数据
(数据来源同表 4; 时间单位为访存数)

路由表	1	5	9
平均更新时间 (读/写/读写)	4.88/5.44/ 10.32	4.88/5.40/ 10.27	4.71/5.60/ 10.31
最大更新时间 (读/写/读写)	48/256/ 256	48/256/ 256	48/256/ 256
总空间(字节)	1.53MB	1.57MB	3.68MB

表 8　LPE_MH_ES 算法搜索性能与 BS 算法相比测试数据(N= 64K; 数据来源同表 5)

搜索性能(时间单位: 访存数)	IP Trace 1			IP Trace 2			IP Trace3		
路由表	1	5	9	1	5	9	1	5	9
BS 算法平均/最大搜索时间	3.09/9	3.40/13	3.44/17	3.06/8	3.06/13	3.33/15	3.07/8	3.08/13	3.35/17
ES 算法平均/最大搜索时间	1.14/7	1.14/13	1.24/15	1.10/8	1.10/13	1.24/15	1.13/8	1.13/13	1.22/17

表 9　LPE_MH_ES 算法与其他算法的比较(时间单位为访存数)

算法	平均搜索时间	平均更新时间	空间需求(128K 大小的转发表)	软硬件实现灵活性
LPE_MH_ES	平均略大于 1	约 10 次访存	小于 4MB	软硬件都可灵活实现
24-8 DIR	平均略大于 1	数千次访存	略大于 64MB	软硬件都可实现, 主要用于硬件实现
CAT	等于 1	少于 4 次访存	总容量 1.55MB 的 17 个小 CAM 和一个 1K 项大小的 TCAM	硬件实现
Patricia Trie	前缀数目较多时超过 20	前缀数目较多时超过 20	至少 1.152MB	软件实现硬件, 实现过于复杂

实验所使用的模拟环境为: 模拟程序用 C+ + 编写, vc7. 0 编译器, 编译优化开关全开; CPU 为 Intel Pentium M 1500MHZ 处理器, L1 数据和指令 Cache 各 32KB, 1 MB 集成 L2 cache; 内存 512M DDR333. 算法的实测结果如表 10 所示. 这里的搜索时间是指从 IP 目标地址到达到下一跳索引被检出所使用的时间, 不包含从硬盘读取 Trace 文件到内存的时间.

表 **10** LPE_ MH 算法性能实测结果, 数据来源同表 **5**

算法	实测平均搜索性能	实测平均更新性能
LPE_ MH_ BS	52. 74M 报文/ 秒	7. 96M 路由/ 秒
LPE_ MH_ ES	81. 70M 报文/ 秒	7.26 M 路由/ 秒

从表 10 中可看出 LPE_ MH_ ES 的性能获得了较大提高. 该算法处理能力很强, 在报文最小长度为 40 字节的情况下, 完全可以满足 OC192 的线速转发要求(31. 25M 报文/ 秒), 并能部分能够部分满足 OC768 的线速转发要求(125M 报文/ 秒) . 而在平均报文长度约为 354 字节[18] 的条件下, 算法能完全满足 OC768 的线速转发要求(14. 1M 报文/ 秒) . 不过也可发现, 虽然 LPE_ MH_ ES 算法的平均访存数仅是 LPE_ MH_ BS 算法的约 1/3, 但是由于附加的统计、查找顺序计算等操作, 所获得的性能提高并不与访存次数的降低成正比.

6 LPE_ MH 算法的硬件实现 LPE_ MH_ HW

LPE_ MH 算法易用图 5 所示的硬件流水实现. 由于报文在 B 和 C 中主要在第 1 级命中, 因此在绝大多数情况下查找时间为 1 次访存时间, 而最差情况下需要 3 拍完成. 总平均性能接近 1 次访存.

7 结论

硬件算法有速度快的优势, 但并不能完全替代软件算法. 首先硬件算法通常成本偏高, 而某些低端路由器可能完全由软件实现路由查找; 其次, 路由器上非常核心的路由表管理模块通常由软件, 需要高性能软件 IP 路由查表算法的支持; 此外, 目前很多路由器使用网络处理器进行开发, 一个重要动机是认为易实现性和易修改性比速度更为重要[19], 因此它们通常都能够对路由查表引擎进行灵活配置, 使用固化在硬件中的不同软件查表算法. 因此, 研究易于软件和硬件实现的高性能 IP 路由查表算法的研究, 具有其现实意义和重要性. 所以我们提出了满足这一要求的 LPE_ MH 算法.

IP 路由查找是互连网络性能的关键问题之一, 我们未来在该领域的研究将重点着眼于如下几个方面: 研究利用路由表以及报文流的统计信息设计新的算法, 并对已有算法进行优化, 使之能够满足未来高速链路的转发要求; 研究利用新的器件对算法进行优化, 例如 64 位 CPU、高性能存储器如 QDR (四倍速 SDRAM)、具有并行处理能力的多内核网络处理器等; 研究在新型路由器体系结构中并行和流水地进行路由查表和报文分类的理论、算法和关键技术等.

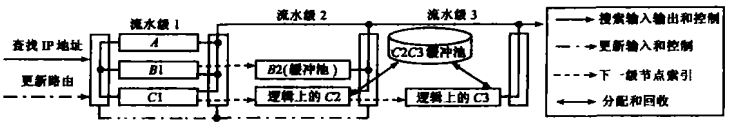


图 5 LPE_MH_HW 算法结构

参考文献:

[1] 谭明锋, 龚正虎. 基于 ASIC 实现的高速可扩展并行 IP 路由查找算法[J] . 电子学报, 2005, 33(2) : 209- 213.

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

[3] D R Morrison. PATRICIA-Pratical Algorithm to Retrieve Information Coded in Alphanumeric[J] . Journal of ACM, New York USA: ACM Press, 1968, 15(4) : 514- 534.

[4] K Sklower. A tree-based packet routing table for Berkeley Unix[A] . Proc of 1991 Winter Usenix Conference[C] . Dallas TX. USA: Usenix Association, 1991. 93- 99.

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

[6] M A Ruiz-Sanchez, E W Biersack, et al. Survey and taxonomy of IP address lookup algorithms[J] . IEEE Network, New York USA: IEEE Press, 2001, 15(2) : 8- 23.

[7] Nilsson G Karlsson. IP address lookup using LC-Tries[J] . IEEE Journal on Selected Areas in Communications, New York USA: IEEE Press, 1999, 17(6) : 1083- 1092.

[8] M Degermark, A Brodnik, et al. Small forwarding tables for fast routing lookups[A] . Proceedings of ACM Sigcomm 97[C] . New York USA: ACM Press, 1997. 3- 14.

[9] T C Chiueh, P Pradhan. High performance IP routing table lookup using CPU caching[A] . Proc of IEEE Infocom 99[C] . New York USA: IEEE Press, 1999. 1421- 1428.

[10] P Gupta, S Lin, et al. Routing Lookups in Hardware at Memory Access Speeds[A] . Proc. of IEEE Infocom 98[C] . New York USA: IEEE Press, 1998. 1240- 1247.

[11] Kai Zheng, Chengchen Hu, et al. An Ultra High Throughput and Power Efficient TCAM-Based IP Lookup Engine[A] . Proc. of IEEE Infocom 2004[C] . New York USA: IEEE Press, 2004, 23(1) : 1985- 1995.

[12] Tan Mingfeng, Gong Zhenghu. High Speed IP Lookup Algorithm with Scalability and Parallelism Based on CAM Array and TCAM[A] . Proc. of IEEE ICC 2004[C] . New York USA: IEEE Press, 2004, 27(1) : 1085- 1089.

[13] 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 Yuanxi, Tang Yuhua, Gong Zhenghu. The study on high-speed algorithms of IP routing lookups based on the compressed next Hop Table[J] . ACTA ELECTRONICA SINICA, 2002, 30(2) : 196- 200. (in Chinese)

[14] 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.

[15] IPMA 组织, Mae-west mae-east 和 aads 等路由器的路由表映象[DB/OL]. ftp://ftp.merit.edu, May 2003.

[16] 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, May 2003.

[17] Passive Measurement and Analysis (PMA) 组织. IP Trace File[DB/OL]. http://pma.nlanr.net, Nov. 2004.

[18] National Laboratory for Applied Network Research (NLANR). Histogram of packet sizes[DB/OL]. http://www.nlanr.net, Jun. 2004.

[19] D E Comer. 网络处理器与网络系统设计[M]. 北京: 机械工业出版社, 2004. 106- 109.

作者简介:

谭明锋　男, 1976 年生于云南. 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.
E-mail: mftan@nudt.edu.cn.

龚正虎　男, 1945 年生于湖南, 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.
E-mail: gzh@nudt.edu.cn.