# Routing Table Construction Method Solely Based on Query Flows for Structured Overlays

Yasuhiro Ando, Hiroya Nagao, Takehiro Miyao and Kazuyuki Shudo
Tokyo Institute of Technology

***Abstract*—In structured overlays, nodes forward a query hop by hop to deliver it to the responsible node for the query. Each node maintains its routing table and determines the next hop by referring to the routing table. Each node has its node identifier and determines which other nodes to be on its routing table based on node distance, that is defined by difference of node identifiers or the number of nodes between two nodes in identifier order. In existing structured overlays, routing table construction and maintenance based on node distance enable efficient lookup, that is a small number of hop counts to the responsible node.**

**We found out that efficient lookup does not require node distance in routing table construction and maintenance. As an example, this paper presents Flow-based Flexible Routing Tables (FFRT), a routing table construction method solely based on query flows. In an FFRT-based overlay, a node calculates query flows, that is the amount of queries forwarded to each node on its routing table. And the node maintains its routing table toward a state in which all nodes on the table have equal query flows. FFRT also provides such a practical merit as it performs efficient lookups though node and queries' target identifiers are distributed nonuniformly. The merit enables range query support.**

## I. Introduction

A structured overlay constructs an application-level network and enables higher level services such as Distributed Hash Table and message delivery. Nodes in an overlay forward a query hop by hop to deliver it to the responsible node for the query. Each node in an overlay maintains its routing table and determines the next hop by referring to the routing table.

In existing structured overlays, routing tables are constructed and maintained based on node distance. Each node has its own node identifier (ID), that is a number or a set of numbers. Node distance is defined by difference of the node IDs in Chord [1], XOR of the node IDs in Kademlia [2], or the number of other nodes between the nodes in ID order in Chord$^{\#}$ [3]. A node determines which other nodes to be on its routing table based on the node distances to them from itself. By choosing the nodes along the rules prescribed by a structured overlay, responsible node lookup is efficiently performed, in other words, a query reaches the responsible node in a small number of hops such as $\mathrm{O}(\log N)$ hops with $N$ nodes.

We found out that efficient lookup does not require node distance in routing table construction and maintenance. This paper presents Flow-based Flexible Routing Tables (FFRT), a routing table construction method and FFRT-Chord, an FFRT-based structured overlay. FFRT-based overlays maintain routing tables solely based on query flows without considering node distance. A query flow is defined as the amount of queries forwarded between two nodes. A node in an FFRT-based overlay calculates query flows for all nodes on its routing table. The node refines its routing table toward a state in which all nodes on the table have equal query flows. FFRT is an application of Flexible Routing Tables (FRT) [4]. FFRT defines a total order $\leq_{\mathrm{FL}}$ based on query flows instead of a total order $\leq_{\mathrm{ID}}$ that FRT-based overlays define based on node IDs. FFRT inherits features of FRT including extensibility and arbitrary routing table size.

FFRT-Chord is an FFRT-based overlay, which adopts Chord-style ID space and distance. The ID space and distance are just used to determine the responsible node for the target ID of a query, and not used for routing table maintenance. Experimental results showed that FFRT-Chord achieved higher efficiency with nonuniform target ID distributions and nonuniform node ID distributions. Its efficiency is comparable to existing structured overlays even with uniform node ID and target ID distributions. Here efficiency means smaller number of hops, in other words, shorter path length.

This paper is organized as follows. Section II presents prior knowledge by introducing related work. Section III describes FFRT and FFRT-Chord. Section IV demonstrates the properties of FFRT-Chord including efficient lookups with nonuniform target ID and node ID distributions. In Section V, we summarize our contributions.

## II. Related Work

### A. Chord

Distributed Hash Table (DHT) is a higher level service constructed on a structured overlay. It stores key-value pairs on decentralized nodes. Both nodes and data have their IDs. The responsible node, a node that holds a key-value pair, is determined according to the node and data IDs.

Chord [1] is a DHT, in which IDs are represented as a ring of numbers from 0 to $2^m - 1$. A node ID is $m$ bits long and determined by hashing the node's IP address. A data ID is generated from a key by a hashing function. A data item is stored at the responsible node, whose ID is equal to the data ID or immediately follows it.

ID distance plays an important role in Chord. A routing table is maintained based on it and then a query is routed according to it. ID distance from $x$ to $y$, $\mathrm{d}(x, y)$ is defined as follows.

*Definition 1:*

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

Chord achieves $\mathrm{O}(\log N)$-hop lookup performance with $N$ nodes.

### B. Flexible Routing Tables

Flexible Routing Tables (FRT) [4] is a method of designing routing algorithms for structured overlays. An algorithm

978-1-4799-6201-3/14/$31.00 ©2014 IEEE

designer can design a structured overlay by defining a total order $\leq_{\mathrm{ID}}$ on the set of all patterns of a routing table and sticky entries. FRT does not suppose a specific ID space and the designer chooses an ID space, for example a Chord-style ring or a Kademlia-style XOR space. A node in an FRT-based structured overlay repeatedly refines its routing table according the order by entry learning and entry filtering.

*1) FRT-Chord:* FRT-Chord [4] is an FRT-based structured overlay. ID space and thus responsible nodes of FRT-Chord are the same as Chord. In this section, we introduce FRT by describing how FRT-Chord is designed based on FRT.

Each node maintains a single unified routing table $E$ unlike Chord, that has three parts as a successor list, a predecessor and a finger table. A node can dynamically change the size of its routing table because FRT-Chord inherits the property from FRT. A routing table $E$ at a node $s$ is a set of entries $\{e_i\}$ which are ordered clockwise from $s$. Each entry $e_i$ consists of a node ID $e_i$.id and an IP address $e_i$.address. Note that $e_i$ is referred to as $e_i$.id afterward. By this definition $e_1$ and $e_{|E|}$ correspond to a successor and a predecessor.

*2) Total order $\leq_{\mathrm{ID}}$ on Routing Table Set:* FRT-Chord defines a total order $\leq_{\mathrm{ID}}$ based on node ID distance. A node refines its routing table according to the total order. FRT-Chord defines *reduction ratio* of a query forwarding as $\mathrm{d}(E.\mathrm{forward}(t), t)/\mathrm{d}(s, t)$, where $E.\mathrm{forward}(t)$ is an entry on the routing table $E$, to which a query for target ID $t$ is forwarded by node $s$. A reduction ratio expresses how close to $t$ the entry $E.\mathrm{forward}(t)$ is in comparison to $s$. In case the target is in the interval between $e_i$ and $e_{i+1}$ $s$ forwards a query to $e_i$. In the case the reduction ratio takes its worst and largest value when $t = e_{i+1}$. The *worst-case reduction ratio* $r_i(E)$ of the forwarding to a node $e_i$ is defined as follows:

*Definition 2:*

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

Let $\{r_{(i)}(E)\}$ be the list of reduction ratios arranged in descending order. Based on $\{r_{(i)}(E)\}$, the total order on the set of all patterns of a routing table is defined as follows.

*Definition 3:*

$$E \leq_{\mathrm{ID}} F \Leftrightarrow \{r_{(i)}(E)\} \leq_{\mathrm{dic}} \{r_{(i)}(F)\} \quad (2)$$

where $\leq_{\mathrm{dic}}$ is the lexicographical order.

*3) Sticky Entry:* *Sticky entries* are routing table entries that are not removed by entry filtering. An algorithm designer defines sticky entries to guarantee reachability to the responsible node. In FRT-Chord, sticky entries are a successor list and a predecessor as in Chord.

*4) Entry Learning:* In FRT-based overlays, a node inserts other nodes into its routing table whenever the node is aware of them. A node notices the others passively by communications for lookups and it can also perform active learning lookups.

*5) Entry Filtering:* When the number of entries $|E|$ exceeds the size of the routing table $L$, A node removes an entry. A node refine its routing table by choosing a removed entry according to the total order $\leq_{\mathrm{ID}}$.

By calculating a *canonical spacing* $S_i^E$ for each entry $e_i$ in $E$, the removed entry can be found efficiently.

*Definition 4:*

$$S_i^E = \log \frac{\mathrm{d}(s, e_{i+1})}{\mathrm{d}(s, e_i)} \quad (3)$$

Steps in entry filtering are as follows. Let $C$ be the set of candidates for a removed entry.

1) Put all entries in $E$ into $C$
2) Remove sticky entries from $C$
3) Select the entry $e_i$ in $C$ that minimizes $S_{i-1}^E + S_i^E$.
4) Remove $e$ from $E$.

By sorting $S_{i-1}^E + S_i^E$ in ascending order, the removed entry can be found efficiently.

## III. Flow-based Flexible Routing Tables

Existing structured overlays [1]–[3] achieve efficient lookup by maintaining routing tables based on node distance. By contrast, we found out that efficient lookup does not require node distance based routing table maintenance. As an example, we present Flow-based Flexible Routing Tables (FFRT), a routing table construction method and FFRT-Chord, an FFRT-based structured overlay. Their routing table maintenance is solely based on query flows without considering node distance.

FFRT is an application of Flexible Routing Table (FRT) (Section II-B). FFRT defines a total order $\leq_{\mathrm{FL}}$ based on query flows in contrast to FRT-based overlays that define a total order based on node IDs. A query flow is defined as the amount of queries forwarded between two nodes. A node in an FFRT-based overlay calculates query flows for all entries on its routing table. The total order is defined as uniformity of query flows of all the entries. Each node refines its routing table toward a state in which all entries on the table have equal query flows. Each node counts the number of queries that it forwards to other nodes regardless of whether the query is originated by the node itself or not. A node calculates query flows based on the counts.

FFRT defines the total order though FRT itself does not define a total order. An FRT-based overlay defines a total order. An algorithm designer designs a structured overlay based on FFRT by defining ID distance and sticky entries.

FFRT inherits features of FRT including extensibility and arbitrary routing table size. Proximity-aware FRT (PFRT) [5] and Grouped FRT (GFRT) [4] are examples of extensions to FRT-based overlays. PFRT is an extension to consider network proximity and GFRT is for node groups. They and other extensions can be combined with all FRT-based and FFRT-based overlays. Arbitrary routing table size means that the sizes of routing tables are not fixed by a structured overlay. They can vary per node and change as time goes on.

FFRT is not just an example of non-node-distance-based structured overlays, but also provides a practical merit. FFRT-based overlays perform efficient lookups though node IDs and queries' target IDs are distributed nonuniformly because they maintain routing tables solely based on query flows. DHT introduces such nonuniformity. Queries' target IDs, that are data IDs in a DHT, are distributed nonuniformly when access frequencies of data are not uniform. For example, in file sharing systems, content popularity is biased. Range queries

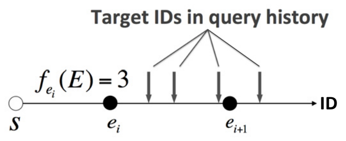

Fig. 1. An example of query flow in FFRT-Chord.

also lead to a nonuniform data distribution. Range queries require assigning continuous data IDs with continuous keys to achieve efficient lookups. Otherwise continuous keys are distributed to numerous nodes, that a range query has to involve. Such a continuous assignment leads to a nonuniform data ID distribution. With nonuniform data ID distributions, load imbalance is a problem. Virtual node is one of solutions, but there is another promising solution, that is to make a node ID distribution follow a data ID distribution. Node IDs can be adjusted in advance or by leave and join of nodes. Such adjustment leads to a nonuniform node ID distribution.

### A. Total Order $\leq_{\mathrm{FL}}$ Based on Query Flows

We define *query flow* as a basis of the definition of total order $\leq_{\mathrm{FL}}$. A node in an FFRT-based overlay keeps its query history $Q$ with the maximum size of the history $H$. The history holds the target IDs of queries that the node forwarded to other nodes, not limited to nodes currently on its routing table. Note that $q \in Q$ is referred to as a target ID $q.id$.

If an ID distance is capable of greedy routing, we can define query flow for a routing table entry. Capability of greedy routing means that the ID distance enables a node to determine uniquely the routing table entry to forward a query.

Let $E$ be the routing table of $s$, $\mathrm{d}(x, y)$ be an ID distance from ID $x$ to ID $y$ and $f_e(E)$ be the query flow for an entry $e \in E$. $f_e(E)$ is the amount of queries that $e$ is responsible for in the query history $Q$. When an entry $e \in E$ is responsible for $q \in Q$, $s$ would forward $q$ to $e$, that is, following formula holds.

$$\mathrm{d}(e, q) < \mathrm{d}(e', q),\ \forall e' \in E \setminus \{e\} \tag{4}$$

We define query flow $f_e(E)$ as follows.

*Definition 5:*

$$f_e(E) = |\{q \in Q \mid \mathrm{d}(e, q) < \mathrm{d}(e', q),\ \forall e' \in E \setminus \{e\}\}| \tag{5}$$

Fig. 1 shows an example of a query flow in FFRT-Chord, that is described in Section III-D. In the figure, $f_{e_i}(E)$ is 3 because the number of queries to be forwarded to $e_i$ is 3. Note that ID distance of FFRT-Chord is the same as Chord.

We define total order $\leq_{\mathrm{FL}}$ on query flow as follows.

*Definition 6:*

$$E \leq_{\mathrm{FL}} E' \iff \mathrm{V}(F_E) \leq \mathrm{V}(F_{E'}) \tag{6}$$

$\mathrm{V}(F_E)$ is the variance of query flows for entries in a routing table $E$. We define $\mathrm{V}(F_E)$ as follows.

$$\mathrm{V}(F_E) = \sum_{e \in E} (f_e(E) - ave)^2 \tag{7}$$

$$ave = |Q|/|E| \tag{8}$$

### B. Entry Learning

Entry learning is the same as FRT that is described in Section II-B4.

### C. Entry Filtering

When the number of entries $|E|$ exceeds the size of a routing table $L$, an FFRT-based structured overlay removes one of the entries to keep $|E| \leq L$. The removed node is selected according to the order $\leq_{\mathrm{FL}}$. This operation is called entry filtering inherited from FRT.

We define *removal preference* for an entry to determine which node to be removed. The following $v_e(E)$ means how much the variance of the routing table decreases by removing an entry $e$.

*Definition 7:*

$$v_e(E) = \mathrm{V}(F_E) - \mathrm{V}(F_{E \setminus \{e\}}) \tag{9}$$

Steps in entry filtering in FFRT are as follows. Let $C$ be the set of candidates for a removed entry.

1) Put all entries in $E$ into $C$.
2) Remove sticky entries from $C$.
3) Select the entry $e$ in $C$ that maximizes removal preference $v_e(E)$. If there are multiple such entries, select the entry $e$ of which query flow $f_e(E)$ is the lowest among them. If there are still multiple such entries, select the entry $e$ of which $\mathrm{d}(s, e)$ is largest among them.
4) Remove $e$ from $E$.

The following theorem holds. It means the routing table ascends according to the total order $\leq_{\mathrm{FL}}$ by the entry filtering.

*Theorem 1:* Let $E \setminus \{e^*\}$ be a routing table filtered by removing $e^*$. For any $e$ which is not a sticky entry,

$$E \setminus \{e^*\} \leq_{\mathrm{FL}} E \setminus \{e\}. \tag{10}$$

*Proof:* For $e \in E$,

$$\mathrm{V}(F_{E \setminus \{e\}}) = \mathrm{V}(F_E) - v_e(E) \tag{11}$$

holds. $\mathrm{V}(F_E)$ is 0 or larger. Because of them, the larger $v_e(E)$ gives the smaller $\mathrm{V}(F_{E \setminus \{e\}})$. Therefore,

$$v_{e^*}(E) \geq v_e(E) \iff \mathrm{V}(F_{E \setminus \{e^*\}}) \leq \mathrm{V}(F_{E \setminus \{e\}}) \tag{12}$$

$$\iff E \setminus \{e^*\} \leq_{\mathrm{FL}} E \setminus \{e\} \tag{13}$$

■

If a node has few queries in its query history, it may be better to select a removed node based on node distance like an FRT-based overlay. But, especially without active learning lookups, in case the routing table is filled, the query history should have queries much the same.

### D. FFRT-Chord

FFRT-Chord is an FFRT-based structured overlay. FFRT-Chord adopts the ID space and the ID distance of Chord, that is $\mathrm{d}_{\mathrm{chord}}(x, y)$ (Section II-A). The ID space and ID distance are just used to determine the responsible node and not used for routing table maintenance. Responsible nodes in FFRT-Chord are the same as Chord.

Sticky entries are the same as FRT-Chord. But, entry filtering is different from existing overlays. It is carried out according to query-flow-based total order $\leq_{\mathrm{FL}}$.

A routing table $E = \{e_i\}$ satisfies $i < j \Rightarrow \mathrm{d}_{\mathrm{chord}}(s, e_i) < \mathrm{d}_{\mathrm{chord}}(s, e_j)$. By this definition $e_1$ and $e_{|E|}$ correspond to a successor and a predecessor as in Chord.

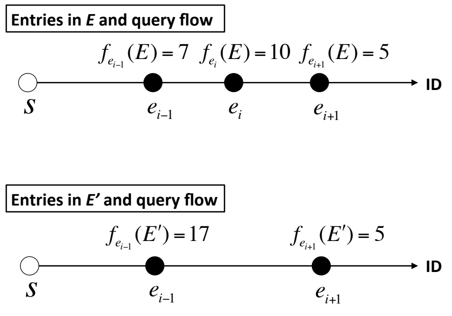
Entries in *E* and query flow
$f_{e_{i-1}}(E) = 7$ $f_{e_i}(E) = 10$ $f_{e_{i+1}}(E) = 5$
ID
$s$
$e_{i-1}$
$e_i$
$e_{i+1}$
Entries in *E′* and query flow
$f_{e_{i-1}}(E') = 17$
$f_{e_{i+1}}(E') = 5$
ID
$s$
$e_{i-1}$
$e_{i+1}$

Fig. 2. Change of query flow in FFRT-Chord

*1) Entry Filtering:* We adopt the following removal preference to simplify the calculation.

*Definition 8:*

$$v'_e(E) = \sum_{e' \in E} (f_{e'}(E) - ave)^2 - \mathrm{V}(F_{E \setminus \{e\}}) \quad (14)$$

$$ave = |Q|/L \quad (15)$$

$v'_e(E)$ is a variant of $v_e(E)$ in Definition 7 and they are not the same. The number of routing table entries to calculate $\mathrm{V}(F_E)$ is $L$, not the precise number $L + 1$. But $v'_e(E)$ can select the removed node correctly. The following theorem holds.

*Theorem 2:* Let $E \setminus \{e^*\}$ be a routing table filtered by removing $e^*$. For any $e$ which is not a sticky entry,

$$E \setminus \{e^*\} \leq_{\mathrm{FL}} E \setminus \{e\}. \quad (16)$$

*Proof:* For $e \in E$,

$$\mathrm{V}(F_{E \setminus \{e\}}) = \sum_{e' \in E} (f_{e'}(E) - ave)^2 - v_e(E) \quad (17)$$

holds. $\mathrm{V}(F_E)$ and the first term of the right side of the formula are 0 or larger. Because of them, the larger $v_e(E)$ gives the smaller $\mathrm{V}(F_{E \setminus \{e\}})$. Therefore,

$$v_{e^*}(E) \geq v_e(E) \iff \mathrm{V}(F_{E \setminus \{e^*\}}) \leq \mathrm{V}(F_{E \setminus \{e\}}) \quad (18)$$

$$\iff E \setminus \{e^*\} \leq_{\mathrm{FL}} E \setminus \{e\} \quad (19)$$

■

We can simplify $v'_{e_i}(E)$ by the fact that a node removal affects $f_e(E)$ of just one node. In Fig. 2, an entry $e_i$ is removed from a routing table $E$ and only $f_{e_{i-1}}(E)$ changes from 7 to 17.

*Theorem 3:*

$$v'_{e_i}(E) = -2 f_{e_i}(E) f_{e_{i-1}}(E) + ave^2 \quad (20)$$

*Proof:* When an entry $e_i$ is removed from a routing table $E$, only query flow $f_{e_{i-1}}(E)$ changes. Therefore,

$$f_{e_j}(E \setminus \{e_i\}) = \begin{cases} f_{e_j}(E) & (1 \leq j \leq i-2) \\ f_{e_{i-1}}(E) + f_{e_i}(E) & (j = i-1) \\ f_{e_{j+1}}(E) & (i \leq j \leq |E|-1) \end{cases}$$

By Definition 8,

$$\begin{aligned} v'_e(E) &= (f_{e_{i-1}}(E) - ave)^2 - (f_{e_{i-1}}(E) + f_{e_i}(E) - ave)^2 \\ &\quad + (f_e(E) - ave)^2 \quad (21) \\ &= -2 f_{e_i}(E) f_{e_{i-1}}(E) + ave^2. \quad (22) \end{aligned}$$

■

This $v'_e(E)$ enables efficient entry filtering as follows.

1) put all entries in $E$ into $C$.
2) Remove sticky entries from $C$.
3) Select the entry $e$ in $C$ that minimizes $f_{e_i}(E) f_{e_{i-1}}(E)$. If there are multiple such entries, select the entry $e$ of which query flow $f_e(E)$ is the lowest among them. If there are still multiple such entries, select the entry $e$ of which $\mathrm{d}(s, e)$ is largest among them.
4) Remove $e$ from $E$.

## IV. Evaluation

This section demonstrates the following properties of path length of FFRT-Chord compared with existing structured overlays.

- With uniform node ID and target ID distributions, comparable.
- With nonuniform target ID distributions, shorter.
- With nonuniform node ID distributions, shorter.

Shorter path length, in other words, smaller number of hops indicates that the structured overlay is efficient. On real networks, efficiency involves network proximity as well not only path length. Evaluation of FFRT combined with Proximity-aware FRT (PFRT) [5] is part of future work.

In the experiments, we compare FFRT-Chord with FRT-Chord described in Section II-B1. FRT-Chord shows shorter path length than Chord with the moderate routing table sizes, that is 20 or larger [4]. Therefore, if it is shorter than FRT-Chord, it is shorter than Chord. FRT-Chord and Chord maintains routing tables based on node ID difference and they do not adapt well to nonuniform node ID distributions. Future work includes comparison with overlays for nonuniform node ID distributions such as FRT-Chord# and Chord#.

We implemented FFRT-Chord on Overlay Weaver [6], [7], an overlay construction toolkit and conducted the experiments with the toolkit. The routing table size $L$ is 20, the successor list size is 4 and the history size $H$ is 500 in all the experiments. The lookup style is iterative in all the experiments but it does not affect the results.

### A. Uniform Distributions

After $N$ nodes join an overlay, each node sends a query 200 times in random order where each query is sent to a randomly chosen target ID ($N = 10^2$, $10^3$, and $10^4$).

Fig. 3 shows the average and the 99th percentiles of path lengths for the last $N$ queries. The average path lengths in FFRT-Chord were comparable to ones in FRT-Chord. In cases of $N = 10^2$ and $10^3$ FFRT-Chord showed shorter average path lengths. In case of $N = 10^4$ FFRT-Chord showed a longer average path length, but the rate of the increase was 0.7%, that is small. The 99th percentiles of the path lengths in FFRT-Chord were equal to ones in FRT-Chord.

These results indicate that FFRT-Chord is comparable to FRT-Chord in path length with uniform node ID and target ID distributions.

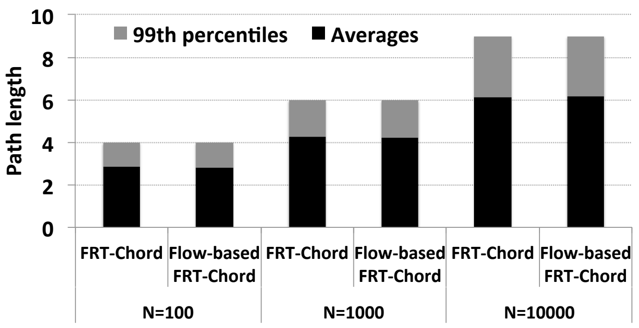

Fig. 3. Path lengths with uniform node ID and target ID distributions.

## B. Nonuniform Target ID Distributions

After 10000 nodes join an overlay, each node sends a query 200 times in random order where each query is sent to target IDs distributed according to Zipf distributions ($\alpha = 0.7$, 0.9, and 1.1).

Fig. 4 shows the average and the 99th percentiles of path lengths for the last 10000 queries. In cases of $\alpha = 0.7$ and 0.9 FFRT-Chord showed shorter path lengths. In case of $\alpha = 1.1$ the average path length in FFRT-Chord was 22% shorter than one in FRT-Chord, but the 99th percentile of the path lengths in FFRT-Chord was 1 longer than one in FRT-Chord.

These results indicate that FFRT-Chord adapts to nonuniform target ID distributions and it achieves efficient lookups with such applications that involve nonuniform target ID distributions, for example range queries. With heavily biased target ID distributions, path lengths to a dense area in the ID space tend to be short but path lengths to a sparse area tend to be long. As a result, it achieves shorter average path lengths.

## C. Nonuniform Node ID Distributions

IDs distributed according to Zipf distributions ($\alpha = 0.7$, 0.9, and 1.1) are assigned to 10000 nodes. After the 10000 nodes join an overlay, each node sends a query 200 times in random order where each query is sent to target IDs distributed according to the same distribution as node IDs. This setting represents a situation in which load imbalance (Section III) has been mitigated by making the node ID distribution follow the nonuniform target ID distribution.

Fig. 5 shows the average and the 99th percentiles of path lengths for the last 10000 queries. FFRT-Chord showed shorter path lengths than FRT-Chord in all the cases. FFRT-Chord also showed shorter path lengths with the nonuniform node ID distributions than with the uniform node ID distribution. By contrast, FRT-Chord showed longer path lengths with nonuniform node ID distributions except the average path length with $\alpha = 1.1$, that is a relatively heavily biased distribution.

These results indicate that FFRT-Chord adapts to nonuniform node ID distributions and it achieves efficient lookups with node ID adjustment for load balance.

## V. Conclusion

Existing structured overlays have to consider node distance when constructing and maintaining routing tables. Otherwise they cannot perform efficient lookups. It has been common

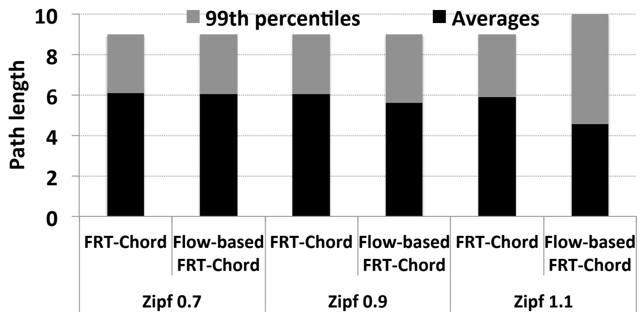

Fig. 4. Path lengths with nonuniform target ID distributions.

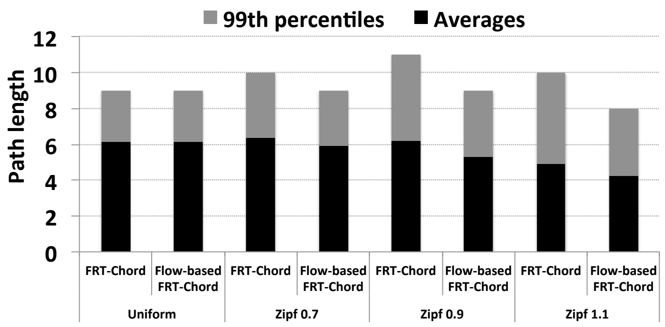

Fig. 5. Path lengths with nonuniform node ID distributions.

knowledge. But we found out that it is not necessary to consider node distance for efficient lookups. Flow-based Flexible Routing Tables (FFRT), a routing table construction method is an evidence of the fact. FFRT-Chord, an FFRT-based overlay performs efficient lookups by maintaining routing tables solely based on query flows without considering node distance. FFRT-Chord shows shorter path lengths at least on average even with nonuniform target ID and node ID distributions.

Future work includes theoretical analysis on path length while the analysis shown in Section IV is experimental.

## Acknowledgment

This work was supported by JSPS KAKENHI Grant Numbers 25700008 and 26540161.

## References

[1] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, Feb. 2003.

[2] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Proc. IPTPS'02*, 2002, pp. 53–65.

[3] A. Datta, M. Hauswirth, R. John, R. Schmidt, and K. Aberer, "Range queries in trie-structured overlays," in *Proc. IEEE P2P'05*, 2005, pp. 57–66.

[4] 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*, 2011, pp. 72–81.

[5] T. Miyao, H. Nagao, and K. Shudo, "A method for designing proximity-aware routing algorithms for structured overlays," in *Proc. IEEE ISCC'13*, 2013.

[6] K. Shudo, Y. Tanaka, and S. Sekiguchi, "Overlay Weaver: An overlay construction toolkit," *Computer Communications*, vol. 31, no. 2, pp. 402–412, 2008.

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