Distributed Line Graphs: A Universal Technique for Designing DHTs Based on Arbitrary Regular Graphs

Yiming Zhang and Ling Liu, *Senior Member, IEEE*

Abstract—Most proposed DHTs engage certain topology maintenance mechanisms specific to the static graphs on which they are based. The designs of these mechanisms are complicated and repeated with graph-relevant concerns. In this paper we propose the "distributed line graphs" (DLG), a universal technique for designing DHTs based on arbitrary regular graphs. Using DLG, the main features of the initial graphs are preserved, and thus people can design a new DHT by simply choosing the graph with desirable features and applying DLG to it. We demonstrate the power of DLG by illustrating four DLG-enabled DHTs based on different graphs, namely, Kautz, de Bruijn, butterfly and hypertree graphs. The effectiveness of our proposals is demonstrated through analysis, simulation and implementation.

Index Terms—Distributed networks, network topology, distributed hash tables, regular graphs

1 Introduction

DISTRIBUTED hash tables (DHTs) [1], [2] become increasingly popular in recent years. DHTs have two important properties, namely, the *node degree* (i.e., the number of neighbors of each node), and the *network diameter* (i.e., the largest distance over all the pairs of nodes). *Constant-degree* [3] DHTs have an exact or asymptotic constant node degree irrespective of the network size, thus ensuring scalability as the network evolves. As a result, recently constant-degree DHTs have attracted significant attention from the academic fields.

Constant-degree DHTs are usually designed based on a specific type of *regular graphs*, in which all nodes have the same number of edges. E.g., CAN [2] is based on *d*-torus; Viceroy [4] is based on butterfly graphs; D2B [5] and Koorde [6] are based on de Bruijn graphs; FissionE [7] and Moore [8] are based on Kautz graphs; Cycloid [9] is based on CCC graphs.

To deal with network dynamics such as node joins/leaves, DHTs engage certain topology maintenance mechanisms, the design of which is usually complicated and repeated. By now, most DHTs have their own clean-slate designs of these mechanisms, which are tightly coupled with the static graphs on which they are based.

In this paper we present the *"distributed line graphs"* (**DLG**), a universal technique for designing different DHTs based on arbitrary regular graphs. DLG's novelty is that it preserves the main features (e.g., the degree, diameter, and routing algorithm) of the underlying graphs **without** knowing them. Suppose that in the future a novel (and currently unknown) graph *X* with desirable features is proposed. Then, people will be able to design a new, *X*-based DHT by simply applying DLG to it.

We prove that in a DLG-enabled, *N*-node DHT, the out-degree is *d*, the in-degree is between 1 and 2*d*, and the diameter is less than $2(\log_d N - \log_d N_0 + D_0 + 1)$, where *d*, D_0 and N_0 represent the base, diameter and number of nodes of the initial graph, respectively. This diameter reaches the theoretical lower bound $\Omega(\log_d N)$ [3] of constant-degree DHTs. The message cost caused by each join/leave is $O(\log_d N)$, and the maximum difference between node identifier lengths is no more than $\log_d N - \log_d N_0 + D_0$. We also provide procedures that deal with network complexity such as ungraceful leaves, concurrency, and robust routing under churn.

Our proposed DLG technique is inspired by a novel technique called *line graph* (**LG**) iteration [10]. The LG iteration, as well as its variations [11-15], has been extensively studied and applied as a universal approach to designing interconnection networks, e.g., multiprocessor networks. However, these techniques require global topology information and centralized control, and thus cannot be applied to DHTs.

The challenge we face is the absence of knowledge for what graphs we are to deal with, together with the characteristics of DHTs such as decentralized control and lack of global information. To address these problems, the key idea behind DLG lies in a simple *local* edge-node transition mechanism for routing table construction, which preserves the main features of the initial graphs.

To our knowledge, we are the first to propose a universal, feature-preserving technique for designing different DHTs based on arbitrary regular graphs. Moreover, our DLG technique provides a number of methodological advantages, some of which are listed as follows.

First, our DLG technique remarkably simplifies the de-

- *Y. Zhang is with the School of Computer, National University of Defense Technology, Changsha 410073, China (e-mail: ymzhang@nudt.edu.cn).*
- *L. Liu is with the College of Computing, Georgia Institute of Technology, Atlanta, GA, 30332 USA. E-mail: lingliu@cc.gatech.edu.*

Manuscript received December 10, 2010.

sign of new DHTs. We demonstrate the power of DLG by illustrating four new, DLG-enabled DHTs based on different graphs, namely, Kautz, de Bruijn, butterfly and hypertree graphs.

Second, the universal technique allows an in-depth study of common design choices of DHTs. To address the topology imbalance problem, for example, in existing DHTs all joining nodes first route to a random node, the cost of which is $O(\log N)$ and might be relatively expensive for highly dynamic networks. In DLG-enabled DHTs, however, we show (in Section 4.4) that this cost can be evicted while still bounding the imbalance to the same range as before.

DLG-Kautz (*DK*), a DLG-enabled, Kautz graph-based DHT, is proved to have the lowest diameter among all the DLG-enabled DHTs. So, we use DK as a model to evaluate the effectiveness of DLG through extensive simulations. We have also implemented a prototype [16] of DK.

Note that, however, the goal of this paper is to provide a *universal* topology maintenance mechanism and simplify the design of DHT's backbone (mainly including handling node join/leave events and message routing), but NOT to design a *particular* DHT. Thus, after getting the backbone of a new DHT by using our DLG technique, people still need to consider more details (e.g., load balancing for resource discovery) for achieving a complete and practical DHT.

The rest of this paper is organized as follows. Section 2 presents the basic design and analysis of DLG. Section 3 discusses the problem of how to support arbitrary base in DLG. In Section 4 we utilize the DLG technique to design different DHTs, which are evaluated by extensive simulations in Section 5. Section 6 discusses related work and Section 7 concludes the paper.

2 Basic Distributed Line Graphs

At the beginning the DHT topology is referred to as *initial graph*. Then it evolves into a family of graphs as nodes join/leave the network. Basically, all topology maintenance mechanisms can be abstracted as addressing the problem of "how to add/delete a node to/from a graph G, while preserving main features of G". For simplicity this section focuses on the problem of adding a node, and we will discuss the inverse deletion operation in Section 4.

2.1 Notation

For convenience we first present the following notation:

- *Node* and *edge*: The DHT topology is modeled by a graph $G=(V,E)$ whose *nodes* $V=V(G)$ and *edges* $E=E(G)$ represent, respectively, the processing elements (nodes in the network) and the links between them. We use $e=[x,y]$ to denote an edge e from node x to node y.
- *Out-neighbor*, *in-neighbor*, *out-edge* and *in-edge*: Let $e=[x,y]$, we say that y is an *out-neighbor* of x, and that x is an *in-neighbor* of y. We use $\Gamma_G^+(x)$ and $\Gamma_G^-(x)$ to denote the sets of out-neighbors and in-neighbors of x, respectively. We also say that e is an *out-edge* (resp. *in-edge*) of x (resp. y).

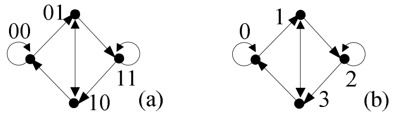


Fig. 1. Example of unified description. (a) shows a standard de Bruijn graph $B(2,2)$ and (b) shows its unified description. In (b), $\zeta(0)=\{0,3\}, \psi(0)=\{0,1\}, \zeta(1)=\{0,3\}, \psi(1)=\{2,3\}, \zeta(2)=\{1,2\}, \psi(2)=\{2,3\}$ $\zeta(3)=\{1,2\}, \psi(3)=\{0,1\}$.

- *Order* of a graph: The number of nodes in graph G is called the *order* of G.
- *Out-degree*, *in-degree* and *degree*: The number of out-neighbors (resp. in-neighbors) of node x is called *out-degree* (resp. *in-degree*) of x. The degree of x is the sum of its out-degree and in-degree.
- *d-out-regular*, *d-in-regular* and *d-regular*: Graph G is *d-out-regular* (resp. *d-in-regular*) if x's out-degree is d (resp. x's in-degree is d) for all $x \in G$. Graph G is *d-regular* (*regular* for short) if it is *d-out-regular* and *d-in-regular*.
- *Distance* and *diameter*: The *distance* from x to y, denoted as $d_G(x,y)$, is the length of a shortest path from x to y. The *diameter* of G, denoted as $D(G)$, is the largest distance over all the pairs of nodes.
- $|u|$: The identifier length of node u.

2.2 Unified Description of Initial Graphs

Regular graphs usually have their different notation to define nodes/edges. In order to simplify the description of DLG's design and analysis, in this section we unify the way to describing a d-regular, N_0-node initial graph G_0.

Let $|u|$ denote the length of the identifier of node u, e.g., if $u=u_1u_2...u_k$ then $|u|=k$. Let X be an *alphabet* of N_0 letters and $\forall x \in X$ satisfies $|x|=1$. In our unified description mechanism, the initial graph G_0 in turn names its N_0 nodes with N_0 letters $x \in X$, and for each node $\alpha \in G_0$ there is an *in-letter set* $\zeta(\alpha)$ and an *out-letter set* $\psi(\alpha)$, which are defined as

$$\zeta(\alpha)=\Gamma_{G_0}^-(\alpha), \tag{1a}$$

$$\psi(\alpha)=\Gamma_{G_0}^+(\alpha). \tag{1b}$$

The letters in X could be any symbols such as $\{a,b,c,\cdots\}$, $\{\alpha,\beta,\gamma,\cdots\}$ or $\{0,1,2,\cdots\}$. For the sake of convenience, we simply name nodes in an initial graph G_0 with identifiers $0,1,2,\cdots,N_0-1$. This "alphabet" approach enables us to describe all d-regular graphs in a unified way. For example, Fig. 1(a) shows the original de Bruijn graph $B(2,2)$, and Fig. 1(b) shows its unified description.

2.3 Formal Definition of Distributed Line Graphs

The basic idea of distributed line graphs (DLG) is simple: with a *special* node v (which will be formally defined in Definition 1), DLG (i) deletes v, (ii) turns v's in-edges into new nodes, and (iii) generates new in-/out-edges.

Before we present the definition of DLG, we first introduce an operator $\circ$ for turning an edge into a node. Let $u=u_1u_2...u_m$, $v=v_1v_2...v_n$, and $m \geq n$. Define the conjunction operator $\circ$ as (2):

$$u \circ v = u_{m-n+1}v_1v_2...v_n = u_{m-n+1}v. \tag{2}$$

For example, $12 \circ 20 = 120$, $012 \circ 23 = 123$. Clearly $u \circ v$

contains the information of *v*. On the other hand, as discussed in Theorem 1 (see Section 2.5), for an edge $[u, v] = [u_1u_2...u_m,\ v_1v_2...v_n]$ satisfying $m \geq n$ in distributed line graphs, $u = \beta v_1v_2...v_{n-1}$ if $m = n$, or $u = \beta\beta v_1v_2...v_{n-1}$ if $m > n$ (in this case we have $m = n + 1$). Consequently, we have

- $u \circ v = \beta v_1v_2...v_n = u_1u_2..u_mv_n$ if $m = n$, or
- $u \circ v = \beta v_1v_2..v_n = u_2..u_mv_n$ if $m > n$.

Thus $u \circ v$ also contains the information of *u*. So, we can say that $u \circ v$ contains the information of edge $[u,v]$.

Next we will present the definition of *distributed line graphs* in an iterative manner, that is, we define the nodes and edges of G_{i+1}, the (i+1)th graph of a series of distributed line graphs, by describing how to obtain G_{i+1} from G_i, $i = 0,1,2,\cdots$.

Definition 1. *Let the initial graph* $G_0 = (V,E)$ *be a d-regular graph. A series of graphs* $G_{i+1} = DL(G_i, v^{(i)})$ *with* $i = 0,1,2,\cdots$, *where node* $v^{(i)} \in V(G_i)$ *satisfies*

$$\forall u \in \Gamma^-_{G_i}(v^{(i)}) \cup \Gamma^+_{G_i}(v^{(i)}),\ \left|v^{(i)}\right| \leq |u|, \tag{3a}$$

is said to be a family of ***distributed line (DL) graphs*** *with base d, if the following conditions hold.*

$$V(G_{i+1}) = V(G_i) - \{v^{(i)}\} + \{u \circ v^{(i)} \,|\, u \in \Gamma^-_{G_i}(v^{(i)})\} \tag{3b}$$

$$\begin{aligned} E(G_{i+1}) = E(G_i) &- \{[x, v^{(i)}] \,|\, x \in \Gamma^-_{G_i}(v^{(i)})\} - \{[v^{(i)}, y] \,|\, y \in \Gamma^+_{G_i}(v^{(i)})\} + \\ &\{[u, u \circ v^{(i)}] \,|\, u \in \Gamma^-_{G_i}(v^{(i)})\} + \{[u \circ v^{(i)}, w] \,|\, u \in \Gamma^-_{G_i}(v^{(i)}), w \in \Gamma^+_{G_i}(v^{(i)})\} \end{aligned} \tag{3c}$$

The transition from G_i to G_{i+1} is called **distributed line (DL) iteration**, and node $v^{(i)}$ is called **responsible node**. (We defer the discussion on how to find $v^{(i)}$ to Section 4.1.) We say that the series of DL graphs is derived from initial graph G_0.

In Definition 1: (3a) puts restrictions on the responsible node of each DL iteration for balance purpose (which will be used in the following analysis in Section 2.5), i.e., the identifier length of $v^{(i)}$ is no greater than any of its direct neighbors; (3b) gives the new nodes generated by old edges; and (3c) presents the rules of generating new edges.

Let's take Fig. 2(a) ~ (e) as an example to illustrate the decomposed procedure of DL iteration $G_1 = DL(G_0, v^{(0)})$ with $v^{(0)} = 1$:

1) Let $G_1 = G_0$, as shown in Fig. 2(a);
2) Delete node $v^{(0)}$ and all edges adjacent to/from $v^{(0)}$ in the new graph G_1, as shown in Fig. 2(b);
3) For each in-edge $[u, v^{(0)}]$ adjacent to $v^{(0)}$ in G_0, add a new node $u \circ v^{(0)}$ to G_1, as shown in Fig. 2(c);
4) Add in-edges of the form $[u, u \circ v^{(0)}]$ to the new nodes $u \circ v^{(0)}$ in G_1, as shown in Fig. 2(d); and
5) Add out-edges of the form $[u \circ v^{(0)}, w]$ to $u \circ v^{(0)}$ in G_1 with $w \in \Gamma^+_{G_0}(v^{(0)})$, as shown in Fig. 2(e).

Fig. 2(f) ~ (j) show another five consecutive DL iterations, i.e., $G_2 = DL(G_1, 4)$, $G_3 = DL(G_2, 3)$, $G_4 = DL(G_3, 0)$, $G_5 = DL(G_4, 2)$, and $G_6 = DL(G_5, 01)$. From these examples we can see that each new node $u \circ v^{(i)} \in V(G_{i+1})$ corresponds to an edge $[u, v^{(i)}] \in E(G_i)$, and that all the new nodes in the new DL graph G_{i+1} have the same out-neighbors $w \in \Gamma^+_{G_i}(v^{(i)})$ as the responsible node $v^{(i)} \in V(G_i)$.

Note that a series of DL iterations from G_0 cannot assure to achieve the line graphs [10] of G_0. This is decided by the selection of responsible nodes in each DL iteration. For example, let $G_5 = DL(G_4, 2)$ as shown in Fig. 2(i),

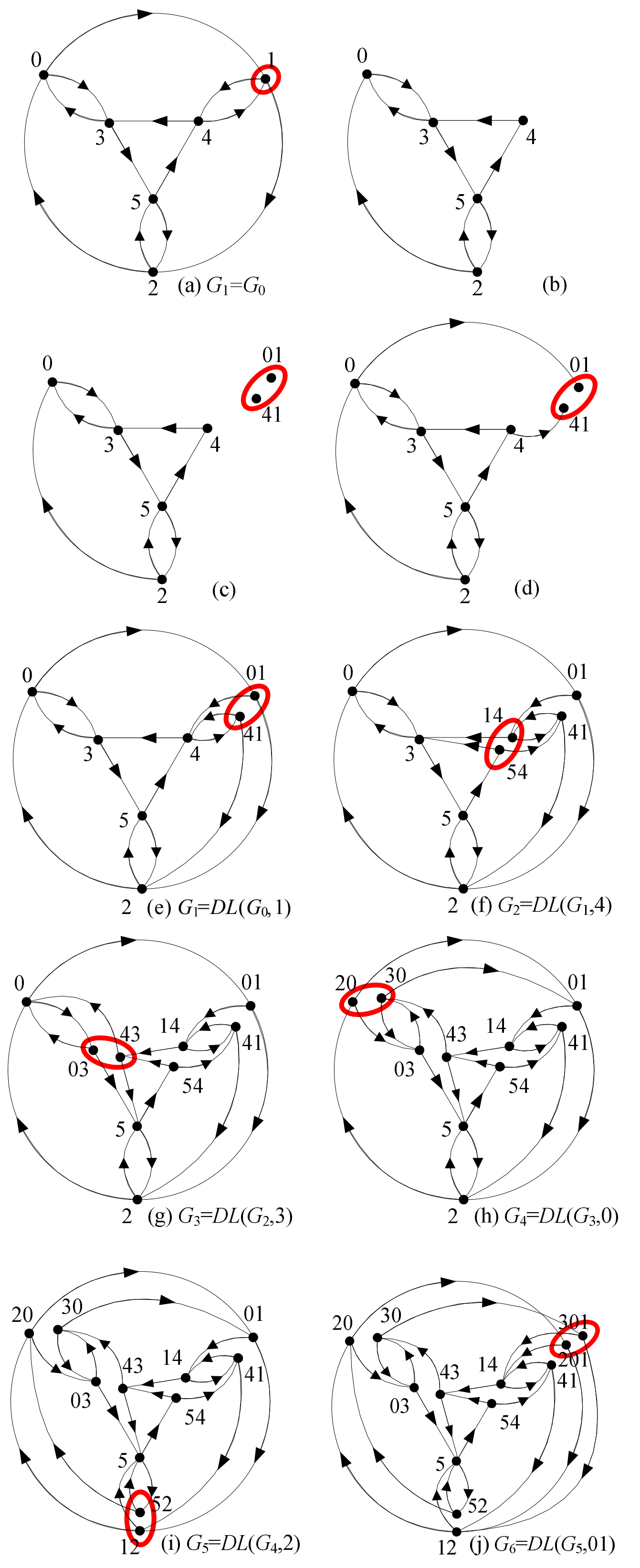


Fig. 2. Examples of DL iterations.

where node 01 satisfies the requirements for being a responsible node. Then, let $G_6 = DL(G_5, 01)$ as shown in Fig. 2(j), currently in G_6 there are two nodes with length 3 (nodes 201 and 301), and a node with length 1 (node 5). Clearly this series of graphs will never achieve the line graph of G_0. And G_6 cannot be achieved by any series of PLG iterations [11] from G_0 either.

2.4 Routing

The DL graph is started with initial graph G_0, then it evolves with a series of DL iterations. Consequently, the routing path from $u = u_1u_2...u_m$ to $v = v_1v_2...v_n$ in a DL graph includes two subpaths. The first has a counterpart in G_0, and the second is from the node with its last letter being v_2 to destination v, corresponding to the letters appended by DL iterations.

(i) The first subpath: Clearly we have $u_i \in V(G_0)$ and $v_k \in V(G_0)$ for $i = 1,2,\cdots,m$ and $k = 1,2,\cdots,n$. Suppose that the diameter of G_0 is D_0, then there must be a path $u_m \to x_1 \to \cdots \to x_t \to v_1$ with $t \le D_0 - 1$. Let $x = x_1x_2...x_t$, $w = uxv = u_1..u_mx_1..x_tv_1...v_n = w_1w_2...w_{m+t+n}$. Then there must be a path from u in graph G: $u = s^{(0)} \to s^{(1)} \to \cdots \to s^{(t+n-1)} \to s^{(t+n)}$ where the kth intermediate node $s^{(k)}$ in the path is of the form $s^{(k)} = w_iw_{i+1}...w_{m+k}$, i.e. a substring of w. Theorem 1 (See Section 2.5) assures the existence of each $s^{(k)}$.

(ii) The second subpath: The last character of $s^{(k)}$ is w_{m+k}. Then, at the $(t+n)^{\text{th}}$ step the last character of node $s^{(t+n)}$ is $w_{m+t+n} = v_n$, and the length of $s^{(t+n)}$ may be one of the three cases: $|s^{(t+n)}| > n$, $|s^{(t+n)}| = n$, and $|s^{(t+n)}| < n$. Note that if $|s^{(t+n)}| = n$ we have $s^{(t+n)} = v_1v_2...v_n = v$. Since there already exists node v in G, only the case $|s^{(t+n)}| = n$ is possible. Thus $s^{(t+n)} = v$ and the routing is over.

The above DL routing procedure is summarized as follows. Note that this algorithm is decentralized and run on individual nodes, i.e., the next-hop is decided by the node where the message arrives. This algorithm would be invoked on each intermediate node along the routing path.

```
Procedure w.DL_Routing (Source u, Destination v, Hops k)
// w is the current node where the message arrives.
// k means how many hops the message already traversed.
1 if (w == v) return;                    // routing is over
2 Let $u = u_1u_2...u_m$ and $v = v_1v_2...v_n$;
3 Suppose in $G_0$ the path from $u_m$ to $v_1$: $u_m \to x_1 \to \cdots \to x_t \to v_1$;
4 if ($k \le t$) {
5     next_hop = the neighbor with last character being $x_k$; }
6 else {
7     next_hop = the neighbor with last character being $v_{k-t}$; }
8 next_hop.DL_Routing (u, v, k+1);
```

For example, the routing path from node 5 to node 01 in Fig. 2(h) is $5 \to 2 \to 20 \to 01$, where the first two hops correspond to Line 3 and the last corresponds to Line 4.

2.5 Analysis

This subsection analyzes the properties of the DL iteration and DL graphs. Note that for the sake of clarity only outlined proofs are presented in this paper and formal proofs are included in our technical report [28].

The following Theorem 1 describes the in-/out-neighbors of a node in DL graphs.

Theorem 1. *Let graph G be a DL graph with base d. Let* $x = x_1x_2...x_n \in V(G)$.

(i) If there is a node $y \in \Gamma_G^-(x)$ *satisfying* $|y| < |x|$, *then*

$$\Gamma_G^-(x) = \{y \mid y = x_1...x_{n-1} \in V(G)\}. \tag{4a}$$

Otherwise, for each $\beta \in \zeta(x_1)$, *either there is an in-neighbor* $y \in \Gamma_G^-(x)$ *satisfying*

$$y = \beta x_1x_2...x_{n-1}, \tag{4b}$$

or there are d in-neighbors $y \in \Gamma_G^-(x)$ *satisfying*

$$y = \beta'\beta x_1x_2...x_{n-1} \tag{4c}$$

with $\beta' \in \zeta(\beta)$. *And x has no other in-neighbors in G.*

(ii) For each $\beta \in \psi(x_n)$, *there is one out-neighbor* $z \in \Gamma_G^+(x)$ *satisfying*

$$z = x_t..x_2x_3..x_n\beta \tag{5}$$

with $1 \le t \le 3$ *(* $x_m...x_n$ *represents a null string if* $m > n$*). And x has no other out-neighbors in G.*

This theorem is proved by utilizing two obvious properties of DL graphs: (i) the identifier of one node cannot be a suffix of another; and (ii) if a node has an in-neighbor with shorter identifier, then it has no other in-neighbors.

Proof outline. We prove Theorem 1 by applying mathematical induction to all DL graphs G_i with $i = 0,1,2,\cdots$. Clearly Theorem 1 holds initially for G_0. Suppose that it holds for G_i. Let $G_{i+1} = DL(G_i, v)$ and $v = v_1v_2...v_m$.

To prove (4), we first consider the case that x is a new node. The new node x is of the form $\alpha v_1v_2...v_m$. If x has some in-neighbor u satisfying $|u| < |x|$, then by (3a) we have $|u| = |v|$ and $u = \alpha v_1v_2...v_{m-1}$. We could easily prove node y is the only in-neighbor of x and thus (4a) holds. Otherwise, similar to the above case, by (2), (3a), (4a) and (3c), node v has d distinct in-neighbors in the form of $y = \lambda\alpha v_1v_2...v_{m-1}$ with $\lambda \in \zeta(\alpha)$, for each $\alpha \in \zeta(v_1)$ in G_i. Thus (4) holds when x is a new node. Similarly, (4) still holds for G_{i+1} when x is an out-neighbor of v and thus (4) holds. Similarly, we could prove (5) holds. □

This theorem shows in DL graphs that (i) the in-neighbors are restricted to three forms of (4a), (4b) and (4c), and that (ii) the out-neighbors are also restricted to three forms corresponding to t = 1, 2, 3 in (5). Moreover, we can easily infer that for a DL graph G:

- The difference between the identifier lengths of any two neighbors is no more than 1;
- After each DL iteration there would be $d - 1$ more nodes in the new graph; and
- For any node $x = x_1x_2..x_n$, we have $x_i \in \zeta(x_{i+1})$ and $x_{i+1} \in \psi(x_i)$ for $i = 0,1,\cdots,n-1$, where $\zeta(\alpha)$ and $\psi(\alpha)$ denote respectively the in-letter set and out-letter set of α in the initial graph G_0.

The second inference shows that DL iteration with base $d > 2$ cannot be directly applied to building DHTs. This problem will be addressed in the next section.

The following Theorem 2 describes the in-degree and out-degree of a node in DL graphs.

Theorem 2. *Let graph G be a DL graph with base d. For all nodes* $x \in V(G)$, *the out-degree and in-degree, denoted as* $\delta_G^+(x)$ *and* $\delta_G^-(x)$, *satisfy respectively*

$$\delta_G^+(x) = d, \tag{6a}$$

$$1 \le \delta_G^-(x) \le d^2. \tag{6b}$$

And the average in-degree of all nodes in G is d.

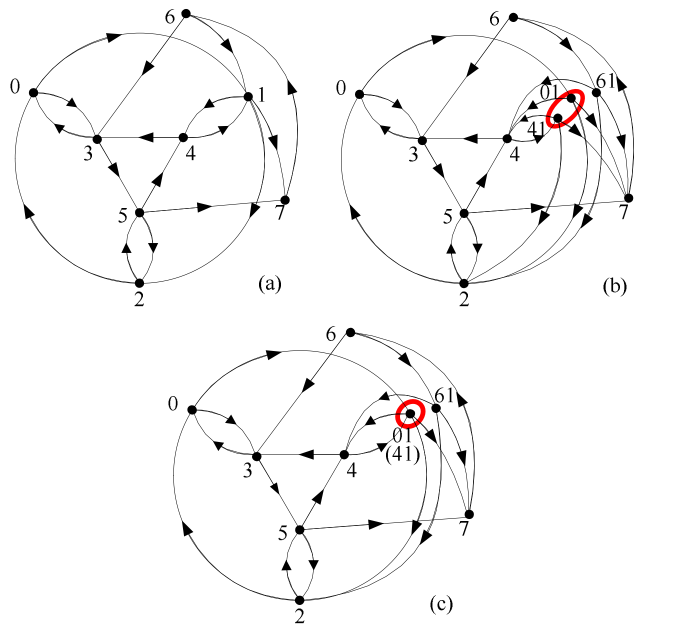


Fig. 3. Example of merge operation. Note that nodes 8, 9, 10, 11, which have nothing to do with this iteration, are omitted here for clarity.

Proof outline. Clearly (6a) holds initially for G_0. Suppose that it holds for G_i with $i \geq 0$. From Definition 1, the nodes in $G_{i+1} = DL(G_i, v)$ all have an out-degree of *d*. Thus (6a) holds. If *x* has an in-neighbor *y* satisfying $|y| < |x|$, then *y* is the only in-neighbor of *x*. Otherwise each in-neighbor *y* satisfies $|y| \geq |x|$. Then by Theorem 1, $d \leq \delta_G^-(x) \leq d^2$ and (6b) holds. Clearly, the sum of in-degrees of all nodes is equal to that of out-degrees, so the average in-degree is *d*. Thus Theorem 2 holds. □

This theorem shows that the DL graphs are always *d*-out-regular. And although the in-degree upper bound is relatively large (d^2) in DL graphs, we would show (in the following Theorem 4 in Section 3.4) that in practice a node in DLG-enabled DHTs has at most 2*d* in-neighbors.

The following Theorem 3 gives the upper bound for the diameter of DL graphs.

Theorem 3. *Let graph G be a DL graph with base d and order N. Let the order and diameter of the initial graph G_0 be N_0 and D_0, respectively. Then the diameter of G satisfies*

$$D(G) \leq 2(\log_d N - \log_d N_0 + D_0). \quad (7)$$

The proof includes two steps: (i) deriving the relationship between diameter and shortest node identifier length; and (ii) calculating the maximum difference between all pairs of node identifier lengths.

Proof outline. Let *u* and *v* be of the shortest and longest identifier in *G*, respectively. Let $|u| = m$, $|v| = n$. Then from the DL_Routing algorithm we can infer that the diameter of *G* satisfies $D(G) \leq D_0 + n - 1$.

The number of nodes (*N*) in *G* satisfies $N \geq N_0 \times d^{m-1}$, thus we have $m \leq \log_d N - \log_d N_0 + 1$. The distance from *v* to *u*, $d_G(v,u)$, satisfies $d_G(v,u) \leq D_0 + m - 1$. If two nodes *x* and *y* are neighbors in *G*, then $\|x| - |y\| \leq 1$. Since the identifier lengths satisfies $n - m \leq d_G(v,u) \leq D_0 + m - 1$, we have $n \leq D_0 + 2m - 1$. So $D(G) \leq 2D_0 + 2m - 2 = 2(\log_d N - \log_d N_0 + D_0)$. □

From the proof of Theorem 3 we can infer the following Corollary.

Corollary 1. The diameter of a DL graph *G* is no more than that of the initial graph *G*0 plus the largest length of identifiers among all nodes minus 1.

From Theorem 3 we can get an interesting property of DL graphs: the maximum difference between node identifier lengths is no more than $\log_d N - \log_d N_0 + D_0$, which will be used to discuss the topology balancing issue of different stabilization mechanisms in Section 4.4.

3 Supporting Arbitrary Base in DLG

As discussed in the above section, we infer by Theorem 1 that after each DL iteration there would be $d-1$ more nodes in the new graph than in the old one. For example, suppose that G_0 is a 3-regular graph $G_0 = K(3,2)$ as shown in Fig. 3(a). After the iteration $G_1 = DL(G_0, 1)$, as shown in Fig. 3(b) there are two more nodes in G_1 than in G_0. Therefore, DL iteration with base $d > 2$ cannot be directly applied to building DHTs.

This section extends DL iteration to support arbitrary base by proposing the following *merge/split* operations.

3.1 Node Merge

Let *G* be a DL graph with base *d* and let $G' = DL(G, v)$ with $v = v_1 v_2 \ldots v_m$. As discussed above, there are *d* new nodes in G', namely, $\alpha_i v_1 v_2 \ldots v_m$ with $\alpha_i \in \zeta(v_1)$ and $i = 0, 1, \cdots, d-1$.

For simplicity the *d* new nodes are sorted on their first letters in an ascending order. Let $h = \lceil d/2 \rceil$. In the **merge** operation, the first half, nodes of the form $\alpha_i v_1 v_2 \ldots v_m$ with $i \leq h$, would merge to one node $s = \beta v_1 v_2 \ldots v_m$ with $\beta = \alpha_0$; and the second half, nodes of the form $\alpha_i v_1 v_2 \ldots v_m$ with $i \geq h+1$, would merge to one node $t = \beta v_1 v_2 \ldots v_m$ with $\beta = \alpha_{h+1}$. The in-edges (resp. out-edges) of *s* and *t* in G'' are the union of the in-edges (resp. out-edges) of their component nodes in G', i.e., $\Gamma_{G''}^-(s) = \cup \Gamma_{G'}^-(\alpha_i v_1 v_2 \ldots v_m)$ and $\Gamma_{G''}^+(s) = \cup \Gamma_{G'}^+(\alpha_i v_1 v_2 \ldots v_m)$ with $i \leq h$.

For the sake of clarity, nodes *s* and *t* will be referred to as **physical nodes**; and the component nodes held by *s* and *t* are referred to as **logical nodes** (shortly nodes). The number of logical nodes of *s* in G'' is denoted as $[\![s]\!]$.

We denote this merge operation as $G'' = Merge(G', v)$. The two merged physical nodes are called *sib-neighbors* to each other. Fig. 3 shows an example of the merge operation, where two logical nodes 01 and 41 in Fig. 3(b) merge to one physical node 01 in Fig. 3(c), and the logical node 61 "merges" to itself since $\lceil d/2 \rceil + 1 = d$. After the merge operation, the two physical nodes 01 and 61 in Fig. 3(c) are sib-neighbors to each other with $[\![01]\!] = 2$ and $[\![61]\!] = 1$.

3.2 Node split

Suppose a physical node $v = \alpha_k v_1 v_2 \ldots v_m \in V(G)$ satisfies $[\![v]\!] > 1$. Then a **split** operation $G' = Split(G, v)$ would divide the logical nodes and edges held by *v* into two shares. Physical node *v* holds the first share, i.e. the logical nodes of the form $\alpha_i v_1 v_2 \ldots v_m$ with $k \leq i \leq k+h$ ($h = \lceil [\![v]\!]/2 \rceil$); and a new physical node holds the second share, i.e. the logical nodes of the form $\alpha_i v_1 v_2 \ldots v_m$ with $k + h < i \leq k + [\![v]\!]$. The identifier of the new physical node is $\alpha_{k+h+1} v_1 v_2 \ldots v_m$.

3.3 DL+ graphs

The basic DL iteration and merge/split operations are generally called **DLG** technique, and the series of graphs generated by DLG are referred to as **DL+ graphs**. Both the DL iteration and the merge/split operations need a responsible node v satisfying

$$\forall u \in \Gamma_G^-(v) \cup \Gamma_G^+(v)\,,\ |v| < |u| \text{ or } (|v| = |u|) \wedge (\llbracket v \rrbracket \geq \llbracket u \rrbracket)\,. \tag{8}$$

A DL+ graph is composed of physical nodes, and a DL graph is composed of logical nodes. At any time the topology of a DLG-enabled DHT is a DL+ graph G, which can be mapped to a *corresponding DL graph* G^* by replacing all the physical nodes with their logical nodes. Clearly, if every physical node holds only one logical node, then G is equal to G^*. We say that the initial graph, DL graphs and DL+ graphs have the same base d.

We refer to DHTs that are built based on DLG as *DLG-enabled* DHTs, and the processing of DLG can be summarized as follows.

```
Procedure DLG_transition (Old Graph G, New Graph G')
1 Choose a physical node $v \in V(G)$ satisfying for any $u$
    $\in \Gamma_G^-(v) \cup \Gamma_G^+(v)$, $|v| < |u|$ or $(|v| = |u|) \wedge (\llbracket v \rrbracket \geq \llbracket u \rrbracket)$;
2 if ($\llbracket v \rrbracket > 1$) { $G' = Split(G, v)$; }
3 else { Let $G^*$ be the corresponding DL graph of $G$;
4     $G' = DL(G^*, v)$; $G' = Merge(G', v)$; }
```

In the above procedure, the split operation (Line 2) and the DL iteration followed by the merge operation (Line 4) both need only direct neighbor information of responsible node v. Note that in Line 3 node v does not know the overall mapping from DL graph to DL+ graph, it only needs to know the mapping of *local* physical neighbors. It is clear that there are one more node in G' than in G. Thus, the DLG technique can be applied to building DHTs based on arbitrary regular graphs with any base.

Since DL+ graphs all have their corresponding DL graphs, routing in a DL+ graph G is straightforward: Let G^* be the corresponding DL graph of G. The path P between any two physical nodes $u, v \in G$ always has a counterpart path P^* between $u^*, v^* \in G^*$, where u^* and v^* are logical nodes of u and v, respectively. Thus the routing in G can be directly conducted by emulating the routing in the corresponding DL graph G^*. The routing procedure in DL+ graphs is summarized as follows.

```
Procedure w.DL+_Routing (Source u, Destination v, Hops k)
// w is the current physical node where the message arrives.
// k means how many hops the message already traversed.
1 if ($w == v$) return;                    // routing is over
2 Arbitrarily choose one of $w$'s logical nodes $w'$;
// Send the message to the logical next-hop, which invokes the
// corresponding physical next-hop's DL+_routing
3 $w'.DL\_Routing(u, v, k)$;
```

As shown in the discussion of Fig. 2 "Examples of DL iterations", all the new nodes in the new DL graph have the same out-neighbors. Thus all the logical nodes of a physical node have the same out-neighbors. So, when a physical node receives a message, it can arbitrarily choose one of its logical nodes to run the DL_Routing algorithm and route to the logical next-hop, and correspondingly, the physical next-hop. And clearly, the physical destination v (in the DL+ graph) is merged by one or more logical nodes (in the DL graph), of which there must be one with the same identifier v. So, when the message arrives at the logical node v in the DL graph, it arrives at the physical destination v and the routing is over.

3.4 Analysis

The following Theorem 4 describes the in-degree and out-degree of a node in DL+ graphs.

Theorem 4. *Let graph G be a DL+ graph with base d. For all physical nodes $x \in V(G)$, the out-degree and in-degree, denoted as $\delta_G^+(x)$ and $\delta_G^-(x)$, satisfy respectively*

$$\delta_G^+(x) = d\,, \tag{9a}$$

$$1 \leq \delta_G^-(x) \leq 2d\,. \tag{9b}$$

And the average in-degree of all nodes in G is d.

Proof outline. (9a) can be directly derived from Theorem 2, and thus the average in-degree of all nodes in G is d. Next we prove the correctness of (9b) by analyzing the in-neighbors of each logical node held by x in the corresponding DL graph of G. We consider the following cases.

Case 1. There is some physical node $v \in \Gamma_G^-(x) \cup \Gamma_G^+(x)$ satisfying $|v| > |x|$. Let $v^* \in G^*$ be the logical node having the same identifier as $v \in G$. Suppose v^* is generated by $G_{i+1} = DL(G_i, v')$. We use v' and x to denote both the logical node and the physical node. Let $x = x_1 x_2 \ldots x_m$. Given any $\alpha \in \zeta(x_1)$, by Theorem 1 in G^* there is one logical node $y^* \in \Gamma_{G^*}^-(x)$ of the form $y^* = \alpha x_1 x_2 \ldots x_{m-1}$, or there are d logical nodes $y^* \in \Gamma_{G^*}^-(x)$ of the form $y^* = \alpha' \alpha x_1 x_2 \ldots x_{m-1}$ with $\alpha' \in \zeta(\alpha)$. In the first case, each $\alpha \in \zeta(x_1)$ has one corresponding physical node $y \in \Gamma_G^-(x)$. In the second case, each $\alpha \in \zeta(x_1)$ has two corresponding physical node $y \in \Gamma_G^-(x)$ and $\delta_G^-(x) \leq 2d$. Thus (9b) holds.

Case 2. There are no physical nodes $v \in \Gamma_G^-(x) \cup \Gamma_G^+(x)$ satisfying $|v| > |x|$. Similar to Case 1, we can prove (9b) still holds for this case and this completes the proof. □

Compared with Theorem 2, this theorem not only guarantees the DL+ graphs to be always d-out-regular, but also assures the in-degrees are bounded by twice the base. This in-degree upper bound is smaller than that in DL graphs (d^2) since different in-neighbors in DL graphs might be merged into one in-neighbor in DL+ graphs.

The following Theorem 5 describes the upper bound for the diameter of a DL+ graph.

Theorem 5. *Let graph G be a DL+ graph with base d and order N. Let the order and diameter of the initial graph G_0 be N_0 and D_0, respectively. Then the diameter of G satisfies*

$$D(G) < 2(\log_d N - \log_d N_0 + D_0 + 1)\,. \tag{10}$$

Since all the paths in a DL+ graph have their counterpart paths with equal lengths in the corresponding DL graph, this theorem can be directly derived by mapping a DL+ routing onto the corresponding DL routing.

Since the topologies of DLG-enabled DHTs are always DL+ graphs, by theorem 5 we see that the diameters of DLG-enabled DHTs reach the theoretical lower bound $\Omega(\log_d N)$ of constant-degree DHTs [3]. Moreover, we show in the case study on DK (at the end of Section 4.6) that *w.h.p.* its de facto diameter is much lower than the analytical result ($2\log_d N + 2$).

4 Building DHTs with DLG

In this section we introduce how to utilize our DLG technique to build DHTs based on different initial graphs.

4.1 Self-Stabilization on Node Joins

This subsection discusses the self-stabilization mechanism on node joins, which follows the traditional manner in many existing DHTs like [2], [7], [14], i.e., first routes to a random node to balance the topology, and then routes to the responsible node to finish the DLG transition.

At the beginning, the topology of a DLG-enabled DHT is a d-regular initial graph G_0 with order N_0 and diameter D_0. Then it evolves into a family of DL+ graphs as nodes join/leave. Similar to other DHT stabilization mechanisms, the processing for node joins in DLG-enabled DHTs can be divided into two phases, namely, (i) looking for a responsible node satisfying (8); and (ii) updating the routing tables of relevant neighbors.

1. Looking for responsible node

Suppose that a new node p joins the network. This phase performs two tasks to find a responsible node v.

First, node p sends a message targeted to a *surrogate node u* chosen uniformly at random in the network. This step is to achieve a balanced DHT topology. The discovery of surrogates is orthogonal to our DLG technique and will be discussed in later case studies (Section 4.6).

Second, node p invokes a JOIN message from u. During the routing of the JOIN message, suppose that it arrives at an intermediate node u'. If u' has a neighbor w satisfying $|w| < |u'|$, then the JOIN message will be forwarded to w; otherwise if u' has a neighbor w' satisfying $(|w'| = |u'|) \wedge ([\![w']\!] > [\![u']\!])$, then the JOIN message will be forwarded to w'. This procedure will continue until the JOIN message reaches a physical node v satisfying (8), which is the *responsible node* for this join event.

2. Updating the routing tables

This phase can be easily accomplished by performing a *DLG transition* ($G' = Split(G,v)$, or $G' = DL(G,v)$ followed by $G'' = Merge(G',v)$), where v is the responsible node that is found in the first phase. After the transition, there will be two new nodes in the DHT topology (i.e. the DL+ graph), one corresponding to v, and the other corresponding to the new node p.

4.2 Self-Stabilization on Node Leaves

In a dynamic network, nodes can leave at any time. Ideally the leaving node would notify its relevant neighbors, the case of which is referred to as *graceful leaves*. In contrast, sometimes nodes might leave without notifying any neighbors. This case is referred to as *ungraceful leaves*.

1. Graceful leaves

When a node p gracefully leaves the network, it will actively transfer the neighbor information in its routing table to other nodes, the procedure of which can be viewed as an inverse operation of node joins.

First, node p invokes a DEPART message. During the routing of the DEPART message, suppose that it arrives at an intermediate node u. If u has a neighbor w satisfying $|w| > |u|$ then the DEPART message is forwarded to w; otherwise if u has a neighbor w' satisfying $(|w'| = |u|) \wedge ([\![w']\!] < [\![u]\!])$, then the message is forwarded to w'. This procedure will continue until the DEPART message reaches the responsible node v, which has locally longest identifier or minimum order, i.e.,

$$\forall v' \in \Gamma_G^-(v) \cup \Gamma_G^+(v),\ |v| > |v'| \text{ or } (|v| = |v'|) \wedge ([\![v]\!] \le [\![v']\!]). \quad (11)$$

Second, node v hands over its logical nodes, as well as their in-/out-edges (i.e. routing table entries), to its sib-neighbor x. Note that the merge/split operations assure the existence of such sib-neighbors. Then node p hands over its logical nodes and their in-/out-edges to v and the change of the logical nodes' ownership will be noticed to relevant neighbors. Then p leaves.

Third, if the physical node x has d logical nodes $x^{(i)} = \alpha_i v_1 v_2 \ldots v_k$ with $i = 0,1,\cdots,d-1$ after the above handover operation, then the d logical nodes will turn into one logical node $v_1 v_2 \ldots v_k$. Afterwards, the physical node x satisfies $[\![x]\!] = 1$, and it has the same identifier with its logical node. Let the two DL graphs before and after this transition be G and G' respectively. Then we have $\Gamma_{G'}^-(x) = \cup \Gamma_G^-(x^{(i)})$, and $\Gamma_{G'}^+(x) = \cup \Gamma_G^+(x^{(i)})$.

Clearly the DHT topology would not be affected if each graceful leave in a succession occurs after its previous leave has already been handled. If two neighbors, say u and v ($u \to v$), occur graceful leaves in a very small time slot, then the first leave, say node v's leave, can be handled normally, and u's leave processing will be blocked until v's leave has already been processed. At this time u still can leave according to its will, but it needs to first choose a live in-neighbor to temporarily act as it until its leave processing is finished, in case that relevant event processing messages route across u. By this means we can handle the succession of graceful leaves of any two nodes. Similarly, using neighbor's neighbor, we can handle that of any specific nodes (at the price of more backups).

2. Ungraceful leaves

Nodes in DHTs might abruptly stop communicating due to a host of reasons. We classify as ungraceful leaves that fail to complete the above-mentioned operations required for graceful leaves. To handle ungraceful leaves, as in traditional approaches [3], [7], a simple backup mechanism is as follows.

A node u stores a backup of its routing table at each of its direct in-neighbors. These in-neighbors are sorted alphabetically and are in turn called primary secretary, second secretary, …, dth secretary of u. u periodically send ALIVE messages to its secretaries. Once messages have not been received for a certain period of time, the secretaries will consider u ungracefully leaves the DHT. The primary secretary is responsible for this situation and initiates and carries through all the operations on the behalf of u. In case that the primary secretary fails unexpectedly (e.g., it crashes shortly after u's crash), the second secretary periodically asks the primary secretary so that it will be able to find out the failure of the primary secretary. Then, the second secretary will take over the responsibility of u's failure recovery. By parity of reasoning, DHT can survive if at least one in-neighbor is alive.

The following Theorem 6 describes the processing cost per node join/leave in DLG-enabled DHTs (measured in terms of the number of hops the message traverses).

Theorem 6. *Let the order of a DLG-enabled DHT be N, and let the order and diameter of its initial d-regular graph be N_0 and D_0, respectively. The processing cost per join/leave event, denoted as T and S respectively, satisfies*

$$T < 3(\log_d N - \log_d N_0 + D_0) + d + 1. \quad (12a)$$

$$S < \log_d N - \log_d N_0 + D_0 + d - 1. \quad (12b)$$

And at most 3d nodes need to update their routing tables.

Proof outline. We divide the joining processing into two part: route to the surrogate (taking T_1 hops) and to the responsible node (taking T_2 hops). Clearly $T = T_1 + T_2$.

By Theorem 5, $T_1 < 2(\log_d N - \log_d N_0 + D_0 + 1)$. Clearly, $T_2 \le (|v| - |u|) + (\llbracket v \rrbracket - \llbracket u \rrbracket)$, $|v| - |u| \le \log_d N - \log_d N_0 + D_0$, and $\llbracket v \rrbracket - \llbracket u \rrbracket < \llbracket v \rrbracket \le d - 1$, thus $T_2 < \log_d N - \log_d N_0 + D_0 + d - 1$. Therefore (12a) holds. Similar to the join procedure, it can be inferred that for the leaving processing (12b) holds.

If a DL operation $G' = DL(G, v)$ followed by a merge operation $G' = merge(G', v)$ occurs, then the responsible node v satisfies $\llbracket v \rrbracket = 1$. Then at most $3d$ nodes need to update their routing tables. Similarly, if a split operation takes place, at most 3*d* nodes need to update their routing tables. The case for node leaves is similar to that for node joins. Thus Theorem 6 holds. □

From this theorem we can see that the processing cost for node leave is remarkably less than that for node join. This is mainly because the first part of join processing (routing to a surrogate), which is for achieving a more balanced DHT topology, takes a relatively high cost. We will show in Section 4.4 that this cost can be evicted while still bounding the imbalance to the same range as before.

4.3 Concurrency

Although concurrency is orthogonal to DLG, this subsection briefly discusses this issue for completeness. In DHTs many nodes might join/leave simultaneously, which cause temporary inaccurate information in routing tables and in turn cause errors in the routing and join/leave processing. To address this problem, a simple *atomic update mechanism* is adopted as follows. When a node joins/leaves the DHT, the routing tables of relevant nodes should be updated. Only when all updates are completed, are the new routing tables allowed to be used. During the period, the relevant JOIN/DEPART messages are suspended. Messages are resent after the update is finished.

4.4 Stabilization Cost vs. Topology Imbalance

T*opology imbalance* [3], [7] can be measured by the maximum difference of identifier lengths over all pairs of nodes. The imbalance of identifiers is closely related to the load balancing issues such as node/edge congestion and hot spot. As discussed in Section 4.1, in DLG-enabled DHTs a joining node first routes to a random node, in order to randomize the responsible node for the forthcoming DLG transition and to alleviate the topology imbalance. By Theorem 6, however, the cost of such balance-oriented processing is $O(\log_d N)$, which might be too expensive for highly dynamic environments where nodes frequently join/leave. Similar problems exist in almost all DHTs.

These problems are actually a tradeoff between the stabilization cost and topology imbalance, where higher cost results in more balanced topology and vice versa. In the rest of this paper, the join algorithm proposed in Section 4.1 will be referred to as *balanced join*, and this subsection will present a relevant but different algorithm named *fast join*, achieving a lower processing cost at the price of inducing some degree of imbalance.

Generally speaking, the fast join algorithm can be viewed as a reduced version of the balanced one, simply omitting the first phase of routing to a random node. That is, the new node would directly invoke the JOIN message from a nearby gateway node and the following processing is the same as the second phase.

Note that the first phase is not a part of DLG transitions and it is not indispensable for DL+ graphs. So, the DLG transition can always be conducted on the topologies with fast join and the result topologies are assured to be DL+ graphs. Thus the routing is assured (by Theorem 5) to be correct.

The following Theorem 7 describes the processing cost per node join/leave in the *fast join* algorithm (measured in terms of the number of hops the message traverses).

Theorem 7. *Let the order of a DLG-enabled DHT be N, and let the order and diameter of its initial d-regular graph be N_0 and D_0, respectively. Then the processing cost per node join in the fast join algorithm, denoted as T, satisfies*

$$T < \log_d N - \log_d N_0 + D_0 + d - 1. \quad (13)$$

And at most 3d nodes need to update their routing tables.

Since the processing of fast join is equal to that of the second phase of balanced join, the correctness of this theorem is straightforward.

This theorem shows that the bound for the cost in fast join is about 1/3 that in the balanced one. On the other hand, although practically the fast join algorithm brings more imbalance to the DHT topology, by Theorem 3 we can infer the difference between any two node identifier lengths u and v satisfies $\| |v| - |u| \| \le \log_d N - \log_d N_0 + D_0$, which shows that fast/balanced joins do have the same imbalance upper bound. This is because the possible topology set for the balanced join algorithm is in theory a superset of those for fast join. For example, it is possible that the randomly chosen responsible node in the balanced join algorithm is always the gateway node used in fast join, resulting a completely same processing in both algorithms. We will further evaluate the tradeoff between stabilization cost and topology imbalance through simulations in Section 5.2.

Topology imbalance affects the *load* balance property. Load balance is much more complicated and related to many aspects, e.g., distribution of user quires, resource distribution in resource space, and mapping from resources onto network *topology* which can be further divided into sub-mappings from resources onto keys and from keys onto *topology*. Generally speaking, imbalanced topology leads to imbalanced load distribution. However, as illustrated by the design of keys in DLG-enabled DHTs starting from $K(d,1)$ (in Section IV.F in the tech report [28]), mapping from keys onto topology is tightly coupled with the *particular* underlying graph that is not concerned by DLG. We will further study the impact of topology imbalance on load balance in our future work.

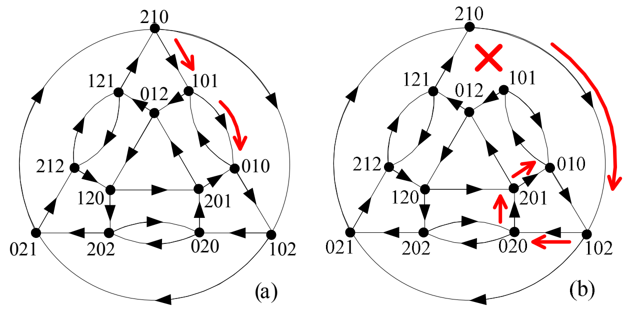


Fig. 4. Robust routing from 210 to 010.

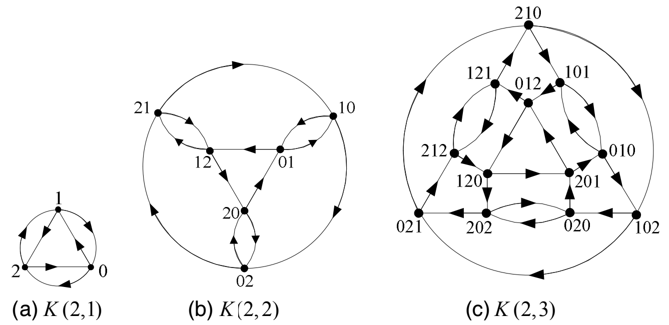


Fig. 5. Examples of line graph iterations.

4.5 Routing under Churn

Frequent node joins/leaves might induce many suspensions and consequently *churn* [25]. As discussed in Section 4.2, we could use traditional backup techniques for routing tables to support routing during stabilization. If the processing of a join/leave event is not affected by other events, then this event will not hurt the topology. This depends on how many events occur in the processing path and how many backups are held. Clearly, if backups are inefficient, then churn might hurt the DHT topology and consequently affect the routing.

To address this problem, this subsection presents a novel DLG-based mechanism that supports efficient routing during stabilization and effectively reduces the affects of churn. Specifically, we add a *detour-around* mechanism to the basic DLG routing, handling the situation where a neighbor has been identified as the next hop but does not respond since it has failed or has been blocked due to other nodes' join/leave.

Let the current DHT topology, its corresponding DL graph, and the initial *d*-regular graph be *G*, G', and G_0, respectively. The basic idea of our detour-around mechanism is to bypass the faulty neighbor and select an *alternative node* in the DL graph G' to forward the message.

As discussed in Section 2, the routing between any two nodes in a DL graph contains two phases. In the first phase it is determined by the initial *d*-regular graph G_0, where there are *d* paths between any pair of nodes since the connectivity is *d*. Based on the knowledge about G_0, the current node can directly compute the alternative node. Next we focus on the second phase where the routing is determined by the target.

Suppose that the message arrives at node *c*, and the normal routing path without node failure should be $\cdots \to c \to v \to x = x_1x_2...x_n \to \cdots$. Suppose that at this moment node *v* fails. Then two cases need to be considered.

If $|v| \ge |x|$, then by Theorem 1, given $\beta \in \zeta(x_1)$, node *x* either has an in-neighbor of the form $\beta x_1x_2...x_{n-1}$, or has *d* in-neighbors of the form $\beta'\beta x_1x_2...x_{n-1}$ with $\beta' \in \zeta(\beta)$. In this case, node *c* can bypass *v* by selecting any two alternate nodes of the form $y = \beta x_1...x_{n-1}$ and $y' = \beta'\beta x_1...x_{n-1}$, where $\beta \in \zeta(x_1)$, $\beta' \in \zeta(\beta)$, and *y* and y' are not held by the faulty (physical) node. Note that node *c* has no direct links to *y* or y', thus it would not know which of them exists in the network. However, Theorem 1 assures one of them must exist.

Otherwise if $|v| < |x|$, then the faulty node *v* is the only in-neighbor of *x*. In this case, after a DL iteration $G_{i+1} = DL(G_i, v^{(i)})$ we can add to each new node $u \circ v^{(i)}$ a temporary link of the form $[w, u \circ v^{(i)}]$ with $u \in \Gamma^-_{G_i}(v^{(i)})$, $w \in \Gamma^-_{G_i}(v^{(i)})$ and $w \ne u$. The temporary links will be deleted when the in-degree of $u \circ v^{(i)}$ exceeds one. Note that by this means the out-degree of *w* may reach $d+1$ temporarily. The following procedure is the same as in the first case. Fig. 4 shows an example of the robust routing from node 210 to node 010. In (a) there is no failure and the routing can be conducted as usual; while in (b) the path between 210 and 101 fails and the detour-around mechanism is taken: $210 \to 102 \to 020 \to 201 \to 010$.

4.6 Case Study

Given an arbitrary initial *d*-regular graph G_0, a DHT backbone can be easily obtained by adopting our DLG technique. As discussed before, the remaining tasks are to

1) describe initial graph G_0, i.e., to define the *in-letter set* $\zeta(\alpha)$ and the *out-letter set* $\psi(\alpha)$ for each $\alpha \in G_0$;
2) design the mechanism to find surrogate nodes for joining events (See Section 4.1); and
3) design the mapping policy from resources onto nodes. Next we introduce four DLG-enabled DHTs based on different graphs, namely, Kautz, de Bruijn, butterfly and hypertree graphs.

1. DLG-Kautz (DK)

We first show how to build DLG-Kautz (shortly DK), a DLG-enabled, Kautz graph-based DHT.

Let $Z_d = \{0,1,2,\cdots,d-1\}$ be an alphabet of *d* letters. The *Kautz string space* $KS(d,k)$ is defined as $KS(d,k) = \{x_1x_2\cdots x_k \mid x_i \in Z_{d+1}, x_i \ne x_{i+1}, i<k\}$. A Kautz graph [7] with diameter *D* and base *d*, denoted as $K(d,D)$, is defined by its node set and edge set: $V(K(d,D)) = KS(d,D)$, $E(K(d,D)) = \{[x_1x_2\cdots x_D, x_2\cdots x_D\alpha] \mid \alpha \in Z_{d+1}, \alpha \ne x_D\}$. Examples of Kautz graphs $K(2,1)$, $K(2,2)$, and $K(2,3)$ are shown in Fig. 5(a), (b), and (c), respectively. For the sake of simplicity, here we only consider the case of initial graph $G_0 = K(d,1)$.

(i) The *in-letter set* and *out-letter set* of each $\alpha \in G_0$ are respectively $\zeta(\alpha) = \psi(\alpha) = \{\beta \mid \beta \in Z_{d+1}, \beta \ne \alpha\}$.

(ii) To find a surrogate, we use the *KHash* algorithm [7] on the joining node's IP address to get a target string *s*; and the surrogate is the node onto which *s* is mapped.

(iii) Mapping from resources onto nodes can be divided into mappings from resources onto keys and from keys onto nodes. Any resource can be consistently and uniformly mapped to a Kautz string by using the *KHash* algorithm [7]. However, the mapping from a Kautz string to a node cannot adopt prefix (or suffix) matching policies [7], [12] since they cannot guarantee the consistency. Introduction of mapping from keys onto nodes is omitted here due to lack of space. See our tech report [28].

We prove (in our tech report [28]) that when the number of nodes $\geq d^{2m}$, the DK topology could be approximated to Kautz graph $K(d,m)$. Thus we can say that the main features of the initial Kautz graph are preserved after a series of DLG transitions. We will further discuss the properties of DLG-Kautz through analysis and evaluations in Section 4.6.5 and 5, respectively.

2. DLG-de Bruijn (DdB)

A de Bruijn graph [6] with diameter D and base d, denoted as $B(d,D)$, is defined by its node set and edge set: $V(B(d,D)) = Z_d^D$, $E(B(d,D)) = \{[x_1x_2\cdots x_D, x_2\cdots x_D\alpha] \mid \alpha \in Z_d\}$.

Similar to DLG-Kautz, next we show how to build DLG-de Bruijn (shortly DdB), a DLG-enabled, de Bruijn graph-based DHT, by describing initial graph $G_0 = B(d,1)$.

The *in-letter set* and *out-letter set* of each $\alpha \in G_0$ are $\zeta(\alpha) = \psi(\alpha) = \{\beta \mid \beta \in Z_d\}$.

The mechanisms of surrogate node discovery and consistent resource mapping are orthogonal to our DLG technique and can be designed in a way similar to those in DK. Here we omit these mechanisms due to lack of space. Fig. 6 shows three examples of the DdB topologies as nodes join/leave the network.

3. DLG-Butterfly (DBF)

A (*k*,*r*)-butterfly [14] is a graph with $n = kr^k$ nodes, where *k* and *r* are diameter and degree of the graph, respectively. Each node is of the form $(x_0x_1...x_{k-1};i)$ with $0 \leq i \leq k-1$ (*i* is called *level*) and $0 \leq x_0, x_1, ...x_{k-1} \leq r-1$. For each node $(x_0x_1...x_{k-1};i)$, there is an edge to all nodes of the form $(x_0x_1...x_iyx_{i+2}...x_{k-1};i+1)$ when $i \neq k-1$ and $(yx_1...x_{k-1};0)$ when $i = k-1$. An Example of the (2,2)-butterfly graph is shown in Fig. 7(a).

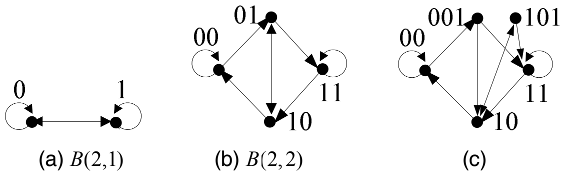
In the basic definition, diameter *k* decides both the number of levels and the length of node identifiers. Since the number of levels is hard to extend, as in other butterfly-based DHTs such as Ulysses [14], we decouple the number of levels (*k*) and the length of node identifiers (*m*), and extend the length *m* only. By this means, a *generalized* butterfly graph can be represented as $GBF(k,r,m)$, where *k*, *r*, *m* are the number of levels, the degree, and the length of node identifiers, respectively. An example of the generalized butterfly $GBF(2,2,1)$ is shown in Fig. 7(b), which is *isomorphic* to Fig. 7(a). Note that the identifiers "0, 1, 2, 3" of level 0 and level 1 in Fig. 7(b) are mapped from "00, 01, 10, 11" of corresponding levels in Fig. 7(a) according to the *unified description* mechanism.

Next we show how to build DLG-Butterfly (shortly DBF), a DLG-enabled, butterfly graph-based DHT, by describing the initial graph $G_0 = GBF(2,2,1)$.

The *in-letter sets* and *out-letter sets* of each $\alpha \in G_0$ are respectively:

$$\zeta(\alpha,0) = \{(\beta,1) \mid \beta = \alpha\} \cup \{(\beta,1) \mid \beta = (\alpha+2) \bmod 4\},$$

$$\psi(\alpha,0) = \{(\beta,1) \mid \beta = \alpha\} \cup \{(\beta,1) \mid \beta = \alpha+1 \text{ and } \alpha \text{ is even}\} \cup \{(\beta,1) \mid \beta = \alpha-1 \text{ and } \alpha \text{ is odd}\},$$



(a) $B(2,1)$ (b) $B(2,2)$ (c)

Fig. 6. Examples of DdB topologies.

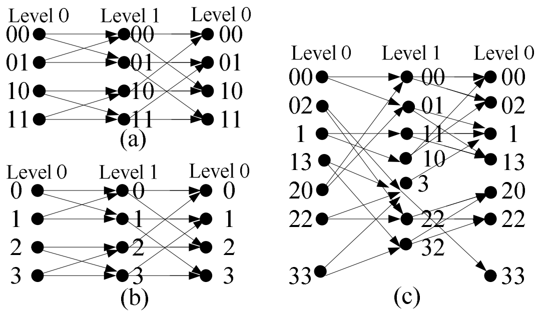


Fig. 7. Example of DBF topologies.

$$\zeta(\alpha,1) = \{(\beta,0) \mid \beta = \alpha\} \cup \{(\beta,0) \mid \beta = \alpha+1 \text{ and } \alpha \text{ is even}\} \cup \{(\beta,0) \mid \beta = \alpha-1 \text{ and } \alpha \text{ is odd}\},$$

$$\psi(\alpha,1) = \{(\beta,0) \mid \beta = \alpha\} \cup \{(\beta,0) \mid \beta = (\alpha+2) \bmod 4\}.$$

Again we omit the design of surrogate node discovery and consistent resource mapping due to lack of space. Fig. 7 (b) and (c) show two examples of the DBF topologies.

4. DLG-Hypertree (DH)

A hypertree graph [26] is of *n* leaves and depth $\log_d N$, where *d* and *N* are the base and maximum number of nodes. The Owlet Research Group [27] implements a DLG-enabled, hypertree-based DHT, DLG-Hypertree (shortly DH), by using our DLG technique. Space precludes further discussion and interested readers are referred to their homepage [27].

5. Comparison

As discussed above, the DLG technique can be used to design many different DHTs. The following Theorem 8 compares the diameters of all the DLG-enabled DHTs.

Theorem 8. *Given the base d, DK has the lowest diameter upper bound among all DLG-enabled DHTs.*

Proof outline. The diameter of DLG-enabled DHTs satisfies $D(G) < 2(\log_d N - \log_d N_0 + D_0 + 1)$, where d, N_0 and D_0 are the base, order and diameter of the initial graph G_0, and N is the current network size. By the Moore bound [11], N satisfies $N \leq 1 + d + d^2 + \cdots + d^D = (d^{D+1}-1)/d-1$. Therefore, $D_0 - \log_d N_0 \geq 1 - \log_d(d+1)$. The minimum value of $D_0 - \log_d N_0$ can be attained ONLY by the *complete symmetric graph* where each node is adjacent to all others, i.e., Kautz graph $K(d,1)$. □

Since we use $K(d,1)$ as the initial graph, it is easy to infer that DK's diameter is bounded by $2\log_d N + 2$. However, for DK with balanced join, the *KHash* algorithm [7] assures a uniform distribution of surrogate nodes *u* in the network, which consequently assures a uniform distribution of responsible nodes *v* (See Section 4.1). Thus it is very likely that DK's topology is always relatively balanced [11], i.e., the difference between the maximum and minimum identifier lengths is small. If the difference is no more than 1, then the topology is equivalent to a PLG

Kautz graph [11] with diameter $\lceil \log_d N \rceil$ (See the technical report [28]). Evaluations in Section 5.1 show that the de facto DK diameter is no more than $\lceil \log_d N \rceil$ +1.

It is well known that Kautz graphs have the lowest diameter $\log_d N - \log_d(1+1/d)$ achieving the Moore bound [10]; while to our knowledge with the same routing table size 2*d* DK has the lowest *rigorous* (as opposed to *asymptotic*) diameter upper bound $2\log_d N + 2$ as well as the lowest de facto diameter $\le \lceil \log_d N \rceil + 1$. So, we conclude that our DLG technique preserves the topology feature of Kautz graphs while being applied to designing DK.

Note that although DK has the optimal diameter, it does not mean that DK is always the best choice among all the DLG-enabled DHTs. This is because there are many different concerns for other topology properties such as congestion [7], robustness [14] and churn [25].

5 Evaluation

In this section, we choose DK as a model to evaluate DLG and compare DK with other constant-degree DHTs. Note that, however, since the main purpose of this section is to prove the effectiveness of DLG, we omit the evaluations of DK-specific properties.

In Section 5.1 we show that with the same base *d* (meaning the same average routing table of size 2*d*), the maximum and average routing path lengths are remarkably less for DK than for other constant-degree DHTs.

Section 5.2 evaluates the stabilization cost and topology imbalance of DK with balanced/fast join algorithms, respectively, which are correspondingly referred to as *balanced DK* (**B-DK**) and *fast DK* (**F-DK**).

In Section 5.3, we show that the DK routing is robust under churn, even when a certain percentage of nodes become faulty. By following the detour-around mechanism, when as much as 10% nodes are faulty, most of routing messages targeted to healthy nodes can still reach the destination without failure.

The evaluations are conducted by modifying the Tapestry simulator [15]. In our evaluations, the number of nodes is varied from 256 up to 1M, and the experiment for each property is repeated at least 1,000 times.

5.1 Routing Path Length

We evaluate the maximum and average routing path lengths (measured in terms of overlay hops) of balanced DK, as a function of the number of nodes, for different bases $d = 4$ and $d = 16$. In each experiment we randomly choose two nodes and a message is routed by using the proposed routing algorithm.

The results are plotted in Fig. 8, where the maximum and average routing path length of DK are denoted as DK (avg) and DK (max), respectively. We compare them with other high performance DHTs including CAN, Koorde and FissionE. Note that although FissionE can only have a fixed base $d = 2$, it is also included in Fig. 8(a) for comparison.

Curves for all DHTs except CAN look "linear" since the *x*-axis is in log scale. (By [3], the average path length *L* of CAN with order *N* and base *d* satisfies $L = (d/4)(N^{1/d})$.)

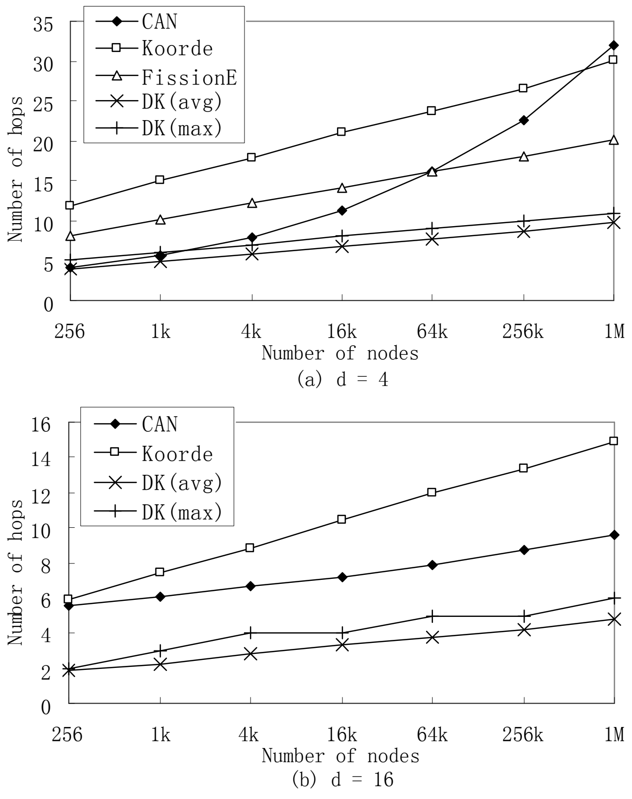


Fig. 8. Maximum and average routing path length.

From Fig. 8 we conclude both the maximum and average routing path lengths of DK are remarkably less than those of others. Koorde performs worse since its *imaginary nodes* [6] consume more hops.

Note that although we cannot contain in Fig. 8 all existing DHTs, to the best of our knowledge DK does have the lowest routing path lengths (under condition of the same routing table size) in the literature. Basically, this holds because DLG enable DK to inherit the feature of optimal routing path length of Kautz graphs whose diameters achieve the Moore bound [11].

We evaluate the probability distribution of path lengths of DK with bases $d = 4$ and $d = 16$, for a fixed network size of 1 million nodes. The results are shown in Fig. 9. We see that DK has a low variance both for $d = 4$ and for $d = 16$, indicating the lengths of most routing paths are very close to the mean value.

5.2 Stabilization Cost and Topology Imbalance

We evaluate the average message cost per join/leave event in balanced/fast join algorithms, which are correspondingly referred to as *balanced DK* (**B-DK**) and *fast DK* (**F-DK**), for a fixed network size of 1 million nodes, and compare them with Koorde that has the lowest maintenance cost among other DHTs according to the authors. There are 100 gateway nodes and a new node randomly chooses a gateway to start the processing. The base is set to $d = 4$ and $d = 16$ respectively. Since the results obtained with the two values of *d* have similar characters, we show results in Fig. 10 corresponding to $d = 16$ only.

From this figure we can see that the average cost of

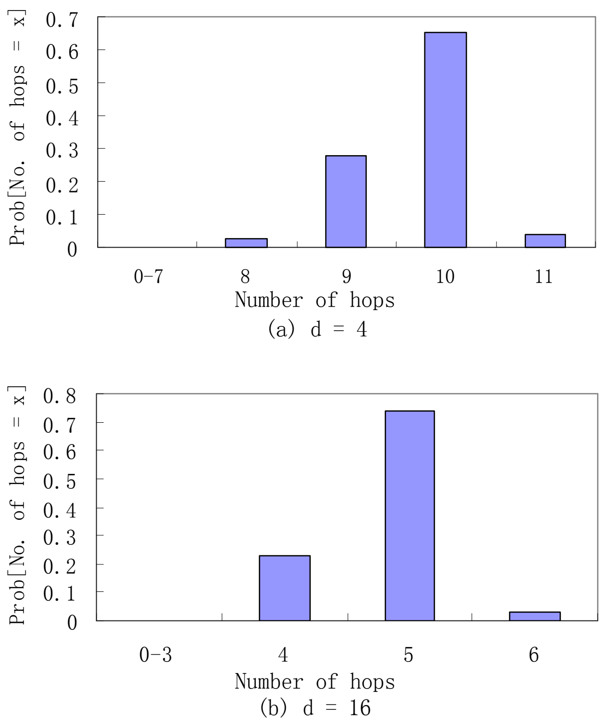


Fig. 9. Routing path length distribution.

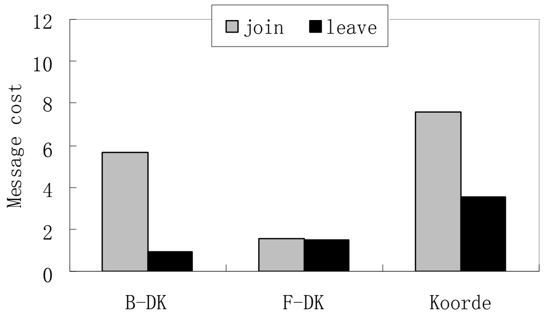


Fig. 10. Stabilization cost.

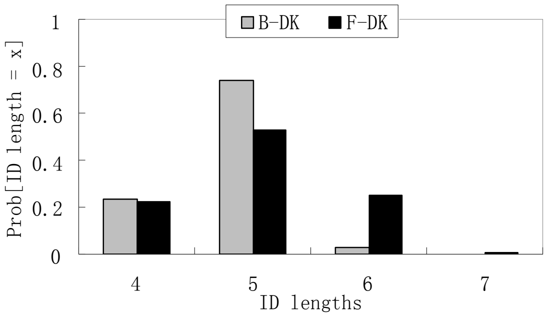


Fig. 11. Identifier length distribution.

join events in F-DK is remarkably less than that in B-DK, while the average cost of leave events in F-DK is a little more than that in B-DK. Thus, F-DK might be more attractive in highly dynamic networks where nodes frequently join/leave. Clearly, both F-DK and B-DK have a less maintenance cost than Koorde.

As discussed in Section 4.4, the fast join might induce more imbalance to the topologies. Thus, we evaluate the balancing property measured in terms of the distribution of node identifier lengths in B-DK and F-DK. The results are shown in Fig. 11.

First, both B-DK and F-DK have a small difference between the maximum identifier lengths and the minimum ones. The maximum zone held by a B-DK (resp. F-DK) node is larger than the minimum at most by a factor of 2 (resp. 3), both smaller than $\log_d N = 5$. Since the identifier of a DK node cannot be a suffix of another, one more digit of a node identifier means d times smaller the Kautz space held by that node.

Second, the variance of node identifier lengths for B-DK is remarkably lower than that for F-DK, thus we conclude that B-DK has better balancing property. Therefore, the selection of the two join algorithms can be viewed as a tradeoff between balanced topology and stabilization cost, where the balanced join algorithm achieves a more balanced topology and the fast join algorithm achieves a faster stabilization but less stabilization cost.

5.3 Routing under Churn

A routing message might not be responded since its next hop has failed or has been blocked due to other nodes' join/leave events. This subsection evaluates the DK routing under churn. Note that in order to highlight the effectiveness of the *detour-around* mechanism, here we utilize the traditional backup mechanism (discussed in Section 4.2) only for DHT topology maintenance but not for normal DHT routing.

In our experiments, we treat both failed nodes and blocked ones as *faulty* and simulate network churn by choosing faulty nodes randomly and uniformly from the DK network of 1 million nodes. The percentage of faulty nodes is varied from 0 to 10%, and the base is set to $d = 4$. Fig. 12 shows the percentage of messages encountering a failure during the routing without the detour-around mechanism, as a function of the percentage of faulty nodes. The results are computed by simulating 10,000 messages targeted to the healthy nodes that are randomly chosen from the network. From this figure we can see that the probability that a message encounters a failure on its path increases quickly with the percentage of faulty nodes. Thus, it is likely that messages need to retry many times in a highly dynamic environment, which may result in unacceptable routing delay or even failures.

Clearly, DHTs with higher base (e.g., $d = 16$) are more robust. However, the higher base results in more bandwidth consumption for periodic ping (keep-alive messages), which further incurs more churns. Detailed analysis on the influence of DK's base on churn will be studied in our future work.

As discussed in Section 4.5, the detour-around mechanism can help to bypass the faulty nodes and remarkably alleviate this problem. Fig. 13 plots the percentage of failed routing while following the detour-around mechanism, as a function of the percentage of faulty nodes. From this figure we could see that most routing messages can bypass the faulty nodes and successfully reach the

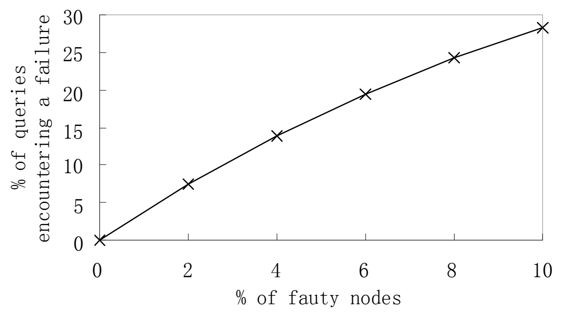


Fig. 12. Probability of encountering a failure.

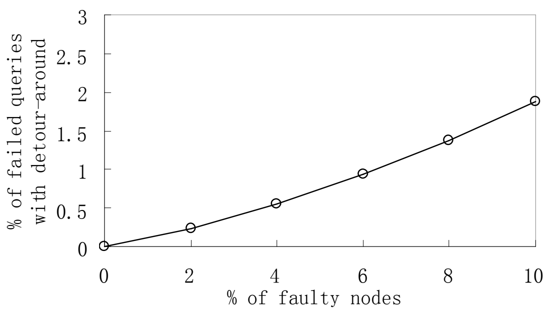


Fig. 13. Percentage of failed routing with detour-around mechanism.

destination. This takes effect as long as there is at least one healthy alternate node. The (less than 2%) failure happens when all the alternate nodes are faulty and waiting for stabilization. Thus, we conclude that DK achieves robust routing by simply using the detour-around mechanism.

6 Related Work

6.1 Line Graph Iteration

Our DLG technique is inspired by the novel technique of *line graph* iterations. Let the initial graph G_0 be a d-regular graph. In the line graph [10] of G_i, denoted as $G_{i+1} = L(G_i)$, $i = 0,1,2,\cdots$, each node represents an edge of G_i, i.e., $V(G_{i+1}) = \{uv \mid [u,v] \in E(G_i)\}$; and a node uv is adjacent to a node wz if and only if $v = w$, i.e., $[uv, wz] \in E(G_{i+1})$ when $[u,v]$ is adjacent to $[w,z]$ in G_i.

LG iterations preserve the main feature of the initial graph such as degree, diameter and routing algorithm. But the series of graphs generated by LG iterations cannot accommodate arbitrary number of nodes. Many variations, mainly including partial line graphs (PLG) [11], necklaces [12], and factorization [13], were proposed to address this problem. For instance, let $G = (V,E)$ be a graph. Let $E' \subseteq E$ be a subset of edges which are adjacent to all nodes of G. The PLG technique generates a new graph $G' = PL(G,E')$ where the nodes in G' represent the edges of E', and a node uv is adjacent to the nodes $v'w$ for each out-neighbor w of v in G, where $v' = v$ if $[v,w] \in E'$, or v' is any other in-neighbor of w.

The LG iteration and its variations have been studied and applied as a universal approach to designing interconnection networks (e.g., multiprocessor networks). However, they are all centralized algorithms requiring global information and centralized control, thus cannot be applied to DHTs. Our DLG technique shares the idea of edge-node transition with LG/PLG techniques. In essence, however, DLG is more than a distributed version of LG/PLG and has its distinct theory, applications, and designs of maintenance mechanism and routing algorithm, totally different from those of LG/PLG.

6.2 Constant-degree DHTs

CAN [3] simulates a d-dimensional Cartesian coordinate space with the degree of $2d$. Each node in CAN corresponds to a zone. Zones split or merge when nodes join/leave. CAN utilizes a bit-correct routing algorithm with diameter $O(dN^{1/d})$.

Koorde [6] uses the de Bruijn graph to achieve a diameter of $O(\log N)$ with two neighbors per node, or a diameter of $O(\log N / \log\log N)$ with $O(\log N)$ neighbors per node. Koorde uses the immediate real predecessor to simulate nodes that are not active on the ring. Since Koorde might travel several real hops to simulate an imaginary hop, the diameter bound can only be assured in the expectation in the big Omega (Ω) notation.

FissionE [7] utilizes Kautz graph $K(2,D)$ as its initial topology. It uses a CAN-style, space partitioning method to add new nodes to the overlay. The routing algorithm in FissionE is similar to that in static Kautz graphs. FissionE has an average degree of 4 and a diameter of $O(\log_2 N)$.

Guo, et al [8] proposed Moore, a DHT-based Peer-to-Peer network that is built based on arbitrary Kautz graph $K(d,D)$. Moore designs the *incomplete Kautz (IK) digraph* for its topology maintenance. However, since the IK digraph requires centralized control and global information, Moore is not a practical scheme. BAKE [17] presents the balanced Kautz tree to maintain the topology of a Kautz graph-based DHT, which suffers from similar problems with Moore [8].

To address the problem of Moore and BAKE, SKY [18] utilizes a decentralized technique called *distributed Kautz* (*D-Kautz*) graphs for dynamic topology stabilization. D-Kautz is ONLY a customized technique specific to Kautz graphs and cannot be applied to other graphs. In contrast, the DLG technique proposed in this paper is a universal technique which can be applied to arbitrary regular graphs with any base. So, the DLG technique could be viewed as a generalization of D-Kautz.

Hypertree [26] uses a tree for naming nodes and form a Plaxton [15] mesh-like structure for connections between nodes. It ensures the diameter is bounded by $\log_d N$, but it can only provide a logarithmic degree bound $(d-1)\lceil \log_d N \rceil$ in contrast to DK's constant out-degree d and rigorous in-degree bound $2d$.

Lupu, et al [29] utilized the abstract algebra Cayley graph to compare various properties of different DHTs. They also created a framework by which a designer can implement and simulate easily different network structures. This work is complementary to our DLG technique: DLG aims to design new DHTs preserving the properties of the underlying (regular) graphs, which can be compared with existing DHTs and implemented in the Cayley graph-based framework.

7 Conclusion

This paper presents a universal DHT topology maintenance technique called DLG. Basically, the new topologies are obtained from old ones by iteratively turning some of the edges into new nodes and redefining new edges between them. Thus, the new topologies keep the structure of those they proceed from. In the future, we plan to extend DLG to a **black box**: The input is some kind of description (e.g., XML) of any regular graph, and the output is a DHT backbone software package satisfying some special requirements.

References

[1] I. Stoica, R. Morris, D. L. Nowell, et al, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," IEEE/ACM Trans. Networking, vol. 11, no. 1, pp. 17-32, Feb. 2003.

[2] S. Ratnasamy, P. Francis, M. Handley, et al, "A Scalable Content Addressable Network," Proc. ACM SIGCOMM '01, pp. 149-160, 2001.

[3] 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-SAC, vol. 22, no. 1, pp. 151-163, Jan. 2004.

[4] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: A Scalable and Dynamic Emulation of the Butterfly," Proc. ACM Symp. Principles of Distributed Computing (PODC '02), pp. 183-192, 2002.

[5] P. Fraigniaud, P. Gauron, "D2B: A De Bruijn Based Content Addressable Network," Theor. Comput. Sci., vol. 355, no.1, pp. 65-79, Jan. 2006.

[6] F. Kaashoek and D. Karger, "Koorde: A Simple Degree-optimal Distributed Hash Table," Proc. IPTPS '03, 98-107, 2003.

[7] D. Li, X. Lu, and J. Wu, "FissionE: A Scalable Constant Degree and Low Congestion DHT Scheme Based on Kautz Graphs," Proc. IEEE INFOCOM '05, pp. 1677-1688, 2005.

[8] D. Guo, J. Wu, H. Chen, and X. Luo, "Moore: An Extendable Peer-to-Peer Network Based on Incomplete Kautz Digraph with Constant Degree," Proc. IEEE INFOCOM '07, pp. 821-829, 2007.

[9] H. Shen, C. Xu, and G. Chen, "Cycloid: A Scalable Constant-Degree P2P Overlay Network," Perform. Eval., vol. 63, no.3, pp. 195–216, 2005.

[10] M. A. Fiol, J. L. A. Yebra, and I. Alegre, "Line digraph iterations and the (d, k) digraph problem," IEEE Trans. Computers, vol. 33, no. 5, pp. 400–403, May 1984.

[11] M. A. Fiol and A. S. Llado, "The Partial Line Digraph Technique in the Design of Large Interconnection Networks," IEEE Trans. Computers, vol. 41, no. 7, pp. 848–857, Jul. 1992.

[12] P.Tvrdik, "Kautz Necklaces," Technical Report 94-08, LIP ENSL, 69364 Lyon, France, Mar. 1994.

[13] P.Tvrdik, "Factoring and Scaling Kautz Digraphs," Technical Report 94-15, LIP ENSL, 69364 Lyon, France, Apr. 1994.

[14] A. Kumar, S. Merugu, J. Xu and X. Yu, "Ulysses: A Robust, Low-diameter, Low-latency Peer-to-Peer Network," Proc. IEEE International Conference on Network Protocols (ICNP '03), pp. 258-267, 2003.

[15] B.Y. Zhao, L. Huang, J. Stribling, et al, "Tapestry: A Resilient Global-Scale Overlay for Service Deployment," IEEE J. Selected Areas in Comm., vol. 22, no. 1, pp. 41-53, Jan. 2004.

[16] Y. Zhang, "DLG-Kautz Implementation," Technical Report, Apr. 2010, available at http://www.kylinx.com/Papers/DLG_impl.pdf.

[17] D. Guo, H. Chen., Y. Liu, and X. Li, "BAKE: A Balanced Kautz Tree Structure for Peer-to-Peer Networks," Proc. of IEEE INFOCOM (Mini-symposium), 2008.

[18] Y. Zhang, X. Lu, and D. Li, "SKY: Efficient Peer-to-Peer Networks Based on Distributed Kautz Graphs," Science in China Series F, vol. 52, no. 4, pp. 588–601, Apr. 2009.

[19] G.S. Manku, "Routing Networks for Distributed Hash Tables," Proc. ACM Symp. Principles of Distributed Computing (PODC '03), pp. 133-142, 2003.

[20] D. Loguinov, J. Casas and X. Wang, "Graph-Theoretic Analysis of Structured Peer-to-Peer Systems: Routing Distances and Fault Resilience," ACM Trans. Networking, vol. 13, no. 5, Oct. 2005.

[21] M. Naor and U Wieder, "Novel Architectures for P2P Applications: The Continuous-Discrete Approach," ACM Trans. Algo., vol. 3, no. 3, 2007.

[22] M. Adler, E. Halperin, R. M. Karp, and V. V. Vazirani, "A Stochastic Process on the Hypercube with Applications to Peer-to-Peer Networks," Proc. ACM Symposium on Theory of Computing (STOC '03), pp. 575–584, 2003.

[23] X. Wang and D. Loguinov, "Load-Balancing Performance of Consistent Hashing: Asymptotic Analysis of Random Node Join," ACM Trans. Networking, vol. 15, no. 4, pp. 892–905, Aug. 2007.

[24] K. Kenthapadi and G. Manku, "Decentralized Algorithms Using Both Local and Random Probes for P2P Load Balancing," Proc. ACM SPAA '05, pp. 135–144, 2005.

[25] G. S. Manku, M. Bara and P. Raghavan, "Symphony: Distributed Hashing in a Small World," Proc. USITS '03, Mar. 2003.

[26] S.i Dolev and R. I. Kat, "HyperTree for self-stabilizing peer-to-peer systems," Distrib. Comput., vol. 20, no. 5, 375–388, 2008.

[27] http://owlet-code.sourceforge.net/

[28] Y. Zhang. "Distributed Line Graphs: A Universal Technique for Designing DHTs Based on Arbitrary Regular Graphs," Technical Report NUDT-CS-20101117, NUDT, Nov. 2010, http://www.kylinx.com/Papers/DLG_tech.pdf.

[29] M. Lupu, B. C. Ooi, Y. C. Tay: Paths to Stardom: Calibrating the Potential of a Peer-based Data Management System. ACM SIGMOD, Vancouver, 2008.

Yiming Zhang received the BSc degree and M.Sc. degree in Mechanics Engineering in 2001 and 2003, and the Ph.D. degree in Computer Science in 2008, from National University of Defense Technology (NUDT), Changsha, Hunan, China. He is now an assistant Professor in the National Laboratory for Parallel and Distributed Processing, NUDT. His research interests include distributed computing, peer-to-peer computing, and operating systems. He received the China Computer Federation (CCF) Best Dissertation Award in 2011 and the HP Distinguished Chinese Student Scholarship in 2008. Dr Zhang's current research is primarily sponsored by the National Natural Science Foundation of China (NSFC).

Ling Liu is a full Professor in College of Computing, Georgia Institute of Technology. Her research focus is on performance, availability, security, privacy, and energy efficiency. Dr. Liu has published over 250 International journal and conference articles in the areas of databases, data engineering, and distributed computing systems. Dr. Liu served on the editorial board of IEEE Transactions on Knowledge and Data Engineering (TKDE) and International Journal of Very Large Databases (VLDBJ) from 2004 - 2008. She is currently on the editorial board of several international journals, including Distributed and Parallel Databases (DAPD, Springer), IEEE Transactions on Service Computing (TSC), and Wireless Network (WINET). Dr Liu's current research is primarily sponsored by NSF, IBM, and Intel.