Flexible Routing Tables: Designing Routing Algorithms for Overlays Based on a Total Order on a Routing Table Set

Hiroya Nagao, Kazuyuki Shudo
Tokyo Institute of Technology, Tokyo, Japan
Email: {hiroya.nagao, shudo}@is.titech.ac.jp

***Abstract*—This paper presents Flexible Routing Tables (FRT), a method for designing routing algorithms for overlay networks. FRT facilitates extending routing algorithms to reflect factors other than node identifiers.**

An FRT-based algorithm defines a total order on the set of all patterns of a routing table, and performs identifier-based routing according to that order. The algorithm gradually refines its routing table along the order by three operations: guarantee of reachability, entry learning, and entry filtering.

This paper presents FRT-Chord, an FRT-based distributed hash table, and gives proof that it achieves $O(\log N)$-hop lookups. Experiments with its implementation show that the routing table refining process proceeds as designed.

Grouped FRT (GFRT), which introduces node groups into FRT, is also presented to demonstrate FRT's flexibility. GFRT-Chord resulted in a smaller numbers of routing hops between node groups than both Chord and FRT-Chord.

I. Introduction

Numerous distributed hash tables (DHTs) have been proposed and actively researched over the last decade [1]–[7] . DHT routing algorithms provide scalability, fault tolerance, and reliability to overlay networks. Here, we focus on two features lacking in the routing algorithms of existing DHTs.

The first feature is *dynamic routing table size*. Existing DHTs limit routing table size, in other words the maximum number of routing table entries, to tens or hundreds. DHTs set the limitation because DHTs assume an unreliable, large-scale network such as the Internet, and it is difficult for nodes to maintain all of the millions of other nodes on such a network. Routing algorithms that achieve shorter path lengths with a small routing table have therefore been considered to be better algorithms. It is not always true, however, that the routing table size must be kept small. In OneHop [8] and EpiChord [9], for instance, routing table sizes are large and each routing table maintains a list of all nodes in the network. Moreover, it is difficult to estimate a suitable routing table size because the optimal size depends on node lifespan, node availability, and the number of nodes, all of which may change dynamically while the overlay network evolves and operates. It is therefore necessary to design a routing algorithm that can adapt to any routing table size and can change dynamically.

This work was supported by MEXT KAKENHI (22680005).

The second feature is a *node identifier consideration* that does not restrict routing table candidates. Each node has a node identifier, an address in an overlay network, and messages are forwarded according to those identifiers. Routing tables are therefore constructed using node identifiers. Existing routing algorithms reflect node identifiers by restricting routing table candidates to a subset of routing tables with some desirable property, such as $O(\log N)$-hop lookup performance in an N-node network. For instance, Chord [1] restricts node identifiers in a routing table at a node s to those nodes that most closely follow $s + 2^i$. Kademlia [5] restricts the number of nodes whose identifiers are $[2^i, 2^{i+1})$ away from s, based on XOR metrics, to be less than a constant k. Such restrictions are comprehensible and make data structures and construction processes simple, and are suitable to reflect only node identifiers. Such restrictions cause problems, however, not only in that routing tables do not know all nodes, despite the small number of nodes present, but also in that they eliminate opportunities to consider factors other than node identifiers, making extension of the routing algorithm problematic.

Algorithm extension is a promising approach to overcoming inherent problems in overlays, such as a number of relay nodes and insufficient consideration of network proximity. For instance, LPRS-Chord [10] and Coral [11] reflect network latency in routing tables, and Diminished Chord [12] and GTap [13] reflect node groups. Extendibility is a key property of DHTs and determines what extensions can be implemented. As mentioned above, however, the existing method of considering node identifiers interferes with constructing routing tables that are desirable in terms of such factors. The restriction on routing table candidates poses difficulty in considering factors other than node identifiers. It is difficult to achieve a balance between node identifier considerations and other factor considerations. A scheme of reflecting node identifiers without such restrictions is important for future applications.

As a solution we propose *flexible routing tables* (FRT), a method for designing routing algorithms for overlay networks. An FRT-based algorithm defines a total order $\leq_{\mathrm{ID}}$ as an indicator of the relative merits between node identifier combinations in a routing table, and continuously refines a routing table in accordance with the order. By doing so, the algorithm is able to dynamically change the routing table size and reflect node

Proc. 11th IEEE Int'l Conf. on Peer-to-Peer Computing (IEEE P2P'11), pp.72-81, August 2011

identifiers in a routing table without restriction on routing table candidates.

FRT has the following features.

Broadening of target domain: A node can route a message in $O(1)$-hop if the node can hold all nodes in an overlay. Otherwise, the algorithm routes messages as multi-hop lookups. The algorithm is also able to continuously change between those lookup styles without knowledge of the number of nodes.

Improved extendability: FRT supports simultaneous consideration of node identifiers and other factors because routing table construction does not restrict candidates for node identifier combinations in a routing table.

Effective utilization of entry information: In existing DHTs, routing tables tend to obtain only *entry information* needed for an eventual routing table, where entry information is knowledge needed to construct entries. Entry information that is not required, therefore, get ignored. An FRT-based algorithm does not ignore any entry information, and thus is able to construct better routing tables by evaluating all entry information.

Flexibility: An FRT-based algorithm is able to construct situation-dependent routing tables, based on, for example, the number of nodes in the system, node lifespan, node availability, performance requirements, without restriction on routing table size since the routing table is able to be resized dynamically.

Facilitation of node identifier considerations: An FRT-based algorithm expresses the relative merits of node identifier combinations in a routing table by a total order $\leq_{\mathrm{ID}}$, and thus the algorithm can consider node identifiers by referring only to an order on the routing table set. As a result, the algorithm is able to easily choose a routing table with more desirable node identifier combinations from among routing table candidates fulfilling complex restrictions on factors other than node identifiers.

Continual extension: FRT is able to continuously extend and improve existing routing algorithms for overlay networks by inserting additional entries into routing tables in the algorithms according to their order, which is defined in advance. FRT-based routing table construction protocols can be designed as extensions of existing algorithms while retaining all entries in the original routing table, and thus, at the least, we can retain the routing efficiency and other features of the original algorithm.

We describe a concrete FRT algorithm by taking *FRT-Chord*, a DHT we designed based on FRT, as an example. We also implement the proposed algorithm, and perform experiments.

In Section IV, we discuss how to design DHTs for future extension, and the extendibility of DHTs based on FRT.

II. Related Work

A. Chord

Chord is a distributed hash table (DHT), where node identifiers are represented as a circle of natural numbers from 0 to $2^m - 1$ in a clockwise direction (m is a bit length). The identifier space is called a *Chord ring*. The node responsible for a key is that node whose identifier most closely follows the key, called a *successor*.

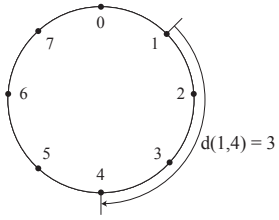


Fig. 1. Chord ring ($m = 3$).

In Chord, each node maintains three routing tables (successor list, predecessor, and finger table). Each entry in the routing tables is a node identifier and IP address pair. A node is able to send messages to another node with a specified node identifier, because the routing tables convert an node identifier to its corresponding IP address.

A successor list at node s contains a certain number of closest nodes from s in the clockwise direction, and a predecessor contains the closest node from s in the anticlockwise direction. These two routing tables are maintained by periodically running a stabilization routine, wherein the node sends messages to the successor for guaranteeing lookup correctness. The i th entry in a finger table is the first node that succeeds s by at least 2^{i-1} in the clockwise direction. Using the finger table significantly reduces the remaining distance.

In Chord, the identifier distance from x to y, $\mathrm{d}(x, y)$, is defined as follows (Fig.1).

$$\mathrm{d}(x,y) = \begin{cases} y - x, & \text{for } x < y \quad (1) \\ 2^m, & \text{for } x = y \quad (2) \\ y - x + 2^m, & \text{for } x > y \quad (3) \end{cases}$$

A query for a key identifier t is forwarded to a node s' in the routing table for which $\mathrm{d}(s', t)$ is minimized among any other entries. Routing schemes like this one that repeat forwarding through nearest nodes to a target are called *greedy routing* schemes. Chord achieves $O(\log N)$-hop lookup performance with N nodes.

B. EpiChord

EpiChord is a DHT in which the number of entries in the routing table is not limited. EpiChord divides the Chord ring into two symmetric sets of exponentially smaller slices, where the number of entries is greater than some constant at all times. This constraint provides an $O(\log N)$-hop guarantee on lookup path length. Such a constraint, however, also restrict candidates for node identifier combinations in routing tables. Although FRT, on the other hand, is able to construct routing tables from among all nodes, and FRT reflects node identifiers without such constraints.

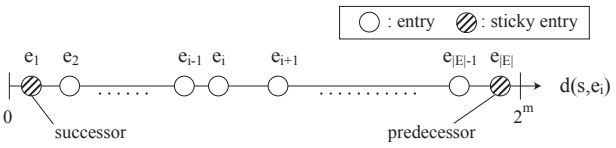


Fig. 2. A routing table $E = \{e_i\}_{i=1,\ldots,|E|}$.

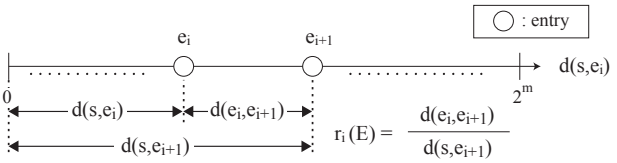


Fig. 3. the worst-case reduction ratio $\mathrm{r}_i(E)$ of a forwarding to a node e_i.

C. Symphony

Symphony [7] is also a DHT using a Chord ring. In Symphony, each node maintains two different entries, *short distance links* (SDLs) and *long distance links* (LDLs). SDLs are fixed entries maintained for reachability, namely a successor and a predecessor. Each LDL is determined probabilistically according to an identifier generated by a specific probability distribution based on the *Small World* phenomenon [14]. It therefore might seem that any node may be selected as an LDL, and that Symphony does not restrict candidates for node identifier combinations. However, each LDL is selected deterministically according to a probabilistically generated identifier, and thus, there is no opportunity to reflect factors other than node identifiers. FRT differs from Symphony in terms of flexibility of entry selections.

III. FRT-CHORD

In this section we describe *FRT-Chord*, an FRT-based DHT. In FRT-Chord the identifier space is a Chord ring and the node responsible for a key is determined as in Chord. FRT-Chord also performs a greedy routing as in Chord, but the method of routing table construction has some unique features.

In FRT-Chord, each node maintains a single routing table E without distinguishing between successor list, predecessor, and finger table because doing so restricts routing tables. The routing table E is a set of entries $\{e_i\}_{i=1,\ldots,|E|}$ (see Fig.2). Each entry e_i consists of a node identifier $e_i.\mathrm{id}$ and an IP address and a port pair $e_i.\mathrm{addr}$ (note that e_i is referred to as $e_i.\mathrm{id}$). A routing table $\{e_i\}$ at a node s is aligned clockwise from s, so it satisfies $i < j \Rightarrow \mathrm{d}(s, e_i) < \mathrm{d}(s, e_j)$. By this definition, e_1 and $e_{|E|}$ correspond with a successor and a predecessor, respectively. We assume that the correctness of these entries is guaranteed by the stabilization routine and regard them as *sticky entries* (Section III-D).

A. $\leq_{\mathrm{ID}}$: Total Order of the Routing Table Set

An FRT-based algorithm defines a total order $\leq_{\mathrm{ID}}$ on node identifier combinations in a routing table. The order represents the relative merits between routing tables in terms of node identifiers. The algorithm iteratively refines the routing table according to the order. In this section, we illustrate the design of the order $\leq_{\mathrm{ID}}$ in FRT-Chord, and for simplicity we set the length of the equivalent of the successor list to 1.

1) Definition of the Best Routing Table: Let $E.\mathrm{forward}(t)$ be an entry in a routing table E to which a query is forwarded at a node s toward a target identifier t. The *reduction ratio* of a forwarding of the query is defined as $\mathrm{d}(E.\mathrm{forward}(t), t)/\mathrm{d}(s, t)$. This means that the smaller the *reduction ratio*, the more efficient is the forwarding. Here we will focus on the *worst-case reduction ratio* $\mathrm{r}_i(E)$ of a forwarding to a node e_i, other than its predecessor. The *reduction ratio* of a forwarding to e_i takes the worst-case value when the key identifier t equals e_{i+1} (see Fig.3), so

$$\mathrm{r}_i(E) = \frac{\mathrm{d}(e_i, e_{i+1})}{\mathrm{d}(s, e_{i+1})}, \qquad (i = 1, \ldots, |E| - 1). \tag{4}$$

We define the *best routing table* $\tilde{E} = \{\tilde{e}_i\}$ as follows.

Definition 1: In FRT-Chord, the *best routing table* $\tilde{E} = \{\tilde{e}_i\}$ minimizes $\max_i\{\mathrm{r}_i(E)\}$.

Lemma 1:

$$\mathrm{r}_i(\tilde{E}) = 1 - \left(\frac{\mathrm{d}(s, \tilde{e}_1)}{\mathrm{d}(s, \tilde{e}_{|\tilde{E}|})}\right)^{\frac{1}{|\tilde{E}|-1}} \tag{5}$$

Proof: From (6), $\max_i\{\mathrm{r}_i(E)\}$ takes the minimum value when all of $\mathrm{r}_i(E)$ are equal.

$$\prod_{i=1}^{|E|-1} (1 - \mathrm{r}_i(E)) = \frac{\mathrm{d}(s, e_1)}{\mathrm{d}(s, e_{|E|})} (= \text{const.}) \tag{6}$$

■

Theorem 1: With high probability (or under standard hardness assumptions), assuming that all nodes have the *best routing table* with $O(\log N)$ entries in an N-node network, path lengths are $O(\log N)$.

Proof: Let $\tilde{E} = \{\tilde{e}\}$ be the *best routing table* at a node s. With high probability, the distance between two generic consecutive nodes is at least $2^m/N^2$ [15], namely

$$\mathrm{d}(s, \tilde{e}_1) > \frac{2^m}{N^2}. \tag{7}$$

The distance between any nodes is at most 2^m, namely

$$\mathrm{d}(s, \tilde{e}_{|E|}) < 2^m. \tag{8}$$

Thus, according to Lemma 1, for any $i = 1, \ldots, |\tilde{E}| - 1$,

$$\mathrm{r}_i(\tilde{E}) < 1 - \left(\frac{1}{N}\right)^{\frac{2}{|\tilde{E}|-1}}. \tag{9}$$

For $|\tilde{E}| = 1 + 2\log N$, the path length needed to reduce the remaining distance to $2^m/N$ or less is at most

$$\log_{\mathrm{r}_i(\tilde{E})} \frac{1}{N} < \log N. \tag{10}$$

The path length is therefore $O(\log N)$. When the remaining distance is at most $2^m/N$, the number of node identifiers landing in a range of this size is, with high probability, $O(\log N)$.

Thus the query reaches the key t within another $O(\log N)$ steps, meaning that the entire path length is $O(\log N)$. ■

2) Definition of $\leq_{\mathrm{ID}}$ based on $\mathrm{r}_i(E)$: In FRT-Chord, the order $\leq_{\mathrm{ID}}$ represents an indicator of closeness to the *best routing table*, and is defined as follows.

Definition 2: Let $\{\mathrm{r}_{(i)}(E)\}$ be the list arranged in descending order of $\{\mathrm{r}_i(E)\}$,

$$E \leq_{\mathrm{ID}} F \Leftrightarrow \{\mathrm{r}_{(i)}(E)\} \leq_{\mathrm{dic}} \{\mathrm{r}_{(i)}(F)\}. \quad (11)$$

In this definition, $\leq_{\mathrm{dic}}$ is a lexicographical order, namely

$$\{a_i\} <_{\mathrm{dic}} \{b_i\} \Leftrightarrow a_k < b_k, \qquad (k = \min\{i | a_i \neq b_i\}) \quad (12)$$

$$\{a_i\} =_{\mathrm{dic}} \{b_i\} \Leftrightarrow a_i = b_i \quad (13)$$

$$\{a_i\} \leq_{\mathrm{dic}} \{b_i\} \Leftrightarrow (\{a_i\} <_{\mathrm{dic}} \{b_i\}) \cup (\{a_i\} =_{\mathrm{dic}} \{b_i\}). \quad (14)$$

When we define $\leq_{\mathrm{ID}}$ as above, Theorem 2 holds.

Theorem 2: Let E be a candidate for a routing table and $\tilde{E}$ be a *best routing table*, $\tilde{E} \leq_{\mathrm{ID}} E$.

Proof: From Lemma 1 and Definition 2, $\tilde{E}$ is the minimum routing table candidate according to $\leq_{\mathrm{ID}}$. ■

We design an algorithm framework to use the order $\leq_{\mathrm{ID}}$ in three parts, *guarantee of reachability*, *entry learning*, and *entry filtering*. FRT-based algorithms consist of these three parts.

B. Guarantee of Reachability

In FRT, all operations to guarantee reachability are called a *guarantee of reachability*.

FRT-Chord's *guarantee of reachability* is a stabilization routine like Chord.

C. Entry Learning

We define *entry information* as information needed to compose an entry including a node identifier, an IP address, and a port number. In FRT, learning entry information and inserting the entry into a routing table are called *entry learning*. FRT does not limit how and when learning occurs so that opportunities to refine a routing table will not be wasted.

The following are examples of how FRT learns entries.

1) When a node first joins an overlay, it learns entries by transferring a routing table from a closest node from itself. This transfer is called *transfer at join*.
2) When a node communicates with another node when routing processes, it learns the connected nodes.
3) Nodes actively look up and learn entries with which they communicates, as in 2). These lookups are called *active learning lookups*.

In FRT-Chord, *active learning lookups* are similar to Symphony [7]. A node looks up a key generated from a probability distribution based on identifiers in the *best routing table*. The probability distribution at a node s is in inverse proportion to the distance from s. Letting $\mathrm{d}_s(x) = \mathrm{d}(s, x)$, the cumulative distribution function $\mathrm{F}(x)$ is defined as

$$\mathrm{F}(x) = \begin{cases} \dfrac{\ln \frac{\mathrm{d}_s(x)}{\ln \mathrm{d}_s(e_1)}}{\ln \frac{\mathrm{d}_s(e_{|E|})}{\ln \mathrm{d}_s(e_1)}}, & \text{for } \mathrm{d}_s(e_1) < x < \mathrm{d}_s(e_{|E|}) \quad (15) \\ 0, & \text{for otherwise.} \quad (16) \end{cases}$$

When the probability distribution is defined as above, letting rnd produce a random number between 0 and 1, the key is generated from the expression:

$$s + \mathrm{d}_s(e_1)(\mathrm{d}_s(e_{|E|})/\mathrm{d}_s(e_1))^{\mathrm{rnd}}. \quad (17)$$

FRT-Chord uses *transfers at join*. If the need arises, *active learning lookups* are performed.

In FRT, new entries learned by the above methods are inserted into the routing table. Through repeated *entry learning*, a node can forward a query to a closer node to a given key. As a separate issue, because the number of entries $|E|$ increases continuously, a node should prune entries at some future time.

FRT-Chord sets a *routing table size* L, the maximum number of entries, to prevent the number of entries from increasing without limit. L is configured dynamically and flexibly, based on the number of entries that should be retained according to node lifetime, machine performance, network latency, and so on.

For instance, if the *routing table size* L is larger than N, the network is stable, routing tables are able to contain all nodes in the system, and the algorithm achieves $O(1)$-hop lookup performance.

D. Entry Filtering

If $L < N$, when $|E|$ exceeds L, FRT-Chord will remove either the most recently learned new entry or entries in the current routing table in order to retain $|E| \leq L$.

In FRT, such entry removal operations are called *entry filtering*. Through continuous *entry learning* and *entry filtering*, FRT-based algorithms refine routing tables incrementally according to the order $\leq_{\mathrm{ID}}$.

An FRT-based algorithm defines some entries as *sticky entries*. FRT excludes *sticky entries* as removal candidates for *entry filtering*. For example, *short distance links* (SDLs) are one type of *sticky entry*. By designating *sticky entries*, we can easily design a total order $\leq_{\mathrm{ID}}$. *Entry filtering* in FRT is summarized as follows.

1) Substitute entries in E into C.
2) Remove *sticky entries* from C.
3) Select an entry from C to refine E according to $\leq_{ID}$.

In this way, FRT can consider node identifiers without restrictions on candidate node identifier combinations in a routing table with order $\leq_{\mathrm{ID}}$. We can also extend the algorithm with good results by introducing consideration of factors other than node identifiers when performing *entry filtering*.

In the rest of this section, we describe FRT-Chord *entry filtering* in detail.

Let e_{i^*} be an entry removed from a routing table E by FRT-Chord's *entry filtering*, and S_i^E be a *canonical spacing* defined as follows (see Fig.4).

Definition 3:

$$S_i^E = \log \frac{\mathrm{d}(s, e_{i+1})}{\mathrm{d}(s, e_i)}, \qquad (i = 1, \ldots, |E| - 1) \quad (18)$$

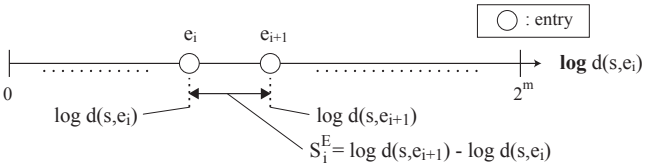


Fig. 4. The *canonical spacing* S_i^E.

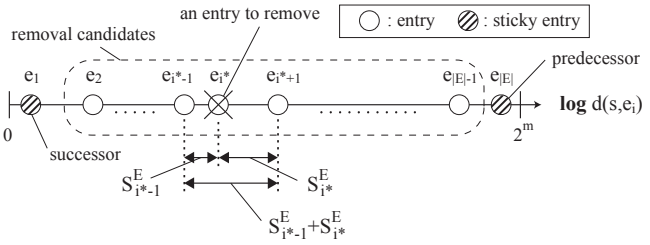


Fig. 5. FRT-Chord *Entry filtering*.

When S_i^E is defined as above, an entry e_{i^*} is selected from other than e_1 and $e_{|E|}$ because these entries are *sticky entries*, and (19) holds (see Fig.5).

$$S^E_{i^*-1} + S^E_{i^*} \leq S^E_{i-1} + S^E_i, \qquad (i{=}2,\ldots,|E|{-}1) \tag{19}$$

This way FRT-Chord can search for i^* at low cost by maintaining a list in ascending order of $S^E_{i-1} + S^E_i$, because only constant parts of the list need be changed with each *entry learning* and *entry filtering*.

Letting $\{S^E_{(i)}\}$ be the list arranged in descending order of $\{S^E_i\}$, the following lemma holds.

Lemma 2:

$$E \leq_{\mathrm{ID}} F \Leftrightarrow \{S^E_{(i)}\} \leq_{\mathrm{dic}} \{S^F_{(i)}\} \tag{20}$$

Proof: From the definitions of $\mathrm{r}_i(E)$ and S^E_i, we have

$$\mathrm{r}_i(E) = 1 - 2^{-S^E_i}. \tag{21}$$

Thus, small and large elements in the lists, $\mathrm{r}_i(E)$ and S^E_i, correspond with each other. ■

Theorem 3: In FRT-Chord, let $E \setminus \{e_{i^*}\}$ be a routing table filtered by removing e_{i^*}. For any $e_i (i = 2,\ldots,|E|-1)$,

$$E \setminus \{e_{i^*}\} \leq_{\mathrm{ID}} E \setminus \{e_i\}. \tag{22}$$

Proof: By calculating lists $\{S^{E-\{e_{i^*}\}}_{(j)}\}$ and $\{S^{E-\{e_i\}}_{(j)}\}$, arranged in descending order of the *canonical spacings* after *entry filtering* (note that $S^E_{i^*-1} + S^E_{i^*} \leq S^E_{i-1} + S^E_i$), the equation $\{S^{E\setminus\{e_{i^*}\}}_{(j)}\} \leq_{\mathrm{dic}} \{S^{E\setminus\{e_i\}}_{(j)}\}$ is derived, and $E \setminus \{e_{i^*}\} \leq_{\mathrm{ID}} E \setminus \{e_i\}$ holds by Lemma 2. ■

Theorem 4: In FRT-Chord, let $(E \cup \{e_{\mathrm{learn}}\}) \setminus \{e_{\mathrm{filter}}\}$ be a routing table after an *entry learning* process and a succeeding *entry filtering* process, where $e_{\mathrm{learn}} \notin E$ is a learned entry and $e_{\mathrm{filter}} \in (E \cup e_{\mathrm{learn}})$ is a removed entry in these processes.

$$(E \cup \{e_{\mathrm{learn}}\}) \setminus \{e_{\mathrm{filter}}\} \leq_{\mathrm{ID}} E \tag{23}$$

Proof: Since $(E \cup \{e_{\mathrm{learn}}\}) \setminus \{e_{\mathrm{learn}}\} = E$,

$$(E \cup \{e_{\mathrm{learn}}\}) \setminus \{e_{\mathrm{learn}}\} \leq_{\mathrm{ID}} E. \tag{24}$$

According to Theorem 3,

$$(E \cup \{e_{\mathrm{learn}}\}) \setminus \{e_{\mathrm{filter}}\} \leq_{\mathrm{ID}} (E \cup \{e_{\mathrm{learn}}\}) \setminus \{e_{\mathrm{learn}}\}. \tag{25}$$

Therefore, since $\leq_{ID}$ is a total order, (23) holds. ■

Theorem 4 means that FRT-Chord repeats to refine a routing table by *entry learning* and *entry filtering*.

In FRT-Chord, it is possible that routing table refinement through repeated *entry learning* and *entry filtering* will stop when any most recently learned entry is selected as an entry to remove. Such a routing table E is called a *convergent routing table*, and the following theorem holds.

Theorem 5: Assuming that all nodes have *convergent routing tables* with $O(\log N)$ entries in an N-nodes network, path lengths are $O(\log N)$ with high probability.

Proof: Let J be a set of i, where a node exists in a range from e_i to e_{i+1}, and K be a set of i otherwise. From the definition of *convergent routing tables*, for any $j \in J$, when an entry is inserted between e_j and e_{j+1}, that entry will be removed, and the following inequality holds.

$$S^E_j \leq S^E_{i-1} + S^E_i, \qquad (j \in J, i{=}2,\ldots,|E|{-}1) \tag{26}$$

Thus, by aggregating (26),

$$S^E_j \leq \frac{\sum_{i=2}^{|E|-1}(S^E_{i-1} + S^E_i)}{|E| - 2} \tag{27}$$

$$\leq \frac{2(\sum_{i=1}^{|E|-1} S^E_i)}{|E| - 2} = \log\left(\frac{\mathrm{d}(s, e_{|E|})}{\mathrm{d}(s, e_1)}\right)^{\frac{2}{|E|-2}}. \tag{28}$$

According to the definitions of $\mathrm{r}_j(E)$ and S^E_j, we can apply the proof of Theorem 1:

$$\mathrm{r}_j(E) < 1 - \left(\frac{1}{N}\right)^{\frac{4}{|E|-2}}. \tag{29}$$

When we consider the upper limit of path lengths needed to reduce the remaining distance to $2^m/N$ or less, we must focus on the case where each node forwards to $e_j (j \in J)$, because there is no node between e_k and e_{k+1} and the query forwarding will stop if the query is forwarded to e_k.

From (29), path lengths are $O(\log N)$ by similar reasoning to the proof of Theorem 1. ■

When we wish to set the length of successor lists at $c(> 1)$, we need only add the entries as *sticky entries*. In this case, an entry to remove will be selected optimally by the same filtering method, and routing tables will be refined continuously through *entry learning* and *entry filtering*.

As a result, we can summarize *entry filtering* in FRT-Chord as follows.

1) Substitute entries in E into C.
2) Remove *sticky entries* from C.
3) Select the entry e_{i^*} from C that minimizes $\{S^E_{i^*-1} + S^E_{i^*}\}$.

IV. Extensions of FRT

Existing DHTs consider node identifiers by putting restrictions on candidates for node identifier combination in a routing table to make path lengths short such as $O(\log N)$-hop lookup performance. To extend these DHTs, we must either construct routing tables under these restrictions or relax these restrictions altogether, and thus the restrictions too strongly limit opportunities to extend algorithms.

On the other hand, an FRT-based algorithm is able to flexibly reflect factors other than node identifiers in routing table construction, and node identifier consideration follows naturally because the algorithm manifests policies on how to evaluate each routing table according to node identifiers as an order $\leq_{\mathrm{ID}}$. The routing table is refined incrementally in terms of node identifiers even under restriction of factors other than node identifiers. This is the aspect most different from other DHTs.

A. GFRT-Chord

Grouped FRT (GFRT) is an extension of FRT. GFRT reflects node groups in routing tables at each node x by adding a policy to preferentially keep entries belonging to the same group as x in the routing table. GFRT-Chord achieves reduction of hops between nodes belonging to different node groups while keeping path lengths short. For instance, we can reduce communications over ISPs or data centers by configuring them as node groups. In GFRT, each node x belongs to a node group $x.\mathrm{group}$, and a group identifier is attached to a node identifier, and thus each entry e has node group information as $e.\mathrm{group}$.

Like FRT-Chord, GFRT-Chord also consists of three parts, *guarantee of reachability*, *entry leaning*, and *entry filtering*. It uses the same methods for *guarantee of reachability* and *entry learning*, and is characterized by its method for *entry filtering*.

1) Entry Filtering: GFRT-Chord maintains a *group successor list* and a *group predecessor* in a similar way to FRT-Chord. The group successor list and the group predecessor at a node s means a successor list and a predecessor respectively in a network limited to nodes belonging to the same group as s. In GFRT-Chord, therefore, *sticky entries* are a successor list, a predecessor, a group successor list, and a group predecessor.

We define the following variables for the routing table $E = \{e_i\}$ at a node s (see Fig.6).

- $E_G = \{e \in E | e.\mathrm{group} = s.\mathrm{group}\}$.
- $E_{\bar{G}} = \{e \in E | e.\mathrm{group} \neq s.\mathrm{group}\}$.
- e_α is the nearest entry in E_G from s.
- e_β is the farthest entry in E_G from s.
- $E_{\mathrm{near}} = \{e_i \in E | \mathrm{d}(s, e_i) < \mathrm{d}(s, e_\alpha)\}$.
- $E_{\mathrm{far}} = \{e_i \in E | \mathrm{d}(s, e_\alpha \leq \mathrm{d}(s, e_i) < \mathrm{d}(s, e_{|E|})\}$.
- $E_{\mathrm{leap}} = E_{\mathrm{far}} \cap E_{\bar{G}}$.

Using the variables defined above, GFRT-Chord performs *entry filtering* as follows:

1) Substitute $E_{\bar{G}}$ into C if $E_{\mathrm{leap}} \neq \emptyset$, otherwise substitute E into C.
2) Remove *sticky entries* from C.

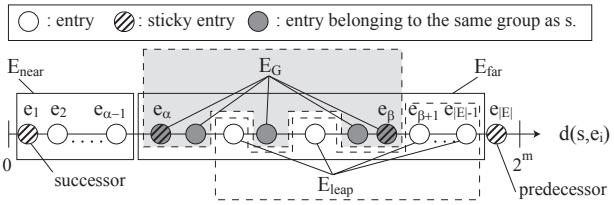


Fig. 6. Variables for GFRT-Chord *entry filtering*.

3) Select the entry e_{i^*} from C that minimizes $\{S^E_{i^*-1} + S^E_{i^*}\}$.

As above, *entry filtering* in GFRT-Chord consists of the filtering steps in FRT-Chord with the addition of only one step. The step 1) in particular represents the policy of preferentially maintaining entries belonging to the same group. The other two steps are the same as those in FRT-Chord, and these steps refine the routing table according to the order $\leq_{\mathrm{ID}}$. Thus, GFRT-Chord reflects node identifiers in routing table construction after reflecting node groups. In this way, all filtering steps in GFRT-Chord simultaneously reflect node identifiers and node groups.

From this, for convenience we set the length of successor lists and group successor lists as 1. Theorem 6 holds as in FRT-Chord.

Theorem 6: Let $E^* = E \setminus \{e_{i^*}\}$ be a routing table filtered by removing e_{i^*} according to the filtering operation of GFRT-Chord. For any entry e_i other than *sticky entries*, (30) and (31) hold.

$$\frac{|E^*_{\mathrm{far}}| - |E^*_{\mathrm{leap}}|}{|E^*_{\mathrm{far}}|} \geq \frac{|E_{\mathrm{far}}| - |E_{\mathrm{leap}}|}{|E_{\mathrm{far}}|} \tag{30}$$

$$E \setminus \{e_{i^*}\} \leq_{\mathrm{ID}} E \setminus \{e_i\} \tag{31}$$

Proof: We will prove Theorem 6 in two parts.

1) When $E_{\mathrm{leap}} = \emptyset$, (30) holds. (31) also holds because we can apply the proof of Theorem 3.
2) When $E_{\mathrm{leap}} \neq \emptyset$, since GFRT-Chord selects an entry e_{i^*} from E_{far} if and only if $e_{i^*} \in E_{\mathrm{leap}}$ according to the first step in *entry filtering*, $|E_{\mathrm{far}}|$ decreases by one if and only if $|E_{\mathrm{leap}}|$ decreases by one. So, $|E_{\mathrm{far}}|$ will not decrease by one without removing an entry from E_{leap}. Thus, (30) holds. (31) also holds because we can apply the proof in Theorem 3 under the restriction of the first step.

■

In Theorem 6, each of (30) and (31) means that the *entry filtering* reflects both node groups according to the ratio of the entries belonging to the same group as s and node identifiers according to the order $\leq_{\mathrm{ID}}$. Therefore, Theorem 6 represents a consideration of node groups and node identifiers simultaneously.

We define a *group localized routing table* at a node s as follows:

Definition 4: Let E be a *group localized routing table* at a node s belonging to a group G_s. When the node forwards a

message for a target identifier t to a node v in $E_{\bar{G}}$, there is no entry belonging to G_s in the range from v to t.

Definition 5: A *fully group localized routing table* E at a node s in a group G_s is defined as $E_{\mathrm{leap}} = \emptyset$.

Theorem 7: A *fully group localized routing table* E is a *group localized routing table*.

Proof: A forwarded node $v \in E_{\bar{G}}$ is in the range from s to the group successor of s. ■

Lemma 3: In GFRT-Chord, when a routing table E is a *convergent routing table* and $E_{\mathrm{leap}} \neq \emptyset$, all nodes belonging to G_s are in E.

Proof: We assume a hypothesis that E does not include a node belonging to G_s. If E learns the node, the entry will be removed during *entry filtering* because E is a *convergent routing table*. But this contradicts the first step in *entry filtering* in GFRT-Chord. Therefore, E has all nodes belonging to G_s. ■

Theorem 8: In GFRT-Chord, when a routing table E is a *convergent routing table*, E is a *group localized routing table*.

Proof: Let s be a node with a *convergent routing table* E and belonging to a group G_s, and consider forwarding a message for a target identifier t to a node $v \in E_{\bar{G}}$. We will prove Theorem 8 in three parts.

1) When v is $e_{|E|}$, since $e_{|E|}$ is the predecessor of s, there is obviously no entry in the range from v to t.
2) When v is in E_{near}, since v is nearer from s than e_α and t is in a range from v to e_α, there is no entry belonging to G_s in a range from v to t.
3) When v is in E_{far}, since there is at least an entry belonging to $E_{\bar{G}}$ in E_{far}, according to Lemma 3 there is no entry belonging to G_s in the range from v to t.

■

Theorem 9: Let $P = \{v_i\}$ be a set of nodes in a forwarding path from the source node $s = v_0$ to the predecessor of the responsible node v_{n-1}. In GFRT-Chord, when each node in the path has a *convergent routing table*, for any $v_i, v_j \in P(i < j)$,

$$v_i.\mathrm{group} = v_j.\mathrm{group} \Rightarrow i < \forall k < j, v_k.\mathrm{group} = v_i.\mathrm{group}. \tag{32}$$

Proof: Since Theorem 8 holds, each node in the path has a *group localized routing table*. We assume that v_i and v_j belong to a group G_0.

If we assume that the proposition is false, there exists $k(i < k < j)$ such that,

$$v_{k-1}.\mathrm{group} = G_0, \tag{33}$$

$$v_k.\mathrm{group} \neq G_0. \tag{34}$$

Since v_{k-1} belonging to G_0 forwarded a message to v_k belonging to a group other than G_0, then according to the definition of the *group localized routing table* there is no entry belonging to G_0 in the range from v_k to t. But this contradicts the hypothesis that v_j belonging to G_0 is in the range from v_k to t. ■

Thus, when all routing tables are sufficiently updated and they are *convergent routing tables*, once a node belonging to a group forwards a message to a node belonging to another group, the message will never be forwarded to nodes belonging to the group (except for the last hop). This is derived from the routing table construction of GFRT-Chord to reflect node groups.

Theorem 10: With high probability, assuming that all nodes have a *convergent routing table* with $O(\log N)$ entries in an N-node network, path lengths are $O(|G| + \log N)$ in a GFRT-Chord system composed of groups $G = \{G_i\}$.

Proof: We are focusing on a node s and its routing table E. Let G_s denote a group that s belongs to. Consider a case where s forwards a message to a node s' belonging to $G_{s'}$. If $G_s \neq G_{s'}$, then according to Theorem 9 the forwarding takes at most $|G| - 1$ times.

If $G_s = G_{s'}$, let s'' be an entry in either E_G or s, where s'' minimizes $\mathrm{d}(s', s'')$, and consider this situation in three parts.

1) When there is no node in the range from s' to s'', forwarding from s to s' will be the last one, so forwarding takes place only once.
2) When there is no node belonging to G_s but a node belonging to one of the other groups in the range from s' to s'', s' forwards a message to a node belonging to a group other than G_s. Thus, according to Theorem 9 the forwarding takes at most $|G| - 1$ times.
3) When there is a node belonging to G_s in the range from s' to s'', let j be an index of $e_j = s'$ and F be a set of entries in E other than *sticky nodes*. Then according to the definition of the *convergent routing table* the following inequality holds.

$$S_j^E \leq S_{i-1}^E + S_i^E, \qquad (e_i \in F) \tag{35}$$

Thus, since *sticky entries* are e_1, e_α, e_β and $e_{|E|}$, the following inequalities hold by aggregating these inequalities (note that the four entries may be equal to each other).

$$S_j^E \leq \frac{\sum_{e_i \in F}(S_{i-1}^E + S_i^E)}{|F|} \tag{36}$$

$$\leq \frac{2(\sum_{i=1}^{|E|-1} S_i^E)}{|E| - 4} = \log \left(\frac{\mathrm{d}(s, e_{|E|})}{\mathrm{d}(s, e_1)} \right)^{\frac{2}{|E|-4}} \tag{37}$$

Thus, the forwarding takes $O(\log N)$ times by similar logic to the proof of Theorem 5.

Therefore, the path length is $O(|G| + \log N)$. ■

B. Extendibility of FRT

We can easily design such extended algorithms because FRT offers us a simple way to reflect node identifiers in the design methodology composed of three steps, *guarantee of reachability*, *entry leaning*, and *entry filtering*, by defining a total order $\leq_{\mathrm{ID}}$ on the routing table set. It is important that FRT replaces consideration of how we should construct routing tables with consideration of what entries we should remove from the routing table. In the rest of this section, we will demonstrate the extendibility of FRT by taking GFRT-Chord as an example.

When we designed GFRT-Chord, we first decided to maintain some nodes in the routing table, such as the successor list, the predecessor, the group successor list, and the group predecessor. These are the *sticky entries*. DHTs often define exceptional entries to maintain in order not only to guarantee reachability but also to achieve fault tolerance, localize communications, or replicate data efficiently. FRT is designed to not interfere with such constraints for a routing table with *sticky entries* and offers a way to exclude these entries through *entry filtering*. No matter what entries we define as *sticky entries*, *entry filtering* can reflect node identifiers because the order on routing tables can be applied to any subset of the routing table set.

When we designed GFRT-Chord, we adopted the policy that it is better for a routing table at a given node to maintain more nodes belonging to the same group as that node. Factors we wish to introduce into routing algorithms are often independent of node identifiers because node identifiers are determined without regard of node characteristics, yet we must integrate such factors and node identifiers into an algorithm. FRT facilitates resolution of this difficulty. FRT converts rigid data structures that are hard to deal with into a single routing table by defining the order $\leq_{\mathrm{ID}}$ on candidates for the routing table. As a result, we can easily introduce factors other than node identifiers into routing algorithms by considering not how to keep better combinations of node identifiers but how nodes should be maintained based on the factors.

On the other hand, it is not always true that path lengths accurately represent routing efficiency, due to factors other than node identifiers. Rather, node identifiers should be maximally reflected in routing tables after sufficient reflection of other factors. FRT provides the ability to do this by detachment of the concerns of node identifiers from data structures, i.e. routing tables in the form of an order $\leq_{\mathrm{ID}}$. This way, we can reflect node identifiers in routing tables with sufficient reflection of other factors.

V. Evaluation

We implemented FRT-Chord on Overlay Weaver [16], [17], an overlay construction toolkit, and performed experiments.

A. Entry Learning and Entry Filtering in FRT-Chord

Here we will show that routing tables will approach the *best routing table* by *entry learning* and *entry filtering*, and confirm the effectiveness of *transferring at join* and *active learning lookups*.

In the experiments, the routing table size L is 80 and m is 160. We successively sent 10^5 queries, each query being sent to a randomly chosen key by a randomly chosen node in a system with $N = 10^4$ nodes. Next, we added a single node to the system and had it send 10^2 queries to a randomly chosen key or according to *active learning lookups*. We adopted an *iterative* style to route queries in all experiments, like EpiChord [9]. In this method, the first node on a path forwards queries by repeatedly referring current nodes to next nodes.

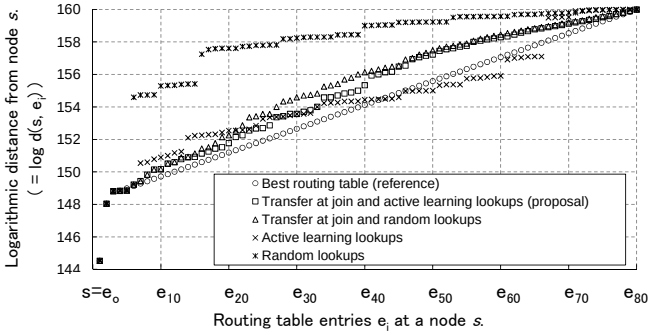


Fig. 7. Routing table entries after 25 lookups.

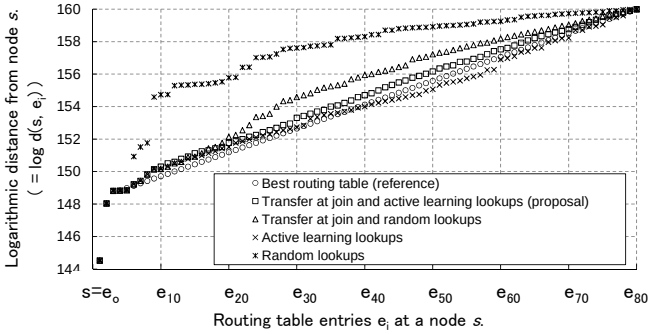


Fig. 8. Routing table entries after 100 lookups.

We varied whether each joining node receives an entire routing table from a successor (*transferring at join*) or not, and recorded node identifiers of entries in a routing table at the last joined node. We plotted $\log \mathrm{d}(s, e_i)$, the logarithmic distance from the node to each entry in its routing table after 25 queries and 100 queries, using its *best routing table* as a guide (Fig.7, Fig.8). The closer the graph of an experimental routing table is to the *best routing table*, the better its learning method. Fig.7 and Fig.8 show that routing tables approach the *best routing table* through repeated *entry learning* and *entry filtering*.

Fig.7 and Fig.8 also show that *transferring at join* and *active learning lookups* perform well because the routing tables without them are quite different from their *best routing table*. By comparing routing table learning only by random lookups to routing table learning by *transferring at join* along with random lookups, we can see that *transferring at join* is effective. This is because the node learns more entries by transferring from the successor in joining, and the *best routing table* for a node is similar to the *best routing table* of its successor. On the other hand, we can also see that even if a routing table does not obtain entries through a *transfer at join*, the routing table can still approach the *best routing table* through *active learning lookups*.

When we compare Fig.7 to Fig.8, we can see that routing tables with *active learning lookups* approach the *best routing table* more quickly than the others. This means that *active leaning lookups* are efficient for leaning as compared with random lookups.

These results show that *transferring at join* and *active learning lookups* work efficiently for learning, and *entry filtering* also works as expected.

B. Learning and Path Lengths in FRT-Chord

After N nodes join an FRT-Chord system, we repeat sending a query $50N$ times, where each query is sent to a randomly chosen key by a randomly chosen node. This means the average number of queries sent by a node is 50. We vary the number of nodes N and the routing table size L and we set the length of the successor list as 4.

Fig.9 indicates the average path lengths for every N queries. This figure shows that repeating lookups shortens the average path lengths. The number of lookups shortens the path lengths at almost the same range for every node, regardless of the number of nodes in the system. Thus, under FRT-Chord the system is able to scale at *entry learning* and *entry filtering*.

C. Path Lengths and the Number of Nodes in FRT-Chord

We varied N and L, and measured how path lengths change with greatly refined routing tables. Fig.10a and Fig.10b plot the average and the 99th percentile of path lengths. These figures show that FRT-Chord achieves $O(\log N)$-hop lookup performance, as described by Theorem 5. We can confirm that we are able to tune the trade-off between L and path lengths. When $L > N$ ($N = 10^2$, $L = 160$), routing tables have all nodes in the system and FRT-Chord achieves $O(1)$-hop lookup performance.

D. Path Lengths in GFRT-Chord

We also implemented GFRT-Chord on Overlay Weaver. Experiments with the implementation showed that average path lengths grow slightly as compared with FRT-Chord.

N nodes joined the system and nodes were composed of $|G|$ groups. Each group had $N/|G|$ nodes. Each node repeated an *active learning lookup* 500 times. The variables, N, $|G|$, and L are parameterized. The successor list length and the group successor list length were set as 4.

Fig.11a and Fig.11b plot average path lengths and average *group path length*. A *group path length* is the number of hops between two nodes belonging to different groups in a path, and thus a group path length is smaller than a path length.

In Fig.11a and Fig.11b, average path lengths in GFRT-Chord are larger than in FRT-Chord. This is because the routing table construction in GFRT-Chord is taken with the restriction of node groups, unlike FRT-Chord. In every situation, however, they differ slightly from each other, because the restriction of node groups in GFRT-Chord is not overly rigid and FRT is able to balance node identifier considerations and node group considerations in parallel. For example, for $|G| = 10$ and $L = 20$, when $N = 10^2$ each node group has only 10 nodes and the routing table at most includes only 10 nodes belonging to the same group. GFRT-Chord does not try to maintain the ratio of same group entries to other entries, but it uses the rest of the routing table at maximum and refines the entries according to the order $\leq_{\mathrm{ID}}$. GFRT-Chord therefore experiences only 1% path length growth, while attaining a 22% group path length decrease. Such percentages in path lengths growth and group path length decrease are not particularly important. This experiment shows that path lengths do not become overly large due to consideration of node groups in spite of the small number of nodes belonging to the same group. On the other hand, when $N = 10^3$ each node group has 100 nodes. GFRT-Chord is therefore able to choose entries belonging to the same group from a number of entry candidates according to the order $\leq_{\mathrm{ID}}$, and it treats entries not belonging to the same group like-wise. In this situation, therefore, GFRT-Chord also achieves only 6% path length growth, while decreasing group path length by 38%.

VI. Conclusion

This paper proposed *flexible routing tables* (FRT), a method to design routing algorithms for overlay networks, and proposed FRT-Chord, an FRT-based DHT.

An FRT-based algorithm is able to reflect node identifiers in routing table construction without restrictions on routing table candidates by defining and referring to a total order $\leq_{\mathrm{ID}}$ on a routing table set.

To analyze FRT-Chord, we implemented and experimented on the algorithm, and showed that the routing table is efficiently refined as expected, and that the algorithm achieves $O(N)$-hop lookup performance in an N-node network and $O(1)$-hop lookup performance in a small network.

This paper also proposed *Grouped FRT* (GFRT), an extended method based on FRT to reflect node groups, and designed GFRT-Chord, a GFRT-based DHT.

Experiments on GFRT-Chord show that GFRT-Chord reduces the number of hops from one group to another while avoiding long path. It shows the extendability of FRT-based algorithms, in that GFRT-Chord reflects node identifiers and node groups simultaneously.

We are finishing design of FRT-Kademlia, an FRT-based DHT with an identifier space based on XOR metrics. In future, we will design more FRT-based DHTs and extend them to deal with real world problems in addition to node grouping.

References

[1] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," in *Proc. ACM SIGCOMM '01*, 2001, pp. 149–160.

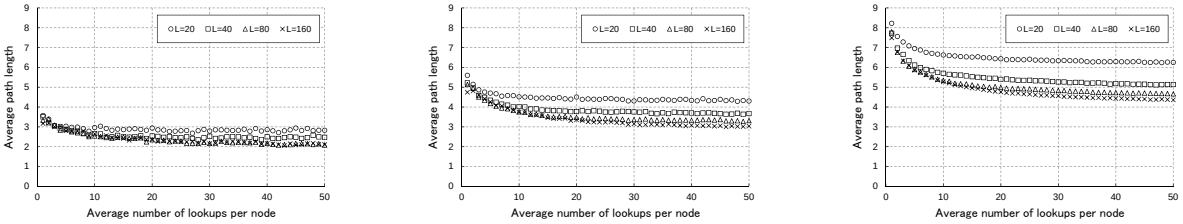
[2] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A Scalable Content-Addressable Network," in *Proc. ACM SIGCOMM '01*, 2001, pp. 161–172.

[3] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/CSD-01-1141, Apr 2001. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2001/5213.html

[4] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Proc. IFIP/ACM Middleware 2001*, 2001, pp. 329–350.

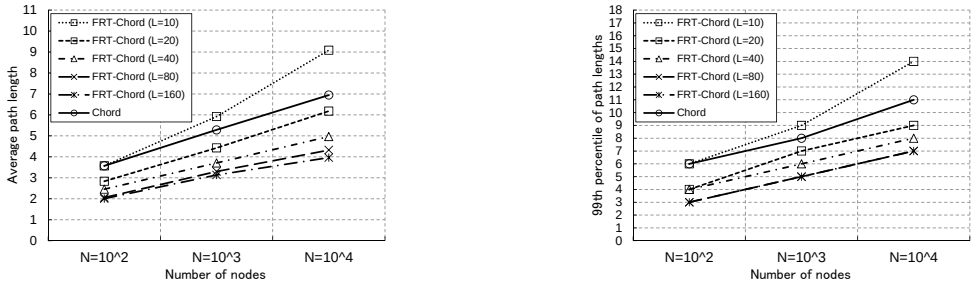
[5] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *Proc. IPTPS '02*, 2002, pp. 53–65.

[6] M. F. Kaashoek and D. R. Karger, "Koorde: A Simple Degree-Optimal Distributed Hash Table," in *IPTPS '03*, 2003, pp. 98–107.



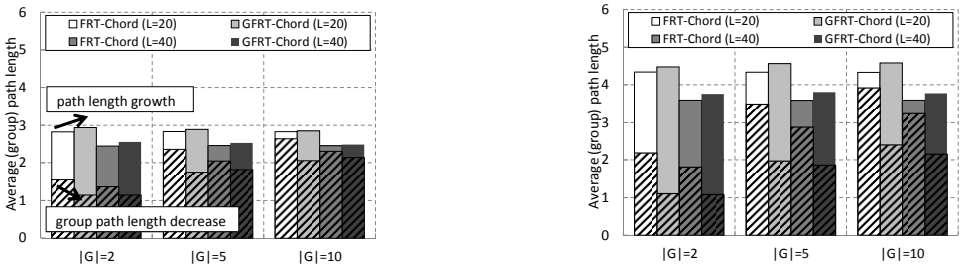
(a) $N = 10^2$ (b) $N = 10^3$ (c) $N = 10^4$

Fig. 9. Change in average path length with the number of queries per node.



(a) Average path length (b) 99th percentile of path length

Fig. 10. Correlation between routing table size and path length.



(a) $N = 10^2$ (b) $N = 10^3$

Fig. 11. Average path length with average group path length (shaded portion).

[7] G. S. Manku, M. Bawa, and P. Raghavan, "Symphony: Distributed Hashing in a Small World," in *Proc. USITS'03*, 2003, pp. 127–140.

[8] P. Fonseca, R. Rodrigues, A. Gupta, and B. Liskov, "Full-Information Lookups for Peer-to-Peer Overlays," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, pp. 1339–1351, September 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1591896.1592267

[9] B. Leong, B. Liskov, and E. D. Demaine, "EpiChord: Parallelizing the Chord Lookup Algorithm with Reactive Routing State Management," *Comput. Commun.*, vol. 29, pp. 1243–1259, May 2006. [Online]. Available: http://portal.acm.org/citation.cfm?id=1646651.1646839

[10] H. Zhang, A. Goel, and R. Govindan, "Improving lookup latency in distributed hash table systems using random sampling," *IEEE/ACM Trans. Netw.*, vol. 13, pp. 1121–1134, October 2005.

[11] M. J. Freedman and D. Mazières, "Sloppy Hashing and Self-Organizing Clusters," in *IPTPS '03*, 2003, pp. 45–55.

[12] D. R. Karger and M. Ruhl, "Diminished Chord: A Protocol for Heterogeneous Subgroup Formation in Peer-to-Peer Networks," in *IPTPS '04*, 2004, pp. 288–297.

[13] Y. Zhang, D. Li, L. C. 0002, and X. Lu, "Flexible Routing in Grouped DHTs," in *Proc. 2008 8th Int. Conf. Peer-to-Peer Computing*, 2008, pp. 109–118.

[14] J. Kleinberg, "The Small-World Phenomenon: An Algorithmic Perspective," in *Proc. ACM STOC 2000*, 2000, pp. 163–170.

[15] G. Cordasco and A. Sala, "2-Chord Halved," in *Proc. HOT-P2P '05*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 72–79. [Online]. Available: http://portal.acm.org/citation.cfm?id=1090948.1091377

[16] K. Shudo, Y. Tanaka, and S. Sekiguchi, "Overlay Weaver: An Overlay Construction Toolkit," *Comput. Commun.*, vol. 31, no. 2, pp. 402–412, 2008.

[17] K. Shudo, "Overlay Weaver: An Overlay Construction Toolkit," http://overlayweaver.sourceforge.net/.