論文 / 著書情報
Article / Book Information

題目(和文)	経路表間順序関係に基づく構造化オーバレイの経路表構築方法論
Title(English)	A Routing Table Construction Methodology Based on Routing Table Orders for Structured Overlays
著者(和文)	長尾洋也
Author(English)	Hiroya Nagao
出典(和文)	学位:博士（理学）, 学位授与機関:東京工業大学, 報告番号:甲第9747号, 授与年月日:2015年3月26日, 学位の種別:課程博士, 審査員:首藤　一幸,増原　英彦,渡辺　治,鹿島　亮,脇田　建
Citation(English)	Degree:, Conferring organization: Tokyo Institute of Technology, Report number:甲第9747号, Conferred date:2015/3/26, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Ph.D. Thesis, 2015

A Routing Table Construction Methodology Based on Routing Table Orders for Structured Overlays

Hiroya Nagao

Department of Mathematical and Computing Sciences
Graduate School of Information Science and Engineering
Tokyo Institute of Technology

Supervisor: Shudo Kazuyuki

February, 2015

Abstract

Structured overlay network algorithms are routing algorithms that construct routing tables whose entries are selected on the basis of logical positions defined in advance. Existing structured overlay algorithms construct desirable routing tables by restricting routing table candidates on the basis of logical positions. The restriction reduces flexibility to cope with a wide range of nodes and a varying number of routing table entries. Moreover, it restricts the ability to extend algorithms because routing tables that can be constructed on the basis of a metric other than logical positions are limited.

I propose flexible routing tables (FRT), an algorithm design framework for structured overlay routing algorithms that is designed to achieve the following desirable features: dynamic and arbitrary routing table size and network size adaptability without restricting the candidates for routing tables. FRT-based algorithms are characterized by constructing and maintaining routing tables using two procedures: entry learning and entry filtering. An entry learning procedure adds an entry corresponding to a node to be learned to its routing table. An entry filtering procedure evicts an entry from the routing table according to an order on the routing table space and a sticky entry function. Using these procedures, FRT-based algorithms can construct and maintain routing tables and achieve the desirable features.

I propose FRT-Chord, which is an FRT-based algorithm designed to demonstrate that concrete algorithms can be designed on the basis of FRT. Analyses and experimental results show that FRT-Chord performs as the design of FRT intended. Furthermore, FRT-Chord achieves desirable features derived from FRT such as dynamic and arbitrary routing table size and network size adaptability without restricting routing table candidates. As a result, FRT-Chord can seamlessly transition between the $O(1)$-hop and multi-hop routing, and its performance is optimized according to the size of routing tables. FRT-Chord repeatedly improves routing tables by the entry learning procedure and the entry filtering procedure, and I prove that the converged routing tables achieve $O(\log |N|)$ path length with high probability.

I also propose Grouped FRT-Chord (GFRT-Chord), which is an FRT-

based algorithm that extends FRT-Chord to construct routing tables that consider node groups as well as logical positions. GFRT-Chord achieves the reduction of inter-group hops while maintaining short path length derived from FRT-Chord and inherits dynamic and arbitrary routing table size and network size adaptability. This implies that an FRT-based algorithm can be extended to construct routing tables that consider metrics other than logical positions. This ability of FRT-based algorithms is important for real applications where the physical environment must be considered. I prove some desirable properties that result from assigning appropriate priority to node groups. Experimental results show that the path length and the inter-group path length are stably shortened with various patterns of algorithm and network parameters. This means that GFRT-Chord achieves a balance between logical position considerations and node group considerations.

In addition to FRT, I propose mergeable-FRT, an algorithm design framework for real applications that can consider two or more metrics in addition to path length by improving algorithm modularity and reusability. Mergeable-FRT offers a method to merge parts of extensions to mergeable-FRT-based algorithms. This method produces new algorithms that consider multiple metrics, i.e., those considered by the original algorithms (Note that this does not mean that the new algorithms automatically inherit all characteristics of the original algorithms). A mergeable-FRT-based algorithm supports the method to merge with other mergeable-FRT-based algorithms by defining an entry filtering procedure using the sequence of functions. I propose two merged algorithms—PGFRT-Chord and GPFRT-Chord—by reusing implementations of a mergeable-FRT-based GFRT-Chord and PFRT-Chord. Experimental result show that these algorithms reflects the features of the original algorithms.

By proposing these concrete algorithms, I demonstrate the features and abilities derived from FRT. I believe that the FRT frameworks allow us to design new algorithms that are based on various ideas and lead to a systematical design methodology for structured overlay algorithms.

Contents

List of Figures

Chapter 1

Introduction

Structured overlay network algorithms (structured overlays) are distributed algorithms that can deploy a messaging overlay network on top of an existing network with numerous computers such as the Internet. A structured overlay network offers a message routing function that is based on **logical positions**. Each node in the system has an assigned logical position* that is independent of the physical or geographical position of the node. As a result, nodes can send messages to these logical positions rather than IP or other address. Each node forwards the message hop by hop to deliver it to the destination node for the message. The destination node is defined based on logical positions.

Numerous structured overlays have been proposed, and actively researched over the last decade[1, 10, 12, 15, 18, 25, 26, 28–30, 41, 42, 47, 49, 55]. Structured overlay routing algorithms provide scalability, fault tolerance, and reliability to applications. Existing algorithms construct logical-position-based routing tables by restricting candidates for the routing table or limiting the ways by which entries can be learned.

I focus on two features that are lacking in the routing algorithms of existing structured overlays due to such routing table construction methods.

The first is **dynamic and arbitrary routing table size**. Many previously proposed algorithms fix the routing table size to $O(\log |N|)$ in $|N|$-node networks[1, 8, 30, 41, 47, 55]. Other algorithms[15, 16, 20, 25, 44, 49, 54] limit routing table size to less than $O(\log N)$ such as $O(1)$. In contrast to algorithms with small routing table sizes, other algorithms[9, 17, 18, 24, 33, 48] adopt a large routing table, e.g., $O(|N|)$; thus, there are various routing table sizes. However, all of these algorithms construct routing tables under fixed or limited routing table size constraints. In actual applications, a suitable

* Generally referred to as "node ID." However, I refer to this as "logical position" due to its meanings in this thesis.

routing table size is determined by various parameters such as the number of nodes, stability of nodes, resources, and the popularity of the application. As a result, it is difficult to predetermine appropriate routing table sizes for each application. Therefore, a dynamic and arbitrary routing table size is desirable for structured overlay algorithms.

The second feature is **restriction-free logical position consideration**. To construct routing tables, existing algorithms consider logical positions by restricting routing table candidates to a small subset of routing tables with some desirable properties, such as $O(\log |N|)$-hop lookup performance in an $|N|$-node network. For example, Chord[47] restricts the logical positions in a routing table at a node s to nodes that most closely follow $s + 2^i$. Kademlia[30] restricts the number of nodes whose logical positions are $[2^i, 2^{i+1})$ 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 they are suitable for considering only logical positions. However, such restrictions cause problems by which routing tables do not include all nodes in the system despite the small number of nodes. Additionally, they constrict opportunities to reflect factors other than logical positions and make designing extensions to the routing algorithm problematic.

Algorithm extension is a promising approach to overcoming inherent problems in structured overlay algorithms such as inadequate routing paths in terms of network proximity. In this thesis, extending a structured overlay algorithm means modifying the algorithm to consider additional factors other than logical positions. For example, LPRS-Chord[51] and Coral[11] construct routing tables with consideration of network latency, and Diminished Chord[21] and GTap[53] construct routing tables with consideration of node groups. Extensibility is a key property of structured overlays that determines what extensions can be designed and what applications can be implemented. However, as mentioned above, the existing method to consider logical positions interferes with constructing routing tables that are desirable in terms of such factors. The restriction on routing table candidates poses a difficulty in considering factors other than logical positions. It is difficult to achieve a balance between logical position considerations and other factors. A scheme to reflect logical positions without such restrictions is important for future applications.

To address these problems, I propose flexible routing tables[34] (FRT), which is an algorithm design framework for structured overlay algorithms

with desirable features such as dynamic and arbitrary routing table size and network size adaptability, without restricting routing table candidates.

FRT-based algorithms are characterized by constructing and maintaining routing tables using two procedures, entry learning and entry filtering. The entry learning procedure adds an entry to its own routing table, whereas the entry filtering procedure evicts an entry from the routing table according to a sticky entry function StickyEntry(E) and routing table order $\leq_{\mathrm{RT}}$. The sticky entry function is designed to guarantee message reachability and extend the algorithm. The routing table order is defined for routing tables as an indicator of the relative merits between logical position combinations in a routing table. Each node continuously refines its own routing table in accordance with the order using entry learning and entry filtering procedures. As a result, the algorithm can change routing table size dynamically and can consider logical positions in a routing table without restrictions on routing table candidates.

FRT has the following features.

Dynamic and Arbitrary Routing Table Size: An FRT-based algorithm can resize its routing table dynamically to be based on, for example, the number of nodes, node lifespan, node availability, and performance requirements.

Network Size Adaptability: A node can forward a message in $O(1)$ hops if the number of nodes is less than the capacity of entries in a single routing table. Otherwise, the algorithm forwards a message as multi-hop lookups. Moreover, the algorithm can seamlessly transition between those lookup styles without knowledge of the number of nodes.

Effective Utilization of Entry Information: In existing structured overlay algorithms, only entries required for the best routing table are added to routing tables; entries that are not required for the best routing table are ignored. An FRT-based algorithm does not ignore any entries, and thus it can construct routing tables by evaluating all node information.

Improved Extensibility: FRT offers a concrete extension point for considering factors other than logical positions and supports simultaneous consideration of logical positions and other factors. Improved extensibility is effectively utilized in GFRT-Chord (Chapter 6), PFRT-Chord (Section 7.4) and mergeable-FRT-based algorithms (Chapter 8).

In this thesis, I propose and describe the following FRT-based algorithms:

FRT-Chord[34], GFRT-Chord[35], FRT-2-Chord[2], FRT-Chord$^{\#}$[32], FFRT-Chord[3], and PFRT-Chord[31]. These algorithms are designed utilizing the FRT framework, and each algorithm demonstrates diverse features in addition to the desirable features and abilities derived from FRT.

FRT-Chord: An FRT-based algorithm, wherein nodes repeatedly improve routing tables using the entry learning and entry filtering procedures. It is proved that the converged routing tables achieve $O(\log |N|)$ path length with high probability. Experimental results show that the path length of FRT-Chord is approximately equal to Chord, and its performance is optimized according to the size of the routing tables. FRT-Chord achieves the desirable features derived from FRT, such as dynamic and arbitrary routing table size and network size adaptability with restriction-free routing table candidates. Thus, FRT-Chord can seamlessly transition between $O(1)$-hop routing and $O(\log |N|)$-hop routing. This feature is also achieved by all of the following algorithms.

GFRT-Chord (Grouped FRT-Chord): GFRT-Chord extends FRT-Chord to reflect node groups in addition to logical positions. GFRT-Chord reduces inter-group hops while maintaining short path length derived from FRT-Chord. I prove some desirable features are achieved as a result of giving priority to node groups appropriately. Experimental results show that path length and inter-group path length are stably shortened with various parameter patterns.

FRT-2-Chord: FRT-2-Chord adopts symmetric distance on logical positions. By adopting symmetric distance, FRT-2-Chord achieves symmetry of routing tables; when the routing table of node n_1 contains node n_2, the routing table of node n_2 tends to contain node n_1. Note that this property is lacking in FRT-Chord. This property reduces the cost of routing table maintenance and improves the efficiency of learning entries.

FRT-Chord$^{\#}$: FRT-Chord$^{\#}$ is characterized by routing table order on the basis of entries in neighbors' routing tables rather than logical positions. FRT-Chord$^{\#}$ supports non-uniform node logical position distributions and enables applications to support range queries while maintaining the desirable features derived from FRT.

FFRT-Chord (Flow-based FRT-Chord): FFRT-Chord demonstrates that nodes can construct efficient routing tables based solely on query flows. Thus, its routing table order is defined on the basis of the history of received and relayed queries. FFRT-Chord does not assume logical positions

of nodes and query targets are distributed uniformly. This is a practical merit that supports range queries. In contrast to FRT-Chord$^{\#}$, each node does not need to refer to the routing tables of other nodes in its routing table. Then, FFRT-Chord saves bandwidth.

PFRT-Chord (Proximity-aware FRT-Chord): PFRT-Chord extends FRT-Chord to reflect both logical positions and network proximities to routing tables. PFRT-Chord reduces routing latencies while maintaining the advantageous features of FRT.

In addition to FRT, I propose mergeable-FRT[36], another algorithm design framework to improve algorithm modularity. A mergeable FRT-based algorithm supports merging with other mergeable FRT-based algorithms by defining the entry filtering procedure using a sequence of functions (Note that this does not mean that the new algorithm automatically inherits all characteristics of the original algorithms). A merged algorithm can be defined by merging this sequences. I propose two merged algorithms, PGFRT-Chord and GPFRT-Chord that reuse implementations of mergeable FRT-based GFRT-Chord and PFRT-Chord. Experimental results show that these algorithms reflect the features of the original algorithms.

1.1. Composition of Thesis

In Chapter 2, structured overlay routing algorithms are introduced.

In Chapter 3, existing algorithms, i.e., Chord[47], Symphony[29], and Accordion[26] are discussed. These algorithms employ different routing table construction methodologies.

In Chapter 4, FRT[34], a framework for designing structured overlay routing algorithms, is described.

FRT-Chord[34] is discussed in Chapter 5.

In Chapter 6, GFRT-Chord[35], which considers node groups, is described.

In Chapter 7, other FRT-based algorithms, i.e., FRT-2-Chord[2] (Section 7.1), FRT-Chord$^{\#}$[32] (Section 7.2), FFRT-Chord[3] (Section 7.3), and PFRT-Chord[31] (Section 7.4), are introduced.

In Chapter 8, I propose the mergeable FRT framework[36], PGFRT-

Chord, and GPFRT-Chord. Note that PGFRT-Chord, and GPFRT-Chord are designed on the basis of the mergeable FRT framework.

Chapter dependencies are illustrated in Figure 1.1.

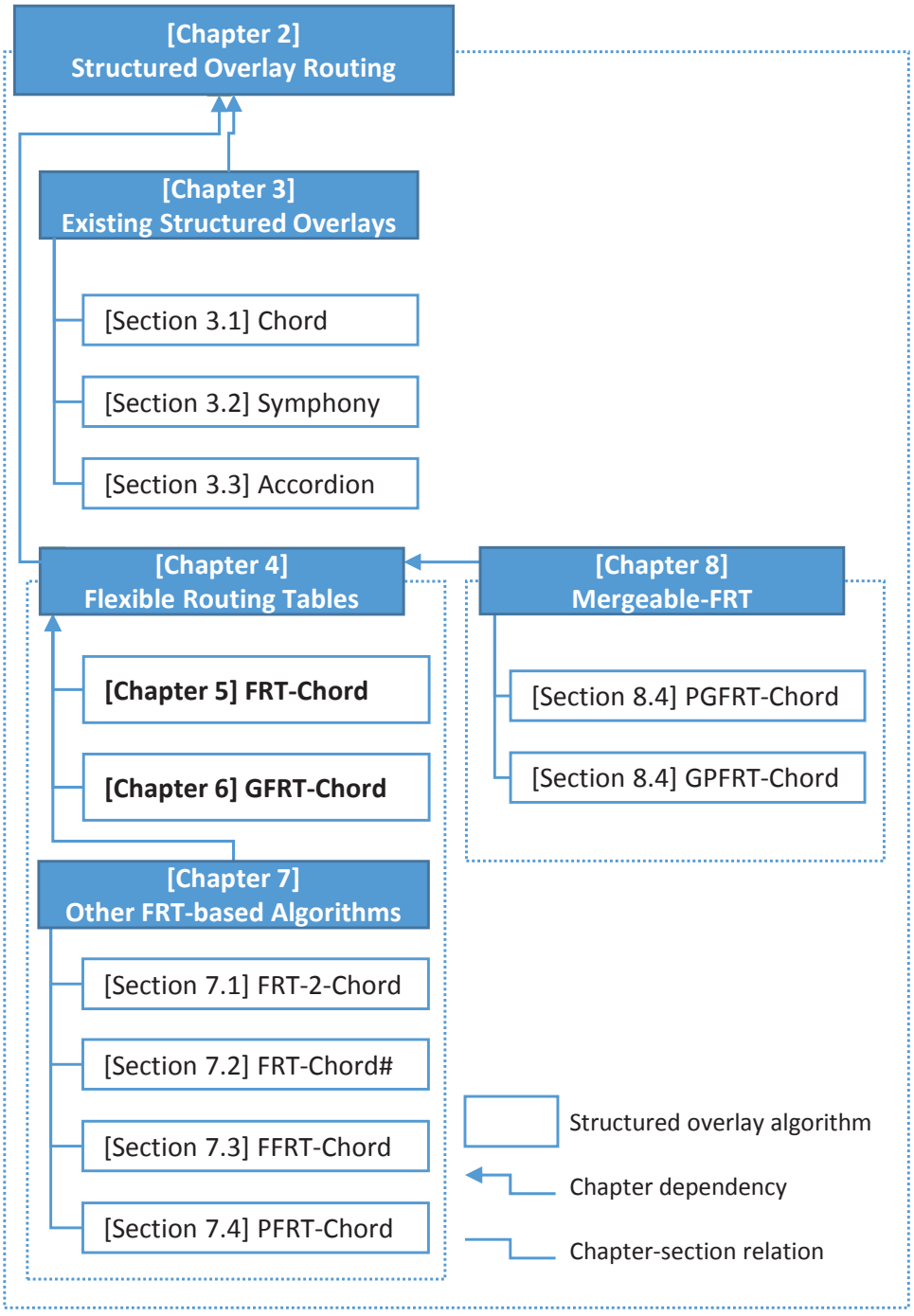


Figure 1.1: Chapter dependencies

Chapter 2

Structured Overlay Routing

Structured overlay routing algorithms (structured overlays) are distributed autonomous algorithms that deploy a messaging overlay network on the top of an underlay network, such as a TCP/IP network, that guarantees reachability to and from all nodes to all nodes.

Here, I introduce the basic common architecture of structured overlay routing algorithms.

2.1. Overview

Structured overlays are characterized by **routing tables that are constructed on the basis of logical positions**.

2.1.1. Logical Position and Forwarding

In structured overlays, each node in the system has a structured-overlay-specific address called a logical position assigned autonomously. Note that this is commonly independent of physical or geographical node position. Each message is sent to a target logical position, which is repeatedly transferred from a node to another according to the logical positions of both the destination and neighbors until the message reaches the destination node for the target logical position. For example, a message is transferred to the neighbor closest to its target on the basis of the remaining distance defined by logical positions (Figure 2.1).

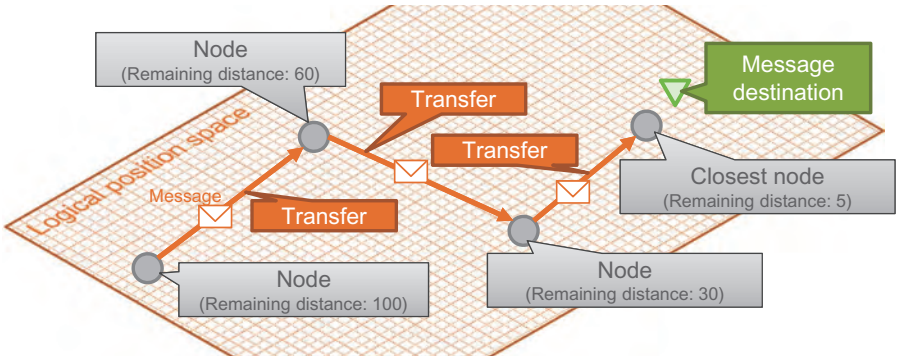


Figure 2.1: Message forwarding on the basis of the remaining distance defined by logical positions.

2.1.2. Routing Table and Forwarding

Each node builds its own **routing table**. Each entry of the routing table has at least two node properties, **logical position** and **underlay network address**, e.g., an IP address with a port number. When node s has an entry for node n with logical position n.lp and underlay address n.ua, node s can forward a message to another node whose logical position is n.lp by sending the message to n.ua using the underlay network. Each routing table entry represents a **neighbor** in the **network topology** of the structured overlay network.

In structured overlays, it is essential to build a routing table that will accurately reflect the topology of the overlay on the basis of logical positions. Since topology construction algorithms for structured overlays are so powerful, forwarding target selection algorithms can be simple and straightforward, which differs from routing algorithms that require prior advertisement of a node to the whole network (Figure 2.2).

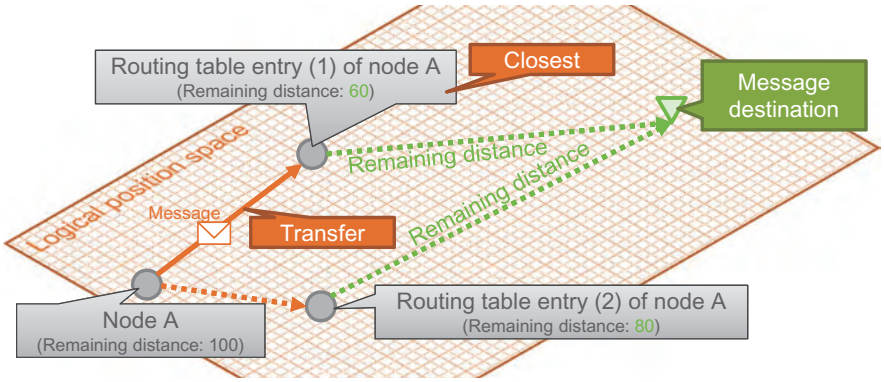


Figure 2.2: Node that chooses the closest node from two routing table entries on the basis of logical positions.

2.2. Logical Position

Each node $n \in N$ has **logical position** $n.\mathrm{lp} \in P$ and $n.\mathrm{ua}$ ($n.\mathrm{lp}$ is denoted simply as n). For example, many algorithms adopt an m-bit number as the universal set P.

Each node autonomously determines its logical position from P. Thus, a logical position is not a value assigned by a central server or an administrator. Thus, there is generally no node or central server that knows all of the logical node positions.

2.2.1. Routing and Responsible Node

Structured overlays offer a function to send a message to a logical position. A message is passed through one or more nodes. Then, the message reaches the node responsible for the logical position of the destination. Each algorithm defines which node is responsible for logical position $t \in P$ in a system with nodes N.

Definition 2.2.1: Responsible Node for Message

A **responsible node** $\text{res}^N(t)$ for a message is a node $n \in N$ where the message for logical position t terminates in the system with nodes N.

A **responsible domain** $\text{Dom}^N(n)$ is a set of logical positions defined as follows (Figure 2.3):

Definition 2.2.2: Responsible Domain

$$\text{Dom}^N(n) \overset{\text{def}}{\Leftrightarrow} \{t \in P \mid \text{res}^N(t) = n\} \tag{2.1}$$

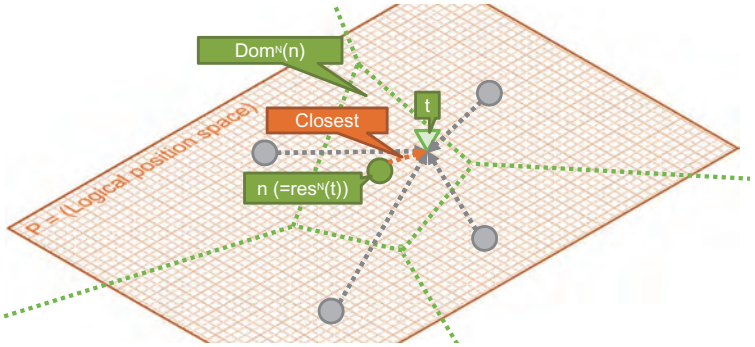


Figure 2.3: $\text{res}^N(t)$ and $\text{Dom}^N(n)$ in progressive structured overlays (Section 2.4).

A typical application utilizing structured overlays is a **distributed hash table** (DHT), which constructs a hash table on given nodes N and offers functions to put and obtain key-value pairs. For example, it can be implemented by storing a key-value pair $(\text{key}, \text{value})$ at node $\text{res}^N(\phi(\text{key}))$, where ϕ is a hash function that maps the key space to the logical position space P. The DHT's put and get methods can be implemented on top of a structured overlay message routing function.

2.3. Routing Table

A **routing table** is a set of candidates for an entry to which a message is forwarded. Thus, node s, which received a message for logical position t, selects another node $e = \mathrm{fwd}_s^{E_s}(t)$ from its own routing table $E_s = \{e_i^s\}_{i=0,\ldots,|E_s|-1}$*. Note s then forwards the message to the selected node e.

Note that a next hop selection function $\mathrm{fwd}_s^E(t)$ is defined; therefore, message flows are determined only by routing tables. In structured overlay, the topology (i.e., the routing tables) constructed by nodes is critical. There are two important features of network topologies, i.e., message reachability and the ability to shorten path length.

2.3.1. Routing Table Entry

Each **routing table entry** e has at least two columns, an address in the structured overlay layer (i.e., **logical position** e.lp) and an **underlay network address** e.ua (e.g., IP address with port number). Message forwarding from node s to a node whose logical position is e.lp is processed by sending the message to e.ua in the underlay network (Figure 2.4). In structured overlays, node e is considered a neighbor and a routing table is a list of neighbors. Thus, all routing tables comprise the network topology of the overlay network.

* Note that in this thesis, s's routing table E_s includes an entry corresponding to s as e_0, and the routing table $E_s = \{e_i^s\}_{i=0,\ldots,|E_s|-1}$ could be simply denoted $E = \{e_i\}$

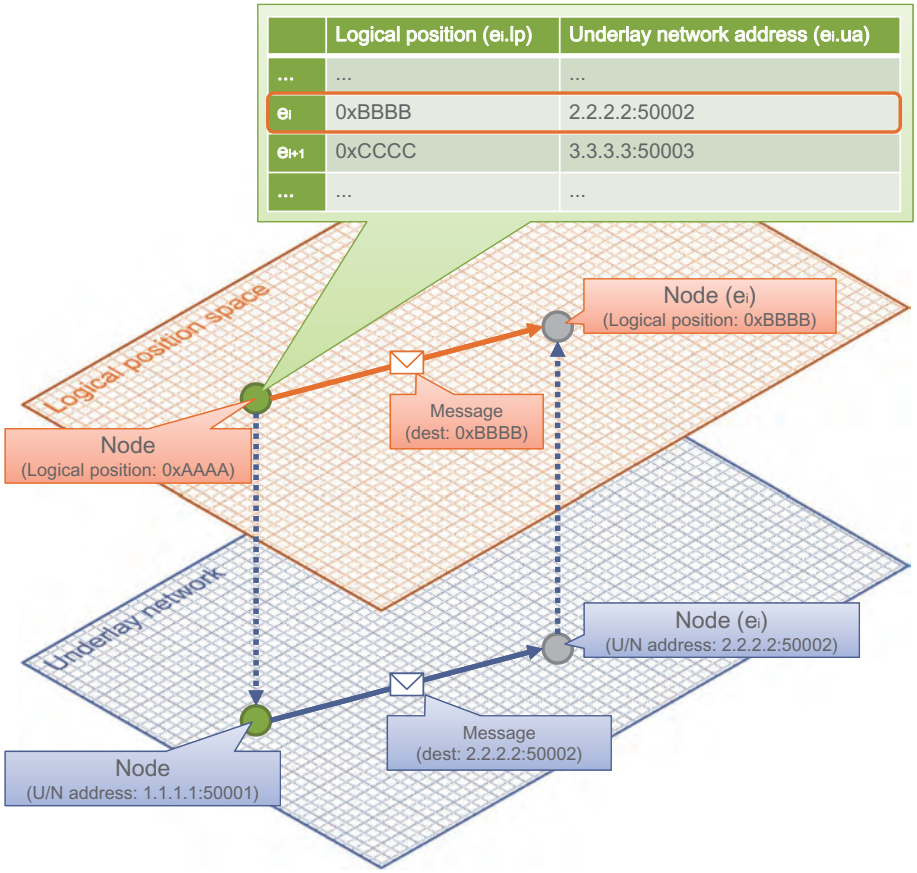


Figure 2.4: Message forwarding on the logical position space using a underlay network.

2.4. Progressive Structured Overlay

Progressive structured overlay is a structured overlay whereby each node forwards a message for a target logical position to reduce the **remaining distance** to the target logical position for each forwarding.

2.4.1. Remaining Distance

A progressive structured algorithm defines **remaining distance*** $\mathrm{d}(x, y)$ from logical positions x to y.

2.4.2. Responsible Node

In a progressive structured overlay with remaining distance function $\mathrm{d}(x, y)$, a responsible node $\mathrm{res}^N(t)$ is defined by the remaining distance as follows:

Definition 2.4.1: Responsible Node in Progressive Structured Overlay

$$\mathrm{res}^N(t) = \operatorname*{argmin}_{n \in N} \mathrm{d}(n, t). \tag{2.2}$$

This strategy means that the responsible node for t is assigned to the node closest to t among all nodes N.

2.4.3. Next Hop Selection

In a progressive structured overlay, node s with a routing table E forwards a message for t to node $\mathrm{fwd}_s^E(t)$, which is required to satisfy the following condition:

Definition 2.4.2: Progressive Routing

$$e = \mathrm{fwd}_s^E(t) \text{ s.t. } \mathrm{d}(e, t) < \mathrm{d}(s, t). \tag{2.3}$$

This means that s selects an entry $e \in E$ that is closer to t than s. This strategy is called **progressive routing**.

* Generally referred to as simply "distance". Although it is named "distance," the function does not always satisfy distance metric conditions such as symmetry and triangle inequality.

When the following condition is satisfied, such a strategy is called **greedy routing**.

Definition 2.4.3: Greedy Routing

$$\mathrm{fwd}_s^E(t) = \underset{e \in E}{\operatorname{argmin}}\, \mathrm{d}(e, t). \tag{2.4}$$

This means selecting the **closest** entry $e \in E$ to t (Figure 2.5).

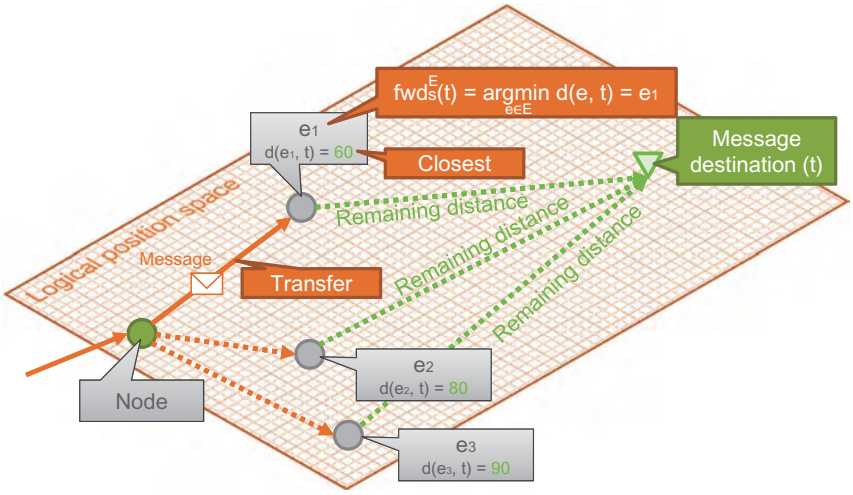


Figure 2.5: Message forwarding in greedy routing algorithms.

2.5. Conclusion

In this chapter, I have introduced the basic structure of routing algorithms for structured overlays.

In structured overlays, each node builds its own routing table. A message is forwarded according to logical positions to reach its responsible node. Because the entry to which each node forwards a message is defined by the next hop function $\mathrm{fwd}_s^E(t)$, message flows are determined by routing tables

E. Therefore, routing table construction is a critical part of structured overlays. Note that I focus on this construction method in this thesis.

Progressive routing is a routing strategy to reduce the remaining distance to a target logical position every forwarding, and greedy routing is a special case of progressive routing strategy where each node forwards a message to the closest node in its routing table. In this thesis, I primarily focus on structured overlay with greedy routing.

In the next chapter, I describe methods to build routing tables with typical examples.

Chapter 3

Existing Structured Overlays

In this chapter, I introduce concrete existing structure overlay algorithms, i.e., Chord, Symphony, and Accordion.

These algorithms are characterized by their own routing table construction methodologies. The routing table construction of Chord takes a typical approach. Each node of Chord constructs a routing table by collecting entries rigidly. However, Symphony and Accordion relax the restriction of routing table candidates by a probabilistic and a biased learning approach. However, Symphony cannot enlarge routing table size while maintaining the efficiency of routing tables. Moreover, Symphony has no space to consider factors other than logical positions. In contrast to Symphony, Accordion can increase the routing table size. However, Accordion limits its learning method and the behavior of its applications, and lacks extensibility.

In either approach, the desirable features described in Chapter 1 are not achieved.

3.1. Chord

Chord[47] is a protocol and algorithm for a DHT using a structured overlay network.

3.1.1. Remaining Distance

Chord performs progressive routing with **logical position space** $P = \{0, \ldots, 2^m - 1\}$ like a ring and **clockwise distance** $\mathrm{d}_{\mathrm{chord}}$ denoted $\mathrm{d}_{\mathrm{chord}}$:

Definition 3.1.1: Remaining distance in Chord

$$n, t \in P,$$
$$\mathrm{d}_{\mathrm{chord}}(n,t) \stackrel{\mathrm{def}}{=} t - n \bmod 2^m = \begin{cases} t-n, & (n < t) \\ t-n+2^m, & (t \leq n) \end{cases} \quad (3.1)$$

This logical position space is referred to as the **Chord ring** (Figure 3.1).

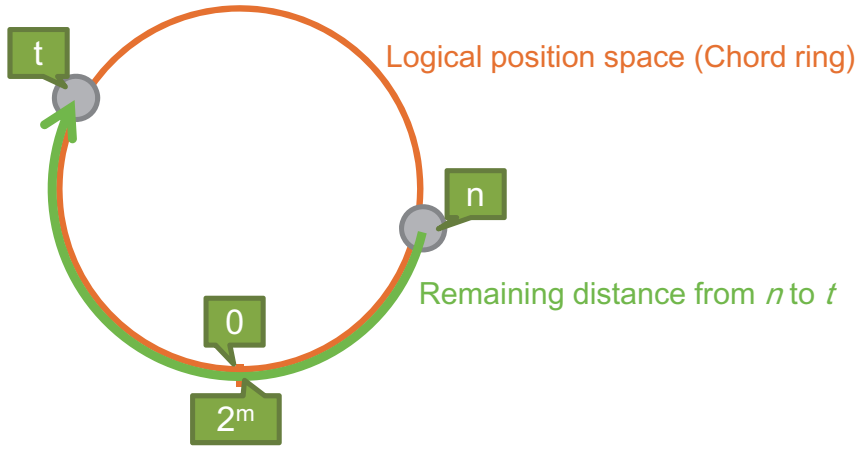


Figure 3.1: Chrod ring.

3.1.2. Responsible Node

The **responsible node** $\mathrm{res}^N(t)^*$ for a message for target logical position t in a network with nodes N is the closest node n to t, i.e., the node is a preceding node of t in the clockwise direction on the Chord ring.

* In Chord DHT, a value for key t is stored in the succeeding node of t. However, in this thesis, it is considered that message forwarding terminates in the node that follows t, and the initiator determines the succeeding node and stores the value in that node.

Definition 3.1.2: Predecessor Node

$$\text{pred}^N(t) = \underset{n \in N}{\text{argmin}}\, \text{d}(n, t). \tag{3.2}$$

Node n is referred to as a **predecessor** node $\text{pred}^N(t)$ for logical position t in a system with nodes N.

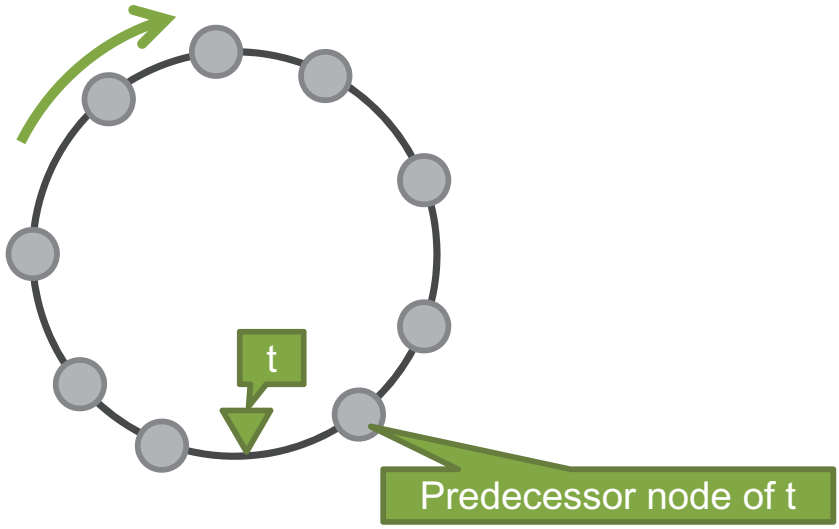


Figure 3.2: Predecessor node $\text{pred}^N(t)$.

Definition 3.1.3: Responsible Node in Chord

$$\text{res}^N(t) = \text{pred}^N(t). \tag{3.3}$$

3.1.3. Routing Table

Chord has two routing tables, **successor entry** (Section 3.1.4) and **finger table** (Section 3.1.5). These routing table entries are maintained by periodic procedures.

An example of these entries is briefly illustrated in Figure 3.3.

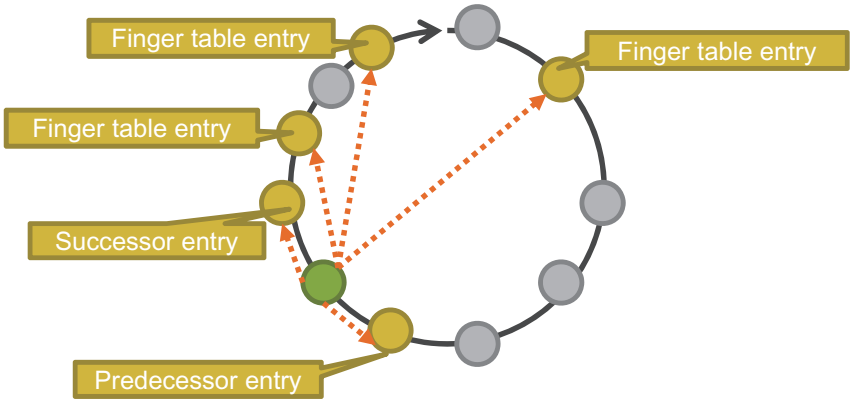


Figure 3.3: Successor entry (Section 3.1.4), predecessor entry (Section 3.1.6) and finger table entries (Section 3.1.5).

3.1.4. Successor Entry

A **successor entry** e_{succ} is an entry that satisfies the following constraints:

Definition 3.1.4: Successor Entry

$$e_{\text{succ}} = \text{succ}(s), \tag{3.4}$$

where a **successor** $\text{succ}(s)$ of s is defined as follows:

Definition 3.1.5: Successor Node

$$\mathrm{succ}^N(t) = \underset{n \in N}{\operatorname{argmin}}\, \mathrm{d}(t, n). \tag{3.5}$$

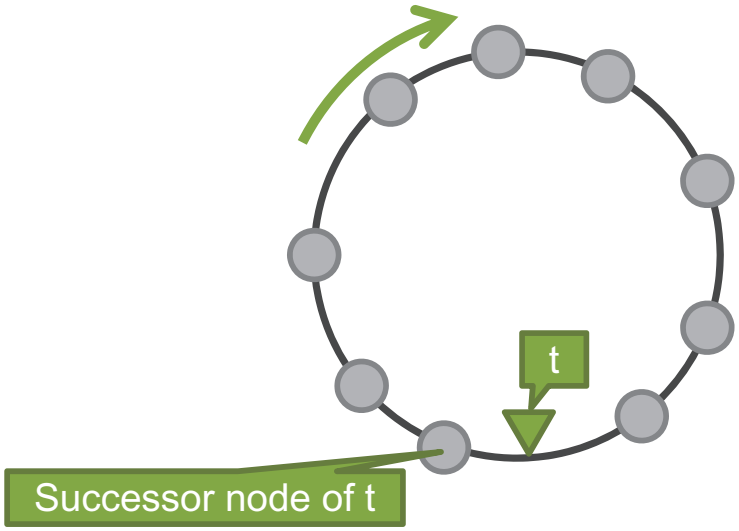


Figure 3.4: Successor node $\mathrm{succ}^N(t)$.

To obtain and maintain the successor entry, each node maintains an additional **predecessor** entry e_{pred}.

Definition 3.1.6: Predecessor Entry

$$e_{\mathrm{pred}} = \mathrm{pred}^N(s), \tag{3.6}$$

When joining the system, each node s looks up its own logical position s using the current network. Subsequently, s knows the responsible node $\mathrm{res}^N(s)$ for s and sets the predecessor entry of $\mathrm{res}^N(s)$ to the successor entry of s.

Each node periodically processes a procedure called **stabilization** to update the successor entry. The outline of the stabilization procedure as follows:

Procedure 3.1.1: Stabilization

1. Ask a current successor entry $e' = e_{\text{succ}}$ of s for a predecessor entry e'_{pred} of e'.
2. Update e_{succ} with e'_{pred} if $\mathrm{d}(s, e'_{\text{pred}}) < \mathrm{d}(s, e')$ and $e' \neq s$.

*In Step 1, before sending a reply, e' substitutes s to e'_{pred} if s is closer to e' than e'_{pred}

3.1.5. Finger Table

An i-th entry $e_{\text{fingers}[i]}$ of a **finger table** of node s satisfies the following constraint:

Definition 3.1.7: Finger Table

$$e_{\text{fingers}[i]} = \text{succ}(s + 2^i \bmod 2^m). \tag{3.7}$$

To update each entry in the finger table, s looks up the responsible node $\text{res}^N(s+2^i \bmod 2^m)$ for a logical position $s+2^i \bmod 2^m$ and updates the i-th finger table entry $e_{\text{fingers}[i]}$ with the successor entry of $\text{res}^N(s + 2^i \bmod 2^m)$.

3.1.6. Performance

The performance of Chord can be summarized as follows.

Routing Table Size

The number of entries in Chord's routing tables is $O(\log |N|)$. Although the number of finger table entries is m, all entries that are closer than the successor entry correspond to the successor entry.

Path Length

The path length is $O(\log |N|)$ with high probability because the remaining distance to the target logical position is shortened by half or more by forwarding due to the finger table entries.

Routing Table Flexibility

In Chord, each routing table entry is determined only for given nodes N. Thus, there is only one candidate for the routing table. Therefore, routing tables are rigidly restricted in Chord. Each node does not allow the addition of arbitrary entries.

3.2. Symphony

Symphony[29] is a structured overlay algorithm that adopts a ring logical position space from 0 to 1 and a clockwise remaining distance similar to Chord. Symphony is characterized by a scheme to gather routing table entries called long distance links that correspond to finger table entries in Chord.

Symphony maintains two types of routing table entries, a short distance link (SDL) and long distance links (LDL). An SDL is an entry that guarantees message reachability, which is the same as the successor entry in Chord. Conversely, LDLs are entries that reduce routing path length, which correspond to the finger table entry in Chord. Symphony performs greedy routing using these links.

Symphony guarantees message reachability by obtaining and updating the short distance link entries e_{SDL} in the same way as the successor entries in Chord. In contrast, each node determines each LDL by a method that

differs from Chord, i.e., LDL determination is based on the small world phenomenon[23]. In Chord, the i-th finger table entry is determined as the succeeding node of a logical position $x = s + 2^i$. In Symphony, each LDL is determined as the succeeding node of a logical position x randomly chosen from $1/|N|$ to 1 on the basis of the probability distribution function $p(x) = 1/(x \ln |N|)$. Symphony achieves $O(\frac{1}{k} \log^2 |N|$ expected path length with $k = O(1)$ LDLs gathered by such a probabilistic approach.

Symphony does not limit the patterns of routing table entries. However, because each entry is determined and fixed on the basis of the logical position generated by the probability distribution function, Symphony cannot reflect factors other than logical positions in future extensions. Moreover, symphony cannot arbitrarily increase the size of routing tables while maintaining the efficiency of routing tables because it takes a proactive routing table construction approach and discards node information learned by ordinary lookups.

3.3. Accordion

Accordion[26] also adopts the same logical position space and remaining distance as Chord. In Accordion, each node maintains routing table entries by entry learning and entry eviction. The entry eviction process selects entries on the basis of entry freshness, and Accordion considers logical positions by controlling which nodes are learned.

Each node receives entries from another node and adds these entries to its own routing table. By adopting recursive routing whereby intermediate nodes forward a message directly to the next hop, each node tends to communicate with closer nodes more frequently and with more distant nodes less frequently. By forcing applications to adopt recursive routing, Accordion limits communicating nodes to bias learning nodes so that routing tables learn entries with desirable distribution.

Accordion does not limit the patterns of routing table entries and can increase the size of routing tables; however, Accordion does limit the way entries are learned; i.e., a set of entries to be learned, and it limits applications. Accordion lacks extensibility because it considers logical positions by limiting communicating node during lookups.

3.4. Conclusion

I have introduced three approaches to constructing routing tables in Chord, Symphony, and Accordion. Symphony and Accordion relax the restrictions of routing table candidates by probabilistic and biased learning approaches. However, Symphony cannot increase the size of routing tables while maintaining the efficiency of routing tables and has no space to consider factors other than logical positions. Accordion can increase the size of routing tables. However, it limits the method by which entries are learned and the behavior of its applications. Note that Accordion also lacks extensibility.

In Chapter 4, I propose FRT to solve these problems by offering dynamic and arbitrary routing table size and network size adaptability without restricting routing table candidates and the use of routing tables in applications.

Chapter 4

Flexible Routing Tables

I propose flexible routing tables (FRT) [34], which is an algorithm design framework for structured overlay routing algorithms that is designed to achieve the desirable features described in Chapter 1, i.e., dynamic and arbitrary routing table size and network size adaptability without restricting routing table candidates.

FRT-based algorithms are characterized by maintaining routing tables using two procedures, entry learning and entry filtering. The entry learning procedure adds an entry corresponding to a node to be learned to the routing table. The entry filtering procedure evicts an entry from the routing table according to an order on the routing table space and a sticky entry function. Using these procedures, FRT-based algorithms can construct and maintain routing tables and achieve the desirable features.

In this Chapter, I introduce only the framework. I introduce its applications in Chapters 5, 6, 7, and 8.

4.1. Logical Positions and Remaining Distance

An FRT-based algorithm is a structured overlay algorithm that performs greedy routing with a logical position space P and a remaining distance function $\mathrm{d}(x, y)$.

In FRT, each node s has a single routing table $E = \{e_i\}$. Each entry e_i represents a candidate to which s forwards messages. Thus, E does not include entries for nodes not used as a forwarding target such as a predecessor in Chord. Each entry e_i has two properties, a logical position $e.\mathrm{lp} \in P$ and an underlay network address $e.\mathrm{ua}$ (e.g., IP address with port number).

4.2. Entry Learning Procedure

FRT offers two procedures, entry learning and entry filtering.

The **entry learning procedure** has one argument e_{new} that can be used as a candidate for a forwarding target. The procedure simply adds the argument to the current routing table (4.2).

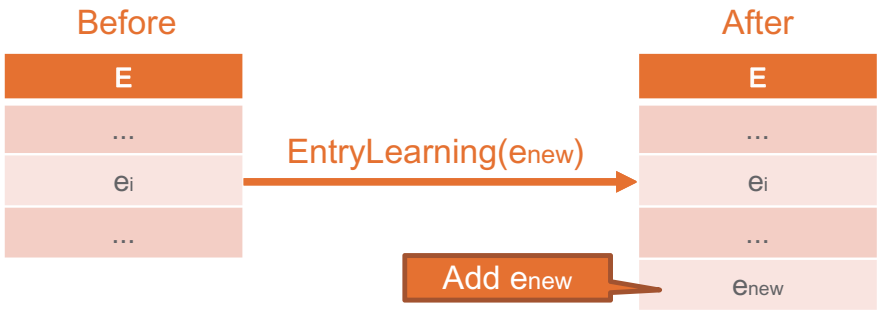


Figure 4.1: Evicts one entry in the entry filtering procedure.

Thus, it can be considered a "plus one" procedure. The entry filtering procedure is as follows:

Procedure 4.2.1: Entry Learning

[Step 1] `LET` E `=` $E^{+e_{\text{new}}}$,

where $E^{+e} = E \cup \{e\}$.

4.2.1. Execution Timing

The entry learning procedure is executed by various cues such as, but not limited to, the followings.

- Ordinary lookups* invoked by applications
- Active lookup at joining
- Active lookup to stabilize network
- Active lookup to increase routing table size
- Copy a routing table from a close node

Note that an argument entry is not added to a routing table if the routing table includes an entry that corresponds to the argument.

4.3. Entry Filtering Procedure

The **entry filtering procedure** has no arguments. It evicts a single entry from the current routing table E (Figure 4.2).

Figure 4.2: Evicts one entry in the entry filtering procedure.

Thus, it can be considered a "minus one" procedure.

The entry to be evicted is selected with routing table consideration based on a **sticky entry function** (Section 4.3.3) and a **routing table order** (Section 4.3.4).

* When a node looks up a node responsible to t, communications proceed around nodes on its forwarding path, and the nodes send or receive messages from unknown nodes, which are arguments of the entry filtering procedure.

4.3.1. Arbitrary Execution Timing

Each node can execute an entry filtering procedure whenever the node wants to reduce the number of entries in the routing table. FRT-based algorithms can achieve dynamic and arbitrary routing table sizes using the entry filtering procedure with the following three strategies to resize routing tables.

1. **Reduce**: For example, a routing table that includes excessive entries can be reduced using this procedure.
2. **Enlarge**: In contrast, in stable networks that allow large routing tables, nodes can increase the size of their routing table by not executing entry filtering.
3. **Maintain**: Nodes can also maintain the size of routing tables by executing entry filtering after entry learning.

Due to the arbitrary timing of the execution of the entry filtering procedure, FRT-based algorithms achieve network size adaptability. If the number of nodes is large, FRT-based algorithms repeatedly invoke entry filtering procedures to keep routing tables small and achieve short path length. On the other hand, even if the number of nodes in the system is smaller than the maximum number of routing table entries, routing table can keep information for all nodes.

4.3.2. Internal Structure

The entry filtering procedure is defined by the following steps using a sticky entry function StickyEntry(E) and a routing table order $\leq^{s}_{\mathrm{RT}}$, which will be introduced in the following sections.

Procedure 4.3.1: Entry Filtering

[Step 1] `LET` C `=` E.

[Step 2] `LET` C `=` $(C \setminus \mathrm{StickyEntry}(E))$.

[Step 3] `REMOVE` $\underset{c \in C}{\mathrm{argmin}}(E^{-c}, \leq_{\mathrm{RT}})$ `FROM` E.

Here, $E^{-e} = E \setminus \{e\}$, and the function argmin is defined as follows:

Definition 4.3.1: argmin with $\leq^{s}_{\mathrm{RT}}$

$$\underset{e \in E}{\mathrm{argmin}}(E^{-e}, \leq^{s}_{\mathrm{RT}}) = e^{*} \left(\text{s.t. } \forall e \in E, E^{-e^{*}} \leq^{s}_{\mathrm{RT}} E^{-e} \right). \tag{4.1}$$

This definition splits the consideration of path length and others into routing table order and a sticky entry function.

The behavior of entry filtering procedure is illustrated in Figure 4.3.

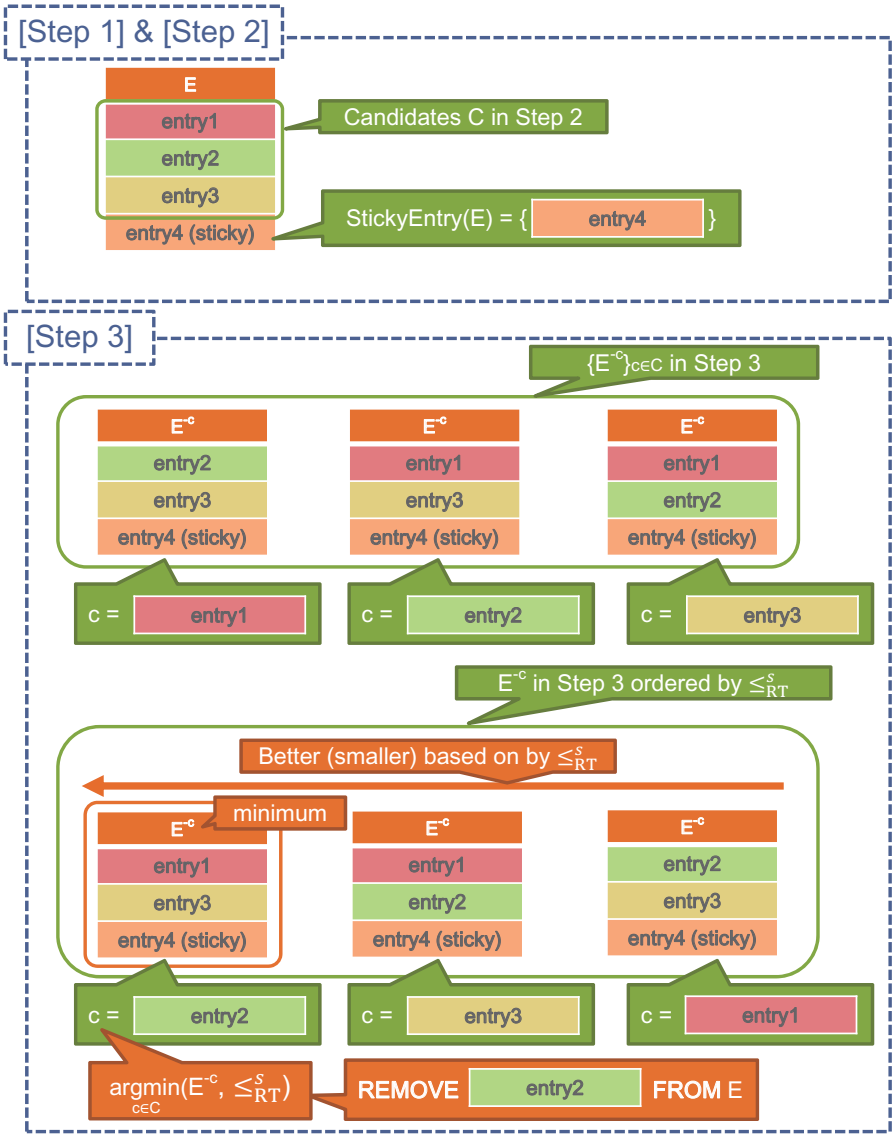


Figure 4.3: Sample behavior of entry filtering with 4 entries.

4.3.3. Sticky Entry Function

In the entry filtering procedure, the **sticky entry function** performs an important role. Algorithm designers can define a sticky entry function to specify entries that should not be evicted from routing tables (Procedure 4.3.1). By appropriately defining a sticky entry function, the entry filtering procedure maintains message reachability. A sticky entry function can also be designed to facilitate the extension of an existing algorithm. Note that the sticky entry function StickyEntry(E) must return a subset of E.

4.3.4. Routing Table Order

An entry filtering procedure selects an entry to be removed according to a statically predefined **routing table order** $\leq^s_{\mathrm{RT}}$ (Procedure 4.3.1). The routing table order can compare two routing tables. In this thesis, the direction of $\leq^s_{\mathrm{RT}}$ is considered that smaller table is better; i.e., an inequality $E_1 \leq^s_{\mathrm{RT}} E_2$ means that E_1 is better for node s than E_2. Algorithm designers can define a routing table order that represents how routing tables can reduce path lengths more effectively.

This internal structure of entry filtering facilitates variable and effective structured overlay algorithms such as FRT-Chord, GFRT-Chord, and FRT-2-Chord. These algorithms achieve the desirable features derived from FRT such as dynamic and arbitrary routing table size and network size adaptability without restricting routing table candidates.

Thus, by redefining the sticky entry function, algorithm designers can design extensions without considering logical positions to shorten path length. This feature is utilized in the definition of extensions such as GFRT-Chord and PFRT-Chord. Moreover, using this feature, mergeable-FRT, which is an algorithm design framework, improves algorithm modularity by separating extensions from the entry filtering procedure.

4.3.5. Restriction-free Routing Table Candidates

As mentioned above, FRT is characterized by the entry filtering procedure to consider the logical positions of entries in routing tables.

In Accordion[26], the learning-eviction structure also achieves dynamic and arbitrary routing table size. However, Accordion considers the logical

positions when learning entries. This structure limits the way entries are learned, i.e., the lookup process. Nodes are not allowed to perform lookup in an the iterative routing manner. More over, the structure limits the entries to be learned.

In contrast to Accordion, FRT-based algorithms do not need to limit the entries to be learned by considering logical positions in the eviction process rather than the learning process, i.e., the entry filtering procedure. Thus, FRT does not limit the design space of applications and does not limit the ability to learn entries.

Moreover, by defining the sticky entry function, FRT-based algorithms can maintain desirable entries in the routing tables independent of routing table order. Thus, FRT-based algorithms can achieve network size adaptability and restriction-free routing table candidates.

4.4. Reachability Guarantee Procedures

If the sticky entry function is appropriately defined, the entry filtering and entry learning procedures maintain routing tables such that message reachability of the current routing tables is not lost. However, because new nodes may join the network and some nodes may leave, each node must perform additional communications to maintain reachability. In FRT, such behaviors are referred to as **reachability guarantee procedures**, which are defined by algorithm designers. For example, obtaining a successor and stabilization in Chord are typical procedures classified as reachability guarantee procedures.

4.5. Conclusion

In this chapter, I have proposed FRT, which is an algorithm design framework for structured overlay routing algorithms, that is designed to achieve the desirable features described in Chapter 1, i.e., dynamic and arbitrary routing table size and network size adaptability without restricting routing table candidates.

FRT includes two procedures, entry learning and entry filtering. The entry

learning procedure adds an entry corresponding to a node to be learned to the routing table. FRT is characterized by considering logical positions in the entry filtering procedure. The entry filtering procedure evicts an entry from the routing table according to the order on the routing table space and a sticky entry function. Thus, FRT achieves dynamic and arbitrary routing table size and network size adaptability.

Note that, in this chapter, I have introduced only the framework. In the remaining chapters, I will introduce concrete algorithms designed based on FRT that achieve the advantageous features that existing algorithms lack.

In Chapter 5, I introduce FRT-Chord, a concrete FRT-based algorithm that achieves short path length and functions according to FRT design. In Chapter 6, I introduce GFRT-Chord, a concrete extension of an FRT-based algorithm that considers logical positions and node groups. In Chapter 7, I introduce other FRT-based algorithms. Each algorithm is designed to achieve a different goal. In Chapter 8, I introduce mergeable-FRT, which is another framework based on FRT. Mergeable-FRT is designed to produce algorithms that consider two or more metrics other than path length.

Chapter 5

FRT-Chord

In Chapter 4, I introduced the structure of the FRT framework. However, FRT is only a framework; thus it is unclear whether concrete algorithms can be designed on the basis of the FRT framework. In this chapter, I present the design and implementation of an FRT-based algorithm named FRT-Chord[34], which adopts the remaining distance of Chord. Analyses and experimental results show that FRT-Chord works according to FRT design. Moreover, FRT-Chord achieves desirable features derived from FRT such as dynamic and arbitrary routing table size and network size adaptability without restricting routing table candidates.

In FRT-Chord, nodes improve routing tables repeatedly by invoking the entry learning and entry filtering procedures. I prove that the converged routing tables achieve $O(\log |N|)$ path length with high probability.

FRT-Chord is the basis of other FRT-based algorithms introduced in later chapters.

5.1. Sticky Entry Function

In FRT-Chord, let routing table $E = \{e_i\}$ of node s be aligned in a clockwise direction from $s = e_0$. Then, a sticky entry function FRT-Chord::StickyEntry(E) is defined as follows:

Definition 5.1.1: Sticky Entry of FRT-Chord

$$\text{FRT-Chord::StickyEntry}(E) = \{e_0, e_1\}. \tag{5.1}$$

In this definition, entry e_1 corresponds to a successor entry in Chord.

The sticky entry function can be defined to give redundancy to the successor entry as follows:

$$\text{FRT-Chord::StickyEntry}(E) = \{e_0, e_1, \ldots, e_k\}. \tag{5.2}$$

The sticky entry function can be defined to maintain predecessor entry in the routing table as follows:

$$\text{FRT-Chord::StickyEntry}(E) = \{e_0, e_1, \ldots, e_k, e_{|E|-1}\}. \tag{5.3}$$

In this case, since the predecessor entry may not have joined the system, a node selects the predecessor as the next hop entry with caution. In this thesis, for simplicity, let k be 1 and assume that the predecessor is excluded from the routing table.

5.2. Routing Table Order

In FRT-Chord, a routing table order is defined according to the reduction ratio, a ratio that represents how much the remaining distance can be reduced by a single forwarding.

Definition 5.2.1: Reduction ratio

A reduction ratio for forwarding a message for logical position t from node s to node e_i is defined as follows:

$$\frac{\mathrm{d}(e_i, t)}{\mathrm{d}(s\ , t)}. \tag{5.4}$$

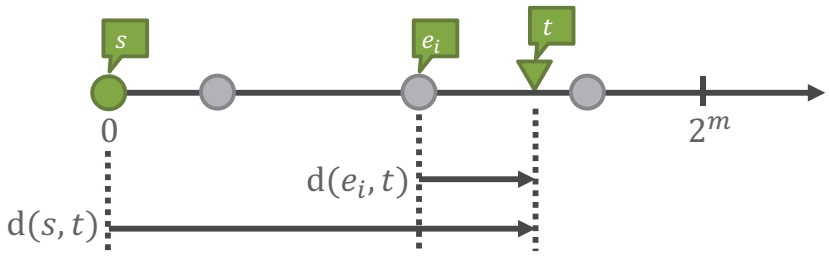


Figure 5.1: Reduction ratio.

This definition corresponds to the remaining distance after forwarding divided by the remaining distance prior to forwarding.

FRT-Chord utilizes the **worst-case reduction ratio** r_i^E. This worst-case reduction ratio of each entry is defined as the maximum reduction ratio of a forwarding to that entry. In the distance of Chord, r_i^E can be calculated as follows.

Definition 5.2.2: Worst-case Reduction Ratio

The worst-case reduction ratio r_i^E for entry e_i in routing table $E = \{e_i\}_{i=0,\ldots,|E|-1}$ is defined as follows:

$$\mathrm{r}_i^E (= \mathrm{r}^E(e_i)) = \frac{\mathrm{d}(e_i, e_{i+1})}{\mathrm{d}(s\ , e_{i+1})},\ (i = 1, \ldots, |E| - 1, e_{|E|} = s). \tag{5.5}$$

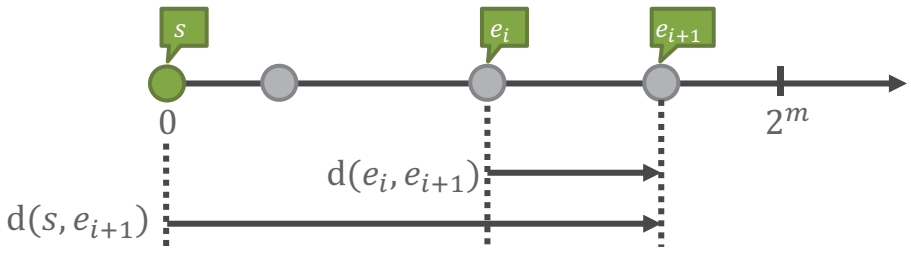


Figure 5.2: Worst-case reduction ratio.

Definition 5.2.3: Routing Table Order in FRT-Chord

Let $\left(\mathrm{r}_{(i)}^E\right)_{i=1,\ldots,|E|-1}$ be the worst-case reduction ratio sequence arranged in descending order of $\{\mathrm{r}_i(E)\}_{i=1,\ldots,|E|-1}$. Subsequently, the routing table order in FRT-Chord $\leq_{\text{FRT-Chord}}$ is defined as follows:

$$\begin{aligned} & E_1 \leq_{\text{FRT-Chord}} E_2 \\ \overset{\text{def}}{\Leftrightarrow}\ & \left(\mathrm{r}_{(i)}^{E_1}\right)_{i=1,\ldots,|E_1|-1} \leq_{\text{dic}} \left(\mathrm{r}_{(i)}^{E_2}\right)_{i=1,\ldots,|E_2|-1}, \end{aligned} \tag{5.6}$$

where $\leq_{\text{dic}}$ is a lexicographical order.

This routing table order compares the worst-case reduction ratios of entries in the two routing tables. Here, $\leq_{\text{FRT-Chord}}$ is designed to mean that E_1 can shorten the remaining distance more than E_2 when $E_1 \leq_{\text{FRT-Chord}} E_2$.

5.3. Entry Filtering Procedure

The entry filtering procedure of FRT-Chord is defined as follows when the order is defined as above.

Procedure 5.3.1: Entry Filtering in FRT-Chord

[Step 1] `LET` C `=` E.

[Step 2] `LET` C `=` $C \setminus \{e_0, e_1\}$.

[Step 3] `REMOVE` $\operatorname{argmin}_{c \in C} (E^{-c}, \leq_{\text{FRT-Chord}})$ `FROM` E.

*C is a temporary variable.

This is just an entry filtering procedure to which $\leq_{\text{FRT-Chord}}$ and the sticky entry function FRT-Chord::StickyEntry(E) are applied.

When [Step 3] is implemented in this entry filtering procedure, the definition must be converted to some concrete algorithm. Note that naive algorithms could be adopted for this task, such as calculating all worst-case reduction ratios for every E^c and sorting them. However, FRT-Chord adopts a more effective algorithm.

5.4. Entry Filtering Procedure for Implementation

The **canonical spacing** S_i^E of entry e_i in routing table $E = \{e_i\}_{i=0,\ldots,|E|-1}$ is defined as follows:

Definition 5.4.1: Canonical Spacing

$$S_i^E = S^E(e_i) = \log \frac{\mathrm{d}(s, e_{i+1})}{\mathrm{d}(s, e_i)}, \ (i = 0, \ldots, |E| - 1, e_{|E|} = s). \tag{5.7}$$

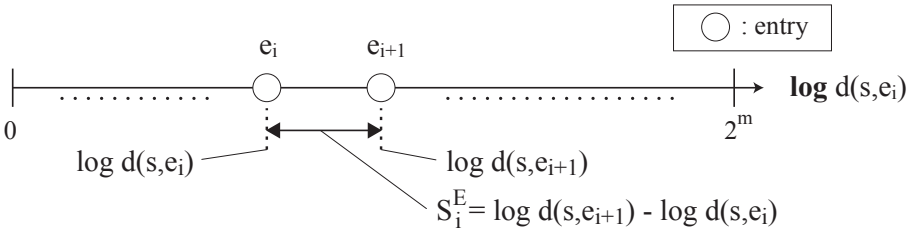


Figure 5.3: Canonical spacing S_i^E.

This is the spacing between the entry and its succeeding entry on the logarithmic axis.

This canonical spacing satisfies the following equation:

Lemma 5.4.1: Canonical Spacings

$$\mathrm{r}_i^E = 1 - 2^{-S_i^E}. \tag{5.8}$$

Since $1 - 2^{-x}$ is a monotonic function, Lemma 5.4.2 holds.

Lemma 5.4.2: Routing Table Order Based on Canonical Spacings

Letting $\left(S^E_{(i)}\right)_{i=0,\ldots,|E|-1}$ be the sequence arranged in descending order of $\left\{S^E_i\right\}_{i=0,\ldots,|E|-1}$,

$$E_1 \leq_{\text{FRT-Chord}} E_2 \Leftrightarrow \left(S^{E_1}_{(i)}\right)_{i=0,\ldots,|E_1|-1} \leq_{\text{dictionary}} \left(S^{E_2}_{(i)}\right)_{i=0,\ldots,|E_2|-1} \quad (5.9)$$

The routing table order defined with the reduction ratio is represented by a formula with canonical spacing. Here, I introduce a **new canonical spacing** $\check{S}^E_i$, which is defined as follows.

Definition 5.4.2: New Canonical Spacing

$$\check{S}^E_i (= \check{S}^E(e_i)) = S^E_{i-1} + S^E_i, \ (i = 1, \ldots, |E| - 1) \quad (5.10)$$

$\check{S}^E_i$ represents the size of a spacing that is newly generated around e_i when e_i is removed from E. When e_i is removed from E, the sequence of canonical spacings $\left(S^E_i\right)_{i=1,\ldots,|E|-1}$ changes to $\left(S^{E^{-e_i}}_i\right)_{i=1,\ldots,|E^{-e_i}|-1}$ by adding a new canonical spacing $\check{S}^E_i$ and removing two spacings, i.e., S^E_{i-1} and S^E_i. Since these three spacings satisfy $\check{S}^E_i > S^E_{i-1}$ and $\check{S}^E_i > S^E_i$, the new canonical spacing $\check{S}^E_i$ is a dominant value relative to lexicographical order. Thus, Lemma 5.4.3 holds.

Lemma 5.4.3: Routing Table Order Based on New Canonical Spacings

$$E^{-e_i} \leq_{\text{FRT-Chord}} E^{-e_j} \Leftrightarrow \check{S}^E_i < \check{S}^E_j \quad (5.11)$$

This means that, after removing different entries from the same routing table, routing tables can be compared using the values of $\check{S}^E_i$ for a current routing table without reference to lexicographical order. Thus, [Step 3] in

the entry filtering procedure can be defined using the following equation:

Theorem 5.4.1

$$\operatorname*{argmin}_{c \in C}(E^{-c}, \leq_{\text{FRT-Chord}}) = \operatorname*{argmax}_{c \in C} \check{S}^E(c) \tag{5.12}$$

Using this theorem, the minimum value in [Step 3] can be easily searched by maintaining the sorted sequence $(\check{S}^E(e_i))_{i=1,\ldots,|E|-1}$, and entry filtering corresponds to the following procedure:

Procedure 5.4.1: Entry Filtering in FRT-Chord for Implementation

[Step 1] `LET` C = E.

[Step 2] `LET` $C = C \setminus \{e_0, e_1\}$.

[Step 3] `REMOVE` $\operatorname*{argmax}_{c \in C} \check{S}^E(c)$ `FROM` E.

When an entry is added or removed, the sorted sequence $(\check{S}^E(e_i))_{i=1,\ldots,|E|-1}$ is updated by adding, removing, or modifying at most $O(1)$ new canonical spacings.

5.4.1. Analysis of Entry Filtering

A routing table E is improved by the entry filtering procedure following the entry learning procedure. The sequence of these procedures is referred to as a **routing table improvement process**, and E.Improve(n) is referred to as the improved routing table by learning an entry n. Here, such a process becomes stuck at a local minimum. A **convergent routing table** is defined as follows:

Definition 5.4.3: Convergent Routing Table

$$E \text{ is a } \textbf{convergent routing table} \overset{\text{def}}{\Leftrightarrow} \forall n \in N, E.\text{Improve}(n) = E, \tag{5.13}$$

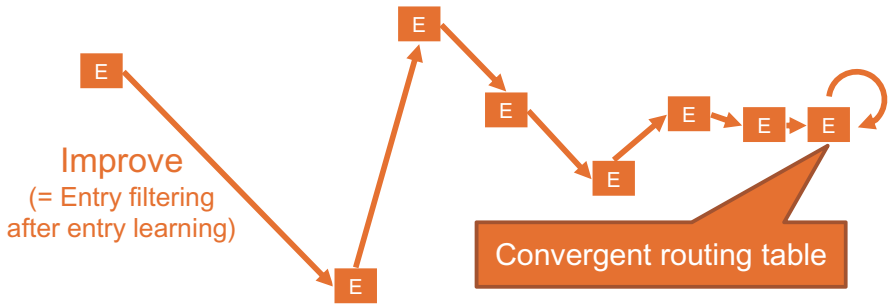


Figure 5.4: Convergent routing table.

To avoid such a local minimum, algorithms should generally adopt appropriate techniques such as simulated annealing [22]. However, Theorem 5.4.2 about the local minimum holds:

Theorem 5.4.2: $O(\log |N|)$ Path Length

With high probability, assuming that all nodes have the convergent routing table with $O(\log |N|)$ entries in an N-node network, **path lengths are $\boldsymbol{O(\log |N|)}$**.

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 ($i = 1, \ldots, |E| - 1, e_{|E|} = s$). Due to 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.

$$S_j^E \leq S_{i-1}^E + S_i^E, \quad (j \in J, i = 2, \ldots, |E| - 1) \tag{5.14}$$

Thus, by aggregating Equation (5.14),

$$\begin{aligned} S_j^E &\leq \frac{\Sigma_{i=2}^{|E|-1}(S_{i-1}^E + S_i^E)}{|E| - 2} \\ &\leq \frac{2\Sigma_{i=1}^{|E|-1}(S_i^E)}{|E| - 2} = \log\left(\frac{\mathrm{d}(s, e_{|E|})}{\mathrm{d}(s, e_1)}\right)^{\frac{2}{|E|-2}}. \end{aligned} \tag{5.15}$$

With high probability, the distance between two generic consecutive nodes is at least $2^m/N^2$[6], namely

$$\mathrm{d}(s, e_1) > \frac{2^m}{N^2}. \tag{5.16}$$

According to the definitions of r_j^E and S_j^E,

$$\mathrm{r}_j^E < 1 - \left(\frac{1}{N}\right)^{\frac{4}{|E|-2}}. \tag{5.17}$$

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

For $|E| = 2 + 4\log N$, the path length needed to reduce the remaining distance to $2^m/N$ or less is at most

$$\log_{\mathrm{r}_i^E} \frac{1}{N} < \log N. \tag{5.18}$$

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

> Thus the query reaches the target logical position t within another $O(\log N)$ steps, meaning that the entire path length is $O(\log N)$. □

Therefore, even if the routing table improvement process stagnates at any local minimum, the path length is sufficiently short and is equivalent to the path length in Chord.

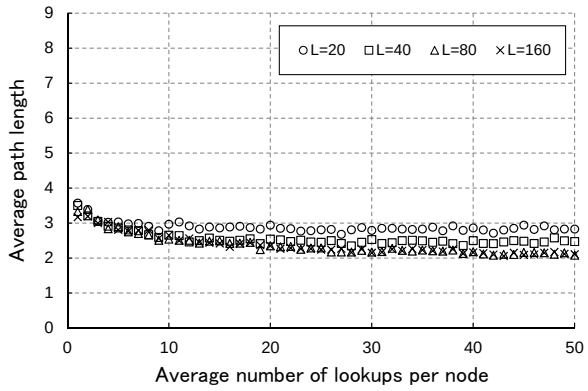
5.5. Experimental Results

I implemented FRT-Chord on Overlay Weaver [45, 46] and performed experiments[34].

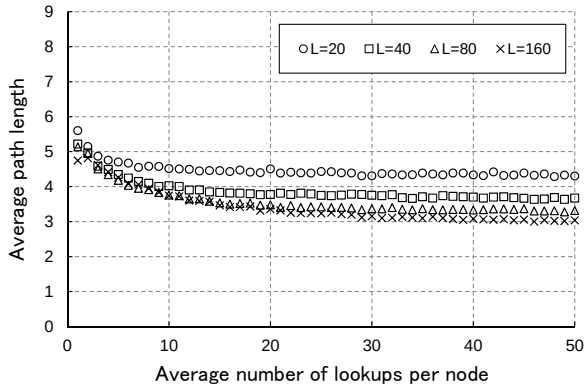
5.5.1. Routing Table Improvement

After $|N|$ nodes join an FRT-Chord system, I repeat sending a query $50|N|$ 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. I vary the number of nodes $|N|$ and the routing table size L and I set the length of the successor list as 4.

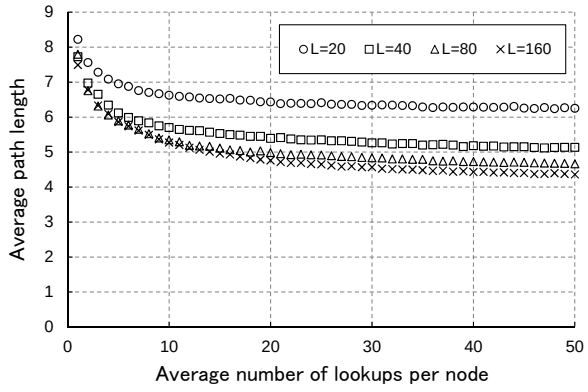
Figure 5.5 indicates the average path lengths for every $|N|$ queries. This figure shows that repeating lookups shortens the average path lengths. The number of lookups shortens the path lengths at almost the same range for every node, regardless of the number of nodes in the system. This means that FRT-Chords can adapt the size of the network.



(a) $N = 10^2$



(b) $N = 10^3$

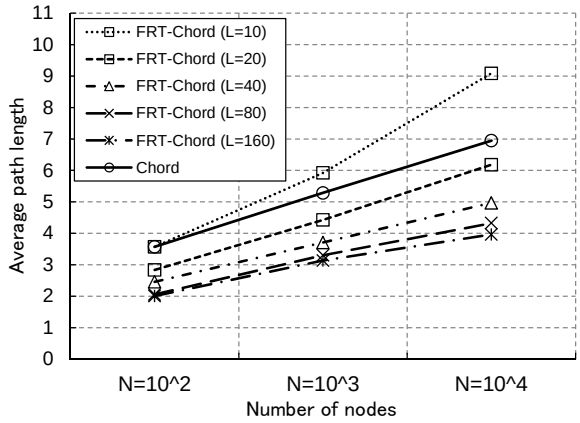


(c) $N = 10^4$

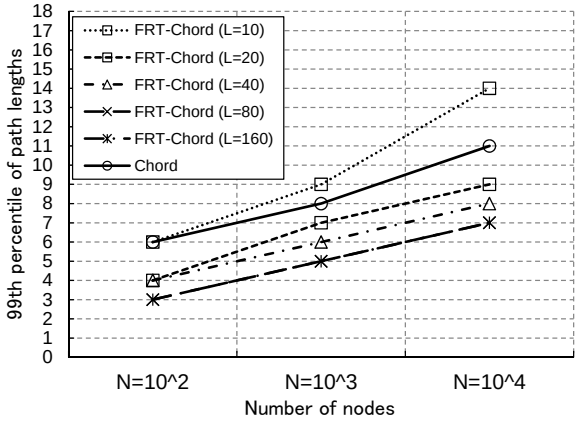
Figure 5.5: Change in average path length with the number of queries per node.

5.5.2. Path length

I varied $|N|$ and the maximum routing table size L, and measured how path lengths change. Figure 5.6 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 5.4.2. These figures also show that the trade-off between L and path lengths can be tuned, and FRT-Chord maintains efficient routing tables with arbitrary routing table size L. When especially $L > N$ such as $N = 10^2, L = 160$, routing tables have all nodes in the system and FRT-Chord achieves $O(1)$-hop lookup performance like existing $O(1)$-hop algorithms[9, 33].



(a) Average path length



(b) 99th percentile of path length

Figure 5.6: Correlation between routing table size and path length.

5.6. FRT-based algorithms based on other logical position space

FRT-Chord is an FRT-based algorithm that adopts the same logical position space and the same distance function as that of Chord. Almost all FRT-based algorithms introduced in the following chapters also adopt the same logical position space. However, this does not mean that FRT can adopt only the logical position space and the distance function of Chord. Note that FRT does not prevent one from designing FRT-based algorithms based on various logical position spaces and various distance functions. I have designed and implemented FRT-Kademlia, which adopts the XOR metric used in Kademlia[30].

5.7. Conclusion

I have proposed FRT-Chord, which is an FRT-based algorithm that uses the same logical position space as Chord to demonstrate that concrete algorithms can be designed based on FRT.

FRT-Chord defines a routing table order according to the worst-case reduction ratios, and FRT-Chord defines entries for the same role as that of the sticky entry in Chord using the sticky entry function of FRT. With such definitions, nodes of FRT-Chord can maintain message reachability and repeatedly improve routing tables by entry learning and entry filtering procedures. I have shown that the routing table improvement process based on the routing table order achieves $O(\log |N|)$ path length with high probability. Experimental results show that FRT-Chord achieves other desirable features derived from FRT such as dynamic and arbitrary routing table size and network size adaptability.

Chapter 6

GFRT-Chord

For real applications, it is important that structured overlays consider other metrics in addition to logical positions, i.e., path length because nodes should construct routing tables with considerations of the physical environment[5, 27, 38, 39, 55].

In Chapter 5, it was shown that a concrete algorithm (i.e., FRT-Chord) that work as intended and achieve short path lengths can be designed based on the FRT framework. However, it is unclear whether FRT-based algorithms can be designed to consider other metrics in addition to logical positions with the same desirable features of FRT such as dynamic and arbitrary routing table size and network size adaptability. Thus, in this chapter, I show the design and implementation of an FRT-based algorithm named GFRT-Chord[35], which considers node groups and shortens inter-group path length.

GFRT-Chord is a structured overlay algorithm that extends FRT-Chord to construct routing tables with consideration of node groups in addition to logical positions. GFRT-Chord achieves reduction of inter-group hops while maintaining short path length derived from FRT-Chord. GFRT-Chord also inherits dynamic and arbitrary routing table size and network size adaptability. This means that FRT-based algorithms can be extended to construct routing tables with consideration of other metrics in addition to logical positions. I prove some desirable properties are made available as a result of assigning appropriate priority to node groups. Experimental results show that the path lengths and the inter-group path lengths are shortened stably with various algorithms and network parameters. This means that GFRT-Chord achieves a balance between logical position considerations and node group considerations.

6.1. Background

Structured overlay algorithms that only consider logical positions aim to minimize path length. Consequently, inefficient forwarding sometimes occurs because nodes forward messages independent of the physical environment. Care must be taken to ensure that routing tables reflect the physical environment for improvements various metrics such as performance and robustness[39].

For example, if such structured overlays are used in data centers, communication via network switches increases, and the load becomes concentrated on the switches. If such structured overlays are used on the Internet, communication across internet service providers (ISP) or countries increases, which leads to network issues and increased cost.

By assuming that nodes on the same server rack or with the same autonomous system (AS) number or country identifier belong to the same node group, the aforementioned issues are considered a problem that can be solved by reducing communication across node groups. Routing methods that reduce communication between node groups can assist various applications. To date, various algorithms[11, 19, 28, 42, 53] that divide all nodes into node groups or node clusters and reduce transmission between them have been proposed. However, these structured overlay algorithms assume a network setup with many nodes; as a result, there are commonly many nodes in each group. Moreover, existing algorithms do not support dynamic and arbitrary routing table size.

Structured overlays are unsuitable for applications that assign a small number of machines, such as a single server rack, to a node group.

Moreover, the best structured overlay algorithms must be able to use small routing tables for large networks. Such algorithms are unsuitable for applications that can increase the number of entries in a routing table; for example, in HPC clusters such as OneHop[9], D1HT[33], or 1h-Calot[48].

I propose **GFRT-Chord**[35], a structured overlay algorithm that makes routing more efficient and reduces the need for communication across groups. GFRT-Chord demonstrates desirable features derived from FRT such as dynamic and arbitrary routing tables, network size adaptability, and effective utilization of entry information.

6.2. Node Groups

GFRT-Chord performs greedy routing with the same logical position space and the same remaining distance $\mathrm{d}(n, t)$ as that of Chord. In GFRT-Chrod, each node n has a node group identifier n.group in addition to a logical position n.lp and an underlay address n.ua. For example, node group identifier represents the AS number, server rack number, country code, or data center name. Then, GFRT-Chord regards nodes that have common node group identifiers as belonging to the same group, and constructs routing tables by considering node group identifiers in addition to logical positions.

Let $G_i = \{n_j\}_{j=1,\ldots,|G_i|}$ be each node group, and $G = \{G_i\}_{i=1,\ldots,|G|}$ be a set of all groups, where n_j is a node.

6.3. Routing Table Order

GFRT-Chord adopts the same routing table order $\leq_{\text{FRT-Chord}}$ as FRT-Chord (Definition 5.2.3). GFRT-Chord considers node groups in its own sticky entry function. Thus, GFRT-Chord can be implemented by utilizing an entry filtering implementation of FRT-Chord.

6.4. Sticky Entry Function

Four types of entry subsets, $G(E) = \{g_i\}_{i=0,\ldots,|\mathrm{G}(E)|-1}(i < j \Rightarrow \mathrm{d}(s, g_i) < \mathrm{d}(s, g_j))$, $\overline{\mathrm{G}}(E)$, $\mathrm{Far}(E)$, $\mathrm{Leap}(E)$, and a subset of nodes in the system G_s are defined as follows:

Definition 6.4.1

$$\mathrm{G}(E) \stackrel{\text{def}}{=} \{e \in E \mid e.\text{group} = s.\text{group}\} \tag{6.1}$$

$$\overline{\mathrm{G}}(E) \stackrel{\text{def}}{=} \{e \in E \mid e.\text{group} \neq s.\text{group}\} \tag{6.2}$$

$$\mathrm{Far}(E) \stackrel{\text{def}}{=} \{e \in E \mid \mathrm{d}(s, g_1) \leq \mathrm{d}(s, e)\} \tag{6.3}$$

$$\mathrm{Leap}(E) \stackrel{\text{def}}{=} \mathrm{Far}(E) \cap \overline{\mathrm{G}}(E) \tag{6.4}$$

$$G_s = \mathrm{G}_s(N) \stackrel{\text{def}}{=} \{n \in N \mid n.\text{group} = s.\text{group}\} \tag{6.5}$$

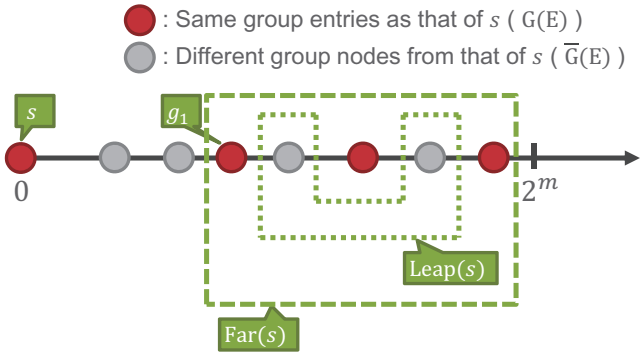


Figure 6.1: $G(E)$, $\overline{\mathrm{G}}(E)$, $\mathrm{Far}(E)$ and $\mathrm{Leap}(E)$ in a routing table of a node s.

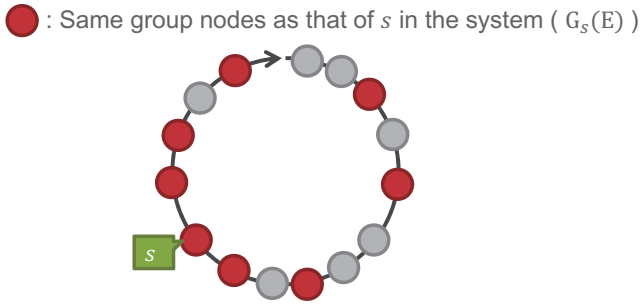


Figure 6.2: G_s: The same group nodes as that of a node s in the system.

Thus, the sticky entry function of GFRT-Chord is defined as follows:

Definition 6.4.2: Sticky Entry Function of GFRT-Chord

$$\text{GFRT-Chord::StickyEntry}(E) \stackrel{\text{def}}{=} \begin{cases} \{e_0, e_1, g_1\} & (\text{Leap}(E) = \emptyset) \\ \{e_0, e_1, g_1\} \cup \text{G}(E) & (\text{Leap}(E) \neq \emptyset). \end{cases} \tag{6.6}$$

6.5. Entry Learning Procedure

The entry learning procedure in GFRT-Chord is the same as that in FRT-Chord.

6.6. Entry Filtering Procedure

According to the entry filtering procedure of FRT-Chord, the entry filtering procedure of GFRT-Chord is summarized as follows:

Procedure 6.6.1: Entry Filtering of GFRT-Chord for Implementation

[Step 1] `LET` C = E.

[Step 2] `LET` C = $C \setminus$ GFRT-Chord::StickyEntry(E).

[Step 3] `REMOVE` $\underset{c \in C}{\text{argmax}}\, \check{S}^E(c)$ `FROM` E.

The routing table order of GFRT-Chord is the same as that of FRT-Chord. Thus, the implementation of [Step 3] in FRT-Chord can be reused in GFRT-Chord. This is achieved by the structure of FRT. This means that separating consideration of routing table order and other factors in FRT facilitates the modularity of algorithms.

The difference between GFRT-Chord and FRT-Chord is in a small part of these sticky entry function. This compact definition of extension is a characteristic of FRT-based algorithms. Such definitions are achieved by two characteristics of FRT, unrestricted routing table candidates and the ability to reflect logical positions to routing tables in the routing table order. Note that FRT does not prevent addition of entries that the new algorithms want to possess. Thus, to shorten path lengths, FRT automatically considers the combination of logical positions in the current routing table that includes such entries.

6.7. Reachability Guarantee Procedures

GFRT-Chord performs the same reachability guarantee procedures as Chord as well as two additional procedures. The **group join** and **group stabilize** procedures build and maintain a **group successor** g_1 and ensure message reachability within each node group, respectively. The basic behaviors of the group join and stabilize procedures are the same as FRT-Chord and Chord. The only difference is that the two group procedures are executed using only nodes that belong to a single node group. Thus, a join procedure (including group join procedures) and a stabilize procedure (including group stabilize procedures) are defined as followings:

Procedure 6.7.1: Join Procedure in GFRT-Chord

[Step 1] Perform the join procedure in FRT-Chord.

[Step 2] Perform the join procedure in FRT-Chord limited to the same group nodes (group join procedure).

Procedure 6.7.2: Stabilize Procedure in GFRT-Chord

[Step 1] Perform the stabilize procedure in FRT-Chord.

[Step 2] Perform the stabilize procedure in FRT-Chord limited to the same group nodes (group stabilize procedure).

The join procedure in GFRT-Chord obtains two entries to guarantee reachability, and the stabilize procedure periodically updates these entries.

6.8. Analysis

Here, I analyze the performance of GFRT-Chord.

6.8.1. Group Localized Routing Table

I analyze routing paths with the concept of **group localized routing table** defined as follows:

Definition 6.8.1: Group Localized Routing Table

Where E is the routing table of node s, a **group localized routing table** is defined as follows:

$$E \text{ is a } \textbf{group localized routing table}$$
$$\overset{\text{def}}{\Leftrightarrow} \forall t \in I, \left(\begin{array}{ll} & \mathrm{fwd}_s^E(t) \in \mathrm{G}_s(N) \\ \vee & \forall g \in \mathrm{G}_s(N), \mathrm{d}(\mathrm{fwd}_s^E(t), t) \leq \mathrm{d}(g, t). \end{array} \right)$$

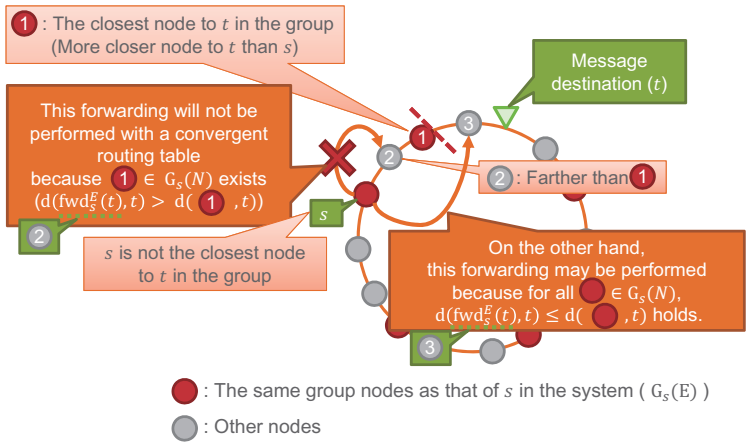


Figure 6.3: Message forwarding at a node s for the destination t with a group localized routing table when $\mathrm{fwd}_s^E(t) \notin \mathrm{G}_s(N)$.

Lemma 6.8.1 about networks with group localized routing tables holds.

Lemma 6.8.1

For any path $V = \{v_i\}$ from an initiator node v_1 to a destination node v_n, when all the nodes in the path have a group localized routing table,

$$1 \leq \forall i < \forall j < \forall k \leq |V|,$$
$$(v_i.\text{group} = v_k.\text{group}) \Rightarrow (v_i.\text{group} = v_j.\text{group}). \quad (6.7)$$

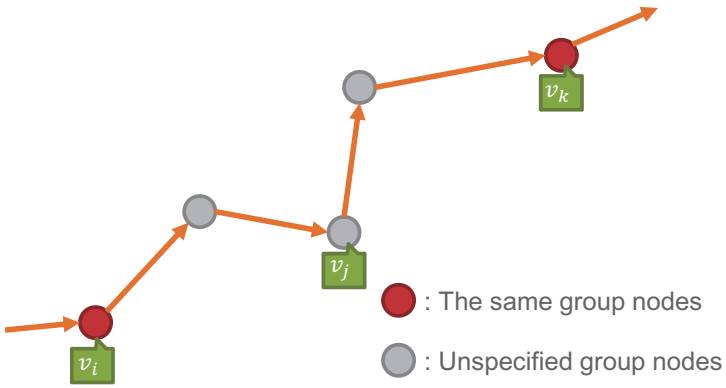


Figure 6.4: A node v_j belongs to the same group because of Lemma 6.8.1.

Proof. Assume that $v_i.\text{group} = v_k.\text{group}$ and that there is a node v_j in the path s.t. $v_i.\text{group} = v_{j-1}.\text{group} \neq v_j.\text{group}$. v_{j-1} has a group localized routing table E_{j-1}; therefore,

$$\forall g \in \mathrm{G}_{v_{j-1}}, \mathrm{d}(\mathrm{fwd}_s^{E_{j-1}}(t), t) \leq \mathrm{d}(g, t) \quad (6.8)$$

,where t is the target logical position of the message. Thus, $v_k \in \mathrm{G}_{v_{j-1}}$ and v_{j-1} forwards messages to v_j; therefore, $\mathrm{d}(v_j, t) \leq \mathrm{d}(v_k, t)$. However, this contradicts the property of greedy routing; i.e., $\mathrm{d}(v_j, t) > \mathrm{d}(v_k, t)$. □

Theorem 6.8.1: Message Don't Come Back to Group

In a GFRT-Chord network wherein nodes have group localized routing tables, after a message leaves a node group G_{passed}, it will never be forwarded to any other node that belongs to G_{passed}.

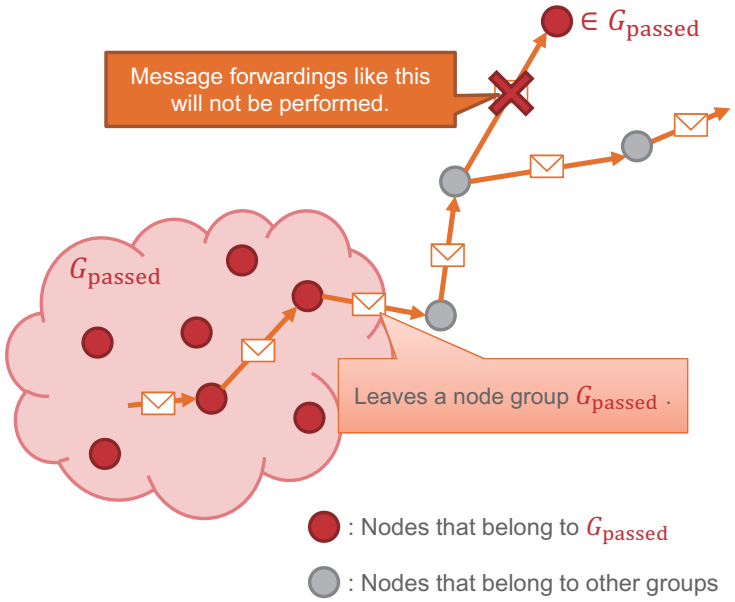


Figure 6.5: A message will not come back to the group G_{passed}

Proof. If a message returns to node group G_{passed} after having passed through it; i.e., if there is a path $V = \{\dots, v_i, v_{i+1}, \dots, v_k, \dots\}$, where $v_i, v_k \in G_{\text{passed}}$ and $v_{i+1} \in G_{\text{another}}$, then behavior contradicts Lemma 6.8.1. □

Let $V_{\text{G}} = \{G_i\}$ be a **group path**, which is a path in units of groups rather than nodes, and the **group path length** is $|V_{\text{G}}| - 1$.

GFRT-Chord has the following group path length features.

Lemma 6.8.2: Group Path Length

In a network that comprises $|G|$ node groups and all nodes have group localized routing tables, the group path length is at most $|G| - 1$.

> **Proof.** From Theorem 6.8.1, the group path length cannot be greater than $|G| - 1$. □

Lemma 6.8.3: The Number of Group Hops

In a network that comprises $|G|$ groups and all nodes have group localized routing tables, the group path length is $O(|G|)$ independent of the number of nodes $|N|$.

> **Proof.** This can be clearly derived from Lemma 6.8.2. □

Therefore, if the number of groups is constant, the group path length is $O(1)$, even when the number of nodes $|N|$ increases.

I will explain stronger features of group path length.

6.8.2. Group Localized Routing Table in GFRT-Chord

Lemmas 6.8.4 and 6.8.5 describe the conditions by which routing tables are group localized routings.

Lemma 6.8.4: $G_s(N) \subset E$

If E is a routing table of node s, E is a group localized routing table if $G_s(N) \subset E$.

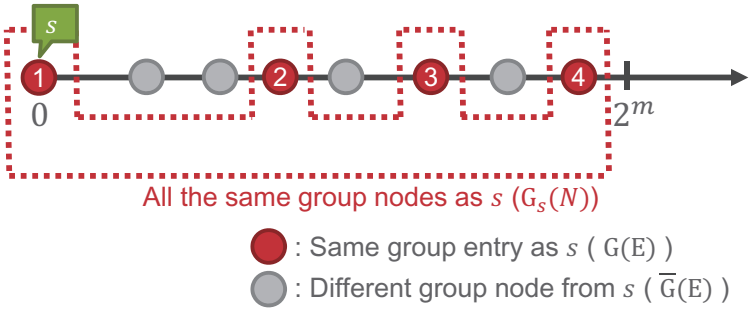


Figure 6.6: A routing table when $G_s(N) \subset E$ and $G_s(N) = 4$.

Proof. GFRT-Chord performs greedy routing; therefore,

$$\forall t \in I, \forall e \in E, \mathrm{d}(\mathrm{fwd}_s^E(t), t) \leq \mathrm{d}(e, t). \tag{6.9}$$

Because $G_s(N) \subset E$,

$$\forall t \in I, \forall e \in G_s(N), \mathrm{d}(\mathrm{fwd}_s^E(t), t) \leq \mathrm{d}(e, t). \tag{6.10}$$

□

Lemma 6.8.5: $\mathrm{Leap}(E) = \emptyset$

If E is a routing table of a node s, E is a group localized routing table if $\mathrm{Leap}(E) = \emptyset$.

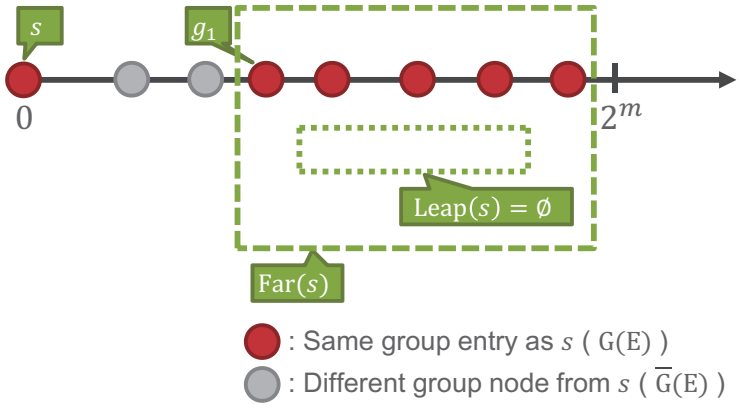


Figure 6.7: A routing table when $\mathrm{Leap}(E) = \emptyset$.

Proof. Let s be a node that belongs to node group G_s and $\mathrm{Dom_G}(s)$ be a responsible domain of s in G_s, i.e., $\mathrm{Dom_G}(s) = \{t \in I \mid \forall g \in G_s, \mathrm{d}(s,t) \leq \mathrm{d}(g,t)\}$.

1. If $t \notin \mathrm{Dom_G}(s)$,
   $\forall t, \mathrm{fwd}_s E(t) \in G_s(N)$ because $\mathrm{Leap}(E) = \emptyset$.

2. If $t \in \mathrm{Dom_G}(s)$,
   $\forall g \in G_s, \mathrm{d}(s,t) < \mathrm{d}(g,t)$ because $t \in \mathrm{Dom_G}(s)$. In this case, $\mathrm{d}(\mathrm{fwd}_s^E(t), t) < \mathrm{d}(s,t)$ because GFRT-Chord performs greedy routing, so $\forall g \in G_s(N), \mathrm{d}(\mathrm{fwd}_s^E(t), t) < \mathrm{d}(g,t)$.

□

Theorem 6.8.2 about convergent routing tables in GFRT-Chord holds.

Theorem 6.8.2: Convergent Routing Table in GFRT-Chord

A convergent routing table E_{conv} is a group localized routing table.

Proof.

1. If $G_s(N) \subset E$, due to Lemma 6.8.5, E is a group-localized routing table.
2. If $G_s(N) \not\subset E$, $\forall g \in G_s(N), g \notin E \Rightarrow E.\text{Improve}(g) = E$ because E is convergent. This means that g is removed from $E.\text{Learn}(g)$ and g is not a sticky entry, so $\text{Leap}(E) = \emptyset$. Therefore, due to Lemma 6.8.5, E is a group-localized routing table.

□

This means that a routing table becomes a group localized routing table after a sufficient number of routing table improvement processes.

6.8.3. Path Length in Two Growth Model

Here, I analyze path lengths for the GFRT-Chord algorithm in the system with two different network-growth models: group-increasing model and member-increasing model (Figure 6.8).

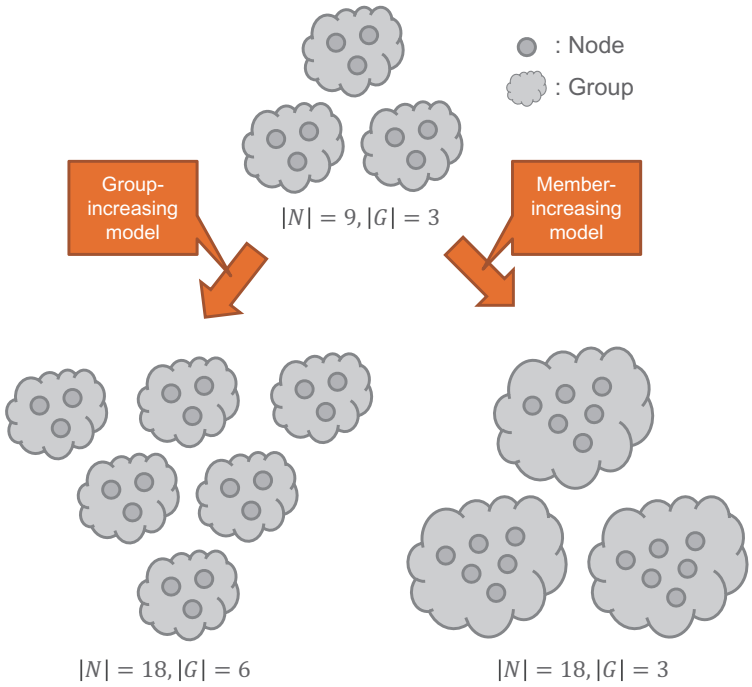


Figure 6.8: Two network-growth models.

Theorem 6.8.3 governs the selection of nodes to be removed in the entry filtering procedure.

Theorem 6.8.3

Let $E = \{e_i\}$ be a routing table and $E^* = E^{-e_{i^*}}$ be a routing table from which e_{i^*} has been removed as part of the entry filtering, where e_i is not a sticky entry for E. Thus, the following is obtained:

$$E^{-e_{i^*}} \leq_{\text{FRT-Chord}} E^{-e_i}. \tag{6.11}$$

Definition 6.8.2: $O(\log |N|)$-hop

$O(\log |N|)$-hop is a class of hops performed at most $O(\log |N|)$ times prior to arriving at the node that is responsible for a message.

For example, when an algorithm performs only $O(\log |N|)$-hops, its path length is $O(\log |N|)$.

Definition 6.8.3: Degree of Freedom

Degree of freedom of a convergent routing table E is the minimum number of non-sticky entries in E.

Theorem 6.8.4: FRT-Chord and Degree of Freedom

In an FRT-based Chord algorithm (including GFRT-Chord) network with $|N|$ nodes, a hop to e_i is an $O(\log |N|)$-hop with high probability when a node e_{add} exists in (e_i, e_{i+1}) such that the degree of freedom of $E^{+e_{\text{add}}}$ is greater than $4 \log |N|$.

Proof. E is convergent; therefore, for all entries $e_{\text{free}} \in E$ that are not sticky entries of $E^{+e_{\text{add}}}$:

$$E^{+e_{\text{add}}-e_{\text{add}}} \leq_{\text{FRTChord}} E^{+e_{\text{add}}-e_{\text{free}}}. \tag{6.12}$$

Let e_j be e_{free}; thus, $S_i \leq S_{j-1} + S_j$. By adding both sides for all j, I obtain the following:

$$S_i = \frac{\Sigma_{j \in J} S_i}{|J|} \leq \frac{2}{|J|} \Sigma_{j \in E^{-s}} S_j = \log \left(\frac{2^m}{\mathrm{d}(s, e_1)} \right)^{\frac{2}{|J|}}. \tag{6.13}$$

Since any distance $\mathrm{d}(s, e_1) > 2^m / |N|^2$, the following equation holds with high probability when $|N|$ is sufficiently large:

$$\mathrm{r}_i(E) \leq 1 - (1/|N|)^{\frac{4}{|J|}}. \tag{6.14}$$

Thus, since $|J|$ equals the degree of freedom of $E^{+e_{\text{add}}}$, $|J|$ is $4\log|N|$. Therefore, the number of times such forwarding is performed by the time the remaining distance becomes smaller than $2^m/|N|$ is expressed as follows:

$$\log_{\mathrm{r}_i(E)}(1/|N|) \leq \log|N|. \tag{6.15}$$

Therefore, the hop is $O(\log|N|)$-hop. □

Here, I focus on two network-growth models. Various applications can be classified into one of these two models (Figure 6.8).

The first model is a **group-increasing model**. In such networks, the number of groups increases but the number of nodes in each group does not increase significantly. For example, groups represent network switches, network routers, or server racks. In such a network, there is a constant C, which is not less than the number of nodes belonging to each group $|G_i|$.

Theorem 6.8.5: Path length in Group-Increasing Model

In a GFRT-Chord network that comprises $|G|$ groups with a capacity of C nodes, the path length is $O(\log|N|)$ with high probability when all nodes have convergent routing tables with $O(\log|N|)$ entries.

Proof. I prove that a hop from s to e_i is an $O(\log|N|)$-hop. Let $|E|$ be $|G_s| + 4\log|N|$; thus, the following holds.

1. When e_i.group $\neq G_s$, the hop is from one group to another.
   (a) When $\text{Leap}(E) = \emptyset$, e_i is in (s, g_i). Thus, there are no nodes that belong to G_s in (e_i, e_{i+1}).
      i. When there are no nodes in (e_i, e_{i+1}), this hop is the last hop.
      ii. When there is a node $e_{\text{add}}(\notin G_s)$, the degree of freedom of $E^{+e_{\text{add}}}$ is larger than $4\log|N|$.
   (b) When $\text{Leap}(E) \neq \emptyset$, all nodes belong to G_s in E. Thus, there are no nodes that belong to G_s in (e_i, e_{i+1}).

i. When there are no nodes in (e_i, e_{i+1}), this hop is the last hop.

ii. When there is a node $e_{\text{add}} (\notin G_s)$, the degree of freedom of $E^{+e_{\text{add}}}$ is $4 \log |N|$.

Therefore, given Theorem 6.8.4, hops over groups are performed $O(\log |N|)$ times.

2. When $e_i.\text{group} = G_s$, such hops are performed at most $|G_i|$ times in each group G_i on a routing path. Hops over groups are performed $O(\log |N|)$ times; thus, messages are routed through $O(\log |N|)$ groups. As a result, hops between nodes in the same group are performed at most $O(C \log |N|) = O(\log |N|)$ times.

Therefore, the number of hops is $O(\log |N|)$. □

The other network-growth model is a **member-increasing model**. In such networks, the number of nodes in each group increases but the number of groups does not increase significantly. For example, groups can represent ISPs, countries, or continents. In such a network, there is a constant $G_{\max}$ which is the number of groups $|G|$ is not more than.

Theorem 6.8.6: Path length in Member-Increasing Model

In a GFRT-Chord network that comprises $|G|$ node groups, where $|G|$ is less than $G_{\max}$, the path length is $O(\log |N|)$ with high probability when all nodes have convergent routing tables with $O(\log |N|)$ entries.

Proof. I prove that a hop from s to e_i is $O(\log N)$-hop.

Let the maximum routing table size be $2 + 4 \log |N|$ because $|G|$ is less than $G_{\max}$; thus, $|N|$ is sufficiently large. Therefore, $|G_s|$, i.e., the number of nodes belonging to a group that is the same as s, is larger than $|E|$. Thus, there are nodes that belong to G_s but not in E. Therefore, because E is a convergent routing table, $\text{Leap}(E) = \emptyset$.

1. When $e_i.\text{group} \neq s.\text{group}$, the hop is from one group to

another. Given Theorem 6.8.1, such a hop is performed at most $|G| - 1$ times.

2. When e_i.group $= s$.group, the hop is an intra-group hop. Thus, because $\mathrm{Leap}(E) = \emptyset$, e_i is the same node or farther than g_1.

   (a) When there are no nodes in (e_i, e_{i+1}), the hop is the last hop.

   (b) When there are one or more nodes that do not belong to G_s in (e_i, e_{i+1}), the hop is the last hop. Thus, this hop is performed only $|G|$ times.

   (c) When there are one or more nodes e_{add} that belong to G_s in (e_i, e_{i+1}), $\mathrm{Leap}(E^{+e_{\mathrm{add}}})$ remains empty. Therefore, the degree of freedom of $E^{+e_{\mathrm{add}}}$ is $4 \log |N|$.

Given Theorem 6.8.4, all hops are $O(\log N)$-hop; thus, the path length is $O(\log N)$. □

6.9. Naive Grouped FRT-Chord

GFRT-Chord efficiently builds group localized routing tables and acquires the outstanding characteristics derived from the routing tables.

I have defined GFRT-Chord simply; however, I can also define naive grouped FRT-Chord, which is an algorithm by considering node groups more simply and intuitively as follows:

Definition 6.9.1: Sticky Entry Function of Naive Grouped FRT-Chord

$$\text{NaiveGroupedFRT-Chord::StickyEntry}(E) = \{e_0, e_1, g_1\} \cup \mathrm{G}(E) \quad (\mathrm{Leap}(E) \neq \emptyset). \tag{6.16}$$

This algorithm is based on the idea that $\mathrm{G}(E)$ should be protected from eviction in entry filtering procedure. However, this algorithm accords too

much priority to node groups compared with GFRT-Chord. Thus, the naive grouped FRT-Chord does not shorten path lengths and may send messages along a long path.

In GFRT-Chord, the priority of node groups and that of choosing the shortest paths are effectively balanced. I demonstrate this property by implementing naive grouped FRT-Chord and GFRT-Chord (Section 6.10).

6.10. Experimental Analysis

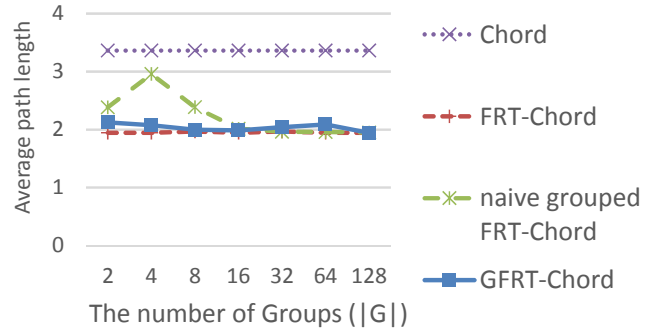
To quantitatively evaluate the properties of GFRT-Chord, I implemented Chord, FRT-Chord, naive grouped FRT-Chord, and GFRT-Chord, and ran simulations.

6.10.1. Path Length and Group Path Length Relative to $|N|$ and $|G|$

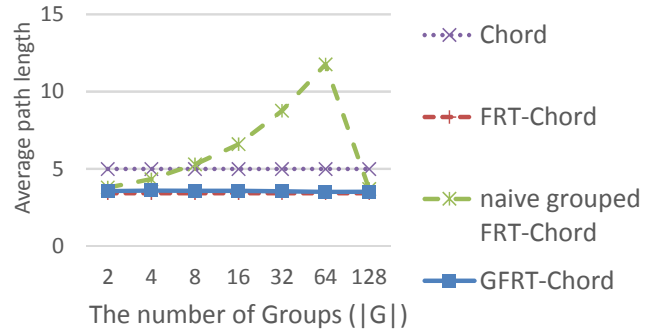
In our simulations, I implemented the proposed algorithms with the number of nodes $|N|$ equaling 120, 1280, and 12800 and the number of node groups $|G|$ equaling $2, 8, 32$, and 128. For FRT-Chord, naive grouped FRT-Chord, and GFRT-Chord, for which the maximum number of entries L can be set, I set $L = 20$. Note that each routing table E includes its owner node s; thus, the number of entries other than s is $L - 1$. Each node looks up 100 logical positions selected randomly to initialize routing tables, and lookups are repeated 10000 times from randomly selected nodes to randomly selected logical positions. To evaluate routing characteristics, I measured the average path length and average group path length (i.e., the number of hops from one group to another).

Figure 6.9 shows that the average path length of GFRT-Chord is not significantly less than FRT-Chord, which does not consider groups. Figure 6.9 also shows that GFRT-Chord reduced the number of inter-group hops efficiently for the various parameters, and that the group path length is kept smaller even when $|G|$ is large (i.e., group path length is proportional to $\log |G|$).

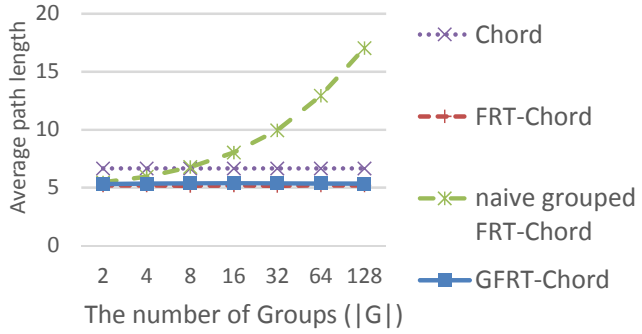
Naive grouped FRT-Chord demonstrates performance that is nearly equivalent to GFRT-Chord if the number of nodes or the number of node groups



(a) $N = 128$



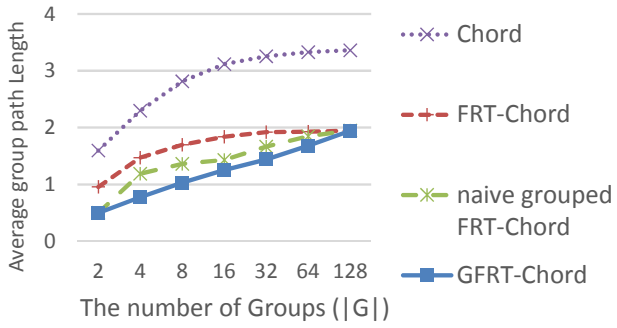
(b) $N = 1280$



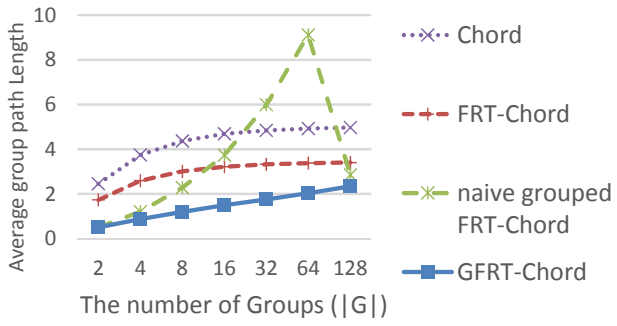
(c) $N = 12800$

Figure 6.9: Average path length

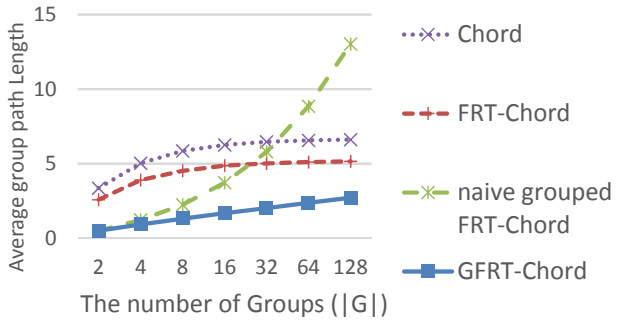
is small. In these cases, the algorithm seems to be as good as GFRT-Chord. However, the path length extends remarkably when $(|N|, |G|) = (1280, 64)$



(a) $N = 128$



(b) $N = 1280$



(c) $N = 12800$

Figure 6.10: Average group path length

or $(|N|, |G|) = (12800, 128)$. In these cases, most entries in the routing tables of s belong to G_s because $|N|/|G| = |G_s| > L$. Thus, inter-group hops

are improved by improve procedures, but intra-group hops are not improved because there are few nodes that belong to other groups. However, this is not significant problem when $|G|$ is small, because intra-group hops are performed more often than inter-group hops. However, when $|G|$ is large, which means that $|G_s|$ is small, routing tables cannot be significantly improved by repeating improve procedures because the number of entry patterns that belong to G_s is also small.

GFRT-Chord solves this issue by accurately restriction of keeping same group nodes in its routing tables by the dividing of cases about Leap(E) in the sticky entry function.

GFRT-Chord performs in the same manner as FRT-Chord in an extreme situation, wherein the number of forwards across groups cannot be reduced; i.e., when $|N| = |G|$ or $|G| = 1$. GFRT-Chord does not excessively extend path lengths, even though it gives priority to the consideration of node groups.

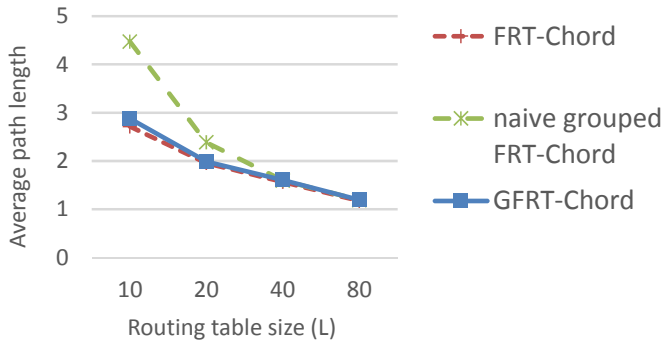
6.10.2. Path Length and Group Path Length Relative to Routing Table Size (L)

GFRT-Chord allows the routing table size (L) to be set freely and dynamically. I evaluated path length in various routing table sizes about this function. We compared GFRT-Chord with FRT-Chord and naive grouped FRT-Chord, which also allow the routing table size to be set.

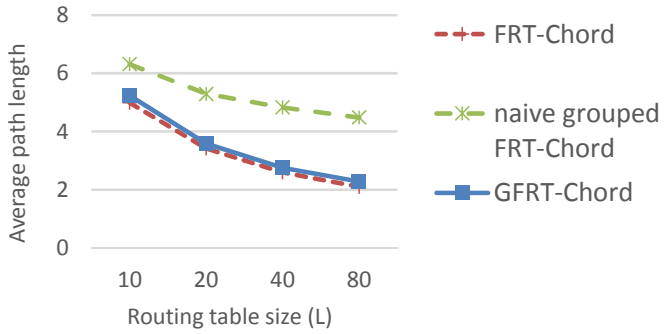
In various situations, GFRT-Chord shortened the group path length by extending the total path length slightly, as is shown in Figure 6.11, Figure 6.12, Figure 6.13 and Figure 6.14.

In contrast, with naive grouped FRT-Chord, when $(|N|, |G|) = (1280, 8)$, $(12800, 32)$, etc., the average group path length is constant and independent of L, and expanding the routing table size provides no improvement because routing tables have too many entries that belong to their groups and thus cannot be improved.

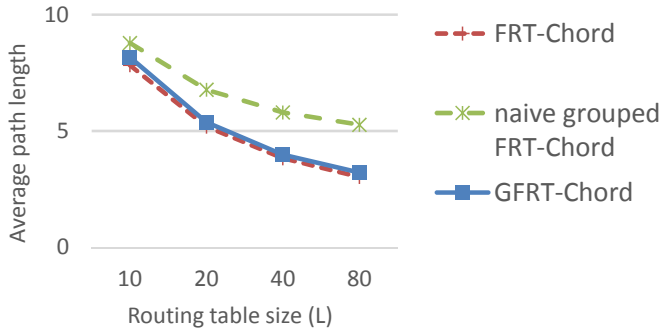
A unique improvement to the group path length of naive grouped FRT-Chord occurs when $(|N|, |G|, L) = (1280, 32, 80)$, as is shown in Figure 6.13 because $|N|/|G| = |G_s| = 40 < L = 20$. Thus, the routing tables include at most 20 entries. The remaining capacities of the routing tables is used to improve inter-group hops.



(a) $N = 128$



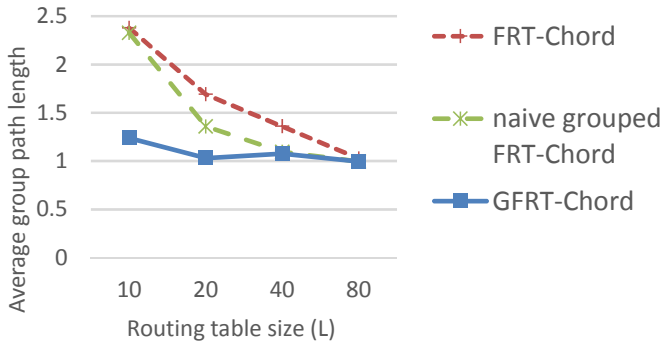
(b) $N = 1280$



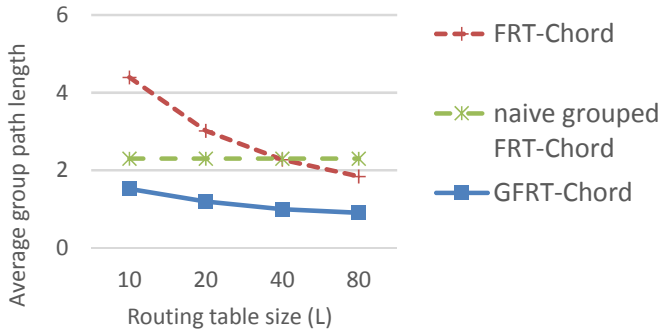
(c) $N = 12800$

Figure 6.11: Average path length ($|G| = 8$)

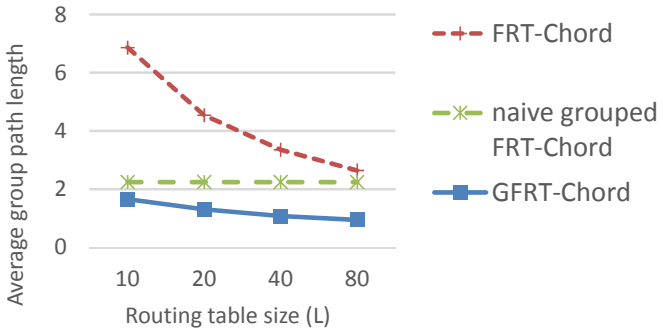
In contrast, even in such situations, GFRT-Chord performs acceptably. GFRT-Chord shortens the average path length and average group path



(a) $N = 128$



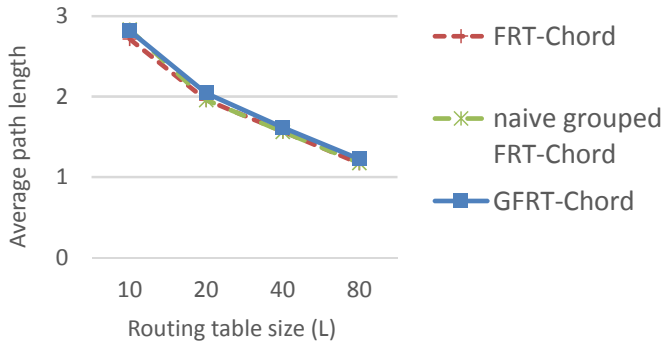
(b) $N = 1280$



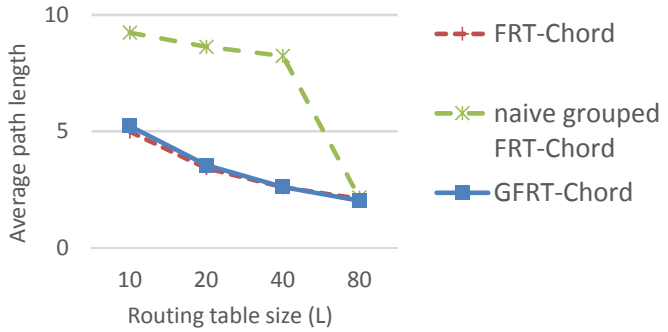
(c) $N = 12800$

Figure 6.12: Average group path length ($|G| = 8$)

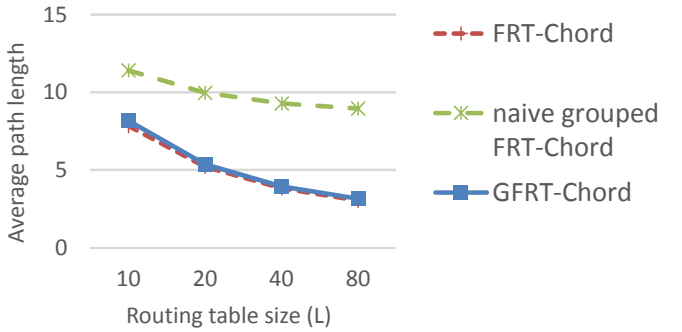
length steadily depending on $(|N|, |G|, L)$. This shows the characteristics derived from FRT-Chord, which can set L freely and dynamically irrespec-



(a) $N = 128$



(b) $N = 1280$

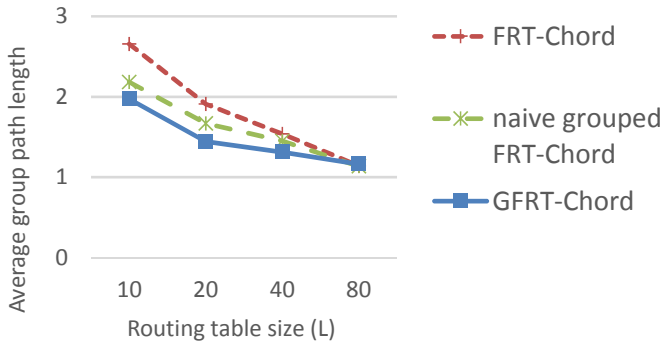


t

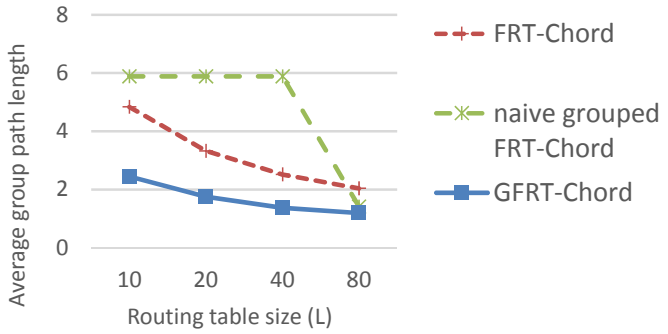
(c) $N = 12800$

Figure 6.13: Average path length ($|G| = 32$)

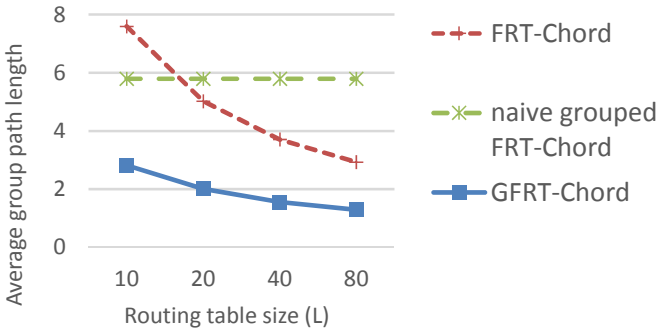
tive of $|N|$ or $|G|$, and whether nodes know $|N|$ or $|G|$.



(a) $N = 128$



(b) $N = 1280$



(c) $N = 12800$

Figure 6.14: Average group path length ($|G| = 32$)

6.11. Conclusion of This Chapter

In this chapter, a concrete algorithm, i.e., GFRT-Chord, has been proposed to demonstrate that FRT-based extended algorithms can be designed to consider metrics other than path length and retain the desirable features of FRT such as dynamic and arbitrary routing table size and network size adaptability.

GFRT-Chord, which is based on FRT-Chord, considers node groups in addition to logical positions and reduces inter-group hops and total path length.

GFRT-Chord restricts the growth of the entire path length regardless of whether the number of nodes in the system or the number of nodes that belong to each node group is extremely small or large. Moreover, GFRT-Chord does not need to know the number of nodes and groups in advance; therefore, it can be used when this information cannot be predicted.

I have demonstrated that path length is $O(\log |N|)$ in two typical network-growth models and shown that, after a message leaves a node group, the message will never be forwarded to any other node that belongs to the group, i.e., a message does not pass the same group two times.

The features of the proposed routing algorithm were verified in simulations, and the proposed method was compared to existing algorithms and a simple algorithm that uses node groups. I have designed another naive grouped FRT-Chord algorithm by extending FRT-Chord that is defined in a more simple and straightforward manner than the proposed GFRT-Chord. I have shown that this algorithm experiences problems when network and algorithm parameters meet certain conditions, and I have demonstrated that GFRT-Chord consistently performs efficient routings in such cases.

Chapter 7

Other FRT-based Algorithms

Here, in addition to GFRT-Chord in Chapter 6, other FRT-based algorithms are introduced to show the applications of FRT for various targets. These algorithms utilize the structure of FRT and achieve the desirable features derived from FRT, such as dynamic and arbitrary routing table size and network size adaptability.

To compare these FRT-based algorithms and emphasize the differences among them, the following elements of FRT are summarized.

- Remaining distance
- Routing table order
- Sticky entry function
- Entry learning procedure
- Entry filtering procedure
- Entry filtering procedure for implementation
- Reachability guarantee procedures

7.1. FRT-2-Chord

FRT-2-Chord[2] is an FRT-based algorithm that is characterized by a symmetric remaining distance. By adopting symmetric distance, FRT-2-Chord achieves routing table symmetry; when the routing table of node n_1 contains node n_2, the routing table of node n_2 tends to contain node n_1. This property reduces the cost of routing table maintenance and improves

the efficiency of learning entries. Symmetric distance is also used in various algorithms, such as 2-Chord[6], Kademlia[30], and CAN[39].

In the following, I explain how to design an FRT-based algorithm with symmetric remaining distance.

7.1.1. Remaining Distance

In FRT-2-Chord, each node is assigned a logical position from 0 to $2^m - 1$ like in Chord. FRT-2-Chord adopts symmetric distance from x to y. Here, $\mathrm{d}(x, y)$ is defined as follows:

Definition 7.1.1

$$\mathrm{d}(x, y) = \min\{|x - y|, 2^m - |x - y|\} \tag{7.1}$$

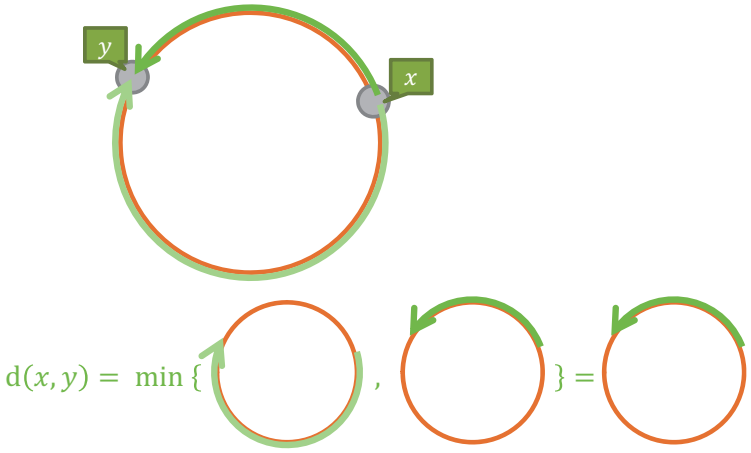


Figure 7.1: Remaining distance function of FRT-2-Chord.

By this definition, $\mathrm{d}(x, y) = \mathrm{d}(y, x)$ holds for any x and y, i.e., distance is symmetric. In FRT-2-Chord, routing tables are constructed according to the total order on the basis of remaining distance. Consequently, routing tables will be symmetric, i.e., when the routing table of node n_1 contains

another node n_2, the routing table of node n_2 tends to contain node n_1.

7.1.2. Routing Table Order

To define the routing table order $\leq_{\text{FRT-2-Chord}}$, FRT-2-Chord defines the worst-case reduction ratio in the same way as Chord.

Let $E = \{e_i\}_{0,\ldots,|E|-1}$ be the routing table of node s aligned clockwise from $s = e_0$, where $\mathrm{d}_{\text{chord}}(x, y)$ is the clockwise remaining distance of Chord. Then, logical position m_i is defined as follows:

Definition 7.1.2

$$m_i = e_i + \mathrm{d}_{\text{chord}}(e_i, e_i + 1) \bmod 2^m. \tag{7.2}$$

FRT-2-Chord performs a greedy routing strategy, and the node responsible node for a target logical position t is the closest node to t based on the remaining distance. In FRT-2-Chord, messages may be forwarded anticlockwise. Assuming clockwise message forwarding to entry e_i, the worst-case reduction ratio $r_i^{E,\text{cw}}$ occurs when the target t equals m_i. Assuming anticlockwise message forwarding to an entry e_i, the worst-case reduction ratio $r_i^{E,\text{acw}}$ occurs when the target t equals m_{i-1}.

In FRT-2-Chord, a set of the worst-case reduction ratios $\{r_i^E\}$ is defined as follows:

Definition 7.1.3

$$\{r_i^E\} = \{r_i^{E,\text{cw}}\} \cup \{r_i^{E,\text{acw}}\} \tag{7.3}$$

Here, routing table order $\leq_{\text{FRT-2-Chord}}$ is defined as follows:

Definition 7.1.4

$$E_1 \leq_{\text{FRT-2-Chord}} E_2 \Leftrightarrow \{r_{(i)}^{E_1}\} \leq_{\text{dic}} \{r_{(i)}^{E_2}\}, \tag{7.4}$$

where $\{r_{(i)}^{E}\}$ is the sequence of reduction ratios arranged in descending order.

7.1.3. Sticky Entry Function

In FRT-2-Chord, the sticky entry function is defined as follows:

Definition 7.1.5

$$\text{FRT-2-Chord::StickyEntry}(E) = \{e_0, e_1, e_{|E|-1}\} \tag{7.5}$$

This function differs from the sticky entry function in FRT-Chord because the predecessor entry $e_{|E|-1}$ is required to guarantee message reachability in the anticlockwise direction.

7.1.4. Entry Learning Procedure

The entry learning procedure of FRT-2-Chord is the same as that of FRT-Chord.

7.1.5. Entry Filtering Procedure

The entry filtering procedure of FRT-2-Chord is summarized as follows:

Procedure 7.1.1: Entry Filtering Procedure in FRT-2-Chord

[Step 1] `LET` $C = E$.

[Step 2] `LET` $C = (C \setminus \text{FRT-2-Chord::StickyEntry}(E))$.

[Step 3] `REMOVE` $\underset{c \in C}{\operatorname{argmin}}(E^{-c}, \leq_{\text{FRT-2-Chord}})$ `FROM` E.

This procedure is based entirely on the FRT framework.

7.1.6. Entry Filtering Procedure for Implementation

For simple implementation, FRT-2-Chord uses another form of definition for its entry filtering procedure.

When e_i is removed from E, the **updated worst-case reduction ratio** R_i^E is defined as follows:

Definition 7.1.6: Updated Worst-Case Reduction Ratio

$$R^E(e_i) = \begin{cases} \dfrac{|\mathrm{d}(s, e_{i+1}) - \mathrm{d}(s, e_{i-1})|}{\mathrm{d}(s, e_{i+1}) + \mathrm{d}(s, e_{i-1})} & (i \neq k, i \neq k+1) \\ \dfrac{2^m - \mathrm{d}(s, e_{i+1}) - \mathrm{d}(s, e_{i-1})}{2^m - |\mathrm{d}(s, e_{i+1}) - \mathrm{d}(s, e_{i-1})|} & (i = k, k+1) \end{cases} \tag{7.6}$$

Here, the following entry filtering procedure is the same as Definition 7.1.1.

By sorting $R^E(e_i)$ in ascending order, the entry to be removed can be found efficiently, and the following theorem holds:

Theorem 7.1.1

$$R^E(e^*) \leq R^E(e) \Leftrightarrow E^{-e^*} \leq_{\text{FRT-2-Chord}} E^{-e} \tag{7.7}$$

Thus, the following entry filtering procedure removes the same entry as Procedure 7.1.1.

Procedure 7.1.2: Entry Filtering in FRT-2-Chord for Implementation

[Step 1] `LET` C = E.

[Step 2] `LET` C = $(C \setminus \text{FRT-2-Chord::StickyEntry}(E))$.

[Step 3] `REMOVE` $\operatorname{argmin}_{c \in C} R^E(c)$ `FROM` E.

By updating the sequence of $\left(R^E(e)\right)_{e \in E}$ sorted in ascending order, the entry to be removed can be found efficiently.

7.1.7. Reachability Guarantee Procedures

Due to the two-direction property, the reachability guarantee procedure has two elements, i.e., a clockwise stabilization, which is the same as that of FRT-Chord, to update the successor entry and anticlockwise stabilization to update the predecessor entry.

7.1.8. Analysis

The entry filtering procedure of FRT-2-Chord also achieves the same $O(\log |N|)$ path length as FRT-Chord.

In a network where all nodes have convergent routing tables, the following Theorem 7.1.2 holds:

Theorem 7.1.2: $O(\log |N|)$ Path Length

With high probability (or under standard hardness assumptions), assuming that all nodes have a convergent routing table with $O(\log |N|)$ entries in an $|N|$-node network, path lengths are $O(\log |N|)$.

7.2. FRT-Chord#

FRT-Chord#[32] is an FRT-based algorithm characterized by routing table order that is based on entries in neighbor routing tables rather than logical position. Given this characteristic, FRT-Chord# supports non-uniform logical position node distributions and enables a DHT to support range queries. Data keys may be biased in a DHT that supports range queries, and FRT-Chord# routing tables are constructed without the assumption that the data keys are distributed uniformly.

7.2.1. Remaining Distance

The remaining distance used in FRT-Chord# is the same as that used in FRT-Chord. FRT-Chord# selects a next hop entry according to this remaining distance. However, FRT-Chord# does not use a logical-position-based remaining distance to define routing table order.

7.2.2. Routing Table Order

Let $E_s = \{e_i^s\}_{i=1,\ldots,|E|}$ be the routing table of node s aligned in a clockwise direction from s (note that $s \notin E$). Then, $E_s(x)$ and $n_s(x)$ are defined as follows:

Definition 7.2.1

$$E_s(x) = \operatorname*{argmin}_{e \in E_s(x)} \mathrm{d}(e, s) \tag{7.8}$$

$$n_s(x) = \max\{i \mid \mathrm{d}(s, e_i^s) \leq \mathrm{d}(s, E_s(x))\}. \tag{7.9}$$

The entry $E_s(x)$ is the entry closest to a logical position x, and the value $n_s(x)$ is the number of entries from s to $E_s(x)$.

FRT-Chord$^{\#}$ defines $f_s(x)$ as follows:

Definition 7.2.2

$$f_s(x) = n_s(x) - n_{E_s(x)}(x). \tag{7.10}$$

$f_s(x)$ denotes the gap of $n_s(x)$ between node s and entry $E_s(x)$.

Here, FRT-Chord$^{\#}$ defines the worst-case reduction value of the number of entries g_i^E forwarded to entry e_i as follows:

Definition 7.2.3

$$g_i^E = f_s(e_{i+1}). \tag{7.11}$$

Let $\{g_{(i)}^E\}$ be the sequence of $\{g_i^E\}$ arranged in descending order. Then, the routing table order of FRT-Chord$^{\#}$ is defined as follows:

Definition 7.2.4

$$E_1 \leq_{\text{FRT-Chord}^{\#}} E_2 \overset{\text{def}}{\Leftrightarrow} \{g_{(i)}(E_2)\} \leq_{\text{dic}} \{g_{(i)}(E_1)\}. \tag{7.12}$$

7.2.3. Sticky Entry Function

The sticky entry function of FRT-Chord# is the same as that of FRT-Chord.

7.2.4. Entry Learning Procedure

The entry learning procedure of FRT-Chord# is the same as that of FRT-Chord.

7.2.5. Entry Filtering Procedure

The entry filtering procedure of FRT-Chord# is summarized as follows:

Procedure 7.2.1: Entry Filtering in FRT-Chord#

[Step 1] `LET` $C = E$.

[Step 2] `LET` $C = (C \setminus \text{FRT-Chord::StickyEntry}(E))$.

[Step 3] `REMOVE` $\underset{c \in C}{\operatorname{argmin}}(E^{-c}, \leq_{\text{FRT-Chord\#}})$ `FROM` E.

7.2.6. Reachability Guarantee Procedures

The reachability guarantee procedures of FRT-Chord# are the same as that of FRT-Chord.

7.2.7. Performance

FRT-Chord# achieves short path lengths even if the logical positions of nodes are defined with non-uniform distributions and maintains the previously mentioned desirable features derived from FRT.

The design of routing table order enables FRT-Chord# features that support supporting range queries. Because FRT-Chord# works with non-uniform logical position distributions, nodes do not require a hash function to determine individual logical positions. In a DHT that supports range queries, the distribution of keys is often non-uniform because the sequence of keys cannot be changed. In FRT-Chord, the logical positions of nodes must be determined with uniform distributions; thus, node loads are seriously imbalanced. However, in FRT-Chord#, each node can determine its own logical position arbitrarily; consequently, the system can balance loads.

7.3. FFRT-Chord

FFRT-Chord[3] is an FRT-based algorithm that demonstrates nodes can construct efficient routing tables solely based on query flows. Each node records query flows, and constructs its own routing table according to the target logical position of the queries. FFRT-Chord does not assume that the logical positions of nodes and query targets are uniformly distributed. This is a practical advantage that supports range queries. In contrast to FRT-Chord#, which also supports range queries, each node does not need to refer to other nodes' routing tables. Thus, FFRT-Chord reduces bandwidth consumption.

In the following subsections, I explain how to design an FRT-based algorithm with query flows.

7.3.1. Remaining Distance

The remaining distance used in FFRT-Chord is the same as that of FRT-Chord. FFRT-Chord selects a next hop entry according to this function based on a greedy routing strategy.

7.3.2. Routing Table Order

Each node in FFRT-Chord maintains a query history for received queries Q with maximum size H. Each record includes the target logical position. FFRT-Chord defines a routing table order $\leq_{\mathrm{FL}}$ based on this query history.

Query flow f_e^E is defined as follows:

Definition 7.3.1

$$f_e^E = |\{q \in Q \mid \mathrm{d}(e, q) < \mathrm{d}(e', q), \forall e' \in E^{-e}\}|, \tag{7.13}$$

where q is a target logical position of the query. Routing table order $\leq_{\mathrm{FL}}$ is defined as follows:

Definition 7.3.2

$$E_1 \leq_{\mathrm{FL}} E_2 \overset{\mathrm{def}}{\Leftrightarrow} \mathrm{V}(F_{E_1}) \leq \mathrm{V}(F_{E_2}), \tag{7.14}$$

where $\mathrm{V}(F_{E_1})$ is the variance of query flows and is defined as follows:

Definition 7.3.3

$$\mathrm{V}(F_{E_1}) = \sum_{e \in E} (f_e^E - \mathrm{ave})^2 \tag{7.15}$$

$$\mathrm{ave} = \frac{|Q|}{|E|} \tag{7.16}$$

7.3.3. Sticky Entry Function

The sticky entry function of FFRT-Chord is the same as that of FRT-Chord.

7.3.4. Entry Learning Procedure

The entry learning procedure of FFRT-Chord is the same as that of FRT-Chord.

7.3.5. Entry Filtering Procedure

The entry filtering procedure of FFRT-Chord is summarized as follows:

Procedure 7.3.1: Entry Filtering in FFRT-Chord

[Step 1] `LET` $C = E$.

[Step 2] `LET` $C = (C \setminus \text{FRT-Chord::StickyEntry}(E))$.

[Step 3] `REMOVE` $\underset{c \in C}{\operatorname{argmin}}(E^{-c}, \leq_{\mathrm{FL}})$ `FROM` E.

7.3.6. Entry Filtering Procedure for Implementation

The value v_e^E is defined as follows:

Definition 7.3.4

$$v_e^E = \mathrm{V}(F_E) - \mathrm{V}(F_{E^{-e}}) \tag{7.17}$$

The following theorem holds:

Theorem 7.3.1

$$E^{-e^*} \leq_{\mathrm{FL}} E^{-e} \Leftrightarrow v_{e^*}^E \geq v_e^E. \tag{7.18}$$

Thus, the following entry filtering procedure selects the same entry to be removed as Procedure 7.3.2.

Procedure 7.3.2: Entry Filtering in FFRT-Chord for Implementation

[Step 1] `LET` $C = E$.

[Step 2] `LET` $C = (C \setminus \text{FRT-Chord::StickyEntry}(E))$.

[Step 3] `REMOVE` $\underset{c \in C}{\operatorname{argmax}}(v_c^E)$ `FROM` E.

This entry filtering procedure executes quickly by updating sorted sequence of $\left(v_e^E\right)_{e \in E}$.

7.3.7. Reachability Guarantee Procedures

The reachability guarantee procedures of FFRT-Chord are the same as that of FRT-Chord.

7.3.8. Performance

By routing tables constructed based solely on query flow, FFRT-Chord achieves path lengths that are nearly the same as those achieved by FRT-Chord. FFRT-Chord achieves efficient lookups with applications, e.g., range queries, even if the logical positions of nodes or queries are distributed non-uniformly as long as they are not heavily biased.

7.4. PFRT-Chord

Proximity-aware FRT-Chord[31] (PFRT-Chord) is a simple extension of FRT-Chord to consider both logical positions and network size adaptability. PFRT-Chord reduces routing latencies while maintaining the advantageous features of FRT such as dynamic and arbitrary routing table size and a one-hop property.

7.4.1. Remaining Distance

The remaining distance of PFRT-Chord is the same as that of FRT-Chord.

7.4.2. Routing Table Order

The routing table order of PFRT-Chord is the same as that of FRT-Chord.

7.4.3. Entry Learning Procedure

The entry learning procedure of PFRT-Chord is similar to that of FRT-Chord. The difference is that nodes measure the proximity of each entry e. The proximity is referred to as e.proximity.

PFRT-Chord adopts communication latency as the index of proximity. Each node n measures the round trip time $\mathrm{l}(n, e)$ between n and an entry e when sending a round-trip message.

7.4.4. Sticky Entry Function

Sticky entry function of PFRT-Chord is defined as follows:

Definition 7.4.1: Sticky Entry Function of PFRT-Chord

$$\begin{aligned}&\text{PFRT-Chord::StickyEntry}(E)\\ &\stackrel{\text{def}}{=} \{e_0\} \cup \{e \in E \mid e.\text{proximity} < e_a.\text{proximity}\},\end{aligned} \tag{7.19}$$

where e_a is the latest entry added to the routing table in the entry learning procedure.

In this sticky entry function, e_a is a threshold entry for considering network proximities.

7.4.5. Entry Filtering Procedure

The entry filtering procedure of PFRT-Chord is summarized as follows:

Procedure 7.4.1: Entry Filtering in PFRT-Chord

[Step 1] `LET` C = E

[Step 2] `LET` C = $(C \setminus \text{PFRT-Chord::StickyEntry}(E))$.

[Step 3] `REMOVE` $\operatorname{argmin}_{c \in C}(E^{-c}, \leq_{\text{FRT-Chord}})$ `FROM` E.

I focus on an entry filtering procedure just after an entry learning procedure. Let E_{next} be the routing table after the entry filtering procedure and let E_{prev} be the routing table prior to the entry learning procedure. thus, the following theorems hold.

Theorem 7.4.1

$$E_{\text{next}} \leq_{\text{FRT-Chord}} E_{\text{prev}} \tag{7.20}$$

Theorem 7.4.2

$$E_{\text{next}} \leq_{\text{PR}} E_{\text{prev}}, \tag{7.21}$$

where $\leq_{\text{PR}}$ is defined as follows:

$$E_1 \leq_{\text{PR}} E_2 \overset{\text{def}}{\Leftrightarrow} \frac{1}{E_1} \sum_{e \in E_1} e.\text{proximity} \leq \frac{1}{E_2} \sum_{e \in E_2} e.\text{proximity}. \tag{7.22}$$

These two theorems show that PFRT-Chord simultaneously improves two factors, logical positions and network proximities of entries.

7.4.6. Reachability Guarantee Procedures

The reachability guarantee procedures of PFRT-Chord are the same as that of FRT-Chord.

7.4.7. Performance

Experimental results with transit-stub model[50] networks show that PFRT-Chord achieves less routing latencies than those in Chord and LPRS-Chord[51], which is an existing proximity-aware routing algorithm. PFRT-Chord maintains the desirable features derived from FRT.

7.5. Conclusion

In this chapter, I have introduced FRT-based algorithms to demonstrate applications of FRT for various targets. These algorithms achieve desirable features while maintaining the desirable features derived from FRT, such as dynamic and arbitrary routing table size and network size adaptability.

The features of each algorithm are summarized as follows. FRT-2-Chord achieves a symmetric routing table using asymmetric remaining distance. FRT-Chord$^{\#}$ supports non-uniform logical position distributions using a routing table order based on neighbor routing tables. FFRT-Chord supports non-uniform logical position distributions using a routing table order based on a unique factor, i.e., query flows. PFRT-Chord reduces routing latency by a sticky entry function that considers network proximities. Note that these algorithms are all designed based on the FRT framework. As such, these algorithms are compact and simple, and their definitions have a lot in common.

Each algorithm can reuse implementations despite the fact that they offer different characteristics. This feature is derived from the extensibility of FRT.

In the next chapter, I introduce a method to merge these different characteristics.

Chapter 8

Mergeable-FRT

In Chapter 6 and Chapter 7, I introduced various types of FRT-based algorithms. However, no algorithm exists to consider two or more metrics in addition to path length. In structured overlays for real applications, multiple metrics (other than path length) should be considered. Thus, in this chapter, I propose mergeable-FRT[36], an FRT-based framework to design such structured overlay algorithms. Mergeable-FRT improves the modularity and reusability of the FRT-based extended algorithms that reflect factors other than logical positions, and mergeable-FRT offers a method to merge these algorithms and design new algorithms that consider the various metrics considered by the original algorithms.

Mergeable-FRT has similar structure to that of the FRT framework, and its entry filtering procedure consists of two parts, sticky entry ([Step 2] in Figure 8.1) and routing table order ([Step 3] in Figure 8.1). Mergeable-FRT differs from FRT in that the inner structure of the sticky entry is redefined to support algorithm merging. Specifically, using the merge method of mergeable-FRT, algorithm designers can merge two or more sticky entry parts of different algorithms with the same distance function and same routing table order. The method produces a new mergeable-FRT-based algorithm with the sticky entry parts derived from that of the merged algorithms with the same distance function and same routing table order. Note that this does not mean that the new algorithm automatically inherits all characteristics of the original algorithms.

In this chapter, I propose the mergeable-FRT framework and two merged algorithms, i.e., PGFRT-Chord and GPFRT-Chord, by reusing implementations of mergeable-FRT-based GFRT-Chord and PFRT-Chord. For the sticky entry part of mergeable-FRT, I also propose the minimum through parameter to balance path length considerations as well as considerations for other factors. Experimental results show that PGFRT-Chord and GPFRT-

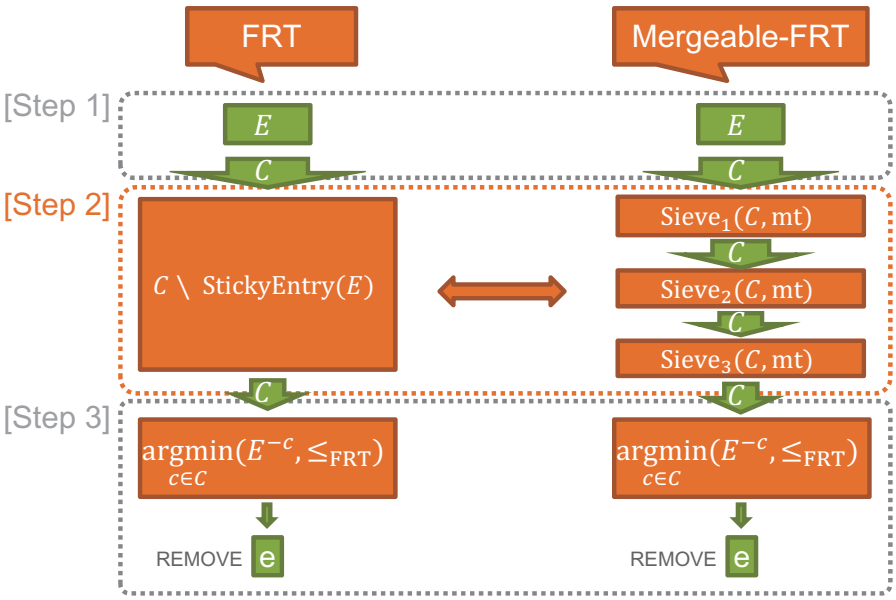


Figure 8.1: Entry filtering procedure of mergeable-FRT with three sieve functions vs. that of FRT.

Chord exhibit merged features of the original algorithms, and that the minimum through parameter supports control of the balance between path length considerations and other factors considerations.

8.1. Background

In structured overlay networks, it is essential to shorten path length. However, various metrics other than the number of hops should be considered, such as the physical network topology[13, 39, 40, 43], network latencies[14] and throughput[7], node lifespan[4, 37]. As described previously, various extended algorithms to consider one or some of these metrics other than logical positions, i.e., path length, have been proposed. However, a real-world application requires a suitable set of metrics, and the set varies by application.

I propose mergeable-FRT, a structured overlay algorithm design frame-

work that supports efficient design of algorithms by improving modularity and reusability. A mergeable-FRT-based extended algorithm can be designed by merging an existing mergeable-FRT-based extended algorithm with new metrics.

8.2. Sticky Entry Sieve Function

In mergeable-FRT, a **sticky entry sieve function** plays the most important role. A sticky entry sieve function Sieve(C, mt) is a function that must be designed in accordance with the following definition:

Definition 8.2.1: Sticky Entry Sieve Function

A **sticky entry sieve function** Sieve(C, mt) must meet the following requirements.

- C is a set of candidate entries.
- Minimum through mt is a positive integer
- Returns a subset of C
- The number of entries in the returned set is greater than or equal to mt

For the mergeable-FRT framework, entry filtering is defined using a **routing table order** ($\leq_{\mathrm{FRT}}$), a **sequence of entry filtering sieves** (SieveSeq), and a **minimum through parameter** (mt).

Procedure 8.2.1: Entry Filtering in Mergeable-FRT

The entry filtering procedure for mergeable-FRT is defined by a sequence of sieves SieveSeq as follows:

[Step 1] `LET` C `=` E.

[Step 2] `FOR` Sieve `IN` SieveSeq { `LET` C `=` Sieve(C, mt) }

[Step 3] `REMOVE` $\underset{c \in C}{\operatorname{argmin}}(E^{-c}, \leq_{\mathrm{FRT}})$ `FROM` E.

[†] C is a temporary variable.
[††] mt is a *minimum through* parameter.

The deference between the entry filtering procedure of mergeable-FRT and that of FRT is in Step 3 (Figure 8.1).

8.3. Minimum Through Parameter

[Step 2] of Procedure 8.2.1 applies all sieve functions in a sequence of sieves SieveSeq to a set of candidates C. However, entry filtering sieve functions do nothing after the number of entries in C is mt (Definition 8.2.1). This behavior guarantees that the selection using routing table order $\leq_{\mathrm{RT}}$ receives mt or more entries (Figure 8.2). This behavior solves the problems of FRT whereby the sequence of sieve functions limits candidates to be removed extremely rigidly to reduce path lengths in [Step 3].

The minimum through parameter mt adjusts the extent of the influence that the sequence of sieve functions has on a routing table and the influence that routing table order has on a routing table. Priority will be given to the reduction of path lengths when a large minimum through value is used. This feature will be examined in simulations.

When minimum through parameter mt is $O(\log |N|)$, Theorem 8.3.1 holds regardless of the sequence of sieves.

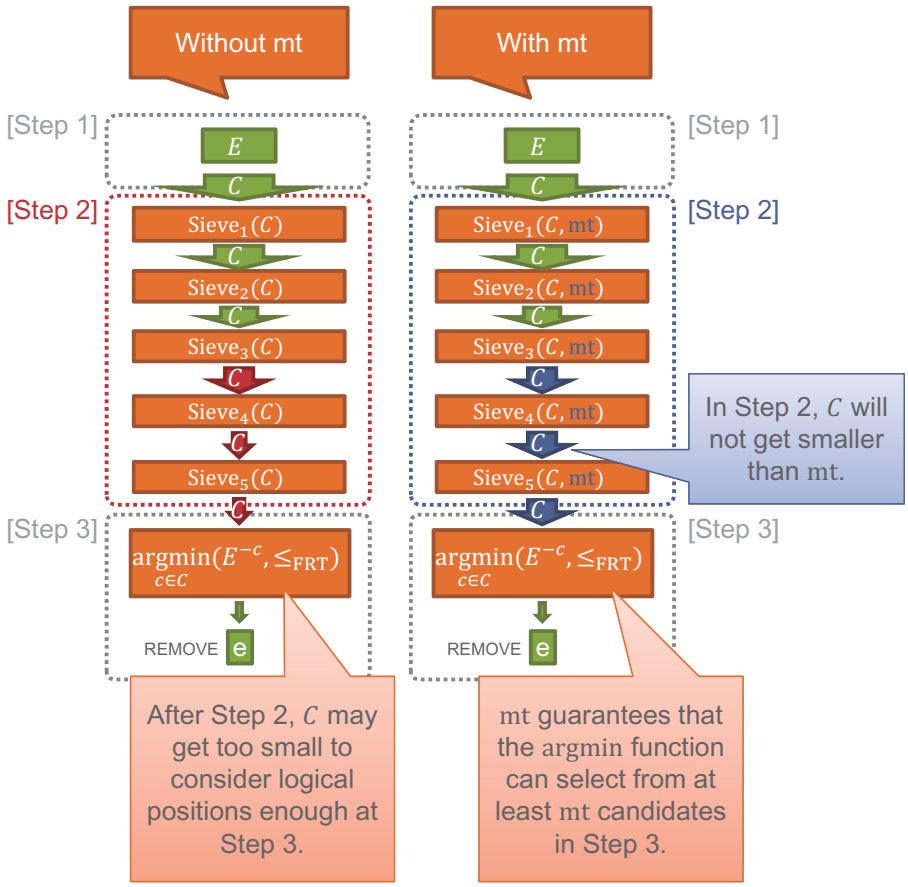


Figure 8.2: Role of minimum through parameter mt in mergeable-FRT.

Theorem 8.3.1: Minimum Through

In an FRT-based Chord algorithm network with $|N|$ nodes, the path length is $O(\log|N|)$ with high probability when all nodes have convergent routing tables with $L = O(\log|N|)$ entries and $\mathrm{mt} > L/2$.

Proof.

Because E is convergent, for each entry $e_{\mathrm{free}} \in E$ that belongs to a set of candidates C in [Step 3] of the entry filtering procedure

of $(E^{+e_{\text{add}}})$:

$$E^{+e_{\text{add}}-e_{\text{add}}}(=E) \leq_{\text{FRT-Chord}} E^{+e_{\text{add}}-e_{\text{free}}}. \tag{8.1}$$

Letting e_j be e_{free},

$$S_i \leq S_{j-1} + S_j. \tag{8.2}$$

By adding both sides for all j,

$$S_i = \frac{\Sigma_{j\in J} S_i}{|J|} \leq \frac{2}{|J|}\Sigma_{j\in E\setminus\{s\}} S_j = \log\left(\frac{2^m}{\mathrm{d}(s,e_1)}\right)^{\frac{2}{|J|}}. \tag{8.3}$$

Because any distance $\mathrm{d}(s,e_1) > 2^m/|N|^2$, the following equation holds with high probability when $|N|$ is suficiently large:

$$\mathrm{r}_i(E) \leq 1 - (1/|N|)^{\frac{4}{|J|}}. \tag{8.4}$$

Thus, because $\mathrm{mt} \leq |J|$ when $L = 8\log N$ and $\mathrm{mt} = 4\log N$, the number of times of such forwarding is performed by the time the rest distance becomes smaller than $2^m/|N|$ is expressed as follows:

$$\log_{\mathrm{r}_i(E)}(1/|N|) \leq \log|N|. \tag{8.5}$$

□

This theorem shows that the minimum through parameter can guarantee $O(\log|N|)$ path length even if the number of sieves is large by merging. In Section 8.6, I examine the variance of path length by changing the minimum through parameter.

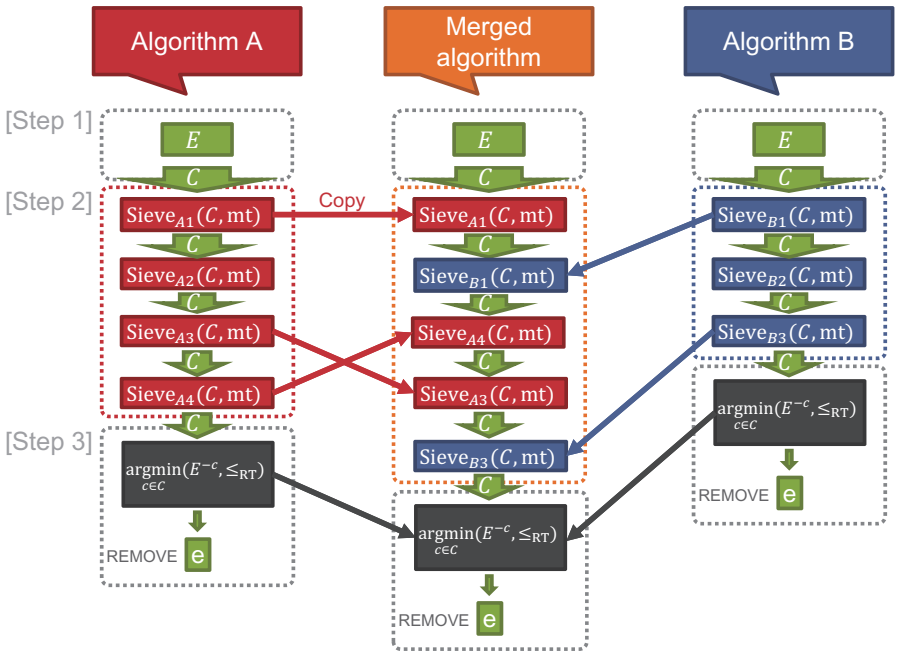


Figure 8.3: Algorithm merging with two mergeable-FRT-based algorithms.

8.4. Algorithm Extension Merging

Mergeable-FRT offers a method to merge the second steps of entry filtering procedures and to produce a new algorithm, and the new algorithm has features derived from the entry filtering procedures of original algorithms Note that this does not mean that the new algorithm automatically inherits all characteristics of the original algorithms.

An entry filtering procedure of a new algorithm can be designed by sorting entry filtering sieves selected from those of existing original algorithms with the same logical distance and same routing table order (Figure 8.3). This is the merging method of mergeable-FRT.

8.5. Algorithms based on Mergeable-FRT

In this section, I propose two mergeable-FRT-based algorithms and two algorithms designed using the algorithm merging method.

8.5.1. Sieve Sequence for FRT-Chord

I redefine FRT-Chord to be compatible with mergeable-FRT. The sequence of FRT-Chord is defined by two sieves, i.e., **SelfSieve** and **SuccessorSieve**.

SelfSieve is a sieve designed to remove an owner of a routing table from candidates to be removed.

Definition 8.5.1: Self Sieve

$$\text{SelfSieve}(C, \text{mt}) = C \setminus \{e_0\} \tag{8.6}$$

SuccessorSieve is a sieve that protects a succeeding node in a clockwise direction to guarantee message reachability.

Definition 8.5.2: Successor Sieve

$$\text{SuccessorSieve}(C, \text{mt}) = C \setminus \{e_1\} \tag{8.7}$$

A sequence of sieves for FRT-Chord is defined as follows:

Definition 8.5.3: Sieve Sequence for FRT-Chord

$$\text{FRT-Chord::SieveSeq} = (\textbf{SelfSieves}, \textbf{SuccessorSieve}). \tag{8.8}$$

The entry filtering procedure of mergeable-FRT-based FRT-Chord is summarized in Figure 8.4.

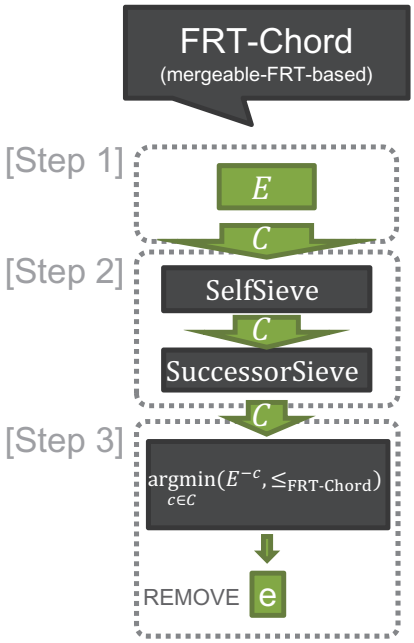


Figure 8.4: Entry filtering procedure of mergeable-FRT-based FRT-Chord.

FRT-Chord defined based on mergeable-FRT is equivalent to FRT-Chord.

8.5.2. Sieve Sequence for GFRT-Chord

GroupSuccessorSieve and **GroupSieve** are defined as follows:

Definition 8.5.4: Group Successor Sieve

$$\text{GroupSuccessorSieve}(C, \text{mt}) = C \setminus \{g_1\}, \tag{8.9}$$

Definition 8.5.5: Group Sieve

$$\text{GroupSieve}(C, \text{mt}) = \tag{8.10}$$

$$\begin{cases} C, & \text{if Leap}(E) = \emptyset \vee |C \setminus \text{G}(E)| \leq \text{mt} \\ C \setminus \text{G}(E), & \text{otherwise.} \end{cases} \tag{8.11}$$

A sequence of sticky entry sieves for GFRT-Chord is defined as follows.

Definition 8.5.6: Sieve List of GFRT-Chord

$$\text{GFRT-Chord::SieveSeq} = \text{FRT-Chord::SieveSeq} \tag{8.12}$$

$$\| \ (\textbf{GroupSuccessorSieve}, \textbf{GroupSieve}). \tag{8.13}$$

* || is a sequence concatenation operator.

The entry filtering procedure of mergeable-FRT-based GFRT-Chord is summarized in Figure 8.5.

When mt = 1, mergeable-FRT-based GFRT-Chord is equivalent to GFRT-Chord.

8.5.3. Sieve Sequence for PFRT-Chord

In PFRT-Chord, for each routing table entry e, e.proximity is a value that represents network proximity. Here, Dist(C) is defined as follows:

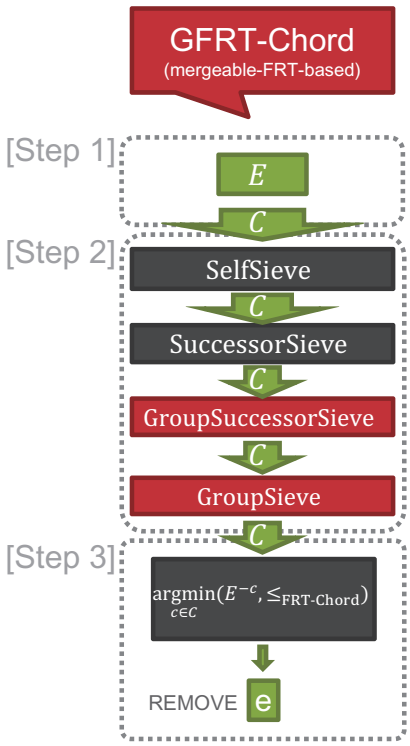


Figure 8.5: Entry filtering procedure of mergeable-FRT-based GFRT-Chord.

Definition 8.5.7

$$\mathrm{Dist}(C) = \{c \in C \mid c_{\mathrm{add}}.\mathrm{proximity} < c.\mathrm{proximity}\}, \tag{8.14}$$

where c_{add} is a previously added entry.

Let $\{c_i\}_{i=1,\ldots,|C|}$ represent the sorted elements of C in ascending order of proximity (c.proximity). Then, **ProximitySieve** is defined as follows:

Definition 8.5.8

$$\text{ProximitySieve}(C, \text{mt}) = \begin{cases} \text{Dist}(C) & (\text{mt} \leq |\text{Dist}(C)|) \\ \{c_i\}_{i=|C|-\text{mt}+1,\ldots,|C|} & (\text{otherwise}). \end{cases} \tag{8.15}$$

ProximitySieve protects entries that have a better network proximity value than an entry e_{add} that is learned in the previous entry learning. ProximitySieve is optimized to return mt worst entries when the returned value is less than mt.

A sequence of sieves for PFRT-Chord is defined as follows:

Definition 8.5.9

$$\begin{aligned} \text{PFRT-Chord::SieveSeq} \quad = \quad & \text{FRT-Chord::SieveSeq} \\ & \text{|| (ProximitySieve).} \end{aligned} \tag{8.16}$$

The entry filtering procedure of mergeable-FRT-based PFRT-Chord is summarized in Figure 8.6.

When mt = 1, PFRT-Chord based on mergeable-FRT is equivalent to PFRT-Chord based on FRT.

PFRT-Chord has a feature whereby the average proximity of a routing table is refined by each entry improvement procedure.

8.5.4. PGFRT-Chord and GPFRT-Chord

PGFRT-Chord is a merged mergeable-FRT-based algorithm defined as follows:

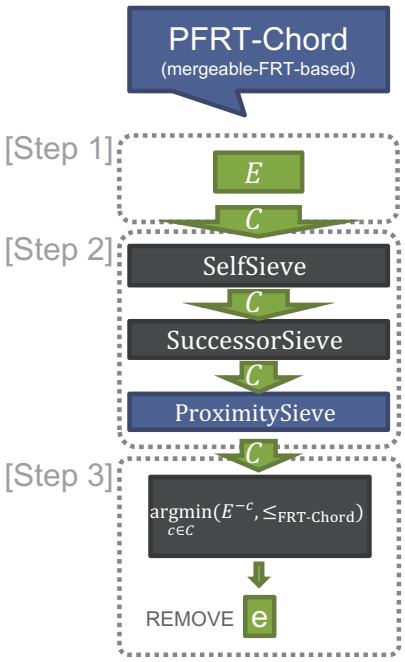


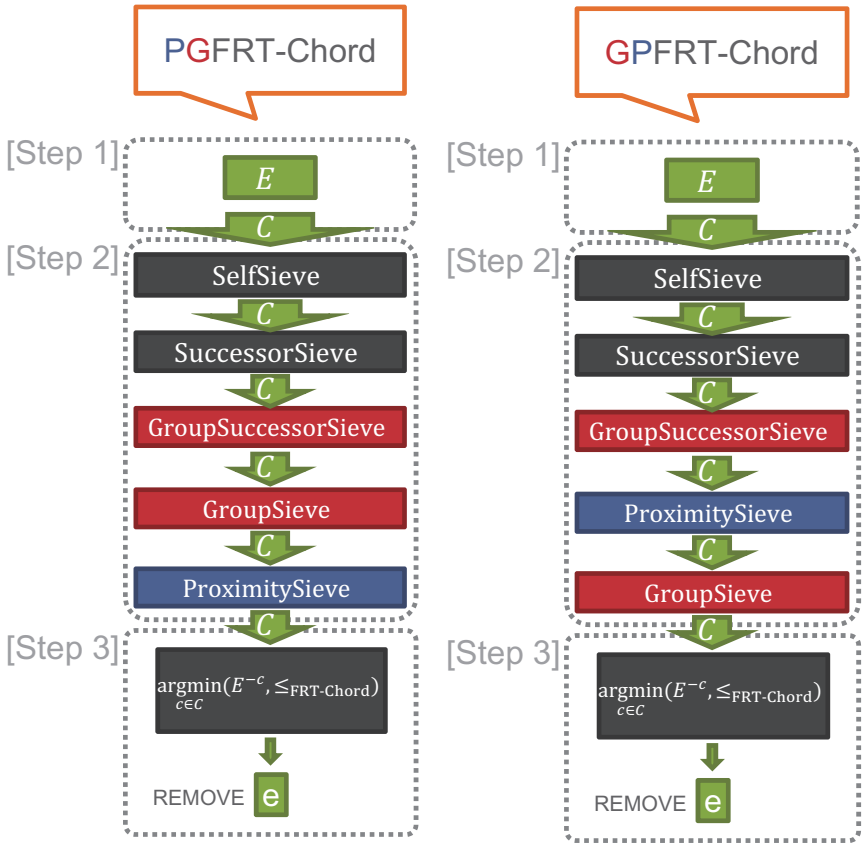
Figure 8.6: Entry filtering procedure of mergeable-FRT-based PFRT-Chord.

Definition 8.5.10

$$\begin{aligned} \textbf{PG}\text{FRT-Chord::SieveSeq} \quad = \quad & \text{FRT-Chord::SieveSeq} \\ & \| \ (\text{GroupSuccessorSieve}, \\ & \quad \textbf{GroupSieve}, \\ & \quad \textbf{ProximitySieve}). \end{aligned} \tag{8.17}$$

This merged algorithm reflects both node groups and network proximity in addition to logical positions using two sieves, GroupSieve and ProximitySieve. The entry filtering procedure of mergeable-FRT-based PGFRT-Chord is summarized in Figure 8.7a.

On the other hand, GPFRT-Chord is defined as follows:



(a) PGFRT-Chord (b) GPFRT-Chord

Figure 8.7: Entry filtering procedures of two merged algorithms.

Definition 8.5.11

$$\begin{aligned} \textbf{GP}\text{FRT-Chord::SieveSeq} \quad = \quad & \text{FRT-Chord::SieveSeq} \\ & \| \ (\text{GroupSuccessorSieve}, \\ & \quad \textbf{ProximitySieve}, \\ & \quad \textbf{GroupSieve}). \end{aligned} \tag{8.18}$$

The entry filtering procedure of mergeable-FRT-based GPFRT-Chord is summarized in Figure 8.7b.

The order of the sieves differs from PGFRT-Chord. The order of sieves represents a design parameter for merging algorithms based on mergeable-FRT.

8.6. Experimental Results

In this section, I compare extended algorithms based on mergeable-FRT and its merged algorithms PGFRT-Chord and GPFRT-Chord.

I simulate the network latencies between any two nodes using a transit-stub model[50]. The communication latencies of the inter-transit node links, stub-transit node links and inter-stub node links were set at 100, 20, and 5 ms, respectively.

I adopt two methods for assigning node groups to nodes: transit group assignment and random group assignment. The transit group assignment assigns the same node group to the nodes under the same transit node, and the random group assignment assigns node groups at random independent of the transit-stub structure.

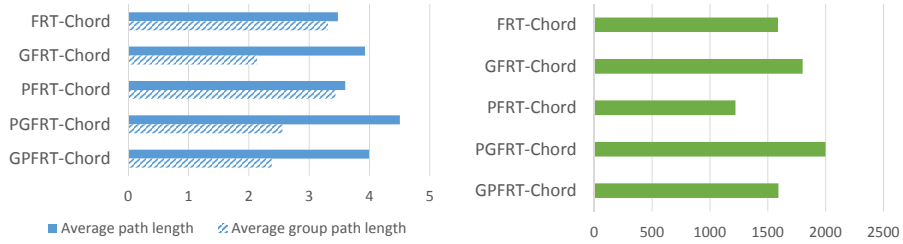
The number of node groups $|G|$ is 20, and the number of nodes in each group is 50 for both the transit group assignment and random group assignment.

In the experiments, the number of nodes $|N|$ is 1000, the routing table size L is 20 and I adopt an iterative routing style.

After $|N|$ nodes join a system, each node repeats sending a query 100 times to logical positions chosen randomly to improve routing tables. A query is sent 10000 times to logical positions chosen randomly, and the forwarding results are analyzed.

8.6.1. Algorithm Merging

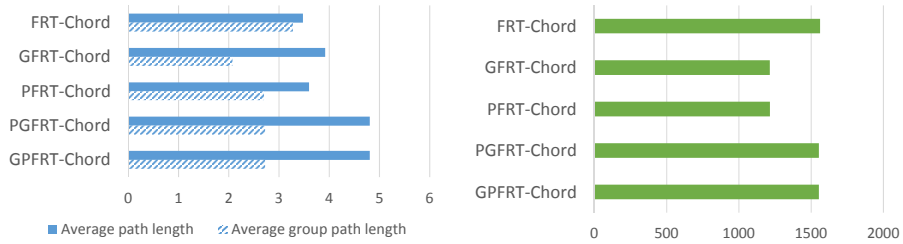
I examine the extent to which the mergeable-FRT-based algorithms, PGFRT-Chord and GPFRT-Chord, inherit the features of GFRT-Chord and PFRT-Chord.



(a) Average path length and average group path length

(b) Average network latency (ms)

Figure 8.8: Random group assignment.



(a) Average path length and average group path length

(b) Average network latency (ms)

Figure 8.9: Transit group assignment.

I adopt mt = 1 such that GFRT-Chord and PFRT-Chord perform as well as the originals. Figure8.8 and Figure8.9 shows path lengths, group path lengths and latencies for random group assignment and transit group assignment respectively.

For random group assignment, the experimental results indicate that GPFRT-Chord inherits the features GFRT-Chord and PFRT-Chord. In Figure8.8, GPFRT-Chord performs better than PGFRT-Chord in path lengths, group path lengths and latencies. This indicates that the sieve order of the merged algorithms effect their performance.

For transit group assignment shown in Figure8.9, the performance of GFRT-Chord is similar to the performance of PFRT-Chord because inter-group communication latencies are likely to be small so that the effect of GroupSieve and the effect of ProximitySieve are similar. In this case, a performance difference between PGFRT-Chord and GPFRT-Chord is scarcely

discernible.

8.6.2. Minimum-Through Parameter

Here, I examine the extent to which the minimum through parameter mt balances the effect of extensions by a sequence of sieves with the effect of the path length reduction by the routing table order $\leq_{\mathrm{FRT}}$. To this end, I set mt to 1, 54, 10 and 15 for each algorithm.

For all extended algorithms, path length is shortened so that minimum through parameter mt (Figure 8.6.2 and Figure 8.6.2) .

When mt is a small value, the effect of extensions is strong in many cases. Therefore, configuring the minimum through parameter balances the effect of path length reduction and the effect of extensions.

For PGFRT-Chord and GPFRT-Chord, when mt = 5 and mt = 10, the group path length and network latency are better than the algorithms for mt = 1 (Figure 8.6.2, Figure 8.6.2, Figure 8.6.2).

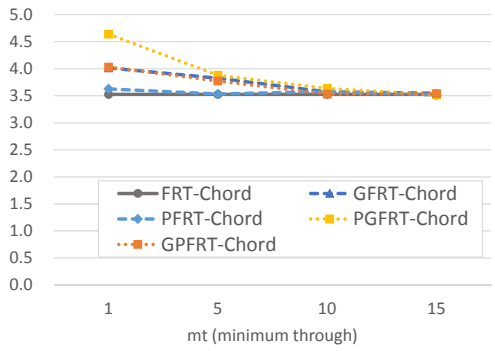
This indicates that path length is extended by setting mt too small and results in exceeding the recduction effect of the group path length and the latency. Thus, it is confirmed that the performance of merged algorithms can be improved by appropriately configuring the minimum through parameter mt.

8.7. Conclusion

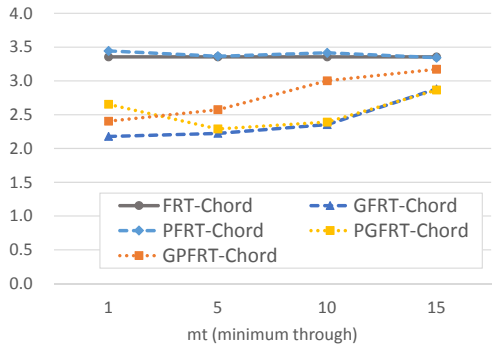
In this Chapter, I have proposed mergeable-FRT, which is an algorithm design framework to design structured overlay algorithms that consider two or more metrics in addition to logical positions.

Mergeable-FRT improves algorithm reusability by utilizing extensibility derived from the FRT framework, and a mergeable-FRT-based algorithm can be merged with another mergeable-FRT-based algorithm to produce new algorithms that consider two or more metrics. Note that this does not mean that the new algorithms automatically inherit all characteristics of the original algorithms.

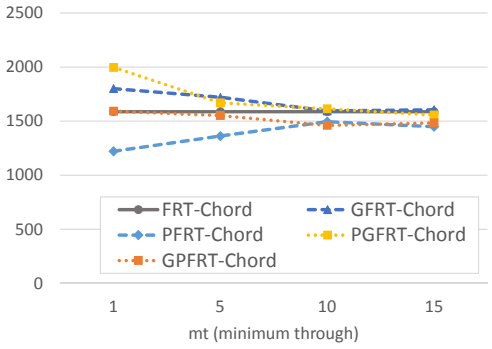
Using mergeable-FRT, I have redesigned and implemented GFRT-Chord



(a) Path length

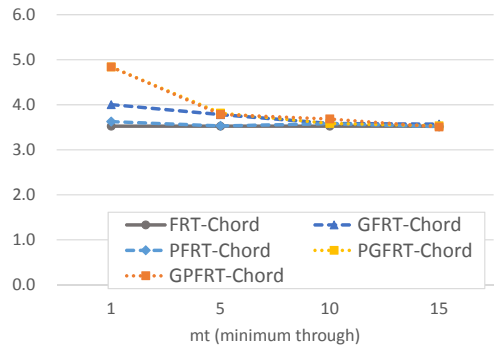


(b) Group path length

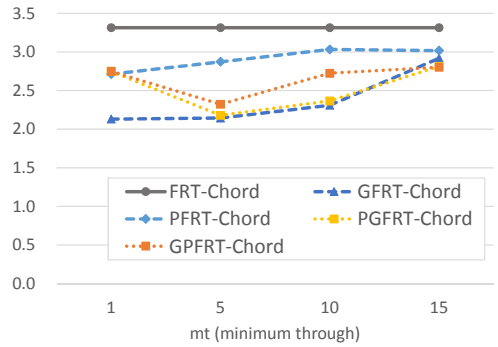


(c) Network latency (ms)

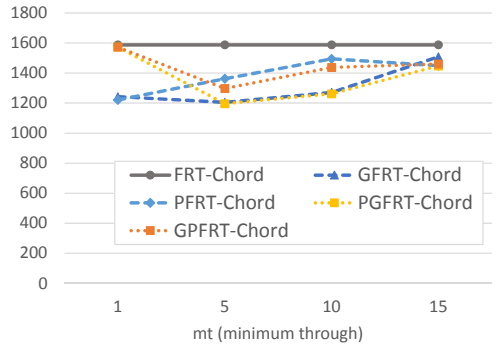
Figure 8.10: Changing minimum through parameter (random group assignment).



(a) Path length



(b) Group path length



(c) Network latency (ms)

Figure 8.11: Changing minimum through parameter (transit group assignment).

and PFRT-Chord. Furthermore, I have designed and implemented their merged algorithms, PGFRT-Chord and GPFRT-Chord, which consider both node groups and network proximities.

Experimental results show that the extended algorithms exhibit merged features of the original algorithms and that the minimum through parameter supports control of the balance between logical position considerations and consideration of other factors.

Chapter 9

Conclusion

In this thesis, I have proposed flexible routing tables (FRT), an algorithm design framework for structured overlays. Furthermore, I have introduced concrete FRT-based algorithms that demonstrate desirable features derived from FRT such as dynamic and arbitrary routing table size and network size adaptability without restricting routing table candidates.

Existing structured overlays construct desirable routing tables by restricting the candidates for the routing table or limiting the way in which entries are learned. As a result, these restrictions constrict adaptability relative to the number of nodes and the number of routing table entries. As such, they also constrict the ability to extend such algorithms.

In Chapter 2, I introduced the basic structure of routing algorithms for structured overlays. In structured overlays, each node builds its own routing table. A message is forwarded according to logical positions to reach its responsible node. The entry each node a message is forwarded to is defined by a next hop function; therefore, message flows are determined by routing tables. Routing table construction is an essential and vitally important part of structured overlays, and I have focused on this construction method in this thesis.

In Chapter 3, I introduced three methodologies to construct routing tables with example algorithms, i.e., Chord, Symphony, and Accordion. Symphony and Accordion relax routing table candidate restrictions by a probabilistic approach and a biased learning approach, respectively. However, Symphony cannot increase routing table size while maintaining routing table efficiency. Moreover, Symphony has no space to consider factors other than logical positions. In contrast, Accordion can increase routing table size. However, Accordion limits the method by which entries are learned and the behavior of its applications; thus, Accordion lacks extensibility.

In Chapter 4, I proposed flexible routing tables (FRT) designed to achieve the desirable features described in Chapter 1, i.e., dynamic and arbitrary routing table size and network size adaptability without restricting routing table candidates. FRT offers two procedures, entry learning and entry filtering. Entry learning adds an entry that corresponds to a node to be learned to the routing table. FRT is characterized by considering logical positions in the entry filtering procedure. An entry filtering procedure evicts an entry from the routing table according to the order of the routing table space and a sticky entry function. Thus, FRT achieves dynamic and arbitrary routing table size and network size adaptability.

In Chapter 5, I proposed FRT-Chord to demonstrate that concrete algorithms can be designed based on FRT. FRT-Chord defines routing table order according to the worst-case reduction ratios, and FRT-Chord defines entries for the same role as that of the sticky entry in Chord using the sticky entry function of the FRT framework. With such definitions, the nodes of FRT-Chord can maintain message reachability and repeatedly improves routing tables using the entry learning and entry filtering procedures. I have shown that the routing table improvement process based on routing table order achieves $O(\log |N|)$ path length with high probability. Experimental results show that FRT-Chord achieves other desirable features derived from FRT such as dynamic and arbitrary routing table size and network size adaptability.

In Chapter 6, I showed that FRT-based extended algorithms can be designed to consider other metrics in addition to path length with the same desirable features of FRT by proposing a concrete algorithm, i.e., GFRT-Chord. GFRT-Chord considers node groups in addition to logical positions. GFRT-Chord reduces inter-group hops while reducing entire path lengths. GFRT-Chord considers group path length and inherits the desirable features of FRT-Chord. Thus, GFRT-Chord restricts the growth of the entire path length regardless of whether the number of nodes in the system or the number of nodes that belong to each node group is extremely small or large. Moreover, GFRT-Chord does not need to know the number of nodes and groups in advance; therefore, it can be used when this information cannot be predicted. I have shown that path length is $O(\log |N|)$ in two typical network-growth models, and I have also demonstrated that, after a message leaves a node group, the message will never be forwarded to any other node that belongs to the same group, i.e., a message does not pass the same group twice. I have verified the features of the proposed routing algorithm in sim-

ulations. The proposed method was compared to existing algorithms and a simple algorithm that uses node groups. I also designed a naive grouped FRT-Chord algorithm that is simpler than the proposed GFRT-Chord. I have shown that this algorithm experiences problems when network and algorithm parameters meet certain conditions, and I also demonstrated that GFRT-Chord consistently performs efficient routings in such cases.

In Chapter 7, I introduced important FRT-based algorithms to demonstrate applications of the FRT framework for various targets. These algorithms achieve unique desirable features while maintaining the desirable features derived from FRT. The features of each algorithm are summarized as follows. FRT-2-Chord achieves a symmetric routing table using asymmetric remaining distance. FRT-Chord$^{\#}$ supports non-uniform logical position distributions using a routing table order based on neighbor routing tables. FFRT-Chord supports non-uniform logical position distributions using a routing table order based on a unique factor, i.e., query flows. PFRT-Chord reduces routing latencies using a sticky entry function that considers network proximities. These algorithms are designed based on the FRT framework; thus, their definitions are simple and compact, and have a lot in common. Each algorithm can reuse implementations despite the fact that they offer different characteristics. This feature is derived from the extensibility of the FRT framework.

In Chapter 8, I proposed mergeable-FRT, an algorithm design framework to design structured overlay algorithms that consider two or more metrics in addition to logical positions. Mergeable-FRT improves algorithm reusability by utilizing the extensibility derived from the FRT framework. A mergeable-FRT-based can be merged with another mergeable-FRT-based algorithm to produce algorithms that consider two or more metrics (Note that this does not mean that the new algorithms automatically inherit all characteristics of the original algorithms). Using mergeable-FRT, I redesigned and implemented GFRT-Chord and PFRT-Chord. I designed and implemented their merged algorithms, PGFRT-Chord and GPFRT-Chord, which consider both node groups and network proximities. Experimental results show that the extended algorithms exhibit features of the original algorithms and that the minimum through parameter supports control of the balance between logical position considerations and consideration of other factors.

By proposing concrete algorithms based on the FRT framework and the mergeable-FRT framework, I have demonstrated that the desirable features and abilities extend to FRT-based algorithms. FRT-based algorithms are

simple and compact, and the FRT framework improves modularity and reusability. I believe that the FRT frameworks will enable the design of new algorithms based on various ideas, which may lead to a systematical design methodology for structured overlay algorithms.

Acknowledgment

I am deeply grateful to my adviser, Associate Prof. Kazuyuki Shudo, for his practical support. I want to thank all members in our group, and discussion with Mr. Takashi Yaguchi and Mr. Takehiro Miyao has been especially insightful.

Reference

[1] Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., and Schmidt, R. P-grid: A self-organizing structured p2p system. *SIGMOD Rec.*, 32(3):29–33, Sept. 2003. ISSN 0163-5808. doi: 10.1145/945721.945729. URL `http://doi.acm.org/10.1145/945721.945729`.

[2] Ando, Y., Nagao, H., Miyao, T., and Shudo, K. FRT-2-Chord: A DHT Supporting Seamless Transition Between One-Hop and Multi-Hop Lookups With Symmetric Routing Table. In *Information Networking (ICOIN), 2014 International Conference on*, pages 170–175, Feb 2014. doi: 10.1109/ICOIN.2014.6799686.

[3] Ando, Y., Nagao, H., and Shudo, K. Routing Table Construction Method Solely Based on Query Flows for Structured Overlays. In *Proc. 2014 14th Int. Conf. Peer-to-Peer Computing*, Sep 2014.

[4] Beverly Yang, B. and Garcia-Molina, H. Designing a Super-peer Network. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 49–60, March 2003. doi: 10.1109/ICDE.2003.1260781.

[5] Chand, R. and Felber, P. A Scalable Protocol for Content-Based Routing in Overlay Networks. In *Network Computing and Applications, 2003. NCA 2003. Second IEEE International Symposium on*, pages 123–130, April 2003. doi: 10.1109/NCA.2003.1201146.

[6] Cordasco, G. and Sala, A. 2-Chord Halved. In *Hot Topics in Peer-to-Peer Systems, 2005. HOT-P2P 2005. Second International Workshop on*, pages 72–79, July 2005. doi: 10.1109/HOT-P2P.2005.1.

[7] Dabek, F., Li, J., Sit, E., Robertson, J., Kaashoek, M. F., and Morris, R. Designing a DHT for Low Latency and High Throughput. In *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*, NSDI'04, pages 7–7, Berkeley, CA, USA, 2004. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=1251175.1251182`.

[8] Datta, A., Hauswirth, M., John, R., Schmidt, R., and Aberer, K. Range Queries in Trie-structured Overlays. In *Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on*, pages 57–66, Aug 2005. doi: 10.1109/P2P.2005.31.

[9] Fonseca, P., Rodrigues, R., Gupta, A., and Liskov, B. Full-Information Lookups for Peer-to-Peer Overlays. *IEEE Trans. Parallel Distrib. Syst.*, 20:1339–1351, September 2009. ISSN 1045-9219. doi: 10.1109/TPDS.2008.222. URL `http://portal.acm.org/citation.cfm?id=1591896.1592267`.

[10] Fraigniaud, P. and Gauron, P. D2B: A de Bruijn Based Content-Addressable Network. *Theor. Comput. Sci.*, 355(1):65–79, Apr. 2006. ISSN 0304-3975. doi: 10.1016/j.tcs.2005.12.006. URL `http://dx.doi.org/10.1016/j.tcs.2005.12.006`.

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

[12] Freedman, M. J. and Vingralek, R. Efficient Peer-to-Peer Lookup Based on a Distributed Trie. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 66–75, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-44179-4. URL `http://dl.acm.org/citation.cfm?id=646334.687798`.

[13] Ganesan, P., Gummadi, K., and Garcia-Molina, H. Canon in G Major: Designing DHTs with Hierarchical Structure. In *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pages 263–272, 2004. doi: 10.1109/ICDCS.2004.1281591.

[14] Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., and Stoica, I. The Impact of DHT Routing Geometry on Resilience and Proximity. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, pages 381–394, New York, NY, USA, 2003. ACM. ISBN 1-58113-735-4. doi: 10.1145/863955.863998. URL `http://doi.acm.org/10.1145/863955.863998`.

[15] Guo, D., Wu, J., Chen, H., and Luo, X. Moore: An Extendable Peer-to-Peer Network Based on Incomplete Kautz Digraph With Constant Degree. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 821–829, May 2007. doi: 10.1109/INFCOM.2007.101.

[16] Guo, D., Liu, Y., Jin, H., Liu, Z., Zhang, W., and Liu, H. Theory and Network Applications of Balanced Kautz Tree Structures. *ACM Trans. Internet Technol.*, 12(1):3:1–3:25, July 2012. ISSN 1533-5399. doi: 10.1145/2220352.2220355. URL `http://doi.acm.org/10.1145/2220352.2220355`.

[17] Gupta, A., Liskov, B., and Rodrigues, R. Efficient Routing for Peer-to-Peer Overlays. In *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*, NSDI'04, pages 9–9, Berkeley, CA, USA, 2004. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=1251175.1251184`.

[18] Hu, J., Li, M., Zheng, W., Wang, D., Ning, N., and Dong, H. SmartBoa: Constructing P2P Overlay Network in the Heterogeneous Internet Using Irregular Routing Tables. In Voelker, G. M. and Shenker, S., editors, *IPTPS*, volume 3279 of *Lecture Notes in Computer Science*, pages 278–287. Springer, 2004. ISBN 3-540-24252-X. URL `http://dblp.uni-trier.de/db/conf/iptps/iptps2004.html#HuLZWND04`.

[19] Jia, C., Hongke, Z., and Huachun, Z. Topology-based Data Dissemination Approaches for Large Scale Data Centric Networking Architecture. *Communications, China*, 10(9):80–96, Sept 2013. ISSN 1673-5447. doi: 10.1109/CC.2013.6623506.

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

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

[22] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983. ISSN 00368075. doi: 10.1126/science.220.4598.671. URL `http://www.jstor.org/stable/1690046`.

[23] Kleinberg, J. The Small-World Phenomenon: An Algorithmic Perspective. In *Proc. ACM STOC 2000*, pages 163–170, 2000.

[24] Leong, B., Liskov, B., and Demaine, E. D. EpiChord: Parallelizing the Chord Lookup Algorithm with Reactive Routing State Management. *Comput. Commun.*, 29:1243–1259, May 2006. ISSN 0140-3664.

doi: 10.1016/j.comcom.2005.10.002. URL http://portal.acm.org/citation.cfm?id=1646651.1646839.

[25] Li, D., Lu, X., and Wu, J. FISSIONE: A Scalable Constant Degree and Low Congestion DHT Scheme Based on Kautz Graphs. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 1677–1688 vol. 3, March 2005. doi: 10.1109/INFCOM.2005.1498449.

[26] Li, J., Stribling, J., Morris, R., and Kaashoek, M. F. Bandwidth-efficient management of dht routing tables. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 99–114, Berkeley, CA, USA, 2005. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1251203.1251211.

[27] Li, X. and Plaxton, C. G. On Name Resolution in Peer-to-peer Networks. In *Proceedings of the Second ACM International Workshop on Principles of Mobile Computing*, POMC '02, pages 82–89, New York, NY, USA, 2002. ACM. ISBN 1-58113-511-4. doi: 10.1145/584490.584507. URL http://doi.acm.org/10.1145/584490.584507.

[28] Locher, T., Schmid, S., and Wattenhofer, R. equus: A provably robust and locality-aware peer-to-peer system. In *Peer-to-Peer Computing, 2006. P2P 2006. Sixth IEEE International Conference on*, pages 3–11, Sept 2006. doi: 10.1109/P2P.2006.17.

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

[30] Maymounkov, P. and Mazières, D. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Proc. IPTPS '02*, pages 53–65, 2002. ISBN 3-540-44179-4.

[31] Miyao, T., Nagao, H., and Shudo, K. A Method for Designing Proximity-aware Routing Algorithms for Structured Overlays. In *Computers and Communications (ISCC), 2013 IEEE Symposium on*, pages 000508–000514, July 2013. doi: 10.1109/ISCC.2013.6754997.

[32] Miyao, T., Nagao, H., and Shudo, K. A Structured Overlay for Non-Uniform Node Identifier Distribution Based on Flexible Routing Tables. In *Computers and Communication (ISCC), 2014 IEEE Symposium on*, pages 1–6, June 2014. doi: 10.1109/ISCC.2014.6912614.

[33] Monnerat, L. and Amorim, C. D1HT: A Distributed One Hop Hash Table. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 10 pp.–, April 2006. doi: 10.1109/IPDPS.2006.1639278.

[34] Nagao, H. and Shudo, K. Flexible Routing Tables: Designing Routing Algorithms for Overlays Based on a Total Order on a Routing Table Set. In *Peer-to-Peer Computing (P2P), 2011 IEEE International Conference on*, pages 72–81, Aug 2011. doi: 10.1109/P2P.2011.6038664.

[35] Nagao, H. and Shudo, K. GFRT-Chord: Flexible Structured Overlay Using Node Groups. In *Proc. 11thUbiquitous Intelligence and Computing, 2014 IEEE 11th International Conference on and 11th International Conference on Autonomic and Trusted Computing (UIC/ATC)*, pages 999–9999, Dec 2014.

[36] Nagao, H. and Shudo, K. Mergeable Flexible Routing Tables: A Design Framework for Reusable Structured Overlay Algorithms. In *ICOIN 2015*, Jan 2015.

[37] Ohzahata, S. and Kawashima, K. A Measurement Study on Peer Behaviors for a Pure P2P Network. In *Advanced Information Networking and Applications, 2009. AINA '09. International Conference on*, pages 241–248, May 2009. doi: 10.1109/AINA.2009.107.

[38] Plaxton, C. G., Rajaraman, R., and Richa, A. W. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '97, pages 311–320, New York, NY, USA, 1997. ACM. ISBN 0-89791-890-8. doi: 10.1145/258492.258523. URL `http://doi.acm.org/10.1145/258492.258523`.

[39] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Schenker, S. A Scalable Content-Addressable Network. In *Proc. ACM SIGCOMM '01*, pages 161–172, 2001. ISBN 1-58113-411-8. doi: http://doi.acm.org/10.1145/383059.383072.

[40] Ratnasamy, S., Handley, M., Karp, R., and Shenker, S. Topologically-aware Overlay Construction and Server Selection. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1190–1199 vol.3, 2002. doi: 10.1109/INFCOM.2002.1019369.

[41] Rowstron, A. and Druschel, P. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Proc. IFIP/ACM Middleware 2001*, pages 329–350, 2001. ISBN 3-540-42800-3.

[42] Sánchez-Artigas, M. and García López, P. Echo: A peer-to-peer clustering framework for improving communication in dhts. *J. Parallel Distrib. Comput.*, 70(2):126–143, Feb. 2010. ISSN 0743-7315. doi: 10.1016/j.jpdc.2009.06.002. URL `http://dx.doi.org/10.1016/j.jpdc.2009.06.002`.

[43] Sanchez-Artigas, M., Garcia-Lopez, P., Gomez-Skarmeta, A. F., and Santa, J. TR-clustering: Alleviating the Impact of False Clustering on P2P Overlay Networks. *Computer Networks*, 52(17):3185 – 3204, 2008. ISSN 1389-1286. doi: http://dx.doi.org/10.1016/j.comnet.2008.08.011. URL `http://www.sciencedirect.com/science/article/pii/S1389128608002612`.

[44] Shen, H., Xu, C.-Z., and Chen, G. Cycloid: A Constant-degree and Lookup-efficient P2P Overlay Network. *Perform. Eval.*, 63(3):195–216, Mar. 2006. ISSN 0166-5316. doi: 10.1016/j.peva.2005.01.004. URL `http://dx.doi.org/10.1016/j.peva.2005.01.004`.

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

[46] Shudo, K., Tanaka, Y., and Sekiguchi, S. Overlay Weaver: An Overlay Construction Toolkit. *Comput. Commun.*, 31(2):402–412, 2008. ISSN 0140-3664. doi: http://dx.doi.org/10.1016/j.comcom.2007.08.002.

[47] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. ACM SIGCOMM '01*, pages 149–160, 2001. ISBN 1-58113-411-8. doi: http://doi.acm.org/10.1145/383059.383071.

[48] Tang, C., Buco, M. J., Chang, R. N., Dwarkadas, S., Luan, L. Z., So, E., and Ward, C. Low traffic overlay networks with large routing tables. In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '05, pages 14–25, New York, NY, USA, 2005. ACM. ISBN 1-59593-022-1. doi: 10.1145/1064212.1064216. URL `http://doi.acm.org/10.1145/1064212.1064216`.

[49] Xu, J., Kumar, A., and Yu, X. On the Fundamental Tradeoffs Between Routing Table Size and Network Diameter in Peer-to-peer Networks. *IEEE J.Sel. A. Commun.*, 22(1):151–163, Sept. 2006. ISSN 0733-8716. doi: 10.1109/JSAC.2003.818805. URL `http://dx.doi.org/10.1109/JSAC.2003.818805`.

[50] Zegura, E., Calvert, K., and Bhattacharjee, S. How to model an internetwork. In *INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*, volume 2, pages 594–602 vol.2, Mar 1996. doi: 10.1109/INFCOM.1996.493353.

[51] Zhang, H., Goel, A., and Govindan, R. Incrementally Improving Lookup Latency in Distributed Hash Table Systems. *SIGMETRICS Perform. Eval. Rev.*, 31(1):114–125, 2003. ISSN 0163-5999. doi: http://doi.acm.org/10.1145/885651.781042.

[52] Zhang, H., Goel, A., and Govindan, R. Improving lookup latency in distributed hash table systems using random sampling. *IEEE/ACM Trans. Netw.*, 13:1121–1134, October 2005. ISSN 1063-6692. doi: http://dx.doi.org/10.1109/TNET.2005.857106.

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

[54] Zhang, Y., Lu, X., and Li, D. SKY: Efficient Peer-to-Peer Networks Based on Distributed Kautz Graphs. *Science in China Series F: Information Sciences*, 52(4):588–601, 2009. ISSN 1009-2757. doi: 10.1007/s11432-009-0016-x. URL `http://dx.doi.org/10.1007/s11432-009-0016-x`.

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

List of Publications

Reviewed International Conference Proceedings

1. **Hiroya Nagao**, Kazuyuki Shudo: "Flexible Routing Tables: Designing Routing Algorithms for Overlays Based on a Total Order on a Routing Table Set", Proc. 11th IEEE Int'l Conference on Peer-to-Peer Computing (IEEE P2P'11), pp.72-81, 2011. (Chapter 1, 4, 5)

2. **Hiroya Nagao**, Kazuyuki Shudo: "GFRT-Chord: Flexible Structured Overlay Using Node Groups", Proc. 11th IEEE Int'l Conf. on Ubiquitous Intelligence and Computing (IEEE UIC 2014), 2014. (Chapter 6)

3. **Hiroya Nagao**, Kazuyuki Shudo: "Mergeable Flexible Routing Tables: A Design Framework for Reusable Structured Overlay Algorithms", Proc. Int'l Conf. on Information Networking 2015 (ICOIN 2015), 2015. (Chapter 8)