学位论文精选

动态快速路由查找算法

刘亚林
（电信科学技术研究院，北京 100083）

[摘要] 给出了动态快速路由查找算法（DFR）的原理和实现。该算法采用特殊的数据结构来构建索引表，能支持动态插入、删除和更新路由；DFR 算法最多 4 次访存，最少 2 次访存，就能找到完整的路由信息。该算法不仅适用于软件实现，而且由于查找简单，也适合于硬件实现。

[关键词] 前缀扩展；DFR；路由；路由查找
[中图分类号] TP393 [文献标识码] A [文章编号] 1009－1742（2002）07－0060－09

1 引言

进入 20 世纪 90 年代以来，因特网业务发展迅猛。随着网络的发展，IP 在网络中起着越来越重要的作用。当前，整个网络呈现用户业务 IP 化、IP 业务综合化、IP 技术多样化的趋势。

为了适应未来宽带网络的发展，世界上各大电信厂商纷纷研制高性能路由器。但是，因特网的迅猛增长使得路由器中的路由表迅速增长。路由表规模的增长限制了路由器的转发速率。因而路由查找技术成为路由器的关键技术之一，是衡量路由器性能的重要指标。为了解决路由器速度瓶颈，近年来，快速路由查找技术是一个热门课题，也提出了不少路由查找算法，如 LC 算法[1]，页面压缩（page compress）算法[2]，采用硬件的分级索引算法[3,4]等。这些算法虽然优化了某些性能，但都在路由查找的时间复杂度、空间复杂度以及插入、删除和更新路由的速度等三个方面有一定的局限性。一种好的算法应该在这三者之间寻求一种好的平衡。

动态快速路由查找算法（DFR）是针对现有算法的不足提出的。它利用前缀扩展的特性，通过构造特定的数据结构，使得算法支持动态插入、删除和更新表项。该算法只需最多 4 次访存就能完成 1 次查找。通过压缩技术，该算法需要的存储空间很小，不仅适合于硬件实现，也适合于软件实现。

2 DFR 算法原理

DFR 算法的实现原理主要依据前缀扩展的特性。将一个前缀用前缀/长度来表示，假定有前缀 p/l，扩展后的前缀长度为 m，且 $m>l$。设前缀 p 的二进制表示为

$$p_1p_2\cdots p_l,\ p_i = 0 \text{ 或 } 1。$$

将扩展后的前缀的二进制表示为

$$p'_1p'_2\cdots p'_lp'_{l+1}p'_{l+2}\cdots p'_m,\ p'_i = 0 \text{ 或 } 1。$$

如果 $p_1p_2\cdots p_l = p'_1p'_2\cdots p'_l$，且 $p'_{l+1}p'_{l+2}\cdots p'_m$ 构成一个完备集合，则称前缀 p 为前缀扩展。

前缀扩展的目的是为了把任意长度的前缀变成固定长度的前缀，扩展的位数可以是任意值。一个前缀经扩展后形成一个集合，且集合中的前缀在数值上是连续的，利用这个特性来存放前缀集合，这种连续性将会简化路由查找操作。通过前缀扩展，在进行路由查找时可以一次比较多位。根据 IP 包的目的地中对应位的值，查到对应位的前缀。

[收稿日期] 2001－10－11；修回日期 2002－01－15
[作者简介] 刘亚林（1973－），男，湖北黄梅县人，硕士，大唐电信光通信公司助理工程师

前缀扩展可以按需分段扩展。例如，对路由器中所有的前缀可以分三段来进行扩展，第一段可以将所有前缀 <16 b 的前缀扩展成 16 b 位长，第二段将长度为 17 b 到 24 b 的前缀都扩展成 24 b 位长，第三段可以将所有长度 >24 b 的前缀扩展成 32 b 位长。前缀扩展增加了前缀表项数，增加表项数意味着需要更大的存储空间，但是前缀扩展却能成倍地提高路由查找的速度。

前缀扩展的主要特性包括长度特性、覆盖性和无交叠性。

2.1　长度特性

长度特性是指前缀扩展所覆盖的范围长度刚好是 2 的指数次幂。如果一个前缀扩展 x 位，那么它所覆盖的范围长度将是 2^x。

例如有两个前缀 01/2 和 011/3，将这两个前缀分别扩展成 4 b 位长，形成的前缀集合 P 为

$$P = \{0100,0101,0110,0111\}。$$

如果一个前缀 p 开始扩展了 x 位，形成一个前缀集合 P，如果改为扩展 x' 位，其中 $x' < x$，则相当于将所有的前缀缩小 $2^{x-x'}$ 倍。所有在数值上不是 $2^{x-x'}$ 的整数倍的前缀都是扩展所形成的，因而也不是路由表中真正存在的前缀，缩小时则将这些前缀忽略。缩小时必须保证缩小的位数不大于扩展的最小位长，即不能影响路由表中真正的前缀，也就是保证缩小后所有的前缀都必须至少有 1 位扩展位或没有扩展。同样地，当有多个前缀时，缩小时不能影响长度最长的前缀。

2.2　覆盖性

前缀扩展的覆盖性是指，如果将一个前缀用前缀/长度/输出端口来表示，假如存在两个前缀 $p_1/l_1/o_1$ 和 $p_2/l_2/o_2$，将 p_1 和 p_2 分别进行前缀扩展，形成两个新的集合 P_1 和 P_2，如果集合 P_2 是 P_1 的子集，那么 P_1 和 P_2 的交集部分的输出端口应该为 P_2 的输出端口 o_2。

根据最长前缀匹配原则，在 P_1 和 P_2 的交集部分用 P_2 的输出端口 o_2 时，则需要满足：

$$l_1 \leqslant l_2。 \tag{1}$$

假设前缀 p_1 和 p_2 都被扩展成 m 位长，扩展的位数分别为 x_1 和 x_2，则：

$$x_1 = m - l_1, \tag{2}$$

$$x_2 = m - l_2。 \tag{3}$$

根据前缀扩展的长度特性，p_1 和 p_2 扩展后所覆盖的范围分别为 2^{x_1} 和 2^{x_2}。假定集合 P_1 和 P_2 中的最小值分别是 S_1 和 S_2，由于集合 P_2 是 P_1 的子集，则需要满足以下两式：

$$S_2 \geqslant S_1, \tag{4}$$

$$S_1 + 2^{x_1} \geqslant S_2 + 2^{x_2} \tag{5}$$

设 $S_2 = S_1 + \Delta x$，代入式 (5) 有

$$2^{x_1} \geqslant \Delta x + 2^{x_2}, \tag{6}$$

其中 $\Delta x \geqslant 0$，故有

$$x_1 \geqslant x_2。 \tag{7}$$

由式 (2) 和式 (3) 可知 $l_1 \leqslant l_2$。

从数轴上看，一个前缀经扩展后对应数轴上的一条线段，每条线段就代表一个前缀经扩展后所形成的前缀集合，集合中的最小值对应线段的起点，而且扩展后集合中所有的前缀都有相同的下一跳 (Next-hop)。

用前缀/长度/下一跳来表示一个前缀的属性。假定有两个前缀 01/2/3 和 011/3/5，将这 2 个前缀分别扩展成 4 b 位长，形成 2 个前缀集合：

$$P_1 = \{0100,0101,0110,0111\},$$

$$P_2 = \{0110,0111\}。$$

它们在数轴上的表示如图 1 所示。

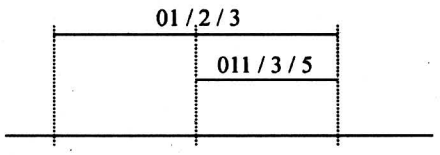

图 1　多个前缀扩展的数轴表示

Fig.1　Presentation of several prefix expansion on the axis

如果要为 0111 查找下一跳，根据最长匹配原则，相匹配的前缀应该是 011/3/5，而不应该是 01/2/3，因为 011/3/5 匹配得更长。

前缀扩展的覆盖性使得路由查找支持最长前缀匹配。在进行前缀扩展时，扩展后的重叠区域的下一跳路由信息将会是该区域内有最大长度的前缀，从而保证了路由查找时的最长匹配。同时在进行路由更新时也将会只对局部范围进行更新，从而简化了路由更新操作。当两个前缀经扩展后，如果一个前缀集合是另一个前缀集合的子集，而且这两个前缀有相同的下一跳路由信息，在生成路由转发表时，可以将这两个前缀进行合并，在数轴上看，将只有一个前缀的下一跳路由信息。

2.3　无交叠性

如果前缀用前缀/长度来表示，假如存在 2 个

前缀 p_1/l_1 和 p_2/l_2，将 p_1 和 p_2 分别进行前缀扩展，形成2个新的集合 P_1 和 P_2，如果集合 P_2 中最小的元素属于 P_1，但集合 P_1 中最小的元素不属于 P_2，那么 P_2 中最大的元素也属于 P_1。

假设前缀 p_1 和 p_1 都扩展成 m 位长，集合 P_1 和 P_2 中最小元素的二进制表示分别为：

$$a_1a_2\cdots a_l \underbrace{00\cdots0}_{m-l_1\text{个}0},\ a_i = 0 \text{ 或 } 1;$$

$$b_1b_2\cdots b_{l_1}\underbrace{b_{l_1+1}\cdots b_{l_2}}_{\text{不全为}0}\underbrace{00\cdots0}_{m-l_2\text{个}0},\ b_i = 0 \text{ 或 } 1。$$

由于 $b_1b_2\cdots b_{l_1}b_{l_1+1}\cdots b_{l_2}\underbrace{00\cdots0}_{m-l_2\text{个}0}\in P_1$，故

$$a_i = b_i,\ i = 1,\cdots,l_1。 \tag{8}$$

根据前缀扩展的定义，集合 P_1 和 P_2 中最大元素的二进制可以分别表示为：

$$a_1a_2\cdots a_{l_1}\underbrace{11\cdots1}_{m-l_1\text{个}1},\ a_i = 0 \text{ 或 } 1;$$

$$b_1b_2\cdots b_{l_1}\underbrace{b_{l_1+1}\cdots b_{l_2}}_{\text{不全为}0}\underbrace{11\cdots1}_{m-l_2\text{个}1},\ b_i = 0 \text{ 或 } 1。$$

由式（8）可知

$$a_1a_2\cdots a_{l_1}\underbrace{11\cdots1}_{m-l_1\text{个}1} \geqslant b_1b_2\cdots b_{l_1}\underbrace{b_{l_1+1}\cdots b_{l_2}}_{\text{不全为}0}\underbrace{11\cdots1}_{m-l_2\text{个}1}。 \tag{9}$$

由前缀扩展的完备性可知

$$b_1b_2\cdots b_{l_1}\underbrace{b_{l_1+1}\cdots b_{l_2}}_{\text{不全为}0}\underbrace{11\cdots1}_{m-l_2\text{个}1}\in P_1。$$

这就说明如果集合 P_2 中最小的元素属于 P_1，但集合 P_1 中最小的元素不属于 P_2 时，P_2 中最大的元素也属于 P_1。前缀扩展的无交叠性说明不会出现图2所示的情况。

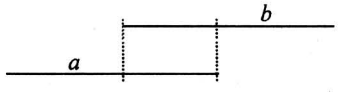

图2 前缀扩展的无交叠性

Fig.2 Non-overlapped characteristic of prefix expansion

前缀扩展的无交叠性使得两个前缀要么没有重叠，要么一个前缀扩展后的集合是另一个前缀扩展后的集合的子集。如果知道前缀扩展的起点位置和扩展的位数，就很容易计算终点位置。在插入、删除和更新路由时，可以利用前缀扩展的无交叠性来进行局部操作，从而减小操作范围，提高插入、删除和更新速度。

3 DFR算法设计

DFR算法采用两级索引3次查找结构来构建索引表。

3.1 一级索引表的结构

DFR算法将IP地址的前16 b的完备集合形成一级索引，共65536个表项构成一个线性表，每个表项用4 B表示，将4 B分为4部分：标记、掩码长度/指针、输出端口、偏移量。格式如图3所示。

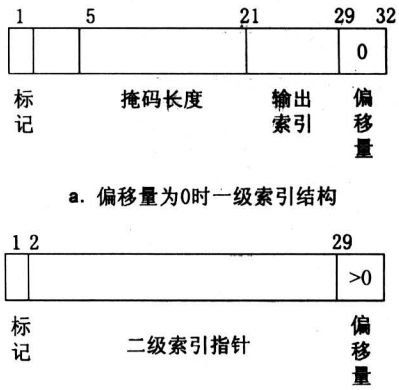

图3 一级索引表结构

Fig.3 Structure of the first level index table

标记 标记用来识别该项是否存在一个前缀经扩展后形成的起点位于该项。当标记位为1时，表示有某个前缀长度 ≤16 b的前缀经扩展后形成的起点位于该项，否则标记为0。

掩码长度/指针 掩码长度/指针域用来记录掩码信息或下一级索引的头指针，该域存放的信息与偏移量的值相关。当偏移量为零时，没有二级索引表，如果有某个前缀经扩展后的起点位于该表项，那么一级索引的第5—20位（16 b）用于记录起于该点的前缀的长度。将这16 b分为4组，每4位记录一个掩码，共记录4个掩码，并将掩码长度域中的掩码按自左至右依次减小的顺序放置。

当偏移量>0时，此时一级索引中存放的是指针，第1位仍然用作标记，低4 b用作记录偏移量，将第2—28位用于存放二级索引的指针，指针域共27 b。此时如果需要查找二级索引就可以通过指针域中的指针来查找下一级索引表。

输出端口 输出端口用于存放下一跳信息。将一级索引表项中的第21—28位用于记录端口索引。在进行路由查找时，查找到输出端口就能查找到完整的下一跳路由信息。

偏移量　偏移量是前缀长度减去 16 b 所得到的值，这里要求前缀长度 > 16 b，如果前缀长度 < 16 b，可以直接在一级索引中存放输出端口，此时偏移量不起作用。设前 16 b 相同的前缀形成的集合为 P，前缀 P_i 是集合 P 中的元素，设 P_i 的前缀长度为 l_i，则偏移量 $k_i = l_i - 16$，设 $k_{max} \in k_i$ 且 $k_{max} \geqslant k_i$，因此 k_{max} 是集合 P 中最大的偏移量，一级索引中的偏移量域中存放的是这个最大的偏移量。当偏移量为 0 时，表示没有二级索引，此时可以直接通过一级索引中的输出端口查找到下一跳，当偏移量 > 0 时，表示该一级表项中有前缀长度 >16 b 的前缀存在，此时的路由信息存放在二级索引表中，需要通过二级索引表来进行查找。

3.2　二级索引表结构

当偏移量 > 0 时，存在二级索引。二级索引结构如图 4 所示。二级索引主要包括码字数组 CWA 和压缩下一跳数组 CNHA 两大部分，其中 CWA 由多个码字 CW 组成，CNHA 由多个下一跳索引 NHI 组成。当一级索引的标记位为 1 时，二级索引的头部有一个 2 B 的掩码记录 MR，二级索引头部的掩码记录用来记录一级索引中长度 < 16 b 的前缀的掩码长度，当一级索引的标记位为 0 时，二级索引中不需要头部的 MR。

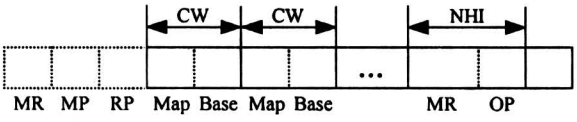

图 4　二级索引表结构

Fig.4　Structure of the second level index table

每个 CWA 包含一个 Map 和 Base。Map 是由于采用压缩技术后形成的一个位映射，Map 中的每一位顺序对应完整的二级索引表的每个表项，每个 1 对应一个前缀扩展的起点或终点的下一位。Base 表示此 CWA 之前的 Map 中累积的 1 的个数，第一个 CWA 的 Base 为 0。每个 Map 和 Base 的长度都是 2 B (16 b)。当偏移量 < 5 时，由于最多有 $2^4 = 16$ 个二级索引表项，此时用一个 Map 来存放二级索引表的位映射就足够了，因此可以省去 2 字节的 Base。

每个 CNHA 包含掩码记录 MR 和输出端口 OP 两部分。MR 用 2 B 来表示，每 4 b 记录一个掩码长度，共能记录 4 次重叠。注意，NHI 中的掩码记录和二级索引表头部的掩码记录的意义不一样。NHI 中的掩码记录是用来记录二级索引表中前缀的偏移量的，而头部的掩码记录是用来记录一级索引表中的前缀长度的（如果一级索引的标记位为 1），因为当存在二级索引表时，一级索引表中存放的是二级索引表的头指针，不能存放掩码记录，此时把一级索引表中对应的掩码记录转移到二级索引表的头部。输出端口用 1 个字节来表示，输出端口只是一个下一跳路由的索引，通过输出端口可以找到下一跳路由信息。每个 Map 中的 1 顺序对应一个下一跳索引（NHI）。

从图 4 中可以看到，在第一个 CW 之前有一个最大端口数（MP）和一个实际端口数（RP），MP 和 RP 只有当偏移量 > 4 时才存在。最大端口数用于存放当前 CNHA 中能容纳的最大端口索引的数目，实际端口数用于存放当前 CNHA 中实际存放的端口索引的数目，MP 和 RP 各为 2 B。设置这两个域的目的是为了提高插入和删除路由的速度，如果每次添加或者删除路由表项都重构二级索引表，使得效率很低，为了尽量减小重构二级索引表的频率，为 CNHA 预留一定的空间，只有当预留的空间全部用完时才重构二级索引表。当一级索引的偏移量 < 5 时，由于最多只有 $2^4 = 16$ 个二级索引表项，也就是只有 16 个位映射对应一个 Map，因此存在的 NHI 的数目很小，如果出现添加或者删除路由表项，这种操作的复杂度并不大，因此当一级索引的偏移量 < 5 时出现添加或删除表项的操作将重构二级索引表，此时也就不需要为 CNHA 预留空间。

3.3　前缀扩展范围的计算

由于前缀扩展的覆盖性，在进行路由插入、删除和更新操作时需要知道前缀扩展的范围，前缀扩展范围的计算可以利用前缀及其掩码来计算扩展范围的起点和终点。一级索引表和二级索引表都应该分别计算前缀的扩展范围。

对二级索引表，将前 16 b 相同的前缀形成一个集合，记为 P。如把 192.1.X.X 的有相同前缀 192.1 的 IP 地址归为一类。这些 IP 地址在一级索引里位于同一组，从具有相同一级索引的 IP 地址（即前 16 b 相同）中选取一个最长的前缀，设其前缀长度为 l_m，$l_m - 16$ 为偏移量，设为 k，每个一级索引最多对应 2^k 个二级索引表项。这样，二级索引的表项数将是动态变化的，随着前缀长度的不同而不同。

设集合 P 中前缀 p_i 的前缀长度为 l_i，其输出端口为 h_i，并设

$$o_i = l_i - 16,\ k = \max\{o_i \mid p_i \in P\},\quad (10)$$

其中 P 是前 16 b 相同的前缀的集合。

设 $p_i = a.b.x.y$，用 x_0，x_1，x_2，…，x_{15} 代表 $x.y$ 的二进制形式。如果将 P_i 的掩码从第 17 位起截取 k 位，并用 s_0，s_1，s_2，…，s_{k-1}代表截取的掩码的起点，其中，当 $j<o_i$ 时，$s_j=1$，当 $j\geqslant o_i$ 时，$s_j=0$。用 e_0，e_1，e_2，…，e_{k-1}代表截取的掩码的终点，其中，当 $j<o_i$ 时，$e_j=0$，当 $j\geqslant o_i$ 时，$e_j=1$。由于前缀扩展的起点位置所对应的扩展位都为 0，因此有：

$$M(S_i^0) = (x_0,x_1,x_2,\cdots,x_{k-1}\ \text{AND}\ s_0,s_1,s_2,\cdots,s_{k-1})。\quad (11)$$

根据前缀扩展的特性，扩展后的终点位置对应的扩展位必定是全 1，因此可以计算终点对应的物理存储地址为：

$$M(E_i^0) = (x_0,x_1,x_2,\cdots,x_{k-1}\ \text{OR}\ e_0,e_1,e_2,\cdots,e_{k-1})。\quad (12)$$

对每个 IP 地址 P_i 都有一对 S_i^0 和 E_i^0 与它相对应，处于地址段 $[M(S_i^0),M(E_i^0)]$ 中的 IP 地址都有相同的下一跳路由。由于经前缀扩展后，就会存在地址重叠的问题，根据前缀扩展的覆盖性，长度越短的前缀覆盖的范围越大，如果一个前缀 p_2 经扩展后处于 p_1 的扩展范围之内，p_1 的下一跳路由为 h_1，p_2 的下一跳路由为 h_2，则处于 p_2 的扩展范围之内的 IP 地址的出口应该为 h_2。

对一级索引表同样可以采用以上方法来计算前缀扩展的起点和终点，根据计算的起点和终点位置对起点与终点之间的端口进行更新，从而减小了更新范围和更新时间。

3.4 压缩原理与压缩方法

假定一级索引的偏移量为 k，当直接构建时，二级索引的表项数将是 2^k，前缀扩展后的前缀集合中每个前缀对应 1 个二级索引表项。在每个前缀扩展的范围内都有相同的输出端口，如果有些范围没有被扩展后的前缀所覆盖，将在其中填入默认输出端口。

当偏移量较大时，直接构建的二级索引表比较大，由于在前缀扩展的范围内输出端口相同，在相应的物理空间中对应连续的端口索引号，如果连续的端口索引号能用一个端口号来表示，就可以省下大量的空间。方法是用一个位来标记每个输出端口，连续端口的第一个端口对应的位被置为 1，以后连续相同的端口都置 0，每两个相邻的 1 之间的位对应相同的输出端口。通过压缩，形成了一个位映射。将位映射分成每 16 b 一组，每组后面跟一个 2 B 的 Base，每 16 b 的映射（Map）和 2 B 的 Base 就形成一个压缩码字 CW。

假如有位映射为 1000000010000000000000010001000…，那么形成的码字数组如图 5 所示。第一个 Base 为 0，由于第一个 CW 的 Map 中有 2 个 1，所以第二个 CW 的 Base 为 2，当前 CW 中的 Base 为前面所有 CW 中 1 累积的个数。

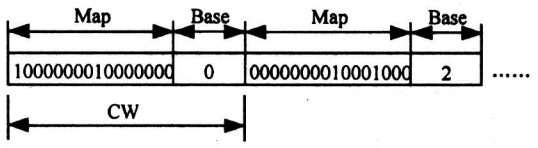

图 5 CWA 的形成

Fig.5 Formation of CWA

4 DFR 算法实现

4.1 算法数据结构

为了减小存储空间和实现操作上的便捷性，算法采用特殊的数据结构。一级索引表的每个表项用 4 B 来表示，根据标志位的值和偏移量的不同，各个字段的安排有所不同。

二级索引表将 CWA 和 CNHA 作为一个整体，放在一个线性表中，CWA 的空间可以计算，假设偏移量是 k，那么 CWA 所需要的字节数为 $4\times 2^{k-4}=2^{k-2}$，其中 $k>4$；如果 $k\leqslant 4$，那么 CWA 结构需要作一些变化，此时由于只有 1 个 Map，因此省去 Base，这时 CWA 所占的字节数是 2 B 的空间。在最后一个 CW 之后是第一个 NHI，每个 NHI 包括一个 2 B 的 MR 和一个 1 B 的下一跳端口索引号，为了能重构二级索引，每个 NHI 都采用这种结构。当 $k>4$ 时，在二级索引的头部有一个最大端口数和一个实际端口数，分别为 2 B，如果一级索引的标志位为 1，在最大端口数之前有一个 2 B 的掩码记录，用来记录一级索引中对应的掩码。可以通过标志位来判定二级索引是否存在一个头部的掩码记录，通过偏移量来判定是否存在最大端口数和实际端口数。

定义 CW 的数据结构为：

typedef struct-IndexToCWA

{
UINT2 u2Map；记录 1 个 Map 的值；
UINT2 u2Base；记录 1 个 Base 的值；
} IndexToCWA；
定义 NHI 的数据结构为：
typedef struct-IndexToCNHA
{
UINT2 u2MskLen；记录掩码，2 B 长；
UINT1 u1Port；记录输出端口索引；
UINT1 alignment1；对齐字节边界；
} IndexToCNHA；
添加路由的信息数据结构为：
typedef struct-RtTransInfo
{
UINT4 u4Dest；目的地址；
UINT4 u4Mask；路由的掩码；
UINT4 u4NextHop；下一跳地址；
UINT1 u1IfIndex；端口索引；
UINT1 alignment1；对齐字节边界；
UINT2 alignment2；对齐字节边界；
} tRtTransInfo；

以上数据结构的定义采用与 Future 协议源码的定义风格相一致的定义方式，是为了兼容和将来的升级替换。

4.2　一级索引表的构建

构建一级索引表时，将所有的前缀扩展到 16 b 位长。在一级索引实现插入操作时，首先计算前缀（假设为 a）扩展的起点和终点，然后在起点和终点间逐个表项操作，对每个表项，首先检查标志位，如果该位为 1，说明有另一个前缀（假设为 b）也起于该点，如果 a 的扩展范围大于 b，对前缀 b 扩展范围内的端口索引不改变。对前缀 b 扩展范围内的输出端口进行保护的最好方法是确定它的扩展范围，并跳过这个范围。

如果在添加的起点位置还有其他的前缀也起于该点，则首先需要确定是更新路由还是添加新的路由，如果前缀的掩码长度与掩码记录中某个掩码长度相同，则说明是更新情况，此时需要确定更新范围，如果有某个前缀的长度大于当前前缀长度，则计算该前缀的终点，然后从终点加 1 的位置起开始更新，后面的更新操作同上。

4.3　二级索引表的构建

在二级索引表中也存在前缀扩展的覆盖问题和起点重叠的问题，由于二级索引的结构和一级索引表不同，并采用了压缩技术，因此在实现路由插入、删除和更新时需要首先找到起点和终点位置对应的位映射位和对应的下一跳索引（NHI）。

4.3.1　插入操作　插入操作主要包括两种情况：更新和插入新路由。

1）*更新操作*

更新操作首先计算前缀扩展的起点 $m(S_i^0)$ 和终点 $m(E_i^0)$，并根据 $m(S_i^0)$ 的值计算起点所在的码字 $W(S_i^0)$ 和在对应的 Map 中的位 $B(S_i^0)$，根据 $W(S_i^0)$ 中 Base 的值和 Map 中 $B(S_i^0)$ 之前 Map 中 1 的个数来确定起点对应的下一跳索引(NHI)，然后根据 NHI 中的掩码记录 MR 来确定是否是更新操作。类似地，计算终点 $m(E_i^0)$ 所在的码字 $W(E_i^0)$ 和对应的 Map 中的位 $B(E_i^0)$，然后更新码字 $W(S_i^0)$ 和 $W(E_i^0)$ 所对应的起点和终点的位 $B(S_i^0)$ 和 $B(E_i^0)$ 之间的内容及其对应的端口索引。

2）*插入新路由*

如果插入路由的偏移量 k 小于最大偏移量 K_m，且 $K_m>4$，则判断起点与终点的重叠情况，根据判断确定插入的端口索引的个数来确定是否需要重新构建二级索引表。如果 $K_m \leqslant 4$ 或者 $k>K_m$，重新构建二级索引表。

判断起点和终点是否重叠的方法是检查起点对应的 Map 中的位被置为 1，则说明起点重叠，否则不重叠；终点重叠的判别是检查终点对应位的下一位被置为 1，则说明重叠，否则不重叠。如果起点重叠而终点不重叠，或者起点不重叠而终点重叠，则需要增加 1 个输出端口索引；如果起点和终点都不重叠，则需要增加 2 个输出端口索引。插入新路由不会出现起点和终点都重叠的情况，因为这可以作为更新路由的情况来处理。

对 $k \leqslant K_m$ 且 $K_m>4$ 的情况，二级索引的头部有 1 个最大端口数 MP 和实际端口数 RP。此时首先判断添加的端口索引的个数 n，如果 RP 中的值与 n 的和大于 MP 中的值，此时重构二级索引表，否则，更新二级索引表。

当起点重叠时，只需在起点所对应的 NHI 中的掩码记录中添加前缀的偏移量值，从起点到终点间的操作同更新情况，要考虑前缀扩展的覆盖性，在终点位置，将终点对应位的下一位置 1，因为下一位所对应的下一跳索引不同，并在终点对应的 NHI 之前插入 1 个新的 NHI。

对终点重叠情况，类似于起点重叠情况，不同的是要将起点对应的位设置为1，并在起点对应的NHI之后添加1个下一跳路由信息，并将前缀的偏移量添加到掩码记录中，并将它后面所有的NHI后移1个NHI的位置。

起点和终点都不重叠的情况属于以上2种情况综合。

对 $k \leqslant K_m$ 且 $K_m > 4$，但是RP的值与添加的下一跳路由数之和大于MP的值的情况，需要重新构建二级索引表。重构二级索引表时只需增加相应的输出端口索引的个数，其他和原来的二级索引表一样，将原二级索引表的CNHA之前的部分都复制到新表中，根据重叠情况，将起点和终点对应位的下一位相应地置1，然后将原表的CNHA的相应部分复制到新表中，并在起点与终点相对应的位置添加NHI，这样就构成了新的二级索引表。

对 $k > K_m$ 且 $k \leqslant 4$ 或者 $k \leqslant K_m$ 且 $K_m \leqslant 4$ 的情况，只用1个Map就能存放所有的信息，此时不需要码字中的Base，也就是此时只用1个2 B的Map来存放位映射就可以了。

对 $k > K_m$ 且 $K_m > 4$ 的情况，需要重新分配CWA的空间，分配时根据 k 的值来计算，CWA所需的空间是 2^{k-2} 个字节，再根据RP的值与要添加的端口索引的个数来确定需要分配的最大端口数的值，从而可以计算需要分配的二级索引表的空间大小。对这种情况，关键是如何将原表的CWA构建成新表的CWA。从原表的CWA构建成新表的CWA需要利用前缀扩展的长度特性，前缀扩展的长度特性表明，前缀每扩展1位，扩展后的起点对应的值扩大1倍。构建时，可以先将新表全部清零，然后将原表中每个1对应的位置扩大1倍，并将新表中相应的位置设置为1。

4.3.2 删除操作 删除操作需要在删除前缀的扩展范围内填充一个适当的输出端口，因此删除操作要解决的关键问题就是确定一个替代输出端口。

1）一级索引表前缀的删除

删除一级索引表中的前缀时，首先要检查删除前缀对应的起点和终点的重叠情况。起点重叠只要检查起点对应的NHI的掩码记录就可以。终点重叠需要检查删除前缀的扩展范围内所有表项，检测的最大范围是256。如果有某个前缀的起点位于删除前缀的扩展范围内，计算其前缀扩展的终点，如果终点和删除前缀的终点相同，则说明终点重叠。

对起点重叠、终点不重叠的情况，将起点对应的表项的掩码记录中要删除前缀的掩码删除，并将删除前缀扩展范围内的相应表项的输出端口替换为适当的输出端口就完成了操作。

对起点不重叠、终点重叠的情况，将起点对应的输出项的标志位清零，并将起点到终点间所有项的输出端口都改为适当的替代输出端口。

对起点与终点都重叠的情况，用将删除前缀的起点对应的表项中的掩码记录更新，将要删除前缀的偏移量从掩码记录中删除，然后在起点与终点间更新相应表项的输出端口索引就完成删除操作。

对起点和终点都不重叠的情况，删除前缀要求将起点对应的表项的标志位清零，再查找替代输出端口，然后将起点到终点之间所有表项的输出端口索引改为替代输出端口。

2）二级索引表前缀的删除

假设删除前缀的偏移量为 k，最大偏移量 K_m，当 $K_m > 4$ 时，设最大端口数MP的值为 m，实际端口数RP中的值是 n。删除二级索引表中的前缀主要有以下几种情况：

情况1 $k \leqslant K_m$，且 $K_m \leqslant 4$。此时需要重新构建二级索引表，二级索引表的构建需要根据重叠情况来分配存储空间。由于 $K_m \leqslant 4$，此时只有一个2 B的Map，没有Base，二级索引表的头部也没有MP和RP，因此对Map的操作比较简单，此时的删除操作主要是对NHI进行操作。

情况2 $k < K_m$，且 $K_m > 4$。此时二级索引表的头部有一个MP和一个RP，当删除一个前缀时，要根据删除前缀的重叠情况和RP的值来判断是否需要重新构建二级索引表。假设要删除的NHI的个数为 d，当起点重叠而终点不重叠或起点不重叠而终点重叠时，$d=1$；起点与终点都不重叠时，$d=2$；起点与终点都重叠时，$d=0$。如果 $n-d$ 小于某一常数 c（比如10），此时只需要对二级索引表进行更新就可以，如果 $n-d \geqslant c$，则需要重新构建二级索引表。

情况3 $k = K_m$，且 $K_m > 4$。由于 $k = K_m$，说明删除的是具有最大偏移量的前缀，具有最大偏移量的前缀在数轴上的表示是一个点，因此需要找到某个前缀，它的偏移量不大于最大偏移量，但大于所有其他前缀的偏移量，然后再将它作为新的最大偏移量（设为 k'）来构建二级索引表。如果 $k'=k$，则采用和情况2同样的方法来判断是否需要

重新构建二级索引表，操作方法也完全一样。这里只讨论 $k' < k$ 时重构二级索引表的情况。

$k' < k$ 时，需要缩小二级索引表，设删除前缀的值为 v，MP 的值为 m，RP 的值为 r，要删除的 NHI 的个数为 n，同样地，根据重叠情况来确定 n 的值。如果 $r - n < c$（c 为某个常数），则重新分配的二级索引表的空间为

$$2^{k'-2} + 3m + 4。$$

如果 $r - n > c$，则重新分配的二级索引表的空间为

$$2^{k-2'} + 3(m - c) + 4。$$

$k' < k$ 时重构二级索引表的关键是如何确定 k'及如何将原表中的 1 映射到新表中。

k'确定的方法是从第一个 NHI 开始，逐个检测 r 个 NHI 的掩码记录（不包含删除前缀对应的 NHI），找到一个最大的偏移量，它就是 k'。

将原表中 Map 中的 1 映射到新表中，需要利用前缀扩展的长度特性来确定它在新表中的位置。假设某个前缀的起点对应的值是 w，前缀扩展缩小的倍数 $e = 2^{k-k'}$，则该前缀的起点在新表中对应的位置为 $p = w/e$（一定是整数），则 p 对应的 Map 为 $p/16$（向下取整），对应的位 b 为 p mod 16，将相应的位设置为 1。

由于在删除操作时，除了要删除的前缀外，其他所有前缀都至少扩展了 $k - k'$位，每个前缀在原表中覆盖的范围至少是 $e = 2^{k-k'}$，因此检测位时可以一次跨越 e 个位来检测。

4.4　快速查找

DFR 算法的查找操作最多 4 次访存就能找到下一跳信息。首先根据 IP 地址的高 16 b 来确定一级索引，当偏移量为 0 时，直接找到对应的端口索引，通过端口索引查找下一跳路由信息。当偏移量 > 0 时，通过一级索引的指针域查找二级索引表，根据目的地址可以确定目的地址在二级索引中对应的 CWA 和 NHI。查找过程如图 6 所示。图中 $V(B_{15}^{0})$和 $V(B_{15+k}^{16})$分别表示目的地址的第 0—15 位的值和第 16—(15 + k)位的值，pSecIndex 表示二级索引的首指针。

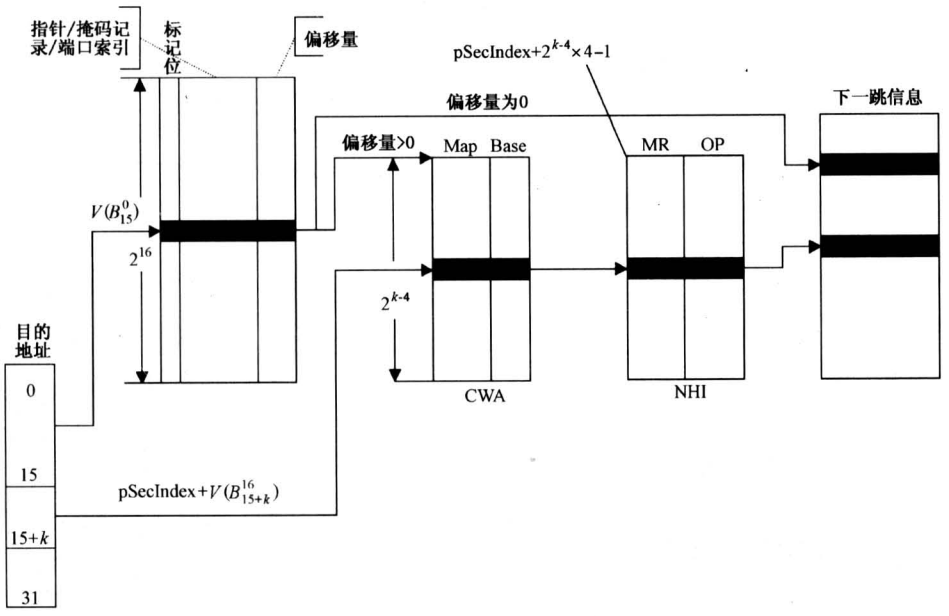

图 6　路由查找的实现过程

Fig.6　Implementation of Route Lookup

5　DFR 算法性能分析

5.1　空间复杂度

设一级索引的偏移量为 k，二级索引中 NHI 的个数为 n，由二级索引表的数据结构可以计算，当 $k > 4$ 时，二级索引的 NHI 的个数是常数 c 的整数倍，即 $n = \lambda c$，λ 为整数，此时二级索引表的存储空间为

$$m_1 = 2^{k-2} + 3\lambda c + 4。\tag{13}$$

当 $K \leqslant 4$ 时，只有一个 2 B 的 Map 和 NHI，因此

二级索引表的存储空间为

$$m_2 = 2 + 3n。 \quad (14)$$

根据因特网中前缀分布的特征，通过以上公式可以计算，构建整个转发表所需要的存储空间的量很小，只需要 800 kB ～ 1 MB 就能存放一个有40 000条路由表项的路由表。

5.2 查找操作的时间复杂度

DFR 算法的查找操作较为简单，进行一次路由查找最多 4 次访存，最少只需要 2 次访存。对 400 MHz 的 CPU，平均 4 次访存约需 100 ns，查找速度可达到 10 Mp/s (packet per second)，如果 2 次访存就能查找成功，那么，查找速度能达到 20 Mp/s。在多任务的操作系统环境下，考虑到代码的执行时间，实际查找速度要慢些。

用 DFR 算法分别对 500，1 500，3 500，…，9 500 条路由的添加情况分别进行了仿真测试。对以上各组数据，分别测试了前缀长度为 8～16 b 和 8～24 b 的情况。表 1 给出了 DFR 算法的仿真结果，并给出了与其他算法的比较。

表 1 算法比较

Table 1 Comparison of algorithms

实现方式	算法类别	访存次数 最少/最多	查找速度 / $Mp \cdot s^{-1}$	内存需求 / MB	实现复杂度	插入、删除和更新难易
软件	SFT	3/9	2	0.15～0.16	较大	不易
	LC	2/6	5	2.3	较大	不易
	分页索引	2/4	5	3	小	不易
	DFR	2/4	4.4～10	0.8～1	小	较易并支持动态操作
硬件	DIR－24－8	2/3	20	33	小	较易
	DIR－21－3	2/4		9	小	不易
	NRL	2/4	100	0.475	较小	不易

6 结论

DFR 算法利用前缀扩展的特点，通过构造特定的数据结构来构建路由表，使得该算法支持动态插入和删除路由表项。DFR 算法的特点是：访存次数少，存储空间小，支持动态插入、删除和更新路由，较好地处理了时间复杂度和空间复杂度的关系。由于以上特点，DFR 算法不仅适合软件实现，也适合硬件实现。

参考文献

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

[2] Henry Hongyi, Tzeng. On fast address-lookup algorithms [J]. IEEE Journal on Selected Areas in Communications, 1999,17(6): 1067～1082

[3] Huang Nenfu, Zhao Shiming. A novel IP-routing lookup scheme and hardware architecture for multigigabit switching routers [J]. IEEE Journal on Seleted Areas in Communications, 1999,17(6): 1093～1104

[4] Gupta P, Lin S, McKeown M. Routing lookups in hardware at memory access speeds [A]. Proc IEEE INFOCOM'98 [C], Session 10B-1, San Francisco, CA, 1998. 1240～1247

Dynamical Fast IP-Routing Lookup Algorithm

Liu Yalin

(*China Academy of Telecommunication Technology, Ministry of Information Industry, Beijing* 100083, *China*)

[**Abstract**] This paper proposes a dynamical fast IP-routing lookup algorithm (DFR). This algorithm uses special data structure to construct index table, and can support inserting, deleting and updating route dynamically. DFR algorithm accesses memory at most four times and at least two times for a route look up. DFR is suitable not only for hardware implementation but also for software implementation.

[**Key words**] prefix expansion; dynamical fast IP-routing lookup algorithm (DFR); route; route lookup