Beyond the Lower Bound: A Unified and Optimal P2P Construction Method

Ye Yuan[1,2], Deke Guo[2], Guoren Wang[1], Yunhao Liu[2]
[1]Northeastern University, Shenyang 110004, China
[2]Hong Kong University of Science and Technology, Hong Kong SAR, China

Abstract—**The topological properties of overlay networks are critical for the performance of peer-to-peer (P2P) systems. The size of routing table and the diameter are among the most important parameters which measure the autonomy, efficiency, robustness and load balancing of P2Ps, and the Moore bound sets the optimal tradeoff between diameter and degree for any graph. In order to improve the four features for structured P2P networks, researchers have proposed many designs based on the interconnect networks which can approximately achieve the Moore bound. However, the performances of these systems in dynamic environments are far worse than their corresponding interconnect networks in the static environment. Moore bound cannot give a good description for existing P2P systems due to their dynamic features. In this study, therefore, we first prove a new lower bound of the network diameter and average query distance in a highly dynamic environment. The routing latency of the existing systems cannot be more better than the new bound. This is because the existing systems require a fixed number of peers known a priori. P2P systems on the other hand, typically are dynamic, meaning that peers frequently join or leave. To solve the issue, we propose a dynamic multi-way trie tree structure, based on which any interconnect network can be adopted to construct a P2P system. The unified construction method also consists of a series of optimal schemes for P2P systems. DeBruijn and butterfly representing the single and hybrid protocols based on the dynamic multi-way trie tree structure are used to construct two new P2P systems (DPhoenix, BPhoenix) whose performances can exceed the new bound. Also, DPhoenix and BPhoenix can be easily applied to other interconnection networks after minimal modifications, such as, hypercube, kautz, shuffle-exchange and CCC.**

I. Introduction

Following Napster and Gnutella-like networks [5], structured P2P networks based on distributed hash tables (DHTs) [1]–[3] have emerged. After that, more innovations [6]–[11] have been introduced for better scalability, accuracy and efficiency.

Two important properties give an indication of the quality and efficiency of structured P2P networks: (1) peer degree, the size of routing table, and (2) network diameter, the number of hops a lookup needs to travel in the worst case. In traditional structured P2P designs, the peer degree and network diameter increase logarithmically with respect to the size n of the network, such as Chord [3], Pastry [1] and Tapestry [2] based on the hypercube topology. Such schemes publish and lookup resources within $\log n$ hops, while they often introduce huge maintenance overhead and suffer from poor scalability.

To address the problems, several architectures based on constant degree interconnection networks have been proposed. For example, the Viceroy [9] and Ulysses [11] (based on the butterfly topology), Cycloid [13] (based on the CCC topology) [14], CAN [1] (based on the d-dimensional torus topology), Koorde [7], Distance Halving [10], D2B [6] and ODRI [8] (based on the deBruijn topology), and FissionE [15] and Moore [19] (based on the kautz topology). In the above designs, the network diameter increases logarithmically with respect to the size of a P2P system.

The Moore bound sets the optimal tradeoff between diameter and degree for any static graph [20]. The Moore bound, however, cannot give a good description for P2P systems due to their dynamic features. The diameters of kautz and deBruijn graphs mostly achieve the Moore bound. However, the diameters of the systems based on them, like D2B, Koorde and FissionE, are much larger than the Moore bound. Other systems like Pastry, Viceroy and Cycloid have far worse performance than their corresponding interconnect networks. The fundamental reason is that most existing P2P systems cannot adapt to the dynamic environments. One critical requirement of these systems is that the number of peers must be some given values determined by the peer degree and the network diameter. Hence, the corresponding approaches are often impractical when peers frequently join, leave, and fail. Several recent designs like BAKE [21], support arbitrary number of nodes at the cost of huge overhead, which has an impact on the routing efficiency. In this work, we propose the dynamic multi-way trie tree structure to support P2P systems with arbitrary number of peers. The construction method has a unified feature, since any kind of static network can use the trie tree to design a dynamic network. The design objective is to inherit properties of static networks in the dynamic environment. A series of optimal schemes is given for P2P systems so as to maximize the query performance while minimizing the maintenance cost.

The main contributions of this paper are as follows:

(1) We prove a probabilistic lower bound of the network diameter and average query distance for existing P2P systems in a highly dynamic environment.

(2) We propose a dynamic multi-way trie structure and a series of optimal construction methods. Based on these schemes, any kind of static interconnection network can be adopted to construct an overlay topology for a dynamic P2P system.

(3) Based on the proposed methods, we use deBruijn and butterfly graph to design DPhoenix and BPhoenix: two

effective and robust P2P systems which retain desirable properties of their corresponding static networks, such as optimal diameter and constant degree. The designs are able to deal with the dynamic peer operations, such as join, depart, fail, and topology changes, and their routing efficiencies can exceed the lower bound.

(4) We evaluate the properties of DPhoenix and BPhoenix, the robustness of the routing scheme, the delay and message costs of basic operations and load balance through formal analysis and comprehensive simulations, and compare it with previous P2P designs.

The rest of the paper is organized as follows. The search delay is analyzed for the dynamic network, and two lower bounds are proved for the network diameter and average query distance, respectively in Section III. A universal method is proposed to address the issue of the static network in Section IV. The dynamic operations of P2P systems based on the dynamic trie tree structure are proposed in Section V. Finally, the performance of the two proposed systems are evaluated in Section VI and we conclude the work in Section VII.

II. RELATED WORK

The Moore bound [20] is the lower bound for diameter of any graph, and the value is given as follows:

$$d \geq \lceil \log_k (n(k-1)+1) \rceil - 1 \qquad (1)$$

A deBruijn graph [6] with fixed node degree d and diameter k consists of d^k nodes. There is a directed edge from node $u = (x_1 x_2, ..., x_k)$ to node $v = (x_2 x_3, ..., x_{k+1})$. Butterfly [9] is a directed graph with kd^k nodes, where k and d are referred to as the diameter and the degree, respectively. From each node $(x_0 x_1, ..., x_{k-1}; i)$, there is a directed edge to all nodes of the form $(x_0, ..., x_i, y, x_{i+2}, ..., x_{k-1}; i+1)$ when $i \neq k-1$ and $(y, x_1, ..., x_{k-1}; 0)$ when $i = k-1$.

Systems based on deBruijn and butterfly like D2B [6], Koorde [7], Viceroy [9] and Ulysses [11] try to solve the problems of the dynamic environment. However, their schemes introduce new problems such as, serious distortion of the topology, huge maintenance overhead and imbalanced node degrees, which have an impact on the query efficiency. In the next section, we prove a query delay lower bound for these systems instead of Moore bound.

III. A LOWER BOUND OF QUERY DELAY FOR DYNAMIC P2P NETWORK

Most existing systems are static, and they cannot be adapted to the dynamic network. The negative effects due to possible dynamical behaviors need to be carefully considered for topology properties. In this section, we discuss the impact on the diameter and average query distance.

It is well known that Moore bound is a lower bound of diameter for any non-trivial graph. It is satisfied in the dynamic environments, but its value is too small for dynamic networks. Moore bound for static networks cannot give a well description for dynamic networks. We, therefore, need reconsider the lower bound.

In [16], the authors analyze the load balancing problem by using a binary trie tree model. Such a two way tree has different length strings, and our idea comes from this model, since the trie tree can be viewed as a routing tree which is used to emulate a dynamic network. Without loss of generality, we set up a $d-$way trie tree in our model.

For a static network with maximum out-degree d, any node cannot reach more $d + d^2 + ... + d^h$ nodes in at most h hops. Hence, in order to reach all $n-1$ nodes in at most h hops, the value of h should satisfy $d + d^2 + ... + d^h \geq n-1$. But in a dynamic environment, the number of nodes reached from a beginning node in i hops may not be d^i nodes, for $1 \leq i \leq h$. In other words, the routing tree may not be completed in each level.

To emulate a dynamic network, let us consider a case that nodes join and leave randomly from a trie tree. The joining process of a node is equivalent to that of a ball with a prefix x is dropped into the root of a virtual tree and then reaches a most matched node with x of d children as the next step. The leaf at which the ball ends up is the node's parent, and shares the longest common prefix with x. The process is similar with the most existing DHTs. Peer departure is an opposite process of peer joining.

Now we estimate a lower bound of the diameter for a dynamic network. Here, the diameter is defined as the largest depth of the tree.

Theorem 1: With probability $1 - o(1)$, the lower bound d_{max} of the diameter of a dynamic network concentrates on three values d_l, $d_l + 1$, and $d_l + 2$, where

$$d_l = \lfloor \log_d n + \sqrt{d \log_d n} - 1.5 \rfloor \qquad (2)$$

Proof: Since d_{max} is the maximum height of the tree, to compute d_{max}, we need to consider the leave process of a node. This process is constructed by dropping balls and getting rid of leaf, and it is very similar with the well know structure called the Patricia trie [17]. The tree is a collapsed version of the regular trie in which all intermediate nodes with a single child are removed. Recall that with probability $1 - o(1)$, the height of a random Patricia trie is concentrated on three values d_l, $d_l + 1$, and $d_l + 2$, where d_l is given by (2), we obtain the result from [17]. Q.E.D

The diameter of a graph is simply the largest distance between any pair of nodes and only provides an upper bound on the delay (number of hops) experienced by users. A much more crucial metric is the average distance between any pair of nodes. We will focus on a lower bound of this metric in a dynamic network.

Let h denote the minimum depth of the d way tree. The tree is divided by two parts. First part consists of all nodes which locate between the root and h level. Second part includes all nodes which locate between level $h+1$ to d_{max}. Naturally, the first part is a full d way tree with depth h. Let $d(root, x)$ (denoted by d_x) denote the distance from the root to any node x in the tree. We will discuss the distribution of d_x for the two parts separately.

To compute the distribution of the first part, we define a sequence of indicator random variables A_i, $i \geq 0$, where $A_i = 1$ if level i of the tree is full after users joined the system and $A_i = 0$ otherwise. We say that a level is full if all nodes of that level are present and nonleaf. Notice that level i can be full only if $i \leq h$ and that $A_i = 1$ implies that $A_k = 1$, for all $k < i$. It immediately follows d_x is at least $k+1$ if and only if all levels from 0 to k are full:

$$P(d_x \geq k+1) = P(\bigcap[A_i = 1]) \tag{3}$$

We then have the deduction of formula (3):

$$P(d_x \geq k+1) = P(d_x \geq k)P(A_k|A_{k-1}) \tag{4}$$

where $P(d_x \geq 0) = 1$ and $P(A_k|A_{k-1})$ is the conditional probability of level k being full given that all previous levels $0, ..., k-1$ are full:

$$P(A_k|A_{k-1}) = P(A_k = 1|h \geq d_x) \tag{5}$$

For the first part, we can draw out the following theorem.

Theorem 2: With probability $1 - o(1)$, the distribution of d_x is $\exp\{-d^k \exp\{\frac{n(d-1)+1}{d^{k+1}(d-1)} - \frac{1}{d-1}\}\}$.

Proof: First notice that the first part is a d way tree with $h{+}1$ level. The d way tree built using peers contains $\frac{n(d-1)+1}{d}$ leaves and nonleaf $\frac{n(d-1)+1-d}{d(d-1)}$ nodes. Next, examine level k of the tree and observe that all d^k possible nodes at this level must be non-leaf for level k to be fully joined. Assuming that all previous levels are full, exactly $\frac{d^{k-1}}{d-1}$ non-leaf nodes contributed to filling up levels $0, ..., k-1$ and the remaining $\frac{n(d-1)+1-d}{d(d-1)} - \frac{d^{k-1}}{d-1} = \frac{n(d+1)+1-d^{k+1}}{d(d-1)}$ non-leaf nodes had a chance to be joined. After the first levels $k-1$ have been filled up, each node at level k is hit by an incoming ball with an equal probability d^{-k}. Thus, our problem reduces to finding the probability that $u = \frac{n(d+1)+1-d^{k+1}}{d(d-1)}$ uniformly and randomly placed balls into $m = d^k$ bins manage to occupy each and every bin with at least one ball. There are many ways to solve this problem, one of which involves the application of well-known results from the coupon collector's problem [18]. We use this approach below. Define $Z(u)$ to be the random number of non-empty bins after balls u are thrown into m bins. Thus, we can write $P(A_k|A_{k-1}) = P(Z(u) = m)$. Recall that in the coupon collector's problem, u coupons are drawn uniformly randomly from a total of m different coupons. Then, the probability $Z(u) = m$ to obtain m distinct coupons at the end of the experiment is given by [18].

$$P(Z(u) = m) = \sum(-1_j)(m/j)(1 - \frac{j}{m})^u \tag{6}$$

For large u, the term $(1 - j/m)^u$ can be approximated by $e^{-uj/m}$, yielding

$$P(Z(u) = m) \approx \sum(-1_j)(m/j)e^{-uj/m} = (1 - e^{-u/m}) \tag{7}$$

Since we are only interested in asymptotically large $m = O(log_d n)$, (7) allows a further approximation

$$\begin{aligned} P(Z(u) = m) &\approx e^{-me^{-u/m}} \\ &= \exp\{-d^k \exp\{\tfrac{n(d-1)+1}{d^{k+1}(d-1)} - \tfrac{1}{d-1}\}\} \end{aligned} \tag{8}$$

From (6)-(8), we get the distribution of d_x:

$$\begin{aligned} P(d_x \geq k+1) &= P(d_x \geq k)P(A_k|A_{k-1}) \\ &\approx \exp\{-d^k \exp\{\tfrac{n(d-1)+1}{d^{k+1}(d-1)} - \tfrac{1}{d-1}\}\} \end{aligned} \tag{9}$$

$Q.E.D$

From (9), we know that

$$\begin{aligned} h &= max(d_x) \\ &= \log_d n - \log_d((1+\varepsilon)\log n - O(\log\log n)) \end{aligned} \tag{10}$$

with probability at least $1 - n^{-\varepsilon}, \varepsilon \leq 1$.

For the second part of the tree, we have the following theorem from [17].

Theorem 3: With probability $1 - o(1)$, the distribution of d_x of PATRICIA tries is:

$$d_x \sim \sqrt{1 + d\xi\Phi'(\xi) + \xi^d\Phi''(\xi)} e^{-n\Phi(\xi)} \tag{11}$$

where $\xi = nd^{-k}$, $0 < \xi < 1$, and $\Phi(\xi) \sim \frac{1}{d}\rho_0 e^{\varphi(log_d \xi)}\xi^{3/2}\exp(-\frac{\log^d \xi}{d \log d})$.

From the distribution of d_x for two parts of the tree, we can get the lower bound of average distance for a dynamic network. We have the following theorem.

Theorem 4: The lower bound of average distance for a dynamic network is $log_d n + \sqrt{\log_d n/d}$ with probability of at least $1 - n^{-\varepsilon}, \varepsilon \leq 1$.

Proof: From the above, we get the average distance: $d_{avg} = \sum_{k=0}^{h-1} k \cdot (\exp\{-d^k \exp\{\frac{n(d-1)+1}{d^{k+1}(d-1)} - \frac{1}{d-1}\}\}) + \sum_{k=h}^{d_{max}-1} k \cdot (\sqrt{1 + d\xi\Phi'(\xi) + \xi^d\Phi''(\xi)} e^{-n\Phi(\xi)}) \approx log_d n + \sqrt{\frac{\log_d n}{d}}$ Q.E.D

Now we have obtained the lower bounds of the diameter and average distance for a dynamic network. The search delay of any existing P2P system cannot be smaller than the lower bound.

IV. A UNIFIED CONSTRUCTION METHOD FOR DYNAMIC P2P NETWORKS

Any interconnect network has desirable properties graph only if all nodes exist and are stable. Such as deBruijn and butterfly have d^k and $d^k k$ nodes respectively. This requirement, however, is impractical in the dynamic scenario. To address this issue, we propose a dynamic multi-way trie tree structure to achieve the desired topology.

A. Dynamic Multi-way Trie Tree Structure

Definition 1: A dynamic d way trie tree with depth k is a rooted tree. Each node has at most d child nodes. Each edge at the same level is assigned a unique label. Each 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 $x_1^i = i$ for $0 \leq i \leq d-1$. The ith child of root node is labeled as $x_1^i = i$, and is arranged from left to right. The root node does not contain any string.

2. The edge from a node x_1 to its ith child is labeled as $x_2^i = i$ for $0 \leq i \leq d-1$. The ith child is labeled as $x_1 x_2^i$, and is arranged from left to right.

3. The edge from a node $x_1 x_2 ... x_{k-1}$ to its child is labeled as x_k^i. The child of x_k^i is labeled as $x_1 x_2 ... x_{k-1} x_k$, and is arranged from left to right.

4. There is a bidirectional edge between any node and its parent, and all the nodes at the same level form a ring.

From the definition, we know the trie tree might be an imbalanced tree which reflects the features of an dynamic scenario. The levels of all leaf nodes for a balanced tree are at the same level.

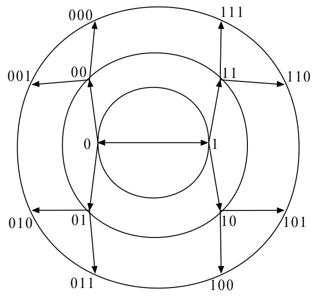


Fig. 1: An example of trie tree

The tree shown in Fig. 1 is a complete a trie tree. If the leaf node 011 fails, the tree becomes an incomplete trie tree. If the node 010 and 011 both fails, the tree becomes a unbalanced tree.

For any level, the left-to-right traversal of nodes at that level can form a total ordering, denoted as the trie ordering. We sort all children of root node in ascending order within level 1, and then rank all the child nodes of each node at level 1 respectively. Note that the level 2 nodes inherit the order of nodes in level 1. By sorting nodes from level 1 to k recursively, we can find a trie ordering of nodes at each level. From the trie tree structure, we know each level forms a ring and we call it the trie ring. We arrange the trie ordering in each ring along the anti-clockwise direction. In the ring, each node connects its predecessor and successor.

Definition 2: For any node x, its predecessor is the first existing node clockwise from it in a trie ring of existing nodes at the same level, and its successor is the first existing node anti-clockwise from it in the same trie ring. The concept of left adjacent node is similar to the predecessor, but the trie ring is consisted of all possible nodes not just existing nodes. So do the right adjacent node and successor node.

In a complete trie tree, the left adjacent node and predecessor are the same nodes. In an unbalanced tree, however, this might not be the case. Therefore we should establish the successor and predecessor of a node for both the balanced and unbalanced tree. In Fig. 1, the node 010 is the predecessor of the node 011 and its successor is the node 100. In the unbalanced tree, (the node 010 and 011 are not alive) the predecessor of the node 100 is the node 001 and 001's successor is 100.

Algorithm 1 can establish the successor of a node for both a balanced and an unbalanced tree. For the algorithm, $x = x_1 x_2 ... x_k$ is a peer in the trie tree. The symbol x represents the same meaning in other algorithms.

Algorithm 1 Successor(x)

```
1: $i \leftarrow k$
2: do{
3: $x_1x_2...x_i$ forwards the request to $x_1x_2...x_{i-1}$ using its up link.
4: $i \leftarrow i-1$
5: }while(the up link to $x_1x_2...x_i$ is its last down link and
   $x_1x_2...x_i$ has not fixed its successor)
6: $j \leftarrow i$
7: if ($x_1x_2...x_j$ has fixed its successor) then
8:     do{
9:     $x_1x_2...x_j$ forwards the message including the identifier of its
       successor $x_1x_2...x'_j$ to $x_1x_2...x_{j-1}$.
10:    if ($x_1x_2...x'_j$ has a child) then
11:        $x_1x_2...x_{j-1}$ initiates a request to $x_1x_2...x'_j$ to fetch its first
           child as its successor.
12:    else
13:        return false.
14:    end if
15:    $j \leftarrow j-1$
16:    }while($j > k$)
17: else
18:    $x_1x_2...x_j$ returns its first child as the successor of
       $x_1x_2...x_{j-1}$.
19:    $m \leftarrow j-1$
20:    do{
21:    $x_1x_2...x_m$ forwards the message including the identifier of its
       successor $x_1x_2...x'_m$ to $x_1x_2...x_{m-1}$.
22:    if ($x_1x_2...x'_m$ has a child) then
23:        $x_1x_2...x_{m-1}$ initiates a request to $x_1x_2...x'_m$ to fetch its
           first child as the its successor.
24:    else
25:        return false.
26:    end if
27:    $m \leftarrow m-1$
28:    }while($m > k$)
29: end if
```

For a complete tree, it's easy for a node to establish its successor based only on its identifier. Also, a node of a balanced tree can use Algorithm 1, and the leaf node at most forwards the request to its grandparent. In dynamic scenarios, unfortunately, the tree is often unbalanced, and we should use Algorithm 1.

For a unbalanced tree, Algorithm 1 consists of two steps. In the first step, x forwards the request along the up link. When an ancestor y has established its accessor, the request stops. When an ancestor z has at least two children and the requesting link is not the last child, its immediate right sibling is the successor of the requesting child. At this time, the request stops. In the second step, the ancestor returns the information obtained in the first step back to its requesting child, and its child can establish its successor according to the information. At last, all the ancestors from x to y or z establish their successors.

Theorem 5: With probability $1-o(1)$, the cost of establishing a successor for a node in a unbalanced trie tree is at most $2\lfloor\sqrt{d\log_d n}+0.5+\log_d((1+\varepsilon)\log n-O(\log\log n))\rfloor$.

Proof: Suppose the deepest leaf wants to establish its successor. From (2), we know $h_{max}=\lfloor\log_d n+\sqrt{d\log_d n}-1.5\rfloor$ $w.h.p$. From the above analysis of Algorithm 1, we know in the first step, the request can reach at most the grandparent of the smallest depth leaf node. From (10), we have $h_{min}=\log_d n-\log_d((1+\varepsilon)\log n-O(\log\log n))$ $w.h.p.$ Therefore, we got the cost for the first step: $cost=\lfloor 2-10+2\rfloor$. The cost of second step is almost same. Therefore the total cost is:

$cost=2\lfloor 2-10+2\rfloor=2\lfloor\sqrt{d\log_d n}+0.5+\log_d((1+\varepsilon)\log n-O(\log\log n))\rfloor$ $Q.E.D$

Algorithm 1 also establishes the predecessors of many nodes. Due to the fact, the cost of establishing a successor and a predecessor for each new node is low. Using the same principle, we can establish the predecessor of a node.

B. Mapping Interconnect Network to Dynamic Trie Tree

In this section, we will propose optimal structuring strategies to organize peers and resources in an efficient overlay network that can guarantee all the desirable properties of static interconnect networks. The structuring strategies are based on the concept of $d-$way dynamic trie tree with depth k mentioned as above.

1) Optimal Topology Construction Rule: To address the issues of dynamic networks, a universal rule is proposed to map any static network to the trie tree.

For a node $x=x_1x_2...x_k$ of an interconnect network, $\sigma^i(x), 1\le i\le d$, denotes a neighbor node of x. The label will be used throughput the paper. Any d-ary interconnect network is mapped to the $d-$way trie tree as follows:

(1) If a peer $\sigma^i(x)$ has appeared in the overlay network, it is the ith neighbor of peer x;

(2) Otherwise, if $\sigma^i(x)$ and its predecessor y have a common prefix with length $k-1$, peer y is the ith neighbor of peer x and y stores another identifier $\sigma^i(x)$.

(3) Otherwise, if $\sigma^i(x)$ and its successor z have a common prefix with length $k-1$, peer z is the ith neighbor of peer x and y stores another identifier $\sigma^i(x)$.

(4) Otherwise, the youngest living ancestor of $\sigma^i(x)$ is the ith neighbor of peer x and the ancestor stores another identifier $\sigma^i(x)$.

The mapping rule can guarantee that peer x of system based on any interconnect network must connect to its neighbor peer $\sigma^i(x)$.

From the mapping rule, we can draw the following conclusion.

Theorem 6: For an inner peer x and leaf peer y of system based on a interconnect network, the former rules can make the following conclusion:

(1) For a balanced tree, x has $2d+3$ neighbors and y has $d+3$ neighbors.

(2) For an unbalanced tree, x has at least $2d+1$ neighbors and y has at least $d+1$ neighbors.

Proof: In an unbalanced tree, an inner peer x always has d $\sigma^i(x)$ neighbors according to the rule. If x's predecessor and successor is too far away from x, Algorithm 1 cannot guarantee x can fix them. x also has d children and one parent. Therefore the number of x's total neighbors is at least $2d+1$. For a leaf peer, it dose not have any child. Therefore, it has at least $d+1$ neighbors. For a balanced tree, since all leaf peers are at the same level, an peer can establish its any neighbor. So, we get the conclusion for a balanced tree. $Q.E.D$

Practically, d is often a large value for the sake of decreasing search delay and improving its connectivity at the cost of higher maintenance overhead. Therefore, d is an important parameter for P2P systems, and we should choose a proper value for d. How large d to choose is a trade-off issue.

Since query operations and update operations are dominant operations in P2P systems, a cost model (C) based on them can be built as follows.

Let λ be the percentage of query operations, and $1-\lambda$ be the percentage of update operations in the system. Then $C=\lambda A+(1-\lambda)B$, $0\le\lambda\le 1$, where A and B denote the query and update cost respectively. Since the approximate costs of a query operation and an update operation are respectively $log_d n$ and d, we have,

$$C(d)=\lambda log_d n+(1-\lambda)d \tag{12}$$

Different Equation (12), we have,

$$(\ln d)^2 d=\frac{\lambda\ln n}{1-\lambda} \tag{13}$$

It is easy to find a numerical solution for Equation (13) by numerical methods such as Newton's method [12]. Upon finding d_0 satisfying the equation, we calculate $C(d_1)$, $d_1=\lfloor d_0\rfloor$ and $C(d_2)$, $d_2=\lceil d_0\rceil$, and select the value corresponding to the smallest cost.

To determine d, we should know the number of peers in the system. The number n can be approximated from the identifer of a leaf peer. Though n is not stable due to frequent joining and leaving of peers, the mapping rule still guarantees that the determined value of d is in effect for the performance of P2P systems.

2) Optimal Resource Placement Rule: The resource placement is very important for P2P systems. Normal resource placement polices may not accurately and efficiently organize resources and support resource queries. The following scheme can assure that resources are distributed based on the longest prefix matching policy. In addition, the scheme is fault-tolerant.

Suppose the identifier of a resource is x whose length is longer than a node identifier. The identifier x is acquired with a hashing algorithm by each peer locally.

Algorithm 2 Placement(x)

```
1: if peer $x' = x_1x_2...x_k$ has appeared in the overlay network
2:     The peer $x_1x_2...x_k$ stores the resource $x$.
3: else if its predecessor $y$ and $x'$ have common parent node in the tree
4:     peer $y$ is the host peer of the resource $x$.
5: else if its successor $z$ and $x'$ have common parent node in the tree
6:     peer $z$ is the host peer of the resource $x$.
7: else the youngest living ancestor of $x'$ is the host peer of resource $x$
```

From the mapping and placement rule, we can see that the mapping from a node of the trie tree to a peer of the system is one to one. All the resources are kept in the leaf peers, and the inner nodes become routing peers. Our strategy is different from the previous works which map some nodes or branches of a virtual tree structure to a peer of an overlay. The scheme leads to additional overhead and destroys the features of interconnect network, such as the constant degree of the deBruijn graph. In addition, the number of leaf peers is almost $d-1$ times than the total number of inner peers. Therefore, the leaf peers are enough to store all the resources.

As shown in Fig. 1, resource 100... is stored by peer 100, and is taken over by peer 101 when 100 is not alive. If peer 100 and 101 are both not alive, peer 10 will keep the resource.

The peers of traditional systems usually publish all their resources to the network, which results in large maintenance overhead. In addition, the resource placement strategy brings additional load to some peers which take over the resources of their failed brother or children peers. We should reduce the number of the resources kept in a peer. To solve this dilemma we introduce a model.

Let G denote the number of resources a peer P_i should keep. Let a_j denote the size (in bytes) of the jth resource. Specifically, we suppose that the request probability for each resource j is a known value, q_j, with $q_1 + q_2 + \cdots + q_G = 1$. Let y_j be a zero-one variable which is equal to one if P_i keeps the resource j and is zero otherwise. The probability of a hit to P_i for an arbitrary query is $P_{query} = \sum_{j=1}^{G} q_j y_j$.

The maintenance cost of a resource is mainly composed of two parts, namely the: refresh cost and the update cost. Refresh messages are used to guarantee the data are consistent and to prevent data loss from network corruption. On average, the refresh messages are sent to peer P_i responsible for the resources every T seconds. The approximate cost of refresh operations is $C_{refresh} = \frac{1}{T} \log_d n \sum_{j=1}^{G} y_j$. Suppose f resources are inserted (deleted) to (from) P_i per second. The update operation will incur a cost of $C_{update} = f \log_d n$. Therefore the approximate total cost of maintaining G resources for peer P_i is

$$C_{total} = \frac{1}{T} \log_d n \sum_{j=1}^{G} y_j + f \log_d n \tag{14}$$

For peer P_i, we want to maximize the probability of successful queries, P_{query}, and minimize the total cost of maintenance. The kept resources must satisfy the local storage constraints. Therefore the variable y_j of the two objective functions satisfies the condition:

$$\sum_{j}^{G} a_j y_j \leq S \tag{15}$$

where S is the maximum load P_i can assume.

This is an issue of the multi-objective function optimization, and the objectives of two function are the opposite. we can set up an objective function to solve the issue. But the quantities class of the values of the two functions must be the same. After analyzing the actual issue, we can construct one objective function as follows:

$$C_{virtual} = \lambda P_{query} \log_d n - (1-\lambda) C_{total} \tag{16}$$

We can decide which resource is kept by maximizing the values of the function. The parameter λ represents the percentage of the query operations for the resources, and $1-\lambda$ is the percentage of the maintenance operations.

From Equation 16, we have,

$C_{virtual} = \lambda \log_d n \sum_{j=1}^{G} q_j y_j - (1-\lambda) \left(\frac{1}{T} \log_d n \sum_{j=1}^{G} y_j + f \log_d n\right) = \log_d n (\sum_{j=1}^{G} y_j (\lambda q_j - \frac{1-\lambda}{T}) - (1-\lambda) f)$

Therefore, we can solve the optimization issue using dynamic programming [4]. The standard dynamic programming formula for the function is given:

$C_{1 \leq j \leq G, 0 < i \leq S}(j, i) = max\{C(j-1, i-a_j) + \lambda q_j - \frac{1-\lambda}{T}, C(j-1, i)\}$

A recursive programm is used to compute the dynamic programming formula. We can record the value of y_j while executing the programm. We omit the detail process due to the limited space. The results will be shown in the experimental part.

Any peer in the system performs the optimal resource placement strategy locally, which dose not cause additional network overhead.

C. DPhoenix and BPhoenix

In this subsection, deBruijn and butterfly representing the single protocol (like kautz, hypercube and shuffle exchange) and hybrid protocol (like ccc) based on the trie tree and above proposed schemes are used to construct two new P2P systems (DPhoenix, BPhoenix) whose performances can exceed the lower bound.

The mapping rule of the static network is the most important step for constructing a P2P system. For a deBruijn graph, $\sigma^i(x) = x_2...x_k x_{k+1}^i$, the mapping rule can be directly applied. In the case of butterfly, $\sigma^i(x) = (x_0, x_1, \cdots, x_i, y, x_{i+2}, \cdots, x_{k-1}; i+1)$, there are several $d-$way trie trees. For more detail, $x = (x_0 x_1 \cdots x_{k-1}; i)$ in the ith trie tree connects to the peer $\sigma^i(x)$ in the $i+1$th tree, and the mapping rule for $i+1$th tree can be used. The $i+1$th trie tree might be empty in a highly dynamic scenario. In such situation, peer x becomes the root of the $i+1$th tree.

Using an example for deBruijn graph, in Fig. 1, peer 010 connects to peer 100 and 101. If peer 100 fails, peer 101 replaces 100. As a result, peer 010 connects to 101 again and peer 101 stores the identifier 100. If peer 100 and 101 both fail, peer 010 connects to their parent 10 and peer 10 stores the identifier 100 and 101.

The parameter d for degrees of DPhoenix and BPhoenix is important to their performances. The scheme that choosing an optimal d can be applied to the two systems without any modification.

DPhoenix and BPhoenix can directly adopt the optimal resource placement rule to enhance the performances of their data managements.

From the examples of DPhoenix and BPhoenix, we can believe that the construction methods have an unified feature.

V. Dynamic Operations of P2P System

A P2P system is a highly dynamic network. Peers may join, leave and fail frequently. These operations have an impact on the topology and routing efficiency of P2P systems. The robust schemes to handle the dynamic operations of peers are thus necessary. Next these operations are introduced.

A. Peer Joining

To ensure that our routing scheme, that will be proposed later, executes correctly as a new peer participates P2P systems, the peer joining algorithm must ensure that the routing entries of each peer are up to date, and reduce the imbalance of the trie tree at a low cost.

To address the issue, we design Algorithm 3, where x is the hashing value produced by a hashing algorithm for the joining peer, and y is the contacted peer.

Algorithm 3 Join(x)

```
1: $x'$ =Route($y, x$)
2: while(peer $\sigma^i(x')$ contains at least two identifiers)
3: {peer $x'$ joins the place of $\sigma^i(x')$.
4: peer $x'$ acquires the identifier $\sigma^i(x')$.}
5: peer $x'$ shares one identifer and finally joins the place of its
   sharing identifier.
6: $a$ = Predecessor($x'$), $b$ = Successor($x'$).
7: $x'$ connects to its parent $x'' = x_1x_2...x_{k-1}$
8: Update $x'$'s $d$ neighbors of the interconnect network according
   to $x''$'s $d$ neighbors.
```

Line 1 in Algorithm 3 indicates that a new peer x generates its identifier using a hashing algorithm, and then probes a peer y in the system. Peer y routes the joining request of x to a suitable peer using the routing algorithm.

The algorithm can reduce the imbalance of the system (Line 2-4), since the additional identifiers kept by a peer uncover the signs of imbalance. For DPhoenix, peer x can choose a low level to join. As shown in Fig. 1 (peer 100 and 101 fail and 010 connects to 10), let us assume a peer 0101 wants to become a child of 010, it finds that one neighbor peer 10 stores another string, then 0101 becomes 100 or 101 and it becomes a child of 10. For BPhoenix, peer x can choose a low level of another trie tree such that the algorithm reduces the unbalance of different trie trees.

For both DPhoenix and BPhoenix, we have the following theorem.

Theorem 7: For peer x and its d neighbors $\sigma^i(x')$, the neighbors of all the offsprings of x are the offsprings of $\sigma^i(x')$.

Therefore peer x can find its neighbors according to the neighbors of its parent at a low cost (Line 8).

Theorem 8: With the probability $1-o(1)$, the largest cost to handle a joining peer is $\lfloor\sqrt{d\log_d n} - 1.5 + \log_d((1+\varepsilon)\log n - O(\log\log n))\rfloor + 2\log_d n + o(1)$.

Proof: The rejoining process can be analyzing as follows: Suppose peer x joins the deepest leaf peer and the tree is unbalanced. In the worst case, the neighbor of the leaf peer connects to the upper one level, and so on, until the link connects to the smallest leaf peer. The cost is $h_{max} - h_{min}$. The cost of the routing process is $\log_d n$. The cost of establishing the first in-neighbor is $\log_d n$. All other operations costs $o(1)$. From the above theorem in section III, we get the conclusion. $Q.E.D$

B. Peer Departure

An efficient scheme to handle the peer departure is the guarantee of network connectivity. If a leaf peer with wants to leave the system, it can leave freely at any time. Its data is redistributed based on the data placement scheme. The behavior of peer departure should not have any impact on the topology. If non-leaf peers want to leave, Algorithm 4 can repair the topology.

Suppose $(\sigma^i(x))^{-1}$ denotes a virtual neighbor from which there is a directed edge to x for directed interconnect networks. For undirected networks, $(\sigma^i(x))^{-1} = \sigma^i(x)$.

Algorithm 4 Departure(x)

```
1: Peer $x$ forwards a REPLACEMENT message to a offspring leaf
   peer $u$.
2: while($u$ contains at least two identifiers)
3: {$u$ forwards the REPLACEMENT message to peer $(\sigma^i(u))^{-1}$.
4: $u = (\sigma^i(u))^{-1}$. }
5: $u$ is selected to replace peer $x$.
6: Update $x$'s trie tree neighbors.
7: Update $x$'s $d$ $(\sigma^i(x))^{-1}$ neighbors according to the neighbors of
   $x$'s parent.
```

Like the joining algorithm, Algorithm 4 can also reduce the imbalance of the trie tree. Peer x chooses a higher level leaf peer to replace it, and the leaf peer leave its original position. For DPhoenix and BPhoenix, the cost of update operation is low due to the structure of the trie tree.

Theorem 9: With the probability $1-o(1)$, the largest cost to handle a departing inner peer x is $\lfloor\log_d n + \sqrt{d\log_d n} - 1.5\rfloor -$ Length(x).

C. Peer Failure

The correctness and effectiveness of the system rely on the fact that the $2d+1$ neighbors of each inner peer and $d+1$ neighbors of each leaf peer are up to date.

To increase robustness and keep high effectiveness, each peer periodically verifies their links. Once a peer is missing, the peer is treated as a departed peer.

D. Load Balance

To demonstrate the load balancing scheme we use the example as shown in Fig. 2. For DPhoenix, according to Theorem 7, if node c stores the IP data for node e and all the nodes downstream from e. In other words, c has a global view of the subtree e. When c is over loaded, it initiates the load balancing algorithm.

There are two major steps of the load balancing algorithm. The first step aims at making the subtree e be balanced. The second step tries to put the leaf peers of subtrees e and c are at the same level. A rotation scheme in an AVL tree is used in the first step. Peer c sorts all the peers of the subtree e in an in-order traversal. The order is m, l, k, j, h, e, i. The rotation process is that m replaces l; l replaces k; and so on. Finally, e becomes a child of i. Fig. 3(a) shows the result. Using the same in-order traversal method, the next result can be seen in Fig. 3(b). Finally, the subtree becomes balanced. Fig. 3(c) shows the final result. The second step is that m and, k become the children of c, and h and, i become the children of m. Fig. 4 shows the result. After the algorithm is performed, the original subtree c and b are both balanced. The offsprings of c have shared load with it. For BPhoenix, a loaded peer choose peers of another trie tree. In the example, peer c is a node of a trie tree. Peer e, h, i, j, k, l and m of a neighboring trie tree have connections to peer c. The process is the same as for DPhoenix.

During the execution of the algorithm, the cost of data movement is low due to the efficient rotation scheme. However, several peers change their position in the tree, and their routing tables should be up to date. The influenced peers are only a small part of all the peers. It takes $O(\log n)$ hops to adjust routing table of a affected peer. Therefore, the load balancing algorithm does not significantly change the entire topology.

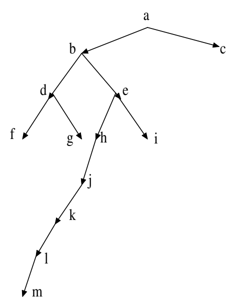


Fig. 2: Example of an unbalanced tree. Node c is overloaded.

E. Robust and effective routing scheme

The major messages handled by a P2P system can be partitioned into two categories: 1) messages to publish or query a resource, and 2) messages to maintain the topology. Besides the above messages, the system should also support the routing between any two peers. In order to route these kinds of messages to correct destinations effectively, each peer in the trie tree should keep a routing table and establish overlay connections to at least $2d + 1$ other peers based on the topology construction rules mentioned above. Due to the highly dynamic character of P2P systems and delay of related topology maintenance algorithms, however, some connections may not obey the construction rules all the time. This fact requires the routing algorithm to be fault-tolerant.

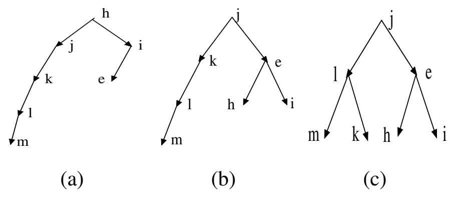


Fig. 3: The process of balancing for subtree e

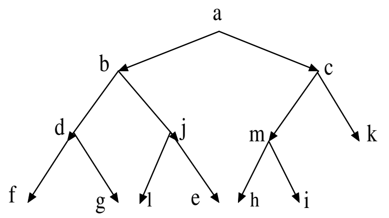


Fig. 4: The result of the load balancing algorithm for a loaded peer

Traditional routing schemes of DHTs work well in a stable environment, but suffer from poor robustness in dynamic P2P networks. The reason for this is that a message is always forwarded towards a unique neighbor that is closer to the destination than itself and other neighbors, and is not allowed to transfer along other paths.

To address these issues, we use a robust routing scheme for any interconnect network mapped to the trie tree. The algorithm can be found in Algorithm 5, in which x is the requesting or the current peer, and y is the destination or the requested resource.

For DPhoenix and BPhoenix, $\sigma^i(x)$ denotes a neighbor of node x in deBruijn and butterfly graph. A query towards a resource y can be routed by two steps. First, it is sent to a peer which is located at the same level as the beginning peer and has a common identifier prefix with y. Second, it is transmitted along down-link until the request reaches the leaf peer which holds the y. For DPhoenix, as shown in Fig. 1, if peer 00 wants to acquire a resource 111.... The request is forwarded along the link $00 \rightarrow 01 \rightarrow 11$, and then along the link $11 \rightarrow 111$. A common feature of those scenarios is that all related peers are alive. If some related peers fail, the algorithm still works.

Theorem 10: In a dynamic environment, with the probability $1 - o(1)$, the diameters of DPhoenix and BPhoenix are $\lfloor \log_d n \rfloor$ and $2\lfloor \log_d n \rfloor - 1$ respectively.

Proof: The above schemes can assure that the height of the trie tree fluctuates around $\lfloor \log_d n \rfloor$ $w.h.p.$ From theorem 6, we know the degree of each peer keeps stable, which can keep the routing algorithm stable. In addition, for an

Algorithm 5 Route(x, y)

```
1: CASE1: Length($x$) = Length($y$) or ($y$ is a resource and x is not a prefix $y$)
2: if $x = y$ then
3:     Return available.
4: else
5:     if Comprefix($x$, Successor($x$)) $= k - 1$ and Successor($x$) is less than $y$ in the ring then
6:         Forward the message to peer Successor($x$).
7:     else
8:         if Comprefix($x$, Predecessor($x$)) $= k - 1$ and Predecessor($x$) is larger than $y$ in the ring then
9:             Forward the message to peer Predecessor($x$).
10:        end if
11:    end if
12: else
13:    $x$ forward message to its neighbor $\sigma^i(x)$ which has the largest value Comprefix($\sigma^i(x), y$) among all the neighbors.
14: end if
15: CASE2: Length($x$) < Length($y$) or ($y$ is a resource and $x$ is a prefix $y$)
16: if $x$ contains $y$ then
17:    return available.
18: else
19:    if $x$ has at least one child then
20:        Forward message to its child $z$ which has the largest value of Comprefix($z, y$)
21:    end if
22: else
23:    $x$ forward message to its neighbor $\sigma^i(x)$ which has the largest value of Comprefix($\sigma^i(x), y$) among all the neighbors.
24: end if
25: CASE3: Length($x$) > Length($y$)
26: if $x$ has a link to its parent then
27:    $x$ forwards message to its parent.
28: else
29:    $x$ forwards message to its neighbors $\sigma^i(x)$ which has the largest value of Comprefix($\sigma^i(x), y$) among all the neighbors.
30: end if
```

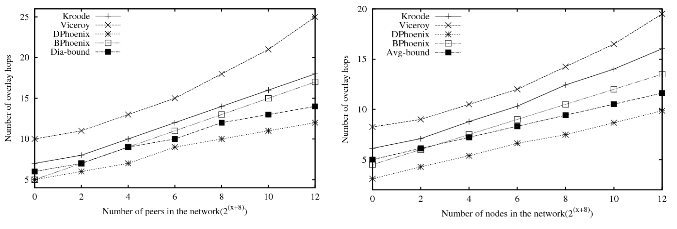
imbalance level, a peer in the level has connections to a peer in the balance level. Meantime, as the routing algorithm, the routes can go along the connections. Thus, a peer is usually routed along a shortest path as deBruijn and butterfly protocol. Therefore the diameters are $\lfloor \log_d n \rfloor$ and $2\lfloor \log_d n \rfloor - 1$ $w.h.p.$ $Q.E.D$

VI. PERFORMANCE EVALUATION

We build a P2P simulator to evaluate the performances of DPhoenix and BPhoenix. We evaluate networks consisting of 256 to 1 million peers. Queries are generated randomly and uniformly from the network, and the identifiers of targeted resources are also uniformly distributed in the key-space. The simulation environment is windows XP, pentium 4 CPU 3.0 GHz, 1 G memory. For comparison purposes, we also implement Kroode based on deBruijn, and Viceroy based on butterfly in the simulator. During the tests, d is set to 4 for all experiments, and thus the degree of each graph is also 4.

A. Routing Efficiency of Dynamic Networks

In this section, we show the maximum and average routing path lengths of each graph in the dynamic networks. For



(a) Maximum query distance (b) Average query distance

Fig. 5: Routing efficiency of dynamic network

different numbers of peers, 10% nodes of the total peers are allowed to join and (depart) in (from) the network. The maximum and average number of hops are recorded in the dynamic scenario. The diameter lower bound ($Dia - bound$) and average distance lower bound ($Avg - bound$) are also shown for the two experiments. The results are plotted in Fig.5. This figure shows that the delays of DPhoenix are both less than the lower bounds. The query distances of BPhoenix are a little larger than the lower bounds. This is because BPhoenix is based on the butterfly graph which does not have an optimal diameter as deBruijn does. DPhoenix and BPhoenix have far better performance than Kroode and Viceroy whose routing efficiencies are worse than the lower bounds. The experiment confirms that our construction method generates P2P networks that can adapt to the dynamic environment, and the performances are not limited by the lower bound any more.

B. Effects of Optimal Design

The performances of optimal design for DPhoenix and BPhoenix are evaluated in this section. For the optimal resource placement strategy, we suppose the maximum assumed load of each peer is 10^{2+x} where $x = 0, ..., 12$, corresponding to the number of peers in each system. The ratios of query and updating operations are approximately 60% and 40% respectively. Based on these experimental conditions, we get the following results for optimal topology construction method and optimal resource placement rule for BPhoenix and DPhoenix. Fig. 6 shows the average cost per node of the optimal and none-optimal systems. From the figure, we observe that as the network sizes increase, the routing cost for all systems also increases. However, OPT-DPhoenix and OPT-BPhoenix incur much lower cost than the none-optimal designs.

C. Load Balancing

A desirable feature of P2P systems is to evenly distribute keys among peers and make each peer receive similar number of requests forwarded by its neighbors. The percentage of peers whose load approximately equals to the average load is used as a measure for the load balancing capability of a P2P system. At most 5% error is allowed to evaluate the load balancing scheme. For example, the average load is 100 resources per peer, and all the values between 95 and 105 are calculated in the percentage.

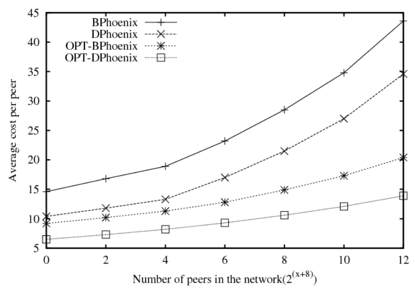


Fig. 6: Average cost of the optimal and none-optimal systems

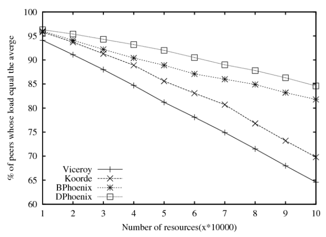


Fig. 7: Variances of the resource distribution

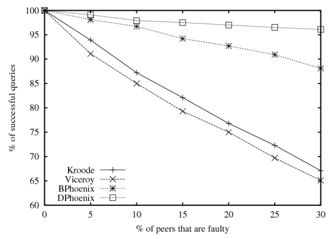


Fig. 8: Percentage of successful queries

In total 10000 peers are used in our experiment for each system. We vary the number of resources from 10^5 to 10^6 in increments of 10^5. The results are plotted in Fig.7. As the increasing of number of resources, proportion of each system decreases. DPhoenix and BPhoenix are more stable with the help of their load balancing algorithms. BPhoenix, however, takes more cost. There are no efficient load balancing algorithms for Viceroy and Kroode; and they have poor resource distributions compared with DPhoenix and BPhoenix.

D. Robustness

A number of 4000 queries are uniformly generated in the network consisting of 10000 peers. When the network becomes stable, we let a node fail with probability p ranging from 0.1 to 0.3. Fig. 8 plots the percentage of successful queries when using the forwarding algorithm of corresponding protocol. For the same percentage of failed nodes, the percentages of successful queries for DPhoenix and BPhoenix are much higher than that of other systems, and their variations are minimal. This is because all the schemes for the systems are robust to guarantee the successful query. However BPhoenix possess worse fault tolerance capability than DPhoenix due to the links between different trie trees. Though Viceroy and Kroode are based on the same interconnect networks as BPhoenix and DPhoenix, they encounter more failed queries. The reason is that a message is always forwarded towards a unique neighbor that is closer to the destination and is not allowed to transfer along other paths during their routings.

VII. Conclusions

In this study, we present the lower bounds of network diameter and average routing distance of P2P systems in highly dynamic environments. We then propose a dynamic d-way trie tree as a universal method to support any static network, further more we propose a series of optimal schemes for P2P systems. Based on these schemes, two classical static graphs (deBruiijn and butterfly) are adopted to construct two practical P2P architectures (DPhoenix and BPhoenix). The topology management and routing schemes guarantee flexible resource distribution and robust topology for the two systems. DPhoenix and BPhoenix achieve an optimal routing efficiency which exceeds the lower bounds, and they also own balanced node degrees and good connectivity in a dynamic scenario. We also have the plan of using DPhoenix as an infrastructure to support other large-scale and distributed applications.

References

[1] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In Proceedings of the ACM SIGCOMM, 2001.

[2] A. Rowstron and P. Druschel. Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. Lecture Notes in Computer Science, 2218:329C350, 2001.

[3] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup service for internet applications. In Proceedings of the ACM SIGCOMM, 2001.

[4] M. R. Garey and D. S. Johnson. Computers and intractability: a Guide to the theory of NP-completeness. Freeman, 1979.

[5] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, Making gnutella-like P2P systems scalable. In ACM SIGCOMM, Aug. 2003, pp. 407C418.

[6] P. Fraigniaud and P. Gauron. The content-addressable network d2b. Technical Report Technical Report LRI 1349, Univ. Paris-Sud, 2003.

[7] F. Kaashoek and D. R. Karger. Koorde: a simple degree-optimal hash table. In 2nd International Peer-To-Peer Systems Workshop (IPTPS), 2003.

[8] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. Graph-theoretic analysis of structured peer-to-peer Systems: Routing Distances and Fault Resilience. Texas A&M Technical Report, 2003.

[9] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In Proceedings of the 21st ACM Symposium on Principles of Distributed Computing, 2002.

[10] M. Naor and U.Wieder. Novel architectures for P2P applications: The continuous-discrete approach. In ACM SPAA, Jun. 2003, pp.50C59.

[11] J. Xu, A. Kumar, and X. Yu. On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks. IEEE J. Sel. Areas Commun., vol. 22, no. 1, pp. 151C163, Jan. 2004.

[12] F. S. Action. Numerical methods that work. Harpercollins College Div, 1970.

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

[14] S. Banerjee and D. Sarkar. Hypercube connected rings: a scalable and fault-tolerant logical topology for optical networks. Computer communications, 24:1060C1079, 2001.

[15] 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 1677C1688, Miami,Florida,USA, March 2005.

[16] X. Wang, D. Loguinov. Load-balancing performance of consistent hashing: asymptotic analysis of random node join. In IEEE/ACM Transaction on Networking, 2007.

[17] W. Szpankowski. Patricia trees revisited again. Journal of the ACM, vol. 37, 1990.

[18] M.H. DeGroot. Probability and statistics. Addison-Wesley, 2001.

[19] D. Guo, J. Wu, H. Chen, and X. Luo. Moore: an extendable P2P network based on incomplete kautz digraph with constant degree. In Proc. 26th IEEE INFOCOM, May 2007.

[20] W.G. Bridges and S. Toueg. On the impossibility of directed moore graphs. Journal of Combinatorial Theory, series B29, no. 3, 1980.

[21] D. Guo, Y. Liu and X. Li. BAKE: a balanced kautz tree structure for peer-to-peer networks. In Infocom, 2008.