基于改进 CAN 的查找算法

王湛昱[1]，孙名松[2]，邸明星[2]

（1. 哈尔滨工业大学 机电工程学院，哈尔滨 150001，wangzhanyu2005@yahoo.com.cn；2. 哈尔滨理工大学 网络信息中心，哈尔滨 150080）

摘　要： 为了减少 CAN 网络的查询跳数，提高搜索效率，将指针表的概念引入到 CAN 网络中. 在规模为 2^L 的标识符空间上采取折半查找的方法对各维坐标进行划分，并建立相应的下一跳节点集合——指针表，使搜索空间由全网缩减到一个相对较小的指定局部区域. 仿真实验表明，改进后的查找算法所产生的节点坐标相对于原算法有着更为均匀的分布. 在规模为 2^6 和 2^7 的 CAN 网络中，各有 90% 和 70% 的查询跳数减少，平均减少长度为 53.2% 和 31.5%. 扩大实验样本空间后，给出了规模分别为 2^5、2^6 和 2^7 的 CAN 网络的查询长度缩短率分布. 实验证明，改进后的 CAN 算法较原算法有更少的查询跳数.

关键词： CAN 网络；查询；指针表；路由

中图分类号： TP393.02　　**文献标志码：** A　　**文章编号：** 0367－6234(2010)07－1080－06

An improved search algorithm based on CAN

WANG Zhan-yu[1], SUN Ming-song[2], DI Ming-xing[2]

(1. School of Mechatronics Engineering, Harbin Institute of Technology, Harbin 150001, China, wangzhanyu2005@yahoo.com.cn; 2. Network Information Center, Harbin University of Science and Technology, Harbin 150080, China)

Abstract: In order to reduce the hoop count and increase search efficiency, the notion of pointer table is introduced into CAN. In a scale of 2^L identifier space, coordinates in each dimension are divided with binary search. The nodes corresponding to the divided coordinates compose the next hoop set as a pointer table, which cuts down the search scope from the whole CAN to a local CAN area. Simulation results show that the distribution of nodes' coordinates generated by improved CAN search algorithm is more uniform than that of the original CAN model. In the scale of 2^6 CAN and the scale of 2^7 CAN, 90% and 70% searches cut their hoop figures, and the rates of decrease are 53.2% and 31.5% respectively. The distribution rates of search length reduction in the scales of 2^5, 2^5 and 2^7 CANs are given after the sample space is extended. The experiment demonstrates that the improved algorithm based on CAN has less search hoop count than the original one.

Key words: CAN; search; pointer table; routing

CAN(Content-Addressable-Network)网络是由 Ratnasamy[1] 等人于 2001 年提出. 在 CAN 中，关键字和值被映射到数值上相近的节点. 如果 d 维空间中的两个节点的坐标在一个维度上重叠，而在余下 $d-1$ 中都相邻，则称这两个节点互为邻居节点. 在 CAN 中定位对象和路由消息是非常简单的，仅需要关于一个节点的直接邻居节点的信息. CAN 网络中的节点，只保存其直接邻居节点的信息. CAN 网络中的路由信息由 IP 地址、端口号和该节点的每个邻居所拥有的区域范围所组成.

在理想状态下，如果节点的路由信息的哈希函数值都服从均匀分布，那么一个 d 维的标识符空间被分割为大小相等的区域，则每个节点便有了 $2d$ 个邻居. 因此，即便 CAN 网络的规模变得非常庞大，拥有大量的节点，而每个节点所维护的邻居节点数量却依然为常数. CAN 网络可以有效地防止由于网络规模的不断增加，造成节点负载的持续增长，最终导致网络效率降低. 理想化的 CAN 模型，是在 d 维空间中将 CAN 网络划分成均

收稿日期： 2010－01－26.
作者简介： 王湛昱(1982—)，男，硕士研究生.

等的 n 个区间，其平均查找的长度为 $O((d/4)(n^{1/d}))$[2-3]. 然而在实际的应用中，维数 d 的取值往往较小，这使得节点所维护的邻居数量较少，在网络环境相对较好、硬件功能十分强大的情况下，这种使用效率明显是十分低下的. 此外，由于 CAN 网络是“重叠网”，所以与其他的 P2P 模式网络一样面临着拓扑失配[4-6]的问题，即逻辑上相邻的两个节点，地理上可能相距甚远，这使得查询时每一跳的延时变得非常不稳定. 为了进一步充分、合理地使用网络资源并提高 CAN 网络的查询效率，本文对 CAN 网络的查找算法加以改进. 实验证明，改进后的算法相对于原算法有着更小的平均查找跳数，一定程度上提高了 CAN 网络的性能.

1　改进的 CAN 算法

1.1　添加的数据结构

1.1.1　虚拟圆

对于 CAN 网络的 d 维标识符空间上的每一个坐标分别设定一个虚拟圆（本文中的标识符空间为二维空间 v、k，所以虚拟圆为 v 圆和 k 圆）. 并且，如果其应用的哈希函数 $f(a)\rightarrow b$ 能提供 2^L bit 的值域空间，则节点 N 在每个虚拟圆上都要维护 L 个特定的坐标值. 即在 v 圆上维护坐标 $v_1, v_2, \cdots, v_6$；在 k 圆上维护坐标 $k_1, k_2, \cdots, k_6$. 这样，由 d 个虚拟圆上的坐标值按照下脚标相等的原则配对，即 (v_i, k_i)，便可以表示标识符空间上 L 个不同的节点. 这 L 个不同的节点和与当前节点相邻的 $2d$ 个节点，构成了当前节点的指针表.

例：在 v 轴和 k 轴上分别建立虚拟的 v 圆（如图 1 所示）和 k 圆（如图 2 所示）. 假设当前节点 N 在 CAN 坐标为 (v_0, k_0)，哈希函数 $f(a)\rightarrow b$ 能提供 2^L bit 的值域空间，则节点 N 在 v 圆上所维护的坐标的值为 $f(v_0, i) = v_0 + 2^i \bmod 2^L$；在 k 圆上所维护的坐标的值为 $f(k_0, i) = k_0 + 2^i \bmod 2^L, i = 0, 1, \cdots, L-1$. 不妨假设 $L = 6$（则虚拟圆的周长为 64，起始坐标为 0，最大坐标为 63）、$v_0 = 5$、$k_0 = 40$. 那么节点 $N(5, 40)$ 在 v 圆上所维护的 6 个特定坐标分别是 $v_0 + 2^i$（如图 1，表 1 所示）；在 k 圆上所维护的 6 个特定坐标分别是 $k_0 + 2^i$（如图 2，表 2 所示），其中 $i = 0, 1, \cdots, 5$. 各自虚拟圆上的坐标按照下脚标相等的原则配对，即 (v_i, k_i)，便得到点坐标 $a_1(6, 41)$、$a_2(7, 42)$、$a_3(9, 44)$、$a_4(13, 48)$、$a_5(21, 56)$ 和 $a_6(37, 8)$. 节点 $a_1, a_2, \cdots, a_6$ 与当前节点 N 的邻居节点共同构成当前节点 N 的指针表.

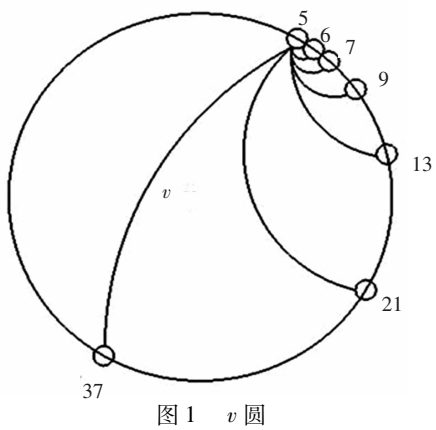


图 1　v 圆

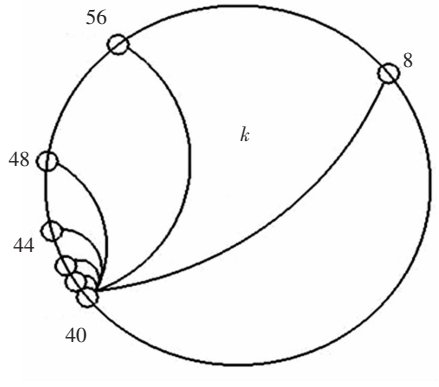


图 2　k 圆

表 1　节点 N 所在的 v 圆维护的坐标

索引	目的坐标
0	5 + 1
1	5 + 2
2	5 + 4
3	5 + 8
4	5 + 16
5	5 + 32

表 2　节点 N 所在的 k 圆维护的坐标

索引	目的坐标
0	40 + 1
1	40 + 2
2	40 + 4
3	40 + 8
4	40 + 16
5	(40 + 32) mod 64 = 8

1.1.2　指针表

如果一个节点 S 所拥有的区域[7]——zone_S 包含 $a_1, a_2, \cdots, a_6$ 中的某个点，则将这个的节点称为当前节点 N 的后继结点，记为 S_i. 当前节点 N 的所有后继结点，组成 N 的后继结点集合 $\text{Successor} = \{S \mid \forall a_i \in \text{zone}_S\}$. 当前节点 N 的邻居节点，记为 N_i，N 的所有邻居节点组成邻居节点集合 Neighbor. 当前节点 N 的下一跳节点集合 Next =

Successor ∪ Neighbor.

如图 3 所示，点 a_1、a_2 落在节点 S_1 上（某些情况下，如节点退出或节点失效，则一个节点可以保存两个节点的信息），a_3 落点节点 S_2 上，a_4 落在节点 S_3 上，a_5 落在节点 S_4 上，a_6 落在节点 S_5 上，则 Successor = $\{S_1, S_2, S_3, S_4, S_5\}$. 不妨令当前节点 N 的坐标为(5,40)，则可以找出 Neighbor = $\{S_2, N_1, N_2, S_3\}$. 所以 Next = $\{S_1, S_2, S_3, S_4, S_5, N_1, N_2\}$，而当前节点 N 需要维护这张 Next 集合的指针表（如表 3 所示）.

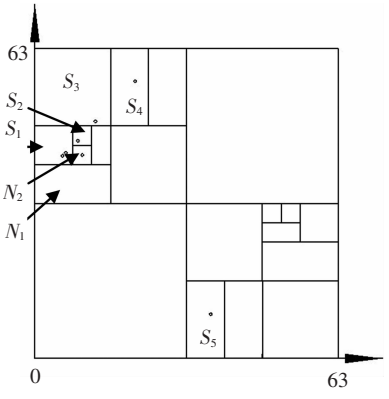


图 3 CAN 平面的节点分

表 3 当前节点 N 的 Next 指针表

Ip	节点
xx. xx. xx. xx	S_1
xx. xx. xx. xx	S_2
xx. xx. xx. xx	S_3
xx. xx. xx. xx	S_4
xx. xx. xx. xx	S_5
xx. xx. xx. xx	N_1
xx. xx. xx. xx	N_2

1.2 节点的加入

如果节点 M 加入到这个标识符空间（如图 4 所示），且假设 M 所带资源的 (v,k) 对应的值为 (10,41). 在改进算法中，节点的加入和原始 CAN 算法的节点加入方式[8] 基本相同. M(10,41) 在已存在的 B 点的区域(7 ~ 11,39 ~ 47) 内，所以 M 要与 B 平分这一区域（如图 5 所示），并将这一更新告知 B 与 M 的邻居结合. 这个更新的消息并不需要立即发送给那些指针表中含有 B 点的节点，来更新它们的指针表.

假设节点 N 的下一跳结合为 Next = $\{\cdots, B, \cdots\}$，即 B 点在节点 N 的指针表中. 如果由 N 发起的一个查询的目的坐标在新加入的节点 M 的区域 (7 ~ 11,39 ~ 43) 内，那么按照上述查询方法，节点 N 在其指针表中找到节点 B 作为应答节点，从而在 B 点处对目的坐标发起 CAN 模式的查询. 此时，节点 B 所拥有的区域已经发生了变化，则节点 B 会将更新后的情况附加到查询消息中，待查询返回给节点 N 时，将这个更新告知节点 N.

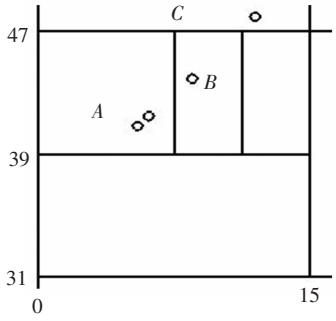


图 4 节点 M 加入前

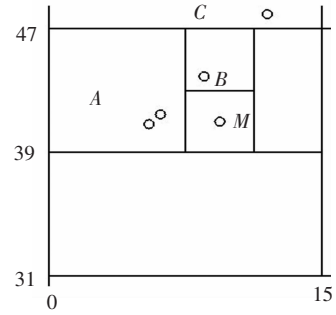


图 5 节点 M 加入后

1.3 节点的退出

某个节点退出时，要由它的一个邻居节点来接管其所拥有的区域. 如果将要退出的节点的某个邻居所拥有的区域能和这个将要退出的节点所拥有的区域组成一个矩形，那么将由这个邻居节点来接管这个区域；如果不能找到这样的一个邻居节点，那么由将要退出的节点的面积最小的邻居节点来接收这个区域. 无论是哪种接收方式，将要退出的节点和接受节点都要将这一变化告知它们的邻居节点.

如果一个查询发起节点的指针表中的节点退出，当这个查询发起节点不能在一定时间内与这个退出的节点取得通信，那么便对这个节点所在的区域发起查找，则查找返回的结果将是接收这个区域的节点. 查询发起节点，根据这个返回结果更新自己的指针表.

2 查询方式

2.1 查询方式

设当前节点为 $N(v,k)$，被查询的目的坐标为 (v_d, k_d)，步骤为：

① 检查目的坐标 (v_d, k_d) 是否在当前节点所拥有的区域内. 如果是，则转到 ④；否则转到 ②；

② 在下一跳节点集合中分别找到点 $N_i(v + 2^i, k + 2^i)$ 满足 $v + 2^i \leqslant v_d < v + 2^{i+1}$ 和点 $N_j(v +$

$2^j, k+2^j)$，满足 $k+2^i \leqslant k_d < +k2^{i+1}$. 在 N_i 和 N_j 中选择距离目的节点较近的作为“下一跳节点”，转到 ③；

③ 查询方式判断：在“下一跳节点”上估计经过若干跳后是否会有比按照原始 CAN 查询方式具有更短跳数的节点. 若是，将下一跳节点置为当前节点，跳数计数器 $j++$，转到 ①，否则转到 ④；

④ 发起原始 CAN 查询，转到 ⑤；

⑤ 返回查询结果.

根据上述描述，可以看出这是一个递归过程：

```
Search (N,(v_d,k_d))
{if ((v_d,k_d) 不在 N 所拥有的区域内)
{N_next = compare(v_d,k_d)
if(满足条件③)
{N = N_next
Search(N,(v_d,k_d))
}
CAN Search(N,(v_d,k_d))
}
Return(search result)// 返回查询结果
}
```

其中，compare(v_d,k_d) 返回当前节点 N 的 Next 集合中满足条件 ② 的节点；CAN Search$(N,(v_d,k_d))$ 按照原始 CAN 方式向坐标(v_d,k_d) 发起查询，如图 6 所示.

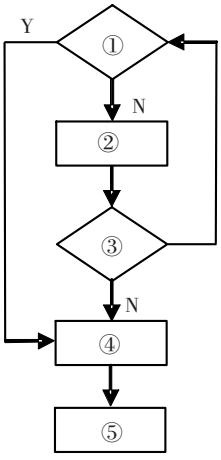


图 6　查询流程图

2.2　查询方式判断

根据文献[9 - 11]，简单地通过使用哈希函数可以得到一个均匀分布的值空间. 因此，假定任一节点 $N(v,k)$ 在其每个维度上的坐标值服从均匀分布，即 $v \sim U(0,2^L-1)$，$k \sim U(0,2^L-1)$. 假设系统中共有 n 个节点，则一个节点在坐标轴 v 和 k 上所占有的距离为 $2^L/n$，即按照原始的 CAN 查询方式，每跳所跨越的距离为 $2^L/n$. 任一节点 $N(v,k)$ 与目的坐标(v_d,k_d) 在 v 轴上估计的跳数为 $|v-v_d|/(2^L/n)$；在 k 轴上估计的跳数为 $|k-k_d|/(2^L/n)$，则在当前节点 $N(v,k)$ 上按照原始的 CAN 查询方式的估计跳数为 $j_{\text{CAN}}=|v-v_d|/(2^L/n)+|k-k_d|/(2^L/n)$.

那么任一节点 $N(v,k)$ 的下一跳节点为 $N_{\text{next}}(v_n,k_n)$. 则在 $N_{\text{next}}(v_n,k_n)$ 上，经过 m 跳之后的节点坐标为 $N_m(v_m,k_m)$，则

$$\begin{cases} v_m = v_n + 2^{I1} + 2^{I2} + \cdots + 2^{\text{Im}}, \\ k_m = k_n + 2^{I1} + 2^{I2} + \cdots + 2^{\text{Im}}. \end{cases}$$

$$I_1,\cdots,I_m \in [0,L-1].$$

令

$$\begin{cases} \{|v_d - [(v_n + 2^{I1} + 2^{I2} + \cdots + 2^{Im}) \bmod 2^L]|\}/(2^L/n) = V, \\ \{|k_d - [(k_n + 2^{I1} + 2^{I2} + \cdots + 2^{Im}) \bmod 2^L]|\}/(2^L/n) = K. \end{cases}$$

$$I_1,\cdots,I_m \in [0,L-1].$$

显然，$V=|v_d-v_n|$，$K=|k_d-k_n|$，而 $(K+V)/(2^L/n)$ 为在 $N_m(v_m,k_m)$ 上对目的坐标发起原始 CAN 查询所需的跳数的估计.

如果存在一组数列$\{I_1,I_2,\cdots,I_m\}$ 满足 $(K+V)/(2^L/n) < j_{\text{CAN}}-m-1$，则在 $N_{\text{next}}(v_n,k_n)$ 上经过 m 跳之后存在一个节点，它到目的坐标的跳数要小于从点 $N_{\text{next}}(v_n,k_n)$ 上直接对目的坐标(v_d,k_d) 发起原始 CAN 查询的跳数，即 N_{next} 是可行的下一跳节点，将查询转发给 N_{next}. 接下来，N_{next} 节点再作为当前节点，重复上面的查询过程.

为了求解是否存在数列$\{I_1,I_2,\cdots,I_m\}$（只找出是否存在即可，不用求出数列），建立一棵查询树. 设当前的节点 $N_{0,1}(v,k)$ 为树的根，即查询发起节点. 对于任意节点 $N_{i,j}$（i 为层数，j 为所在层的第 j 个节点），而其下一跳集合 Next 中的节点作为节点 $N_{i,j}$ 的孩子结点（如图 7 所示）. 依此类推，查询数列的过程也就是查询以当前节点 $N_{0,1}(v,k)$ 为根，高度为 m 的树中，是否能在第 m 层上找到一个节点，最终满足条件 $(K+V)/(2^L/n) < j_{\text{CAN}}-m-1$ 的过程，其中，$m=1,2,\cdots,j_{\text{CAN}}$. 对于这个过程，用广度优先遍历树的算法来实现.

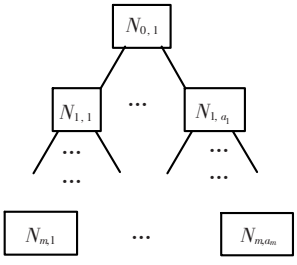


图 7　查询树示意图

```
isjump = 0//初始化
Q = Enqueue (N) //树的根节点入队列
while (j_CAN > 1 | !isEmpty(Q))// 计数器 > 1 且队列非空
{N = Dequeue(Q)// 取队首进行判断
if (get_length(N,N_d) < j_CAN)// 当前节点是否满足
{return (isjump = 1) (K + V)/(2^L/n) < j_CAN - m - 1
break
}
else
{ Nextset = get_ Nextset (N) //不满足,则找到 N 的孩子结点,即下一跳节点集合
Q = Enqueue( Nextset)// N 的孩子结点入队列
}
if (N 是左孩子)// 当前节点 N 是最左叶子,则说明开始了新的一层,也
{j_CAN -- 即新的一跳,计数器 - 1
layer + +//层数 + 1
if ( layer = limit)//为了控制算法的规模,层数限制在 limit 层以内
{return ( isjump = 0)}
}
}
```

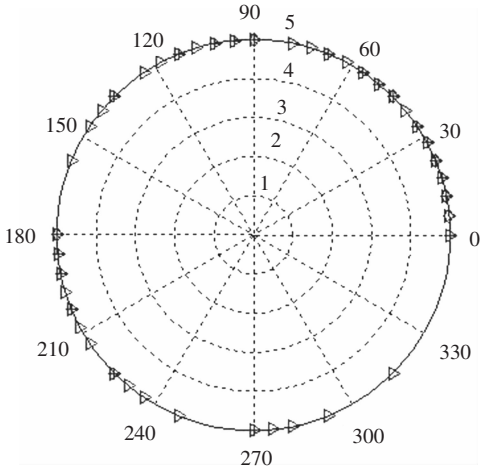
3 仿 真

3.1 仿真说明

在假定节点服从均匀分布的前提下,分别对查询方式判断中的参数 L、layer 取不同的值,得到不同的网络规模和不同的算法执行规模. 在此基础上,分别使用原有 CAN 算法和改进的 CAN 算法计算从随机生成源节点 $N(v,k)$ 到随机生成目的节点 $N_d(v_d,k_d)$ 的路由跳数.

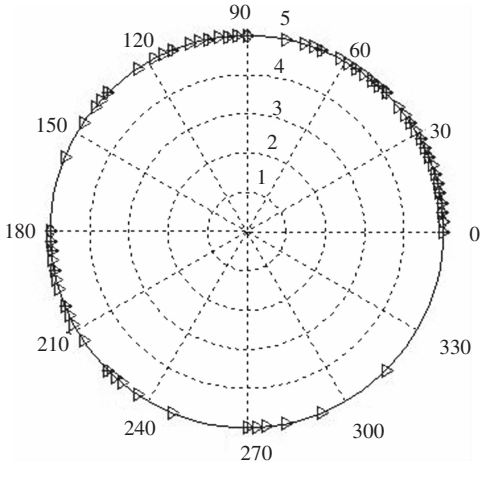
3.2 仿真结果

图 8 显示了当 $k = 0, L = 6$, layer = 3 时的坐标分布. 此时每个坐标轴的长度为 2^6, CAN 网络中可容纳的节点数为 2^{12}; 图 9 显示了当 $k = 0$, $L = 7$, layer = 3 时的坐标分布, 此时每个坐标轴长度为 2^7, CAN 网络中可容纳节点数为 2^{14}. 以 k 为源节点坐标, 则在虚拟的 k 圆上, layer 跳内的坐标数为 $\sum_{i=1}^{\text{layer}} L^i$. 显然, $L = 6, L = 7$ 时, 改进的 CAN 网络在一个坐标上 3 跳内的 $\sum_{i=1}^{3} L^i$ 个坐标有着相对于原始 CAN 网络无论 L 取何值时(即与网络规模无关), 在一个坐标上 3 跳内只有 $\sum_{i=1}^{3} 2^i$ 个坐标更加广泛、均匀的分布.



◇—第 1 跳的坐标 +—第 2 跳的坐标 △—第 3 跳的坐标

图 8 $k = 0$, $L = 6$, layer = 3 时坐标在虚拟圆上的分布



◇—第 1 跳的坐标 +—第 2 跳的坐标 △—第 3 跳的坐标

图 9 $k = 0$, $L = 7$, layer = 3 时坐标在虚拟圆上的分布

表 4, 表 5 表示随机选取的 10 个源节点 (v,k) 和目的节点 (v_d,k_d), n 为根据查询方式判断的结果是否应用了改进算法; $N\_$ len 为改进算法的查找跳数; $O\_$ len 为原 CAN 算法查找跳数.

表 4 $L = 6$, layer = 5

k	v	k_d	v_d	n	$N_$len	$O_$len
26	27	42	54	1	15	43
24	28	39	37	1	8	24
46	33	50	32	0	5	5
12	45	63	52	1	49	58
46	32	8	43	1	17	49
24	9	37	53	1	36	57
44	64	62	4	1	15	78
24	36	17	39	1	12	10
4	37	45	62	1	19	66
36	19	55	22	1	17	22

表 5　$L=7$，layer = 5

k	v	k_d	v_d	n	N_len	O_len
29	25	49	51	1	9	46
12	32	29	42	1	8	27
46	49	18	44	0	33	33
42	11	8	32	1	55	55
62	22	38	15	0	31	31
49	17	33	45	1	44	44
58	62	36	9	1	34	75
10	17	54	17	0	44	44
53	16	60	23	1	3	14
13	17	40	31	1	14	41

当 $L=6$ 时，有 90% 的查询应用了改进算法，$\sum N\_\text{len}=193$，$\sum O\_\text{len}=412$，算法改进后比原算法平均缩短查询路径长度为 $(\sum O\_\text{len}-\sum N\_\text{len})/\sum O\_\text{len}=53.2\%$.

当 $L=7$ 时，有 70% 的查询应用了改进算法，$\sum N\_\text{len}=281$，$\sum O\_\text{len}=410$，算法改进后比原算法平均缩短查询路径长度为 $(\sum O\_\text{len}-\sum N\_\text{len})/\sum O\_\text{len}=31.5\%$.

为了获得更可靠的统计数据，实验扩大了样本空间. 分别在 $L=5,6,7$ 和 layer = 4,5 的条件下进行 100 个从随机源节点到随机目的节点的查询，结果如图 10 所示.

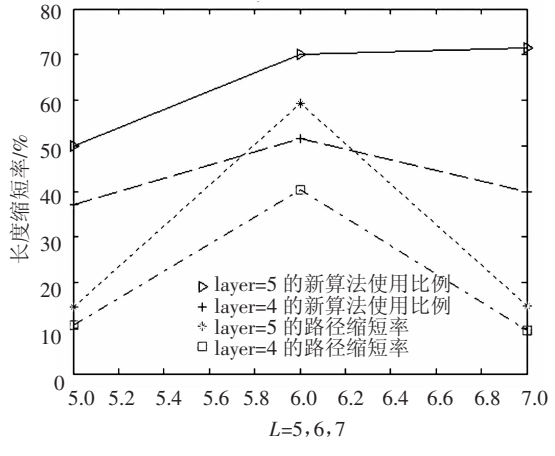


图 10　查询长度缩短率

layer = 5 时，改进算法在被使用的比例和缩短查询路径方面有着较高的效率. 而改进算法的性能根据 L 和 layer 的不同有着较大幅度的波动，但是相对原 CAN 算法依然具有一定的提高.

4　结　论

1）为了减少 CAN 网络的查询跳数，提高搜索效率，改进后的算法对标识符空间内的每一维坐标进行了划分，使得下一跳坐标在每一维上有更均匀的分布. 由这些坐标所构成的节点作为下一跳节点，分别对应 CAN 平面上的一个特定区域，从而将查询范围由全网缩小到一个由下一跳节点负责的指定区域，有效地从逻辑上减少了查询跳数.

2）仿真结果表明，改进后的算法在一定概率下可以减少查询跳数，在没有大幅度提高节点负载的前提下，改善了原 CAN 算法的查询效率.

参考文献：

[1] RATNASAMY S, FRANCIS P, HANDLEY M, *et al.* A scalable content-addressable network[C]//Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. New York: ACM, 2001: 161 – 172.

[2] 严蔚敏，吴伟民. 数据结构(C 语言版)[M]. 北京：清华大学出版社，1996.

[3] 杨静，王亚民. 基于查询扩展和节点聚合的 P2P 搜索方法[J]. 现代图书情报技术，2009(9)：51 – 56.

[4] 邱彤庆，陈贵海. 一种令 P2P 覆盖网络拓扑相关的通用方法[J]. 软件学报，2007，18(2)：382 – 390.

[5] RATNASAMY S, HANDLEY M, KARP R, *et al.* Topologically-aware overlay construction and server selection [C]//Proceedings of the INFOCOM 2002 Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. New York: IEEE, 2002: 1190 – 1199.

[6] 张爱萍. P2P 网络技术综述[J]. 科技信息，2008(15)：65.

[7] STEINMETZ R, WEHRLE K. P2P 系统及其应用[M]. 王玲芳，陈炎，译. 北京：机械工业出版社，2008.

[8] RATNASAMY S, HANDLEY M, KARP R, *et al.* A Scalable Content Addressable Network[D]. Berkeley: University of California, 2002.

[9] BYERS J, CONSIDINE J, MITZENMACHER M. Simple load balancing for DHTs[C]//Proceedings of 2nd International Workshop on Peer-to-Peer Systems(IPTPS03). Heidelberg: Springer-Verlag, 2003: 42 – 45.

[10] Rao A, LAKSHMINARAYANAN K, SURANA S, *et al.* Load balance in structed P2P systems[C]//Proceedings of 2nd International Workshop on Peer-to-Peer Systems (IPTPS `03). Heidelberg: Springer-Verlag, 2003: 68 – 79.

[11] RIECHE S, PETRAK L, WEHRLE K. A Thermal Dissipation-based approach for balancing data load in distributed hash tables[C]//Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks. Washington: IEEE Computer Society, 2004: 15 – 23.

（**编辑**　张　红）