Self-Refining Skip Graph: Skip Graph Approaching to an Ideal Topology

Takafumi Kawaguchi, Ryohei Banno, Masashi Hojo, Masaaki Ohnishi and Kazuyuki Shudo
Tokyo Institute of Technology
Ookayama 2-12-1, Meguro-ku, Tokyo, 152-8552 Japan
Email: {kawaguchi.t.ae, banno.r.aa, hojo.m.aa, ohnishi.m.aa}@m.titech.ac.jp, shudo@is.titech.ac.jp

***Abstract*—In Skip Graph, a structured overlay, each node constructs its routing table by choosing connected nodes based on its membership vector. However, membership vectors are determined randomly; therefore, nodes do not always form an ideal topology. This can cause the route length to be long. Therefore, we propose Self-Refining Skip Graph, a structured overlay where each node refines its routing table toward an ideal Skip Graph topology. Our proposed method has shorter route length as approaching to an ideal topology while maintaining the robustness derived from the mechanism of membership vectors. Our evaluation confirms that the topology approaches to an ideal one and that the route length becomes shorter than that in Skip Graph.**

I. Introduction

Overlay networks are virtual networks constructed on low layer networks for applications such as Peer-to-Peer systems. They are required to provide communication between any two nodes with the number of message forwarding hops as small as possible.

To fulfill this requirement, several techniques of managing distributed data based on one-dimensional structures have been proposed [1], [2], [3], [4], [5]. Such one-dimensional structures have several advantages, e.g., the number of connected nodes is relatively small and the maintenance cost of connecting to other nodes is relatively low. Each node is typically assigned to a one-dimensional identifier. Nodes are arranged in the identifier space based on their identifiers. They connect not only to their closest neighbors but also to their distant nodes to establish shortcut links. When they choose such distant nodes to connect, the following conditions are expected to be considered.

- The number of message forwarding hops and the number of connected nodes should not grow unreasonably fast as the number of nodes increases.
- The maintenance cost of connecting to other nodes should be low.

As shown in Fig. 1, an ideal topology is composed of nodes that consider such conditions when they choose their distant nodes to connect. In an ideal topology, all nodes connect to nodes that are $2^i (i = 0, 1, 2, ...)$ away from their identifiers. This is because the number of rest nodes to a target node is reduced by more than half in one hop when a node communicates with other nodes in a multi-hop manner. Therefore, the number of message forwarding hops and the number of connected nodes become less than log N, where N is the number of nodes.

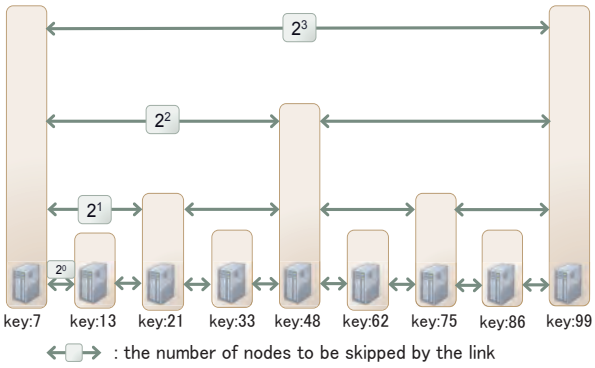

Fig. 1. An ideal topology.

Chord$^\#$ [6] is a structured overlay that strictly maintains an ideal topology. Therefore, the maintenance cost of connecting to other nodes is relatively high. In Skip Graph [7], however, an ideal topology is not strictly maintained. Nodes establish links to other nodes based on m-ary random digits; therefore, they do not always form an ideal topology. Therefore, the maintenance cost of connecting to other nodes is relatively low; however, this can cause the number of message forwarding hops to be large.

Therefore, we propose Self-Refining Skip Graph, a structured overlay where each node refines its connections to other nodes while it joins or leaves the network as in Skip Graph. Our proposed method enables each node to construct its connections to other nodes quickly as in Skip Graph and enables the topology to approach to an ideal one. This means that the maintenance cost of connecting to other nodes is relatively low, but the number of message forwarding hops becomes smaller than that in Skip Graph.

In this paper, Skip Graph is described in Sec. II as background for our proposed method. Self-Refining Skip Graph, our proposed structured overlay, is described in Sec. III. The evaluation of Self-Refining Skip Graph is described in Sec. IV.

II. Background

In this section, we describe Skip Graph [7] and its non-ideal feature.

Proc. 14th IEEE Consumer Communications & Networking Conference
(IEEE CCNC 2017), pp.441-448, January 2017

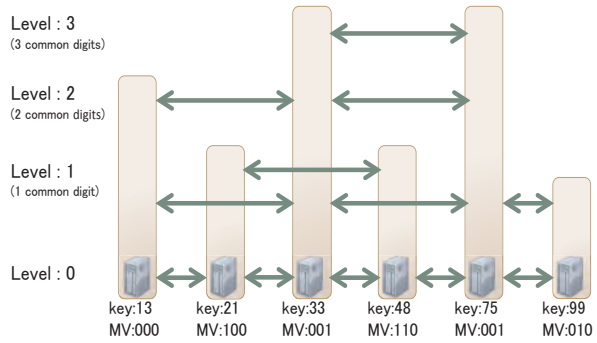

Fig. 2. A Skip Graph topology.

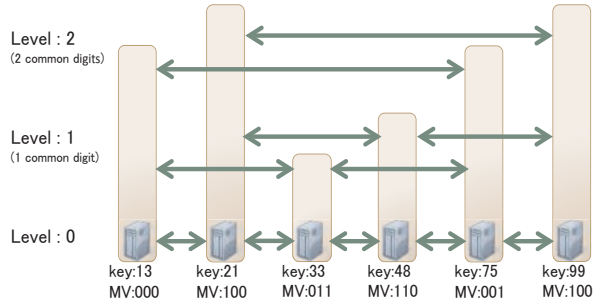

Fig. 3. An ideal Skip Graph topology.

A. Concept of Skip Graph

Skip Graph is a structured overlay based on Skip List [8]. Each node is assigned to a key as its identifier. Nodes are arranged in their key order in the key space. Therefore, Skip Graph supports range queries.

Nodes are located on hierarchical lists based on their keys and their membership vectors (MVs). An MV is a sequence of m-ary random digits (in this paper, let m be 2). As shown in Fig. 2, nodes establish bidirectional links to their closest neighbors at level 0 and to their closest nodes that have i common digits in their MVs at level i.

Each node constructs two routing tables, i.e., neighbor[R] and neighbor[L]. Neighbor[R] is a list that contains the closest nodes' information from its key in ascending order at each level. Neighbor[L] is a list that contains the closest nodes' information from its key in descending order at each level.

Routing in Skip Graph is performed by repeatedly forwarding a message to the closest node to a target node's key so that the message reaches a target node. Each node chooses an appropriate node from the highest level in its neighbor[L] and neighbor[R]. If the message cannot continue its approach at a given level, the node chooses an appropriate node from the next lower level in its neighbor[L] and neighbor[R]. In Skip Graph, message reachability is guaranteed by performing routing using neighbor[L] and neighbor[R], including the closest neighbors. The route length, the number of message forwarding hops, is $O(\log\ N)$, where N is the number of nodes, when performing routing using neighbor[L] and neighbor[R]. This is because the number of rest nodes to a target node can probability be reduced by more than half in one hop.

When a node joins or leaves the network, it constructs its routing table quickly and only its neighbor nodes update their routing tables because nodes establish bidirectional links to other nodes based on their MVs. Therefore, the maintenance cost of routing tables is relatively low.

B. The non-ideal feature of Skip Graph

In Skip Graph, nodes do not always form an ideal topology due to the deviations in the MVs that are determined randomly. The longer the length of common digits in the MVs between neighbor nodes, the larger the deviations in the MVs. Therefore, the route length can relatively be long.

As shown in Fig. 3, in an ideal Skip Graph topology, it is expected that all nodes' neighbors at level i are the same as their neighbors' neighbors at level $i-1$. In this way, all nodes can connect to nodes that are $2^i (i = 0, 1, 2, ...)$ away from their keys. This results in efficient routing. However, in reality, neighbor nodes have some common digits in their MVs. Therefore, it is often the case that one node's neighbors at level i are the same as its neighbors at level $i-1$. This means that some nodes have overlapping entries between neighbor levels in their routing tables. Therefore, a Skip Graph topology is not always ideal, which does not always achieve efficient routing.

III. Proposed Method

We propose Self-Refining Skip Graph, a structured overlay where each node refines its routing table by modifying its MV while it constructs its routing table based on its MV as in Skip Graph [7]. First, we describe the basic idea of Self-Refining Skip Graph.

A. Basic idea

As described in Sec. II-B, in Skip Graph, nodes do not always form an ideal topology due to the deviations in the MVs. Therefore, we propose a method where each node detects and modifies the deviations in the MVs so that the topology approaches to an ideal one.

The modification of MVs is achieved by inverting one digit in MVs. The nodes that need to modify their own MVs are the even-numbered nodes in an *MV deviation node sequence*. As shown in Fig. 4, an *MV deviation node sequence* at level i is a sequence that consists of nodes that have an i-th common digit and an $(i-1)$-th common digit in their MVs, i.e., nodes whose neighbors at level i are the same as their neighbors at level $i-1$. If the even-numbered nodes in the sequence invert the i-th digit in their MVs, in all neighbor nodes in the sequence, the i-th digit will be different in their MVs, and their neighbors at level i will be different from their neighbors at level $i-1$. In this way, if we remove the *MV deviation node sequence*, the topology approaches to an ideal one.

When each node performs such a modification of its MV once, the property of an ideal Skip Graph topology where all

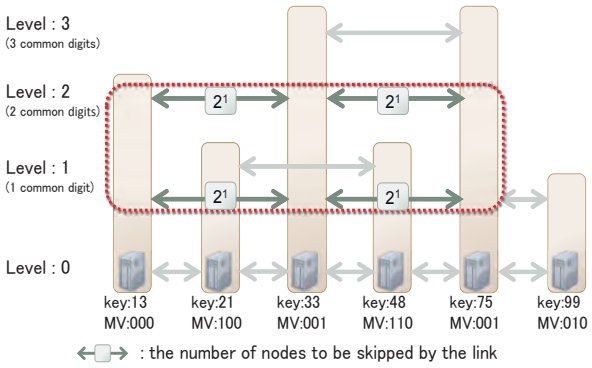

Fig. 4. An MV deviation node sequence in Skip Graph.

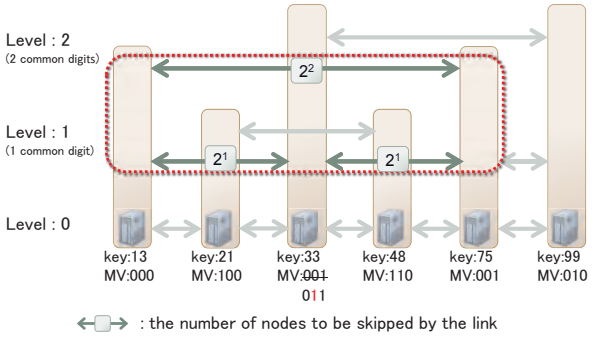

Fig. 5. The removal of an MV deviation node sequence.

nodes' neighbors at level i are the same as their neighbors' neighbors at level $i-1$ is partially realized, which the topology approaches to an ideal one. If each node performs this modification multiple times, the property of an ideal Skip Graph topology is completely realized, and the resulting topology is ideal.

For example, in Fig. 4, there is an *MV deviation node sequence* at level 2 that consists of key:13, key:33, and key:75. These nodes have two common digits in their MVs, namely, 00; therefore, key:13 connects to key:33 at levels 1 and 2, key:33 connects to key:13 and key:75 at levels 1 and 2, and key:75 connects to key:33 at levels 1 and 2. This *MV deviation node sequence* consists of three nodes; therefore, only the second node, key:33, is the even-numbered node. Therefore, key:33 inverts the 2nd digit in its MV. As shown in Fig. 5, in all neighbor nodes in the sequence, i.e., in key:13 and key:33, and key:33 and key:75, the 2nd digit is different in their MVs; therefore, key:13 connects to key:75 at level 2 and to key:33 at level 1, key:33 connects to key:99 at level 2 and to key:13 and key:75 at level 1, and key:75 connects to key:13 at level 2 and to key:33 at level 1. Therefore, we could remove an *MV deviation node sequence*.

B. Concept of Self-Refining Skip Graph

The methods of assigning keys to nodes and of constructing routing tables are the same as those in Skip Graph. This keeps the maintenance cost of routing tables as low as that in Skip Graph. The method of assigning MVs to nodes is also the same as that in Skip Graph. If nodes are assigned to their MVs not by generating m-ary random digits but by calculating for an ideal topology, the route length can be less than $\log N$, where N is the number of nodes. In this case, however, all nodes are forced to modify their MVs and to update their routing tables toward an ideal topology every time a node joins or leaves the network; therefore, the maintenance cost of routing tables becomes higher than that in Skip Graph. Therefore, each node refines its routing table by modifying its MV after it is assigned to its MV by generating m-ary random digits.

After each node joins the network by using the same method in Skip Graph, it executes the *MV modification protocol*, an algorithm where a node autonomously refines its routing table by modifying its MV, at regular intervals. In the following sections, we describe the details of the *MV modification protocol*.

Executing the *MV modification protocol* results in the number of overlapping entries between neighbor levels in the routing tables decreasing, and the property of an ideal Skip Graph topology where all nodes' neighbors at level i are the same as their neighbors' neighbors at level $i-1$ is gradually realized. If each node continues executing the *MV modification protocol* for a long time, the property of an ideal Skip Graph topology is completely realized. However, this is not our goal. Our goal is to keep the maintenance cost of routing tables as low as that in Skip Graph while improving the routing efficiency by performing normal routing as each node refines its routing table. Therefore, we can perform normal routing without waiting to reach an ideal topology.

C. The MV modification protocol

In the *MV modification protocol*, a node detects and modifies the deviations in the MVs, and updates its routing table. Each node autonomously refines its routing table; therefore, decreasing the number of overlapping entries between neighbor levels in one node's routing table may cause an increase in the number of overlapping entries in other nodes' routing tables. However, this is not a problem because the modification of MVs does not oscillate. This is because the topology is not a ring but a list and messages for detecting and modifying the deviations in the MVs must be sent in the direction of ascending order.

1) Detecting and modifying the deviations in MVs: First, a node examines whether its own MV causes the deviations in the MVs. In other words, a node examines whether it is contained in an *MV deviation node sequence* by examining whether its neighbor nodes between any neighbor levels in its routing table are the same.

Next, if a node is contained in an *MV deviation node sequence*, it decides whether it should modify its own MV by cooperating with its neighbor nodes. From Sec. III-A, if a node knows whether it is the even-numbered node in the sequence, it can decide whether it should modify its own MV. However, all nodes except the first one in the sequence do not know their location in the sequence; therefore, they have to learn

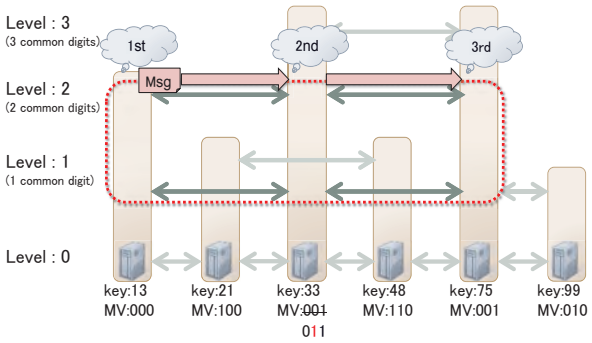

Fig. 6. Sending an MV deviation detection message.

their location by receiving an *MV deviation detection message* sent by the first node. An *MV deviation detection message* is a message containing the number of forwarding hops. The nodes that receive this message can learn their location in the sequence by examining how many times the message is forwarded from the first node. Therefore, the first node sends an *MV deviation detection message* to the closest node in the *MV deviation node sequence*. A node that receives this message learns its location in the sequence from the number of message forwarding hops, and if it is the even-numbered node in the sequence, it decides to modify its own MV.

Finally, if a node decides to modify its own MV, it inverts the one appropriate digit in its own MV.

In Self-Refining Skip Graph, therefore, each node has the following function to execute the *MV modification protocol*.

1) When a node receives an *MV deviation detection message*, it learns its location in the *MV deviation node sequence* from the number of message forwarding hops.
   - If it is the even-numbered node in the sequence, it modifies its own MV.
2) The node forwards the message to the closest node in the *MV deviation node sequence*. If there is no next closest node, it deletes the message.

Under these circumstances, the modification of MVs is achieved by executing the following *MV modification protocol* at regular intervals and sending an *MV deviation detection message* as shown in Fig. 6.

1) A node examines whether it is contained in an *MV deviation node sequence*.
2) If the node is contained in an *MV deviation node sequence* and is the first node in the sequence, it sends an *MV deviation detection message* to the closest node in the sequence.

For example, in Fig. 6, key:13 is contained in an *MV deviation node sequence* and is the first node in the sequence; therefore, it sends an *MV deviation detection message* to the closest node in the sequence, i.e., key:33. Then, key:33 receives the *MV deviation detection message*, learns its location in the sequence, and forwards the message to the closest node in the sequence, i.e., key:75. Key:33 also modifies its own MV because it is the even-numbered node in the sequence. Then, key:75 learns its location in the sequence and deletes the message because there is no next closest node in the sequence.

If a node is contained in the *MV deviation node sequence* at multiple levels, the decision to send an *MV deviation detection message* is performed at the lowest level. As described in Sec. III-C2, if a node inverts the i-th digit in its MV, it needs to update the entries of more than level i in its routing table. We describe an example where a node is contained in the *MV deviation node sequence* at levels i and j ($i < j$). First, if a node inverts the j-th digit in its MV, it updates the entries of more than level j in its routing table. Next, it inverts the i-th digit in its MV and updates the entries of more than level i, including the entries of more than level j, in its routing table. However, this is a wastage because the entries of more than level j have just been updated. To remove such a wasteful operation, the decision to send an *MV deviation detection message* is always performed in the *MV deviation node sequence* at the lowest level. In this way, updating the routing tables is performed from the lowest level.

For example, in Fig. 6, key:75 is contained in the *MV deviation node sequence* at level 2, which consists of key:13, key:33, and key:75, and at level 1, which consists of key:75 and key:99. Under these circumstances, the decision of whether key:75 sends an *MV deviation detection message* is performed in the *MV deviation node sequence* not at level 2 but at level 1.

2) Updating routing tables: In Self-Refining Skip Graph, each node constructs its routing table based on its MV when it joins the network. Therefore, if a node modifies its MV after it joins the network, several nodes will need to update their routing tables based on the node's new MV. This operation is nearly the same as the operation where a node that modified its MV leaves and joins the network.

We describe an operation in which when one node, A, inverts the i-th digit in its MV, several nodes update their routing tables based on node A's new MV. This operation consists of two phases; updating the routing tables of the nodes that contain node A in their routing tables and updating the routing table of node A.

First, we describe how to update the routing tables of the nodes that contain node A in their routing tables. The nodes that need to update their routing tables in the nodes that contain node A in their routing tables are the nodes that are contained in more than level i in node A's routing table. Therefore, node A sends an *MV notification message* to them. An *MV notification message* is a message to notify other nodes of node A's new MV. The nodes that receive this message can update their routing tables based on node A's new MV.

In Self-Refining Skip Graph, therefore, each node has the following function to update its routing table when other nodes modify their MVs.

1) When a node receives an *MV notification message*, it updates the entries of the message source node in its routing table based on the new MV.

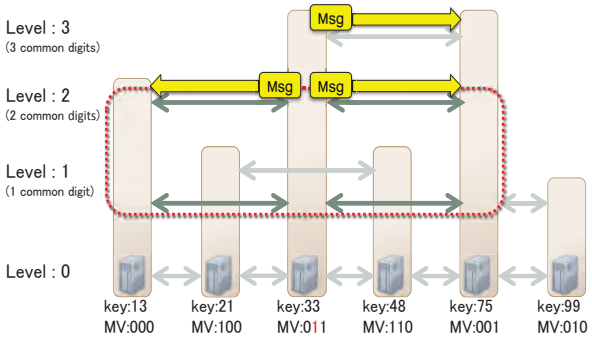

Fig. 7. Sending an MV notification message.

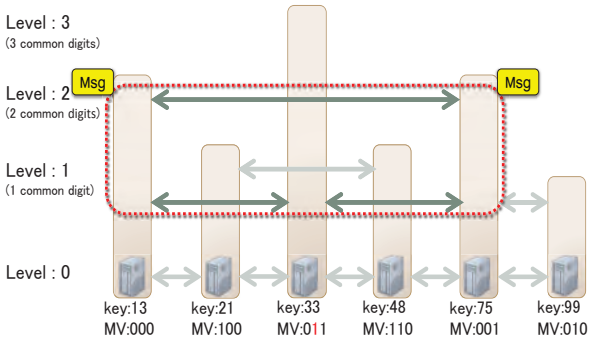

Fig. 8. Progress of updating the routing tables.

Under these circumstances, updating the routing tables of the nodes that contain node A in their routing tables is achieved by performing the following operation and sending an *MV notification message* as shown in Fig. 7.

1) Node A sends an *MV notification message* to the nodes that are contained in more than level i in its routing table.

For example, in Fig. 7, key:33 sends an *MV notification message* to the nodes that are contained in more than level 2 in its routing table, i.e., key:13 and key:75, because it inverts the 2nd digit in its MV. As shown in Fig. 8, key:13 and key:75 receive the *MV notification message*; therefore, they update the entries of key:33 based on the new MV.

Next, we describe how to update the routing table of node A. The entries that need to be updated in node A's routing table are the entries of more than level i. Therefore, node A updates these entries in its routing table based on its new MV. In this way, updating the routing table of node A is achieved.

For example, in Fig. 8, key:33 updates the entries of more than level 2 in its routing table based on its new MV. Therefore, as shown in Fig. 5, we could remove an *MV deviation node sequence*.

IV. Evaluation

We implemented Self-Refining Skip Graph on Overlay Weaver [9] [10], an overlay construction toolkit, and performed experiments. The experimental environment is as shown in TABLE I.

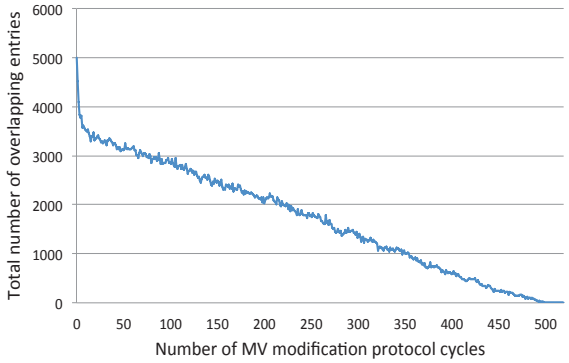

Fig. 9. Behavior confirmation (N = 1000).

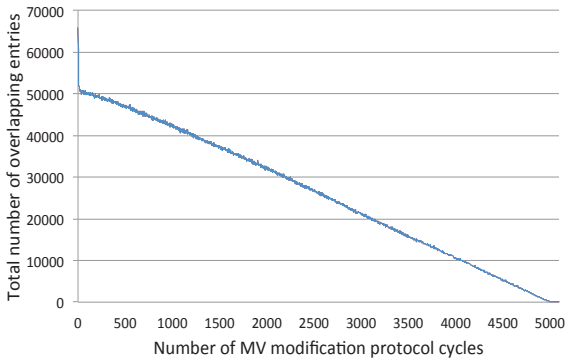

Fig. 10. Behavior confirmation (N = 10000).

In these experiments, we set the number of nodes to N = 1000 and 10000. Then, we measured the route length when all nodes sent messages to other nodes after executing T *MV modification protocol* cycles. We define an *MV modification protocol* cycle as a period when all nodes autonomously execute the *MV modification protocol* once and completely update their routing tables. $T = 0$ indicates that no nodes have modified the deviations in the MVs; therefore, its performance is the same as that of Skip Graph [7]. In other words, the result of $T = 0$ is the same as that of Skip Graph.

A. Behavior confirmation of the MV modification protocol

As described in Sec. II-B, in Skip Graph, some nodes have overlapping entries between neighbor levels in their routing

TABLE I
Experimental environment.

Simulator	Overlay Weaver 0.10.4
OS	Mac OS X 10.10.3
CPU	Intel Core i7-5557U 3.1GHz
Memory	16GB
Java	Java SE 8 Update 45

tables. As described in Sec. III-B, executing the *MV modification protocol* results in the number of overlapping entries between neighbor levels in the routing tables decreasing.

To confirm this, we counted the total number of overlapping entries between neighbor levels in all nodes' routing tables every time when executing an *MV modification protocol* cycle. For example, when the entries of levels 0, 1, and 2 in one node's routing table are the same, we can find the overlapping entries between levels 0 and 1, and between levels 1 and 2. Therefore, the number of overlapping entries in this node's routing table is 2. The total number of overlapping entries becoming 0 infers that the topology reaches an ideal one.

As shown in Figs. 9 and 10, we found that the total number of overlapping entries decreased as the number of *MV modification protocol* cycles increased. In the case of N = 1000, when we executed 500 *MV modification protocol* cycles, the total number of overlapping entries became 0. In the case of N = 10000, when we executed 5014 *MV modification protocol* cycles, the total number of overlapping entries became 0. Therefore, we confirmed that the *MV modification protocol* behaved as we intended.

B. Relationship between the MV modification protocol cycles and the route length

We measured the route length after executing 0, 5, and N / 2 *MV modification protocol* cycles and calculated the average and maximum route lengths. These results include the result of 0 *MV modification protocol* cycles; therefore, we can compare the performance of Self-Refining Skip Graph with that of Skip Graph.

As shown in Figs. 11 $-$ 16, we found that the route length decreased overall as the number of *MV modification protocol* cycles increased. As shown in TABLEs II and III, the average route length decreased. The maximum route length also decreased, which shows that executing the *MV modification protocol* improves the problem where the maximum route length is often long in Skip Graph. Compared to the result of $T = 0$, i.e., the result of Skip Graph, we found that the proposed method, where each node executes the *MV modification protocol*, provided more efficient routing.

TABLE II
AVERAGE AND MAXIMUM ROUTE LENGTHS (N = 1000).

MV modification protocol cycles	0	5	500
Average route length	8.34	6.58	4.48
Maximum route length	30	25	9

TABLE III
AVERAGE AND MAXIMUM ROUTE LENGTHS (N = 10000).

MV modification protocol cycles	0	5	5014
Average route length	11.40	9.99	6.14
Maximum route length	47	37	13

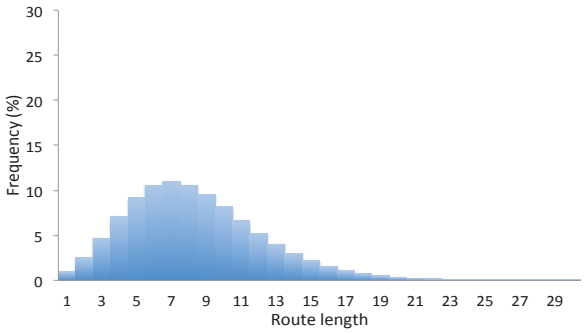

Fig. 11. Route length (N = 1000, T = 0).

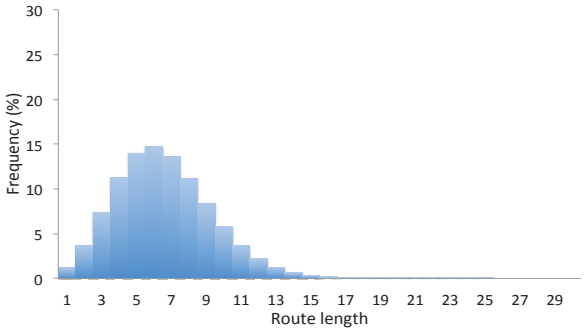

Fig. 12. Route length (N = 1000, T = 5).

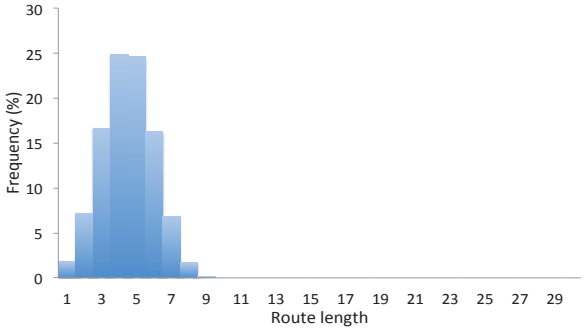

Fig. 13. Route length (N = 1000, T = 500).

C. Approaching to an ideal topology

We measured the route length every time when executing an *MV modification protocol* cycle and calculated the average route length. The process of a decrease in the average route length illustrates how a topology approaches an ideal one.

As shown in Figs. 17 and 18, we found that the average route length decreased nearly linearly as the number of *MV modification protocol* cycles increased. In the case of N = 1000, when the number of *MV modification protocol* cycles was 0, the average route length was 8.34. As we executed several *MV modification protocol* cycles, the average route length dramatically decreased from 8.34 to 7.81, 7.28, 6.99, 6.84, 6.58, and so on. In the case of N = 10000, when the number of *MV modification protocol* cycles was 0, the

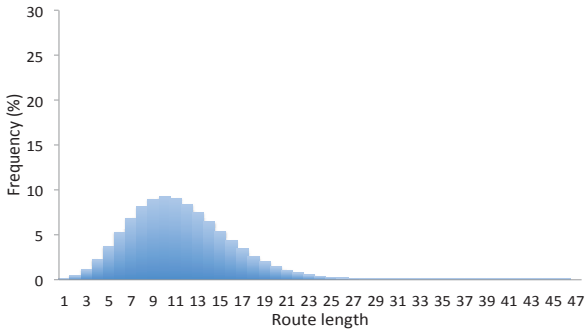

Fig. 14. Route length (N = 10000, T = 0).

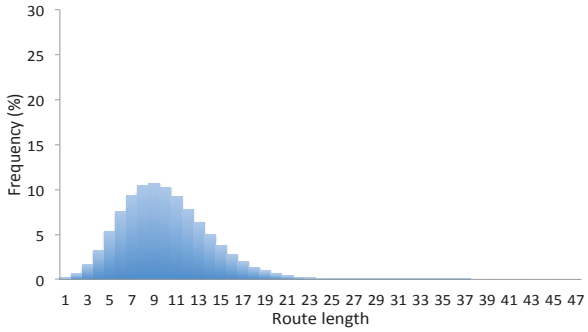

Fig. 15. Route length (N = 10000, T = 5).

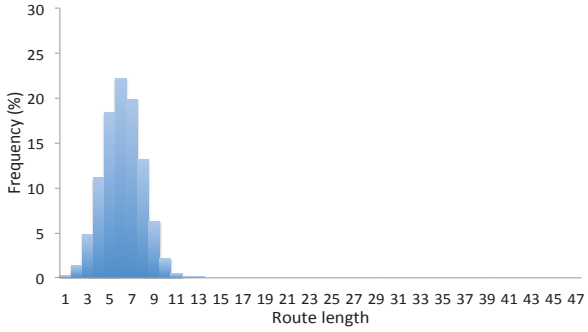

Fig. 16. Route length (N = 10000, T = 5014).

average route length was 11.40. As we executed several *MV modification protocol* cycles, the average route length dramatically decreased from 11.40 to 10.85, 10.48, 10.21, 10.04, 9.99, and so on. These illustrate that the topology dramatically approaches to an ideal one after we execute several *MV modification protocol* cycles.

When we continued executing the *MV modification protocol*, the average route length continued to decrease. In the case of N = 1000, when we executed 500 *MV modification protocol* cycles, the average route length converged to 4.48, which the topology reached an ideal one. In the case of N = 10000, when we executed 5014 *MV modification protocol* cycles, the average route length converged to 6.14. Therefore, the number of *MV modification protocol* cycles when the average

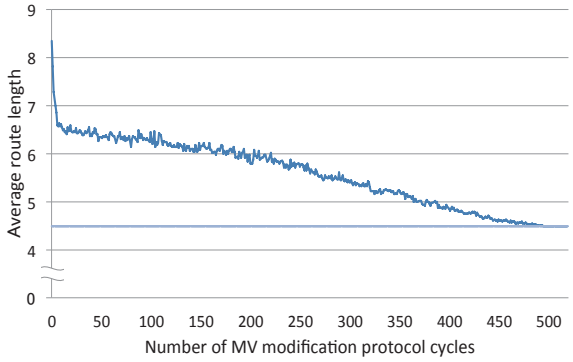

Fig. 17. Approaching to an ideal topology (N = 1000).

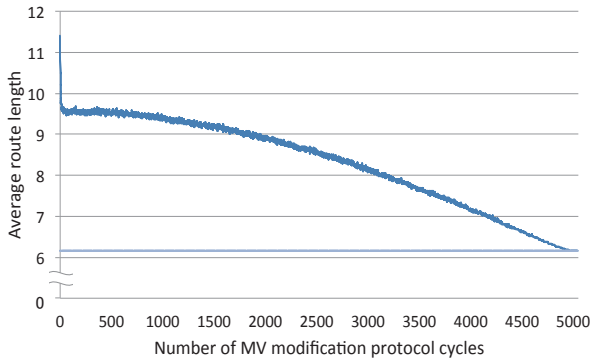

Fig. 18. Approaching to an ideal topology (N = 10000).

route length converges is nearly half of the number of nodes. Experimentally, we found that the number of *MV modification protocol* cycles when the average route length converges is proportional to the number of nodes. However, this does not affect the performance of Self-Refining Skip Graph because it is not our goal to reach an ideal topology.

These experiments emphasize that the topology dramatically approaches to an ideal one after we execute several *MV modification protocol* cycles in the Skip Graph topology. This means that the route length becomes shorter than that in Skip Graph as long as we continue executing the *MV modification protocol* even if nodes join or leave the network frequently. This is a practical result.

V. Conclusions and Future Work

In this paper, we proposed Self-Refining Skip Graph, a structured overlay where each node refines its routing table toward an ideal Skip Graph topology. Our proposed method has shorter route length as approaching to an ideal topology while maintaining the robustness derived from the mechanism of MVs. Structured overlays where nodes are arranged in their key order are often applied to range queries. Our proposal enables range queries to be performed with the maintenance cost of routing tables as low as that in Skip Graph while improving the routing efficiency.

Proc. 14th IEEE Consumer Communications & Networking Conference
(IEEE CCNC 2017), pp.441-448, January 2017

Our evaluation confirmed that the topology dramatically approaches to an ideal one after we execute several *MV modification protocol* cycles even if each node constructs its routing table based on its MV as in Skip Graph. Therefore, the route length becomes shorter than that in Skip Graph as long as we continue executing the *MV modification protocol* even if nodes join or leave the network frequently.

Future work includes theoretical evaluations of the message forwarding cost in the *MV modification protocol* and the time taken for the topology to reach an ideal one. We will also optimize the *MV modification protocol* to keep the message forwarding cost low and to improve the decreasing rate of route length per an *MV modification protocol* cycle.

ACKNOWLEDGMENTS

This work was supported by JSPS KAKENHI Grant Numbers 25700008, 26540161, and 16K12406.

REFERENCES

[1] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *Networking, IEEE/ACM Transactions on*, 11(1):17–32, 2003.

[2] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of IFTP/ACM Middleware 2001*, pages 329–350, 2001.

[3] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, pages 53–65, 2002.

[4] M. F. Kaashoek and D. R. Karger. Koorde: A Simple Degree-Optimal Distributed Hash Table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, pages 98–107, 2003.

[5] A.R. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting Scalable Multi-Attribute Range Queries. *ACM SIGCOMM Computer Communication Review*, 34(4):353–366, 2004.

[6] T. Schutt, F. Schintke, and A. Reinefeld. Range Queries on Structured Overlay Networks. *Computer Communications*, 31(2):280–291, 2008.

[7] J. Aspnes and G. Shah. Skip Graphs. *ACM Transactions on Algorithms*, 3(4):Article 37, 2007.

[8] W. Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees. *Communications of the ACM*, 33(6):668–676, 1990.

[9] K. Shudo. Overlay Weaver. http://overlayweaver.sourcegorge. net, 2006.

[10] K. Shudo, Y. Tanaka, and S. Sekiguchi. Overlay Weaver: An Overlay Construction Toolkit. *Computer Communications*, 31(2):402–412, 2008.