# Supporting $k$-Nearest Service Discoveries for Large-Scale Edge Computing Environments

Yuuichi Teranishi, Takashi Kimata, Hiroaki Yamanaka, Eiji Kawai, Hiroaki Harai
National Institute of Information and Communications Technology,
4-2-1, Nukui-Kitamachi, Koganei, Tokyo 184-8795, Japan

***Abstract*—This paper presents an overlay network protocol called "LASK: Locality-Aware Service discovery protocol for K-nearest search", for supporting scalable and locality-aware distributed $k$-Nearest Service Discovery (kNSD). The kNSD provides a lookup function to find $k$ service nodes located close to the requester. Importance of such function increases under the situation of growth in size and the diversity of the resources in the so-called edge computing environments. In LASK, efficient cooperative routing algorithm is implemented utilizing the two-dimensional structure of key-order preserving structured overlay networks that are constructed according to the network topology and the name of the service. This structure achieves scalable and locality-aware routing of the lookup messages to the matched nodes avoiding redundant data transmissions across the edge networks. Extensive simulation evaluations show that LASK could achieve small discovery latency in the large-scale edge computing environments; a typical kNSD took less than 5 ms in a data center network model and less than 100 ms in the Internet model even when there are 100,000 nodes in the entire network.**

## I. Introduction

The edge computing architecture has been considered as a next-generation network computing paradigm [1][2]. In the edge computing architecture, *edge computers* executes tasks instead of cloud servers. The edge computer can be defined as any computing resource that connects to the network, physically located around the path between data sources and cloud data centers. For example, a smartphone is an edge computer for wearable things, a gateway in a smart home is an edge computer for home sensors, a cloudlet [3] is an edge computer for mobile devices/vehicles, and so on. Each edge computer can offload the computing tasks to the other (more powerful) edge computers or cloud servers. The edge computing architecture enables applications to have a shorter response time, reduces the computation load on devices, and localizes the data processing. These properties are suitable for the real-time and context-aware Internet of Things (IoT) applications. The number of edge computing resources is expected to be over a million in the world [4].

The offloaded tasks from a real-time IoT application often require parallel executions to be finished within a short period [5]. For such tasks, multiple edge computers must be allocated for the parallel executions. The multiple allocations are also required to achieve k-out-of-n redundancy [6] for a fault-tolerant system. The same functionality is needed to form a group of small-latency edge computers for collaborative executions of real-time applications. A typical use case of such group forming is making a party in an interactive P2P online-game. The applications need to find the edge computers that satisfy requirements (e.g. communication latency, network/storage capacity, running process, etc.) for the task. In this case, the *service discovery* plays an important role for efficient executions of tasks under the situation of growth in the size and the diversity of the edge computer resources.

In this paper, we propose a service discovery function called *$k$-Nearest Service Discovery (kNSD)*, which is suitable for the above-mentioned situations. The kNSD finds $k$ nearest services which match to the request. The 'nearest' means that the physical network distance in the network is smallest. $k = 1$ finds the nearest service and $k > 1$ finds multiple services.

A straightforward approach to implement kNSD is to use a centralized registry. This type of approach corresponds to the classical client/server architecture, where an increase in the number of services brings the system to saturation or failure. Moreover, the physical distance to the centralized data center located at a remote place causes longer turn around time of request and response for a service discovery, which is not preferable for real-time applications. Though some implementations can form a cluster of servers to achieve scalability, they require relatively complex parameter settings by hands for each server. Such complexity of configuration does not work in the edge computing environment where the resources are managed autonomously.

To achieve scalability and self-configuration features of service discovery, many distributed approaches with no centralized registry have been proposed [7][8][9]. These approaches treat distributed nodes that provide/require services as peers in a peer-to-peer system, and construct an overlay network. However, none of the existing approaches could directly support kNSD. Simply applying a distributed approach breaks the routing localities to achieve a small latency for a discovery. Our goal is to implement a distributed kNSD to cope with these issues.

In this paper, we present a novel distributed kNSD scheme for edge computing architecture. The contributions of the paper are as follows:

- Defines a name-based kNSD on the basis of a hierarchical network model that models the edge computing environments.
- Proposes a scalable and locality-aware protocol for the kNSD called "LASK" (an acronym of Locality-Aware Service discovery protocol for K-nearest search), which utilizes two-dimensional key-order preserving structured overlay networks (KOPSONs).
- Evaluates the feasibility of LASK by extensive simulations using two practical scenarios (data center scenario and Internet scenario) assuming up to 100,000 edge computers.

As far as the authors know, this is the first work that proposes a distributed kNSD protocol.

978-1-5386-4727-1/18/$31.00 ©2018 IEEE

## II. A $k$-Nearest Service Discovery in Edge Computing Environments

### A. Assumed edge computing environment

First, we define the target edge computing environment assumed for the kNSD in this paper. An example structure of the edge computing network model is shown in Fig. 1. The edge computing environment on the Internet has natural hierarchical properties such as the access networks, the Internet service providers' networks, and the wide-area networks between countries and continents. The proprietary edge computing environment corresponds to the distributed data center network which can also be modeled as a hierarchical network with the local area network in a data center and the wide-area network among data centers.

The edge computers (hereafter, *nodes* in the network model) can belong to any subnetwork in the hierarchy. The unit of the subnetwork depends on the assumed network model, such as Autonomous System (AS) of the Internet, an urban-area network, a LAN of the organization, and a data center facility. We call the top layer in the hierarchy layer 0, which contains the entire network.

Each subnetwork in each layer has an *identifier*. The identifier is denoted as $I_{l,u}$, where $l$ is the layer in the hierarchy and $u$ is the unique value in the layer $l$, e.g., a sequence number in each layer. The network identifier of a node can be denoted as an ordered list of the identifiers. For instance, the leftmost subnetwork in the layer 2 in Fig. 1 is denoted as $\{I_{0,0}, I_{1,0}, I_{2,0}\}$. The length of the network identifier is not always the same for all nodes. In the figure, four nodes in the middle have the network identifier $\{I_{0,0}, I_{1,1}\}$. Only the leaf subnetwork accommodates nodes. We can say the network identifier represents the location of a node. We assume that each node can obtain its network identifier by the network infrastructure. This assumption is practical in the software-defined network infrastructures [10], [11].

We define the order of network identifier as the lexicographical order $<_{\text{NETID}}$, formally,

$$\{I_{0,a_0}, I_{1,a_1}, \ldots, I_{H-1,a_{H-1}}\} <_{\text{NETID}} \{I_{0,b_0}, I_{1,b_1}, \ldots, I_{H-1,b_{H-1}}\} \iff a_0 < b_0 \vee \{(\exists m > 0)(\forall i < m)(a_i = b_i) \wedge (a_m < b_m)\}$$

where $H$ corresponds to the height of the network hierarchy.

We also define a notion of *network distance*. The network distance $D(x, y)$ between two network identifiers $x$ and $y$ can be calculated as follows:

$$D(x, y) = max(|x.net\_id|, |y.net\_id|) - |p(x.net\_id, y\_net\_id)|$$

where $x.net\_id$ denotes the network identifier of $x$ and $p$ is a function to pick up the common prefix in the two network identifiers. For example, $D(\{I_{0,0}, I_{1,1}\}, \{I_{0,0}, I_{1,1}\})$ is 0 and $D(\{I_{0,0}, I_{1,2}, I_{2,0}\}, \{I_{0,0}, I_{1,1}\})$ is 2. The network distance corresponds to the number of hops among subnetworks. This distance definition stands on an assumption that the latency between the nodes in the same subnetwork is ignorable.

### B. The definition of $k$-Nearest Service Discovery

The kNSD which we assume in this paper is defined as follows.

> Given a set of all nodes in the entire network $N$, a query for the name $q$, and a query requester $p$, kNSD finds a set of nodes $S$ such that $|S| = k$ and for any $s \in S$ and $s' \in V - S$, $D(p.net\_id, s.net\_id) \leq D(p.net\_id, s'.net\_id))$, where $V = \{v | sv(v) = q, v \in N\}$ and $sv(v)$ is the the name of the service that the $v$ provides.

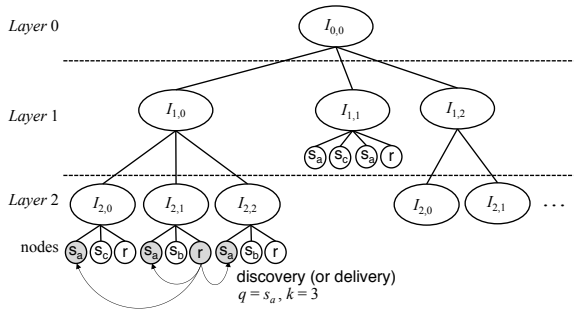


Fig. 1. An example of the network model and kNSD

The kNSD definition above is a name-based service discovery. The name-based queries can be extended to the multi-attribute queries by using the name as an index. Obtaining a set of nodes $S$ corresponds to getting an endpoint list of the target nodes. For simplicity, we assume one node corresponds to one service. We also assume that the kNSD also accepts the cases where $|S| < k$ when not enough number of nodes are discovered within a specified timeout $T$. Fig.1 shows a case where a requester node $r$ requests $3 (= k)$ services named $s_a$.

kNSD can deliver data to the $k$-nearest services directly instead of obtaining an endpoint of the target node. We call this mode *delivery mode*. The delivery mode kNSD supports to treat lease-based resources by delivering lease request to the $k$-nearest services and reserve them. The delivery mode kNSD also supports an application class of small event-driven data processing, in which the data needs to be delivered to the $k$-nearest services only once.

We define the requirements of kNSD in the edge computing environment as follows:

- Minimize the discovery/delivery latency
  The applications assumed in the edge computing requires low latency. To catch up with the dynamic situation changes, the latency to find the matched $k$ services must be as small as possible. In the delivery mode, the data needs to be received as quickly as possible.
- Maximize the number of successful discovery/delivery before timeout
  A time-critical application should be able to specify a service discovery query response timeout $T$. In the delivery mode, the data needs to be received on the $k$-nearest services within the period $T$. In such cases, the implementation needs to maximize the number of services discovered or delivered within $T$, where $T$ varies by the application demand.

### C. Related Work

There have been many studies and systems for service discovery such as Bluetooth SDP, SLP, Bonjour, Salutation, etc. $k$-nearest neighbor query processing in an ad-hoc network is also proposed [12]. Though such an approach works for small-scale networks such as wireless sensor networks, it is

hard to apply to the global network where up to millions-billions of nodes connect to.

As a distributed name-based distributed service discovery scheme for global discovery scope, Distributed Hash Table (DHT) supporting key-value store is a promising approach [13][14]. However, this approach constructs overlay networks for discovery that do not consider physical topology. In this case, inefficient and redundant message exchanges arise, especially when there are a huge number of nodes. For example, a message from a requester node may be forwarded via other networks to a service node, even when the requester node and the service node are located on the same network.

There are some extensions of DHTs to achieve the routing localities such as extensions for information-centric networks [15], hierarchical structured overlay construction schemes [16], etc. To support kNSD, one approach is to use the name of the service as a key and the list of endpoints of target nodes as values on such locality-aware DHTs. Then, the lookup process selects the $k$-nearest nodes from that list. However, if a large number of node registers the same name, which is often the case in edge computing environment where processes are offloaded to edge servers, the service registration load or communication load are concentrated on a node that holds the corresponding name as a key.

As a distributed name-based routing scheme, topic-based pub/sub messaging schemes are applicable for kNSD to deliver messages to the nodes that have keys of the same name. Among such schemes, STLA [17] is a locality-aware topic-based pub/sub message routing scheme which utilizes Skip Graph [18] as a base overlay network. For an efficient message delivery, STLA sorts all subscribers on the overlay network by the topic name and the network identifier. If the publisher knows the topic used in pub/sub beforehand, the messages in STLA are delivered preserving the locality. However, since it is impossible to know the topic used in applications, STLA causes redundant message forwarding across the subnetworks for the initial message delivery for a topic.

Another approach to support kNSD is to find services by their locations using a location-based overlay. There have been many studies for location-based overlays and service discoveries based on them [9][19]. This approach can perform location-based searches with routing locality. However, such location-based overlays require geographical coordinates. Such coordinate-based discovery is rather too complex for the network model we assume for kNSD where a network device can be a node.In addition, they do not support $k$-nearest discoveries but support queries for a particular geographical region. If no service was located in the specified region, the discovery fails. ZZW [20], which is a similarity search scheme on an overlay network, is also applicable for kNSD if the network identifier is used as a similarity metric. ZZW looks up the nodes in order of the closeness of a key, node by node. Though such approach suppresses the traffic for nearest searches, it can cause a long latency if there are a large number of nodes in a subnetwork.

## III. LASK

In this paper, we propose a new kNSD scheme called LASK. LASK assumes KOPSON as a base algorithm of the distributed discovery. As a KOPSON, we assume the structure of multiple levels of bi-directional links. First, we describe the overview of the assumed bi-directional KOPSON used in LASK.

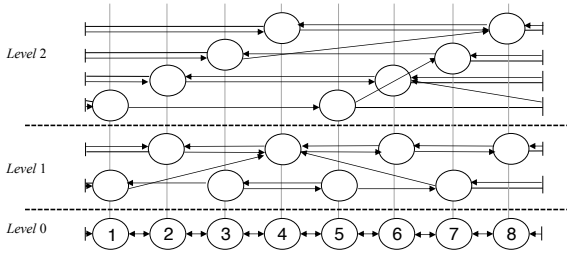


Fig. 2. An example structure of bi-directional KOPSON

### A. Bi-directional KOPSON

Fig. 2 shows an example structure of KOPSON with bi-directional skip links. Skip Graph [18] and its variants are of this type of KOPSON. Chord# [21] is a uni-directional KOPSON, but it is easily extended to bi-directional. Donut [22] has a similar structure to the bi-directional Chord#. The level 0 becomes a doubly-linked list for each node, ordered by their keys with a ring structure (the smallest key is larger than the largest key, and vice versa).In this figure, the keys have the integer type (1-8). In KOPSON, when the keys of two nodes are adjacent, these two nodes become neighbors in the overlay network. The links in higher than level 0 are used as skip links for lookups. LASK does not assume that the links in the level 1 or above are symmetric, i.e. when a node $p$ pointed by a node $q$, $p$ does not need to have a pointer to $q$ (like Fig. 2). A scalable lookup of a specified key is achieved by a greedy routing using all levels of links. The nodes forward the message cooperatively to the target node that matches to the key. The number of hops required to lookup a node is $O(\log\ N)$ when $N$ nodes are joined to the overlay network. LASK assumes a bi-directional greedy routing, in which the query can be forwarded to the left or right. Implementations of KOPSONs have been put to practical use in recent years.

### B. The overview of LASK

The principle idea of LASK is to use two KOPSONs for routing to the nearest service and $k$ services. By using KOPSON, the communication loads are balanced among nodes and the node failures are recovered in a self-organized manner, while the distributed nodes only need to conform to the protocol of KOPSON.

LASK uses two KOPSONs for two-dimensions: The one for the location-name (LN) axis and the other for the name-location (NL) axis. The LN axis is used for lookup the nearest node that matches to the query. The NL axis is used to lookup $k-1$ nodes (rest-of the specified $k$ nodes) that match to the query in order of the closeness in network distance. LASK can perform scalable discovery because it searches a target node using the structure of KOPSON on subnetworks on LN axis. In addition, the locality of the routing is preserved by the bi-directional routings in LN axis and NL axis.

Fig. 3 shows the interaction between a node $p$ and both axises for a node registration. The sequence of the registration is as follows.

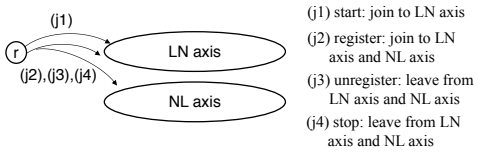


Fig. 3. Registration sequence of LASK

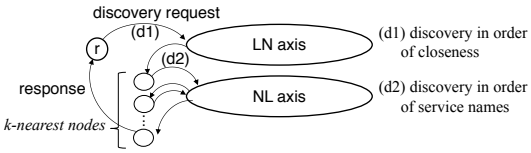


Fig. 4. Discovery sequence of LASK

(j1) When $p$ starts running, $p$ joins to LN axis using its network identifier. If $p$ registers a service from start time, this process can be skipped.

(j2) When $p$ registers a service, $p$ joins to LN axis and newly joins to NL axis, using the network identifier of $p$ and the name of service.

(j3) When $p$ unregisters the service, $p$ leaves from both LN and NL axis for the keys with the name of service. If $p$ stops running, this process can be skipped.

(j4) When p stops running, p leaves from LN and NL axis.

Fig. 4 shows an overview of the discovery sequence of LASK. The sequence of discovery follows:

(d1) When a requester issues a discovery query for a name, the query is forwarded to LN axis to the node that has a key matches to the name with the nearest network identifier.

(d2) After finding the nearest node, the message is forwarded to the node with the next closest node from the requester on NL axis. The forwarding is repeated until $k$ nodes are visited on NL axis.

### C. The keys in LASK

LN axis uses the location-name key (LNK). LNK is denoted as follows:

$$<net\_id, name, unique\_id>$$

The $net\_id$ is the network identifier of the node. The $name$ is the name of the service that the node provides. A node which corresponds to a requester has an empty name. The $unique\_id$ is the unique identifier of the node, which is unique to each node. For example, a hash value of the IP address and sequence number can be a unique identifier. The $name$ and $unique\_id$ are string type values that have an alphabetical order. The order of LNK is decided by the lexicographical order of all elements.

NL axis uses the name-location key (NLK). NLK is denoted as follows:

$$<name, net\_id, unique\_id>$$

The order of the name and net_id is switched compared to LNK.

**Algorithm 1:** LASK protocol of LN axis on node $u$

```
1  upon receiving ⟨LASK_LN, q⟩
2  begin
3      if (q.mode = SEARCH) then
4          if (q.name = u.name) then
5              q ← {from : q.from, k : q.k, name : q.name,
                    left : u, right : u, results : φ};
6              send(u, LASK_NL, q);
7          else
8              q.mode ← MOVE;
9              if (q.dir = RIGHT) then
10                 q.key ← <u.net_id, u.name, ∞>;
11             else
12                 q.key ← <u.net_id, u.name, −∞>;
13     else if (q.mode = MOVE) then
14         if (D(q.from.net_id, q.right.net_id) ≥
               D(q.from.net_id, q.left.net_id)) then
15             if (q.dir = RIGHT) then
16                 q.right ← u.LN.next.net_id;
17                 q.dir ← LEFT;
18                 q.key ← <q.left.net_id, q.name, ∞>;
19             else
20                 q.key ← <u.LN.next.net_id, q.name, ∞>;
21         else
22             if (q.dir = LEFT) then
23                 q.left ← u.LN.prev.net_id;
24                 q.dir ← RIGHT;
25                 q.key ← <q.left.net_id, q.name, ∞>;
26             else
27                 q.key ← <u.LN.prev.net_id, q.name, ∞>;
28         q.mode ← SEARCH;
29     u.LN.route(q.key, LASK_LN, q);
```

### D. Algorithm

Algorithm 1 and Algorithm 2 show the pseudo code for the protocol of LASK on LN axis and NL axis. The protocol on LN axis has two modes: SEARCH mode and MOVE mode. At the initial state, a requester node $r$ starts a discovery process of $k$-nodes which match to $name$ as follows:

$$q \leftarrow \{\texttt{name} : name, \texttt{k} : k, \texttt{mode} : \texttt{SEARCH}, \texttt{dir} : \texttt{RIGHT}, \texttt{from} : r, \texttt{right} : r, \texttt{left} : r\};$$
$$r.LN.\texttt{route}(<r.net\_id, name, \infty>, q);$$

where $r.LN$.route is a greedy routing function which forwards the message to the specified LNK with a payload. In this case, the payload is a query $q$ with some properties. The direction of the bi-directional greedy routing is specified as $q.dir$ which contains RIGHT or LEFT. The discovery is started on LN axis with the SEARCH mode.

In the SEARCH mode, if the specified name matches to the LNK of the node, the query continues on NL axis (line 6). Otherwise, the node did not match the query, the query shifts its state to MOVE mode (line 8). The query is forwarded to the edge of the subnetwork. To move to the edge of the network, the maximum identifier($\infty$) (line 10) or the minimum identifier($-\infty$) (line 12) is assigned to the next key for routing at line 29. In the MOVE mode, the query is forwarded to the right neighbor subnetwork or the left neighbor subnetwork. The direction is not changed as long

**Algorithm 2:** LASK protocol of NL axis on node $u$

```
1  upon receiving ⟨LASK_NL, q⟩
2  begin
3      if (q.name = u.name) then
4          q.results ← q.results ∪ {u};
5      if (q.name ≠ u.name or |q.results| = q.k) then
6          send(q.from, LASK_RESULT, q.results);
7      if (q.dir = RIGHT and u.NL.next.net_id <NETID
         u.net_id) then
8          q.right ← nil;
9      if (q.dir = LEFT and
10         u.NL.prev.net_id <NETID u.net_id) then
11         q.left ← nil;
12     if (q.right = nil and q.left = nil) then
13         send(q.from, LASK_RESULT, q.results);
14     if (q.dir = RIGHT) then
15         if (q.right = nil or q.left ≠ nil and
             D(q.from.net_id, u.next.net_id) >
             D(q.from.net_id, q.left.net_id)) then
16             q.right ← u.NL.next;
17             q.dir ← LEFT;
18             send(q.left, LASK_NL, q.results);
19         else
20             send(q.NL.next, LASK_NL, q.results);
21     else
22         if (q.left = nil or q.right ≠ nil and
             D(q.from.net_id, u.NL.prev.net_id) >
             D(q.from.net_id, q.right.net_id)) then
23             q.left ← u.NL.prev;
24             q.dir ← RIGHT;
25             send(q.right, LASK_NL, q.results);
26         else
27             send(q.NL.prev, LASK_NL, q.results);
```

as the distance to the next neighbor subnetwork is closer than the neighbor subnetwork in another direction. For simplicity, the algorithm does not show the discovery timeout handling. If the query exceeds the specified timeout $T$, the query needs to stop forwarding. The SEARCH and MOVE are scalable by KOPSON even when there are a large number of nodes in a subnetwork.

In the Algorithm 2, the query is forwarded node by node to the next right node or next left node, as long as the name of NLK matches. If the name does not match to the NLK or the number of matched nodes equals $k$, the query finishes (lines 5, 6). The query is forwarded to the right direction or left direction, using the level 0 links on KOPSON (lines 14-27). The direction is not changed as long as the distance to the next node is closer than the node in another direction.

Fig. 5 shows an example case which searches two services with the name $s_c$ from a requester node $r$.

### E. Analysis

In the following, we analyze the performance of LASK for the network which has two layers of hierarchy. The network with more layers can be analyzed by analogy. We denote the number of nodes in the entire network as $N$ and the average number of nodes in an edge subnetwork as $E$. Here we assume the number of nodes is uniformly distributed in the subnetworks. We also denote $L_E$ as the latency between

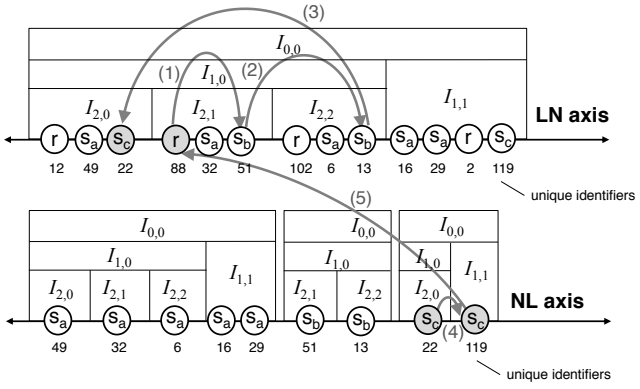


Fig. 5. Routing example of LASK. The search target is $s_c$ and $k = 2$. The query routing starts from $r$ is forwarded in the order (1) to (5).

the nodes across the subnetworks and $L_e$ as the latency between nodes inside a subnetwork. For the latency analysis, we assume that the KOPSON is at the converged (or ideal) state where the number of hops to the node with the distance $d$ becomes $\log_2 d$.

In LASK, the expected latency to reach to the nearest node on LN axis is $L_e \log_2 E$, if the target node exists inside the same edge network. We can expect $L_e \log_2 E$ as a small value because $L_e$ is the network latency between nodes inside a subnetwork. The worst case latency of LASK is when the target node locates at the furthest of the network. In this case, the latency is $\frac{N}{E}(L_E + L_e \log_2 E) + L_e \log_2 E$, which means all subnetworks are visited and searched. However, the worst case is avoided by setting the timeout $T$. For $k$ nodes traversal in NL axis, the latency is $kL_e$ if all $k$ nodes exist inside the same edge network.

Table I shows the comparison with the other existing schemes to perform kNSD. The table shows the comparison with the cases when kNSD is implemented using STLA [17] and ZZW [20]. Even if the target node exists inside the same edge network with the requester, the expected latency of STLA becomes $L_E \log_2 \frac{N(1-R/2)}{2E} + L_e \log_2 E$, where the rate of the requester is denoted as $R$. $\frac{N(1-R/2)}{2}$ is the average number of nodes between the requester node and the target node on STLA. The worst case for the STLA is when the target node locates at the furthest subnetwork from the requester. In this case, the latency of STLA is $L_E \log_2 \frac{N}{E} + L_e \log_2 E$. By using ZZW, the expected maximum latency when the target node exists inside the same edge network becomes $L_e E$. This means the larger $E$, the longer the latency. The worst case of ZZW is also when the target node locates at the furthest subnetwork. In this case, the latency is $\frac{L_E N}{E} + L_e N$. About the $k$ nodes traversal, ZZW requires at most $ELe$ to traverse $k$ nodes because it forwards messages one by one. Though LASK uses two-dimensional KOPSONs, the join overhead is $O(\log_2 N)$, which is the same order of complexity with the other schemes.

## IV. Evaluations

We implemented a simulator of LASK on PIAX [23] using bi-directional Chord# as a KOPSON. We evaluated the performance of discovery latency of LASK. The discovery

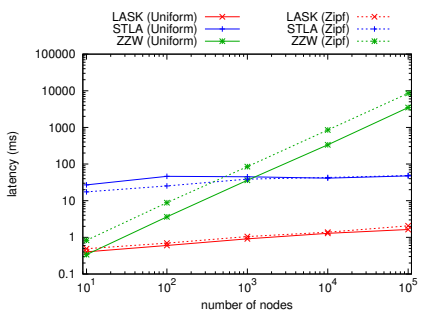


Fig. 6. Average latency by changing # of nodes (DC scenario)

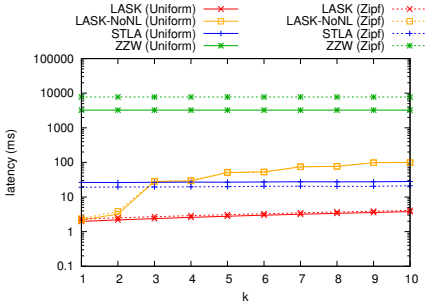


Fig. 7. Average latency by changing $k$ for 100,000 nodes (DC scenario)

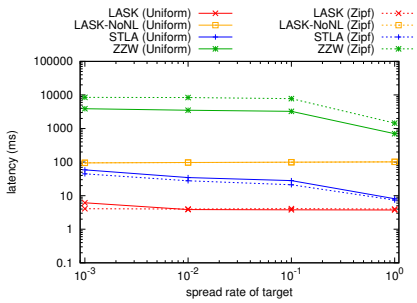


Fig. 8. Average latency by changing service spread rates for 100,000 nodes (DC scenario)

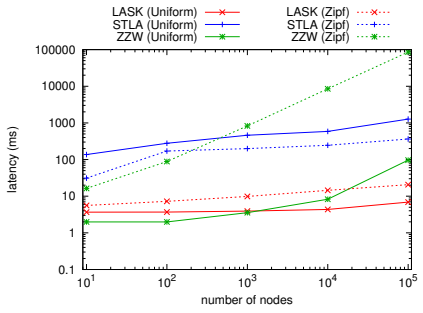


Fig. 9. Average latency by changing # of nodes (Internet scenario)

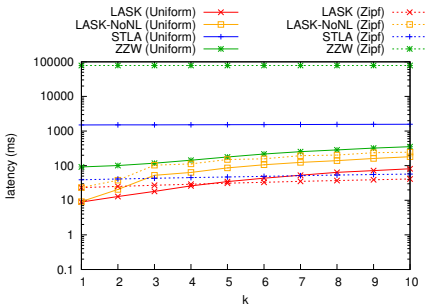


Fig. 10. Average latency by changing $k$ for 100,000 nodes (Internet scenario)

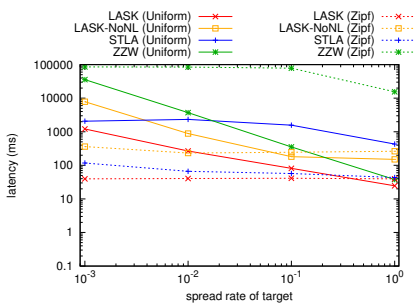


Fig. 11. Average latency by changing service spread rates for 100,000 nodes (Internet scenario)

TABLE I
COMPARISON OF THE SCHEMES

| | ZZW | STLA | LASK |
|---|---|---|---|
| The intra-network latency | $L_e E$ | $L_E \log_2 \frac{N(1-R/2)}{2E} + L_e \log_2 E$ | $L_e \log_2 E$ |
| The worst case latency | $\frac{L_E N}{E} + L_e N$ | $L_E \log_2 \frac{N}{E} + L_e \log_2 E$ | $\frac{N}{E}(L_E + L_e \log_2 E) + L_e \log_2 E$ |
| $k$ nodes traversal | $EL_e$ | $kL_e$ | $kL_e$ |
| Join overhead | $O(\log_2 N)$ | $O(\log_2 N)$ | $O(\log_2 N)$ |

TABLE II
SIMULATION PARAMETERS

| Parameters | Internet scenario | DC scenario |
|---|---|---|
| Hierarchy height | 3 | 2 |
| # of clusters | 1, 7, 400 | 1, 5 |
| Latency of each cluster(ms) | 100, 20, 2 | 20, 0.2 |
| Distribution | Uniform, Zipf | Uniform, Zipf |

latency implies the number of successful discovery/delivery before timeout because it increases when the discovery latency is smaller in LASK. The evaluation compared LASK with STLA and ZZW.

To evaluate the scalability, we varied the number of nodes from 10 to 100,000. We evaluated two scenarios: an Internet scenario and a data center (DC) scenario. In the Internet scenario, Layer 1 is the continents layer, and Layer 2 is the autonomous systems (ASes) layer. We suppose 7 continents, with 400 ASes in each continent. The latency between continents was set to 100ms, that between ASes in the same continent was set to 20ms, and the intra-AS latency was set to 2ms. In the DC scenario, Layer 0 is the inter-DC layer, and Layer 1 is the DC layer. We assumed there were five data centers, and set the latency between DCs to 20 ms, whereas the latency inside DCs was set to 0.2 ms. These values are taken from measurements of an actual data center network in the JOSE test-bed [24]. In both scenarios, the latency varied according to the Gaussian distribution. Table II presents a summary of the parameters in both scenarios. These parameters are the same as that of the [17].

The nodes join the service discovery overlay network according to the Zipf distribution and uniform distribution. The Zipf distribution can reproduce the scenarios in which the popularity of nodes is concentrated within a small number of subnetworks. Combination of these scenarios and distributions reproduce typical edge computing environments.

In the evaluations, the setting of rates of the requester is set as 30%. For example, if there are 10 nodes, 7 nodes register their service name, but 3 nodes are just requesters and do not register a service name.

First, we evaluated the basic performance of discovery latency changing the total number of nodes to show the scalability of LASK. Fig. 6 (DC scenario) and Fig. 9 (Internet scenario) show the results to discover one service($k = 1$) located in the same edge network with the requester. Note that both x-axis and y-axis are log scale. We can see the latency

of LASK is smallest among the comparison schemes if the number of nodes is large. The latency of LASK does not grow proportionally in both Zipf and Uniform distributions though the number of total nodes in the network grows exponentially. Even when there are 100,000 nodes, the latency of LASK is around 4 ms in the DC scenario and 81ms (Uniform) and 41 ms (Zipf) in the Internet scenario. In the Internet scenario, because the number of nodes in one subnetwork is small, ZZW could achieve small latency when the number of the total nodes is small. However, ZZW could not achieve the scalability because it traverses node by node. STLA could achieve scalability, but it requires longer latency as it requires forwarding the query via different networks.

Next, we evaluated the latency changing $k$ of the query to see the performance of the scalability for $k$. Fig. 7 (DC scenario) and Fig. 10 (Internet scenario) show the results. In this evaluation, the spread rate, the rate of the nodes that match to the query, is set as 0.1. This graph presents only the y-axis as a log scale. The number of nodes is set as 100,000. As we can see from these graphs, the tendency of LASK, STLA, and ZZW are same as the previous evaluation. In this evaluation, we added the case of LASK without NL axis (LASK-NoNL). In this case, the LASK does not use NL axis to traverse $k$ matched nodes. The Evaluation result shows that the NL axis works effectively to traverse $k$ matched nodes in LASK. The latency is approximately 100 ms in the DC scenario, 181ms (Uniform) and 246 ms (Zipf) in the Internet scenario when $k = 10$.

Lastly, we evaluated the latency changing the spread rate of the target. If the rate is small, then the number of nodes that matches to the query is small. That is, it becomes harder to discover matched nodes if the rate is small. Fig. 8 (DC scenario) and Fig. 11 (Internet scenario) show the results. As expected, the smaller the rate, the longer the latency. LASK keeps the smallest latency even when the rate is small. In the Internet scenario with Uniform distribution, the result becomes closer to the STLA when the rate is $10^{-3}$ (LASK took 1208 ms, and STLA took 2069 ms). That is because the number of networks to traverse in LN axis increases in LASK. However, such nodes are located at further subnetwork. To avoid finding a node with longer latency, the timeouts $T$ should be specified.

As a summary, we could see that LASK could achieve small latency in the typical edge computing environment of the Internet and DC scenarios.

## V. Conclusion

In this paper, we proposed a novel overlay network protocol LASK, for supporting scalable $k$-Nearest Service Discovery. Without requiring a global server and distributed server settings, LASK runs autonomously and provides a responsive lookup function in the distributed edge computing environment. As the evaluations by simulation show, LASK can achieve small-latency and scalability for the service discovery. When there are 100,000 distributed resources, we can expect less than 5 ms latency in the data center network and less than 100 ms in the Internet environment. These results are much faster than conventional distributed algorithms.

As a future work, we are planning to evaluate LASK using a real implementation applying to an actual application such as a dynamic data flow processing with automatic resource allocations. In addition, further algorithm improvements and evaluations with other related schemes considering overheads are also our future work.

## References


[1] H. Tanaka, M. Yoshida, K. Mori, and N. Takahashi, "Multi-access edge computing: A survey," *Journal of Information Processing*, vol. 26, pp. 87–97, 2018.

[2] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges," *Future Generation Computer Systems*, vol. 78, pp. 680–698, 2018.

[3] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, 2009.

[4] iGillottResearch Inc., "The Business Case for MEC in Retail: A TCO Analysis and its Implications in the 5G Era," 2017.

[5] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. of INFOCOM 2012*, 2012, pp. 945–953.

[6] D. Bai, W. Yun, and S. Chung, "Redundancy optimization of k-out-of-n systems with common-cause failures," *IEEE Transactions on Reliability*, vol. 40, no. 1, pp. 56–59, 1991.

[7] R. Ranjan, L. Zhao, X. Wu, A. Liu, A. Quiroz, and M. Parashar, "Peer-to-peer cloud provisioning: Service discovery and load-balancing," in *Cloud Computing*. Springer, 2010, pp. 195–217.

[8] E. Caron, F. Chuffart, H. He, and C. Tedeschi, "Implementation and Evaluation of a P2P Service Discovery System: Application in a dynamic large scale computing infrastructure," in *Proc. of CIT 2011*, 2011, pp. 41–46.

[9] S. Cirani, L. Davoli, G. Ferrari, R. Léone, P. Medagliani, M. Picone, and L. Veltri, "A scalable and self-configuring architecture for service discovery in the internet of things," *IEEE Internet of Things Journal*, vol. 1, no. 5, pp. 508–521, 2014.

[10] H. Yamanaka, E. Kawai, Y. Teranishi, and H. Harai, "Proximity-Aware IaaS for Edge Computing Environment," in *Proc. of ICCCN 2017*, 2017, pp. 1–10.

[11] T. Akiyama, Y. Teranishi, R. Banno, K. Iida, and Y. Kawai, "Scalable pub/sub system using openflow control," *Journal of Information Processing*, vol. 24, no. 4, pp. 635–646, 2016.

[12] Y. Komai, Y. Sasaki, T. Hara, and S. Nishio, "$k$ NN Query Processing Methods in Mobile Ad Hoc Networks," *IEEE Transactions on Mobile Computing*, vol. 13, no. 5, pp. 1090–1103, 2014.

[13] 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 Transactions on Networking*, vol. 11, no. 1, pp. 17–32, 2003.

[14] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Proc. of International Workshop on Peer-to-Peer Systems*, 2002, pp. 53–65.

[15] C. Dannewitz, M. DAmbrosio, and V. Vercellone, "Hierarchical DHT-based name resolution for information-centric networks," *Computer Communications*, vol. 36, no. 7, pp. 736–749, 2013.

[16] N. Fotiou, K. V. Katsaros, G. Xylomenos, and G. C. Polyzos, "H-pastry: An inter-domain topology aware overlay for the support of name-resolution services in the future internet," *Computer Communications*, vol. 62, pp. 13–22, 2015.

[17] Y. Teranishi, R. Banno, and T. Akiyama, "Scalable and locality-aware distributed topic-based pub/sub messaging for iot," in *Proc. of GLOBECOM 2015*, 2015, pp. 1–7.

[18] J. Aspnes and G. Shah, "Skip graphs," *ACM Transaction on Algorithms*, vol. 3, no. 4, pp. 1–25, 2007.

[19] M. Waldvogel and R. Rinaldi, "Efficient topology-aware overlay network," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, pp. 101–106, 2003.

[20] Y. Teranishi and N. Nishinaga, "Building a large-scale distributed live video analysis system for movement of pedestrians in urban areas," in *Proc. of COMPSAC 2016*, vol. 1, 2016, pp. 648–657.

[21] T. Schütt, F. Schintke, and A. Reinefeld, "Range queries on structured overlay networks," *Computer Communications*, vol. 31, no. 2, pp. 280–291, 2008.

[22] Y.-J. Joung, W.-T. Wong, H.-M. Huang, and Y.-F. Chou, "Building a network-aware and load-balanced peer-to-peer system for range queries," *Computer Networks*, vol. 56, no. 8, pp. 2148–2167, 2012.

[23] "PIAX distributed computing framework." [Online]. Available: http://piax.org/

[24] Y. Teranishi, Y. Saito, S. Murono, and N. Nishinaga, "JOSE: An Open Testbed for Field Trials of Large-scale IoT Services," *Journal of the National Institute of Information and Communications Technology Vol*, vol. 62, no. 2, 2015.