FISSIONE: A Scalable Constant Degree and Low Congestion DHT Scheme Based on Kautz Graphs

Dongsheng Li, Xicheng Lu, and Jie Wu

***Abstract*—The distributed hash table (DHT) scheme has become the core component of many large-scale peer-to-peer networks. Degree, diameter, and congestion are important measures of DHT schemes. Many proposed DHT schemes are based on traditional interconnection topologies, one being the Kautz graph, which is a static topology with many good properties such as optimal diameter, optimal fault-tolerance, and low congestion. In this paper, we propose FISSIONE: the first effective DHT scheme based on Kautz graphs. FISSIONE is constant degree, $O(\log N)$ diameter, and $(1+o(1))$-*congestion-free*. FISSIONE shows that a DHT scheme with constant degree and constant congestion can still achieve $O(\log N)$ diameter, which is better than the lower bound $\Omega(N^{1/d})$ conjectured before. The average degree of FISSIONE is 4, the diameter is less than $2\log N$, and the maintenance message cost is less than $3\log N$. The average routing path length is about $\log N$ and is shorter than CAN or Koorde with the same degree when the peer-to-peer network is large-scale. FISSIONE can achieve good load balance, high performance, and low congestion and these properties are carefully evaluated by formal proofs or simulations in the paper.**

***Index Terms*—Peer-to-peer networks, distributed hash table (DHT), Kautz graph, congestion-free.**

I. Introduction and Related Work

In recent years, peer-to-peer (P2P) computing has attracted significant attention from both industry and academic research [1], [2]. Applications of peer-to-peer networks vary among file sharing, persistent data storage, cooperative web-caching, DNS, and application level multicast. Many peer-to-peer systems have been deployed on the Internet, and some of them have become popular Internet applications.

The core component of many P2P systems is a distributed hash table (DHT) scheme [3], [4] that uses a hash-table-like interface to publish and look up data objects. In DHT schemes, the objects are hashed into a namespace, and each peer is assigned a small segment of the namespace. When peers join or depart, the responsibility is reassigned among the peers to maintain the hash table structure. DHT schemes have attracted significant attention in academic research for their desirable characteristics, such as scalability, robustness, adaptability, self-management, and generality.

Dongsheng Li and Xicheng Lu are with School of Computer, National University of Defense Technology, Changsha, Hu Nan, P. R. China 410073. E-mail:{dsli,xclu}@nudt.edu.cn.

Jie Wu is with the Department of Computer Science & Engineering, Florida Atlantic University, Boca Raton, FL 33431, USA. Email: jie@cse.fau.edu.

The work of Li and Lu was supported in part by the National Natural Science Foundation of China under Grant No. 90412011, 90104001 and 90204005 and the National Basic Research Program of China under Grant No. 2003CD314802. The work of Wu was supported in part by US NSF Grants ANI 0073736, CCR 0329741, CNS 0422762, CNS 0445533, and EIA 0130806.

Two important measures of DHT schemes are *degree*, the size of routing table to be maintained on each peer, and *diameter*, the number of hops a lookup needs to travel in the worst case. Chord [5], Tapestry [6], Pastry [7], and CAN [8] are well-known DHT schemes. The degree of Chord [5] is $\log N$ and its diameter is also $\log N$, where N is the number of peers in the P2P network. The average path length of Chord is $1/2\log N$. Tapestry and Pastry are similar DHT schemes designed with the concept of prefix routing [9]. The degree of either is $O(d\log_d N)$ and the diameter is $O(\log_d N)$ where d is the base in which the peer identifiers are encoded. CAN uses a d-dimensional Cartesian coordinate space (for some fixed d) and its degree is $2d$. The diameter of CAN is $1/2dN^{1/d}$, and the average path length is $1/4dN^{1/d}$. From these schemes, it was observed in [4] that existing DHT schemes tend to achieve either $O(\log N)$ degree and $O(\log N)$ diameter (e.g., Chord, Tapestry, and Pastry) or $O(d)$ degree and $O(dN^{1/d})$ diameter (e.g., CAN). Thus it was asked in [4] whether there exists a DHT scheme with $O(d)$ degree and $O(\log N)$ diameter.

Recent work [10]–[13] showed that there are DHT schemes that achieve $O(\log N)$ diameter with $O(d)$ degree, but these schemes cause congestion. Xu *et al.* [13] systematically studied the degree/diameter tradeoff of DHT schemes and clarified the role that *c-congestion-free* (which is defined as the maximum traffic that nodes or edges deal with is no more than c times the average) plays in the degree/diameter tradeoff. Their research showed that $\Omega(\frac{\log N}{\log\log N})$ and $\Omega(\log N)$ are the asymptotic lower bounds for the diameter when the degree is $O(\log N)$ and d respectively, and a conjecture posed in [13] is that "when the network is required to be *c-congestion-free* for some constant c, $\Omega(N^{1/d})$ is the asymptotic lower bound for the diameter when the degree is no more than d". In this paper, we propose FISSIONE, a DHT scheme based on Kautz graphs, that can achieve constant degree, $O(\log N)$ diameter, and be $(1+o(1))$-*congestion-free*. The result from FISSIONE shows that DHT schemes with constant degree and constant congestion can still achieve $O(\log N)$ diameter, which is better than the lower bound $\Omega(N^{1/d})$ conjectured before.

Many proposed DHT schemes are based on some traditional interconnection topologies: Chord, Tapestry, and Pastry are based on the hypercube topology, CAN is based on the d-torus topology, Koorde [10], D2B [11] and ODRI [14] are based on the de Bruijn graph, and Viceroy [12] and Ulysses [13] are based on the Butterfly topology. Gummadi *et al.* [15] studies how basic geometric approaches (i.e., interconnection topologies) affect the resilience and proximity properties of DHT schemes. Loguinov *et al.* [14] examined graph theoretic properties of existing DHT schemes and proposed a new

DHT scheme, ODRI, based on de Bruijn graphs; however the details of ODRI are still under investigation. Compared with the hypercube, the de Bruijn graph, or the torus, the Kautz graph [16], [17] has some better properties, but there were no DHT schemes based on Kautz graphs. In this paper, we show the optimal diameter and optimal fault tolerance properties of the Kautz graph and demonstrate that the Kautz graph is $(1 + o(1))$-*congestion-free* when using the long path routing algorithm. Then we propose FISSIONE, the first effective DHT scheme based on Kautz graphs.

FISSIONE is based on well-known Kautz graphs; however, there are some challenges in building dynamic P2P networks with good properties based on static Kautz graphs:

1) First, the identifiers of peers or objects in P2P networks should be Kautz strings, but there is no existing hash algorithm that can determinately generate Kautz strings uniformly distributed in the Kautz namespace (as SHA-1 algorithm used in Tapestry to generate GUID for objects). We design a *KautzHash* algorithm to achieve that and prove its correctness and efficiency.
2) Second, the shortest path routing algorithm [17] generally used causes severe congestion in the Kautz graph. We adopt the long path routing algorithm and demonstrate its low congestion characteristic. Then we modify it to fit the routing algorithm used in dynamic P2P networks.
3) Third, the Kautz graph is a static topology and some mechanisms are required to adapt the static Kautz graph gracefully to dynamic P2P networks. In FISSIONE, peers are organized to form an approximate Kautz graph according to their identifiers, and the neighborhood and identifiers are adapted dynamically to the changing population of peers. FISSIONE keeps a topology rule called *neighborhood invariant* at all times to acquire good load balance and low diameter. The *split large and merge small* self-stabilization mechanism is proposed to deal with joining or departing of peers. The *atomic update* mechanism is adopted to avoid temporary inaccurate routing or other mistakes in P2P networks when updating. Based on these mechanisms, FISSIONE can be built as a scalable and high performance P2P network.

FISSIONE can achieve many good characteristics, such as load balance, high performance, and low congestion. FISSIONE is constant degree, $O(\log N)$ diameter and $(1 + o(1))$-*congestion-free*. The degree of FISSIONE is between 3 and 6, and its average degree is 4. The diameter of FISSIONE is less than $2\log N$, which matches the theoretical low bound $\Omega(\log N)$ of constant degree DHT schemes. The average routing path length is about $\log N$ and the maintenance cost is $O(\log N)$. Similarly to CAN, D2B, Viceroy, and Ulysses, each peer in FISSIONE owns a zone in virtual 2-dimensional Cartesian space and repositions the space when peers join or leave. However, the design of FISSIONE is very different from them, and thus it can achieve different tradeoffs. The identifiers of peers and data objects in FISSIONE are Kautz strings with different lengths, and the neighborhood is represented as an approximate Kautz graph. The topology rule and maintenance mechanism in FISSIONE are significantly different from existing approaches. Compared with FISSIONE, Ulysses is $O(\log N)$ degree and achieves a different tradeoff. D2B and Viceroy are DHT schemes to achieve expected constant degree and expected $O(\log N)$ diameter. The expected degree of D2B is constant, but its high probability bound is $O(\log N)$, i.e., a few peers would be of degree $\Omega(\log N)$. The expected diameter of Viceroy is about $3\log N$; however its $O(\log N)$ diameter is achieved not with certainty but "with high probability". Among the known DHT schemes, only CAN, Koorde, and FISSIONE are definitely constant degree. CAN is of $2d$ degree, but its diameter is $O(dN^{1/d})$. Koorde is constant degree and $O(\log N)$ diameter, but it is not $(1+o(1))$-*congestion-free* and its congestion is more severe than that of FISSIONE. The average routing path length of FISSIONE is shorter than that of CAN or Koorde with the same degree when the P2P network is large-scale.

The remainder of the paper is organized as follows. Section II introduces the Kautz graph and shows its good properties. Section III describes the detailed design of FISSIONE. Section IV evaluates the characteristics of FISSIONE. Conclusions and future work are discussed in Section V.

II. Kautz Graph and Low Congestion

Many DHT schemes are based on traditional interconnection network topologies. Different from dynamic P2P networks, the traditional interconnection topology poses some limits on the number of nodes it can support and does not support the dynamic joining or departing of nodes. To distinguish them, the traditional interconnection networks are called static networks in the paper.

A. Static Kautz Graph

FISSIONE adopts the Kautz graph as its static network topology. This section reviews the Kautz graph [16], [17] and its properties.

Definition 1: The *Kautz string* ξ of length k and base d is defined as a string $u_1u_2\ldots u_k$ where u_i belongs to an alphabet of $d+1$ symbols $\{0,1,2,\ldots,d\}$ and $u_i \neq u_{i+1}$ $(1 \leq i \leq k-1)$.

Definition 2: The *Kautz namespace* $KautzSpace(d,k)$ is defined as the set containing all Kautz strings of length k and base d, i.e., $KautzSpace(d,k) = \{u_1u_2\ldots u_k | u_1u_2\ldots u_k$ is a Kautz string$\}$.

The main feature of the Kautz string is that two consecutive symbols in it are always different. It is easy to see that there are $N = d^k + d^{k-1}$ strings in $KautzSpace(d,k)$ since the first symbol in the Kautz string has $d+1$ possibilities and all subsequent symbols have d possibilities.

Definition 3: The *Kautz graph* $K(d,k)$ [16], [17] is a directed graph whose node set is given by all strings in $KautzSpace(d,k)$. There is an edge from node U to node V (denoted by $U \rightarrow V$) iff V is a left-shifted version of U, i.e., there is an outgoing edge from $U = u_1u_2\ldots u_k$ to V iff $V = u_2u_3\ldots u_kx$ for any $x \neq u_k$ and $x \in \{0,1,2,\ldots,d\}$.

Obviously, each node in the Kautz graph $K(d,k)$ is of in-degree d and out-degree d and there are $N = d^k + d^{k-1}$ nodes in $K(d,k)$. Figure 1 shows Kautz graph $K(2,3)$.

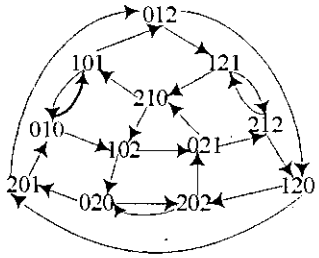


Fig. 1. Kautz graph $K(2,3)$.

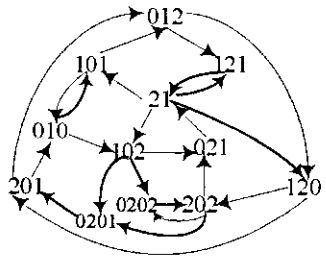


Fig. 2. Neighborhood of FISSIONE.

The Kautz graph is similar to the de Bruijn graph except that its nodes' labels are not normal strings (as in the de Bruijn graph) but Kautz strings. However, the Kautz graph can achieve some better properties, such as optimal diameter, optimal fault tolerance, and good load balance.

Given degree d and diameter k, the upper bound on the number of nodes N in a graph is given by the *Moore bound* [18] $1+d+d^2+\ldots+d^k$. The Moore bound is not achievable except in the trivial case when $d=1$ or $k=1$. The number of nodes in Kautz graph $K(d,k)$ is $d^{k-1}+d^k$, very close to the Moore bound. Furthermore, if $k=2$, the largest number of nodes in a graph is $d+d^2$ and then Kautz graphs are the densest graphs when the diameter is 2 (since in $K(d,2)$, $N=d+d^2$). From the Moore bound, it is easy to find that the low bound of the diameter of graphs with N nodes is $\lceil \log_d(N(d-1)+1)\rceil - 1$ and the diameter k of Kautz graph $K(d,k)$ reaches the lower bound as: $\lceil \log_d((d^k+d^{k-1})(d-1)+1)\rceil - 1 = \lceil \log_d(d^{k+1}-d^{k-1}+1)\rceil - 1 = k+1-1 = k$.

Thus the Kautz graph $K(d,k)$ has the optimal diameter. Table I shows the degree/diameter tradeoff of relevant topologies.

Kautz graphs are also optimally fault-tolerant [19]. The Kautz graph $K(d,k)$ of degree d has connectivity d (i.e., there are d node-disjoint paths between any two nodes) and failure of any $d-1$ components is tolerated. The corresponding de Brujin graph has connectivity $d-1$. In addition, the Kautz graph can achieve better load balance and lower latency than the de Bruijn graph [17]. Because the Kautz graph has these good features, FISSIONE selects it as the underlying static topology.

B. Low Congestion Routing

There are many routing algorithms for Kautz graphs, such as the *shortest path routing* algorithm and the *long path routing* algorithm [17]. FISSIONE adopts the *long path routing algorithm* in Kautz graphs.

Definition 4: Long path routing algorithm. With the long path routing algorithm, the routing path (called *long path*) from node $U=u_1u_2\ldots u_k$ to node $V=v_1v_2\ldots v_k$ in the Kautz graph $K(d,k)$ is a path of length k shown as below:

$$U=u_1u_2\ldots u_k \rightarrow u_2u_3\ldots u_kv_1 \rightarrow u_3u_4\ldots u_kv_1v_2 \rightarrow \ldots \rightarrow u_kv_1v_2\ldots v_{k-1} \rightarrow v_1v_2\ldots v_k = V \text{ (if } u_k \neq v_1)$$

or a path of length $k-1$ shown as below:

$$U=u_1u_2\ldots u_k \rightarrow u_2u_3\ldots u_kv_2 \rightarrow u_3u_4\ldots u_kv_2v_3 \rightarrow \ldots \rightarrow u_kv_2\ldots v_{k-1}v_k = v_1v_2\ldots v_k = V \text{ (if } u_k = v_1).$$

For example, the long path from node 201 to node 212 is $201 \rightarrow 012 \rightarrow 121 \rightarrow 212$, and the long path from 201 to 102 is $201 \rightarrow 010 \rightarrow 102$.

The long path may contain duplicate nodes and the algorithm keeps it for symmetry and simplicity. Obviously, the length of the long path between any two different nodes is k or $k-1$, and the average path length is $h=\frac{d}{d+1}*k+\frac{1}{d+1}*(k-1)=k-\frac{1}{d+1}$. The average routing path length of long path routing algorithm is a little longer than that of the shortest path routing algorithm (Table II shows the comparison between them), while the long path routing algorithm can achieve better load balance and other good characteristics and its average routing delay may even be less under heavy loads [17] (the severe congestion on some nodes in the shortest path leads to extra queuing delay).

Now we consider the congestion characteristic of long path routing in Kautz graphs. We use the concept "*congestion-free*" from Xu *et al* [13].

Definition 5: A P2P network is *c-congestion-free* [13] (c is a constant and $c \geq 1$) if its static network is both *c-node-congestion-free* and *c-edge-congestion-free* under *uniform all-to-all communication load*. Being *c-congestion-free* is also referred to as having *c congestion* or *constant congestion*. A network is said to be *c-node-congestion-free* if no node is handling more than c times the average traffic per node. A network is said to be *c-edge-congestion-free* if no edge is handling more than c times the average traffic per edge.

The *uniform all-to-all communication load* is defined as: for each pair of nodes U and V ($U \neq V$), there is a unit of traffic from U to V. A static P2P network is defined as the case in which all nodes in the identification space exist and are alive, i.e., nodes in the P2P network form the complete static topology.

Under *uniform all-to-all communication load*, there are $N*(N-1)$ routings in the network. Assuming the average path length of the network is h, then the average load on a node is $(N-1)*h$ and the average load of an edge is $N*(N-1)*h/|E|$ (where $|E|$ is the number of edges in the network).

Theorem 1: With the long path routing algorithm, Kautz graph $K(d,k)$ is $(1+o(1))$*-congestion-free.*

Proof: The detailed proof of Theorem 1 can be referred to [20], here is just the sketch. Define

$S_1 = \{u_1u_2\ldots u_ku_1u_2\ldots u_k | u_1u_2\ldots u_ku_1u_2\ldots u_k \in KautzSpace(d,2k)\}$,

$S_2 = \{u_1u_2\ldots u_ku_2\ldots u_k | u_1u_2\ldots u_ku_2\ldots u_k \in KautzSpace(d,2k-1) \text{ and } u_1=u_k\}$,

$S_3 = KautzSpace(d,2k) - S_1$,

$S_4 = KautzSpace(d,2k-1) - S_2$,

$S = S_3 \cup S_4$

The *uniform all-to-all communication load* is represented by the set M: $M=\{$ long paths from U to V | U and V are different nodes in $K(d,k)$ $\}$

Define mapping f: $\forall\delta \in M$, assuming δ is a routing path of length n: $b_1b_2\ldots b_k \rightarrow b_2b_3\ldots b_{k+1} \rightarrow b_3b_4\ldots b_{k+2} \rightarrow \ldots \rightarrow b_nb_{n+1}\ldots b_{n+k}$, then $f(\delta)=b_1b_2\ldots b_k\ldots b_{n+k}$.

It is easy to prove that f is a bijection from M to S. Thus under the *uniform all-to-all communication load*, for any node $R=r_1r_2\ldots r_k$, its load is equal to the number of the Kautz

TABLE I

DEGREE/DIAMETER TRADEOFF OF DIFFERENT TOPOLOGIES.

Topology	Degree	Diameter	Average path length
Hypercube topology (Chord)	$\log N$	$\log N$	$1/2 \log N$
d-torus topology (CAN)	$2d$	$1/2dN^{1/d}$	$1/4dN^{1/d}$
Butterfly topology	d	$2\log_d N(1-o(1))$ [14]	about $3/2\log_d N$ [14]
de Bruijn graph	d	$\log_d N$	$\log_d N - 1/(d-1)$ [14]
Kautz graph (FISSIONE)	d	$D = \log_d N - \log_d(1+1/d)$	$D - 1/(d+1)$

TABLE II

AVERAGE ROUTING PATH LENGTH (LONG PATH ROUTING ALGORITHM VS. SHORTEST PATH ROUTING ALGORITHM).

Kautz graph	$K(2,10)$	$K(2,11)$	$K(3,6)$	$K(3,7)$	$K(4,5)$	$K(4,6)$	$K(5,5)$	$K(6,4)$	$K(6,5)$
Number of nodes	1536	3072	972	2916	1280	5120	3750	1512	9072
Shortest path routing algorithm [17]	8.7922	9.7865	5.4624	6.4567	4.6541	5.6505	4.7430	3.7983	4.7958
Long path routing algorithm	9.6667	10.6667	5.75	6.75	4.8	5.8	4.8333	3.8571	4.8571

string $r_1 r_2 \ldots r_k$ appeared as a substring (except for the prefix) of Kautz strings in S. It can be found that the load L_n of R is:

$$L_n(R) = \begin{cases} k * d^k + (k-1)d^{k-1} - k & (r_1 \neq r_k) \\ k * d^k + (k-1)d^{k-1} - k + 1 & (r_1 = r_k) \end{cases}$$

The average path length h in $K(d,k)$ is $h = k - \frac{1}{d+1}$. Thus the average load of a node is :

$$\begin{aligned} Avg(L_n) &= (N-1) * h \\ &= (d^k + d^{k-1} - 1) * (k - \frac{1}{d+1}) \\ &= k * d^k + (k-1) * d^{k-1} - k + \frac{1}{d+1} \end{aligned}$$

As $Max(L_n) - Avg(L_n) = \frac{d}{d+1} \ll Avg(L_n)$, and

$$\begin{aligned} Max(L_n)/Avg(L_n) &< 1 + \frac{1}{(k-1) * (d^k + d^{k-1})} \\ &= 1 + \frac{1}{(k-1) * N} \\ &= 1 + O(\frac{1}{N * \log_d N}) \\ &= 1 + o(1) \end{aligned}$$

Thus $K(d,k)$ is $(1+o(1))$-*node-congestion-free*. Similarly, it can be proved that $K(d,k)$ is $(1+o(1))$-*edge-congestion-free*. Therefore Kautz graph $K(d,k)$ is $(1+o(1))$-*congestion-free*.

■

From Theorem 1, it can be derived that the Kautz graph has constant congestion (e.g. it is 2-*congestion-free*). When the number of nodes N is large, the Kautz graph is almost *congestion-free*.

A Kautz graph has optimal diameter and optimal fault-tolerance characteristics. In addition, it is $(1+o(1))$-*congestion-free* when using the long path routing. Thus the Kautz graph is a good static topology to construct DHT schemes. In this paper, we propose a novel constant-degree and $O(\log N)$-diameter DHT scheme, FISSIONE, which is based on Kautz graph $K(2,k)$ with long path routing. From Definition 5 and Theorem 1, FISSIONE has constant congestion.

III. FISSIONE DESIGN

A. Overview

FISSIONE uses a Kautz graph $K(2,k)$ as its static topology. Each peer in FISSIONE owns a zone in a virtual 2-dimensional Cartesian coordinate space. Peer and zone are synonyms in the paper and can be used interchangeably. The identifiers of zones in FISSIONE are Kautz strings with base 2, and zones are formed into an approximate Kautz graph according to their identifiers.

Initially, zones have equal area and the lengths of their identifiers are the same. Zones form a complete static Kautz graph (e.g. $K(2,1)$) at the beginning. When peers join or depart, the entire coordinate space is dynamically partitioned among all peers and the lengths of their identifiers may become different. For example, if a new peer p joins, it first finds a large zone V that has no larger neighbors, and zone V is split into two new zones ("*split large*" policy): one is for the peer that owns V originally and another for peer p. The length of new zone's identifier is one more than that of V and its area is one half of zone V. Then the neighborhoods related to V is adapted to maintain the approximate Kautz graph. When a peer p departs, two brother zones Y_1 and Y_2 which have no smaller neighbors are found and merged ("*merge small*" policy) to a new zone Z. The identifier of Z is one shorter than Y_1 and its area is twice of that of Y_1. Afterward related peers would update their routing tables.

To achieve good characteristics, FISSIONE keeps a topology rule called *neighborhood invariant* which requires that the difference of identifier lengths between neighbors is no more than one at all times. Such a rule can ensure that neighbor zones have similar area size and that the neighborhood is simple.

Each data object in FISSIONE is assigned a unique ObjectID which is a Kautz string of fixed length m. The object is published on the peer whose identifier is the prefix of its ObjectID.

B. FISSIONE Neighborhood

The identifiers of zones in FISSIONE are Kautz strings with base 2. Initially the identifiers of zones are labels of nodes in a static Kautz graph (e.g., $K(2,1)$) and zones own the same

area in the Cartesian coordinate space. However, the lengths of identifiers may be different due to the dynamic arrival and departure of nodes, as explained in a later subsection.

FISSIONE keeps a topology rule called *neighborhood invariant* at all times. Denoting the length of the identifier of U as $|U|$, the *neighborhood invariant* is shown as Theorem 2 (its proof is in Section IV).

Theorem 2 (Neighborhood Invariant): If zone U and V in FISSIONE are neighbors, $||U| - |V|| \leq 1$.

The neighborhoods of zones are based on zone identifiers. Assuming the identifier of zone U is $u_1u_2 \ldots u_k$ (denote it as $U = u_1u_2 \ldots u_k$), to form a approximate Kautz graph, U has two kinds of neighbors:

Out-neighbors: neighbors whose identifiers are $u_2u_3 \ldots u_kq_1 \ldots q_m$ with $0 \leq m \leq 2$ (if $m < 1$, the string in the style $q_1 \ldots q_m$ represents the null string).

In-neighbors: neighbors whose identifiers are $au_1u_2 \ldots u_i$ $(a \neq u_1)$ with $k-2 \leq i \leq k$.

The routing table of U contains all the out-neighbors and in-neighbors of U. Notice that if U is the out-neighbor of V, V also is the in-neighbor of U. Figure 2 shows the neighborhood of FISSIONE (pay special attention to nodes 21, 0202, 0201 and their neighbors).

C. Universal Naming

In many DHT schemes, data objects are encoded by some public hash algorithm (e.g., SHA-1, MD5), and the identifiers of data objects are in a similar namespace related to the nodes' identifiers. For example, the data namespace in Chord scheme is [0, $2^{160} - 1$], which is also the namespace of nodes in Chord scheme. The identifiers of nodes in Fission are Kautz strings with base 2 and the *Kautz_hash* algorithm is proposed to generate a destination Kautz string ξ for each data object O. Similar to SHA-1 [21], the *Kautz_hash* algorithm should be determinate and the Kautz string generated should be uniformly distributed in the Kautz namespace.

The *Kautz_hash* algorithm uses four parameters: key, m, p and n. For each data object O, the *Kautz_hash* algorithm can generate a Kautz string of length m based on its keyword key. The procedure of *Kautz_hash* is shown below:

First *Kautz_hash* appends $0, 1, \ldots, p$ to key and gets key_0, key_1, ..., key_p. Then key_0, key_1, ..., key_p are respectively hashed to 160-bit binary numbers $D_0, D_1, \ldots, D_p$ by the public hash algorithm SHA-1, i.e., $D_0 = SHA-1(key_0)$, $D_1 = SHA-1(key_1)$, ..., $D_p = SHA-1(key_p)$. $D_0, D_1, \ldots, D_p$ are concatenated to acquire a binary number D and D is then converted to a ternary number R'. The low n digits of R' form R. R is a ternary number; however R may not be a valid Kautz string because there may exist certain sequences $bb \ldots b$ with $b \in \{0, 1, 2\}$ in R. Thus R is further converted to a Kautz string Q by substituting a single b for any sequence $bb \ldots b$ in R. If the length of Q is less than m, then *Kautz_hash* keeps the value of $D_0, D_1, \ldots, D_{p-1}$, appends $p+1$ to key, and calculates D_{p+1} by SHA-1, ..., and the procedure above is repeated again until a Kautz string Q with length of no less than m is obtained. The desired destination Kautz string ξ is then acquired from the low m digits of Q. Figure 3 shows the *Kautz_hash* algorithm.

```
Procedure Kautz_hash (Keyword key, Len m, Merg p, Digit n)
  // generate a Kautz string of length m based on keyword key
  // p and n are two adjustable parameters
1  D ← φ   // φ is an empty string
2  for i = 0 to p − 1
3      do D ← D||(SHA − 1(key||i))
           // || is the concatenation operation
4  i ← p
5  repeat
6      D ← D||(SHA − 1(key||i))
7      i ← i + 1
8      R' ← Convert_binary_ternary(D)
           // convert binary D to ternary R'
9      R ← Get_low_digit(R', n)
           // get the low n digits of R' to acquire string R
10     Q ← Merge_string(R)
           // merge identical consecutive characters in R to
             acquire Kautz string Q
11 until |Q| ≥ m
12 ξ ← Get_low_digit(Q, m)
13 return ξ
```

Fig. 3. *Kautz_hash* algorithm.

The *Kautz_hash* algorithm can adjust the parameters p, n and m to acquire destination Kautz strings with different characteristics. When $m = 100$, FISSIONE could support a P2P system with more than 10^{15} peers (refer to section IV for details) that is large enough for general applications. Thus we set $m = 100$. To acquire the uniformly distributed Kautz strings, we set $p = 2$ and $n = 280$. The following Theorem 3 (its proof shown in Appendix A) shows that when $p = 2$ and $n = 280$, the *Kautz_hash* algorithm can efficiently generate a destination Kautz string of length 100 for each data object and the Kautz string generated is uniformly distributed in the Kautz namespace.

Theorem 3: With high probability, the *Kautz_hash* algorithm (with parameters $p = 2$ and $n = 280$) can generate the destination Kautz string of length 100 in one iteration, and Kautz strings generated are uniformly distributed in the Kautz namespace $KautzSpace(2, 100)$.

When the parameters are set $p = 2$ and $n = 280$, the computing complexity of *Kautz_hash* algorithm is about three times of SHA-1 algorithm; thus it is practical.

The procedure of publishing a data object is shown as below: If peer p would like to publish a data object O, it should first get the destination Kautz string S of O. Then peer p invokes a routing to Kautz string S and the routing would arrive at a peer m whose identifier is the prefix of S (the routing algorithm is discussed in the next subsection). Thus the data object O is published on peer m. The lookup of data objects is similar to the publication and omitted here.

D. Routing in FISSIONE

Routing in FISSIONE is similar to that in Kautz graphs. Once a zone $U = u_1u_2 \ldots u_k$ receives a routing message *Routing*(V, L, S) to destination $V = v_1v_2 \ldots v_m$ $(U \neq V)$ with left path length L, U sends a new routing message

Routing$(V, L-1, SX)$ to Q if the following conditions hold: (a) U has one out-neighbor zone $Q = u_2 \ldots u_k X$ where $X = x_1 \ldots x_j$ $(0 \leq j \leq 2)$ and (b) SX is a prefix of V.

The initial values of L and S are set as below: Assume there is a routing from source zone $W = w_1 w_2 \ldots w_k$ to destination Kautz string $V = v_1 v_2 \ldots v_m$. If $w_k = v_1$, then set $L = k-1$, $S = v_1$; else set $L = k$, $S = null$. Figure 4 shows the routing algorithm.

```
Procedure FISSIONE_Routing (SourceZone W, DestString V)
  // routing from source zone W = w1w2...wk to
     destination Kautz string V = v1v2...vm
  if wk = v1
     then return W.Routing(V, k-1, v1)
     else return W.Routing(V, k, null)

Procedure U.Routing (Dest V, PathLen L, ComPrefix S)
  // zone U = u1u2...uk deals with the routing message
     to destination Kautz string V = v1v2...vm
  if L = 0
     then return U // reach destination V
     else if ∃Q ∈ outneighbors(U) and Q = u2...ukX
             and Is_prefix(SX, V)
          then S ← SX
               Q.Routing(V, L-1, S)
               // the routing message is forwarded to Q
```

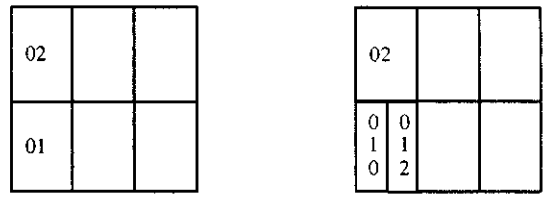
Fig. 4. Routing algorithm.

The routing to destination Kautz String S will stop at a unique peer p whose identifier is a prefix of S (the proof is shown in section IV).

E. Maintenance and Self-stabilization

1) Peer Joins: When a new peer p joins in FISSIONE, its join procedure can be divided into two phases: at the first phase, peer p routes to the peer W that is responsible for the destination Kautz String of p; at the second phase, the JOIN message is propagated from peer W to a large zone V which has no larger neighbors. Then V is split and the routing tables should be updated. The details are shown below.

Split large zones. When peer p joins, it should know a peer n which is already in the P2P network. Peer p first gets a unique destination Kautz string $U = u_1 u_2 \ldots u_{100}$ (e.g., by performing *Kautz_hash* algorithm on its IP address). Then peer p invokes a routing from the gateway peer n to U. The routing will reach a unique zone W whose identifier is the prefix of U, and W invokes a JOIN message. Then starting from zone W, if the current zone has a neighbor zone with larger area, it forwards the JOIN message to the neighbor (if there is more than one neighbors with larger area, select one randomly and forward the JOIN message to it). This process will not stop until the JOIN message reaches a zone V which has no larger neighbors and the JOIN message can not be forwarded any more. Thus zone V is split into two zones V_1 and V_2. The owner of zone V_1 is set to peer m that was originally the owner of zone V before the split, and the owner of zone V_2 is set to peer p. Suppose the identifier of zone V is $v_1 v_2 \ldots v_k$, then the identifier of V_1 is $v_1 v_2 \ldots v_k x_0$ and the identifier of V_2 is $v_1 v_2 \ldots v_k \tilde{x}_0$ $(0 \leq x_0, \tilde{x}_0 \leq 2, x_0 \neq v_k, \tilde{x}_0 \neq v_k, x_0 \neq \tilde{x}_0)$. V_1 and V_2 are brother zones. Figure 5 shows the case that peer p joins into zone 01.



(a) Zones before peer p joins (b) Zones after peer p joins

Fig. 5. Peer p joins in FISSIONE.

From the split procedure above, it is easy to know that the area of a zone is in proportion to 2^{-h} when the length of its identifier is h. The longer the identifier is, the less area the zone occupies. Also, the number of data objects stored on one peer is in proportion to 2^{-h} according to the publication procedure. Thus the number of data objects stored on a peer is in proportion to the area of its zone.

Update the routing tables. Once zone V is split, the routing tables of related zones should be updated. For out-neighbors $R = v_2 \ldots v_k q_1 \ldots q_m$ $(0 \leq m \leq 2)$ of V, the JOIN message stops at zone V, thus $|V| \leq |R|$ and $1 \leq m \leq 2$. If $q_1 = x_0$, R becomes an out-neighbor of V_1; else $q_1 = \tilde{x}_0$ and R becomes an out-neighbor of V_2. Also R should update its routing table accordingly. For in-neighbors $Q = a v_1 v_2 \ldots v_j$ $(j \leq k)$ of V, Q becomes an in-neighbor of both V_1 and V_2. Figure 6 shows the maintenance algorithm for peer joining.

```
Procedure PeerJoin (GatePeer n, NewPeer p)
  // new peer p join in the system through known peer n
  U ← Kautz_hash(IP(p))
      // U is the destination Kautz string of p
  W ← FISSIONE_Routing(n, U)
      // routing from peer n to U and stops at zone W
  V ← W
  while ∃Q ∈ neighbors(V) and |Q| < |V|
     do V ← Q
  Split(V, V1, V2)
      // split zone V to acquire zones V1 and V2
  Build_routingtables(V1, V2)
      // build the routing tables of zones V1 and V2
  Update_routingtables(neighbors(V))
      // update the routing tables of V's neighbors
```

Fig. 6. Maintenance algorithm for peer joining.

2) Peer Departs: When a peer p departs from FISSIONE, the zone V it owns should be occupied by other peers. That force zones to merge. FISSIONE tries to merge the small brother zones which have no neighbors with smaller area.

Merge small zones. If peer p volunteers to depart from the system, it produces a DEPART message. Starting from zone V, if the current zone has a neighbor zone with smaller area, the DEPART message should be forwarded to the neighbor. The DEPART message is propagated until a zone $U = u_1 u_2 \ldots u_k$

is reached which has no neighbors with smaller area. Consider U's neighbor $R = au_1u_2 \ldots u_i$ $(k-2 \leq i \leq k)$, obviously $k-2 \leq i \leq k-1$ (for $|R| \leq |U|$); (from Corollary 1 in Section IV) R has a neighbor $W = u_1u_2 \ldots u_{k-1}\tilde{u_k}q_1 \ldots q_m$ $(0 \leq m \leq 1)$. Then, (1) if $m = 0$, $W = u_1u_2 \ldots u_{k-1}\tilde{u_k}$. If W has a neighbor T with a smaller area, the DEPART message is forwarded to T and continues to propagate; else the two brother zones U and W which will be merged are acquired; (2) if $m = 1$, (from Corollary 1 in Section IV) R has also a neighbor $W' = u_1u_2 \ldots u_{k-1}\tilde{u_k}\tilde{q_1}$. If either W of W' has a neighbor T with a smaller area, the DEPART message is forwarded to T and continues to propagate; else the brother zones W and W' which will be merged are acquired and the DEPART message is stopped.

During the departure procedure, once the DEPART message is forwarded one time, the identifier length of the zone passed decreases by at least one. From Theorem 5 in Section IV, the DEPART message can be forwarded less than $\log N$ hops. Thus we can acquire the brother zones $Y_1 = y_1y_2 \cdots y_{n-1}y_n$ and $Y_2 = y_1y_2 \cdots y_{n-1}\tilde{y_n}$, and Y_1 and Y_2 have no smaller neighbor zones. Assume the owners of Y_1, Y_2 are peer p_1 and peer p_2 respectively:

(1) if Y_1 or Y_2 is V, without loss of generality we can assume Y_1 is V and in this case peer p_1 and peer p are the same peer. Merge zones Y_1 and Y_2 into a new zone $V' = y_1y_2 \cdots y_{n-1}$ and assign peer n_2 as the owner of zone V'.

(2) if either Y_1 or Y_2 is not V, merge zone Y_1 and Y_2 into a new zone $V' = y_1y_2 \cdots y_{n-1}$, then change the owner of the zone V to peer p_1 and assign peer p_2 as the owner of zone V'.

Update the routing tables. After zone Y_1 and zone Y_2 are merged into zone V', the routing table of V' maintains all in-neighbors and out-neighbors of both Y_1 and Y_2. For each in-neighbor R of Y_1 (or Y_2), R substitutes V' for its out-neighbor Y_1 (or Y_2). For each out-neighbor W of Y_1 (or Y_2), W substitutes V' for its in-neighbor Y_1 (or Y_2). Figure 7 shows the maintenance algorithm for peer departure.

Involuntary departure. To deal with involuntary failure, each peer sends *KeepAlive* messages to all neighbors periodically. The deferred absence of a *KeepAlive* message from one neighbor indicates its failure. Once the failure of a peer p is detected by its neighbor n, peer n will generate one DEPART message for peer p. And the remaining process is the same as that in the case of voluntary departure.

3) Simultaneous Join or Departure: Many peers may join in (or depart from) FISSIONE at the same time. That may cause temporary inaccurate information in the routing tables of peers, which in turn may cause errors in the routing, join, or departure procedure. To avoid that, FISSIONE adopts an *atomic update* mechanism:

When a peer joins in (or departs from) FISSIONE, the routing tables of related peers should be updated. Only when all the updates are completed, is the new routing tables allowed to be used. During the update period, the relevant routing requests are forwarded according to the original routing table, but the JOIN or DEPART messages are withheld. The peers that are the sources of the JOIN or DEPART messages are informed to resend these messages after the update is finished.

```
Procedure PeerDepart (DepartPeer p, DepartZone V)
  // peer p that owns zone V departs from the system
  U ← V
  flag ← 1
  repeat
      while ∃Q ∈ neighbors(U) and |Q| > |U|
          do U ← Q
      get neighbor R = au_1 ... u_i (k − 2 ≤ i ≤ k − 1)
          of peer U = u_1 ... u_{k−1}u_k
      get neighbor W = u_1 ... u_{k−1}ũ_k q_1 ... q_m (0 ≤ m ≤ 1)
          of peer R;
      if m = 0
          then if ∃T ∈ neighbors(W) and |T| > |W|
                  then U ← T
                  else Y_1 ← U
                       Y_2 ← W
                       flag ← 0
          else get neighbor W' = u_1 ... u_{k−1}ũ_k q̃_1 of R
              if ∃T ∈ neighbors(W, W') and |T| > |W|
                  then U ← T
                  else Y_1 ← W
                       Y_2 ← W'
                       flag ← 0
  until flag = 0
  V' ← Merger_zones(Y_1, Y_2)
      // merge zones Y_1 and Y_2 to get zone V'
  Update_routingtables(V')
  Update_routingtables(neighbors(V'))
      // update routing tables of V' and its neighbors
```

Fig. 7. Maintenance algorithm for peer departure.

Because the average degree of FISSIONE is only 4 and the average number of hops for a JOIN or DEPART message is propagated is small (referred to Section IV), the overhead caused by the atomic update mechanism is very small.

F. Fault-Tolerant Routing

When involuntary failures of peers occur and the related routing tables have not been updated, the routing messages may be forwarded to failed peers. To increase robustness, FISSIONE can adopt three fault-tolerant mechanisms:

(1) DFTR [19] mechanism. FISSIONE can modify the DFTR mechanism and its extension to choose other routing paths when a failed peer occurs in the routing path.

(2) Fault-tolerant neighbor mechanism. For each peer $U = u_1u_2 \ldots u_k$, peers $\tilde{u_2}u_3 \ldots u_kq_1 \ldots q_m$ $(0 \leq m \leq 2)$ are regarded as U's fault-tolerant neighbors. When U deals with a routing message *Routing*(V, L, S), if the neighbor $u_2 \ldots u_kx_1 \ldots x_j$ $(0 \leq j \leq 2)$ that the routing message should be forwarded to has failed, U can forward the routing message to its fault-tolerant neighbor $\tilde{u_2}u_3 \ldots u_kq_1 \ldots q_m$ where $Sq_1 \ldots q_m$ is a prefix of V.

(3) Multiple out-edges mechanism. For each peer $U = u_1u_2 \ldots u_k$, peers $u_3 \ldots u_kx_1 \ldots x_j$ $(0 \leq j \leq 4)$ are also regarded as U's neighbors. When U deals with routing message *Routing*(V, L, S), it first forwards the routing message to neighbor $V = u_3 \ldots u_kx_1 \ldots x_j$ where $Sx_1 \ldots x_j$ is a prefix of V. If V has failed, U can forward the routing message to

another peer $u_2 \ldots u_k q_1 \ldots q_m$ where $Sq_1 \ldots q_m$ is a prefix of V.

The DFTR mechanism is rather complex, but it does not affect the degree characteristics of FISSIONE. The fault-tolerant neighbor mechanism is simple, but it would cause an increase in the degree of FISSIONE (the fault-tolerant degree is 2). Multiple out-edge mechanisms can decrease the routing path length of FISSIONE while increasing its degree. The detailed comparison of three fault-tolerant mechanisms is omitted here, and this paper focuses on the basic FISSIONE scheme.

IV. Analysis and Evaluation

In this section, We will show and prove three theorems about the properties of the basic FISSIONE scheme. Due to space limitations, we only present the proof sketches.

Theorem 3 (Neighborhood Invariant): If zone U and zone V are neighbors, $||U| - |V|| \leq 1$.

Theorem 4 (Correctness of FISSIONE): In FISSIONE, the routing to Kautz string $V = v_1 v_2 \ldots v_m$ (m is a big enough integer) will arrive at a unique zone V' whose identifier is a prefix of V.

Theorem 5 (Performance Characteristics): In an N-peer FISSIONE system,

(1) The in-degree of each peer is 2 and the out-degree is between 1 and 4. The average out-degree is 2.

(2) The diameter of FISSIONE systems is less than $2 \log N$.

(3) The messages caused by peer joining are propagated less than $3 \log N$ hops (and the JOIN message is propagated less than $\log N$), and the DEPART message is propagated less than $\log N$ hops. Only constant peers need to update their routing tables when a peer joins or departs.

A. Neighborhood Invariant

To prove these theorems, we first give some lemmas.

Lemma 1: For each zone $U = u_1 u_2 \ldots u_k$ in FISSIONE, if zone $V = v_1 v_2 \ldots v_m$ is an out-neighbor of zone U, then $||U| - |V|| \leq 1$, *i.e.*, $|k - m| \leq 1$.

Proof: Lemma 1 holds initially. We will show that if Lemma 1 holds at a certain time, Lemma 1 will also hold after a split or merge.

In the case of a split, the large zone is divided into two zones. Assume after a split there are two zones U and V with $||U| - |V|| \geq 2$ and V is one out-neighbor of U. Recall that before the split for any zone P and zone W in FISSIONE, $||P| - |W|| \leq 1$; thus $||U| - |V|| \leq 2$. Therefore $||U| - |V|| = 2$ and either zone U or zone V is newly produced by the split.

(1) If U is derived from U' in the split, then $||U'| - |V|| \leq 1$ and $|U| = |U'| + 1$. Obviously, to achieve $||U| - |V|| = 2$, $|U'| - |V|$ must be 1 before the split. Recall that V is the out-neighbor of U after the split, thus V must be the out-neighbor of U' before the split. But if $|U'| - |V| = 1$ (which means the area of zone V is larger than that of zone U') and U', V are neighbors, then the JOIN message would be forwarded from U' to V and the zone U' would not have been split. Thus a contradiction occurs.

(2) If V is derived from V' in the split, $|V'| - |U|$ must be 1 before the split. Then zone U' is larger than zone V', and the zone V' wouldn't be split. Thus a contradiction occurs. Therefore, after a split Lemma 1 remains true.

For a merge, the proof is similar and omitted here. ■

Theorem 1 can be easily derived from Lemma 1.

B. Correctness of FISSIONE

Lemma 2: For each zone $U = u_1 u_2 \ldots u_k$ in FISSIONE, there are no zones $V = u_1 u_2 \ldots u_k x_1 \ldots x_j$ with $j \geq 1$ and $U \neq V$.

The proof of Lemma 2 is shown in Appendix B.

Lemma 3: For each zone $U = u_1 u_2 \ldots u_k$ and any Kautz string $S = s_1 \ldots s_m (s_1 \neq u_k$ and $m \geq 2)$ with base 2, U has an out-neighbor $u_2 \ldots u_k x_1 \ldots x_j$ $(0 \leq j \leq 2)$ with $x_1 \ldots x_j$ as a prefix of S.

Proof: From lemma 1, if zone Q is an out-neighbor of U, then $|U| - 1 \leq |Q| \leq |U| + 1$. Thus Q is $u_2 \ldots u_k$ or $u_2 \ldots u_k x_1$ or $u_2 \ldots u_k x_1 x_2$. Lemma 3 holds for the initial static Kautz graph. After a split or merge, it is easy to demonstrate that lemma 3 still holds. Thus lemma 3 is always true. ■

The following two corollaries are direct conclusions from Lemma 3.

Corollary 1: For any zone $U = u_1 u_2 \ldots u_k$, if U has a neighbor $u_2 \ldots u_k x_1 \ldots x_j$ $(1 \leq j \leq 2)$, U has another neighbor $u_2 \ldots u_k \bar{x}_1 q_1 \ldots q_m$ $(0 \leq m \leq 1)$.

Corollary 2: The out-degree of FISSIONE is between 1 and 4.

Lemma 3 and Corollary 1 show that the routing algorithm in Section III could go on until the destination zone V is reached.

Lemma 4: Consider the routing from source zone $W = w_1 w_2 \ldots w_k$ to any destination Kautz string $V = v_1 v_2 \ldots v_m$ ($W \neq V$ and m is an integer big enough), s is an integer: if $v_1 = w_k$, let $s = 1$, else let $s = 0$. Let the routing path from W to V be $U_1 (= W), U_2, U_3, \ldots, U_q (= V)$, then U_i is of the form $w_i \ldots w_{k-s} S$ and the routing message that U_i deals with is in the form of *Routing*$(V, k - s - i + 1, S)$ where S is a prefix of V.

Proof: If $v_1 = w_k$, let string $S_0 = w_k$, else let S_0 be null. Then $U_1 = W = w_1 w_2 \ldots w_{k-s} S_0$, $V = v_1 v_2 \ldots v_m = S_0 v_{s+1} \ldots v_m$. The routing message that W deals with is *Routing*$(V, k - s, S_0)$. Thus initially Lemma 4 holds for U_1.

Suppose current zone U_i $(1 \leq p \leq k - s)$ is $w_i \ldots w_{k-p} S$ and the routing message that U_i deals with is *Routing*$(V, k - s - i + 1, S)$ where S is a prefix of V. From Lemma 3, U_i has an out-neighbor $w_{i+1} \ldots w_{k-s} S x_1 \ldots x_j$ $(0 \leq j \leq 2)$ with $S' = S x_1 \ldots x_j$ as a prefix of V, then $U_{i+1} = w_{i+1} \ldots w_{k-s} S x_1 \ldots x_j = w_{i+1} \ldots w_{k-s} S'$ and the routing message that U_{i+1} deals with is *Routing*$(V, k - s - i, S')$. Thus Lemma 4 holds for U_{i+1}. Therefore Lemma 4 holds. ■

From Lemma 4, when $i = k - s + 1$ $(L = 0)$, the routing message reaches a zone U_i and the routing stops and there is a certain S that is a prefix of V and $U_i = S$. Thus the routing from any source zone to destination V will arrive and stop at a zone V' whose identifier is the prefix of U. From Lemma 2, zone V' is unique. Therefore, Theorem 4 is true.

The path length of routing from W to V is $k-s$ hops. Thus we get the following corollary.

Corollary 3: The path length of routing initiated by any source zone $U = u_1u_2 \ldots u_k$ is no more than k hops.

C. Performance Characteristics

Lemma 5: The in-degree of any zone is always 2.

Proof: Initially there are three zones 0, 1, 2 and each zone has two in-edges. So initially Lemma 4 holds. After a split or merge, lemma 4 still holds. Thus Lemma 4 is always true. ■

Lemma 6: In an N-peer FISSIONE system, the largest zone U satisfies that $|U| < \log N$.

Proof: Let $|U| = k$, then k is the smallest among the lengths of identifiers of zones in FISSIONE. Peers in FISSIONE form an approximate Kautz topology, thus $2^k + 2^{k-1} \leq N$. Then $k \leq \log N - \log 3 + 1 < \log N$. ■

Lemma 7: In an N-peer FISSIONE system, the smallest zone V satisfies that $|V| < 2\log N$.

Proof: Suppose U is the largest zone in FISSIONE system, and consider the routing path from U to V. From Corollary 3, we know that the path length is no more than $|U|$. Thus from Lemma 1, we can infer that $||V| - |U|| \leq |U|$. Because $|V| \geq |U|$, thus $|V| - |U| \leq |U|$ and $|V| \leq 2|U| < 2\log N$. ■

The following corollaries can be derived from Lemma 7 and Corollary 3 directly.

Corollary 4: In an N-peer FISSIONE system, let U and V be the smallest and largest zones in the system respectively. Then $|U| - |V| \leq |V| < \log N$.

Corollary 5: In an N-peer FISSIONE system, the path length of routing is less than $2\log N$ hops.

Lemma 8: When a peer joins in or departs from an N-peer FISSIONE system, the messages caused are totally propagated at most $3\log N$ and $\log N$ hops respectively (and the JOIN or DEPART message is propagated at most $\log N$ hops); and only constant peers need to update their routing tables.

Proof: Take the split procedure as an example: the message is first routed to the destination Kautz string. From Corollary 5, in this phase the message is propagated less than $2\log N$ hops. Then the JOIN message is forwarded to neighbors whose identifiers are at least one shorter than that of the current zone. From Corollary 4, in that phase the message is propagated at most $\log N$ hops. Therefore, the message caused by peer joining is propagated at most $3\log N$ hops. From Corollary 2 and Lemma 5, the number of neighbors that should update their routing tables due to the split is constant. ■

Theorem 5 can be directly got from Corollary 2, Corollary 5, Lemma 5, and Lemma 8.

D. Performance Evaluation

We implement FISSIONE in the simulator and evaluate the following characteristics of FISSIONE: degree distribution, load balance, diameter, average path length, and fault-tolerance.

First we evaluate the degree distribution of FISSIONE. We calculate the degree of peers in FISSIONE when the number of peers is 6,000 and 50,000, and the simulation result is shown in Figure 8. Figure 8 shows that the degree distribution of peers in FISSIONE with different scales is slightly different, but most peers are of degree 4.

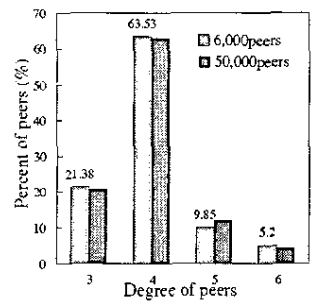


Fig. 8. Degree distribution.

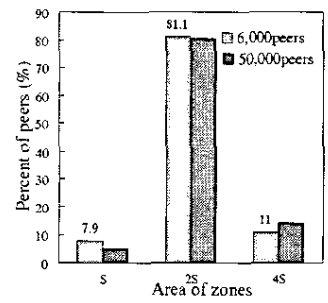


Fig. 9. Area distribution of zones.

We observe the load balance characteristic of FISSIONE and simulate FISSIONE with 6,000 and 50,000 peers. In each experiment, we calculate the area of each zone and let S be the smallest area among all zones. Figure 9 shows the percentage of peers that own a particular area. From Figure 9, it can be inferred that more than 80% of peers own the same area $2S$ and the percentage of peers whose area is more than $4S$ is zero. The number of objects stored on each peer is in direct proportion to its zone's area; thus the distribution of objects over peers is almost uniform. Therefore, FISSIONE has a good load balance.

Then we evaluate the average path length of FISSIONE in different scales (from 256 peers up to 64K peers) and compare it with CAN (with degree 4 ($d = 2$) or degree 6 ($d = 3$)) and Koorde (with degree 4). In each experiment, we select two random zones and invoke a routing from one to the other, and then get the average path length over 10,000 routings. Figure 10 shows the simulation results. Figure 10 also shows the diameter of FISSIONE (denoted as FISSIONE(max)). From Figure 10, we can infer that the average path length of FISSIONE is less than $\log N$ and the diameter of FISSIONE is only a little more than $\log N$. When the number of peers is large, the average routing path length of FISSIONE is shorter than that of CAN or Koorde.

We also evaluate the distribution of routing path length in FISSIONE. Figure 11 shows the distribution of routing path length of 10,000 random routings in FISSIONE when the number of peers is 50,000. From Figure 11, we can infer that more than 50% of routing paths are of the same length.

We evaluate the self-stabilization cost when a peer joins or departs. In the experiment, the number of peers in FISSIONE is 50,000. The experiment is repeated 100 times and each time a peer joins through a random peer (or a random peer departs). Figure 13 shows the average hops the update messages are propagated and Figure 14 shows the distribution of hops that the JOIN message and DEPART message are propagated. From Figure 13 and Figure 14, it can be inferred that the average hops that the JOIN message or DEPART message is

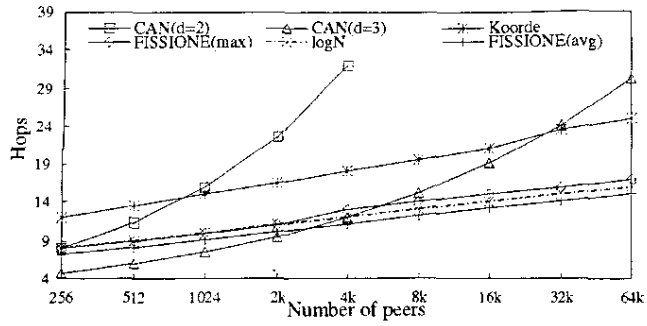


Fig. 10. Average path length.

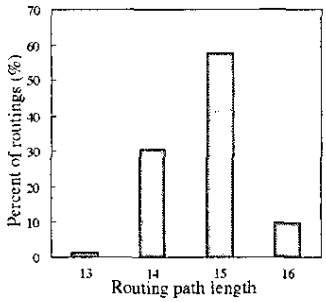


Fig. 11. Path length distribution.

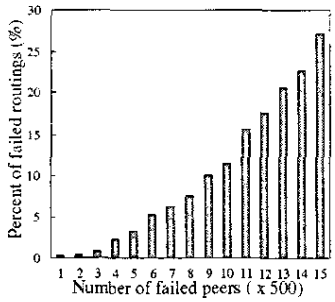


Fig. 12. Fault-tolerance property.

propagated is very small, and they are propagated at most two hops. The message cost of peer joining is larger than that of peer departing because a routing to the destination Kautz string is invoked first.

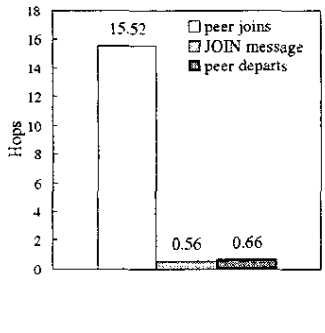


Fig. 13. Average update cost.

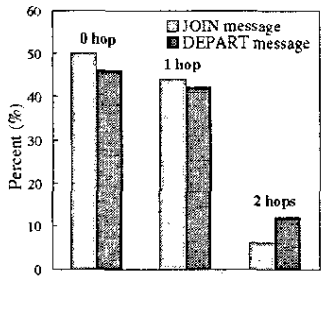


Fig. 14. Hop distribution of messages.

We evaluate the fault-tolerance of FISSIONE and test the fault-tolerant neighbor mechanism. We simulated the failure ratio for 10,000 random routings in FISSIONE with 50,000 peers when the number of failed peers is $500, 1,000, \ldots, 7,500$ respectively. Figure 12 shows the simulation result. From Figure 12, it can be inferred that FISSIONE has some fault-tolerant characteristics.

V. Conclusions

FISSIONE is the first effective DHT scheme based on Kautz graphs and it is $O(\log N)$ diameter with only $O(1)$ degree and $(1 + o(1))$-*congestion-free*. FISSIONE can achieve good load balance, high performance and low congestion. Our future work will focus on two directions. First, the current design of FISSIONE is based on the Kautz graph $K(2, k)$ and needs to be extended to general $K(d, k)$ for flexibility. Second, the physical topological information should be exploited in FISSIONE to reduce latency.

References

[1] D. Clark. Face-to-face with peer-to-peer networking. *IEEE Computer*, 34(1):18–21, 2001.

[2] D. Schoder and K. Fischbach. Peer-to-peer prospects. *Communications of the ACM*, 46(2):27–29, 2003.

[3] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking up data in P2P systems. *Communications of the ACM*, 46(2):43–48, 2003.

[4] S. Ratnasamy, S. Shenker, and I. Stoica. Routing algorithms for DHTs: Some open questions. In *Proc. of 1st Workshop on peer-to-peer Systems (IPTPs'02)*, 2002.

[5] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balalrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of ACM SIGCOMM 2001*, pages 160–177. ACM Press, 2001.

[6] B. Y. Zhao, L. Huang, and J. Stribling. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications(JSAC)*, 22(1), 2004.

[7] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of IFIP/ACM Middleware2001*, Heidelberg, Germany, 2001.

[8] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of ACM SIGCOMM'2001*. ACM Press, 2001.

[9] J. Wu and L. Sheng. Deadlock-free routing in irregular networks using prefix routing. In *Proc. of the ISCA 12th International Conference on Parallel and Distributed Computing Systems*, pages 424–430, 1999.

[10] F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal hash table. In *Proc. of 2nd Intl. Workshop on Peer-to-Peer Systems (IPTPS'2003)*, 2003.

[11] P. Fraigniaud and P. Gauron. The content-addressable network D2B. Tech rept. 1349, CNRS University de paris-Sud,France, 2003.

[12] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic lookup network. In *Proc. of 21st ACM Symp on Principles of Distributed Computing (PODC)*, Monterey, CA, 2002.

[13] J. Xu, A. Kumar, and X. X. Yu. On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks. *IEEE Journal on Selected Areas in Communications(JSAC)*, 22(1), 2004.

[14] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. Graph-theoretic analysis of structured peer-to-peer systems: Routing distances and fault resilience. In *Proc. of ACM SIGCOMM' 2003*, pages 395–406, Karlsruhe, Germany, 2003. ACM Press.

[15] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In *Proc. of SIGCOMM'2003*, 2003.

[16] W. H. Kautz. The design of optimum interconnection networks for multiprocessors. *Architecture and Design of Digital Computer, NATO advances summer Institute*, pages 249–277, 1969.

[17] G. Panchapakesan and A. Sengupta. On a lightwave network topology using kautz digraphs. *IEEE Transaction on Computers*, 48(10):1131–1138, 1999.

[18] W. G. Bridges and S. Toueg. On the impossibility of directed moore graphs. *Journal of Combinatorial theory, series B*, 29:330–341, 1980.

[19] W. K. Chiang and R. J. Chen. Distributed fault-tolerant routing in kautz networks. *Journal of Parallel and Distributed Computing*, 20:99–106, 1994.

[20] D. S. Li, X. C. Lu, and J. S. Su. Graph-theoretic analysis of kautz topology and DHT schemes. In *Proc. of IFIP International Conference on Network and Parallel Computing 2004*, 2004.

[21] D. Eastlake and P. Jones. *RFC3174: US Secure Hash Algorithm 1 (SHA1)*. Available at http://www.faqs.org/rfcs/rfc3174.html, 2001.

Appendix

Appendix A: Proof of Theorem 3

Theorem 3: With high probability, the *Kautz_hash* algorithm (with parameters $p = 2$ and $n = 280$) can generate the

destination Kautz string of length 100 in one iteration, and Kautz strings generated are uniformly distributed in the Kautz namespace $KautzSpace(2, 100)$.

When parameters $p = 2$ and $n = 280$, the work that *Kautz_hash* algorithm finished in the first iteration is shown as below: First, concatenate three binary strings D_0, D_1 and D_2 into D (according to the characteristics of SHA-1 algorithm, D_0, D_1, D_2 are all 160-bit binary strings, and as a result, D is a 480-bit binary string). Following that, D is converted into ternary string R' (R' is a ternary string that is no longer than 303 bits, for $2^{480} < 3^{303}$). Taking the low n ($n = 280$) bits of R', we get a 280-bit ternary string R. Then we merge identical consecutive characters in R and get a new Kautz string Q with base 2. We will first prove that there is a less than 10^{-23} probability that the length of string Q less than 100, and then we will prove that the destination Kautz strings acquired from the lower 100 bits of Q is uniformly distributed in $KautzSpace(2, 100)$.

To prove Theorem 3, we first introduce some lemmas.

Lemma A.1: After merging identical consecutive characters, we can obtain a Kautz string Q with base 2 from an n-bit ternary string R. The probability of length of Q being m ($1 \leq m \leq n$) is as follows: $P(m) = 1.5 * 2^m * \binom{n-1}{m-1}/3^n$.

Proof: For each m-bit Kautz string $Q = b_1 b_2 \ldots b_m$ obtained after merging, let b_i be the result originated from the mergence of x_i consecutive b_i (i.e. the mergence of $\underbrace{b_i \ldots b_i}_{x_i}$) in ternary string R, then we may get:

$$\sum_{i=1}^{m} x_i = n \tag{1}$$

Based on theories of combinatorics, Equation (1) (with variable x_i) has $\binom{n-1}{m-1}$ positive roots, i.e., after merging identical consecutive characters; $\binom{n-1}{m-1}$ different ternary strings will be converted into the same Kautz string Q. There are $1.5 * 2^m$ Kautz strings in $KautzSpace(2, m)$, and any of two different Kautz strings Q_1 and Q_2 ($Q_1 \neq Q_2$) are converted from different ternary strings R_1 and R_2. Thus $1.5 * 2^m * \binom{n-1}{m-1}$ ternary strings would be converted to Kautz strings in $KautzSpace(2, m)$ after merging, and the total number of n-bit ternary strings is 3^n. From this we can know that the probability of acquiring Kautz strings with length m after merging identical consecutive characters is $1.5 * 2^m * \binom{n-1}{m-1}/3^n$. ■

According to Lemma A.1, let $n = 280$ and $m = 100$, we get:

$$\begin{aligned} P(100) &= 1.5 * 2^m * \binom{n-1}{m-1}/3^n \\ &= 1.5 * 2^{100} * \binom{279}{99}/3^{280} \\ &= 1.5 * 2^{00} * \frac{279!}{99! * 180!}/3^{280} \end{aligned}$$

Thus

$$\begin{aligned} \lg P(100) &= \lg 1.5 + 100 * \lg 2 + \lg 279! - \lg 99! \\ &\quad - \lg 180! - 280 * \lg 3 \\ &= -25.81043 \end{aligned}$$

For any $0 < m < 100$, $\binom{279}{m-1} < \binom{279}{100-1}$, therefore

$$P(m) < P(100) = 10^{-25.81043}$$

Thus in one iteration of *Kautz_hash* algorithm, after merging identical consecutive characters of the 280-bit ternary string R, the probability of generating a Kautz string Q with length less than 100 is less than $100 * 10^{-25.81043} < 10^{-23}$. Thus we get Lemma A.2.

Lemma A.2: With high probability, one iteration of *Kautz_hash* algorithm (with parameters $p = 2$ and $n = 280$) can generate a destination Kautz string ξ of length 100.

Lemma A.3: With high probability, the 280-bit ternary string R generated in *Kautz_hash* algorithm (with parameters $p = 2$ and $n = 280$) is uniformly distributed in interval $[0, 3^{280} - 1]$.

Proof: Based on the characteristics of SHA-1 algorithm, with high probability D_0, D_1, D_2 generated by SHA-1 algorithm is distributed evenly in interval $[0, 2^{160} - 1]$, therefore, D that is generated from D_0, D_1 and D_2 is uniformly distributed in the interval $[0, 2^{480} - 1]$ (i.e., $\forall i \in [0, 2^{480} - 1]$, $P(D = i) = 2^{-480} \triangleq P_0$). Thus the ternary string R' converted from D is also uniformly distributed in the interval $[0, 2^{480} - 1]$. Divide interval $[0, 2^{480} - 1]$ into segments with length 3^{280}, and let $2^{480} = r * 3^{280} + d$ ($0 \leq d < 3^{280}$), thereafter $r = \lfloor 2^{480}/3^{280} \rfloor = 79514548487$ (around $7.95 * 10^{10}$). Thus the interval $[0, 2^{480} - 1]$ has r complete segments, together with an interval left with length d, as shown in Figure 15. As a result, for any point R' within interval $[0, 2^{480} - 1]$, let $R' = r' * 3^{280} + d'$ ($0 \leq d' < 3^{280}$), then R (the lowest 280 bits of R') is d'.

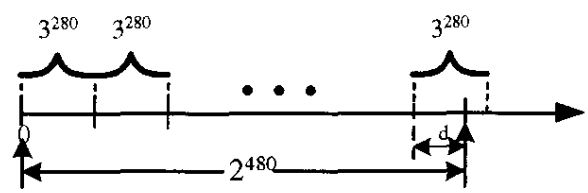


Fig. 15. Interval division.

Therefore, $\forall i \in [0, 3^{280} - 1]$, the probability $P(R = i)$ can be calculated as follows:

$$P(R = i) = \begin{cases} (r+1) * p_0 & (0 \leq i < d) \\ r * p_0 & (d \leq i < 3^{280}) \end{cases}$$

It can be validated that $\sum_{i=0}^{3^{280}-1} P(R = i) = 1$. Here, $r > 7.95 * 10^{10} \gg 1$, so $r + 1 \approx r$; thereafter with high probability the ternary string R is uniformly distributed in $[0, 3^{280} - 1]$. ■

Lemma A.4: With high probability, the destination Kautz string ξ obtained from *Kautz_hash* algorithm (with parameters $p = 2$, $n = 280$) is uniformly distributed in $KautzSpace(2, 100)$.

Proof: In *Kautz_hash* algorithm, it is obvious that any Kautz string ξ can be generated via merging identical consecutive characters of a certain 280-bit ternary string R in $[0, 3^{280} - 1]$ and then truncating the lowest 100 bits. Identical Rs can only be mapped to the same Kautz string in $KautzSpace(2, 100)$.

For any destination Kautz string $\xi = b_1 b_2 \ldots b_{100}$ acquired from *Kautz_hash* algorithm, let b_i in ξ stand for x_i $(1 \leq x_i \leq 280)$ consecutive b_i (i.e., $\underbrace{b_i \ldots b_i}_{x_i}$) in R before merging, thereafter

$$\sum_{i=1}^{100} x_i \leq 280 \tag{2}$$

Let the number of positive roots of Equation (2) (with x_i as the variables) be t, then t is a number that is independent of ξ. Each Kautz string ξ is converted from t different ternary strings in $[0, 3^{280} - 1]$. From Lemma A.3, with high probability the ternary string R is uniformly distributed in $[0, 3^{280} - 1]$, and as a consequence, ξ is uniformly distributed in $KautzSpace(2, 100)$. ■

Theorem 3 can be directly got from Lemma A.2 and Lemma A.4.

APPENDIX

APPENDIX B: PROOF OF LEMMA 2

Lemma 2: For each zone $U = u_1 u_2 \ldots u_k$ in FISSIONE, there are no zones $V = u_1 u_2 \ldots u_k x_1 \ldots x_j$ with $j \geq 1$ and $U \neq V$.

Proof: Lemma 2 holds initially for static Kautz graphs. We would show that if Lemma 2 holds at some time, Lemma 2 will also hold after a split or merge.

(1) In the case of a split, assume the large zone $V = v_1 v_2 \ldots v_k$ is divided into two zones $V_1 = v_1 v_2 \ldots v_k x_0$ and $V_2 = v_1 v_2 \ldots v_k x_1$ $(0 \leq x_0 < x_1 \leq 2, x_0 \neq v_k, x_1 \neq v_k)$. Obviously, either $v_1 v_2 \ldots v_k x_0$ or $v_1 v_2 \ldots v_k x_1$ is not a prefix of the other. Because $v_1 v_2 \ldots v_k$ is not a prefix of any zones' identifiers before the split, $v_1 v_2 \ldots v_k x_0$ and $v_1 v_2 \ldots v_k x_1$ would not be a prefix of any zones' identifiers. If there was a zone Y whose identifier is a prefix of $v_1 v_2 \ldots v_k x_0$ or $v_1 v_2 \ldots v_k x_1$ (without loss of generality we assume Y is a prefix of $v_1 v_2 \ldots v_k x_0$), as Y is not a prefix of $v_1 v_2 \ldots v_k$ before the split, therefore Y equals $v_1 v_2 \ldots v_k x_0$. Then a contradiction occurs for $v_1 v_2 \ldots v_k$ is a prefix of Y before the split. Therefore, Lemma 2 holds after the split.

(2) In the case of a merge, the proof is similar to that in (1) and omitted here.

Therefore, Lemma 2 holds. ■