SkABNet: A Data Structure for Efficient Discovery of Streaming Data for IoT

Philipp Kisters
Department of Computer Science
Universität Hamburg
Hamburg, Germany
philipp.kisters@uni-hamburg.de

Heiko Bornholdt
Department of Computer Science
Universität Hamburg
Hamburg, Germany
heiko.bornholdt@uni-hamburg.de

Janick Edinger
Department of Computer Science
Universität Hamburg
Hamburg, Germany
janick.edinger@uni-hamburg.de

2023 32nd International Conference on Computer Communications and Networks (ICCCN) | 979-8-3503-3618-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCCN58024.2023.10230169

***Abstract*—Applications in the Internet of Things often make use of large networks of independent sensor nodes that generate streams of volatile data. A major challenge in these decentralized networks is to efficiently discover relevant data providers, which might be characterized by properties such as their data type, location, or ownership. Most existing approaches use distributed data structures, such as distributed hash tables, for the organization of sensor nodes. However, these systems lack the ability to consider contextual properties when identifying relevant data sources. *SkipNet* a prominent architecture for data storage and retrieval, provides a scalable overlay network composed of doubly-linked rings. While the data structure allows to locate individual nodes in logarithmic complexity, it fails to identify groups of nodes that share similar characteristics. Thus, in this paper, we propose *SkABNet*, an attribute-based extension for SkipNet which enhances the semantics of the node identifiers in the network. We introduce additional operators that allow SkABNet to accept complex search queries including multi-attribute selections, ranges, and wildcards to find relevant data providers in its decentralized data structure. Further, we define a search algorithm that performs searches with significantly less messages than comparable searches in SkipNet.**

***Index Terms*—Distributed information systems, Internet of Things, Overlay networks**

I. Introduction

Internet of Things (IoT) based applications usually consists of huge amounts of devices, most of which are made available to users and services via the Internet. Many of these IoT devices use a wide variety of sensors and, thus, generate a large number of volatile data streams. One of the biggest challenges posed by the volume of devices is to find relevant information efficiently [1]. In this context, three main characteristics of such data are identified that add complexity to the discovery of streaming data in IoT: its volatility, its locality, and its dependence on other data [2].

Data volatility renders caching and replication impossible as sensor data quickly becomes stale. Data locality results in searches, which target groups of devices located physically close to each other. Dependence on other data further results in searches that not only match a single data type but groups of data types influencing each other. Therefore, searches in the IoT context are expected to target whole groups of data providers that either share common attributes like location and type or are related by data they provide.

To overcome these challenges, different approaches have been presented already: centralized approaches like [3] and [4] utilize a discovery service provided by a central entity. These central entities allow for fast lookup times and minimal message overhead. On the contrary, a central entity creates a bottleneck and poses a single point of failure for large IoT networks possibly decreasing the availability of data streams. Due to the ever-increasing number of IoT devices and available data, the load of this central entity will constantly increase, which lead to reducing its performance or increasing the cost to operate it. While decentralized approaches increase the overall management overhead between IoT devices, they decrease the cost by evenly sharing responsibility between participating devices. Further, due to various repair mechanisms higher resilience can be achieved. Decentralized approaches can be grouped in structured and unstructured networks. Structured networks define rules how participating devices join and how they are arranged within a network. Unstructured networks define no such rules, which decreases the management overhead, but they cannot guarantee complete search results without flooding the whole network.

Unstructured networks based on the geographic distributed of IoT devices as presented in [5], [6]. They focus on local service discovery resulting in fast search results, of nearby services. This comes with the disadvantage that services that lie outside of searches parts of the network are not found, even if they are present.

Structured networks are either based on Distributed Hash Table (DHT) or distributed skip lists. DHTs such as [7]–[10] use hashed identifiers to evenly place data on participating nodes. However, two important aspects get lost when using hashed identifiers. (1) to find relevant data, the exact identifier must be known, since a small change in the identifier results in a completely different hash and therefore a different location within the network. (2) due to the hashing of identifiers, similar data is placed at completly different locations within a network. This further eliminates the coupling of data creation and storage and, thus, the control over data by its owner.

Skip list-based networks such as [11], [12] provide a data structure based on one dimensional user-selected identifiers. These identifiers allow a more descriptive naming scheme such as hierarchical namespaces based on entities, or spacial and

time-related data that can be searched efficiently. Through these user-selected identifiers, participating IoT devices can decide where their provided data is stored (content locality). Messages between two IoT devices from within the same namespace do not leave their namespace (path locality). Likewise, queries matching IoT devices based on a common namespace within the identifier, so called range queries, are supported allowing for searches selecting neighboring nodes within the network. While these range queries allow for searches of groups of IoT devices, these ranges are very limited in their use-cases and depend on the descriptiveness of the one dimensional identifiers since these IoT devices must share a common namespace.

This paper presents Attribute-Based SkipNet (SkABNet), an attribute-based architecture extending SkipNet [11] to provide three main enhancements over the original data structure: First, SkABNet enhances the user-selected identifiers by introducing attribute-value pairs. This allows to include contextual information within identifiers and therefore grouping of similar nodes based on different attributes. Second, compared to range queries provided by SkipNet, more complex searches can be defined in SkABNet, such as multi-attribute selections, ranges, wildcards, and any combination of the above. Further, SkABNet performs searches in a more efficient manner than SkipNet, resulting in an overall reduced lookup time.

The remainder of this paper is structured as follows: Sec. II gives a short overview of related work. Sec. III introduces our overlay architecture and the basic SkipNet. Sec. IV describes the proposal of SkABNet as an appropriate data structure for distributed sensor networks, as well as related enhancements. Finally, Sec. V discusses the scalability of SkABNet. Sec. VI concludes this paper and provides an outlook for future work.

II. Related Work

Efficient discovery of nodes defined by a set of describing attributes increases in importance with the number of attributes and the network size. Existing discovery approaches in distributed systems can be classified into two categories: Finding static data defined by unique identifiers and finding specific nodes participating in the network. In the first category, DHT-based approaches [7]–[10] were used in order to improve load balancing and achieve faster lookup times through data caching and replication. Due to the volatility of generated data within the IoT context, these methods can only be used to a limited extent, since caches and replicas have to be updated and replaced continuously. Similarly, searches in IoT are rarely performed for a single data point, due to its dependence on other data as mentioned before. Instead, searches for categories of data from different IoT devices are to be expected. This indicates that full identifiers are most often unknown when starting a search. Instead, descriptive attributes are used to search for matching nodes.

A. DHT-based Networks

More recent research focuses on discovering nodes based on one or multiple attributes. S. Cirani et al. propose an architecture based on a DHT utilizing gateway nodes with a stable internet connection [13]. Sensors connect with a locally managed gateway node, which participates in a network based on a distributed location service [14] and distributed geographic table [15]. Nodes can be efficiently discovered based on their physical location. However, other attributes (such as sensor type) cannot be used for the discovery.

Also based on a DHT, F. Paganelli and D. Parlanti propose a discovery service which is able to search for multi-attribute range queries [16]. To achieve this, the authors map multidimensional attributes into a one-dimensional namespace using space filling curves. The resulting identifiers are then sorted using a Prefix Hash Tree (PHT). Finally, the PHT node identifiers are hashed and distributed over a DHT. Resulting in an equally distributed load across the network while still enabling range queries performed over the PHT. In addition, multi-attribute searches are possible, with the drawback of potential false positives, which have to be filtered out afterwards. While this work allows for multi-attribute searches, false positives and the repetitive use of the underlying $get()$ on the DHT leads to an unnecessary high load of messages. Furthermore, no selection of multiple specific attributes values is possible (e.g. sensors of type humidity or temperature).

B. Skiplist-based approaches

In parallel approaches utilizing distributed skip lists [17] emerged. Harvey et al. [11] and Aspnes and Shah [12] independently introduced a distributed network architecture based on user-selected identifiers. These user-selected identifiers allow for more descriptive identifiers and group similar content within the network. Further range queries based on these identifiers are supported, to find all nodes within a network that share a common prefix in their identifiers.

Li et al. propose a locality-preserving context-aware service discovery called ”LOCA” [18] based on distributed skip lists. The authors focus on reducing required message hops for queries within an organizational domain. Therefore, they introduced a more efficient naming scheme for services. While this effectively preserves locality and integrity within the specified organizational domains, an ontological model is required to receive all relevant services. While range queries are supported by SkipNets by default, Li et al. focus on an ontology driven discovery to find relevant services based on different criteria. A drawback within this approach is that each service has to be contacted to evaluate whether all required criteria are met.

Ishi et al. [19] introduced bounded range queries, allowing for faster search results targeting a range of nodes, by splitting search messages into multiple subqueries. This approach has been further optimized by Banno and Shudo [20], to ”reduce the average path length by roughly 30%”. While these extensions enhance the efficiency of range queries, selections and ranges on multiple attributes are not supported.

However, all previously presented approaches lack the ability to facilitate selection queries on multiple attributes. To allow for multi-range queries, additional overlays are needed and multiple search queries may have to be created. This

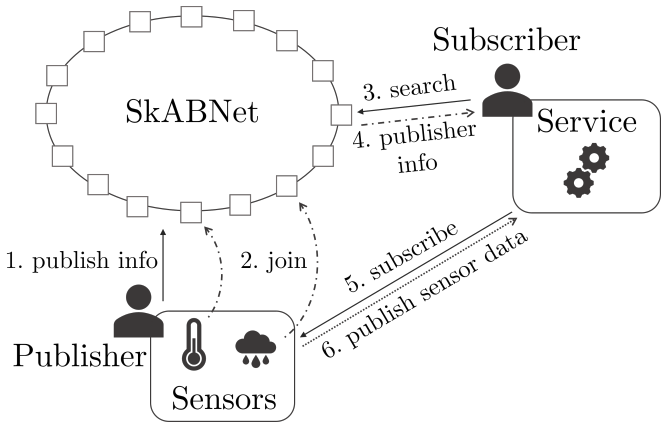


Fig. 1: Discovery architecture of streaming data based on a SkABNet.

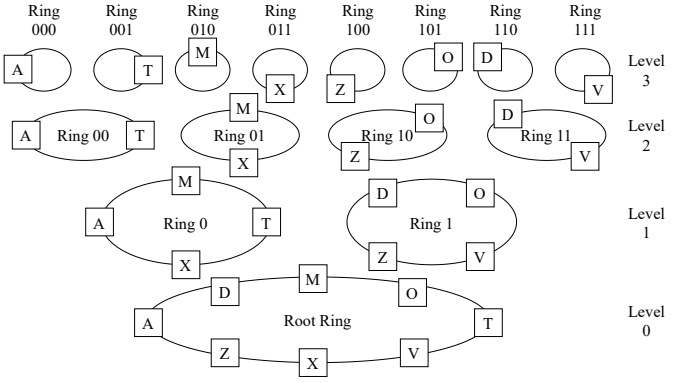


Fig. 2: The SkipNet architecture arranges nodes onto rings, each halving the ring of the underlying layer [11].

further increases the overall messages overhead. Due to the locality-preserving property of SkipNets and its possibility to define range queries, we selected this data structure as a foundation of our attribute-based approach.

III. Overlay Networks for IoT

Organizing a large number of IoT data providers requires a scalable network overlay that allows to discover available publishers and subscribe to their data streams. The overall architecture of such a decentralized system (Fig. 1) is as follows: Geographically distributed publishers own sensors that generate data streams. In absence of a centralized data manager, publishers organize these data sources via a decentralized data structure, which in our case, is a network called SkABNet. Therefore, each publisher participates in the network, by publishing information about provided data streams (1) and deploying nodes in the network representing described data streams (2). Data subscribers, who have a particular interest in some of these data streams (e.g., all air pollution data in Manhattan), search for relevant nodes within this network. They define a parameterized search query (3) that results in a set of publisher nodes providing the requested data streams of interest (4). The exchange of collected data itself is not part of the SkABNet network, but can look like the following: Subscribers can now issue a subscription on discovered data sources (5), which, from this point on forward data streams to the subscriber (6) who use these to provide various services.

To organize these nodes in a decentralized data structure SkABNet augments the architecture of SkipNet [11]. SkipNet is a prominent overlay network based on a distributed skip list [17]. Each node has two identifiers: a user-selected alphanumeric ID (*content ID*) and a randomly generated unique binary *numeric ID*. For the remainder of this section we will refer to the SkipNet architecture as depicted in Fig. 2.

Nodes are located on multiple double-linked rings with each level halving the set of nodes based on their numeric ID. Starting at level 0, a single ring contains all participating nodes. The following level is comprised of two rings each with 50% of the nodes, level 2 already has 4 rings with 25% of the nodes respectively, and so forth. Each node stores its adjacent nodes (*neighbors*) from each of these rings in its neighbor tables. Due to the randomly generated numeric IDs, neighbors on higher ring levels will have a large distance on the level-0 ring allowing to skip neighbors to find a searched node in $O(log\ n)$ hops where n is the total number of nodes in the network.

The user-selected content IDs are then used to order nodes within each ring. This ordering ensures that nodes with identical prefixes in their content ID are adjacent on all shared rings. Messages targeting nodes with identical prefixes can be sent via nodes containing this prefix only. Additionally, all nodes with identical prefixes are found without leaving the prefix-based namespace.

In order to find nodes within a SkipNet for a defined content ID, a node starts the search at their highest-level ring to find another node with a content ID located between its own and the target. If a closer node is found the search is forwarded until either the searched node has been reached or if no closer node exists in the level-0 ring, which means that no node with the searched content ID exists.

$$Manhattan.Broadway\#657 \\ \hookrightarrow .Temperature.Sensor001 \quad (1)$$

$$Q :: Manhattan.Broadway. \quad (2)$$

Multiple nodes can be searched using range queries on a common prefix within their content ID. Listing (1) shows an example content ID representing a data provider collecting temperature data at the Broadway in Manhattan. Content IDs in this form allow for range queries matching all data providers in Manhattan that are located at the Broadway by using the range queries shown in Listing (2). To find all matching nodes, the search is forwarded to the first matching node utilizing the content ID routing algorithm. The found node finally forwards the search on the lowest level ring until all nodes with the given prefix are found.

IV. SkABNet Architecture

We propose SkABNet, a distributed data structure that builds upon the idea of SkipNet and introduces **a**ttribute-**b**ased identifiers. This extension does not only allow to express more complex search queries but also execute these queries more efficiently. This makes it more convenient for users searching for data streams defined by multiple attributes and reduces the total number of messages to find all matching data streams.

To achieve this, we extend the SkipNet architecture with three distinct enhancements: First, we provide *attribute-based identifiers* allowing for more expressive identifiers by dividing them into a list of attribute-value pairs. Further we utilize the introduced attribute-value pairs to create more powerful search queries. By dividing identifiers into a list of attribute-value pairs, each attribute can be interpreted individually and thus allow searches on single or combinations of attribute-value pairs. Second, we introduce four *attribute-based search operators* that enable ranges and selections on individual attributes. Lastly, to optimize for common search patterns we represent data providers by multiple *attribute compositions* reducing required messages to find matching nodes.

In the following we will explain each enhancement on an example of a city-wide distributed IoT network. None the less this extension benefits all distributed networks that require discovery of individual and groups of participating nodes that can be described by a set of attributes.

A. Attribute-Based Identifiers

As mentioned before SkABNet uses attribute-value pairs as identifiers. Attributes can be any descriptive properties about the data source such as its location, sensor type, or measuring interval. When publishers make their data streams available to the network, they enter alphanumeric values for each attribute to make the data source identifiable for subscribers in the network. The choice of attributes is similar for each publisher and subscriber in the network and is defined by the domain. The only requirement SkABNet has for attribute-based identifiers is, that these need to be unique, therefore, an attribute representing a *UUID* should exist to make sure that all participating nodes are described by a unique identifier. A network for sensor data collected by private device owners could, e.g., require the address, sensor type, and measuring interval of the data source. Networks in other domains might require a completely different set or attributes. While, technically, each attribute contains alphanumeric values, some attributes may also require a particular data format. For example, longitude and latitude should be stated as floating point numbers. This can be verified while nodes are registered.

Having independent attribute-value pairs makes it possible to put the attributes in arbitrary order. In contrast to SkipNet, where the identifier is ultimately defined at node creation, SkABNet allows each network to define a unique order of attributes without requiring any further input from the publishers. This order can have a significant impact on the search efficiency as we discuss later in this paper.

The idea of attribute-value pairs also benefits subscribers who search the network for relevant data streams. They do not have to define search queries that exactly match the identifiers as required in SkipNet. Instead, they can define search values for some attributes and omit those which are irrelevant to them. The latter will be replaced by wildcards when the search query is executed.

Listing (3) shows the structure of an SkABNet identifier. The delimiter "/" separates attributes and values as well as attribute-value pairs.

$$/\textit{Dist}/Manhattan/\textit{St}/Broadway/\textit{No}/657/ \\ \hookrightarrow \textit{Type}/Temperature/\textit{UUID}/Sensor001 \tag{3}$$

B. Attribute-Based Search Queries

Subscribers use search queries in SkABNet to lookup relevant data streams. They define values for the attributes defined by a network. When the attribute values of the data source match the query, this data source is considered relevant and its publisher info is sent to the subscriber.

Search queries can have different complexities. In the most basic scenario, subscribers insert exactly one value for each attribute. This results in a straightforward search looking for, e.g., the temperature sensor at a particular place which can be identified by the combination of a city, a district, a street name, and a house number. This type of query is also standard in SkipNet whenever the fully qualified identifier is used.

Continuing the example above, the subscriber may also want to retrieve data from the humidity sensor in this place. While this would require two distinct searches in SkipNet, both using the entire identifier, SkABNet accepts **selections** as an input. In the example, the subscriber could insert both values, *temperature* and *humidity*, in the *type* attribute. SkABNet interprets this selection and performs a search for nodes matching either sensor type. Selections can include not only two but any number of values.

Selections are useful when more than a single value is relevant. However, when the number of values becomes too big, selections may be tedious and error-prone to use. A second drawback of selections is, that each value needs to be known in advance. What may be realistic for, e.g., districts or streets, floating point values such as in longitude or latitude attributes can typically not be provided as a list. For these reasons, SkABNet provides the **range** operator which accepts a lower and an upper bound and matches every value in between. Thus, subscribers can define large search spaces, even for floating point numbers, with minimal additional input. In contrast, SkipNet does not process bounded range queries. Instead, for every value within the range, a new search has to be started. This does not only mean that the number of individual searches can become quite large, it also requires that all values must be discrete and known in advance.

There are many scenarios where some attributes are of no interest for the subscriber. When looking for, e.g., the entire set of temperature sensors in London, there is no need

Operator	Description	Use-Cases	Possible in SkipNet?
Single Value	Finds nodes where the exact value (*Manhattan*) matches for the district attribute. **Example:** $/dist/Manhattan$	This operator is the most basic and is used to find only specific data providers with the given value.	Yes. This is the standard search for SkipNet.
Selection |	Finds nodes with values specified in the selection. Nodes located in between these values are skipped. **Example:** $/St/1stAve\|Broadway\|ParkAve$	Most commonly used for alphanumeric values where several values are of interest.	Partially. Multiple searches have to be performed, one for each element in the selection.
Range $\sim$	Specifies a range of values with inclusive boundaries. Finds all values that are alphanumerically ordered between the upper and lower bound. **Example:** $/No/003 \sim 065$	Most commonly used for numeric values. Numbers are needed to be padded with leading zeros so that they are correctly ordered alphanumerically.	Restricted. Only possible, if all values within the range are known or by implementing the extention discussed in [19], [20].
Wildcard $*$	Matches nodes regardless of the value for this attribute. **Example:** $/Type/*$	Used when attribute is irrelevant for the search.	Restricted. Only possible, if the wildcard attribute is at the last position or all values for this attributes are known.

TABLE I: Search operators in SkABNet. The additional operators (selection, range, wildcard) can be used to intuitively define complex queries. SkipNet, in contrast, requires manual effort to mimic these queries with multiple single-valued queries.

to indicate which streets and house numbers are relevant. Instead, all of them should be included in the query. For that, SkABNet provides a **wildcard** operator which can be used as a placeholder for the whole value set of an attribute. In SkipNet, wildcards are difficult to implement. If the last value in the identifier is the one that is irrelevant, the stub of the identifier (without the final value) can be used for a range query. Wildcards in any other part of the identifier result in a single search for each attribute value and would also only work for discrete values which are known in advance.

These additional operators summorized in Table I allow for a greater flexibility when defining search queries. This advantage becomes even more apparent when multiple operators are combined as in the following example. Listing (4) finds all publishers providing temperature and humidity within *dis1* in the *ExampleStreet* with house numbers ranging between 15 and 25. By considering attribute-value pairs individually, range and selection operators can be combined in a single search.

$$Q :: /Dist/dis1/St/ExampleStreet/No/15 \sim 25 \quad (4)$$
$$\hookrightarrow /Type/Humidity|Temperature/UUID/*$$

To achieve a similar result in a standard SkipNet, an individual search has to be performed for each combination of house number and sensor type, leading to a total of twenty individual searches. In addition, results would have to be merged, after all searches are completed.

$$Q :: /Dist/dis1/St/ExampleStreet/No/15$$
$$\hookrightarrow /Type/\textit{Humidity}/$$
$$Q :: /Dist/dis1/St/ExampleStreet/No/15$$
$$\hookrightarrow /Type/\textit{Temperature}/$$
$$Q :: /Dist/dis1/St/ExampleStreet/No/16$$
$$\hookrightarrow /Type/\textit{Humidity}/$$
$$Q :: /Dist/dis1/St/ExampleStreet/No/16$$
$$\hookrightarrow /Type/\textit{Temperature}/$$
$$\ldots$$

C. Efficient Provider Discovery

Next, we discuss how publishers in SkABNet can be found efficiently. The position of a node representing a published data stream is determined by its identifier that consists of alphanumeric attribute-value pairs. In the lowest-level ring, all nodes are present and sorted in ascending order (compare Fig. 2). The standard SkipNet can efficiently discover a single and a range of nodes, as long as these share a common prefix as described in Sec. III.

SkABNet allows more complex search queries with selections, ranges, and wildcards on each attribute. As a result, matching nodes may be scattered across the lowest-level ring. To find them efficiently anyway, SkABNet implements a new search algorithm which dynamically splits a search into multiple sub-searches each focussing on different parts of lowest-level ring with possible matching nodes. These sub-searches are forwarded in parallel while making sure that no node receives the query twice. We will discuss this algorithm in the following. Further, we will demonstrate how compositions of SkABNet identifiers, i.e., the choice and order of attribute-value pairs, impacts search efficiency.

Search Algorithm

SkABNet's attribute-based search is divided into two stages. Within the first stage, the search message is forwarded to the first node that matches the search query. This part is identical to the search algorithm in SkipNet. The first matching node can directly be calculated using the left values of selections and ranges. Once this node is found, the second stage begins and the search splits up into branches that search the network in parallel. We demonstrate this with the example query from above (see Listing (4)). For easier readability, attribute names are shortend and single values are omitted. The search query now looks like this:

$$Q :: /a1/H|T/a2/15 \sim 25$$

Attribute *a1* represents the sensor type, which can either be a *H*umidity sensor or a *T*emperature sensor. Attribute *a2* represents the house number, where values between 15 and

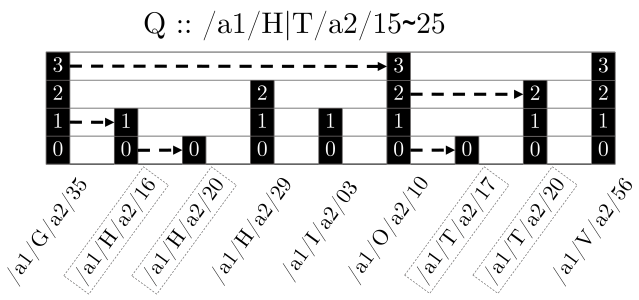


Fig. 3: Nodes forwarding a search to neighbors on different ring levels to find all matching nodes for the given search.

25 are relevant for the search. The first relevant node will therefore have the following identifier:

$$/a1/H/a2/15$$

To find this node, the search algorithm starts on the node owned by the subscriber and then checks its neighbors on each ring. Following SkipNet's search algorithm skipping irrelevant nodes to find the first matching one within $O(log\ n)$ steps. If no node exists matching this identifier, the last node that forwarded the search starts the second stage. Here, the search is split up.

Fig. 3 illustrates the second stage of the search algorithm. Each bar represents a node. The rows indicate the neighbor relationships between the nodes. Node $/a1/G/a2/35$, for example, is neighbor to node $/a1/H/a2/16$ on ring levels 0 and 1, to node $/a1/H/a2/29$ on level 2, and to node $/a1/O/a2/10$ on level 3. Arrows indicate to which neighboring nodes a search message is forwarded.

At some point, the search reached the node with the identifier $/a1/G/a2/35$, which detects, that no node with the minimal identifier ($/a1/H/a2/15$) exists. Therefore, it starts the second stage and splits up the search message in order to find all matching nodes in parallel. $/a1/G/a2/35$ forwards a sub-search to $/a1/H/a2/16$ that matches the search. The upper boundary is set to $/a1/H/a2/29$ since this is the neighboring node on the next higher ring. Setting the upper boundary correctly is important to assert that no nodes receives the same query twice. $/a1/H/a2/29$ receives no search message, since no relevant nodes can be located in between itself and the next neighbor $/a1/O/a2/10$. However, a search message is forwarded to node $/a1/O/a2/10$ even if it is not matching the search since it is the neighbor on the highest ring-level with possible matches located behind it.

Node $/a1/H/a2/16$ received the search message and forwards it to its matching neighbor $/a1/H/a2/20$. The message is not forwarded to $/a1/H/a2/29$ as this node is set as the upper boundary in the received sub-search.

Parallel to that, node $/a1/O/a2/10$ receives the message and forwards the search to its neighboring nodes. The neighbor on the lowest-level ring with the identifier $/a1/T/a2/17$ matches the search and will receive a sub-search with the upper boundary $/a1/T/a2/20$, since this is the neighbor on the next higher ring-level. The search is also forwarded to the node $/a1/T/a2/20$ since it matches the search. $/a1/V/a2/56$ receives no message since it is located behind the last possible matching node.

$/a1/T/a2/17$ received the search from $/a1/O/a2/10$ and does not forward it to $/a1/T/a2/20$ even though it matches the search, since its identifier was set as the upper boundary in the search received, this ensures that $/a1/T/a2/20$ does not receive search messages multiple times. $/a1/T/a2/20$ received the search message from $/a1/O/a2/10$ and has no other neighboring node matching the search, therefore all matching publishers represented by these nodes where found.

Algorithm 1 Forwarding searches to multiple neighbors while no neighbor receives duplicate messages

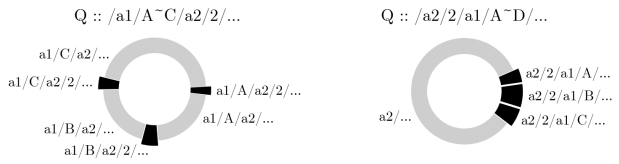
```
1:  procedure FORWARDSEARCH(Search, NeighborTable)
2:      for i ← 0, Search.MaxRing do
3:          neighbor ← NeighborTable[i]
4:          if neighbor > Search.UpperBound then
5:              return No more neighbors
6:          end if
7:          nextNeighbor ← NeighborTable[i+1]
8:          if Search.matches(neighbor) or
              matchBetween(neighbor, nextNeighbor) then
9:              cSearch ← Search
10:             cSearch.UpperBound ← nextNeighbor
11:             cSearch.MaxRing ← i
12:             sendSearch(cSearch, neighbor).
13:         end if
14:     end for
15: end procedure
```

Algorithm 1 shows how the search message is forwarded in the second stage of the algorithm. The upper boundary is initialized with the last matching node which is determined by using the right values of selections and ranges. Now each traversed node is checking for relevant neighboring nodes until the upper boundary is reached (line 5). The search is forwarded to irrelevant nodes as well, as long as there are possible relevant ones between a node itself and its neighbors on the next higher ring (line 8).

Attribute Composition

Following this search algorithm, each relevant publisher can be found in $O(log\ n)$ network hops. The overall number of total messages needed to find all relevant publishers depends on their distribution within the SkABNet. As this location is defined by the ordering of their attributes in the identifier, searches targeting different attributes perform differently. For example, searches specifying attributes located at the beginning of the identifier have to search smaller parts of the SkABNet than searches with wildcards or ranges within attributes at the beginning of the identifier.

Fig. 4 shows the same SkABNet with different placement of publishers due to the order of the attributes in the identifier. In Fig. 4a, publishers that have the same attribute *a1* value are



(a) Attribute *a1* prefixing *a2* (b) Attribute *a2* prefixing *a1*

Fig. 4: Distribution of matching nodes (highlighted in black) within two SkABNets with different ordering of attributes in their contentID for a search targeting a range of A $\sim$ C for *a1* and a value of 2 for *a2*

located next to each other. This makes searches for a single value in attribute *a1* and ranges or selections in *a2* efficient. On the contrary with a range over values for *a1* leads to relevant publishers scattered across the SkABNet as shown in Fig. 4a. Using a different order of attributes, as displayed in Fig. 4b, results in the relevant nodes being neighbors in the SkABNet. However, within this distribution of publishers, searches that target a specific value for *a1* and a range or selection of values for *a2*, matching publishers would be scattered again. Therefore, attribute compositions should be determined by the most common search pattern to group matching publishers and make searches more efficient.

In order to optimize for different search queries, a SkABNet starts multiple representations of publishers via so called *virtual nodes*. The concept of virtual nodes was already introduced in the standard SkipNet as an optional enhancement [11]. Virtual nodes in SkABNet have their own identifier and therefore allow for different orders of attributes, optimizing the network for multiple common search queries.

Utilizing different compositions of attributes leads to another challenge: the number of possible compositions increases factorially with the number of attributes. Therefore, only a small set of attribute compositions should be started. Appropriate attribute compositions are highly dependent on the context and should represent the most common search queries. The number of compositions data providers can start depends on the hardware used in the context, since managing neighbor tables within the SkABNet scales about linearly with the number of compositions. Furthermore, an adequate balance between an efficient discovery and the overall size of the SkABNet must be found.

In the previous example, attributes are: *district*, *street*, *house number*, *sensor type*, and *UUID*. In addition, a *latitude* and *longitude* representation would also be helpful to receive regional data independently from streets or districts. Using any combination of these seven attributes would already result in 5040 virtual nodes per publisher. This number can be considerably decreased by defining specific attribute compositions depending on the IoT context. Often, there are attribute pairs that are always searched in conjunction (e.g., searching by street name and house numbers). Also, there may be attributes that are unlikely to occur together such as a postal address and latitude and longitude representation of a location.

For named attributes such as sensor type, street, and district, we assume single values and selections in most cases so they are set at the beginning of our identifiers. To avoid searches with a wildcard in the beginning of an identifier, since these result in scattered matching nodes and are therefore costly hop-wise, we add two more compositions, that either add the type at the end or omit the district. Attributes that most likely represent numeric values such as the house number are added next, since ranges are costly hop-wise and therefore are preferably used on already limited name groups. The UUID is already unique and most likely unknown to subscribers and therefore in most cases replaced by a wildcard, so it will be appended at the end to fulfil the requirement of unique identifiers. For the case that specific publishers are searched for, an additional composition is added only using the UUID and the sensor type.

$$/Type/District/Street/HouseNumber/UUID$$
$$/District/Street/HouseNumber/Type/UUID$$
$$/Type/Street/HouseNumber/UUID$$
$$/Type/Latitude/Longitude/UUID$$
$$/Type/Longitude/Latitude/UUID$$
$$/UUID/Type$$

Having both orderings for latitude and longitude makes regional searches in the shape of rectangles either aligned with the latitude or longitude more efficient. Searches covering for example a wider range of latitude values identifiers with the longitude attribute before the latitude is chosen, since matching nodes are located closer together within the SkABNet due to the smaller range of longitude values.

With these compositions defined, subscribers can search for publishers with specified attributes and SkABNet orders these searched attributes according to the most efficient attribute composition and therefore only searches a fraction of the overall SkABNet.

V. Evaluation

In the previous section, we have demonstrated that SkABNet provides more complex searches than SkipNet. We have further motivated that SkABNet searches are more efficient than comparable searches in SkipNet that lead to the same result. Here, we perform a quantitative analysis to compare the efficiency of both data structures. Therefore, we first evaluate a set of 14 search queries, which we execute on networks of different sizes. Second, we provide an numeric example that shows the effect of different attribute compositions, i.e., what happens when we change the order of the attribute-value pairs in the SkABNet identifier.

A. Evaluation Setup

To evaluate our SkABNet architecture, we implemented a simulation in *Omnet++* [21] that generates SkABNets of various sizes and run arbitrary search queries. Additionally, we re-implemented SkipNet and its search algorithm as a

Search Name	Example Search	#SkipNet Searches
S1 Single Value	Searching for a specific publisher participating in the network.	1
S2.1 Selection at the end	Searching for publishers in a given district and street providing temperature and humidity data.	2
S2.2 Selection in the middle	Searching for publishers in a given district located at 4 different streets providing temperature data.	4
S2.3 Selection at the start	Searching for publishers located on a street passing through 5 districts collecting temperature data.	5
S2.4 Multiple Selections	Searching for publishers located on 3 different street passing through 3 districts collecting temperature or humidity data.	18
S3.1 Range at the end	Searching for publishers located at a street segment defined by a range of house numbers collecting temperature data.	19
S3.2 Range in the middle	Searching for publishers located within an area defined by a range of streets providing temperature data.	15
S3.3 Range at the start	Searching for publishers located on a range of districts providing temperature data.	7
S3.4 Multiple Ranges	Searching for publishers located on a street segment defined by a range of house numbers passing through a range of districts providing temperature data.	57
S4.1 Wildcard at the end	Searching for publishers of any data located at a specified street within a single districts.	1
S4.2 Wildcard in the middle	Searching for publishers located in a district independent of the street providing temperature data.	40
S4.3 Wildcard at the start	Searching for publishers located on a street independent of the district the street traverses providing temperature data.	10
S5.1 Selection and range	Searching for publishers providing temperature, humidity and air quality data located on a street segment defined by a range of house numbers.	48
S5.2 Selection, range & wildcard	Searching for publishers of any data located on a street segment defined by a range of house numbers traversing two districts.	34

TABLE II: List of searches issued on simulated SkABNets. These searches can be expressed with a single SkABNet search or a number of SkipNet searches indicated by the last column.

benchmark for SkABNet's attribute-based search. To quantify the efficiency of performed searches we compare the total number of search-related messages transmitted between nodes in the network (*message complexity*).

For the simulations, we used the IoT context from above. Publishers are described with the following attributes: *District*, *Street*, *HouseNumber*, *Type*, and a *UUID*. Each publisher represents a sensor that publishes a data stream. Sensor types is randomly chosen from a set of ten possible data types. Further, sensors are randomly located in a fictive city consisting of ten districts and 40 streets each containing 200 house numbers. On initialization, sensors are assigned random values for these attributes, distributing them uniformely across the city.

B. Search Efficiency

To evaluate SkABNet's search efficiency, we defined a total of 14 search queries in which we inserted operators such as selections, ranges, and wildcards in different positions. We also defined a set of corresponding queries that are required to get the same results in the SkipNet. Table II provides an overview about these queries. The column *#SkipNet searches* indicates how many SkipNet queries are required to find the same publishers. It is to be noted that the translation from SkABNet searches into multiple SkipNet queries for ranges and wildcards only works since we know all possible values for the attributes. Other attributes such as the floating point values *latitude* and *longitude* as representations for a location could not be represented by a basic SkipNet search due to infinite amount of values that these attribute can represent.

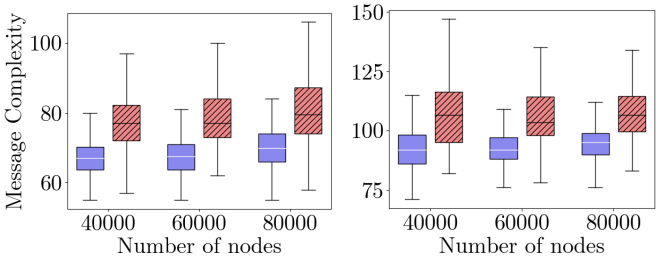
S1 uses single values for all attributes and, therefore, finds a single publisher only. All other searches will find exactly 50 publishers. This makes it easier to compare the efficiency of searches across all three network sizes (40 000, 60 000, 80 000 nodes). For each network size, we randomly created 100 networks and performed the 14 SkABNet searches as well as the equivalent 14 sets of SkipNet searches. The message complexity represents the total number of search-related messages that are transmitted for the single SkABNet search (blue bar in the following figures) and the set of SkipNet searches (red hatched bar).

Table II shows that the number of SkipNet searches that is required to match a single SkABNet query varies significantly and depends on how much matching nodes are scattered across a network. It also stands out that for *S1* and *S4.1* only one SkipNet search is necessary. Therefore, these searches behave exactly the same.

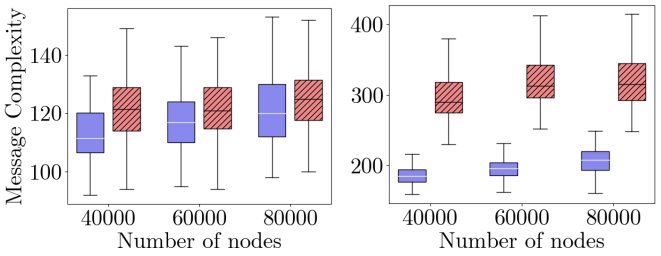
Fig. 5 shows the effect of one or multiple selection operators in a search query. A selection in the end requires less messages than a selection in the middle or the beginning of a query. For combinations of multiple selections, the effects add up and even more messages are exchanged. The individual plots in Fig. 5 also illustrate that SkABNet consistently uses less messages than SkipNet. The savings range from about 14% for selections in the beginning of the query to 7% when the selection is at the end. For multiple selections SkABNet even requires about 38% less messages than SkipNet.

Similar results can be observed for the range operator in Fig. 6. Here, the effect is even greater. While SkipNet needs to perform a large set of searches, SkABNet benefits from the parallelization of the search process. For a single range operator, on average about 75% of all messages can be saved using mulitple ranges, this number increases to roughly 90%.

Finally, Fig. 7 shows the effect of wildcards ($S4.2$ and $S4.3$) as well as combinations of multiple operators ($S5.1$ and $S5.2$). Again, the enhanced search algorithm in SkABNet results in significantly less messages compared to SkipNet and saves of up to 52% using wildcards and 88% in the combination of all search operators. Despite these promising numbers, we do

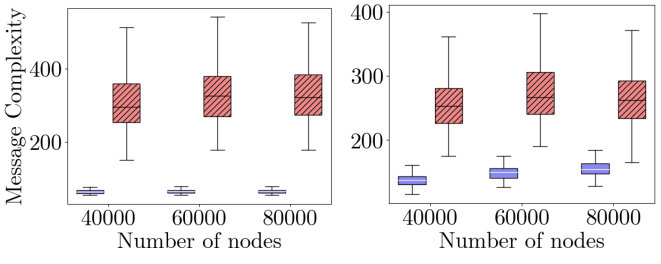


(a) *S2.1 Selection at the end* (b) *S2.2 Selection in the middle*

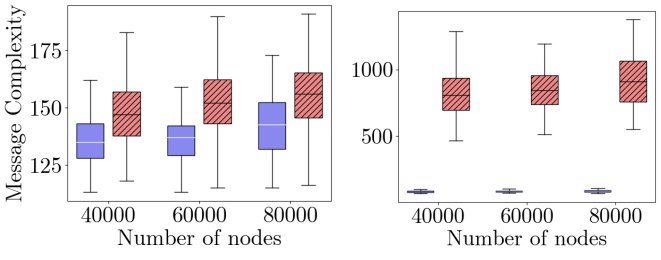


(c) *S2.3 Selection at the start* (d) *S2.4 Multiple Selections*

Fig. 5: Searches utilizing the selection operator at different positions within a search query.



(a) *S3.1 Range at the end* (b) *S3.2 Range in the middle*

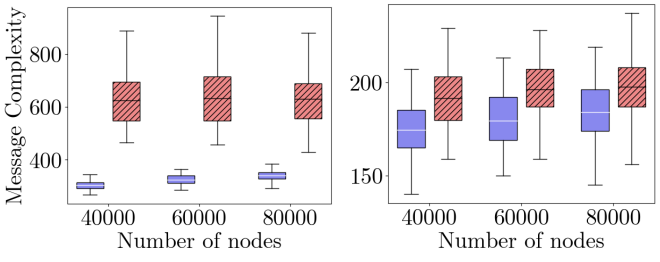


(c) *S3.3 Range at the start* (d) *S3.4 Multiple Ranges*

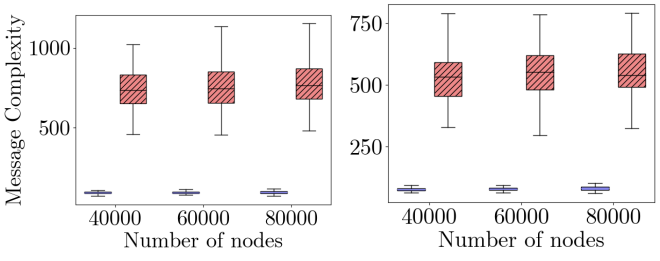
Fig. 6: Searches utilizing the range operator at different positions within a search query.

not want to emphasize to much on the absolute values here as they are highly dependent on how the network is created, how the parameter combination is chosen, and which searches are executed. However, the results show that, in general, large savings can be expected when using SkABNet.

Overall, we can see that efficiency of SkABNet searches depends mainly on two parameters. First, the number of replaced values defines how many basic SkipNet searches are needed to achieve the same results. These differences can easily be seen by searches utilizing range and wildcard operators. Selection



(a) *S4.2 Wildcard in the middle* (b) *S4.3 Wildcard at the start*



(c) *S5.1 Combination of selection and range* (d) *S5.2 Combination of selection, range & wildcard*

Fig. 7: Searches utilizing the wildcard operator at different positions and searches composed of different search operators.

operators, often replace only a small number of known values, therefore these searches can be expressed with a smaller set of SkipNet searches. Second, Fig. 5c, 6c and 7b confirm that search operators in the beginning of a search query reduce the advantage of SkABNet over SkipNet. This can be explained by the distribution of matching nodes within the network. If a search ranges over multiple values within the first attribute matching nodes are scattered over the network as shown in Fig. 4a leading to more messages needed to reach matching nodes. None the less worst-case szenario lead to the same message complexity as in the standard SkipNet.

C. Attribute Composition

In Sec. IV-C, we have discussed how re-arranging attributes in the identifiers impacts the search efficiency. As a last part of our evaluation, we now provide a quantitative example to demonstrate this effect. Therefore, we created three search queries that each benefits form a different composition of attributes and match 50 publishers. We then started three different SkABNets each containing 50 000 publishers. The first provided a single attribute composition only beneficial for the first search. The second SkABNet added an additional compositions, therefore doubleing the network size, but having beneficial compositions for the first and second search. Lastly a third SkABNet defining an additional composition, so that each search query has the most beneficial attribute order. As before experiments were repeated 100 times.

Fig. 8 shows how many search-related messages are transmitted for each individual search within the SkABNet containing one, two, or three compositions (x-axis) to find the 50 relevant nodes. When only a single composition is started, Search #1 performs well, while the other two suffer from the

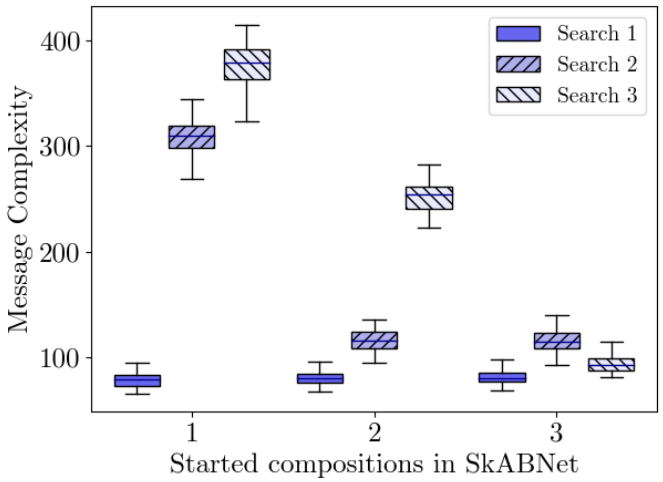


Fig. 8: Message complexity for the three search queries within three SkABNet providing different attribute compositions.

relevant nodes being scattered across the network (hatched boxes). Within the second SkABNet with two attribute compositions, Search #2 performs well, too. Finally, within the SkABNet providing three different compositions and therefore containing 150 000 nodes, all three search queries perform well and the total number of search-related messages across all three searches can be reduced by 50%.

Even though this scenario has been tailored to this particular use case, it shows the relevance of selecting "good" attribute composition. The question of what is a good composition needs to be answered for each context individually. It also suggests that starting multiple compositions can improve search efficiently significantly at linear cost for each node.

VI. Conclusion

In this paper, we presented SkABNet as a distributed data structure based on SkipNets [11] and specifically introduced attribute-based identifiers for IoT data streams in order to increase the efficiency of search query executions. For that, SkABNet introduces four search operators that allow for more complex attribute-based search queries and allow subscribers of data streams to better express required data. In a related qualitative evaluation, we demonstrated that, e.g., utilizing these complex search queries the number of messages needed to find all matching publishers can be reduced by up to 90%. To further increase efficiency across different search queries, publishers are represented by multiple attribute compositions. We could also demonstrate that providing attribute compositions for most common search queries reduces the overall search messages by roughly 50%.

Future evaluations shall concentrate on our SkABNet implementation of the SkABNet method within a deployed IoT sensor network - focusing especially on its resilience against connectivity issues and message loss. In addition, we will investigate self-managed attribute compositions. Currently, attribute compositions are defined when the SkABNet is initialized, which means that most common search queries must already be known. In the future, publishers shall continuously analyze occurring searches and coordinate starting more efficient compositions automatically whenever needed.

References

[1] M. Achir, A. Abdelli, L. Mokdad, and J. Benothman, "Service discovery and selection in iot: A survey and a taxonomy," *Journal of Network and Computer Applications*, vol. 200, p. 103331, 2022.

[2] R. Krishnamurthi, A. Kumar, D. Gopinathan, A. Nayyar, and B. Qureshi, "An overview of iot sensor data processing, fusion, and analysis techniques," *Sensors*, vol. 20, no. 21, p. 6076, 2020.

[3] T. A. Butt, I. Phillips, L. Guan, and G. Oikonomou, "Adaptive and context-aware service discovery for the internet of things," in *Internet of Things, Smart Spaces, and Next Generation Networking: 13th International Conference, NEW2AN 2013 and 6th Conference, ruSMART 2013. Proceedings*. Springer, 2013.

[4] Y.-W. Kuo, C.-L. Li, J.-H. Jhang, and S. Lin, "Design of a wireless sensor network-based iot platform for wide area and heterogeneous applications," *IEEE Sensors Journal*, vol. 18, no. 12, 2018.

[5] W. Osamy, A. M. Khedr, and A. Salim, "Adsda: adaptive distributed service discovery algorithm for internet of things based mobile wireless sensor networks," *IEEE Sensors Journal*, vol. 19, no. 22, 2019.

[6] H. Moeini, I.-L. Yen, and F. Bastani, "Efficient caching for peer-to-peer service discovery in internet of things," in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 196–203.

[7] A. Rowstron, P. Druschel, and R. Guerraoui, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," *18th IFIP/ACM International Conference on Distributed Systems Platforms*, pp. 329–350, 2001.

[8] 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.

[9] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph, "Tapestry : An Infrastructure for Fault-tolerant Wide-area Location and Routing," *Science*, vol. 74, no. April, p. 46, 2001.

[10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," *Computer Communication Review*, vol. 31, no. 4, pp. 161–172, 2001.

[11] N. J. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman, "Skipnet: A Scalable Overlay Network with Practical Locality Properties," *Proceedings of the Fourth USENIX Symposium on Internet Technologies and Systems*, vol. 5, p. 9, 2003.

[12] J. Aspnes and G. Shah, "Skip graphs," *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 384–393, 2003.

[13] 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.

[14] S. Cirani and L. Veltri, "Implementation of a framework for a dht-based distributed location service," in *2008 16th International Conference on Software, Telecommunications and Computer Networks*. IEEE, 2008.

[15] M. Picone, M. Amoretti, and F. Zanichelli, "Geokad: A p2p distributed localization protocol," in *8th IEEE International Conference on Pervasive Computing and Communications Workshops*. IEEE, 2010.

[16] F. Paganelli and D. Parlanti, "A dht-based discovery service for the internet of things," *Journal of Computer Networks and Communications*, vol. 2012, 2012.

[17] W. Pugh, "Skip lists: a probabilistic alternative to balanced trees," *Communications of the ACM*, vol. 33, no. 6, pp. 668–676, 1990.

[18] J. Li, N. Zaman, and H. Li, "A decentralized locality-preserving context-aware service discovery framework for internet of things," in *2015 IEEE International Conference on Services Computing*. IEEE, 2015.

[19] Y. Ishi, Y. Teranishi, M. Yoshida, S. Takeuchi, S. Shimojo, and S. Nishio, "Range-key extension of the skip graph," in *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*. IEEE, 2010, pp. 1–6.

[20] R. Banno and K. Shudo, "An efficient routing method for range queries in skip graph," *IEICE TRANSACTIONS on Information and Systems*, vol. 103, no. 3, pp. 516–525, 2020.

[21] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment." ICST, 5 2010.