Scalable and Robust Structured Peer-to-Peer Based on Balanced Kautz Tree

Deke Guo, *Member, IEEE,* Yunhao Liu, *Senior Member, IEEE,* Honghui Chen, and Xueshan Luo

***Abstract*— In order to improve scalability and reduce maintenance overhead for structured Peer-to-Peer (P2P) systems, researchers design optimal architectures with constant degree and logarithmical diameter. The expected topologies, however, require the number of peers to be some given values determined by the average degree and the diameter. Hence, existing designs fail to address the issue due to the fact that 1) we cannot guarantee how many peers to join a P2P system at a given time, and 2) a P2P system is typically dynamic with peers frequently coming and leaving. In this work, we propose BAKE scheme based on balanced Kautz tree structure with $log_d n$ diameter and constant degree even the number of peers is an arbitrary value. By keeping a total ordering of peers and employing a robust locality-preserved resource placement strategy, resources that are similar in single or multi-dimensional attributes space are stored on a same peer or neighboring peers. Through analysis and simulations, we show that BAKE achieves optimal diameter and good connectivity as the Kautz digraph does (almost achieves the Moore bound), and supports both exact and range queries efficiently. Indeed, the concepts of balanced Kautz tree and Kautz ring introduced in this work can also be extended and applied to other interconnection networks after minimal modifications, for example, de Bruijn digraph.**

***Index Terms*— Kautz Tree, Structured Peer-to-Peer, Distributed Hash Table, Moore Bound, Optimal Digraph**

I. Introduction

Structured Peer-to-Peer (P2P) models have been proposed as a good candidate infrastructure for building large-scale and robust network applications [1], [2], [3]. They impose a certain structure on the overlay network and control the placement of data, thus exhibit several unique properties that unstructured systems lack.

In the design of such networks, the most common concern is the limitation on peer out-degree and network diameter. The out-degree of a peer denotes number of overlay connections attached to it and the size of routing table to be maintained. The diameter indicates the largest number of hops that must be traversed in order to transmit a message between any two peers in the worst case. Traditionally, the peer out-degree and network diameter increase logarithmically with respect to the size N of a network, such as Chord [2], Pastry [3], Tapestry [4], HyperCup [5] and Kademlia [6], which are based on the hypercube topology, and SkipNet, based on skip list structure [7]. Those schemes publish and lookup resources within $O(logN)$ hops. They, however, introduce huge maintenance overhead and suffer from poor scalability.

D. Guo and Y. Liu are with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. E-mail: guodk@cse.ust.hk, liu@cse.ust.hk.

H. Chen and X. Luo are with the School of Information System and Management, National University of Defense Technology, Hu Nan, China.

The preliminary result is included in proceedings of IEEE INFOCOM 2008.

To address this issue, researchers propose architectures based on interconnection networks. For examples, the Viceroy [8] and Ulysses [9] are based on the butterfly topology [8], Cycloid [10] is based on the CCC topology [11], CAN [1] is based on the d-dimensional torus topology, Koorde [12], Distance Halving [13], D2B [14], [15], ODRI [16] and Broose [17] are based on the de Bruijn topology, and FissionE [18], [19] is based on the Kautz topology. In above designs, the network diameter increases logarithmically with respect to the size of the P2P system, while the out-degree of each peer is a constant, setting a better tradeoff between routing table size and routing delay. The critical requirement of those designs is that the number of peers must be some given values determined by the peer degree and the network diameter. Hence, the approaches are often impractical in real implementations, especially when considering peers are frequently coming and leaving [20].

In this study, we aim at designing a novel P2P architecture with smaller diameter and constant peer degree even the number of peers is an arbitrary value. Thus, the scheme is practically easy to implement, without being restricted by peer dynamics. After looking into literatures, we observe that Kautz digraph is the best choice among existing no-trivial digraphs since it almost achieves Moore Bound [21] and has optimal diameter.

We design a *balanced Kautz tree* and associated *Kautz ring* structures [22]. We then propose BAKE: a robust and efficient P2P network based on the balanced Kautz tree with k_l=$\lceil \log_d(n) - \log_d(1+1/d) \rceil$ network diameter and $d+2$ out-degree of each peer in a dynamic environment. The average routing distance is shorter than that of CAN, butterfly, de Bruijn digraph, and $log_d n$, and close to that of complete Kautz digraph. In a static or moderately dynamic environment, the delay and message cost to join a new peer is $2k_l + \alpha + 1$ where α=$\lceil n/(d^{k_l-1} + d^{k_l-2} \rceil$ and $\alpha \leq d$, the delay and message cost to handle a failed peer is $2k_l + \alpha$, and the delay and message cost to publish or lookup a resource is k_l. In a highly dynamic environment, the delay to join a new peer is $3k_l + \alpha + 2$ hops and the message cost is $3 \times (k_l + \alpha)$. The delay and message cost to handle a failed peer is $3k_l + \alpha$. In any environment, the delay and message cost to handle a leaving peer x is bounded by $2k_l + \alpha + 2$.

The main contributions of this paper are as follows:

1) We propose a balanced Kautz tree structure. We also introduce an associated Kautz ring structure to keep a total ordering of peers, and design the clockwise and anti-clockwise distance functions among peers in the Kautz ring.
2) Based on the balanced Kautz tree structure, we propose BAKE: an effective and robust P2P architecture which retains desirable properties of static Kautz digraph, such as optimal diameter and constant out-degree. We design a locality-preserving resource placement strategy, an effective and robust routing scheme, and exact and range query schemes. We also introduce necessary algorithms to deal

with dynamic operations of peers.

3) We evaluate the topology properties of BAKE, the robustness of routing scheme, and the delay and message cost of basic operations through formal analysis and comprehensive simulations, as well as compare BAKE with previous P2P designs based on different constant degree interconnection networks.

The rest of this paper is organized as follows. Section II presents the balanced Kautz tree and associated Kautz ring. Section III discusses the design of BAKE based on balanced Kautz tree. Section IV presents the dynamic operations to maintain the topology. We evaluate BAKE in Section V, and conclude the work in Section VI.

II. Kautz Tree Structure

A. Preliminary

The topology of a structured P2P network is usually modeled by a graph or digraph in which vertices stand for nodes while edges represent overlay connections. Many efforts have been made to address the degree/diameter problem which determines the largest graphs or digraphs of given maximum degree and given diameter. The order n_{dk} of a digraph with maximum out-degree d and diameter k is not larger than a general Moore bound [21], [23] as follows:

$$n_{dk} \leq d^k + d^{k-1} + ... + d^2 + d + 1 = (d^{k+1} - 1)/(d - 1). \quad (1)$$

Many research activities related to the degree/diameter problem have proved that non-existence of digraphs achieve the general upper bound for the parameters $d \geq 3$ and $k \geq 3$ [24]. The best lower bound on the order of digraphs of maximum out-degree d and diameter k is as follows. For maximum out-degree d=2 and diameter $k \geq 4$, $n_{dk} \geq 25 \times 2^{k-4}$. For the remaining values of maximum d and diameter k, a general lower bound is $n_{dk} \geq d^k + d^{k-1}$ [21]. Among existing non-trivial digraphs, this best lower bound is only obtained by Kautz digraph defined as follows.

Definition 1: A digraph with fixed vertex out-degree d and diameter k is a Kautz digraph, denoted as $K(d, k)$, only if it holds the following two conditions [25], [26], [27], [28], [29].

1) Exists $d^k + d^{k-1}$ nodes labeled with string $x_1x_2...x_k$ over $\{0, 1, ..., d\}$, where $x_i \neq x_{i+1}$.
2) Exists an arc from a node $x_1x_2...x_k$ to another node $x_2, ...x_k\alpha$ for each $\alpha \in \{0, 1, ..., d\} - \{x_k\}$. The arc is labeled as $(x_1x_2...x_k, x_2, ...x_k\alpha)$ or $x_1x_2, ...x_k\alpha$.

Besides the degree/diameter problem, a structured P2P network also focuses on another order/degree problem which determines the smallest diameter in a digraph of order n_{dk} and maximum out-degree d. Based on the Moore bound of the degree/diameter problem, a lower bound of the order/degree problem can be derived as $k \geq \lceil \log_d (n_{dk}(d - 1) - 1) \rceil - 1$. In practice, all existing digraphs cannot achieve this lower bound for the parameters $d \geq 3$ and $k \geq 3$ [24]. The best upper bound on the diameter of digraphs of maximum out-degree d and order n_{dk} is $\lceil \log_d \frac{n_{dk}}{d+1} + 1 \rceil$. Among all existing non-trivial digraphs, this best upper bound is only obtained by Kautz digraph defined above.

The most related research work revolves around FISSIONE, which uses a Kautz digraph $K(2, k)$ as its static topology and introduces emulation methods to deal with dynamic operations of nodes. It, however, cannot support Kautz digraphs with arbitrary degree except degree 2, and suffers from poor lookup performance and weak connectivity since the degree of each peer is too small. Furthermore, the emulation methods of $K(2, k)$ are not suitable to a general Kautz graph $K(d, k)$ where $d > 2$.

One of our early attempts is called Moore [29], in which an incomplete Kautz digraph is proposed as a structured P2P topology. The incomplete Kautz digraph retains all advantages of Kautz digraph where $d \geq 2$, and overcomes a disadvantage of Kautz digraph that the order must be some given values determined by the peer out-degree and the network diameter, such as $d + 1, d(d + 1), d^2(d + 1), ..., d^k(d + 1)$. Moore attains the best upper bound of the order/degree problem mentioned above even the order is an arbitrary value.

However, Moore can only work under relative static or moderately dynamic environment, and suffers huge overhead due to maintaining its topology and resource placement strategy in large scale and highly dynamic environments. Recently, a structured P2P scheme based on a distributed linear digraph employs a similar concept and suffers from the same problems [30]. Later discussions will show that BAKE proposed in this paper has a small overhead and more robust routing scheme than Moore in dynamic environments.

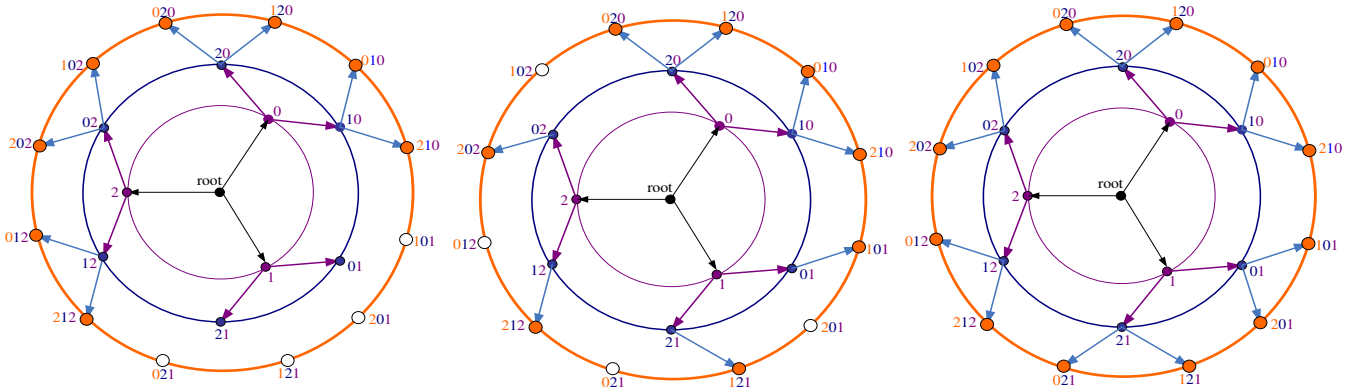
B. The definition of Kautz tree

The kuatz digraph is the optimal topology among non-trivial digraphs only if all nodes exist and are stable, but is impractical in dynamic scenarios. To address this issue, we propose the balanced Kautz tree structure to achieve the desired topology. The Kautz ring keeps an order of nodes to support locality-preserving resource placement in Section III.

Definition 2: A d-ary Kautz tree with depth k is a rooted tree. The root node has $d + 1$ child nodes, and each inner node has at most d children. Each edge at same level is assigned a unique label. Each node except the root node is given a unique label. The label of a node is the concatenation of the labels along the edges on its root path. The label of each edge is assigned based on the following rules.

1) The edge from the root node to its ith child is labeled as x_1^i=$i - 1$ for $1 \leq i \leq d + 1$. The ith child of root node is labeled as x_1^i, and is arranged from left to right.
2) The edge from a node x_1 to its ith child is labeled as x_2^i=$(x_1 - i) \bmod (d + 1)$ for $1 \leq i \leq d$. The ith child is labeled as $x_2^i x_1$, and is arranged from left to right.
3) The edge from a node $x_{k-1}...x_2x_1$ to its first child is labeled as x_k^1 where x_k^1=x_1 if $x_1 \neq x_{k-1}$, otherwise x_k^1=$(x_1 - 1) \bmod (d + 1)$. The first child is labeled as $x_k^1 x_{k-1}...x_2x_1$, and is arranged at the leftmost position.
4) We arrange the values $0, 1, ..., d$ along a ring in ascend order. If a path from point x_k^1 to point x_k^i=$(x_k^1 - i + 1) \bmod (d+1)$ along the anti-clockwise direction does not meet x_{k-1}, the edge from $x_{k-1}...x_2x_1$ to its ith child is labeled as x_k^i=$(x_k^1 - i + 1) \bmod (d + 1)$, otherwise it is labeled as x_k^i=$(x_k^1 - i) \bmod (d+1)$. The ith child of node $x_{k-1}...x_2x_1$ is labeled as $x_k^i x_{k-1}...x_2x_1$ for $2 \leq i \leq d$, and is arranged from left to right.

Note that the identifier $x_kx_{k-1}...x_2x_1$ of each node in a Kautz tree satisfies that $x_{i+1} \neq x_i$ for $1 \leq i \leq k - 1$. We associate each node in the tree with an unique level. The level of the root node is 0, and its immediate child nodes locate at level 1, and so on. In general, the length of the label of each node represents its level in the tree. Definition 2 provides the mechanism of assigning labels



(a) A unbalanced d-ary Kautz tree. (b) An incomplete Kautz tree $IKTree(2,3,8)$. (c) A complete Kautz tree $KTree(2,3)$.

Fig. 1. Kautz tree and Kautz rings.

to edges and nodes, however, does not require which child nodes of each inner node appear in the tree. As a result, there are many different shapes of Kautz tree which satisfy the definition 2 and have same number of nodes. In this paper, we focus on *balanced Kautz tree*.

Definition 3: A d-ary Kautz tree with depth k is balanced if all leaf nodes are at the level k. A balanced kuatz tree is a complete Kautz tree $KTree(d,k)$ only if the parent node of any leaf node has full child nodes, otherwise it is an incomplete Kautz tree $IKTree(d,k,n)$ with n leaf nodes.

In a d-ary balanced Kautz tree with depth k, the root node has $d+1$ children, each inner node at level k-1 has at least one child and at most d child nodes, each inner node at other level has d child nodes, and the number of inner nodes at level i is d^i+d^{i-1} for $1 \le i \le k$-1. The number of leaf nodes in an incomplete Kautz tree is at least $d^{k-1}+d^{k-2}$ which equals to that of a complete Kautz tree $KTree(d,k\text{-}1)$, and at most d^k+d^{k-1} if it becomes a complete Kautz tree $KTree(d,k)$. Figure.1(a) plots an unbalanced Kautz tree, and Fig.1(b) and 1(c) plot an incomplete Kautz tree and a complete Kautz tree, respectively. The white leaf nodes 201, 021, 012, and 102 are the nodes not appear in Fig.1(b). Once those peers appear in Fig.1(c), the incomplete Kautz tree becomes a complete one.

The recursive method in definition 2 can assign a unique label for each child node of any node x. We propose a operation fundamental $\sigma_1(x)$ to implement the same concept in a efficient and practical manner.

Definition 4: Given a node x=$x_k...x_2x_1$, $\sigma_1^i(x)$ produces a label for its ith child node such that $\sigma_1^i(x)$=$x_{k+1}^i x_k...x_2x_1$ for $1 \le i \le d$. If one of the following conditions holds, $x_{k+1}^i = (x_1 - i + 1) \bmod (d+1)$, otherwise $x_{k+1}^i = (x_1 - i) \bmod (d+1)$.

1) $x_k < x_1 - i + 1 \le x_1$
2) $x_1 < x_k$ and $x_k - d - 1 < x_1 - i + 1$

$$\sigma_m^1(x) = \sigma_1^1(\sigma_{m-1}^1(x)) \quad (2)$$

$$\sigma_m^d(x) = \sigma_1^d(\sigma_{m-1}^d(x)) \quad (3)$$

The operations $\sigma_m^1(x)$ and $\sigma_m^d(x)$ denote the leftmost and rightmost nodes when traversing down m steps from node x. The operation $\sigma_0^i(x)$ denotes the peer x itself for any i, for example, results of $\sigma_0^1(x)$ and $\sigma_0^d(x)$ in Algorithm 2 equal to x itself. The traversal process always selects the first child in each step to arrive at the leftmost node, and chooses the last child in each step to reach the rightmost node. For example, $\sigma_2^1(0)$, $\sigma_2^1(1)$, $\sigma_2^1(2)$ denote nodes 020, 101, 212 in Fig.1(c), and $\sigma_2^2(0)$, $\sigma_2^2(1)$, $\sigma_2^2(2)$ denote nodes 210, 021, 102 in Fig.1(c).

The operation $\sigma_1(x)$ is able to allocate an unique label for each child of node x, as well as ranking all the child nodes in an ascend order. Thus, it solves the first obstacle to construct and use a Kuatz tree, however, cannot locate the position of node $\sigma_1(x)$ among all child nodes with same parent node. The position information of node $\sigma_1(x)$ is crucial to find its right and left adjacent nodes at same level, and to calculate the distances from it to other nodes if there is a total order of nodes at same level. We design Algorithm 1 to address this issue as follows.

Algorithm 1 $position(x)$

Require: x=$x_kx_{k-1}...x_2x_1$ is a label of a node in a balanced Kautz tree, and the length of it is larger than 1.

```
number ← 0
if x_1 = x_{k_1} then
    number ← (x_1 − x_k) mod (d + 1)
else
    if x_1 < x_{k−1} then
        if x_1 < x_k < x_{k−1} then
            number ← x_1 − x_k + d + 1
        else
            number ← (x_1 − x_k + 1) mod (d + 1)
    else
        if x_{k−1} < x_k ≤ x_1 then
            number ← x_1 − x_k + 1
        else
            number ← (x_1 − x_k) mod (d + 1)
Return number
```

C. The Kautz ordering of nodes in a complete Kautz tree

There are d^k+d^{k-1} nodes with label $x_kx_{k-1}...x_2x_1$ over $\{0,1,...,d\}$ at any level k in a balanced Kautz tree, where $x_{i+1} \ne x_i$ for $1 \le i < k$. If there is a linear total ordering of nodes at same level, these nodes can form a ring and keep their locality. For any level, the left-to-right traversal of nodes at the level can form a total ordering, denoted as Kautz ordering. We first rank the child nodes of root node in ascend order as well as rank all the child nodes of each node at level 1, respectively. Note that the nodes at level 2 inherit the order of nodes at level 1. The Kautz ordering of nodes at level 1 is 0, 1, 2 as shown in Figure1(c), and the order of nodes 20, 10 must be less than that of nodes 01, 21, and nodes 01, 21 also less than nodes 12, 02. By ranking nodes from level 1 to k recursively, we can find a Kautz ordering of nodes at each level. In the remainder of this paper, we refer to the Kautz ordering of nodes as a Kautz ring. A Kautz ring of nodes at level i stars from node $\sigma_i^1(root)$ for $1 \le i \le k$.

This method requires that each node has global knowledge of the entire tree structure. This requirement, however, is difficult to be satisfied for distributed applications. We address this issue by establishes a predecessor and successor for each node based on the label of each node.

Definition 5: For any node x, its predecessor is the last existing node anti-clockwise from it in a Kautz ring of existing nodes at same level, and its successor is the first existing node clockwise from it in the same Kautz ring. The concept of left adjacent node is similar to the predecessor, but the Kautz ring is consisted of all possible nodes not just existing nodes. So do the right adjacent node and successor node.

For example, the predecessor and successor of node 212 are nodes 121 and 202 in a Kautz ring of solid leaf nodes in Fig.1(b), while the left and right adjacent nodes of node 212 are nodes 021 and 012 in a Kautz ring of solid and white leaf nodes in Fig.1(b).

This method achieves the same result by traversing the nodes from left to right as follows. For the leftmost child node x of root node, we look for its successor node y, find the successor node of node y, and so on, finally get a Kautz ring of nodes at level 1 when meets the node x again. The Kautz ring of nodes at other levels can be achieved similarly. In terms of a complete Kautz tree, the predecessor and left adjacent of each node are the same node, and the successor and right adjacent of each node are also the same node. To discover the right and left adjacent nodes of node x based to its label is another challenge, and we design algorithm 2 to address the issue.

Algorithm 2 Adjacent($x_k...x_2x_1$)

Radjacent($x_k...x_2x_1$)

1: **for** $i = k-1$ down 1 **do**
2: $y = x_i...x_2x_1$, $z = x_{i+1}x_i...x_2x_1$, $j \leftarrow position(z)$
3: **if** $j < d$ **then**
4: Return $\sigma^1_{k-i-1}(\sigma^{j+1}_1(y))$
5: $right \leftarrow (x_1 + 1) \bmod (d+1)$
6: Return $\sigma^1_{k-1}(right)$

Ladjacent($x_k...x_2x_1$)

1: **for** $i = k-1$ down 1 **do**
2: $y = x_i...x_2x_1$, $z = x_{i+1}x_i...x_2x_1$, $j \leftarrow position(z)$
3: **if** $1 < j$ **then**
4: Return $\sigma^d_{k-i-1}(\sigma^{j-1}_1(y))$
5: $left \leftarrow (x_1 - 1) \bmod (d+1)$
6: Return $\sigma^d_{k-1}(left)$

If node x=$x_kx_{k-1}...x_2x_1$ at level k is not the last child of its parent node, the right adjacent node of node x can been found after one loop in Algorithm 2. Otherwise, the problem becomes to find the right adjacent node $y_{k-1}...y_2y_1$ of node $x_{k-1}...x_2x_1$ at the level $k-1$, and then find the leftmost node when traversing down one step from node $y_{k-1}...y_2y_1$. If node $x_{k-1}...x_2x_1$ is not the last child of its parent node $x_{k-2}...x_2x_1$, Algorithm 2 will find the right adjacent node for it after one loop, and response the right adjacent node for node x after two loops. Otherwise, the problem becomes to find the right adjacent node $y_{k-2}...y_2y_1$ of node $x_{k-2}...x_2x_1$ at level $k-2$, and then find the leftmost node when traversing down two steps from the node $y_{k-2}...y_2y_1$, and so on. If the node x represents the rightmost one among the paths from any node at level k to the root node through the node x_1, Algorithm 2 assigns the right adjacent node for it with another node, which represents the leftmost one among the paths from any node at level k to the root node through the node $(x_1 + 1) \bmod (d+1)$. For example, 102 is such a node in Fig.1(c), and the adjacent node of it is the node 020 which represents the leftmost one among the paths from any node at level k to the root node. In this case, the right adjacent node for the node x can be found after $k-1$ loops.

The left adjacent node of the node x can be achieved after $k-1$ loops of Ladjacent method in Algorithm 2 if x is not the first child of its parent node. The Ladjacent method deals with other cases in a similar way as the Radjacent method does. If the node x represents the leftmost one among the paths from any node at level k to the root node, for example the node 020 in Fig.1(c), the left adjacent node of it is node 102, which represents the rightmost one among the paths from any node at level k to the root node. In such case, the left adjacent node of the node x can be found after $k-1$ loops.

On the basis of a Kautz ring of nodes at any level, we measure distance between any two nodes at that level based solely on their labels. The clockwise distance is calculated by Algorithm 3. The anti-clockwise distance equals to $d^k + d^{k-1}$ minus the clockwise distance. The objective of these two algorithms is to estimate the lengths of two paths from a beginning node to a destination node in the clockwise and anti-clockwise order, respectively.

Algorithm 3 ClockwiseDistance(x, y)

Require: $x = x_k...x_2x_1$ and $y = y_k...y_2y_1$ are labels of two nodes in a complete Kautz tree.

1: $distance \leftarrow 0$
2: $u \leftarrow null$, $v \leftarrow null$
3: Let m denotes the length of common suffix of x and y.
4: **if** m=0 **then**
5: $distance \leftarrow |x_1 - y_1 - 1| \times d^{k-1}$
6: **else**
7: **if** $position(x_{m+1}...x_1) < position(y_{m+1}...y_1))$ **then**
8: $u \leftarrow x$,$v \leftarrow y$
9: **else**
10: $u \leftarrow y$, $v \leftarrow x$
11: $l \leftarrow \min\{position(x_{m+1}...x_1), position(y_{m+1}...y_1)\}$
12: $r \leftarrow \max\{position(x_{m+1}...x_1), position(y_{m+1}...y_1)\}$
13: $distance \leftarrow (r - l - 1) \times d^{k-m-1}$
14: **for** $i = m+1$ to $k-1$ **do**
15: $distance+ \leftarrow (d - position(u_{i+1}...u_2u_1)) \times d^{k-i}$
16: $distance+ \leftarrow (position(v_{i+1}...v_2v_1) - 1) \times d^{k-i}$
17: **if** $u = x$ **then**
18: Return $distance + 1$ {The x is less than y in the order}
19: **else**
20: Return $(d+1) \times d^{k-1} - distance - 1$

For a complete Kautz tree with height $k+1$, the subtree of it with any node at level i as root has number of d^{k-i} leaf nodes, for example, the subtree with any node at level 1 as root has d^{k-1} leaf nodes. The clockwise distance between any nodes x=$x_k...x_2x_1$ and y=$y_k...y_2y_1$ can be calculated by a recursive manner as shown in Algorithm 3. Let m denote the length of longest common suffix of the two nodes. At level $m+1$, we first obtain the positions of nodes $x_{m+1}...x_2x_1$ and $y_{m+1}...y_2y_1$ among all child nodes of the node $x_m...x_2x_1$ through Algorithm 1, and then calculate the number of leaf nodes in the subtrees rooted at nodes which locate between the nodes $x_{m+1}...x_2x_1$ and $y_{m+1}...y_2y_1$. We also consider the same issue at other level j where $m+2 \leq j \leq k-1$, respectively. After all nodes locating between the nodes x and y at same level are selected by the above process, we know the clockwise and anti-clockwise distances between the two nodes.

D. The Kautz ordering of nodes in an incomplete Kautz tree

As shown in Fig.1, the predecessor and successor nodes are sometimes not the left and right adjacent nodes for any leaf node in a $IKTree(d,k,n)$. The methods proposed above can construct a Kuatz ring of inner nodes in a $IKTree(d,k,n)$, but does not work for leaf nodes. On the other side, a complete Kautz tree has a unique shape, but an incomplete one might have many shapes. Consequently, a leaf node cannot deduce its predecessor and successor as does in $KTree(d,k)$. If there exists a leaf node which has global knowledge about the tree structure, it answers the predecessor and successor for other leaf nodes. Otherwise, each leaf node cooperates with other leaf nodes to discover its predecessor and successor in a distributed manner as shown in Section III. For the same reasons, Algorithm 3 cannot always correctly measure distance between any two leaf nodes in an incomplete Kautz tree based only on labels.

Algorithm 4 ApproximateDistance(x, y, a)

Require: $x = x_k...x_2x_1$ and $y = y_k...y_2y_1$ are labels of two nodes in a balanced Kautz tree.

1: $u \leftarrow x_{k-1}...x_2x_1$
2: $v \leftarrow y_{k-1}...y_2y_1$
3: **if** $u = v$ **then**
4: Return $ClockwiseDistance(x,y)$
5: $distance \leftarrow ClockwiseDistance(x, \sigma_1^{a+1}(u))$
6: $distance+ \leftarrow ClockwiseDistance(\sigma_1^1(v), y)$
7: $distance+ \leftarrow (a+1) \times (ClockwiseDistance(u,v) - 1)$
8: Return $distance$

In Section III, the topology construction rule imposes constraint on the shape of an incomplete Kautz tree $IKTree(d,k,n)$. It allocates the first a child nodes for each inner node at level $k-1$, and then the $(a+1)$th child node for number of $n - \alpha \times (d^{k-1} + d^{k-2})$ inner nodes at level $k-1$ in Kautz or random order, where $a = \lfloor n/(d^{k-1} + d^{k-2}) \rfloor$. For a pair of leaf nodes with same parent node, the distance between them calculated by Algorithm 3 is equivalent or very close to the real value whenever the tree is stable or dynamic. For other pairs of leaf nodes, the result of Algorithm 3 is usually larger than the real value, for example, the clockwise distance from node 210 to 202 is 4 in Fig.1(b), but the algorithm responses 7.

In order to measure distance between a pair of any leaf nodes as accurate as possible, we design an improved Algorithm 4 based on Algorithm 3 to support an incomplete as well as a complete Kautz tree. The measured result will be used to optimize the routing strategy so as to forward a message along a path as short as possible. For a static or moderately dynamic incomplete Kautz tree, the estimated result is close even the same as the real value.

III. BAKE: A Balanced Katuz Tree Based Overlay

We propose a structuring strategies to organize peers into an efficient overlay network which can guarantee logarithmic network diameter and constant out-degree of each peer. All the structuring strategies are based on the concept of balanced d-ary Kautz tree. First, each peer maps to one leaf node in a balanced Kautz tree, and uses label of its related leaf node and IP address as its logical and physical identifiers, respectively. Second, each peer maintains $d+2$ neighbor peers according to a topology rule. Third, any resource gets an identifier from an identifier space which contains the identifier space of peers. Resources are distributed at given peer based on the longest suffix matching rule. Based on the above three strategies, we propose a robust routing scheme to support different operations effectively, such as resource distribution, resource query and topology maintain.

A. Topology construction rule

As mentioned in Section II, a balanced Kautz tree is usually an incomplete one, and is a complete one in special cases. For a complete Kautz tree $KTree(d,k)$, number of $d^k + d^{k-1}$ peers form an desirable topology by the following rule. For each peer x=$x_k..x_2x_1$, its successor and predecessor are peer $Radjacent(x)$ and peer $Ladjacent(x)$, and its ith out-neighbor and in-neighbor are $\varsigma_1^i(x)$ and $\tau_1^i(x)$ for $1 \le i \le d$, respectively. Peer x maintains total number of $d+2$ links to its predecessor, successor, and number of d out-neighbors. On the other hand, its predecessor, successor, and number of d in-neighbors also maintain $d+2$ links to peer x. The diameter of such overlay network is k. The $\varsigma_1^i(x)$ and $\sigma_1^i(x)$ are defined as follows.

Definition 6: $\varsigma_1^i(x)$ denotes an operation such that $\varsigma_1^i(x) = x_{k-1}...x_1x_0^i$ for $1 \le i \le d$. If one of the following conditions is satisfied, then x_0^i=$(x_k + i - 1) \bmod (d+1)$. Otherwise, x_0^i=$(x_k + i) \bmod (d+1)$.

1) $x_k < x_k + i - 1 \le x_1$;
2) $x_k + i - 1 < x_1 + d + 1$ and $x_1 < x_k$.

Definition 7: Given $\sigma_1^i(x) = x_{k+1}^i...x_2x_1$, $\tau_1^i(x)$ denotes an operation such that $\tau_1^i(x) = x_{k+1}^i...x_3x_2$ for $1 \le i \le d$.

P2P overlays must support arbitrary number of peers in order to deal with the uncontrolled dynamic operations of peers, such as join, depart and fail. Unfortunately, an overlay network based on a d-ary complete Kautz tree holds the desirable topology and works only if number of peers n equals to a series of discrete numbers, such as $d+1$, $d(d+1)$,...,$d^k(d+1)$ and so on. When n is larger than the number of leaf nodes in $KTree(d,k)$ but less than that in $KTree(d,k+1)$ for $1 \le k$, each peer loses partial links because of the absence of some out-neighbors and in-neighbors. In this scenario, a P2P overlay constructed by the above rules no longer possesses the desired features, such as optimal diameter and efficient routing scheme, and suffers from failed routing between partial pairs of nodes.

To address this issue, we propose a general overlay network, BAKE, based on an incomplete Kautz tree $IKTree(d,k,n)$. Number of n peers form an overlay network according to the following rules. For each peer x=$x_k...x_2x_1$, it maintains total number of $d+2$ links to its predecessor, successor, and number of d out-neighbors. The predecessor and successor are identified in a distributed manner as shown in the remainder of this section. The d out-neighbors are the peers satisfying one of the following conditions. For $1 \le i \le d$:

1) If a peer $\varsigma_1^i(x)$ has appeared in the overlay network, it is the ith out-neighbor of peer x;
2) Otherwise, if $\varsigma_1^i(x)$ and its predecessor y have a common suffix with length $k-1$, peer y is the ith out-neighbor of peer x.
3) Otherwise, if $\varsigma_1^i(x)$ and its successor z have a common suffix with length $k-1$, peer z is the ith out-neighbor of peer x.

Theorem 1: The above construction rules can guarantee that any peer $x = x_k...x_2x_1$ in BAKE based on $IKTree(d,k,n)$ has d out-neighbors besides its successor and predecessor.

Proof: All nodes $\varsigma_1^i(x)$ for $1 \leq i \leq d$ are out-neighbors of node x in a d-ary complete Kautz tree $KTree(d,k)$. For a value of i such that node $\varsigma_1^i(x) = x_{k-1}...x_2x_1x_0^i$ does not appear in a d-ary incomplete Kautz tree with depth k, definition 3 can guarantee that at least one child node of node $x_{k-2}...x_2x_1x_0^i$ appears in the tree and replaces the node $\varsigma_1^i(x)$ as the ith out-neighbor of the node x. Thus, d nodes act as the out-neighbors of the node x in the tree. If the right adjacent node y=$Radjacent(x)$ of node x does not appear in the tree, then finds the right adjacent node of node z=$Radjacent(y)$. If node z does not appear in the tree, then finds its right adjacent node, and so on. The definition of incomplete Kautz tree makes sure that the successor of each node is at most d hops away clockwise from it in a related Kautz ring. The predecessor of node x can be found in a similar method. Therefore, Theorem 1 holds. ■

Theorem 1 guarantees that each peer has d out-neighbors and a successor and predecessor in a static environment. In a dynamic environment, the topology adjustment, peer joining and departing strategies make sure that ith out-neighbor of each peer x is available if no peer fails, where $1 \leq i \leq d$. In practice, the ith out-neighbor of peer x becomes unavailable when all peers $\sigma_1^j(x_{k-2}...x_1x_0^i)$ have not joined BAKE or failed simultaneously after joined BAKE, where $1 \leq j \leq d$. To deal with the negative impacts of peers which fail randomly or concurrently, two dedicated mechanisms are presented to maintain the topology in next section. A powerful stabilization strategy is introduced to discover failed peers efficiently, and then a flexible peer joining strategy attempts to make unavailable out-neighbors of each peer become available.

In summary, the number of out-neighbors of an existing peer in BAKE is usually d in a static or moderately dynamic environment, and is sometimes less than d but becomes d after a recovery period in a highly dynamic environment. The successor and predecessor of each existing peer are always exist in any kind of environment.

B. Resource placement based on longest suffix matching

For any resource to be distributed in BAKE, it is assigned a long d-ary identifier x=$x_l...x_k...x_2x_1$ according to its value of single or multiple dimension attributes. A peer $x_k...x_2x_1$ is the preferred host of the resource x if the peer appears in BAKE, otherwise one existing peer with a suffix $x_{k-1}...x_2x_1$ acts as the second host. If no peer fails during runtime, the topology adjusting, peer joining and departing strategies make sure that at least peer $\sigma_1^1(x_{k-1}...x_2x_1)$ always appears in BAKE, and then the preferred or second host of each resource indeed exists. Otherwise, those existing peers with $x_{k-1}...x_2x_1$ as suffix sometimes fail, and then the preferred and second host peers might become unavailable.

To address this issue, we define the predecessor of $x_k...x_2x_1$ as the third host of the resource x. For example, a resource with identifier 012021 will be stored by its preferred host peer 021 if this peer exists, but is really taken over by its second host peer 121 that is the predecessor of node 021 in a Kautz ring as shown in Fig.1(b). If the peer 121 fails, those incoming resources with 021 as suffix will be stored by its third host peer 101 that is the predecessor of the peer 121. If the failed peer recovers or is replaced by other new peer, resource x should be transferred to its first or second host peer.

There are two advantages of this resource placement strategy. First, it makes sure that any resource can be stored by an identified peer successfully even if its preferred and second host peers do not appear in BAKE. Second, it guarantees that any resource is stored at a peer as close as possible to its preferred peer, and provides some useful hints to support exact and range queries about related resources. Algorithm 5 explains the resource placement strategy formally, and related detail issues are illustrated in Algorithm 6.

Algorithm 5 Placement(x)

Require: $x_k...x_2x_1$ is a suffix of the long identifier of resource x.
1: **if** peer $x_k...x_2x_1$ has appeared in the overlay network **then**
2: The peer $x_k...x_2x_1$ stores the resource x.
3: **else**
4: **if** $x_k...x_2x_1$ and its predecessor y have common parent node in related Kautz tree **then**
5: The peer y is the host peer of the resource x.
6: **else**
7: **if** $x_k...x_2x_1$ and its successor z have common parent node in related Kautz tree **then**
8: Peer z is the host peer of the resource x.
9: **else**
10: The peer y is the host peer of the resource x.

C. Robust and effective routing

The messages handled by BAKE are partitioned as two categories, that is, messages to publish or query a resource, and messages to maintain the topology. In order to route these kinds of messages to correct destinations effectively, each peer should keep a routing table and establish overlay connections with at most $d+2$ existing peers based to the topology construction rules mentioned above. A routing table of each peer often contains $d+2$ entries, and each entry is consisted of logical identifier and physical address (such as IP and port number) of a neighbor peer. Fiol et al. proposed a shortest path routing scheme for a similar routing problem [31], and we also introduced an improved and practical scheme in previous works [29]. The idea of those schemes to route a message from peer x to peer y along a shortest path is as follows. The peer x finds the longest suffix u of x that appears as a prefix of y, and then walks towards a neighbor z such that its longest suffix v coincides with a prefix of y and the length of v is larger than that of u.

Those schemes work well in a static environment, but suffer from dynamic P2P networks. For example, peer 202 routes a message to peer 101 along the shortest path $202 \rightarrow 121 \rightarrow 210 \rightarrow 101$ in Fig.1(b), and fails if one peer in the path becomes unavailable. The reason is that a message is only forwarded towards the unique neighbor which is more close to the destination than itself and other neighbors. To address this issue, we propose a robust routing scheme in BAKE, by allowing peers to send a message towards another neighbor when fails to route it along the expected path. Please refer to Algorithm 6 for more details.

When a peer x=$x_kx_{k-1}...x_2x_1$ publishes or lookups a resource with identifier $y_l...y_k...y_2y_1$, the preferred destination is peer y=$y_k...y_2y_1$. If peer y does not exist, the peer x immediately identifies an peer z as the second host of this resource, and then forwards the message to peer z. If such a peer z does not exist, peer x routes the message towards its third host peer. For example, in Fig.1, peer 010 routes a resource with identifier 012021 towards peer 021 along a path $010 \rightarrow 202 \rightarrow 021$, and peer 202 will find that the preferred host peer 021 does not exist. It then forwards the message to second host peer 121. If peer 202 finds the peer

Algorithm 6 Route(y, message,model)

1: **if** $x = y$ **then**
2: **if** model=peer **then**
3: Return $available$.
4: **else**
5: Process the message locally, and return $success$.
6: **else if** Comsuffix$(x, y) = k - 1$ **then**
7: **if** Comsuffix$(x, x.successor) = k - 1$, and $x.successor$ is less than y in the kauz ring **then**
8: Forward the message to peer $x.successor$
9: **else**
10: Process the message locally, and return $success$.
11: **if** Comsuffix$(x, x.predecessor) = k - 1$, and $x.predecessor$ is less than y in the kauz ring **then**
12: Forward the message to peer $x.predecessor$
13: **else**
14: Process the message locally, and return $success$.
15: **else**
16: **if** Exists at least one neighbor z of x such that Common(z, y) is larger than Common(x, y) **then**
17: Forward message to the peer which has the largest value of Common(z, y).
18: **else**
19: **if** $common(z, y) = k - 1$ **then**
20: **if** model=peer **then**
21: Return $unavailable$
22: **else if** Exists a neighbor peer z of peer x such that Comsuffix$(z, y) = k - 1$ **then**
23: Forward the message to peer z.
24: **else**
25: Route($Ladjacent(\sigma_1^1(y_{k-1}...y_2y_1))$, message,model).
26: **else**
27: Forward the message to one of existing neighbors because the neighbor towards the destination peer is unavailable.

Comsuffix(x, y)

1: Return the length of the longest common suffix of x and y.

Common(x, y)

1: Let u be the longest suffix of x which appears as a prefix of y.
2: Return the length of u.

121 also fails, it routes the message towards peer 201 along a path $202 \rightarrow 020 \rightarrow 201$. The resource is finally stored by a more accurate destination, peer 101, identified by peer 020. Note that the decision on new destination can be made based only on local knowledge at each peer.

Algorithm 6 gives a formal and detailed explanation about our routing scheme, and uses the following three parameters. Parameter y denotes an identifier of a resource or destination peer. Parameter $message$ denotes the real message needed to be routed. Parameter $model$ denotes the type of message, which can be $peer$ or $resource$. If a peer issues a message to detect the status of a peer y, then $model$=$peer$. The destination of this message cannot be changed during the routing process in order to reflect the real status of peer y. If a message is used to publish or lookup a resource, then $model$=$resource$ and its destination peer might be adjusted to locate a suitable destination in dynamic environment.

If each peer uses all neighboring peers except its predecessor and successor when routing messages, Algorithm 6 exhibits a short path but not always the shortest. In other words, the routing scheme routes messages among a majority of peers along the shortest path, and messages among all peers along a path with at most k hops. As mentioned above, the clockwise and anti-clockwise distance between source and destination peers can be estimated by Algorithm 4. If each peer also employs its predecessor and successor when routes messages, it can select a shortest one among the traditional path, a path along predecessor links, and a path long successor links. For example, a traditional short path from peer 020 to peer 121 is $020 \rightarrow 101 \rightarrow 212 \rightarrow 121$, and $020 \rightarrow 101 \rightarrow 121$ is the shortest path that walks along the successor link of peer 101. The advantage of this new strategy is more remarkable when $d \leq k$. Algorithm 6 reflects this improvement after minimal modification. In summary, our routing scheme is effective and robust, and the predecessor and successor of each peer can help to enhance the robustness and efficiency.

D. Query processing

Clearly, BAKE can support exact-match queries of resources in an efficient and robust manner. In order to manage complex resources and support more wide applications, BAKE should also support complex query operations besides the exact-match query in a graceful fashion, for example, the range query for numerical values. The pre-condition of rang query in P2P network is to distribute those resources that keep an order in single or multiply attributes space to peers in a locality-preserving manner. In other words, those resources with attribute values close to each other should be stored on a same peer or neighboring peers. To achieve this goal, BAKE must address the following three critical issues, including a total ordering of peers, a locality-preserving naming and placement strategy of resources.

The topology construction rule guarantees that each peer u has an successor link to a peer v such that the clockwise distance from u to v is less than that to other peers, and all peers form a Kautz ring by the successor links. On the other hand, each resource is stored on a peer which has the longest common suffix with the identifier of resource. BAKE selects the first peer meeting the criteria along the Kautz ring in clockwise. Hence, the first two issues are addressed. For the third issue, we recursively partition the attribute space of resources into sub-spaces similarly as constructing a complete Kautz tree, and then associate each sub-space with an identifier in the same way as allocating identifiers for nodes in Kautz tree. Each resource finds the smallest sub-space that contains its attribute values, and uses the identifier of that sub-space as the identifier. Therefore, those resources with attribute values close to each other will obtain adjacent identifiers in the Kautz ring, or even a same identifier, and are stored by same or neighboring peers obeying the locality-preserving placement strategy.

Any peer $x_k...x_2x_1$ that issues a range query can find the identifier y of smallest sub-space that contains the entire query region. If the length of y is larger than k, a peer whose identifier is the suffix of y will process the query. Otherwise, the query covers multiple peers and will be routed towards the peer which reaches the lower bound of the region firstly. Once receiving the query, the peer sends the query message to its successor peer if it cannot cover the entire region of the query. So on and so forth, a query message will traverse all intersection peers and collects all resources satisfying the query constraints.

This method can decrease the message cost caused by transferring the query to all intersection peers, but the delay could be a litter larger than forwarding the query to all related peers simultaneously. We need to set a tradeoff between the delay and the cost. A problem of this method is that it requires a priori partitioning. To address this issue, the previous solution can be

improved by employing a dynamic space portioning method based on a coarse priori partitioning.

IV. Topology management

A. Topology adjustment

A native BAKE based on an initial $KTree(d,k)$ can be constructed in advance. All leaf nodes of $KTree(d,k)$ are allocated to $(d^k + d^{k-1})$ peers. When more new peers want to join, however, there is no additional available leaf nodes to use in $KTree(d,k)$. In such a situation, we expand $KTree(d,k)$ to achieve an incomplete Kautz tree $IKTree(d,k+1,d^k+d^{k-1})$. If number of peers reaches $(d+1)\times(d^k+d^{k-1})$, the incomplete Kautz tree becomes a complete one and is ready to expand further.

It is straightforward to expand $KTree(d,k)$ to achieve an incomplete Kautz tree $IKTree(d,k,d^k+d^{k-1})$ by adding the first child node of each leaf node in the complete Kautz tree $KTree(d,k)$. To expand the topology of BAKE, we introduce an efficient solution that needs only local operations at each existing peer. For an existing peer with identifier x=$x_kx_{k-1}...x_2x_1$ in BAKE, we

1) Update its logical identifier with $\sigma_1^1(x)$.
2) Update the logical identifier of its successor peer $x' = x'_kx_{k-1}...x_2x_1$ with $\sigma_1^1(x')$. The logical identifier of its predecessor peer is updated similarly.
3) Update the logical identifier of its out-neighbor peer $\varsigma_1^i(x)$ with $\sigma_1^1(\varsigma_1^i(x))$ for $1 \le i \le d$.

Theorem 2: The topology expanding process of the entire overlay network does not cause additional network overhead except $d^k + d^{k-1}$ messages to start the process.

Proof: A resource whose identifier has a suffix $x_k...x_2x_1$ is stored by a peer x=$x_k...x_2x_1$. Peer x updates its logical identifier with $\sigma_1^1(x)$=$x^i_{k+1}x_k...x_2x_1$ after expanding. The peer is the preferred host of the resources whose identifiers have suffix $\sigma_1^1(x)$, and becomes the second host of other resources stored at it before. Therefore, the expanding process does not introduce any network overhead because each resource still stays at the original peer after the process.

Before the process, each peer x maintains links to its out-neighbors $x_{k-1}...x_1\alpha$ where $\alpha \in \{0,1,2,...,d\} - \{x_1\}$. After the process, the peer has a new logical identifier $\sigma_1^1(x) = x^1_{k+1}x_k...x_2x_1$, and maintains links to peers $x'_k...x_2x_1\beta$, where $\beta \in \{0,1,2,...,d\} - \{x_1\}$ and the value of x'_k obeys to the topology construction rule of incomplete Kautz tree mentioned above. Note that the parent nodes of out-neighbors of node $\sigma_1^1(x)$ are $x_{k-1}...x_1\beta$ where $\beta \in \{0,1,2,...,d\} - \{x_1\}$, that are the same neighbors before expanding although their logical identifiers are updated with the labels of their first child nodes. On the other hand, the physical identifiers of successor and predecessor of peer $\sigma_1^1(x)$ do not change although the logical identifiers are updated. In words, the links maintained by each peer do not change, and no network overhead is further incurred. Thus, Theorem 2 holds. ■

In contrast to expand the overlay, BAKE shrinks its topology when the number of existing peers decreases to number of leaf nodes in a d-ary complete Kautz tree. We shrink an incomplete Kautz tree $IKTree(d,k,d^k+d^{k-1})$ to a complete Kautz tree $KTree(d,k)$ by deleting all original leaf nodes. For an existing peer $x = x_{k+1}...x_2x_1$, we replace its logical identifier with $x_k...x_2x_1$, and update the logical identifiers of its predecessor, successor, and number of d out-neighbors in a same way. Obviously, we do not cause much network overhead except $d^k + d^{k-1}$ messages to start the process.

B. Peer joins

To ensure that our routing scheme executes correctly after a new peer participates BAKE, all routing entries of each peer must keep up to date. BAKE handles this issue by a series of local operations that each new peer runs when it joins.

The peer related to the leftmost leaf node $\sigma_{k-1}^1(0)$ in a d-ary incomplete Kautz tree with depth k acts as the first entry point of BAKE, and its predecessor acts as a synchronous second entry point. The entry points first serve as general peers, and also manage all the labels of leaf nodes in the incomplete Kautz tree. The entry point allocates a leaf node to a peer as follows. First, if a peer fails and recovers in time, the entry point allocates the previous leaf node to the peer if that leaf node is not assigned yet. This can prevent those resources stored at a failed peer from being transferred to other peer after that peer recovered. Second, it allocates the leaf node x whose $position(x)$ is 1, implying that, it is the first child node of its parent node, and then the leaf node whose position is 2, implying that, it is the second child of its parent, and so on. Third, for the leaf nodes with same position value, it allocates them in the clockwise order at a Kautz ring of their parent nodes, denoted as $Korder$, or based on the load (storage and accessing load) of their predecessors in descend order, denoted as $Border$. If all leaf nodes are allocated, the first entry point starts the topology expanding process. It may also start the topology shrink process when required.

Before participating BAKE, a new peer consults the first entry point for a logical identifier x=$x_k...x_2x_1$, predecessor y and successor z, and construct its topology by the following process. Note that the peers that have $x_{k-1}...x_2x_1$ as common suffix possess the same out-neighbors. If there exists at least one of such peers, peer x can get a copy of the out-neighbors from its predecessor or successor. Otherwise, peer x use Algorithm 7 to discover its out-neighbors obeying the topology rules in the worst case. Indeed, all in-neighbors of each peer $\varsigma_1^i(x)$ for $1 \le i \le d$ are not available, and there is no routing path along in-neighbor links to these peers. Thus, Algorithm 7 routes a query message towards the left or right adjacent peer of $\varsigma_1^i(x)$ for $1 \le i \le d$. The peer forwards a message to a destination along predecessor or successor link. The following algorithm gives a detail explanation about the procedure.

Peer x adds peers y, z as its predecessor and successor.
if Comsuffix$(x,y) < k-1$ **then**
  if Comsuffix$(x,z) = k-1$ **then**
    Peer x informs peer z to update its predecessor with x, then transfers its resource u that Comsuffix(x,u)=k and returns its routing table to peer x. Peer x has the same out-neighbors as peer z. Peer z also informs peer y to update its successor with peer x.
  else
    Peer x informs peer y to update successor with x, and return its resource u that Comsuffix(x,u)=k to peer x. Peer y informs z to update its predecessor with x. Peer x must find its out-neighbors by invoking Algorithm 7.
else
  Peer x informs peer y to update its successor with x, then transfers its resource u that Comsuffix(x,u)=k and

returns its routing table to peer x. Peer x has the same out-neighbors as peer y. Peer y also informs peer z to update its predecessor with peer x.

Peer x also takes some actions to handle the impact of its participation on other peers. It informs peer $\tau_1^i(x)$ for $1 \leq i \leq d$ to update an out-of-date neighbor with it by sending a message. The message is routed towards a peer $u = \tau_1^1(x)$, and forwarded to other related peers along the successor links.

Algorithm 7 Findneighbor(x)

1: **if** Comsuffix($\varsigma_1^1(x), Ladjacent(\varsigma_1^1(x))) = k-1$ **then**
2: $u \leftarrow Ladjacent(\varsigma_1^1(x))$
3: **else**
4: $u \leftarrow Radjacent(\varsigma_1^1(x))$
5: Create a message containing $\varsigma_1^i(x)$ for $1 \leq i \leq d$, and route it towards peer u based on the routing scheme.
6: The message reaches a peer v such that $Common(u, v) = k-1$, and peer v forwards it to peer $\varsigma_1^j(v)$ for $1 \leq j \leq d$.
7: Once peer $\varsigma_1^j(v)$ receives the message, it selects a w from $\varsigma_1^i(x)$ for $1 \leq i \leq d$ such that $Comsuffix(v, w) = k-1$, and then forwards the message to w along the successor links or predecessor links.
8: Peer $\varsigma_1^j(v)$ selects one from peer w, its predecessor, and its successor according to the topology rules, and returns the selection result as one out-neighbor of peer x.

Theorem 3: For BAKE based on an incomplete Kautz tree $IKTree(d, k, n)$, to join a new peer x will affect one out-link of at most $\lceil 2 + n/(d^{k-1} + d^{k-2}) \rceil$ existing peers.

Proof: The first entry point allocates label of $(i+1)$th child node of all inner nodes at level $k-1$ to new peer only if all ith child nodes have been allocated for $1 \leq i \leq d-1$. The number of ith child node of all inner nodes at level $k-1$ in $KTree(d, k, n)$ is $d^{k-1} + d^{k-2}$ for $1 \leq i \leq d$. The topology rules make sure that at most $\lceil n/(d^{k-1} + d^{k-2}) \rceil$ peers use peer x as a out-neighbor. On the other hands, peer x also affects its successor and predecessor. In summary, at most $\lceil 2+n/(d^{k-1} + d^{k-2}) \rceil$ peers need to update one routing entry and link, and thus Theorem 3 holds. ■

C. Peer fails and stabilizes

The correctness and effectiveness of BAKE relies on the fact that the predecessor, successor, and out-neighbors of each peer are up to date. However, this invariant can be compromised if peers fail. For example, in Fig.1(b), if node 120 fails, node 020 will not know that node 010 is now its successor, and node 212 will not know that 120 is not one of its neighbors now. An incorrect neighbor will increase the delay of routing a message, and even fail to deliver messages correctly.

To improve robustness of topology and keep high effectiveness of basic operations, each peer periodically checks the out-links not used in this round, and identify failed peers to inform the first entry point. A failed peer is not found only if $d+2$ peers keeping links to it fail simultaneously, which is very improbable with modest values of d. In practice, a failed peer is often discovered early during communications before the end of the entire round. A failed peer x=$x_k...x_2x_1$ affects at most $d+2$ existing peers. Once the failure of peer x is identified, peers keeping links to peer x repair their local topologies according to the rules.

If predecessor u of peer x first finds that peer x fails, it will detect peers along a related Kautz ring in clockwise, until finds the first existing peer as its new successor v. By comparing peers u, x and v, peer u finds a substitute peer of peer x obeying the following rules. If Comsuffix$(u, x) = k-1$, peer u is the substitute of peer x, else if Comsuffix$(v, x) = k-1$, peer v is the substitute of peer x, else peer u is the substitute of peer x. Peer u will notify this to one in-neighbor peer of peer x, for example, peer $\tau_1^1(x)$. If peer $\tau_1^1(x)$ also fails, the routing schema makes sure the notification message can be routed to related peers. As long as one in-neighbor of peer x receives such notification message, it forwards the message to other related peers along the successor and/or predecessor links. Thus, the negative impact of the failed peer x can be discovered and repaired in time. If the successor of peer x first finds that peer x fails, it executes similar actions as peer u does. In order to decrease the delay and cost of topology adjustment, each peer can keep physical identifiers of multiple successors in Kautz order but establishes link with the first successor only. When the first successor fails, it connects the second successor, and so on. If number of successors of each peer is a moderate value, those successors unlikely fail at the same time.

In most cases, node failures can be handled by peers checking their predecessor and successor. Such a method, however, may fail to handle those peers which connect to each other by predecessor or successor links and fail concurrently. To address this issue, each peer also detects its out-neighbors actively. Among those peers that possess a common suffix with length k-1, only a few peers need to do so because they have same out-neighbors. A peer detects those peers with a suffix $x_{k-1}...x_2x_1$ and identify a substitute once it discovers a failed peer x, and then notifies other related peers of the substitute along successor and /or predecessor link. If no substitute peer exists, it will detect again in next round. The detail process is as follows.

1: **if** Comsuffix$(x, Ladjacent(x)) = k-1$ **then**
2: Detects the left adjacent peer of peer x
3: **if** Does not find a substitute peer of peer x **then**
4: Detects right adjacent of peer x and other related peers in Kautz order, until finds a substitute or meets a peer $\sigma_1^\alpha(x_{k-1}...x_2x_1)$ where $\alpha = \lceil n/(d^{k-1} + d^{k-2}) \rceil$.
5: **else**
6: Detects right adjacent of peer x and other related peers in Kautz order, until finds a substitute or meets a peer $\sigma_1^\alpha(x_{k-1}...x_2x_1)$ where $\alpha = \lceil n/(d^{k-1} + d^{k-2}) \rceil$.

D. Peer departures

It sometimes takes a stabilization period to repair the destroyed routing tables of some peers due to failed peers. Hence, some related messages might suffer from destroyed routing tables. A peer departing from BAKE voluntarily, however, can repair the topology actively before it leaves.

For a peer x=$x_k...x_2x_1$ that is about to leave, if its predecessor y or successor z has a suffix $x_{k-1}...x_2x_1$, it notifies peer y, peer z and the first entry point of BAKE. In turn, peer y will update its successor with z, and peer z will replace its predecessor with peer y. If peer y has a suffix $x_{k-1}...x_2x_1$, peer x transfers its resources to peer y, and notifies its in-neighbor peer $\tau_1^i(x)$ to replace the link to it with another link to peer y for $1 \leq i \leq d$. If peer y does not have a suffix $x_{k-1}...x_2x_1$ while peer z has, peer x transfers

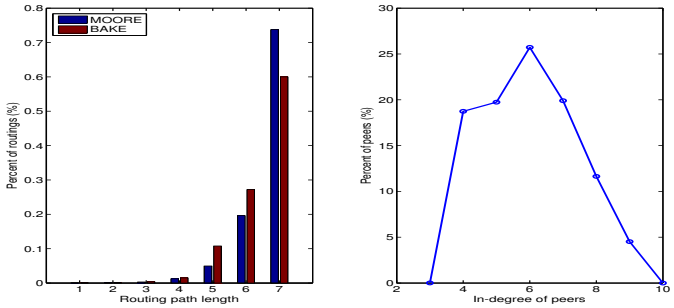


Fig. 2. The routing and in-degree distributions of IKTree(4,7,12800).

its resources to peer z, and notifies its in-neighbor peer $\tau_1^i(x)$ to replace the link to it with another link to peer z for $1 \leq i \leq d$.

If none of peer y and peer z has a suffix $x_{k-1}...x_2x_1$, peer x should find a substituted peer w before it departs. It detects other peers with a suffix $x_{k-1}...x_2x_1$, and selects the peer whose position value is the largest one if there are at least one such peers, otherwise it consults the first entry point of BAKE for peer w. Peer w must satisfy the following two constraints: 1) There are other peers that have a common suffix, length $k-1$, with peer w in BAKE; 2) The value of $position(w)$ should as large as possible. Peer w will perform a voluntary departure operation, and takes over the resources, logical identifier, and routing table of peer x. It then informs neighbors of peer x to update physical identifier of peer x. After this process, the original peer x can depart from BAKE. The selection rule of peer w can guarantee that it does not need to find another substitute peer before it departs.

E. Further improvement

It does not introduce additional overhead to expand or shrink topology except number of n messages to start the two operations, where n denotes number of peers. Clearly, BAKE conducts the less operations less as the order becomes large during its evolution process. BAKE, however, might often expands and shrinks the topology when the value of n fluctuates around $d^k + d^{k-1}$ in the worst case. To address this issue, we propose an delay strategy to decrease frequency of those operations as follows. If all logical identifiers have been assigned to existing peers, BAKE allows a new peer to share a logical identifier with an existing peer. In words, BAKE only expands its topology when the value of n becomes relative stable and exceeds an upper bound. The strategy delays the topology expanding operation, and decreases the load of some peers with the help of new peers. For the similar reason, BAKE only shrinks its topology when the value of n is less than a lower bound and becomes relative stable.

V. Performance

We first show that BAKE achieves optimal diameter and connectivity as the Kautz digraph does through analysis and simulations. We then evaluate the robustness of routing scheme we proposed, and the delay and message cost of major operations.

A. Topology properties

Theorem 4: The out-degree of each peer is $d+2$ in BAKE with n peers, and the diameter is $k_l = \lceil \log_d(n) - \log_d(1+1/d) \rceil$.

Proof: First, we calculate k such that $d^{k-1} + d^{k-2} < n < d^k + d^{k-1}$. Thus, the length of peer identifier must be k, and we can find a pair of peers at distance k along a shortest path. Thus k_l equals to $\lceil \log_d(n) - \log_d(1+1/d) \rceil$, and almost achieves the Moore Bound [21]. ■

The diameter does not reflect the entire view of routing scheme, hence, we study the expected value and distribution of the length of routing paths among many pairs of nodes. In simulations, each peer first issues a message towards others. As shown in the left sub-figure of Fig.2, the length of about 60% routing paths equals to the diameter. If we compare BAKE with MOORE [29] under the same configurations, the length of less routing paths equals to the diameter, and that of more routing paths is less than the diameter. The right sub-figure of Fig.2 indicates that the in-degree of most peers is adjacent to $d+2$, and that of other peers is less than $d+2$.

The expected value of routing path is denoted as ard. We evaluate the ard of BAKE where n ranges from 320 to 23040 and d equals to 4. We compare BAKE with MOORE and other constant degree topologies in which the out-degree of each peer is four, such as CAN, 4-dimensional butterfly, de Bruijn, and Kautz digraph. Fig.3 plots the experiment results. The curves of butterfly, de Bruijn and Kautz digraphs are dashed lines and discrete points since their orders are special discrete sequences. The curves of BAKE and MOORE are solid lines since their orders can be arbitrary values. The ard of BAKE and MOORE are shorter than that of butterfly, CAN, $log_4 n$ and de Bruijn, and the simulation results also confirm this. The ard of BAKE is less than that of MOORE, and is more close to that of Kautz digraph. The ard of Kautz digraph with shortest path routing scheme has been proved in [28]. The ard of BAKE is a little less than that of Kautz digraph when n equals $d^i + d^{i-1}$ for $1 \leq i$, because partial routing paths in Kautz digraph are shortened with the help of the successor and predecessor links of each peer. Hence, BAKE achieves an optimal topology which inherits all good properties of Kautz digraph even its order is out of $d^k + d^{k-1}$ for a given d and any $k > 1$.

B. Robustness of routing

The routing schemes mentioned in [29], [31] suffer from poor robustness in dynamic environments. BAKE addresses this issue by sending a message to another out-neighbor, if it fails to forward the message along a shortest path. The connectivity of each peer in BAKE is $d+2$, implying that, as long as $d+2$ neighbors and/or links do not fail simultaneously, a message can be delivered to another available peer successfully. In words, when d is a modest value, any message can reach its destination with high probability even if peers fail randomly, as shown in Fig.4. BAKE makes sure that a large majority of messages can be routed successfully when number of d even $2d$ peers fail randomly, and outperforms existing schemes.

C. Delay and message cost of basic operations

We define $\alpha = \lceil n/(d^{k_l-1} + d^{k_l-2} \rceil$ for later analysis. For a message to lookup or publish a resource with identifier x=$x_k...x_2x_1$, the routing delay is at most $t\,(k_l - 1) + k_l$ hops before it reaches an available host y=$y_k...y_2y_1$, where t denotes the anti-clockwise distance from node $x_{k-1}...x_2x_1$ to $y_{k-1}...y_2y_1$ in the given Kautz ring. In a static system, delay of all queries is k_l because peer y is always the preferred host. In a moderately dynamic system, delay of a majority of queries is k_l because peer y usually is the preferred or second host. In a highly dynamic

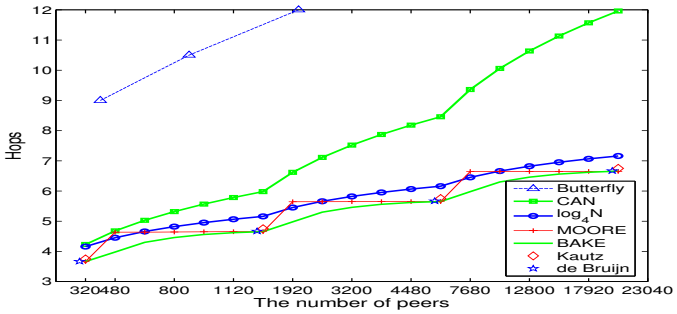


Fig. 3. The average routing distance under different configurations.

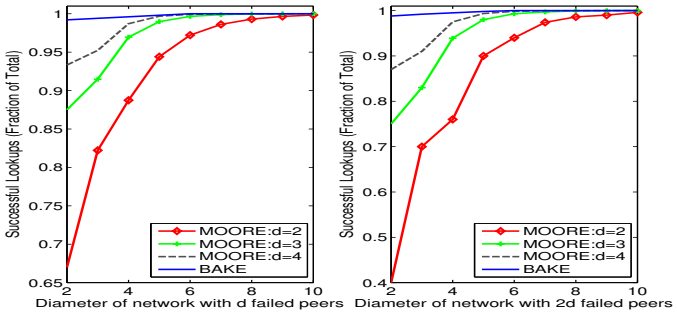


Fig. 4. The fraction of successful lookups as a function of the fraction of peers that fail concurrently in MOORE under different scales.

system, delay of lots of queries is $t\,(k_l-1)+k_l$ because peer y is often a third host peer, where $0 \le t$ but t is not a large value.

Theorem 5: In a static or moderately dynamic environment, delay and message cost of a peer x joining is at most $2k_l+\alpha+1$.

Proof: To notify peer $\sigma_1^1(x_k...x_3x_2)$ and its successor (or predecessor) will traverse at most k_l peers along a routing path. To inform its predecessor (or successor) and other in-neighbor peers will cause α messages. Peer x also visits the first entry point of BAKE and cause an additional message. In summary, a new peer will visit c=$2k_l+\alpha+1$ peers, and Theorem 5 holds. ■

Corollary 1: In a highly dynamic environment, peer x might invoke Algorithm 7. The delay of a peer x joining is at most $3k_l+\alpha+2$ hops. The whole process causes at most $3\times(k_l+\alpha)$ messages.

Proof: The process to find all out-neighbors for peer x traverses at most k_l-1 hops to reach the peer that is one hop away from an original destination. The peer then forwards the message to at most α out-neighbors. These peers forward the message to the destination in one hop. Theorem 5 has discussed the cost of other operations. In summary, a new peer causes $c_1=c+k_l-1+2\alpha$ messages. It is clear that $c_1<3\times(k_l+\alpha)$. The delay is $c+k_l+1=3k_l+\alpha+2$ hops. Thus, Corollary 1 holds. ■

Theorem 6: In a static or moderately dynamic environment, the delay and message cost of handling a failed peer x is at most $2k_l+\alpha$.

Proof: In this case, the predecessor or successor of peer x notifies each other in one hop. To notify the first entry point of BAKE and one in-neighbor of peer x can be finished within at most k_l hops, respectively. It takes α-1 hops to notify at most α-1 other in-neighbors of peer x. In summary, the delay is $2k_l+\alpha$, and causes at most $2k_l+\alpha$ messages. Thus, Theorem 6 holds. ■

Corollary 2: In a highly dynamic environment, the worst delay and message cost of handling a failed peer is $(\alpha+1)\times k_l+\alpha$. In general case, the delay and message cost is $3k_l+\alpha$.

Proof: In this situation, an in-neighbor u of a failed peer x needs to find a substitute of x. In the worst case, peer u detects α peers at the cost of at most $\alpha\times k_l$ messages and $\alpha\times k_l$ hops. In general case, peer u just detects its left and right adjacent peers at the cost of $2k_l$ messages and hops. On the other hand, peer u also notifies the first entry point of BAKE at the cost of at most k_l hops and messages, and notifies other in-neighbors of peer x at the cost of α hops and messages. Thus, Corollary 2 holds. ■

Theorem 7: The delay and message cost of handling a leaving peer x is at most $2k_l+\alpha+2$.

Proof: Peer x notifies its predecessor or successor in one hop, and at most α in-neighbors in one hops. Peer x also notifies its departure to the first entry point of BAKE in at most k_l hops. Peer x also retrieves a substitute from the first entry point at the cost of k_1 hops when it is the only existing peer with a suffix $x_{k-1}...x_2x_1$. Thus, the delay and message cost is $2k_l+\alpha+2$. ■

VI. Conclusion

We propose a balanced Kautz tree structure, based on which we design BAKE, a practical P2P architecture. In BAKE, the locality-preserving naming and placement strategy of resources are employed to support both exact and range queries. The topology management and routing schemes guarantee flexible and efficient resource distribution. BAKE achieves optimal diameter, high performance, and good connectivity for dynamic P2P networks.

References

[1] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proc. ACM SIGCOMM*, pages 161–172, 2001.

[2] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *IEEE/ACM Trans. Networking*, 11(1):17–32, 2003.

[3] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[4] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. Tapestry: a fault-tolerant wide-area application infrastructure. *Computer Communication Review*, 32(1):81, 2002.

[5] M. T. Schlosser, M. Sintek, S. Decker, and W. Nejdl. Hypercup - hypercubes, ontologies, and efficient search on peer-to-peer networks. In *Proc. the 1th International Workshop on Agents and Peer-to-Peer Computing*, pages 112–124, Bologna, Italy, July 2002.

[6] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proc. International Peer-to-Peer Symposium*, pages 53–65, Cambridge, MA, USA, March 2002.

[7] 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 4th USENIX Symposium on Internet Technologies and Systems*, Washington, USA.

[8] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proc. the 21st ACM PODC*, pages 183–192, Monterey,CA, August 2002.

[9] J. Xu, A. Kumar, and X. X. Yu. On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks. *IEEE J. Select. Areas Commun.*, 22(1):151–163, 2004.

[10] H. Shen, C. Xu, and G. Chen. Cycloid: A constant-degree and lookup-efficient p2p overlay network. In *Proc. the 18th International Parallel and Distributed Processing Symposium*, Santa Fe, New Mexico, USA, April 2004.

[11] S. Banerjee and D. Sarkar. hypercube connected rings: a scalable and fault-tolerant logical topology for optical networks. *Computer communications*, 24:1060–1079, 2001.

[12] F. Kaashoek and D. Karger. Koorde: A simple degreeoptimal distributed hash table. In *Proc. International Peer-to-Peer Symposium*, pages 98–107, Berkeley,CA,USA, February 2003.

[13] M. Naor and U. Wieder. Novel architecture for p2p applications: the continuous-discrete approach. In *Proc. ACM Symposiumon Parallel Algorithms and Architectures*, pages 50–59, San Diego, California, USA, June 2003.

[14] P. Fraigniaud and P. Gauron. D2B: a de bruijn based content-addressable network. *Theor. Comput. Sci.*, 355(1):65–79, 2006.

[15] P. Fraigniaud and P. Gauron. An overview of the content-addressable network d2b. In *Proc. the 22st ACM PODC*, page 151, Boston,USA, July 2003.

[16] D. Loguinov, J. Casas, and X. Wang. Graph-theoretic analysis of structured peer-to-peer systems: Routing distances and fault resilience. *IEEE/ACM Trans. Networking*, 13(5):1107–1120, October 2005.

[17] A. T. Gai and L. Viennot. Broose: a practical distributed hashtable based on the de bruijn topology. In *Proc. the International Conference on Peer-to-Peer Computing*, pages 167–174, Switzerland, August 2004.

[18] D. Li, X.Lu, and J. Su. Graph-theoretic analysis of kautz topology and dht schemes. In *Proc. IFIP International Conference on Network and Parallel Computing*, pages 308–315, Wuhan, China, oct 2004.

[19] D. Li, X. Lu, and J. Wu. Fissione: A scalable constant degree and low congestion dht scheme based on kautz graphs. In *Proc. IEEE INFOCOM*, pages 1677–1688, Miami,Florida,USA, March 2005.

[20] M. Ripeanu, A. Iamnitchi, and I. T. Foster. Mapping the gnutella network. *IEEE Internet Computing*, 6(1):50–57, 2002.

[21] M. Miller and J.Siran. Moore graphs and beyond: A survey of the degree/diameter problem. *Electronic Journal of Combinatorics*, 61:1–63, December 2005.

[22] D. Guo, Y. Liu, and X. Ki. Bake: A balanced kautz tree structure for peer-to-peer networks. In *Proc. 27th IEEE INFOCOM*, April 2008.

[23] N. Alon, S. Hoory, and N. Linial. The moore bound for irregular graphs. *Graphs and Combinatorics*, 18(1):53–57, 2002.

[24] R.M. Damerell. On moore graphs. In *Proc. Cambridge Phil. Soc.*, pages 227–236, 1973.

[25] P. Tvrdik. Partial kautz line digraphs with maximal connectivity. Technical Report Research Report 94-15, LIP ENSL, 69364 Lyon, France, April 1994.

[26] M. Imase and M. Itoh. Design to minimize diameter on building-block network. *IEEE Trans. Computers*, 30(6):439–442, June 1981.

[27] M. Imase and M. Itoh. A design for directed graphs with minimize diameter. *IEEE Trans. Computers*, 32(8):782–784, August 1983.

[28] G. Panchapakesan and A. Sengupta. On a lightwave networks topology using kautz digraphs. *IEEE Computer*, 48(10):1131–1138, 1999.

[29] D. Guo, J. Wu, H. Chen, and X. Luo. Moore: An extendable peer-to-peer network based on incomplete kautz digraph with constant degree. In *Proc. 26th IEEE INFOCOM*, page 821, May 2007.

[30] Y. Zhang, D. Li, and X. Lu. Building constant-degree peer-to-peer networks based on distributed line graphs. http://kylinx.com/Papers/, September 2007.

[31] M. A. Fiol and A. S. Llado. The partial line digraph technique in the design of large interconnection networks. *IEEE Trans. Computers*, 41(7):848–857, July 1992.