A Method for Designing Proximity-aware Routing Algorithms for Structured Overlays

Takehiro Miyao, Hiroya Nagao, Kazuyuki Shudo
Tokyo Institute of Technology
2-12-1 Ookayama, Meguro-ku, Tokyo, JAPAN
Email: {miyao8, hiroya.nagao, shudo}@is.titech.ac.jp

***Abstract*—In many structured overlays, nodes strictly maintain their routing tables using node identifiers. However, building routing tables while taking account of the physical network is difficult. We propose proximity-aware flexible routing tables (PFRT) in this paper as a method to systematically design proximity-aware routing algorithms for structured overlays. PFRT was developed by extending the flexible routing tables (FRT) method such that network proximity is considered. Routing tables in FRT- and PFRT-based algorithms are updated according to the order of the set of all routing table patterns. FRT-based algorithms define a total order based on node identifiers, whereas PFRT-based algorithms define two total orders based on node identifiers and network proximity. Because PFRT is a simple extension of FRT, PFRT-based algorithms also have many of the advantageous features of FRT. We extended Chord to design PFRT-Chord, which is a PFRT-based algorithm. Experimental results showed that PFRT-Chord preserves the expected FRT-derived properties and it could take account of network proximity.**

I. Introduction

Today, many peer-to-peer (P2P) systems are available for use over the Internet, such as Gnutella, BitTorrent, and Skype. These P2P systems have several important characteristics, including high scalability and fault tolerance. A P2P system operates by building a virtual network, called an *overlay network*, over a physical network at the application layer. The overlay network is a directed graph in which nodes merely maintain pointers to neighbor nodes. Therefore, P2P systems are decentralized.

Although the topology of overlay networks can be arbitrary, they are classified into two types: unstructured and structured. The type of overlay network affects how a request message reaches its destination node. In unstructured overlay networks, the message is diffused like *flooding* to ensure that it arrives at its destination node. In structured overlay networks, the arrangement of nodes is systematic, and a route from a source to destination node is determined under a logical rule in order to reduce route lengths and traffic volume.

A distributed hash table (DHT) is a P2P technology that enables management of an associative array by numerous computers. Many routing algorithms for structured overlays have been proposed for DHTs [1]–[6].

This work was supported by MEXT KAKENHI (24650025), and "New generation network R&D program for innovative network virtualization platform and its applications", the Commissioned Research of National Institute of Information and Communications Technology (NICT).

Often routing algorithms do not consider the physical network. However, nodes are located worldwide in real P2P systems, and so their network proximity—measured in terms of communication latency, bandwidth, jitter, and so on—is not uniform. Therefore, the overlay network is built as a weighted graph in which each weight represents the network proximity between two nodes.

Rules in structured overlays usually employ node identifiers. Since these identifiers take random values, they determine the route from a source to destination node without considering network proximity. Consequently, a chosen route typically includes many edges with high weights. In contrast, if the rule utilizes not only the node identifiers but also network proximities, the routing algorithm can select routes in which the weights of the edges are low.

There are three policies to consider network proximity in structured overlays [7]: proximity neighbor selection (PNS) [8]–[11], proximity route selection (PRS) [12], [13], and proximity identifier selection (PIS) [9], [11]. In PNS, each node selects its neighbors based on network proximity. In PRS, the selection of an edge for a route from a source to destination node is dependent on network proximity. In PIS, a node identifier is determined based on network proximity. These methods are, in general, independent of one another; that is, a structured overlay based on one method can also consider network proximity by another method [9], [11].

Various proximity-aware routing algorithms have been proposed for structured overlays [8]–[13]. However, they are so ad-hoc extensions that each of them depends on specific routing algorithms. An extension targets only one or a few algorithms. In LPRS-Chord [8], which is one of proximity-aware routing algorithms, LPRS is nearly independent from a base algorithm (e.g., Chord [1]), but it needs modification to be applied to Pastry [4] and Tapestry [6]. Thus we propose Proximity-aware Flexible Routing Tables (PFRT), a methodology to design proximity-aware routing algorithms, that introduces network proximity into Flexible Routing Tables (FRT) [14]. It does PNS.

FRT is a methodology to design a routing algorithm for a structured overlay. An FRT-based algorithm such as FRT-Chord [14], FRT-2-Chord [15] and one based on Kademlia [2] inherits the identifier spaces of its ancestor (e.g., Chord [1]) and defines a total order $\leq_{\mathrm{ID}}$ on the set of all routing table patterns based on node identifiers. And it works along the procedure defined by FRT.

To define a PFRT-based algorithm, algorithm designers define two total orders on the the set of all routing table patterns: an *identifier-based order* $\leq_{\mathrm{ID}}$, and a *proximity-based order* $\leq_{\mathrm{PR}}$. PFRT then maintains routing tables according to these orders. Therefore, PFRT facilitates designers to systematically design proximity-aware routing algorithms.

Because PFRT is a simple extension of FRT, it keeps benefits of FRT such as supporting various identifier spaces, compatibility with other extensions. Existing routing algorithms and extensions can be redesigned based on FRT and work in consistent with PFRT. The extension for network proximity by PFRT is independent from properties of the existing algorithms such as an identifier space because the two total orders are orthogonal. For example, a PFRT-based algorithm adopting Kademlia's identifier space can incorporate with an extension that takes account of node's lifetime. Another benefit of FRT that PFRT inherits is broadened application domains.

To this end, we have designed the PFRT-based algorithm, PFRT-Chord, as an extension of Chord. Since PFRT-based algorithms can consider several metrics for network proximity, we select communication latency to measure network proximity in this work. Our simulation results show that the average routing latency in PFRT-Chord is less than that in not only Chord but also LPRS-Chord, another proximity-aware routing algorithm. In addition, our results show that PFRT-Chord inherits the beneficial features of FRT.

II. RELATED WORK

A. Chord

Chord [1] is one of the most well-known DHT algorithms. DHT manages an associative array by using a large number of computers. A datum consisting of a key and value pair is stored in a computer selected by a routing algorithm. The computer is then called the *responsible node* of the datum. When putting or getting data, a request message is sent to the responsible node of each datum via the other nodes.

Data placement in Chord is managed by *consistent hashing* (Fig. 1) [16], which assigns an m-bit identifier to a node by hashing the node's IP address and to a datum by hashing the datum's key. The identifier space is like a ring. The identifier distance $\mathrm{d}(x, y)$ from identifiers x and y is the clockwise distance in the ring:

$$\mathrm{d}(x,y) = \begin{cases} y - x, & x < y, \\ y - x + 2^m, & y \leq x. \end{cases} \tag{1}$$

The first node that an identifier reaches while progressing in a clockwise direction is called the *successor node*, and the responsible node of a datum is the successor node of the datum's identifier.

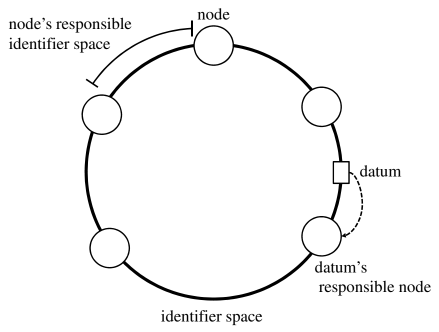


Fig. 1. Consistent hashing.

To build a structured overlay using Chord, each node manages the set of its neighbor nodes in the overlay network. This set is called a *routing table*. Each entry in a routing table holds a node's identifier and its IP address, but an entry is sometimes expressed as the node which is pointed to by the entry in this paper.

Routing tables in Chord consist of three parts: a successor list, predecessor, and finger table. The successor list of a node contains a certain number u closest nodes from the node in the identifier space. The successor list is used for reaching destination nodes. The predecessor contains the farthest node from the node in the identifier space. The finger table is used to decrease route lengths. The ith entry $(i = 0, 1, 2, \ldots m-1)$ of the finger table for a node n contains the successor node of the identifier $n.\mathrm{id}+2^i$. Thus, if N is the number of nodes, then the number of entries in a finger table is about $\log N$. Chord strictly manages these three tables based on node identifiers, and so route lengths are $O(\log N)$-hops.

When putting or getting a datum, a request message arrives at the destination node by *greedy routing*, which selects a forwarding node from the routing table as follows. The message is repeatedly forwarded from each node to the closest neighbor to the datum's identifier in the node's routing table. Once the message arrives at the predecessor p of the destination node, the message is forwarded to the destination node by using p's successor list.

Routing algorithms that use *iterative* or *recursive* routing can be implemented. In iterative routing, a source node sends a message to all forwarding nodes. In recursive routing, a forwarding node sends a message to the next forwarding node. Although iterative routing detects failure in a forwarding node, recursive routing requires only about half of the routing latency of iterative routing because there is no need to wait for a reply message.

Nodes often join and leave P2P networks, and so each node must meticulously maintain its routing table through a *stabilization protocol*. A stabilization protocol consists of four functions: *join*, *stabilize*, *notify*, and *fix_fingers*. When a node is added in Chord, it executes a *join* function to search for its successors. All nodes periodically execute *stabilize* functions to search for their current successors. When a node finds a new successor, the node executes a *notify* function to instruct the successor to update its predecessor. All nodes periodically execute a *fix_fingers* function to search for the current rth entry in its finger table (where r is a random number).

B. Proximity-aware Routing Algorithms

Various proximity-aware routing algorithms, such as LPRS-Chord [8], AChord [9], Quasi-Chord [10], have been proposed for structured overlays. However, they are so ad-hoc extensions that each of them depends on specific routing algorithms. An extension targets only one or a few algorithms. In LPRS-Chord, LPRS [8] is nearly independent from a base algorithm (e.g., Chord), but it needs modification to be applied to Pastry [4] and Tapestry [6]. Besides, it is difficult to introduce another extension considering other view points such as node lifetime into them. In contrast to them, PFRT is orthogonal to base algorithms and other extensions.

LPRS-Chord is a proximity-aware routing algorithm based on PNS, and is an extension of Chord. Therefore, data placement in LPRS-Chord is managed by consistent hashing, and it employs a greedy routing policy. The main difference between LPRS-Chord and Chord is the method of building routing tables. Routing tables in LPRS-Chord are maintained based on network proximity.

LPRS-Chord utilizes the same successor list and predecessor as used by Chord. However, the ith entry in the finger table of a node n is the node in $[n + 2^i, n + 2^{i+1})$ such that the communication latency between the node and n is minimal. Therefore, the number of entries in a routing table in LPRS-Chord equals that in Chord.

To search for the ith entry, n randomly selects a few identifiers in $[n + 2^i, n + 2^{i+1})$ and measures the communication latencies between itself and the successors of these identifiers. Because the nodes in Chord periodically execute the *fix_fingers* function, whereas the nodes in LPRS-Chord measure communication latencies for only a few nodes, searching for the ith entry in LPRS-Chord requires slightly heavier traffic than that in Chord.

C. FRT-Chord

FRT [14] is a method of designing routing algorithms for structured overlays. FRT-based algorithms inherit the identifier spaces and distances utilized by existing routing algorithms, as well as gaining the advantageous features of FRT, compatibility with other extensions and broadened application domains.

FRT-Chord [14] is an FRT-based algorithm based on Chord, and thus consistent hashing and greedy routing are employed. However, a node n in FRT-Chord maintains only a single routing table E containing entries $e_i \quad (i = 1, 2, \dots |E|)$, which are sorted in ascending order according to the identifier distances from n $(i < j \Rightarrow \mathrm{d}(n, e_i) < \mathrm{d}(n, e_j))$. Thus, $\{e_i\}_{i=1,2,\dots u}$ is the successor list of n, where e_1 is n's successor and u is the capacity of successor list, and $e_{|\mathrm{E}|}$ is its predecessor. Since routing tables in FRT-based algorithms are maintained by using a total order $\leq_{\mathrm{ID}}$ on the set of all routing table patterns, designers need to define $\leq_{\mathrm{ID}}$.

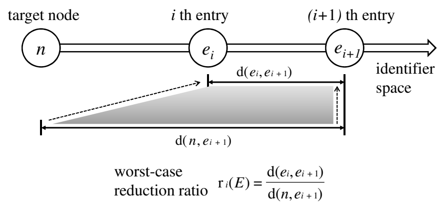

target node
i th entry
$(i+1)$ th entry
n
e_i
e_{i+1}
identifier space
$\mathrm{d}(e_i, e_{i+1})$
$\mathrm{d}(n, e_{i+1})$
worst-case reduction ratio
$\mathrm{r}_i(E) = \frac{\mathrm{d}(e_i, e_{i+1})}{\mathrm{d}(n, e_{i+1})}$

Fig. 2. Worst-case reduction ratio

1) Total Order on Routing Table Set: Routing tables are updated according to $\leq_{\mathrm{ID}}$, which is a total order on the set $\mathbf{E}$ of all routing table patterns. $E \leq_{\mathrm{ID}} F$ then expresses that routing table E is better than F based on the node identifiers. $\leq_{\mathrm{ID}}$ in FRT-Chord is defined as follows.

A node n calculates the worst-case reduction ratio $r_i(E)$ (Fig. 2) of a forwarding to a node e_i, other than its predecessor. A reduction ratio is the fraction that the remaining identifier distance is reduced by when n forwards a message to each entry e_i:

$$r_i(E) = \frac{\mathrm{d}(e_i, e_{i+1})}{\mathrm{d}(n, e_{i+1})} \qquad (i = 1, 2, \dots |E| - 1). \tag{2}$$

Let $\{r_{(i)}(E)\}$ be the list in which e_i are ranked in descending order by $\{r_i(E)\}$. Then,

$$E \leq_{\mathrm{ID}} F \Longleftrightarrow \{r_{(i)}(E)\} \leq_{\mathrm{dic}} \{r_{(i)}(F)\}, \tag{3}$$

where the lexicographical order $\leq_{\mathrm{dic}}$ is defined as follows:

$$\{a_i\} <_{\mathrm{dic}} \{b_i\} \Leftrightarrow a_k < b_k \qquad (k = \min\{i | a_i \neq b_i\}), \tag{4}$$

$$\{a_i\} =_{\mathrm{dic}} \{b_i\} \Leftrightarrow a_i = b_i, \tag{5}$$

$$\{a_i\} \leq_{\mathrm{dic}} \{b_i\} \Leftrightarrow (a_i <_{\mathrm{dic}} b_i) \cup (a_i =_{\mathrm{dic}} b_i). \tag{6}$$

Finally, the *best routing table* $\tilde{E}$ satisfies the following equation:

$$\tilde{E} \leq_{\mathrm{ID}} E, \qquad \forall E \in \mathbf{E}. \tag{7}$$

2) Guarantee of Reachability: These operations guarantee reachability in FRT. The *Guarantee of Reachability* function in FRT-Chord is the same as the *stabilize* function in Chord.

3) Entry Learning: This function obtains a node's information and inserts it into a routing table in FRT. A node obtains another node's information in the following situations.

- When a new node joins the network in FRT-Chord, it obtains information on its successor and the information in the successor's routing table.
- When node n communicates with node a, n obtains a's information.
- Each node periodically executes an *active learning lookups* function to obtain information on other nodes.

Active learning lookups in FRT-Chord actively acquires node information, and is similar to the *fix_fingers* function in Chord. When a node n executes an *active learning lookups* function, a lookup is performed for identifier k calculated by

$$k = n.\mathrm{id} + \mathrm{d}(n, e_1) \left(\frac{\mathrm{d}(n, e_{|E|})}{\mathrm{d}(n, e_1)} \right)^{\mathrm{rnd}}, \tag{8}$$

where rnd is a random number between 0 and 1. n then obtains the node information on k's successor.

4) Entry Filtering: If the number of entries $|E|$ in the routing table E is greater than the capacity of a routing table L, entries should be removed from E. *Entry filtering* is the function that removes an entry in FRT. The function determines the entry for removal e_{r} according to $\leq_{\mathrm{ID}}$ as follows:

$$E \setminus \{e_{\mathrm{r}}\} \leq_{\mathrm{ID}} E \setminus \{e\}, \qquad \forall e \in E. \tag{9}$$

By calculating a *canonical spacing* S_i^E for each entry e_i in E, e_{r} can be found efficiently:

$$S_i^E = \log \frac{\mathrm{d}(n, e_{i+1})}{\mathrm{d}(n, e_i)}. \tag{10}$$

If $S_{i'-1}^E + S_{i'}^E$ is the minimum value of $S_{i-1}^E + S_i^E$ for e_i, $e_{\mathrm{r}} = e_{i'}$. In FRT, any entry that must not be removed from the routing tables is called a *sticky entry*. Sticky entries in FRT-Chord consist of the successor list and predecessor entries that are required in order to reach a destination node.

The step in *Entry filtering* is as follows. Let C be the set of candidates for e_{r}.

1) Add all entries in E to C.
2) Remove the sticky entries from C.
3) When $S_{i'-1}^E + S_{i'}^E$ is found for $e_i \in C$, e_{r} equals $e_{i'}$ and is removed from E.

By sorting $\{S_{i-1}^E + S_i^E\}$ into ascending order, e_{r} can be found in $O(1)$ steps.

III. PFRT-Chord

PFRT is a method of designing proximity-aware routing algorithms for structured overlays. Because PFRT is a simple extension of FRT, PFRT-based algorithms also have many of the advantageous features of FRT, compatibility with other extensions, broadened application domains and so on.

PFRT-Chord is a PFRT-based algorithm designed by extending Chord, and therefore consistent hashing and greedy routing are employed. We designed PFRT-Chord since Chord is one of the most well-known algorithms for structured overlays.

The main difference among PFRT-Chord, FRT-Chord, and Chord is the method of building routing tables. Routing tables in FRT-Chord are maintained based on only node identifiers, whereas routing tables in PFRT-Chord are also maintained based on not only node identifiers but also network proximity. PFRT-Chord updates routing tables using various information, and so an entry e in a routing table of n is given an identifier $e.\mathrm{id}$, an address $e.\mathrm{address}$, and a metric $e.\mathrm{proximity}$ that measures the network proximity from n to e. $e.\mathrm{proximity} < f.\mathrm{proximity}$ then expresses that e has a better network proximity than the entry f. Although a PFRT-based algorithm could consider any metric for network proximity, we select communication latency in this paper.

A. Two Total Orders on Routing Table Set

PFRT-Chord defines both the identifier-based order $\leq_{\mathrm{ID}}$ and the proximity-based order $\leq_{\mathrm{PR}}$, where $\leq_{\mathrm{ID}}$ is the same as in FRT-Chord, in order to update routing tables based on identifier and network proximity.

$E \leq_{\mathrm{PR}} F$ indicates that routing table E is better than F based on network proximity. If f is an entry in the routing table F, then we define $\leq_{\mathrm{PR}}$ as follows:

$$\begin{aligned} & E \leq_{\mathrm{PR}} F \\ & \Leftrightarrow \frac{1}{|E|} \sum_{e \in E} e.\mathrm{proximity} \leq \frac{1}{|F|} \sum_{f \in F} f.\mathrm{proximity}. \end{aligned} \tag{11}$$

Sticky entry is the set of entries which are necessary for reachability, stabilize, churn tolerance, and so on. Therefore, PFRT-Chord compares routing tables except sticky entries in order to find better routing table based on network proximity.

B. Guarantee of Reachability

The *Guarantee of Reachability* function in PFRT-Chord is the same as that in FRT-Chord.

C. Entry Learning

Similar to FRT-Chord, this function obtains a node's information and inserts it into a routing table. However, because an entry e in the routing table of node n has the metric $e.\mathrm{proximity}$ in PFRT, n should also measure $e.\mathrm{proximity}$. When n obtains e's information except for $e.\mathrm{proximity}$, n sets $e.\mathrm{proximity}$ to its maximum value by default.

Since communication latency is used as the metric for network proximity, a node n must measure the latency $\mathrm{l}(n, e)$ between n and an entry e. PFRT-Chord uses the round trip time (RTT) as the measure of communication latency, and n can measure or obtain the RTT $\mathrm{l}(n, e)$ under the following circumstances.

- n measures $\mathrm{l}(n, e)$ in sending a round-trip message.
- When e knows RTT $\mathrm{l}(e, n)$ and sends a message to n, $\mathrm{l}(e, n)$ is sent with the message.

In order to detect failures, messages in general routing algorithms are round-trip messages. Thus, PFRT-Chord can measure RTTs between nodes without causing heavy traffic.

The last appending entry e_{a} is recorded in order to use in *Entry filtering*.

D. Entry Filtering

If the number of entries $|E|$ in the routing table E is greater than the capacity of routing table L, entries should be removed from E. The *Entry filtering* function in PFRT removes an entry according to $\leq_{\mathrm{ID}}$ and $\leq_{\mathrm{PR}}$.

Let E_{next} be the next routing table, which is the routing table after deleting the entry for removal e_{r} and let E_{prev} be the previous routing table, which is the routing table before adding e_{a}:

$$E_{\mathrm{prev}} = E \setminus \{e_{\mathrm{a}}\}, \tag{12}$$

$$E_{\mathrm{next}} = E \setminus \{e_r\}. \tag{13}$$

Entry filtering provides filters, a *proximity filter* and an *identifier filter*, based on these orders that narrow down the candidate set for E_{next}, and nodes in PFRT-Chord can select E_{next} which is better than E_{prev} when e_{a} does not belong to the sticky entry, which is the set of entries which are necessary for reachability, stabilize, churn tolerance, and so on.

e_{r} can be any entry. Let $\mathcal{C}_E$ be the set of candidates for E_{next}. Then,

$$\mathcal{C}_E = \{E \setminus \{e\} | e \in E\}. \tag{14}$$

Because (12), $e_{\mathrm{a}} \in E$ and (14) hold,

$$E_{\mathrm{prev}} \in \mathcal{C}_E. \tag{15}$$

The proximity filter is the first filter in *Entry filtering* and is based on $\leq_{\mathrm{PR}}$. Any routing table that is better, based on $\leq_{\mathrm{PR}}$, than E_{prev} passes through the proximity filter. Therefore, $\mathcal{C}_E$ is narrowed down to the proximity-aware candidate set $\mathcal{C}_E^{\mathrm{PR}}$ through

$$\mathcal{C}_E^{\mathrm{PR}} = \{E | E \leq_{\mathrm{PR}} E_{\mathrm{prev}}, E \in \mathcal{C}_E\}. \tag{16}$$

Because (15) and (16) holds,

$$E_{\mathrm{prev}} \in \mathcal{C}_E^{\mathrm{PR}} \tag{17}$$

holds. Because $\mathcal{C}_E^{\mathrm{PR}}$ is a candidate set for E_{next},

$$E_{\mathrm{next}} \in \mathcal{C}_E^{\mathrm{PR}} \tag{18}$$

holds. Because (17) and (18) hold, E_{next} is better than E_{prev} in terms of network proximity or E_{next} and E_{prev} are the same:

$$E_{\mathrm{next}} \leq_{\mathrm{PR}} E_{\mathrm{prev}}. \tag{19}$$

The identifier filter is the second filter in *Entry filtering* and is based on $\leq_{\mathrm{ID}}$. By utilizing the identifier filter, E_{next} is selected as the best routing table from $\mathcal{C}_E^{\mathrm{PR}}$ through

$$E_{\mathrm{next}} \leq_{\mathrm{ID}} E, \qquad \forall E \in \mathcal{C}_E^{\mathrm{PR}}. \tag{20}$$

Because (17) and (20) hold, E_{next} is better than E_{prev} in terms of its node identifier:

$$E_{\mathrm{next}} \leq_{\mathrm{ID}} E_{\mathrm{prev}}. \tag{21}$$

By defining $\leq_{\mathrm{ID}}$ and $\leq_{\mathrm{PR}}$, algorithm designers can systematically design PFRT-based algorithms where routing tables are updated to E_{next} ((19) and (21) hold) (Fig. 3).

The step in *Entry filtering* is as follows.Let C be a candidate set for e_{r}, let e_{t} be a threshold entry of the network proximity metric, and let u be the capacity of a successor list.

1) Add all entries in E to C.
2) Remove sticky entries from C.
3) Remove $\{e \in C | e.\mathrm{proximity} < e_{\mathrm{a}}.\mathrm{proximity}\}$ from C.
4) When $S_{i'-1}^E + S_{i'}^E$ is found for $e_i \in C$, e_{r} equals $e_{i'}$ and is removed from E.

Steps 1, 2, and 4 are the same as those in FRT-Chord. Thus, by adding only step 3, FRT-Chord is extended to PFRT-Chord. The proximity filter is passed in step 3, and the identifier filter is passed in step 4.

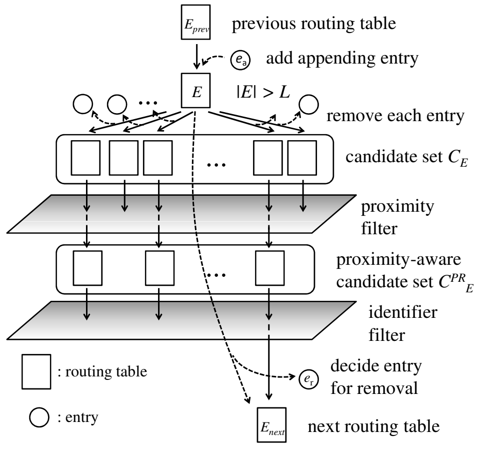


Fig. 3. *Entry filtering* in PFRT-Chord

By sorting $\{S_{i-1}^E + S_i^E\}$ into ascending order, e_{r} can be found in $O(L)$ steps. In contrast, finding e_{r} in FRT-Chord requires $O(1)$ steps. However, the computations to determine e_{r} are performed by each computer. Therefore, the time required to find e_{r} is sufficiently less than that to forward a message.

Because PFRT-based algorithms are based on PNS, a node's routing table contains only those entries with a high value of the network proximity metric. Therefore, the node directly forwards messages to the nodes listed in its routing table, and so recursive routing is more suitable for PFRT-based algorithms than iterative routing.

Nodes often join and leave P2P networks so routing algorithms should have churn tolerance. Therefore, PFRT-Chord has as much churn tolerance as Chord because routing tables in PFRT-Chord necessarily contain the successor list and predecessor like Chord.

IV. Evaluation

In this section, we evaluate PFRT-Chord through a simulation. We implemented PFRT-Chord on Overlay Weaver [17], [18], which is an overlay construction toolkit, and performed experiments on the following machine.

- Simulator: Overlay Weaver 0.10.1
- Operating system: Linux 2.6.35.10-74.fc14.x86_64
- Central processing unit: Intel Xeon E5620 (2.40 GHz)
- Memory: 32 GB
- Java virtual machine: Java SE 6 Update 22

Communication latencies between any two nodes were calculated by using a topology which simulates a physical network by using a transit-stub (TS) model [19]. TS models have two types of nodes: transit and stub. In our simulation, 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. The maximum communication latency is 870 ms and the average is 420 ms. The simulated communication latency is much higher than communication latency over real Internet, but we can evaluate simulation results by not value but rate correctly.

A. Comparison between PFRT-Chord and Existing Routing Algorithms

We measured the routing latency—the time required per request—in Chord, LPRS-Chord, FRT-Chord, and PFRT-Chord. Parameters values in the simulations were set as follows.

- Number of nodes: 10 000
- Capacity of a successor list: 4
- Capacity of a routing table in PFRT-Chord: 16
- Number of requests per node: 300
- Routing type: recursive

Because the number of nodes is 10 000, the number of entries in a finger table in Chord and LPRS-Chord is about 13; that is, the number of entries in all three parts of the routing tables in Chord and LPRS-Chord is about 18. Therefore, the capacity of a routing table in PFRT-Chord is set at 16 so that PFRT-Chord does not have an advantage in the number of entries that can be held.

Fig. 4(a) shows the averages and 99th percentiles of the routing latencies in Chord, LPRS-Chord, FRT-Chord, and PFRT-Chord. The average routing latency is 2 081 ms in PFRT-Chord and 2 756 ms in Chord. The average routing latency in PFRT-Chord is thus about 24.5% less than that in Chord. In addition, the 99th percentile of the routing latency in PFRT-Chord is about 22.1% less than that in Chord. Consequently, PFRT-Chord is verified as an appropriate proximity-aware routing algorithm.

The average routing latency in FRT-Chord is 2 587 ms; that is, the average routing latency in PFRT-Chord is about 19.6% less than that in FRT-Chord. Furthermore, the 99th percentile of the routing latency in PFRT-Chord is about 15.0% less than that in FRT-Chord. Therefore, extending FRT to PFRT in order to consider network proximity improves its performance.

The average routing latency in LPRS-Chord is 2 676 ms; that is, the average routing latency in PFRT-Chord is about 22.2% less than that in LPRS-Chord. Additionally, the 99th percentile of the routing latency in LPRS-Chord is approximately equal to that in Chord. PFRT-Chord is thus a better proximity-aware routing algorithm than LPRS-Chord. Furthermore, the capacity of routing tables in PFRT-Chord can be set to an arbitrary value, and so routing latencies can be decreased further.

Although FRT-Chord does not consider network proximity, the average and 99th percentiles of the routing latency in FRT-Chord are only about 3.3% and 8.7% less than those in LPRS-Chord, respectively. This indicates that route lengths in LPRS-Chord are longer than those in FRT-Chord. Thus, we measured the route lengths in the four algorithms.

The measurements results are given in Fig. 4(b), which shows the averages and 99th percentiles of the route lengths. The average route length in LPRS-Chord is longer than in PFRT-Chord, which may explain why the routing latencies are greater in LPRS-Chord than in PFRT-Chord. To test this hypothesis, we also calculated the average routing latency per hop for each algorithm (Fig. 4(c)). The average routing latency per hop in PFRT-Chord is about 16.0% less than that in LPRS-Chord. Hence, PFRT-Chord is superior to LPRS-Chord at utilizing network proximities.

B. Independence of the Number of Nodes

We finally examined whether PFRT-Chord retains the beneficial features of FRT; specifically, broadened application domains. Since broadening of application domains indicates that the routing algorithm is independent of the number of nodes, and flexibility means that the capacity of each routing table can be changed, we measured the average route length and the average routing latency in PFRT-Chord under the following parameter values.

- Number of nodes: 100, 1000, and 10 000.
- Capacity of each routing table: 20–160.

Figs. 5(a) and 5(b) show the simulation results for the average route length and the average routing latency. As the capacity of the routing tables increases, the average route length and average routing latency both decrease. Therefore, PFRT-Chord is flexible.

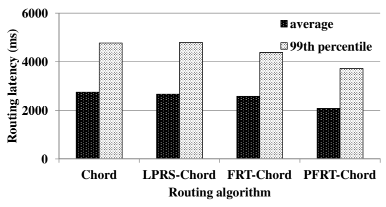
As the number of nodes increases, the average route length and average routing latency increase. Although the number of nodes increases by 10- and 100-fold, the average route length increases by only about 1.66- and 2.34-fold, respectively, and the average routing latency increases by only about 1.59- and 1.94-fold, respectively. In addition, when the number of nodes N is less than the capacity of each routing table L, for example, $N = 100$ and $L = 160$, the average route length is 2 (incidentally, the average route length in LPRS-Chord is 3.28). Hence, each routing table contains the information on all of the nodes in the network, and a source node forwards a message to the predecessor of the destination node in the first hop. PFRT-Chord thus broadens application domains.

V. Conclusion

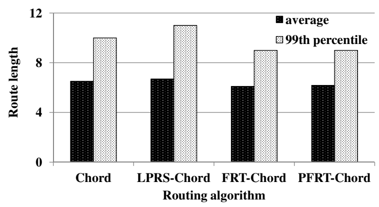
We proposed Proximity-aware Flexible Routing Tables (PFRT), a methodology to design proximity-aware routing algorithms for structured overlays. A PFRT-based algorithm defines two total orders on the set of all routing table patterns: an identifier-based order $\leq_{\mathrm{ID}}$ and a proximity-based order $\leq_{\mathrm{PR}}$. Therefore, PFRT facilitates algorithm designers to systematically design proximity-aware routing algorithms.

To this end, we designed a PFRT-based algorithm, PFRT-Chord, which employs the same data placement method and routing policy as in Chord. The main difference between PFRT-Chord and Chord is the method of building routing tables. To build routing tables in PFRT-Chord, we must define $\leq_{\mathrm{ID}}$ and $\leq_{\mathrm{PR}}$, and so routing tables in PFRT-Chord are maintained according to not only node identifiers but also network proximities in the physical network.

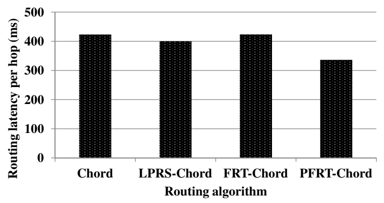
We implemented and simulated PFRT-Chord in a set of experiments. Routing latencies in PFRT-Chord are less than those in not only Chord but also LPRS-Chord, which is an existing proximity-aware routing algorithm. This performance



(a) routing latencies

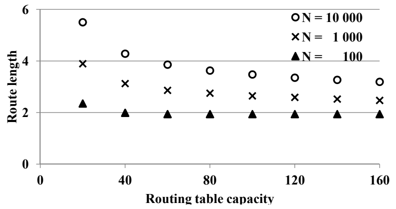


(b) route lengths

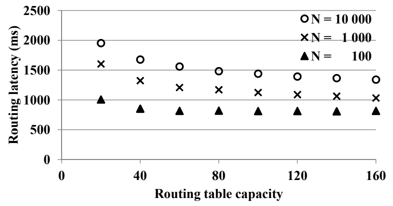


(c) routing latency per hop

Fig. 4. Routing results in each algorithm



(a) route lengths



(b) routing latencies

Fig. 5. Correlation between routing table capacity and routing results

improvement can be achieved even if the number of entries in PFRT-Chord routing tables is less than that in LPRS-Chord. Therefore, our experiments showed that PFRT-Chord is verified as an appropriate proximity-aware routing algorithm for structured overlays.

In addition, we examined whether PFRT-Chord retains the beneficial features of FRT. Our simulations showed that PFRT-Chord is independent of the number of nodes and that the routing table capacity can be changed dynamically.

Future work includes design of extensions of FRT considering other real-world metrics such as node lifetime. FRT can incorporate with those metrics in the same manner as for PFRT. Combinations of PFRT and other extensions should be examined. GFRT [14], an extension of FRT takes account of groups of nodes and reduces communication between nodes. It should work with PFRT.

References

[1] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup protocol for internet applications,” *IEEE/ACM Trans. Networking,*, vol. 11, no. 1, pp. 17–32, 2003.

[2] P. Maymounkov and D. Mazieres, “Kademlia: A peer-to-peer information systems based on the XOR metric,” in *Peer-to-Peer Systems (Proc. IPTPS ’ 02)*, MA, USA, 2002, pp. 53–65.

[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A scalable content-addressable network,” *ACM SIGCOMM Comput. Comm. Rev.*, vol. 31, no. 4, pp. 161–172, 2001.

[4] A. Rowstron and P. Druschel, “Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems,” in *Proc. IFIP/ACM Middleware 2001*, Heidelberg, Germany, 2001, pp. 329–350.

[5] M. Kaashoek and D. Karger, “Koorde: A simple degree-optimal distributed hash table,” Berkeley, USA, 2003, pp. 98–107.

[6] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz, “Tapestry: A resilient global-scale overlay for service deployment,” *Selected Areas in Comm., IEEE Journal on*, vol. 22, no. 1, pp. 41–53, 2004.

[7] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, “The impact of DHT routing geometry on resilience and proximity,” in *Proc. ACM SIGCOMM ’03*, Karlsruhe, Germany, 2003, pp. 381–394.

[8] H. Zhang, A. Goel, and R. Govindan, “Incrementally improving lookup latency in distributed hash table systems,” in *Proc. ACM SIGMETRICS ’03*, CA, USA, 2003, 114–125.

[9] L. H. Dao and J. Kim, “AChord: Topology-aware Chord in anycast-enabled networks,” in *Proc. ICHIT ’06*, Cheju Island, Korea, 2006, pp. 334–341.

[10] S. Mingsong and Z. Zhongqiu, “Quasi-Chord: Physical topology aware structured P2P network,” in *Proc. 11th Joint Conf. Information Sciences*, Shenzhen, China, 2008.

[11] N. B. Dang, S. T. VU, and H. S. Nguyen, “Building a low-latency, proximity-aware DHT-based P2P network,” in *Proc. KSE ’09*, Hanoi, Vietnam, 2009, pp. 195–200.

[12] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, “Wide-area cooperative storage with CFS,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 202–215, 2001.

[13] J. Song, S. Park, and J. Yang, “An adaptive proximity route selection scheme in DHT-based peer to peer systems,” in *Parallel and Distributed Computing: Applications and Technologies (Proc. PDCAT ’04)*, Singapore, 2005, pp. 143–157.

[14] H. Nagao and K. Shudo, “Flexible routing tables: Designing routing algorithms for overlays based on a total order on a routing table set,” in *Proc. IEEE P2P ’11*, Kyoto, Japan, 2011, pp. 72–81.

[15] Y. Ando, H. Nagao, T. Miyao, and K. Shudo, “FRT-2-Chord: A DHT Supporting Seamless Transition between One-hop and Multi-hop with Symmetric Routing Tables,” in *Proc. SACSIS 2012*, Kobe, Japan, 2012, pp. 210–218.

[16] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, “Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web,” in *Proc. 29th Ann. ACM Symp. Theory Computing*, Texas, USA, 1997, pp. 654–663.

[17] K. Shudo, “Overlay Weaver: An overlay construction toolkit,” Available: http://overlayweaver.sourceforge.net/, 2006.

[18] K. Shudo, Y. Tanaka, and S. Sekiguchi, “Overlay Weaver: An overlay construction toolkit,” *Comput. Comm.*, vol. 31, no. 2, pp. 402–412, 2008.

[19] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, “How to model an internetwork,” in *Proc. IEEE INFOCOM ’96*, San Francisco, USA, 1996, pp. 592–602.