路由查找算法研究综述*

徐 恪, 徐明伟, 吴建平, 吴 剑

(清华大学 计算机科学与技术系 网络研究所,北京 100084)
E-mail: xuke@tsinghua.edu.cn; xmw@csnet1.cs.tsinghua.edu.cn
http://netlab.cs.tsinghua.edu.cn

摘要: 随着 Internet 的迅猛发展,用于主干网络互联的核心路由器的接口速率已经达到了 2.5Gbps~10Gbps.这一速率要求核心路由器每秒能够转发几百万乃至上千万个以上的分组.分组转发的重要一步就是查找路由表,因此快速的路由查找算法是实现高速分组转发的关键.路由查找需要实现最长前缀匹配.近年来,研究人员提出了多种路由查找算法,以提高查找性能.分析了路由查找问题及其难点,全面综述了各种查找算法,并对它们进行了详细的分析和比较,最后指出了进一步的研究方向.

关 键 词: 路由查找;最长前缀匹配;Trie 树;哈希;CAM
中图法分类号: TP393 文献标识码: A

路由器的主要功能是按照 IP 分组中的目的地址转发分组.查找路由表决定将分组发往哪个端口,这是转发分组过程中的重要一步.因此,快速的 IP 地址查找算法是实现高速分组转发的关键.传统的查找方法(如顺序查找法、二分查找法)只能完成关键字的精确查找,而路由查找需要完成最长匹配地址前缀的查找.为了提高路由查找的性能,研究人员提出了多种路由查找算法.本文首先分析了路由查找问题和实现难点,然后全面综述了各种查找算法,并对它们进行了详细的分析和比较.

1 Internet 地址结构的发展

在 Internet 的发展初期,IPv4 地址采用的是基于类的地址结构.整个 IP 地址空间一共分为 5 类:为单播地址设计的 A,B,C 类;为组播地址设计的 D 类;为今后使用而保留的 E 类地址.各个类之间由地址的前几个比特位来区分.在 Internet 发展的初期,基于类的地址结构由于其简单性而得到了广泛的应用.但是,随着网络的不断发展,使用这种地址结构产生了两个问题.一个问题是地址分配的不灵活导致了地址空间的大量消耗以及路由表规模的不断增大.基于类的地址结构将整个地址空间简单地分成了 5 个类别,这就导致了地址的分配非常不灵活.另一个问题是,由于路由器需要记录所有已经分配的网络地址,特别是对于 C 类地址来说,由于它的地址前缀特别多,如果记录所有的 C 类地址,那么路由表的规模将变得非常大.

为了降低路由表规模的增长速度以及提高地址空间的利用率,IETF 提出了一种称为无类域间路由(classless inter-domain routing,简称 CIDR)的地址结构[1~2].CIDR 摒弃了传统上基于类的地址分配方式,规定可以使用任意长度的网络地址部分,因此产生了路由地址前缀的概念.使用 CIDR 的好处是明显的,它提高了地址空间的利用率,例如为 300 台主机分配网络地址只需使用一个 23 位网络地址前缀长度的网络地址(可以支持

* 收稿日期: 2001-05-30; 修改日期: 2001-09-14
基金项目: 国家自然科学基金资助项目(90104002);国家 863 高科技发展计划资助项目(863-306-ZD-07-01)

作者简介: 徐恪(1974－),男,江苏洪泽人,博士,讲师,主要研究领域为计算机网络体系结构,分布式路由器操作系统,高性能路由器体系结构;徐明伟(1971－),男,辽宁朝阳人,博士,副教授,主要研究领域为计算机网络体系体系结构,高性能路由器体系结构,网络性能测试;吴建平(1953－),男,山西太原人,博士,教授,博士生导师,主要研究领域为计算机网络体系结构,计算机网络协议测试,形式化技术;吴剑(1977－),男,浙江杭州人,硕士,主要研究领域是高速路由查找算法,分布式路由协议.

512 台主机)就可以满足要求.

为了满足网络规模不断增长的需要,IETF 设计了新一代的网络协议 IPv6,也被称为 IPng.IPv6 与 IPv4 相比,在地址格式上发生了巨大的改变,地址长度由原来的 32 位变成了 128 位,相应地,IPv6 在整个地址分配上也进行了一定的改进[3].尽管 IPv6 的地址结构与 IPv4 相比有很多的不同之处,但是从根本上来说,整个地址空间仍然是层次性结构,仍然支持类似于 IPv4 CIDR 地址结构下的路由合并,因此新一代的网络协议并没有改变路由查找的本质特点.

2 路由查找算法分析

2.1 路由查找算法的分类

最长地址前缀匹配查找的难点在于,在查找过程中不仅需要与地址前缀的比特值进行匹配查找,而且还需要考虑地址前缀的长度,因此各种路由查找算法都可以归结为这两个方面的匹配查找过程.

2.1.1 基于地址前缀值的路由查找算法

基于地址前缀值路由查找算法的特点是通过对整个地址前缀空间进行地址关键字穷举来消除地址前缀长度对查找的影响.第 2.2.1 节中介绍的线性匹配查找算法是基于地址前缀值查找中最为简单、直观的地址前缀查找方法.第 2.3.4 节介绍的地址区域二分法也是基于地址前缀值的查找算法,但由于它采用了二分查找,所以在算法性能上有很大的提高.第 2.3.5 节介绍的 CAM 硬件实现方法从本质上来说也属于地址前缀值的查找算法.

2.1.2 基于地址前缀长度的路由查找算法

基于地址前缀长度的路由查找算法的出发点是在前缀长度空间内进行查找,与基于地址前缀值查找算法类似,查找过程可以使用线性遍历法,也可以使用二分法遍历法.第 2.2.3 节介绍的二进制 Trie 树查找算法就是基于地址前缀长度的线性遍历法,因为对于查找的第 i 步来说,匹配的是长度为 i 的地址前缀.同理,第 2.3.2 节介绍的多分支 Trie 查找算法也是基于在地址前缀长度空间内的线性遍历法,但由于它采用大于 1 的步宽,所遍历的地址长度数目变少了,查找性能得到了提高.第 2.3.3 节介绍的算法能够在前缀长度空间内进行二分查找,所以从算法的复杂度来看,它的性能更好.

2.2 传统的路由查找算法

在下面的算法介绍中,我们用表 1 作为算法分析的实例.

Table 1 Examples of Address Prefix

表 **1** 地址前缀实例

	Prefix	NextHop
P1	0*	a
P2	01000*	b
P3	011*	c
P4	1*	d
P5	100*	e
P6	1100*	f
P7	1101*	g
P8	1110*	h
P9	1111*	i

2.2.1 线性查找

线性表结构是最简单的查找数据结构,因此可以把所有的路由地址前缀用线性链表的方式组织起来.每一次查找需要遍历线性表中的所有表项,在遍历过程中记录最长的路由前缀项,直到遍历完整个线性链表为止.对于 N 个路由前缀表项,查找过程的算法复杂度为 $O(N)$,存储空间复杂度为 $O(N)$,插入删除路由表项的算法复杂度为 $O(1)$(假设表项插入和删除均在线性表的末尾进行).

2.2.2 缓存策略

在路由查找中可以使用缓存策略,把最近查找过的目的地址路由存放在高速的路由缓存表(route cache)中,

只有当在路由缓存表中查找失败的时候,我们才需要进行真正完整的前缀查找匹配过程.

为了能够在查找性能上获得较大的提高,缓存的命中率就需要有一定的保证.假设缓存查找速度是路由前缀查找速度的 20 倍,经过计算可以得出,为了能够使平均查找性能达到原来的 10 倍,缓存的命中率需要达到95%左右.随着 Internet 的不断发展、网络用户的不断增加以及业务数据量的多样化,数据的时间局部性变得越来越不明显,从而大大地降低了缓存的命中率.因此,文献[4]建议缓存的大小应该能够随着网络用户或者网络数据业务量的增加而相应地呈线性增加.

2.2.3 二进制 Trie 树(binary Trie)

用二进制 Trie 结构来表示地址前缀是一个常用的方法.Trie 采用一种基于树的数据结构,通过前缀中每一位比特的值来决定树的分支.图 1 就是用二进制 Trie 结构(树中每一个内部结点最多包含两个子结点)来表示的地址前缀表.

在 Trie 树中,处于第 L 层的结点代表了一类地址前 L 个比特均相同的地址空间,并且这前 L 个比特串就是由从根结点到这个结点路径上的 L 个比特组成.例如,图 1 中处于第 3 层的结点 c 就代表了所有前 3 个比特为011 的地址族,而且比特串 011 就是根结点到结点 c 路径上的比特按照遍历顺序所构成的.图 1 中带阴影的结点表示该结点对应着一个地址前缀,因此这些结点中包含了与该地址前缀相关的转发信息.图 1 中的阴影结点既可以是叶子结点(如结点 b),也可以是中间结点(如结点 a).

2.2.4 路径压缩 Trie 树(path-compressed Trie)

图 1 中的 Trie 树经过路径压缩之后得到的 Trie 树如图 2 所示.比较图 1 和图 2 可以看到,结点 b 的前两个父结点已经被删除,结点 a 从原来单分支结点处移动到了二分支结点处,原来的单分支结点被删除.由于删除的单分支结点可能包含多个地址前缀信息,所以路径压缩 Trie 树结点中可能包含多个地址前缀.Trie 树搜索过程由于某些结点被删除,所以可忽略目的地址中某些比特位的匹配操作,因此结点处需要维护一个变量指示下一个需要检查的比特位.另外,与二进制 Trie 树相比,路径压缩 Trie 树前缀结点处需要保存地址前缀的比特串.查找过程与二进制 Trie 树类似,但是在结点选取分支时考虑的是比特位变量所指示的比特位.当查找过程遇到前缀结点时,需要进行前缀匹配操作.当到达叶子结点或者是前缀匹配操作失败时,查找过程终止,查找结果是查找过程中已被记录的最长匹配前缀.

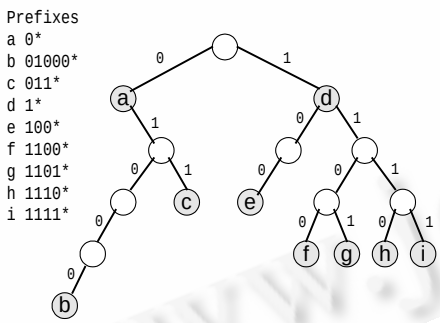

Fig.1 Architecture of binary Trie-tree
图 1 二进制 Trie 树结构

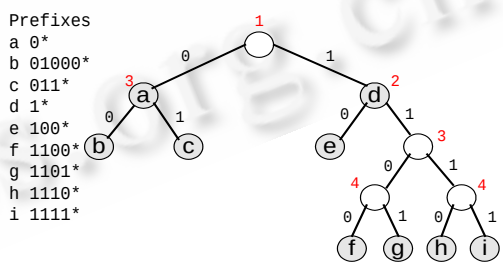

Fig.2 Architecture of path-compressed Trie-tree
图 2 路径压缩 Trie 树结构

路径压缩的思想最初在被称为 PATRICIA[5]的算法中提出,但该算法不支持最长前缀的查找.Sklower[6]对该算法进行改进,使之能够适应最长前缀的查找,并且在 BSD(Berkeley software distribution) Unix 中加以实现.

简单二进制 Trie 树和路径压缩 Trie 树的不足之处在于查找过程需要大量的存储器访问操作,对于 IPv4 地址来说,最差的情况需要 32 次存储器操作.实验表明,使用 BSD 路径压缩 Trie 树来表示典型路由器中 47 113 表项数目的前缀表[7],最大搜索深度为 26,平均搜索深度也达到了 20.使用简单二进制 Trie 树来表示同样的地址前缀表,其最大搜索深度为 30,平均搜索深度为 22.

2.3 路由查找新算法的研究

近几年来,随着对路由器研究的逐步深入以及对路由器性能要求的不断提高,研究人员提出了很多较为新

颖的地址前缀查找算法.与二进制 Trie 树、缓存目的地址法等传统的地址前缀查找方法相比,这些算法在性能方面有了很大的提高.

2.3.1 查找算法使用的辅助策略

为了提高算法的效率,最近提出的算法中都使用了一些辅助策略.

(1) 前缀扩展(prefix expansion).地址前缀是一系列主机地址或者网络地址的合并,因此,地址前缀的转发信息涵盖了所有这些主机或者网络的转发信息,所以我们可以将一条长度较短的地址前缀展开成多条地址长度较长的前缀集,其中这些前缀集的转发信息就是原来地址前缀所对应的转发信息.这种前缀转化方式称为前缀扩展.例如,前缀 1*所覆盖的地址范围既可以用前缀 10*以及前缀 11*来涵盖,也可以用前缀 100*,101*,110*以及 111*来涵盖.

(2) 独立前缀转化(disjoint prefix transformation).最长前缀匹配查找就是为了能够在多个匹配的地址前缀中寻找长度最长的前缀.一种避免进行最长前缀匹配规则但仍然能够找到最精确转发信息的方法是将地址前缀集转化为一些完全独立的前缀集.在独立的前缀集中,各个前缀之间不存在相互包含关系,因此不可能出现某个前缀是另一个前缀的前缀.在根据独立前缀集构造的 Trie 树中,所有对应于前缀的结点都出现在叶子结点.

(3) 压缩技术.压缩技术试图从编码中删除数据的冗余信息,对 Trie 树使用压缩技术主要是考虑到 Trie 树的扩展转化过程大大增加了信息的冗余度,因此可以使用一定的压缩算法将信息的冗余度降低.当然,压缩方法应该不仅能缩小整个 Trie 树的存储空间,而且还应保证从压缩数据中恢复原有信息的操作不能过于复杂.

(4) 优化技术.前缀转发方法是多种多样的,优化方法能够使我们在一些限制条件的前提下找到满足约束条件的最佳前缀集,比如说在查找速度的约束下尽量减少算法的存储空间[8,9]等.

(5) 存储层次设计.目前我们所使用的计算机的整个存储系统具有明显的层次结构,各个层次上的存储介质在速度和容量上存在着差异.如果我们能够把尽可能多的查找表内容放入 Cache 中[10],就可以获得更高的查找速度,因此在算法设计中应尽量减少查找算法数据结构所占据的存储容量.

2.3.2 多分支 Trie 树(multibit Trie)

(1) 基本算法

一种提高 Trie 树查找效率的方法就是在查找的每一步检查地址中的多个比特,而不仅仅是一个.例如,如果我们每次查找检查地址的 4 个比特,那么 IPv4 地址查找最多只需 8 次存储器访问操作就可以了.我们把每一次查找所需检查的比特数称为查找步宽(stride).查找步宽可以是固定的,也可以是可变的.二分支 Trie 树实际上就是查找步宽为 1 的 Trie 树.我们把查找步宽大于 1 的 Trie 树称为多分支 Trie 树(multibit Trie).对于查找步宽为 k 的多分支 Trie 树来说,每一个结点的最大分支数为 2^k.

在多分支 Trie 树中,同一层中不同子树的步宽可以是一样的,也可以是不一样的.一般来说,固定步宽的多分支 Trie 树实现简单,但会浪费较多的存储空间;而可变步宽的多分支 Trie 树在实现上复杂一些,但可以节省一定的存储空间.

多分支 Trie 树查找过程的每一步需要检查多个比特,因此它不能支持任意长度的地址前缀.为了能够用多分支 Trie 树来进行前缀查找,前缀表中的地址前缀需要转换成多分支 Trie 树查找所允许的地址前缀才行,转化方法所采用的就是上一节介绍的地址前缀扩展方法.

多分支 Trie 树的查找过程与二分支 Trie 树的查找过程类似,是在每次结点访问过程中把到目前为止已经匹配上的最长地址前缀记录下来,直至到达叶子节点,搜索过程结束.尽管多分支 Trie 树的查找也是基于地址前缀长度的线性遍历法,但是因为多分支 Trie 树采用的步宽大于 1,所以其搜索效率大大提高了.

(2) 多分支 Trie 树的优化算法

多分支 Trie 树设计的关键是步宽的选择问题,也就是如何在算法查找速度和算法消耗的存储空间两个尺度上进行折衷的问题.在极端情况下,可以使用一层步宽为 32 的多分支 Trie 树.显然,在这种 Trie 树结构下,每次查找只需访问一次存储器操作就可以了,但是需要耗费 2^{32} 个表项存储空间.

一种比较合适的方案就是根据实际地址前缀的分布来选择合适的步宽,即根据二分支 Trie 树结构来构造多分支 Trie 树.例如,在图 1 中,前缀 d 的右分支子树就是一棵满二叉树,显然,我们可以用一棵 1 层 4 分支的 Trie

树来代替这棵 2 层二分支的 Trie 树.在这种情况下,进行前缀扩展转化是很直接的想法.但是,在其他情况下我们如何来进行步宽选择,就需要我们使用一些特殊的优化策略.Srinivasan 等人[11]提出了一种多分支 Trie 树的优化算法.其出发点是在多分支 Trie 树搜索深度固定的情况下,如何选择合适的步宽以使整个 Trie 树的存储空间达到最小.在算法设计中,他们根据实际地址前缀的特点使用了动态规划(dynamic programming)的思想来达到优化算法存储空间的目的.

(3) 多分支 Trie 树的压缩算法

多分支 Trie 树需要通过前缀扩展的方法来建立.在前缀扩展的过程中,前缀的转发信息被扩展到了 Trie 树的多个结点中,因此信息的冗余度非常高.不少研究者试图通过数据压缩方法来降低这种冗余度,从而达到降低算法占用存储空间的目的.

Run-Length 压缩法[12]是一种很简单的压缩方法,但是它能够很好地应用于地址前缀查找问题.Run-Length 压缩的做法是将一系列连续的、拥有同一信息的地址范围表示为该信息和记录该信息出现的次数.文献[13,14]给出了另外两种很有效的多分支 Trie 树的压缩算法.

(4) 多分支 Trie 树的硬件实现算法

Gupta 等人[15]根据 Internet 地址前缀的特点提出了一种多分支 Trie 树的硬件实现算法.在他们设计的算法实现环境下,每一次地址前缀查找过程最多只需要两次存储器访问,如果在硬件中采用流水线策略,那么整个地址前缀查找性能可以再提高一倍,即查找过程只需要一次存储器访问时间.文献[16]进一步对算法进行了改进,可以达到每秒 1 亿次路由查找.

2.3.3 地址前缀长度的二分查找法

为了减少查找的时间,Waldvogel 等人[17]提出了在地址前缀长度空间内进行二分查找的算法.如果能够在前缀长度空间内实现二分查找,那么整个查找性能就可以从 $O(W)$提高到 $O(\log W)$.但是,我们不能简单地将二分查找法直接应用到前缀长度空间内,例如,假设有 3 个地址前缀 0*,00*和 111*.假设现在需要查找 111,二分查找法首先从前缀长度为 2 的 Hash 表开始查找关键字 11,因为 Hash 表中只有表项 00,所以查找失败,而实际上在长度为 3 的 Hash 表中具有匹配关键字.为了解决这个问题,我们在前缀长度为 2 的 Hash 表中添加一个表项 11,这种表项被称为 marker.现在查找关键字 111,首先在前缀长度为 2 的 Hash 表中查找就会查找成功,然后二分查找法将会在前缀长度空间的下半部分继续查找.文献[18]对基于 Hash 的地址前缀长度二分查找算法进行了进一步的改进.

前缀长度的二分查找法将大大提高查找的效率:对于 IPv4 地址来说,查找过程最多需要 $\log_2 32=5$ 次存储器访问;对于今后将会使用的 IPv6 地址来说,查找过程最多需要 $\log_2 128=7$ 次存储器访问.

2.3.4 地址区间的二分查找法

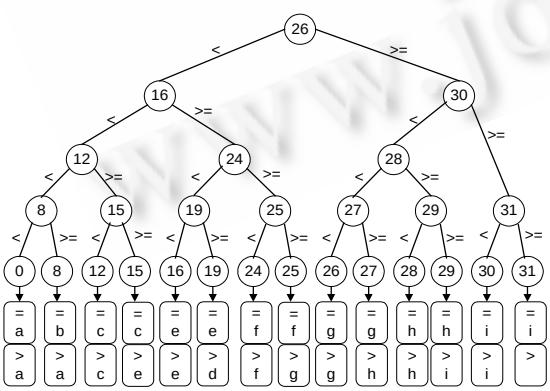

Fig.3 Binary lookup tree on address space
图 3 地址区间的二分查找树

地址前缀在整个地址空间内实际上代表了一段连续的地址区间,因此,如果能够知道目的地址所在的地址区间,通过该地址区间所对应的地址前缀,我们就可以得到最长匹配的地址前缀.由于前缀之间可能存在着包含关系,对应于地址空间将会出现某一段地址区域与多条前缀相对应.根据最长前缀匹配的原则,任何地址区域所对应的最长前缀应该是包含此地址区域的前缀中范围最窄的那一项.

考虑到每个区间都对应了区间的左、右两个端点,所以我们不妨使用区间的左端点来表示区间.例如,区间[12,15]就用 12 来表示,从而可以用一系列左端点集合来表示所有的区间.此时,最长前缀匹配查找就变成了在左端点集合中寻找离目的地址最近的左端点问题.显然,我们可以使用二分查找法来实现.图 3 就是左端点所对应的二分查找树结构图.为了能够区分查找关键

字与叶子结点关键字的关系,树的叶子结点处保存了大于和等于两种匹配情况.例如,查找目的地址 10110(22),首先比较查找树关键字 26,因为 22 小于 26,所以选取查找树的左分支;然后比较查找树关键字 16,同理选取右分支;依此类推,直到叶子结点(关键字为 19)为止.由于 22 大于 19,所以最长匹配的地址前缀为 *d*.

每条地址前缀都对应了两个端点,可以证明对于 N 个地址前缀,最多可以对应 $2N+1$ 个端点,因此地址区间二分查找法的查找算法复杂度为 $O(\log_2 2N)$.由于目前路由表的数目非常大(接近 100 000 条),因此,如果使用简单的二分查找方法,仍然不能取得很好的查找效果.Lampson 等人[19]在二分查找法思想的启发下,根据计算机中 Cache Line 的长度,提出了多路(multiway)查找的算法思想,使算法的复杂度降到 $O(\log_k 2N)$,对于目前 32 字节的 cache line 来说,k 等于 6.另外,如果考虑将多分支 Trie 树的思想引入,比如说在第 1 步首先采用步宽为 16,每个 Trie 树结点保存前 16 位相等的前缀关键字集合,然后在这些集合中再使用区间二分查找法,由于经过 Trie 分支,关键字集合的数目大大降低,因此,此时使用二分查找法则相对获得了更好的性能.

区间二分查找法的最大问题在于地址前缀的更新,因为插入或者删除每一个关键字有可能会影响多个原有的地址区间.文献[19]中分析得到,在最坏情况下,更新一个关键字会影响 $O(N)$个地址区间,因此,二分查找法的更新性能较差.

Gupta 等人在文献[20]中基于路由表访问频度信息,对基于地址空间的二分查找算法进行了进一步的研究,提出了在最坏性能有确定上界条件下的平均查找性能近似最优的算法.文献[20]中的数据结构需要定期进行重构,以反映路由表访问频度信息的变化.因此,在文献[21]中,作者设计了一种称为 Biased Skip List 的动态数据结构,可以进行自动更新,以反映路由表访问频度信息的动态变化.

2.3.5 路由查找的硬件实现方法

随着路由器接口速度的不断提高,使用软件方法实现高速的路由查找已经越来越困难了.目前已经广泛使用的 OC48 和 OC192 速率的接口要求路由查找的速度达到每秒几千万次,因此,我们需要从硬件设计的角度来考虑实现高速的路由查找方法.目前大部分路由器厂商如 Cisco,Juniper 都采用了基于硬件实现的路由查找部件.虽然软件的实现方法比较灵活,可扩展性好,可以适应今后协议的不断变化,但是从目前对于 IPv4 的了解来看,今后在地址体系结构以及最长前缀查找机制方面将不会发生很大的变化,因此从这个角度来看,我们必须采用速度快、集成化程度高的硬件实现方法.

目前使用最多的硬件实现方法是使用内容寻址存储器(content addressable memory,简称 CAM)[22]来进行快速的路由查找.CAM 能够在一个硬件时钟周期内完成关键字的精确匹配查找.我们常用的随机存储器通过输入地址来返回该地址处所对应的数据信息,但是 CAM 的访问方式不同,只需输入关键字的内容,CAM 就会将此关键字与 CAM 中所有的表项同时进行匹配比较,最后返回匹配表项在 CAM 中所对应的地址.

为了能够使用 CAM 来进行最长前缀路由的查找,我们可以为每一类可能的地址前缀长度使用一个 CAM,每个 CAM 保存对应长度的所有前缀集合.对于 IPv4 来说,一共需要使用 32 个 CAM.这种方法有一个明显的缺点,即在对地址前缀长度具体分布没有准确的了解之前,为了能够保证能够存储 N 个前缀表项,每个 CAM 都需要有 N 个表项的空间,因此,CAM 存储空间的利用率大大降低了.

为了能够克服上述方法的缺点,又有一种 CAM 实现机制——TCAM(ternary CAM)提出来.TCAM 的优点是它所保存的表项在长度要求上非常灵活,即可以在同一个 TCAM 芯片中保存任意长度的关键字表项.TCAM 通过保存关键字掩码的方式使得它可以保存任意长度的关键字表项,因此使用 TCAM 非常适合进行最长前缀路由的查找.目前不少工业界的厂商都在进行对 TCAM 的设计研究.由于可能存在多个表项匹配的情况,因此 TCAM 需要在这些匹配的表项中选取一个表项作为最后的查找结果.TCAM 规定在所有匹配的表项中选取地址最低的表项作为最后的结果.为了能够进行最长前缀路由的查找,我们需要保证在 TCAM 的低地址存储前缀较长的关键字表项,而在高地址区域存储前缀较短的关键字表项.当加入一条新的表项时,为了能够仍然保持关键字之间的顺序关系,就需要移动一些前缀长度比新表项要长的表项,因此,TCAM 的更新操作较为复杂.为了解决这一问题,文献[23]提出了一种改进的 TCAM 设计.这种 TCAM 可以任意排列 CAM 表项,其主要思想是在多个表项的并行比较之后引入“或”机制,并将结果反馈到表项区,以得到最长前缀匹配.文献[24]则提出了对现有的 TCAM 进行快速更新的算法.

除了采用 TCAM 设计路由查找引擎之外,研究人员在采用通用存储器进行路由查找方面也进行了深入的研究.文献[25]提出了一种基于存储器层次的 IP 路由查找性能模型.使用该模型可以对各种路由查找算法的存储器使用效率和性能进行比较.文献[26]设计了一种专门面向路由查找的 SRAM 体系结构,设计了相应的内存分配算法和流水线机制.该算法不仅可以支持 10Gbps 以上的接口速率,而且可以保证内存使用率接近 100%,同时具有更新迅速的优点.

3 路由查找算法的评价

3.1 路由查找算法的评价标准

3.1.1 查找速度

查找速度是决定路由查找算法性能以及路由器处理报文能力的最主要因素.常用的路由查找算法都是在软件环境下设计实现的.算法查找速度主要是由查找过程需要进行的存储器访问次数所决定,因此,通过统计算法在查找过程中存储器访问的次数就可以基本估计该算法的查找性能.

3.1.2 存储容量

路由器总是要求路由查找算法占用的存储容量应尽可能小,从而可以为算法的实现带来更大的灵活性.

3.1.3 预处理和更新速度

根据处理路由更新方式的不同,查找算法可以分为两种类型:一种称为动态算法,即发生路由更新(包括增加和删除)时,算法数据结构只需要在原有基础上相应更新,而不需要完全重构;另一种被称为静态算法,即当发生路由更新时,算法数据结构需要完全重新构建.与动态算法相比,静态算法的运行需要一个预处理过程(即重构过程),因此更新消耗的时间更长一些.Internet 路由的不稳定性[27]使得路由增加和删除的过程变得非常频繁,最差情况下每秒内路由变化可以发生几百次,因此要求路由查找算法的更新速度至少能够支持这种路由变化的速度.

3.1.4 算法实现的灵活性

路由查找算法可以软件实现,也可以硬件实现.因此,从查找算法的实现灵活性上考虑,我们希望算法能够同时具有软件和硬件实现方式.在未来的几年中,报文处理速度需要达到一个相当高的水平,硬件实现将是路由查找引擎的主要实现方法,因此,路由查找算法的设计应该考虑算法在硬件实现上的可能性.我们综合多种算法的优点提出的可配置的路由查找算法[28]就具有实现灵活的优点.

3.1.5 算法的可扩展性

在未来的几年中,新一代的 Internet 协议 IPv6 将会逐渐替代目前使用的 IPv4 协议,因此,在设计过程中需要考虑到算法的可扩展性,算法是否能够适用新的网络环境已经成为未来查找算法研究的重点之一.

3.2 路由查找算法的评价

3.2.1 算法复杂度的评价

表 2 给出了主要的路由查找算法的复杂度.假设地址关键字的长度为 W,地址前缀表中的前缀数目为 N,具体的复杂度分析过程不再详细讨论,可参考相应的论文.

Table 2 List of complexity of algorithms

表 2 算法复杂度列表

Algorithm①	Complexity of lookup②	Memory requirement③	Complexity of update④
Binary Trie⑤	$O(W)$	$O(N*W)$	$O(W)$
Path-Compressed Trie⑥	$O(W)$	$O(N)$	$O(W)$
Multibit Trie (k-stride)⑦	$O(W/k)$	$O(2^k*N*W/k)$	$O(W/k+2^k)$
Binary search on prefix lengths⑧	$O(\log W)$	$O(N*\log W)$	$O(N*\log W)$
Binary search on address space⑨	$O(\log 2N)$	$O(N)$	$O(N)$

①算法,②查找过程算法复杂度,③存储容量,④更新过程算法复杂度,⑤二进制 Trie 树,⑥路径压缩 Trie 树,⑦多分支 Trie 树(步宽为 k),⑧前缀长度的二分查找,⑨地址区间的二分查找.

3.2.2 算法的硬件实现灵活性评价

硬件实现要求算法具有以下几个特点.第一是硬件实现不具有软件实现存储空间管理的任意性,为了保证它的运行速度,它往往要求算法的数据结构应该尽量简单,同时组织有序,保证数据访问的快速有效;第二是硬件实现常常需要考虑算法能否被划分几个简单的操作部分,从而利用流水线来提高算法的速度.对于 trie 树(包括二进制 Trie 树和多分支 Trie 树)来说,由于它使用的树结点灵活度非常高,所以直接将 trie 树算法用硬件实现不太可取.前缀长度二分查找和地址区间二分查找算法由于使用二分法作为查找的基本思想,所以使用硬件实现不能获得良好的算法性能.

3.2.3 算法的可扩展性评价

下面对算法的可扩展性做简单的分析和比较.由于新一代 IP 协议 IPv6 使用长度为 128 的地址,所以它给本来已经很复杂的最长匹配前缀查找带来了更大的困难,对于目前的 32 位计算机系统来说,访问一个 128 地址就需要进行 4 次读写操作,查找速度会大大降低.Trie 树的查找复杂度为 $O(W/k)$,其中 $1\leq k\leq W$.如果 W 从 32 增加到 128,在 k 不发生变化的前提下,算法的查找性能将下降到原来的 1/4.为了能够提高查找性能,唯一的做法就是提高 k 的大小.但是,从算法存储容量的复杂度来看,增加 k 的大小势必会增加算法的存储空间,从而影响算法在实际系统中的使用.因此,Trie 树在可扩展性方面较差.由于前缀长度二分查找法的查找算法复杂度为 $O(\log W)$,因此相对而言它受地址长度变化的影响要小很多,对于 IPv4 地址,查找过程需要 5 次内存访问,到了 IPv6,查找过程需要 7 次内存访问,因此查找性能没有发生很大的下降.对于地址区间的二分查找法来说,查找的算法复杂度为 $O(\log 2N)$,因此,查找性能只与关键字的个数相关,而与关键字的长度无关,所以,地址区间二分查找法的性能也不会受到 IPv6 协议变化的影响.

限于篇幅,上述各类算法的实际运行性能的比较不再给出,可以参考文献[29].

4 结论和进一步的研究工作

本文分析了路由查找问题的主要特点和基本的解决方案,在此基础上全面总结了近年来研究人员提出的多种快速路由查找算法,并进行了比较.

随着 Internet 主干速率的进一步提高,核心路由器的接口速率将达到 10Gbps 乃至 40Gbps 以上.支持这样高的接口速率,软件路由查找算法已经力不从心.我们需要深入研究基于硬件存储器的路由查找算法,重点解决查找和更新速率的矛盾.另一方面,我们可以研究哪些软件算法可以采用硬件方式实现,以提高速率,满足核心路由器的要求.如何支持面向 IPv6 的高速路由查找也是值得进一步研究的课题.

References:

[1] Rekhter, Y., Li, T. An Architecture for IP Address Allocation with CIDR. RFC 1518, 1993.

[2] Fuller, V., Li, T., Yu, J., *et al*. Classless Inter-domain Routing (CIDR): an Address Assignment and Aggregation Strategy. RFC 1519, 1993.

[3] Hinden, R., Deering, S. IP Version 6 Addressing Architecture. RFC 2373, 1998.

[4] Partridge, C. Locality and route caches. In: Claffy, K., Garrett, M., Braun, H.-W., eds. Proceedings of the NSF workshop on Internet Statistics Measurement and Analysis. Diego, CA: NSF Press, 1996. 32~34.

[5] Morrison, D.R. PATRICIA——practical algorithm to retrieve information coded in alphanumeric. Journal of the ACM, 1968, 15(4):514~534.

[6] Sklower, Keith. A tree-based routing table for Berkeley Unix. Technical Report, Berkeley: University of California, 1993.

[7] Internet routing table statistics. In: Saha, Debanjan, ed. Proceedings of the IEEE INFOCOM. San Francisco: IEEE Computer Society Press, 2001. 1444~1453. http://www.merit.edu/ipma/routing_table.

[8] Cheung, G., McCanne, S. Optimal routing table design for IP address lookups under memory constraints. In: Ephremides, A., Tripathi, S., eds. Proceedings of the IEEE INFOCOM. San Francisco: IEEE Computer Society Press, 1999. 1437~1444.

[9] Cheung, G., McCanne, S. Dynamic memory model based framework for optimization of IP address lookup algorithms. In: Limb, J.O., ed. Proceedings of the IEEE ICNP. San Francisco: IEEE Computer Society Press, 1999. 11~20.

[10] Chiueh, T.C., Pradhan, P. High performance IP routing table lookup using CPU caching. In: Ephremides, A., Tripathi, S., eds. Proceedings of the

[11] IEEE INFOCOM. San Francisco: IEEE Computer Society Press, 1999. 1421~1428.

[12] Srinivasan, V., Varghese, G. Fast IP lookups using controlled prefix expansion. ACM Transactions on Computer Systems, 1999, 17(1):1~40.

[13] DegerMark, M., Brodnik, A., Carlsson S., *et al*. Small forwarding tables for fast routing lookups. Computer Communication Review, 1997,27(4):3~14.
[14] Nilsson, S., Karlsson, G. IP Address lookup using LC-Tries. IEEE Journal on Selected Areas in Communications, 1999,17(6): 1083~1092.
[15] Tzeng, Henry Hong-Yi., Przygienda, T. On fast address-lookup algorithms. IEEE Journal on Selected Areas in Communications, 1999,17(6):1067~1082.
[16] Gupta, Pankaj, Lin, S., Mckeown, N. Routing lookups in hardware at memory access speeds. In: Akyildiz, I.F., ed. Proceedings of the IEEE INFOCOM. San Francisco: IEEE Computer Society, 1998. 1240~1247.
[17] Huang, Nen-fu, Zhao, Shi-ming. A novel IP-routing lookup scheme and hardware architecture for multigigabit switching router. IEEE Journal on Selected Areas in Communications, 1999,17(6):1093~1104.
[18] Waldvogel, M., Varghese, G., Turner, J., *et al*. Scalable high speed IP routing lookups. Computer Communication Review, 1997, 27(4):25~36.
[19] Broder, A., Mitzenmacher, M. Using multiple Hash functions to improve IP lookups. In: Saha, Debanjan, ed. Proceedings of the IEEE INFOCOM. San Francisco: IEEE Computer Society Press, 2001. 1454~1463.
[20] Lampson, B., Srinivasan, V., Varghese, G. IP lookups using multiway and multicolumn search. IEEE/ACM Transactions on Networking, 1999,7(3):324~334.
[21] Gupta, Pankaj, Prabhakar, Balaji, Boyd, S. Near-Optimal routing lookups with bounded worst case performance. In: Kleinrock, L., ed. Proceedings of the IEEE INFOCOM. San Francisco: IEEE Computer Society Press, 2000. 1184~1192.
[22] Ergun, F., Mittra, S., Sahinalp, S.C., *et al*. A dynamic lookup scheme for bursty access patterns. In: Saha, Debanjan, ed. Proceedings of IEEE INFOCOM. San Francisco: IEEE Computer Society Press, 2001. 1444~1453.
[23] McAuley, A., Francis, P. Fast routing table lookup using CAMs. In: Vastola, K.S., ed. Proceedings of the IEEE INFOCOM. San Francisco: IEEE Computer Society Press, 1993. 1382~1391.
[24] Masayoshi, Kobayashi, Tutomu, Murase, Atsushi, Kuriyama. Longest prefix match search engine for multi-gigabit IP processing. In: Trahan, R.E., Leon, B.J., eds. Proceedings of the IEEE International Conference on Communications. San Francisco: IEEE Communication Society Press, 2000. 1360~1364.
[25] Shah, Devavrat, Gupta, Pankaj. Fast updating algorithms for TCAMS. IEEE Micro, 2001,21(1):36~47.
[26] Narlikar, Girija, Zane, F. Performance modeling for fast IP lookups. In: Biersack, E., Golubchik, L., eds. Proceedings of the ACM SIGMETRICS. Cambridge, Massachusetts, USA: ACM Press, 2001. 1~12.
[27] Sikka, Sandeep, Varghese, G. Memory-Efficient state lookups with fast updates. Computer Communication Review, 2000,30(4): 335~347.
[28] Labovitz, C., Malan, G., Jahanian, F. Internet routing instability. IEEE/ACM Transactions on Networking, 1998,6(5):515~527.
[29] Xu, Ke, Wu, Jian-ping, Wu, Jian. The analysis and design of fast route lookup algorithms for high performance router. In: Kim, Kiseon, ed. Proceedings of the IEEE International Conference on ATM. San Francisco: IEEE Computer Society Press, 2001. 320~325.
[30] Srinivasan, V. Fast and efficient Internet lookups [Ph.D. Thesis]. Washington University, 1999.

Survey on Routing Lookup Algorithms*

XU Ke, XU Ming-wei, WU Jian-ping, WU Jian

(*Institute of Network, Department of Computer Science and Technology, Tsinghua University, Beijing* 100084, *China*)

E-mail: xuke@tsinghua.edu.cn; xmw@csnet1.cs.tsinghua.edu.cn

http://netlab.cs.tsinghua.edu.cn

Abstract: With rapid development of Internet, the line speed of core router has reached 2.5Gbps and even 10Gbps beyond. This speed orders core router to be able to forward ten millions of packets per second. Route lookup is an important step of packet forwarding, so high speed IP address lookup algorithm is the key component of high speed packet forwarding. IP address lookup requires a longest matching prefix search. In the last couple of years, various algorithms for high-performance IP address lookup have been proposed. In this paper, the difficulty of route lookup is analyzed, and a survey of all kinds of lookup algorithms, which are compared in detail, is also presented. At last, some challenging open problems are identified.

Key words: route lookup; longest matching prefix; Trie-tree; Hash; CAM

* Received May 30, 2001; accepted September 14, 2001

Supported by the National Natural Science Foundation of China under Grant No.90104002; the National High Technology Development 863 Program of China under Grant No.863-306-ZD-07-01