

# A New Variation of Chord with Novel Improvement on Lookup Locality

**Jie Wang**

Department of Computer Science  
University of Massachusetts Lowell  
wang@cs.uml.edu

**Zhijun Yu**

Department of Computer Science  
University of Massachusetts Lowell  
zyu@cs.uml.edu

**Abstract-** *Distributed Hash Tables (DHT) are mechanisms used to locate data in P2P and Grid computing. Early DHT schemes, while providing lookups with optimal or near optimal efficiency at the overlay level, tend to neglect efficiency at the physical level. Although one may extend a DHT scheme by simply adding additional nodes in routing tables to provide extra locality choices, we note that there are intrinsically better ways. We illustrate this point using Chord as an example. We generalize Chord from one-sided lookups to two-sided lookups in a new variation called B-Chord. We show that B-Chord achieves substantially better lookup locality than Chord and 4-Extended Chord, a known variant that has approximately the same node degree of B-Chord. The improvement is achieved using a convex combination of finding a shorter physical path and finding a shorter overlay path. Simulating these protocols on common network models, we show that B-Chord, on average, incurs less than 35% and 25% of physical hops than Chord and 4-Extended Chord, respectively.*

**Keywords:** Distributed Hash Tables, locality, routing, data lookup algorithms, performance, simulations

## 1 Introduction

Data lookup is a critical issue in P2P and Grid computing. It locates a node responsible for a given data (key). A number of distributed hash table (DHT) schemes have been proposed in recent years with an objective of providing efficient, scalable, reliable, and fault resilient lookup service. Representatives of these include Chord [13], CAN [11], Pastry [12], Tapestry [15], Koorde [6], Symphony [9], and Viceroy [8]. Let  $N$  denote the total number of nodes in the overlay network. Table I summarizes the number of overlay routing hops of these DHT schemes.

Most distributed lookup protocols achieve lookup efficiency at the overlay level rather than at the physical level (Pastry and Tapestry are two notable exceptions). By lookup locality we mean the minimum physical cost trav-

This research is supported in part by NSF under grant CCF-04080261, by NSF of China under grant 60273062, and by UMass Lowell under a CFCI grant 04-05.

TABLE I

DHT schemes	Node degree	Overlay routing hops
Chord	$O(\log N)$	$O(\log N)$
CAN	$O(d)$	$O(dN^{1/d})$
Pastry	$(b-1)(\log_b N) + O(b)$	$O(\log_b N)$
Tapestry	$b(\log_b N)$	$O(\log_b N)$
Koorde	2	$O(\log N)$
Symphony	$2k+2$	$O(\log^2 N/k)$
Viceroy	3 to 7	$O(\log N)$

eling along an overlay lookup path, which is an important performance measure. We use the following definition of lookup locality.

**Definition:** The physical cost of two adjacent nodes in the overlay network is the number of physical hops on the shortest physical path connecting these two nodes. The *lookup locality* of an overlay path  $p$  is the summation of physical cost of each pair of adjacent overlay nodes on  $p$ .

Providing multiple overlay paths is a standard approach to improving lookup locality, so that one has choices to select a path with better locality (i.e., with less routing latency). This approach also makes the lookup protocol more robust and more fault resilient. One way of doing so is to extend the existing DHT scheme by adding additional nodes in routing tables with a minor modification of the scheme. For example, one may extend Chord by replacing each successor node in a Chord finger table with a list of immediate successor nodes. We note that there are better ways by a novel modification of the existing scheme. We illustrate this idea using Chord as an example.

We devise a new distributed lookup protocol using additional routing information to turn one-sided lookups in Chord to two-sided lookups. We call the new protocol B-Chord, standing for Bilinear Chord. Lookups in B-Chord use two greedy strategies: Obtain a shorter overlay path and obtain a shorter physical path. These two strategies, however, may not agree with each other. To achieve the best performance, B-Chord uses a convex combination of the physical distance from the current node to the next node and the overlay distance from the next node to the key.

We show that B-Chord offers substantial improvements on lookup locality over Chord and 4-Extended Chord, a

known variant of Chord that has approximately the same node degree as B-Chord. Moreover, B-Chord preserves asymptotically the same performance as Chord on node joining and node leaving. Simulating Chord and B-Chord on graphs generated from GT-ITM internetwork topology models of flat random graphs, 2-level transit-stub hierarchies, and N-level hierarchies [4], [14], we show that lookups in B-Chord reduce on average over 35% of physical hops than Chord and over 25% than 4-Extended Chord.

## 2 Chord and simple variations

Chord [13] assigns each participating node a unique Chord ID. It does so by hashing the node's IP address to a binary string of fixed length  $m$ . It also hashes each key to an  $m$ -bit binary string. A ring of  $2^m$  points labeled by  $m$ -bit binary string starting from 0 to  $2^m - 1$  clockwise forms an  $m$ -bit base ring (or simply a base ring when there is no confusion). We call an  $m$ -bit string an *identifier*, a *Chord ID*, or a *base point*. Chord nodes form a sub-ring of the base ring. We will not distinguish a node (or a key) from its Chord ID.

We fix the usage of  $m$  and  $N$  throughout this paper, where  $m$  denotes the length of identifiers and  $N$  a positive number with  $N < 2^m$ . We borrow the notations of intervals on the real line to denote intervals on the base ring in the clockwise direction. For example, let  $a$  and  $b$  be two base points. Then  $(a, b]$  is the interval that includes all the base points from  $a$  to  $b$ , excluding  $a$  and including  $b$ , in the clockwise direction. This use of interval notations would not cause confusion: When values in intervals are base points, they mean intervals on the base ring.

Let  $d(a, b)$  denote the number of base points in  $(a, b]$ . Let  $k$  be an identifier. Denote by  $\text{succ}(k)$  the successor of  $k$ , which is the first node from  $k$  on the base ring in the clockwise direction. Each Chord node maintains a routing table (called a finger table) of  $m$  entries for routing keys, where the  $i$ th entry in the finger table at node  $n$  is a pair  $(C_n[i], \text{succ}(C_n[i]))$  and

$$C_n[i] = (n + 2^{i-1}) \bmod 2^m, \quad 1 \leq i \leq m.$$

Let  $\text{succ}_n[i]$  denote  $\text{succ}(C_n[i])$ , called the  $i$ th finger of  $n$ .

Also included in each entry of the finger table is the location information (e.g., the IP address) of node  $\text{succ}_n[i]$  for routing keys. This information will not be explicitly displayed for simplicity.

The lookup of a key  $k$  at node  $n$  in Chord follows a simple procedure: Check whether  $k \in (n, \text{succ}_n[1]]$ , if so, return  $\text{succ}_n[1]$ . Otherwise, find an integer  $i$  such that  $\text{succ}_n[i] \in (n, k)$  and  $\text{succ}_n[i]$  is closer to  $k$  than any other fingers in  $(n, k)$ . That is, for all  $j \neq i$  with  $\text{succ}_n[j] \in (n, k)$ :  $d(\text{succ}_n[i], k) < d(\text{succ}_n[j], k)$ . Forward a query to node  $n' = \text{succ}_n[i]$  to locate  $k$ . Node  $n'$  repeats the same procedure, forwards a query to the next node if necessary, and to the next, and so on, until  $k$  is located.

In an  $N$ -node Chord network, a lookup can be carried

out with  $O(\log N)$  messages [13]. That is, a query can be routed to the destination node through  $O(\log N)$  nodes. Moreover, node joining and node leaving can each be carried out with  $O(\log^2 N)$  messages.

One may extend Chord by replacing each successor node in a Chord finger table with a fixed-size list of immediate successor nodes [13] to improve locality. The lookup algorithm in Extended Chord is essentially the same as in Chord where the largest node from the list of immediate successors is selected. The simulations presented in [13] show that this approach can improve lookup locality.

We consider Extended Chord with successor-list-size 4, called 4-Extended Chord, for the reason that the node degree in 4-Extended Chord is approximately the same as that in B-Chord.

One may also try to improve Chord's lookup locality by applying Chord twice independently and select a locally better next node using a greedy strategy. This means to hash independently each node twice to have two names  $n_a$  and  $n_b$ , and each key twice to two names  $k_a$  and  $k_b$ . We call it 2-Chord. The key  $k_a$  is stored in node  $\text{succ}(k_a)$  and the key  $k_b$  is stored in node  $\text{succ}(k_b)$ . So every key is stored in two different nodes. For each node in the routing path, there are two successor nodes produced by the Chord lookup algorithm on two different inputs of the key's two names  $k_a$  and  $k_b$ . The node with a smaller number of physical hops from the current node will be chosen.

## 3 B-Chord

B-Chord is defined on the  $m$ -bit base ring, where each node and each key is hashed to an  $m$ -bit identifier as in Chord. Keys are stored in nodes in the same way as in Chord.

### 3.1 Finger tables

In addition to using the successor of  $C_n[i]$  as in Chord, B-Chord also uses the predecessor of  $C_n[i]$ , denoted by  $\text{pred}(C_n[i])$ , which is the first node counterclockwise from  $C_n[i]$  on the base ring. Each node in an  $N$ -node B-Chord network maintains a finger table of  $2m - 1$  entries. Let  $P = \langle n_0, \dots, n_\ell \rangle$  be an overlay path and  $\text{ph}(n_i, n_j)$  the number of physical hops on the shortest path from node  $n_i$  to node  $n_j$  in the underlying physical network. The lookup locality of  $P$  is  $\sum_{i=0}^{\ell-1} \text{ph}(n_i, n_{i+1})$ .

The finger table at node  $n$  contains  $2m - 1$  entries in the form of  $(C_n[i], (\text{pred}_n[i], p_i), (\text{succ}_n[i], s_i))$ , where

$$C_n[i] = \begin{cases} (n + 2^{i-1}) \bmod 2^m, & \text{if } 1 \leq i \leq m, \\ (n - 2^{2m-i-1}) \bmod 2^m, & \text{if } m+1 \leq i \leq 2m-1, \end{cases}$$

$\text{pred}_n[i] = \text{pred}(C_n[i])$ ,  $\text{succ}_n[i] = \text{succ}(C_n[i])$ ,  $p_i = \text{ph}(n, \text{pred}_n[i])$ , and  $s_i = \text{ph}(n, \text{succ}_n[i])$ . The values of  $p_i$  and  $s_i$  are found using a standard network tool (e.g., `traceroute`) and included in the finger table when the table is created.

**Remark:** We only store the values of  $p_i$  and  $s_i$ , not the entire path, to save memory space. This information helps reduce the running overheads of the lookup algorithm.

We call  $pred_n[i]$  and  $succ_n[i]$ , respectively, the  $i$ th left finger and the  $i$ th right finger of  $n$ .

**Example 1:** Let  $m = 7$  and the node set be  $\{5, 14, 25, 36, 45, 54, 65, 74, 83, 92, 102, 113, 123\}$ . Table II shows the finger table of node 123 and the finger table of node 36.

TABLE II

(A) FINGER TABLE OF NODE 123. (B) FINGER TABLE OF NODE 36.

$C_{123}$	$(pred, p)$	$(succ, s)$
$(123 + 2^0) \bmod 2^7 = 124$	(123, 0)	(5, 13)
$(123 + 2^1) \bmod 2^7 = 125$	(123, 0)	(5, 13)
$(123 + 2^2) \bmod 2^7 = 127$	(123, 0)	(5, 13)
$(123 + 2^3) \bmod 2^7 = 3$	(123, 0)	(5, 13)
$(123 + 2^4) \bmod 2^7 = 11$	(5, 13)	(14, 31)
$(123 + 2^5) \bmod 2^7 = 27$	(25, 31)	(36, 7)
$(123 + 2^6) \bmod 2^7 = 59$	(54, 17)	(65, 5)
$(123 - 2^5) \bmod 2^7 = 91$	(83, 18)	(92, 9)
$(123 - 2^4) \bmod 2^7 = 107$	(102, 9)	(113, 19)
$(123 - 2^3) \bmod 2^7 = 115$	(113, 19)	(123, 0)
$(123 - 2^2) \bmod 2^7 = 119$	(113, 19)	(123, 0)
$(123 - 2^1) \bmod 2^7 = 121$	(113, 19)	(123, 0)
$(123 - 2^0) \bmod 2^7 = 122$	(113, 19)	(123, 0)

(A)

$C_{36}$	$(pred, p)$	$(succ, s)$
$(36 + 2^0) \bmod 2^7 = 37$	(36, 0)	(45, 21)
$(36 + 2^1) \bmod 2^7 = 38$	(36, 0)	(45, 21)
$(36 + 2^2) \bmod 2^7 = 40$	(36, 0)	(45, 21)
$(36 + 2^3) \bmod 2^7 = 44$	(36, 0)	(45, 21)
$(36 + 2^4) \bmod 2^7 = 52$	(45, 21)	(54, 14)
$(36 + 2^5) \bmod 2^7 = 68$	(65, 11)	(74, 15)
$(36 + 2^6) \bmod 2^7 = 100$	(92, 23)	(102, 25)
$(36 - 2^5) \bmod 2^7 = 4$	(123, 19)	(5, 13)
$(36 - 2^4) \bmod 2^7 = 20$	(14, 26)	(25, 7)
$(36 - 2^3) \bmod 2^7 = 28$	(25, 18)	(36, 0)
$(36 - 2^2) \bmod 2^7 = 32$	(25, 18)	(36, 0)
$(36 - 2^1) \bmod 2^7 = 34$	(25, 18)	(36, 0)
$(36 - 2^0) \bmod 2^7 = 35$	(25, 18)	(36, 0)

(B)

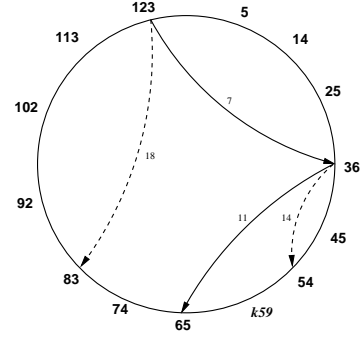


Fig. 1. Lookup for key 59

We note that these two strategies may not agree with each other. That is, we could have  $ph(n, n^-) < ph(n, n^+)$  but  $d(k, n^-) > d(n^+, k)$ , or vice versa. In a lookup process sometimes Strategy 1 is better and sometimes Strategy 2 is better. To take advantage of both strategies we use a linear combination of the two strategies to define the cost from node  $n$  to node  $n'$  such that the physical cost has weight  $a$  and the overlay cost has weight  $b$ , where  $a, b \geq 0$  and  $a + b > 0$ . Since the cost function is used for comparison, it is equivalent to using the following convex combination of the two strategies with a single weight parameter  $\sigma = a/(a + b)$ . That is, for any  $\sigma \in [0, 1]$ , let

$$c(n, n') = \sigma \cdot ph(n, n') + \begin{cases} (1 - \sigma)d(k, n'), & \text{if } n' = n^-, \\ (1 - \sigma)d(n', k), & \text{if } n' = n^+. \end{cases}$$

Empirical results (see Section 4) show that when  $\sigma = \frac{5}{9} \pm \epsilon$  for a small  $\epsilon \geq 0$ , the lookup algorithm yields the best result.

As an example let us assume in Example 1 that node 123 is instructed to locate key 59, which is stored in  $succ(59) = 65$ . Let  $\sigma = 1$ . From Table II(A) we see that nodes 36 and 83 are the two best candidates, for  $36 \in (123, 59)$  is closest to 59 among all nodes  $succ_n[i] \in (123, 59)$ , and  $83 \in (59, 123)$  is closest to 59 among all nodes  $pred_n[i] \in (59, 123)$ . Node 36 is selected because  $ph(123, 36) = 7 < 18 = ph(123, 83)$ , and it is in the clockwise direction. From Table II(B) we see that node 54 from the column of right fingers and node 65 from the column of left fingers are the best candidates, and node 65 is selected that is in the counterclockwise direction.

This lookup procedure, however, may not converge. This is because an alternating clockwise lookup and counterclockwise lookup may form a lookup loop. To solve this problem we introduce two variables  $D^-$  (counterclockwise) and  $D^+$  (clockwise) to keep track of the overlay difference between the contacted node and the key to ensure that the next contacted node is closer to the destination. Note that if  $n$  is a clockwise node, then  $d(n^+, k) < D^+$ ; if  $n$  is a counterclockwise node, then  $d(k, n^-) < D^-$ . Thus, if  $d(n^+, k) \geq D^+$ , then moving to node  $n^+$  in the clockwise direction will be farther away from  $k$  than moving to node  $n^-$ . Likewise, if  $d(k, n^-) \geq D^-$ , then moving to node  $n^-$  will be farther away from  $k$  than moving to node  $n^+$ .  $D^-$

## 3.2 Lookup

For a given key  $k$  at node  $n$ , the lookup procedure proceeds as follows: Find  $i_0$  such that  $pred_n[i_0] \in (k, n)$  and it is closer to  $k$  than any other nodes  $pred_n[j] \in (k, n)$ . This step is referred to as *counterclockwise lookup*. Let  $n^- = pred_n[i_0]$ . Find  $i_1$  such that  $succ_n[i_1] \in (n, k)$  and it is closer to  $k$  than any other nodes  $succ_n[j] \in (n, k)$ . This step is referred to as *clockwise lookup*. Let  $n^+ = succ_n[i_1]$ . Select a node from these two nodes with a smaller cost and forward a lookup query to that node with the key  $k$ . Repeat this process until the key is located. The cost from node  $n$  to node  $n'$  may be the number of physical hops from  $n$  to  $n'$ ,  $ph(n, n')$ , stored in the finger table, or the distance on the base ring between  $n'$  and the key  $k$ . This gives rise to the following two greedy strategies:

- 1) Select the next node with a smaller physical cost.
- 2) Select the next node with a smaller overlay cost.

and  $D^+$  are passed to the next node and get updated along the way.

Described below is the final lookup algorithm  $\text{LOOKUP}_\sigma()$  with a weight parameter  $\sigma$ .

// For a given key  $k$  find the destination node  $\text{succ}(k)$ .  
// Initially, set  $D^- \leftarrow d(k, n)$  and  $D^+ \leftarrow d(n, k)$ .

$n.\text{LOOKUP}_\sigma(k)$

1. **if**  $k \in (n, \text{succ}_n[1])$  **return**  $\text{succ}_n[1]$ ; **terminate**;  
// destination found
2. **if**  $k \in [\text{pred}_n[1], n)$  **return**  $n$ ; **terminate**;  
// destination found
3. **set**  $n^- \leftarrow \text{pred}_n[i_0]$ , where  $\text{pred}_n[i_0] \in (k, n)$  and  $d(k, n^-) = \min\{d(k, \text{pred}_n[i]) \mid \text{pred}_n[i] \in (k, n)\}$ ;
4. **set**  $n^+ \leftarrow \text{succ}_n[i_1]$ , where  $\text{succ}_n[i_1] \in (n, k)$  and  $d(n^+, k) = \min\{d(\text{succ}_n[i], k) \mid \text{succ}_n[i] \in (n, k)\}$ ;
5. **if**  $d(k, n^-) \geq D^-$  **set**  $n' \leftarrow n^+$ ;  
// Choose  $n^+$
6. **else if**  $d(n^+, k) \geq D^+$  **set**  $n' \leftarrow n^-$ ;  
// Choose  $n^-$
7. **else if**  $c(n, n^-) \leq c(n, n^+)$  **set**  $n' \leftarrow n^-$ ;
8. **else set**  $n' \leftarrow n^+$ ;  
// Choose the local optimal node
9. **set**  $D^- \leftarrow d(k, n^-)$  and  $D^+ \leftarrow d(n^+, k)$ ;  
**return**  $n'.$  $\text{LOOKUP}_\sigma(k)$ ;

### 3.3 Correctness proof

The parameter  $\sigma$  does not affect the correctness of the lookup algorithm regardless its value, and so in this section we will not specifically mention this parameter. Let  $n_i$  denotes the  $i$ th node ( $i \geq 0$ ) in the lookup path with  $n_0 = n$ . In  $n_i.\text{LOOKUP}(k)$ , we say that a node is being *checked* if it is either  $n_i^-$  or  $n_i^+$  in lines 3 and 4, and a node is *contacted* if it is where the query is forwarded. No communication is needed for a node being checked.

**Lemma 1:** In  $n_i.\text{LOOKUP}(k)$  we have the following:

- 1)  $d(\text{succ}(k), n_i^-) \leq d(\text{succ}(k), n_i)/2$ .
- 2)  $d(n_i^+, \text{pred}(k)) \leq d(n_i, \text{pred}(k))/2$ .

*Proof:* We will prove the first inequality. The proof of the second inequality is similar. Without loss of generality, we use  $n$  to denote  $n_i$  and  $n^-$  to denote  $n_i^-$ . It follows from the lookup algorithm that  $n^- \in (k, n)$  and

$$\forall \text{pred}_n[i] \in (k, n) : d(k, n^-) \leq d(k, \text{pred}_n[i]). \quad (1)$$

Since  $n^- \in (k, n)$  we have  $n^- \in [\text{succ}(k), n)$ . Starting from point  $n$ , the finger points  $C_n[1], \dots, C_n[2m-1]$  lie one by one in the clockwise direction. These points (including  $n$ ) partition the base ring into  $2m$  segments. For convenience, let  $C_n[2m] = n$ . Since  $n^- \neq n$  and  $\text{pred}_n[1] = n$ , there must be an integer  $i \in [1, 2m-1]$  such that  $n^- \in (C_n[i], C_n[i+1])$ . This implies that  $\text{succ}(k)$  must also be in this interval. To see this suppose that  $\text{succ}(k)$  is not in this interval. Then  $\text{succ}(k) \in (C_n[j], C_n[j+1])$  for some  $j$ . If  $j > i$  then  $n^- \notin [\text{succ}(k), n)$ , a contradiction. So we must have  $j < i$ . This

means that  $\text{pred}_n[j+1] \in (k, C_n[j+1])$  and  $d(k, \text{pred}_n[j]) \leq d(k, C_n[j+1]) < d(k, n^-)$ , which violates Inequality 1.

*Case 1:*  $1 \leq i < m$ . Note that

$$\begin{aligned} d(n, \text{succ}(k)) &\leq d(n, C_n[i+1]) \\ &= 2^i \leq 2^{m-1}, \\ d(\text{succ}(k), n) &= 2^m - d(n, \text{succ}(k)) \\ &\geq 2^m - 2^{m-1} = 2^{m-1}, \\ d(C_n[i], C_n[i+1]) &= 2^i - 2^{i-1} \leq 2^{m-2}. \end{aligned}$$

Thus,

$$d(\text{succ}(k), n^-) \leq d(C_n[i], C_n[i+1]) \leq d(\text{succ}(k), n)/2.$$

*Case 2:*  $i = m$ . Note that

$$\begin{aligned} C_n[m+1] &= (n - 2^{m-2}) \bmod 2^m \\ &= (n + 2^m - 2^{m-2}) \bmod 2^m \\ &= (n + 3 \cdot 2^{m-2}) \bmod 2^m. \\ d(C_n[m], C_n[m+1]) &= (3 \cdot 2^{m-2} - 2^{m-1}) \bmod 2^m \\ &= 2^{m-2}, \\ d(\text{succ}(k), n) &\geq d(C_n[m+1], n) \\ &= (C_n[m+1] - n) \bmod 2^m \\ &= 3 \cdot 2^{m-2}. \end{aligned}$$

Thus,

$$\begin{aligned} d(\text{succ}(k), n^-) &\leq d(C_n[m], C_n[m+1]) \\ &\leq d(\text{succ}(k), n)/3. \end{aligned}$$

*Case 3:*  $m < i \leq 2m-1$ . We first note that  $i$  cannot be equal to  $2m-1$ , for  $n^-$  and  $\text{succ}(k)$  cannot be in the interval  $(C_n[2m-1], C_n[2m]) = ((n-1) \bmod 2^m, n]$  that only contains node  $n$ . Thus, we must have  $i < 2m-1$ . We have

$$\begin{aligned} d(\text{succ}(k), n) &= (n - \text{succ}(k)) \bmod 2^m \\ &\geq (n - C_n[i]) \bmod 2^m \\ &= n - (n - 2^{2m-i-1}) \\ &= 2^{2m-i-1}, \\ d(C_n[i], C_n[i+1]) &= [(n - 2^{2m-(i+1)-1}) - \\ &\quad (n - 2^{2m-i-1})] \bmod 2^m \\ &= 2^{2m-i-2}. \end{aligned}$$

Thus,

$$\begin{aligned} d(\text{succ}(k), n^-) &\leq d(C_n[i], C_n[i+1]) \\ &= 2^{2m-i-2} \leq d(\text{succ}(k), n)/2. \quad \blacksquare \end{aligned}$$

Next we assume that both clockwise and counterclockwise lookups occur in  $\text{LOOKUP}(k)$ . If the  $i$ th node on the lookup path is a clockwise node, then we denote it by  $a_i$ ; otherwise by  $b_i$ .

**Lemma 2:** Let  $P$  be the lookup path of  $\text{LOOKUP}(k)$ .

- 1) If  $P$  contains a sequence  $b_i \rightarrow a_{i+1} \rightarrow b_{i+2}$  for some integer  $i \geq 0$ , then  $d(\text{succ}(k), b_{i+2}) < d(\text{succ}(k), b_i)/2$ .
- 2) If  $P$  contains a sequence  $a_i \rightarrow b_{i+1} \rightarrow a_{i+2}$  for some integer  $i \geq 0$ , then  $d(a_{i+2}, \text{pred}(k)) < d(a_i, \text{pred}(k))/2$ .

*Proof:* We prove Statement 1. The proof of Statement 2 is similar and is omitted here.

Since node  $b_{i+2}$  is in different direction from node  $a_{i+1}$ , it means that  $b_{i+2} = a_{i+1}^-$  and the condition in line 5 in  $a_{i+1}.$ LOOKUP( $k$ ) is false; namely,  $d(k, a_{i+1}^-) < a_{i+1}.D^-$ , where  $a_{i+1}.D^-$  represents the value of  $D^-$  at node  $a_{i+1}$ . It follows from  $a_{i+1}.D^- = d(k, b_i^-)$  that

$$d(k, b_{i+2}) < d(k, b_i^-).$$

By Lemma 1(1) we know that

$$d(\text{succ}(k), b_i^-) \leq d(\text{succ}(k), b_i)/2.$$

Thus,

$$\begin{aligned} d(\text{succ}(k), b_{i+2}) &= d(k, b_{i+2}) - d(k, \text{succ}(k)) \\ &< d(k, b_i^-) - d(k, \text{succ}(k)) \\ &= d(\text{succ}(k), b_i^-) \\ &\leq d(\text{succ}(k), b_i)/2. \quad \blacksquare \end{aligned}$$

**Theorem 3:** Assume that in an  $N$ -node B-Chord network, nodes are uniformly distributed on the base ring. Then for a given key  $k$  at any node, LOOKUP( $k$ ) locates  $k$  by contacting only  $O(\log N)$  nodes.

*Proof:* For convenience, we call  $\text{succ}(k)$  and  $\text{pred}(k)$  a *target node*. It follows from Lemma 1 that for any one step  $n_1 \rightarrow n_2$ , node  $n_2$  is at least two times closer than node  $n_1$  to node  $n_1$ 's target node. Since the distance from the starting node to the destination node is at most  $2^m$ , if directions do not change, LOOKUP( $k$ ) takes at most  $m + 1$  steps to reach the destination node  $\text{succ}(k)$  (the extra one step is needed for the clockwise step). Suppose that LOOKUP( $k$ ) changes its lookup direction. Assume that LOOKUP( $k$ ) alternates its direction at each step for two steps, with  $n_1 \rightarrow n_2 \rightarrow n_3$ , where  $n_2$  is in different direction from  $n_1$  and  $n_3$  is in different direction from  $n_2$ . By Lemma 2 we know that  $n_3$  is more than two times closer than  $n_1$  to  $n_1$ 's target node. Thus, if LOOKUP( $k$ ) alternates its direction at each node, it will take at most  $2m + 1$  steps to reach the destination node. Other situations (i.e., there are a few steps in the same direction, a few steps in different directions, and a few steps in the same direction again, and so on) will take between  $m + 1$  and  $2m + 1$  steps to reach the destination. Since the  $N$  nodes are uniformly distributed on the base ring, the expected number of nodes contacted in the lookup path is  $O(\log N)$ .  $\blacksquare$

Under a good hash function, nodes are uniformly distributed with high probability. Thus, we have the following corollary.

**Corollary 4:** In an  $N$ -node B-Chord network, with high probability the number of nodes to be contacted by LOOKUP( $k$ ) for locating  $k$  is  $O(\log N)$ .

### 3.4 Lookup analysis

Suppose that the same node  $n$  in both B-Chord and Chord networks of the same number of nodes needs to

locate the destination node  $\text{succ}(k)$  for a given key  $k$ . Let  $l_C(n, k)$  and  $l_B(n, k)$  denote, respectively, the number of nodes on the overlay path from node  $n$  to node  $\text{succ}(k)$  in the Chord and B-Chord networks. Let  $l_C^p(n, k)$  and  $l_B^p(n, k)$  denote, respectively, the lookup locality of these two overlay paths.

We assume that nodes are uniformly distributed at random on the base ring and the underlying physical network is a graph selected uniformly at random from  $N$ -node connected graphs. Then on average a shorter overlay path is expected to result in a shorter physical path. For each random variable  $X$  in the set of  $l_B(n, k)$ ,  $l_C(n, k)$ ,  $l_B^p(n, k)$ , and  $l_C^p(n, k)$ , we use  $E_{n,k}[X]$  to denote the expected value of  $X$ .

We claim that  $E_{n,k}[l_B(n, k)] < E_{n,k}[l_C(n, k)]$  and  $E_{n,k}[l_B^p(n, k)] < E_{n,k}[l_C^p(n, k)]$ . To see this we inspect each execution of  $n_i.$ LOOKUP( $k$ ), where  $n_i$  is the  $i$ th ( $i \geq 0$ ) node on the lookup path and  $n_0 = n$ . Let  $HC^+$  denote the half circle of the base ring from node  $n_0$  clockwise and  $HC^-$  the other half.

Assume  $i = 0$ . We note that  $d(n_0, n_0^-) < D^-$  and  $d(n_0, n_0^+) < D^+$ . Suppose  $\text{pred}(k)$  is on  $HC^-$ , then  $n_1$  is closer to  $k$  than the node selected by Chord because B-Chord has fingers in  $HC^-$  and Chord does not. Suppose  $\text{pred}(k)$  is on  $HC^+$ . If  $n_1 = n_0^+$ , then  $n_1$  will also be selected by Chord. If  $n_1 = n_0^-$ , this means  $c(n, n_0^-) < c(n, n_0^+)$ . Since Chord will select  $n_0^+$ , and so B-Chord's selection  $n_0^-$  is better than Chord's selection. Even if  $n_1$  is randomly selected from  $n_0^-$  and  $n_0^+$ , B-Chord still has two out of four chances to select a better node, one out of four chances to select a node that is as good as Chord, and one out of 4 chances to select a node that may or may not be better than Chord. Thus, on average we expect that B-Chord selects a better node than Chord does.

Now assume  $i > 1$ . Suppose  $n_i$  is selected by line 5 or line 6. Then  $n_i$  is at least twice closer to the target node than  $n_{i-1}$  by Lemma 1, regardless whether  $n_i$  is clockwise or counterclockwise. Thus, on average this selection is at least as good as Chord, and when  $\text{pred}(k)$  is on  $HC^-$ , this selection is better. If  $n_i$  is selected by line 7 or line 8, this case is similar to the case of  $i = 0$ .

### 3.5 Node joining and node leaving

Node joining and node leaving in B-Chord follow the same procedures in Chord. When a node joins the network, B-Chord first locates an arbitrary node  $n_a$  on the ring and asks it to look for the successor and the predecessor of the new node. After this is establish the network is functional, although it is not yet at its full capacity. The construction of the finger table can be carried out in the background. A partial finger table, as long as the node knows its immediate predecessor and successor, can provide lookup service. Since the successor and the predecessor of the neighbors of the new node have changed, the corresponding finger tables of the nodes must be updated. The existing node recursively updates the fingers of other nodes. When a node leaves the network, a

similar procedure is followed. To update the successor and the predecessor of each node after a node is added in or removed from the network, we could employ a lazy finger update mechanism that periodically verifies the immediate successor and predecessor of each node, and refreshes its finger table entries.

The communication cost of creating and maintaining a finger table in B-Chord is higher than Chord, for additional nodes need to be contacted and their information need to be maintained. The number of these additional nodes is less than three times of the number of nodes maintained in a finger table of Chord, and so is in the order of  $O(\log N)$ . This is a small price to pay for the improved lookup locality, and the work can be done in the background without interrupting the lookup service. If nodes stay in the network for a while (for instance, from a few minutes to a few hours), the improved lookup locality would be worth the effort.

In summary, when a node joins or leaves the B-Chord network, only  $O(\log N)$  nodes are affected. The finger table of each affected node must be updated. The update for each node incurs  $O(\log N)$  messages, and so the overall update cost is  $O(\log^2 N)$  messages.

## 4 Simulation results

We simulate Chord, B-Chord, 4-Extended Chord, and 2-Chord over the Georgia Tech Internetwork Topology models (GT-ITM). They are flat random graphs (Rand), 2-level transit-stub hierarchies (T-S), and N-level hierarchies (N-level). For detailed descriptions of these models please see [14]. We use the software package GT-ITM [4] to generate physical networks. Our code is written in JAVA.

A Rand graph is generated by connecting each pair of nodes with a chosen edge probability.

In all the Rand graphs we generated, the degree of each node is between 2 and 8.

A T-S graph is generated based on the following parameters: the random seed  $S$ , the number of transit domains  $T$ , the average number of stub domains per transit node  $K$ , the average number of nodes per transit domain  $N_t$ , the average number of nodes per stub domain  $N_s$ , the number of edges between a stub domain and a transit domain  $E_{ts}$ , the number of edges between a stub domain to a stub domain  $E_{ss}$ , the edge probability in transit domains  $P_t$ , and the edge probability in stub domains  $P_s$ .

An N-level graph can be determined by the following parameters: the number of levels  $L$ , the number of nodes  $N_i$  in each internal subgraph at level  $i \in [1, L]$ , the flat random graph model  $M$ , and the edge probability parameters  $\alpha_i$  and  $\beta_i$  that determine the connectivity in each subgraph at level  $i$ .

We carry out our simulations under  $m = 15$  with 32,768 identifier spaces. It allows us to simulate networks with density  $\rho$  (i.e.,  $\rho$  is the ratio of the number of nodes in a network and the size of the underlying ID space) ranged from 0.03 to 0.50. We carried out our simulations on a

TABLE III

PARAMETERS FOR THE T-S MODEL

	$S$	$T$	$K$	$N_t$	$N_s$	$E_{ts}$	$E_{ss}$	$P_t$	$P_s$	size
$G1$	82	8	7	12	23	3	2	0.3	0.2	15,540
$G2$	47	6	9	10	29	2	1	0.32	0.21	15,750
$G3$	19	9	8	12	18	3	0	0.29	0.3	15,648
$G4$	77	7	6	12	31	1	1	0.33	0.17	15,696
$G5$	51	10	7	14	16	4	2	0.21	0.34	15,778

TABLE IV

COMPARISON OF B-CHORD OVER CHORD, AND B-CHORD OVER 4-EXTENDED CHORD

	Rand	T-S	N-level
$h_B^p/h_C^p$	65.01%	63.94%	64.80%
$h_B^p/h_E^p$	79.2%	70.9%	68.7%

sun4u SunFire V240 server. (Simulating denser networks over 16,000 nodes on this machine was arduous.)

In our simulations we choose the parameters (details are given in Table III) to generate T-S graphs. The parameters so chosen guarantee that the degree of each node is between 2 and 7.

Our N-level graphs are generated with the following parameters:  $L \in [3, 5]$ ,  $N_i \in [4, 23]$ ,  $\alpha_i \in [0.3, 0.7]$ , and  $\beta_i \in [0.2, 0.5]$ . The total number of nodes in each graph is between 15,312 and 15,525, and the degree of each node in each graph is between 2 and 7.

For each graph generated we run the following procedure 300 times: Randomly choose  $N$  nodes to form a Chord, a B-Chord, a 4-Extended Chord and a 2-Chord networks, where  $N = 1000j$  ( $j = 1, \dots, 15$ ). For each  $N$ -node overlay network we randomly and independently choose 300 node-key pairs  $(n, k)$  and execute the corresponding lookup algorithm to locate node  $succ(k)$  starting at node  $n$ , with  $\sigma = \frac{5}{9}$ . We calculate the average physical hops for each lookup protocol at each value of  $N$  (see Figure 2). We see that on average 2-Chord has no advantage over Chord.

Averaging physical hops over all lookup paths in our simulations under all three topologies, we obtain comparison data shown in Table IV, where  $h_C^p$ ,  $h_E^p$ , and  $h_B^p$  denote, respectively, the average number of physical hops in Chord, 4-Extended Chord, and B-Chord. From the table we see that the average lookup locality in B-Chord is 35% to 36% better than that in Chord, and 20% to 31% (i.e., more than 25% on average over the three models) better than that in 4-Extended Chord.

We have chosen the weight parameter  $\sigma = \frac{5}{9}$  in our simulations. This indicates that, according to the convex combination of the cost formula  $c(n, n')$  (see 3.2) used in the lookup algorithm, about  $\frac{4}{9}$  of the improvement comes from selecting a shorter overlay path and about  $\frac{5}{9}$  of the improvement comes from selecting the next node with a stronger locality. To further explore how  $\sigma$  affects performance, we have carried our experiments with  $\sigma$  ranged from 0 to 1. Figure 3 shows this effect on a T-S graph. The improvement percentages are calculated based on average physical hops in lookups with different values

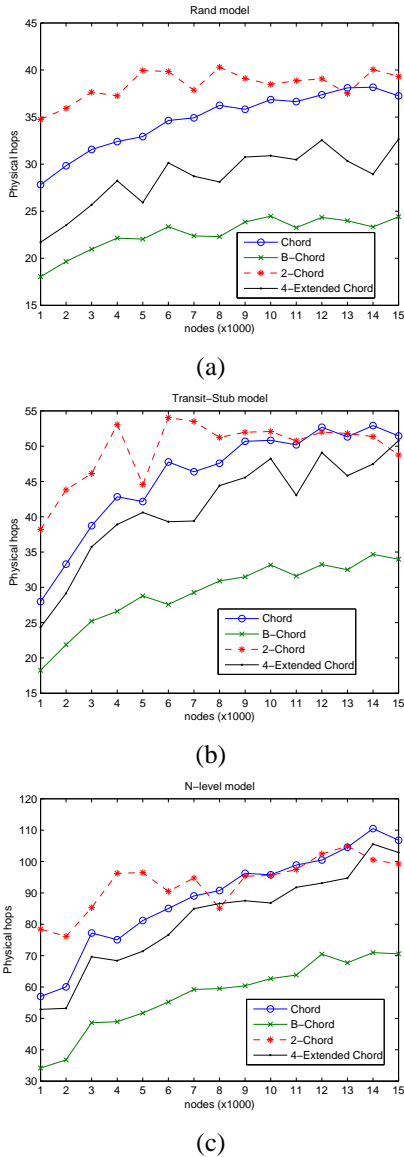


Fig. 2. Comparison of physical hops: (a) over Rand graphs; (b) over T-S graphs; and (c) over N-level graphs.

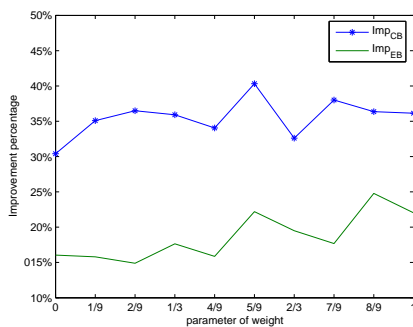


Fig. 3. Lookup comparisons for different values of the weight parameter  $\sigma$ . The improvement variable  $IMP_{CB} = (h_C - h_B)/h_C$  and  $IMP_{EB} = (h_E - h_B)/h_E$ , where  $h_C$ ,  $h_E$  and  $h_B$  denote, respectively, the average number of physical hops in Chord, 4-Extended Chord and B-Chord.

of  $\sigma$  over 5000-node overlay networks. The next node selection, when  $\sigma$  is set to 1 depends only on the physical cost; and when  $\sigma$  is set to 0 depends only on the overlay cost. It indicates that when  $\sigma = \frac{5}{9} \pm \epsilon$  with small  $\epsilon \geq 0$ , the lookup algorithm is optimal and the physical locality plays the most effective role. The physical improvement of B-Chord over Chord can be as much as 41%.

## Acknowledgement

We are grateful to Dr. Benyuan Liu for pointers of network models and Dr. David Martin for suggesting 2-Chord for comparison.

## References

- [1] S. E. Ansary, L. O. Alima, P. Brand, and S. Haridi. A framework for peer-to-peer lookup services based on  $k$ -ary search. *Swedish Institute of Computer Science technical report T2002-06*, 2002.
- [2] N. de Bruijn. A combinatorial problem. In *Proc. Koninklijke Nederlandse Akademie van Wetenschappen*, 49:758-764, 1946.
- [3] P. Ganesan, K. Gummadi, and H. Garcia-Molina. Canon in G major: designing DHTs with hierarchical structure. In *Proc. of the 24th International Conference On Distributed Computing Systems (ICDCS'04)*, Tokyo, 2004, pp. 263-272.
- [4] GT-ITM: Georgia Tech Internetwork Topology Models. URL: <http://www.cc.gatech.edu/projects/gtitm/>.
- [5] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A Scalable overlay network with practical locality properties. In *Proc. of the 4th USENIX Symposium on Internet Technologies and Systems (USITS'03)*, Seattle, 2003, pp. 113-126.
- [6] M.F. Kaashoek and R. Karger. Koorde: a simple degree-optimal distributed hash table. In *Proc. of the Second International Workshop on P2P Systems*, 2003.
- [7] J. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proc. 32nd ACM Symposium on Theory of Computing*, pp. 163-170, 2000.
- [8] D. Malkhi, M. Naor and D. Ratajczak. Viceroy: a Scalable and Dynamic Emulation of the Butterfly. In *Proc. of Principles of Distributed Computing*, 2002.
- [9] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed Hashing in a SmallWorld. In *Proc. of the 4th USENIX Symposium on Internet Technologies and Systems*, 2003.
- [10] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. of the 9th Annual Symposium on Parallel Algorithms and Architectures*, ACM Press, 1997, pp. 311-320.
- [11] S. Rantansamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, San Diego, 2001, pp. 161-172.
- [12] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proceedings of International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, 2001, pp. 329-350.
- [13] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17-32, 2003. (First appeared in *Proc. of ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2001, San Diego, 2001, pp. 149-160.)
- [14] E. W. Zegura, K. L. Calvert, and M. J. Donahoo. A quantitative comparison of graph-based models for Internet topology. *IEEE/ACM Transactions on Networking*, 5(6):770-783, 1997.
- [15] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. *Technical Report UCB/CSD-01-1141*, UC Berkeley, April 2001.