Ballistic Skip Graph: A Skip Graph-Style Constant-Degree Structured Overlay

Yusuke Aoki, Masaaki Ohnishi, Kazuyuki Shudo
Tokyo Institute of Technology
Ookayama 2-12-1, Meguro-ku, Tokyo, 152-8552 Japan
Email: {aoki.y.au, ohnishi.m.aa}@m.titech.ac.jp, shudo@is.titech.ac.jp

***Abstract*—Structured overlays enable the construction of application-level networks from multiple nodes, and the decentralized searching of data in the network. One such structured overlay is the skip graph, which supports range queries. Each node in a skip graph has multi-level shortcut links. The degree of each node is O(log N) on a network with N nodes. The paper, proposes Ballistic Skip Graph, a structured overlay that reduces the degree of each node to O(1) by limiting the number of shortcut links to a randomly selected single level while supporting range queries. Because the nodes are not grouped, the proposed method requires no extensive reconfiguration of the network when nodes join or leave. An evaluation experiment confirmed that the average routing table size is confined to a small constant.**

I. Introduction

Overlay networks are virtual networks constructed on low layer networks. When an overlay network is based on mathematical rules, it is known as a structured overlay network. A structured overlay assigns identifiers (IDs) to the nodes and data. Each node builds a routing table comprising sets of the ID and internet protocol addresses of neighboring nodes. The nodes communicate in a multihop manner; i.e., each node forwards a message to appropriate nodes selected from its routing table under the mathematical rules. Structured overlays maintain a low number of messages even when the number of nodes increases, because they limit the number of nodes receiving the message.

A distributed hash table (DHT) [1] is a key–value storage constructed on a structured overlay. A DHT stores data as key–value pairs and searches the data item associated with a given key. The hash value of the key of each data item is also the ID of that data item, and each node is responsible for the data in its ID range. Therefore, a datum is stored in the node responsible for its data ID. The key of the data on a node sequence is not maintained in a DHT, because the data are stored in the order of their IDs. For this reason, the DHT does not support range queries.

A skip graph [2] is a structured overlay that supports range queries. Each node in a skip graph has $O(\log N)$ shortcut links, where N is the number of nodes. Like many DHT algorithms, the path length in a skip graph is $O(\log N)$ when using a proper shortcut link. However, unlike many DHT algorithms, a skip graph stores the data in key order rather than the hash values of the keys.

If the node degree (i.e., the routing table size) could be reduced to $O(1)$, we can reduce the maintenance cost of the routing table. Several attempts have been made in this direction [3]–[5]. A failed communication should be maintained for sufficient time to judge whether the failure is merely a communication trouble or whether the destination node is broken or disconnected from the network. For this purpose, each node in many cases periodically sends heartbeat messages to the nodes in the routing table, which do not affect the usual routing. Reducing the table size of each node to $O(1)$ would markedly reduce the cost of this process and would simplify the resource management of physical nodes in applications that create several virtual nodes within one physical node.

The degree of each node in a skip graph is $O(\log N)$. Some studies have proposed structured overlays that reduce the routing table size to $O(1)$ by extending skip graphs [4], [5]. However, these structured overlays either require the maintenance of node groups or are intolerant to faults. These problems are especially severe in environments where nodes frequently join and leave. As leaving and joining changes the proper groups of nodes, a node group in this environment requires frequent reconfiguring, which incurs a large cost. Furthermore, even when group reconfiguration is not required, the total number of nodes must be regularly estimated which is also expensive.

The present paper, proposes Ballistic Skip Graph, a structured overlay that reduces the routing table size of $O(1)$ without grouping the nodes. The paper also retains the advantage of skip graphs . However, as nodes in the original skip graph have multiple levels of shortcut links, Ballistic Skip Graph randomly decides the level of the shortcut links to one to each node, thereby reducing the routing table size to $O(1)$. Because the level of each node is determined randomly without depending on other nodes, Ballistic Skip Graph does not require the total number of nodes or maintenance of a node group. Consequently, the influence of node joining and leaving is locally suppressed.

Section II of this paper describes skip graphs and other relevant background research. Section III and Section IV proposes and evaluations our structured overlay, respectively.

Proc. 23rd IEEE Symposium on Computers and Communications
(IEEE ISCC 2018), June 2018

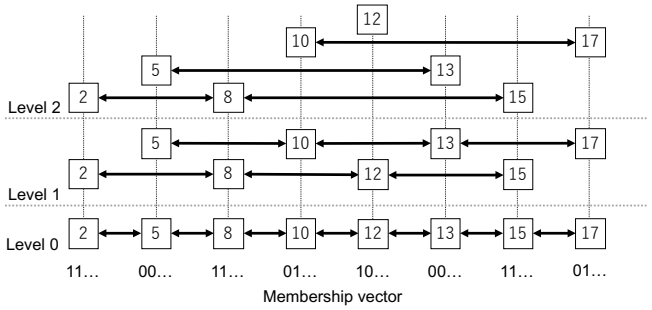

Fig. 1. Example of skip graph

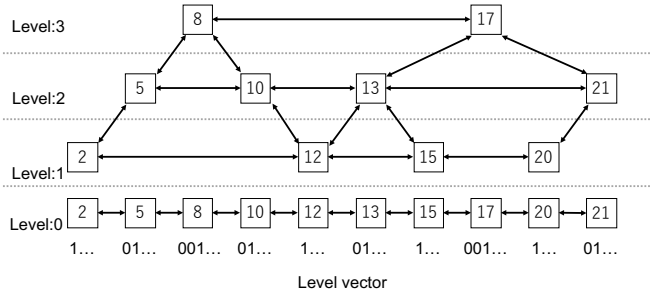

Fig. 2. Example of Ballistic Skip Graph without load balancing

The summary and future challenges are given in Section V.

II. Related work

A skip graph is a structured overlay based on a skip list [6], a data structure that extends the linked list. The network structure of a skip graph is shown in Fig. 1.

Each node in a skip graph is assigned a key and a random-bit string known as a membership vector (MV). The key and MV determine the multilevel lists on which the node is placed. For each level i, the set of nodes with i common prefix bits of the MV forms a level list, defined as a bidirectional list of nodes in the key order. Each node has two routing tables: one containing information of the successor nodes and the other containing information of the predecessor nodes. The entries in the routing table size terminate at the level at which no nodes have common MV prefixes, so the routing table is sized $O(\log N)$.

Routing in a skip graph is performed by repeatedly transferring messages, preferentially though high-level shortcut links. Each node transfers its message to the node closest to the target key that does not pass the target node. As the number of nodes existing between a transferring node and the target node in one transfer is expected to below half the number, the path length is $O(\log N)$. In addition, a skip graph guarantees the reachability of the message by the bidirectional list, in which all nodes participate in the key order.

Although DHT indexes the hash keys, a skip graph supports range queries as mentioned above. Range queries are made through normal routing and flooding. First, normal routing finds a single node in the range, requiring $O(\log N)$ steps and $O(\log N)$ messages. The query is then broadcast to the m nodes in the range by flooding, which requires $O(\log m)$ steps and $O(m \log N)$ messages. The entire operation takes $O(\log N)$ steps.

Constant-degree structured overlays based on the skip graph have also been proposed. For example, a family tree [4] is a constant-degree overlay, that supports range queries by combining skip graph and the hierarchical structure of a constant order structured overlay known as Viceroy [7]. Like Viceroy, a family tree estimates the total number of nodes and randomly decides the level of the node from 0 to $\log N$, maintaining the degree of nodes as 9. However, when a node leaves the family tree, all nodes with a link to the leaving node must be searched and assigned appropriate new link destinations. For this reason, a family tree cannot accommodate sudden leaves due to node failure and is not realistic in a distributed environment.

Rainbow Skip Graph [5] combines several adjacent nodes in the level-zero list of the skip graph, treating them as one supernode, and constructs a skip graph of the supernodes. At this time, the links of each level are shared among the nodes in the supernode; thus, the size of the routing table is $O(1)$. However, to maintain the constant order, the number of nodes in a supernode must be maintained at $O(\log N)$. For this reason, it is necessary to estimate the total number of nodes. Moreover, when the number of nodes in a supernode falls outside the appropriate range, the supernode must be merged or separated and reconstructed by appropriate links. In an environment of frequent leaving and joining of nodes, the high frequency of rebuilding the supernodes increases the cost of maintaining the network.

III. Ballistic Skip Graph

We propose Ballistic Skip Graph, a new structured overlay with a routing table size of $O(1)$ without the need for grouping nodes, with supporting range queries which is a notable advantage of skip graphs (range-query support). Unlike other constant-degree structured overlays based on skip graphs, Ballistic Skip Graph does not group the nodes, thus negating the need to estimate the number of nodes and reconfigure the group. This is especially important when nodes frequently join and leave the network. All nodes in Ballistic Skip Graph belong to level-zero list. In addition, each node belongs to a randomly determined level list and is linked to an upper and lower-level list. We first explain the simple network structure without load balancing, then introduce the load balanced overlay that constitutes Ballistic Skip Graph.

A. Ballistic Skip Graph without load balancing

Fig. 2 shows Ballistic Skip Graph without load balancing. Like the skip graph, a key is assigned to each node, and all nodes in the level-zero list participate in their key order. The key of a node X is denoted by $X.Key$. The level-zero list is constructed from each node X with $X.predecessor$ and $X.successor$, which define the predecessor and successor nodes in the key order, respectively. Besides participating in

```
1: while true do
2:     if X.upperSuccessor.key ≤ v then
3:         X ⇐ X.upperSuccessor
4:     else if X.levelSuccessor.key ≤ v then
5:         X ⇐ X.levelSuccessor
6:     else if X.lowerSuccessor.key ≤ v then
7:         X ⇐ X.lowerSuccessor
8:     else if X.successor.key ≤ v then
9:         X ⇐ X.successor
10:    else
11:        return X
12:    end if
13: end while
```

Fig. 3. Search for key v

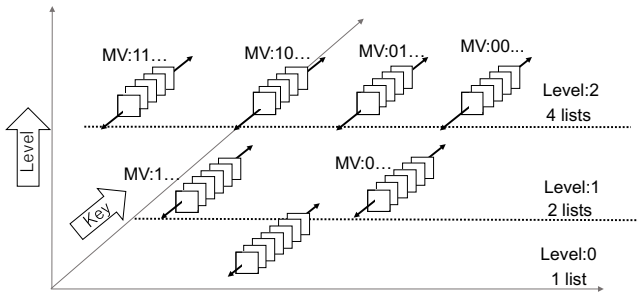

Fig. 4. Structure of Ballistic Skip Graph

the level-zero list, each node participates in the list of its level of responsibility.

To determine the responsible level, each node is assigned a random-bit string named as a level vector (LV). When the LV is counted from the most significant bit, count at the first appearance of one becomes the responsible level of the node. The level of X is denoted as $X.level$. A set of same-level nodes is built into a level list. The level list is constructed from nodes X with the same-level predecessor node $X.levelPredecessor$ and the same-level successor node $X.levelSuccessor$.

Furthermore, each node has links, enabling its movement to lists of upper and lower levels. These links are $X.upperPredecessor$ and $X.upperSuccessor$, defining the predecessor and successor nodes, respectively, in the list of the next upper level. Moreover, $X.lowerPredecessor$ and $X.lowerSuccessor$ are the corresponding nodes in the next lower level. However, to maintain bidirectionally of the upper and lower links, when a same-level node is nearer than the upper or lower link destination node, the link to the upper or lower level is not constructed.

In this network structure, the maximum number of entries in the routing table of any node is 8.

As higher level nodes are sparse in the network, higher-level shortcut links can skip more nodes than low-level shortcut links. Therefore, the routing preferentially uses the higher-level links. Fig. 3 displays the algorithm that searches the data item of key v. In this algorithm, v is the successor key of the start node X. The algorithm that searches the data item of the predecessor key v is not shown, because it differs from Fig. 3 only by the replacement of $successor$ with $predecessor$.

The node that forwards the message checks the upper link, the level link, and the lower link in the target key direction of the message in order. If the link destination key does not exceed the target key, the message is transferred to that link destination. When all destinations of the upper, level, and lower links exceed the target key, the message is transferred to the node adjacent to the level-zero list. Using the link in this priority order, raises the level unless the transfer exceeds the target key, and the routing transfers the message along the high-level shortcut links.

Ballistic Skip Graph was named after its routing path, which moves up and down at an oblique angle like a ballistic object. This routing guarantees the reachability of the message by the level 0 list, in which all nodes participate in the key order.

Like the skip graph, Ballistic Skip Graph conducts range queries by normal routing and flooding. First, normal routing finds a single node in the range, and flooding broadcast the query to m nodes in the range. Flooding requires the same number of steps as the routing path length of the m-nodes Ballistic Skip Graph. The entire operation requires the same number of steps as the normal routing path length.

B. Ballistic Skip Graph

As the routing in the structure of Section III-A favors higher-level links, the load is concentrated at these links. For load balancing in Ballistic Skip Graph, we construct more list at higher levels than lower levels. The structure of Ballistic Skip Graph is shown in Fig. 4.

Besides its LV, each node is assigned a random-bit string known as its membership vector (MV). The MV of node X is denoted as $X.MV$ and its i prefix digits are represented by $X.MV \upharpoonright i$. Each node establishes a level link with a node in the same responsible level and whose MV prefixes equal to each other up to level number digit; that is, we construct a level link between the nearest nodes X and Y satisfying the following conditional expression:

$$X.level = Y.level \wedge X.MV \upharpoonright X.level = Y.MV \upharpoonright Y.level$$

Under this rule, each node participates in the level-zero list and a single level list.

Because 2^i different level lists are constructed at each level i, the number of lists increases at higher levels. More specifically, the number of lists in the next upper level is double the number of lists in the current level, so the upper links of each node are directed toward two lists in the next upper level, and the lower links are directed toward one list of one lower level.

Each node at level i has four upper links. $upper0Predecessor$ and $upper0Successor$ link to the nodes in the list of the next upper level with matching MVs up to prefix i, and with the $(i+1)$th digit of the upper M equaling 0. Likewise, $upper1Predecessor$ and $upper1Successor$ link to nodes in the list of next upper level with matching MVs up to prefix i, with the $(i+1)$th digit of upper MV equaling 1. Like the rule in Section III-A, these

upper links are not constructed when a same-level node that is nearer than the nodes of the upper link destinations.

Each node at level $(i + 1)$ has two lower links. $lowerPredecessor$ and $lowerSuccessor$ link to nodes in the list of the next lower level with matching MVs up to prefix i. Like the rule in Section III-A, these lower links are not constructed when a same-level node is nearer than the nodes of the lower link destinations.

In Ballistic Skip Graph, the maximum number of entries in the routing table of any node is 10.

Routing is performed as described in the same way as Section III-A, except that when there are two upward links, we select the link closest to the target node.

Because the overlay is constructed in an autonomous decentralized manner by each node, it must be correctly constructed even when multiple nodes join and leave simultaneously. Also, given that a node might suddenly leave due to failure or a similar event, our method must allow nodes to leave without giving special notice to the surroundings. Bidirectional lists such as the level-zero and level lists maintain their structure by adopting distributed doubly linked lists, as described in [8]. this structure is robust even in environments where nodes join and leave frequently. To maintain the structure of the upper and lower links, each node regularly updates its correct lower links and the corresponding upper links are simultaneously updated.

1) Join of a node: To join the network, a participating node links to a known node already joined in the network. The already-joined node searches its $predecessor$ and $successor$ for the participating node by the normal routing algorithm. The participating node then searches its own level list, gradually raising the level finding adjacent nodes. In the node-insertion algorithm (see Fig. 5), the appropriate upper link of X is denoted by $X.upper$. Although the level list is searches separately in the predecessor and successor directions, we explain only the successor operation here (the predecessor operation is identical, with $successor$ replaced by $predecessor$). Immediately after joining, the participating node traces its successor through the level-zero list to discover a node at its own level or lower. Stochastically, half of all nodes are at level 1, so the target node is found in approximately two steps. When the discovered and participating node are at the same level, the participating node joins the level list of the discovered node.

If the level of the discovered node is lower than the level of the participating node, the participating node makes that node a temporary lower link destination. By updating this temporary lower link destination, the participating node finds the formal downlink destination. During the update, the participating node traces the level list, seeking an upper link that satisfies the three conditions of the 10th line of Fig 5. Because a qualifying node is usually found after approximately for steps, and temporary lower link is updated less than its own level times, the update operation requires $O(\log N)$ steps. Corresponding to the lower link constructed from the upper-level node, an upper-level link for the participating node is constructed as follow. The upper-level node constructs a lower link and sends an instruction to build an upper link to the linked node. The link destination node checks its own routing table, confirms that the requisite condition is satisfied, and updates the upper link. Each node periodically updates its lower links.A lower link to a node is regarded as a temporary lower link, and is updated by the above operation. When the node does not have a lower link, its searches for a lower link in its own level list. After tracing the list of lower-link destinations, the node confirms whether any node can become its lower-link destination.

```
 1: X ⇐ U.successor
 2: while X.level ≥ U.level do
 3:     X ⇐ X.successor
 4: end while
 5: if X.level = U.level then
 6:     U.levelSuccessor ⇐ X
 7: else
 8:     TEMP ⇐ X
 9:     while X ≠ NULL do
10:         if X.upper.key > U.key
            and X.upper.level ≤ U.level
            and X.upper.MV ⌈ U.level = U.MV ⌈ U.level then
11:             if X.upper.level = U.level then
12:                 U.levelSuccessor ⇐ X.upper
13:                 U.lowerSuccessor ⇐ TEMP
14:                 break
15:             else
16:                 TEMP ⇐ X.upper
17:                 X ⇐ TEMP
18:             end if
19:         else
20:             X ⇐ X.successor
21:         end if
22:     end while
23:     U.lowerSuccessor ⇐ TEMP
24: end if
```

Fig. 5. Algorithm for inserting a node U

2) Leaving of a node: A node can leave without any special notification to the surrounding nodes. Therefore, even if a node fails, it can be deleted by usual leave operation. Each node judges that a node has left if there is no reply from the destination node after a certain period of time. As the upper and lower links are periodically rebuilt to ensure appropriate connections, if the node of the upper or lower link destination is detected as a leave, its link is simply removed.

If the adjacent node in the level list is detected as leave, the level list can be corrected by searching the nearest node on the opposite side of the leaving node. That node is detected in the lower-level list for list levels-two or higher, and in the upper level list for list levels 0 and level 1.

Fig. 6 shows the algorithm that deletes U from the level list and corrects the level list when node D belongs to a level-two or higher list, and when U which is $D.levelSuccessor$. The appropriate upper and lower links of D are denoted as $D.upper$ and $D.lower$,respectively. The algorithm that deletes $D.levelPredecessor$ is identical, but replaces $successor$ with $predecessor$; moreover, the algorithm that deletes a node from level 0 or 1 replaces of the lower list by the upper list.

1: $X \Leftarrow D.levelPredecessor$
2: **while** $X.lower = NULL$ **do**
3: $\quad X \Leftarrow X.levelPredecessor$
4: **end while**
5: $X \Leftarrow X.lower$
6: **while** $X.upper.key < U.key$ **do**
7: $\quad X \Leftarrow X.levelSuccessor$
8: **end while**
9: $X \Leftarrow X.upper$
10: **while** $X.predecessor = NULL$ **do**
11: $\quad X \Leftarrow X.levelPredecessor$
12: **end while**
13: $X.levelPredecessor \Leftarrow D$
14: $D.levelSuccessor \Leftarrow X$

Fig. 6. Algorithm for deleting a node U

IV. Evaluation

In the evaluation experiment, the proposed method Ballistic Skip Graph was simulated on the overlay network construction toolkit Overlay Weaver [9], [10]. As a comparison target, we prepared and simulated the normal skip graph in the same environment. The average routing table sizes and average routing lengths of the algorithms were compared with their theoretical values in the Rainbow Skip Graph, which assumes ideal configuration of the supernode (i.e., a supernode with $\log N$ nodes). Measurements were performed after all nodes had participated and the network was constructed. The experimental environment was as follows.

- Overlay Weaver 0.10.5
- OS : Mac OS X 10.10.5
- CPU : Intel Core i5-4260U 1.4GHz
- Java : Java SE 8 Update 45

A. Comparison of average routing table size

To relate the number of nodes participating in the network to the routing table size, we varied the number of nodes in the experiment and measured the average routing table size at each node number.

1) Average routing table size of Rainbow Skip Graph: The theoretical routing table size was calculated as the estimated total number of links in Rainbow Skip Graph divided by the number of nodes in the network. When the total number of nodes is N, Rainbow Skip Graph contains $2N$ links for building a bidirectional list arranged in the key order. In the ideal state, Rainbow Skip Graph requires additional links for constructing a skip graph of $\frac{N}{\log N}$ supernodes, and bidirectional links that connect the up and down nodes in each supernode. As the nodes responsible for each level of the skip graph (excluding the highest and zero levels) have the same number of up and down links as the level links, the number of upper and lower links is estimated by subtracting $2\frac{N}{\log N}$ from the number of links in the constructed skip graph. Denoting by S_L the number of links required to construct an $\frac{N}{\log N}$-node skip graph, the total number of links in Rainbow Skip Graph is $2S_L - 2\frac{N}{\log N} + 2N$. We measured S_L in the simulation and thereby calculated the average routing table size of Rainbow Skip Graph.

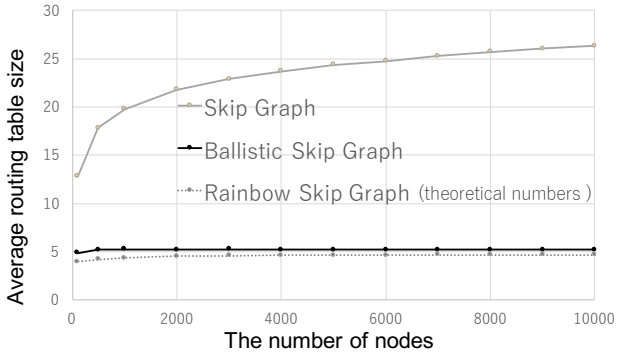

Fig. 7. Average routing table sizes of the compared graph algorithms

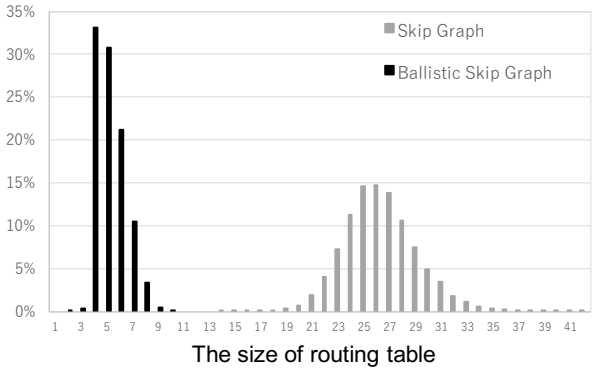

Fig. 8. Distribution of the routing table size

2) Experimental result: The relationship between the node number and the average routing table size is shown in Fig. 7.

The average routing table size of the ballistic skip graph was approximately 5.3, regardless of the number of participating nodes. This result confirms that the average routing table size is O (1), whereas that of the skip graph grows as $O(\log N)$. Moreover, the results of the ballistic and rainbow skip graphs are comparable.

B. Comparison of routing table size distribution

In this experiment, the distributions of routing table sizes were measured in 1000-node network. The routing table size distributions of the normal and ballistic skip graphs are shown in Fig. 8. The routing table size distribution was narrower in Ballistic Skip Graph than in the standard skip graph and was concentrated on small values. In Ballistic Skip Graph, the routing tables of over 60% of the nodes were sized 5 or less. A few nodes had a size10 routing table. Small routing tables dominate in this graph because half of the nodes are level-one nodes, which possess no lower links. Moreover, many nodes in the uppermost levels participate in single-node level lists, which contain no same-level links.

C. Comparison of average path length

To relate the number of nodes to the path length, we varied the number of nodes and measured the average path length at

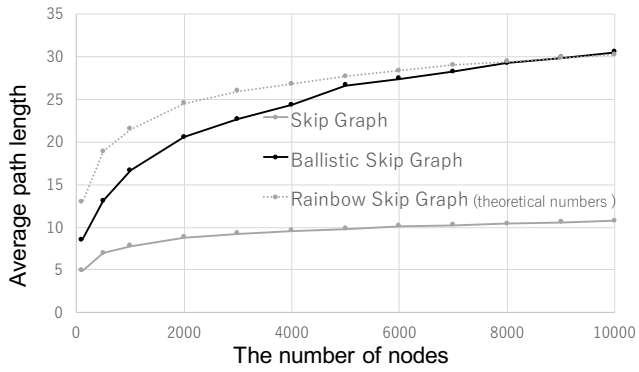

Fig. 9. Average path length

each node number. Each node repeatedly searched for random keys, and the path lengths were averaged 20 trials.

1) Average path length of Rainbow Skip Graph: We consider the number of routing hops in Rainbow Skip Graph with N nodes. In routing, messages are first sent upward from the start node to the highest-level node in the supernode by referencing the bidirectional list connecting the upper and lower levels. Because messages pass through half of the nodes in the supernode, the number of hops is $\frac{1}{2}\log N$. After reaching the highest level, routing is performed in a skip graph constructed from $\frac{N}{\log N}$ supernodes. However, unlike normal routing, messages are passed to lower levels within the same supernode. Assuming that S_P is the path length of an $\frac{N}{\log N}$-node skip graph of supernodes, the highest level of the skip graph should be $\log \frac{N}{\log N}$. therefore, this operation is performed in $S_P + \log \frac{N}{\log N}$ hops. When the supernode containing the target key is reached, the route follows the nodes by their key order in the supernode until it reaches the target key. This behavior is expected to require $\frac{1}{2}\log N$ hops on an average. Therefore, the average path length of Rainbow Skip Graph is $S_P + \log \frac{N}{\log N} + \log N$.

2) Experimental result: The relationship between node number and average path length is shown in Fig. 9. The path length was longer in Ballistic Skip Graph than in skip graph. However, routing was more efficient in the proposed method than in Rainbow Sip Graph when the nodes numbered less than 9000. Although the efficiencies were reversed in graphs with more than 9000 nodes, the efficiency of the ballistic skip graph did not worsen as exponential order.

We now consider the theoretical path length of Ballistic Skip Graph If the level-zero link is not used in routing, the route length is $O(\log N)$, which is ideally short because the appropriate link level can always be used after the level rises. However, when using a level-zero link, current node level change to lower than appropriate link level, so the appropriate level is reached after $O(\log N)$ hops. Since this appropriate level is expected to be lower than the appropriate level before using the level 0 link, the level will re-rise only $O(\log N)$ times in the worst-case scenario. Therefore, in the worst case, the path length is considered as $O((\log N)^2)$. From the above discussion, the expected path length of Ballistic Skip Graph is between $O(\log N)$ and $O((\log N)^2)$, consistent with the order of path length determined in the experiment.

V. Summary and future work

This paper proposed a new structured overlay known as Ballistic Skip Graph. The proposed graph supports range queries by key, as in skip graph. However ,although the nodes of a skip graph have multiple levels of shortcut links, Ballistic Skip Graph determines the responsible level of each node as one, which reduces the routing table size to $O(1)$. Ballistic Skip Graph negates the need for grouping the nodes and estimating the total number of nodes, which greatly suppresses the influence of nodes joining and leaving.

As confirmed in the evaluation experiment, the routing table size of Ballistic Skip Graph was suppressed to a small constant. The average path lengths of Ballistic Skip Graph were then compared with those of skip graph and the theoretical values of Rainbow Skip Graph.

In future work, we will determine the appropriate load distribution of the graph. In the current structure, the number of lists doubles with each increment of the level. In practice, we verify the load of each link and consider the most suitable number of lists.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Numbers 2570008 and 16K12406.

This work was supported by New Energy and Industrial Technology Development Organization (NEDO).

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," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 17–32, 2003.

[2] J. Aspnes and G. Shah, "Skip Graphs," *ACM Trans. Algorithms*, vol. 3, no. 4, Nov. 2007.

[3] M. F. Kaashoek and D. R. Karger, "Koorde: A simple degree-optimal distributed hash table," in *International Workshop on Peer-to-Peer Systems*. Springer, 2003, pp. 98–107.

[4] K. C. Zatloukal and N. J. A. Harvey, "Family Trees: An Ordered Dictionary with Optimal Congestion, Locality, Degree, and Search Time," in *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2004, pp. 308–317.

[5] M. T. Goodrich, M. J. Nelson, and J. Z. Sun, "The rainbow skip graph: a fault-tolerant constant-degree distributed data structure," in *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, 2006, pp. 384–393.

[6] W. Pugh, "Skip lists: a probabilistic alternative to balanced trees," *Communications of the ACM*, pp. 668–676, 1990.

[7] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: A scalable and dynamic emulation of the butterfly," in *Proceedings of the twenty-first annual ACM symposium on Principles of distributed computing*, 2002, pp. 183–192.

[8] K. Abe and M. Yoshida, "Constructing distributed doubly linked lists without distributed locking," in *2015 IEEE International Conference on Peer-to-Peer Computing (P2P)*, Sep. 2015, pp. 1–10.

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

[10] K. Shudo, "Overlay Weaver," http://overlayweaver.sourceforge.net, 2006.