Lightweight Messaging for Efficient Service Discovery in Mobile IoT Environments Using Hierarchical Bloom Filters

Hyeon-Jun Jo, Jung-Hyun Kwon, MinHyeop Kim*
and In-Young Ko

School of Computing, Korea Advanced Institute of Science and Technology (KAIST), 291 Daehak-ro, Yuseong-gu, Daejeon, South Korea
E-mail: hyeonjun.jo@kaist.ac.kr; junghyun.kwon@kaist.ac.kr; minhyeop.kim@kaist.ac.kr; iko@kaist.ac.kr
**Corresponding Author*

Received 22 January 2019; Accepted 29 November 2019;
Publication 05 March 2020

Abstract

In highly dynamic IoT environments, the connection statuses of IoT resources and the availability of IoT-based services change frequently. Therefore, to successfully build distributed service registries for managing and finding the information about available services in an effective manner, it is crucial to minimize the overhead of message exchanges between registries and to reduce the time overhead for identifying the capabilities of available IoT resources and the services that can be provided by utilizing these capabilities. In this paper, we propose a lightweight messaging approach that uses hierarchical Bloom filters to efficiently represent service information to be exchanged and managed by distributed service registries for IoT environments with high mobility. We also propose a method for serializing the dimensions of a Bloom-filter-encoded search space. We conducted experiments to demonstrate the improvement in the service discovery performance, the reduction in message traffic among service registries, and the decrease in the latency when synchronizing distributed service registries.

Journal of Web Engineering, Vol. 19_1, 29–62.
doi: 10.13052/jwe1540-9589.1912

Keywords: Service discovery, service registry, Bloom filter, mobile IoT environments, service-oriented architecture.

1 Introduction

Real-world objects, from small sensors to vehicles, have networking capability nowadays and can be part of an Internet of Things (IoT) environment, where useful services can be provided for users by utilizing IoT devices [4, 7]. In particular, mobile devices such as smartphones, tablet PCs, and wearable devices are becoming more powerful and common in everyday life, and they are regarded as important parts of IoT environments. We define a *mobile IoT environment* as an IoT environment, in which mobile devices are connected and interact with each other to provide services to its users. For this environment, a mobile ad-hoc network (MANET) [14] is an efficient technology for making connections among mobile IoT devices and enabling them to interact with each other to provide users with IoT-based services [6].

Owing to the mobility of many IoT devices in a MANET environment, the connectivity among the IoT devices is highly dynamic and spontaneous. Therefore, it is a challenging issue to discover and provide services that require the capabilities of mobile IoT devices in a reliable and efficient manner. In addition, because the statuses of the services that can be offered by utilizing mobile IoT devices are dynamically changing [22], it is necessary to effectively monitor and manage the availability and statuses of IoT-based services in a large-scale IoT environment.

As we explained earlier, services must access the capabilities of IoT devices to provide their functionalities; therefore, we regard IoT devices as the resources of the services. In addition, IoT-based services can be coordinated together to perform a user task. A *user task* is usually represented as a composition of services that provide the necessary capabilities for accomplishing a user goal. Therefore, a challenging issue is to find the appropriate services that are required for a user task in a highly dynamic IoT environment with mobility.

Unlike service discovery in traditional web-service environments such as Universal Description Discovery and Integration (UDDI),[1] in a mobile IoT environment, it is essential to deal with the dynamic changes in service availability and the connectivity among resources effectively. Therefore, service discovery in mobile IoT environments needs to be performed by keeping

[1] http://uddi.xml.org/

track of the configuration of IoT resources and the availability of the services that utilize them [26], finding available services in a dynamic manner for a user task, and managing the information of runtime service statuses in a distributed manner [1]. In particular, it is crucial to rapidly update the service-status information stored in a distributed service registries in a large-scale IoT environment.

In our previous work [11], we proposed a distributed service registry system that enables fast and flexible service discovery in highly dynamic mobile IoT environments to meet these requirements. In that work, we especially considered the characteristics of IoT environments with mobility, i.e., the low computing power of IoT devices and the limited network bandwidth between the devices. Therefore, to successfully build distributed service registries for storing and managing information about available services in a mobile IoT environment and to find the necessary services in an effective manner, it is necessary to minimize the overhead of message exchanges between registries and to reduce the time overhead for identifying the capabilities of the available IoT resources and the services that can be provided by utilizing these capabilities.

We applied hierarchical Bloom filters (HBFs) to reduce the costs associated with identifying available services and tasks as well as the message exchanges that occur when updating the information about the available services in dynamic mobile IoT environments. By applying HBFs, we represent the capabilities of an IoT resource or a service in a unique integer vector in a coordinate system. An integer vector contains the type information associated with a resource or service; it is presented in the unique space of a coordinate system. Identical capabilities share the same space in a coordinate system, and the integer vectors that represent similar capabilities are presented in nearby spaces. This makes the processes of finding a type of service and identifying alternative services fast and efficient by reducing the network traffic and time overhead.

In highly dynamic mobile IoT environments, the connection statuses of IoT resources and the availability of IoT-based services change frequently. In addition, the location of a resource or service within a MANET environment changes as the device or a user moves around. Therefore, in this paper, we propose an approach for dynamically updating distributed registries in an efficient manner to reflect the changing statuses of resources and services that can be performed by using the resources. To achieve this goal, several requirements must be met. First, the status changes of IoT resources and services should be propagated to all service registries immediately. Second, the

process of discovering the capabilities of resources and services needs to be more efficient for rapid reflection of the environmental changes to the registries.

To meet these requirements, an extended coordinate system is used to represent the capabilities of IoT resources and services in more compact spaces. Only the parts that are affected by the status changes of IoT resources and services are updated in the coordinate system. This results in fast and efficient updating of the distributed service registries, even when frequent changes occur in a mobile IoT environment. In addition, we developed a method that reduces (serializes) the dimensions of an encoding space by removing unused spaces and by dynamically allocating space as needed. With this serialized search space, resource capabilities and services can be determined much faster than the traditional approach of exchanging and parsing XML-based service descriptions.

We conducted experiments to demonstrate the improvement in the service discovery performance, the reduction in the message traffic among service registries when updating the information about the available IoT resources and services, and the decrease in the latency when synchronizing distributed service registries.

The remainder of the paper is organized as follows. In Section 2, we discuss the existing work on distributed service discovery. The characteristics of the mobile IoT-based service environment are described in Section 3. In Section 4, we explain the proposed approach involving the use of HBFs for building an efficient distributed service registry system. Then, Section 5 presents how available services are identified using the proposed approach. Section 6 presents our experimental results and analyses. Finally, Section 7 concludes the paper with a discussion of our contributions and future work.

2 Related Work

2.1 Distributed Service Registries

To overcome disadvantages such as the bottleneck and single point of failure problems of centralized registry systems, there have been efforts to develop distributed service registries. One of the efforts is the development of a backbone-network-based approach [15, 18]. In this approach, a backbone network is formed using several nodes in an ad-hoc manner, and service registries are installed on the network nodes. This approach makes it possible

to discover and manage services in distributed network nodes. However, it may produce a considerable amount of network traffic because it broadcasts queries to discover services to all registries in the network. There is also an overhead associated with the reconfiguration of the backbone network whenever the statuses of the network nodes change.

To overcome the disadvantages of backbone-network-based approaches, service clustering methods have been developed [13, 19]. Service clustering methods classify services in a network in different groups on the basis of their types (e.g., node location) and choose a representative network resource to manage the services in each group. This approach can reduce network traffic by sending service discovery queries only to specific service groups. However, it incurs the additional overhead of selecting a new representative network resource for a service group when the network changes. Although it is possible to use a static clustering method, i.e., one that does not consider network changes, such a method can still be problematic when the representative network resource is not available or not reachable.

As another effort to realize distributed service registries, domain name system (DNS)-based approaches have been proposed [10, 12, 17]. In these approaches, a service is registered with its description to a DNS server in a network environment. Then, the registry broadcasts or multicasts the service description to the other registries on other DNS servers in the network environment. Although these approaches are scalable to manage and a large number of diverse services in a distributed manner, the interactions among service registries usually generate a large amount of network traffic, which may not be acceptable for mobile IoT environments, where there are constraints on the network connections [12]. These approaches also incur the additional cost of configuring the structure of the service registries with the DNS servers [17]. Therefore, in a highly dynamic IoT environment, the configuration cost will be increased significantly.

There have also been efforts to use a distributed hash table (DHT) for service discovery [20, 21]. In such an approach, each service has its own unique key, and the key and location information of the service are stored in a hash table. Multiple registries that are distributed in a network manage the service hash table. The service location information refers to a point in the entire area formed by all of the service registries. However, in practical network environments, it is difficult to assume that each registry knows the exact location of a service rather than the relative location of the service within its local network.

2.2 Use of Bloom Filters for Service Registries

To make service management more efficient, there has been research on the use of a Bloom filter for service discovery.

A *Bloom filter* is a probabilistic data structure used to efficiently manage a large amount of data in a small memory space [2]. A Bloom filter is composed of a bit array and a set of hash functions. Once the bit array is created by undertaking hash functions for a set of elements, efficient checks of whether or not a specific element is in the set without searching all of the elements become possible. Certain lookup mechanisms are necessary to implement a Bloom filter to avoid unnecessary searches [8, 9]. The hash functions generate a series of integers for each element. These integers represent the positions (indices) in the bit array. The number of integers is determined on the basis of the number of hash functions. In other words, every element is assigned a unique sequence of integers as a result of performing the hash functions. It is able to determine whether or not an element is in the set by checking specific positions in the bit array.

Sailhan et al. applied a Bloom filter to service descriptions and reduced the size of the service information stored in a registry [17]. Their study, however, did not apply a Bloom filter to the service-discovery queries exchanged among service registries. Cheng et al. proposed a counter Bloom filter to overcome a limitation of the traditional Bloom filter, in which stored hash values cannot be deleted after they are generated [5]. Using the counter Bloom filter, the service specifications that are stored in a registry can be deleted, and it is possible to reduce the size of the registry. However, this method cannot reduce the network traffic that arises when registries exchange service information.

Zhang et al. [24] proposed an approach for efficiently creating and managing hierarchical clusters of services by using a Bloom filter. In their approach, keywords are extracted from service description texts, and services are hierarchically clustered using a deterministic annealing algorithm while associating each of the clusters with a value generated by applying a Bloom filter to the corresponding keywords. Then, by applying Bloom filters to the keywords that are included in users' service queries, appropriate service clusters can be searched in an efficient manner. This approach is similar to our approach in the sense that a Bloom filter is used to efficiently find services. However, they use a single Bloom filter to locate a service cluster that is matched against a user's query. In contrast, we use multiple Bloom filters that are hierarchically organized according to the service ontology to

discover services across distributed service registries in an efficient manner with a reduction in the message traffic between the registries.

The related works discussed in this subsection mostly focused on rapid information searches within a service registry using a Bloom filter. In contrast, our study seeks not only to search for service information rapidly inside a registry but also to reduce the network traffic exchanged among registries through its use of an HBF, as explained in Section 4. Furthermore, our study proposes an approach that reflects the dynamic changes in distributed service registries in a mobile IoT environment in an efficient manner.

3 Service Provision in IoT Environments

The IoT environments considered in this work is composed of various IoT resources and service gateways that are distributed in a geographical region. The IoT resources and service gateways may have mobility and can move around in a specific geographical region. The IoT resources can be various types of devices, ranging from tiny sensors to security cameras on streets as well as smartphones and other resources. Each resource provides a set of *resource capabilities*, which can be utilized to provide services for users. Because many of the IoT resources do not have sufficient computation and/or networking power to provide services directly to users, there are *service gate ways* that proactively find IoT resources in the surrounding environment and make them available to provide services.

A service needs to be bound to required resource capabilities in order to generate its service capability. The quality of a service is determined according to the capabilities of the resources that are bound to the service. The binding between a service instance and the IoT-resource is not a tight coupling. The binding can be made or removed dynamically for the need of users. A *user task* is represented as a composition of *abstract services* that define the necessary capabilities to accomplish a user goal. A user task needs to be bound to *service instances* that provide service capabilities by utilizing IoT resources. There may be multiple service instances available for an abstract service of a task, and a service instance is chosen on the basis of the quality requirements of the user.

Figure 1 illustrates an IoT environment with mobility that is composed of the following core components. The *task layer* is for selecting and running a user task, and the *IoT resource layer* is composed of service gateways, service registries, and IoT resources. A *service registry* that resides on a service

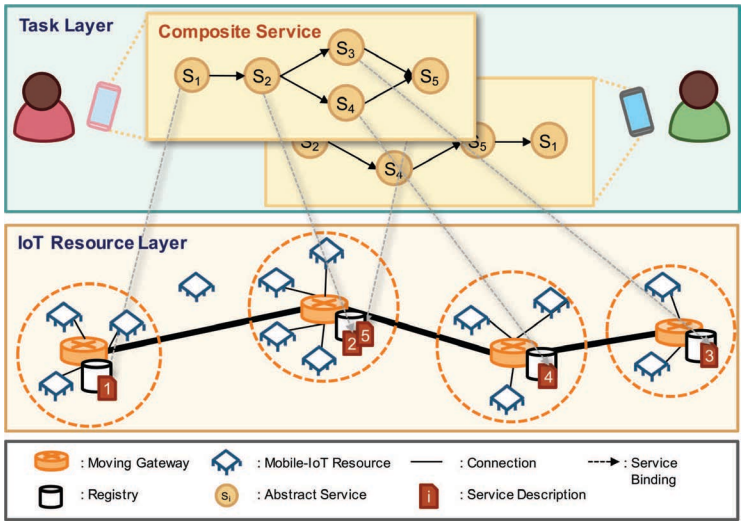


Figure 1 Logical layers of a IoT environment.

gateway discovers nearby IoT resources and monitors the changes in the surrounding environment to update the status information about the available IoT resources and services. A gateway has a connection range in which it can make connections to IoT resources. Gateways can be also connected to each other, and the service registries in the gateways can exchange information about the available IoT resources and services. *Distributed service registries* can be formed by associating the service registries that are connectable via service gateways in a specific geographical region.

A service registry stores the information about the IoT resources that are connected to the corresponding service gateway. It also identifies the services that can be provided by utilizing the IoT resources, and it maintains the information about available services. Service registries also collaborate with each other by exchanging messages to monitor and share the status information about available services. Therefore, it is possible to obtain information about available services by querying any of the service registries in the MANET environment. For instance, if an IoT-resource is connected to two different gateways at the same time, then two gateways shares the information about the resource, and identify the as a service. If two gateways are not connected to each other, the IoT-resource can be registered as two different services to each gateway.

4 Hierarchical Bloom-Filter-Based Light-Weight Messaging for Distributed Service Registries

4.1 Hierarchies of IoT-Resource and Service Capabilities

For the efficient management of IoT resources and services, we have developed an ontology model to represent the relationship between the conceptual entities that are related to the IoT resources and services [11]. Figure 2 shows the model, which is composed of the tasks, services, resources, and capabilities. Tasks, services, and resources are the entities that can provide their own capabilities, and the capabilities can be compared against each other in a semantic manner by the service discovery engine based on the ontology. As explained earlier, a user task is represented as a composition of services that requires the corresponding service capabilities to provide the task capabilities; a service requires certain resource capabilities to deliver its service capability. Before service instances are bound to a task, the task consists of abstract services that specify the necessary service capabilities. Therefore, service capabilities that are needed by a task can be identified by services composing the task. In addition, each service capability must be provided by a service instance which utilizes the capability of an IoT-resource.

To enable more flexible service discovery in a large-scale mobile IoT environment, the required IoT-resource capabilities for a service need to be specified at different abstraction levels. All IoT resources that provide capabilities at a certain abstraction level that a service specifies, or more specialized capabilities than those of the abstraction level, need to be considered as candidate IoT resources for the service. To support this, the IoT-resource capabilities are defined hierarchically in a representation of the subsumption

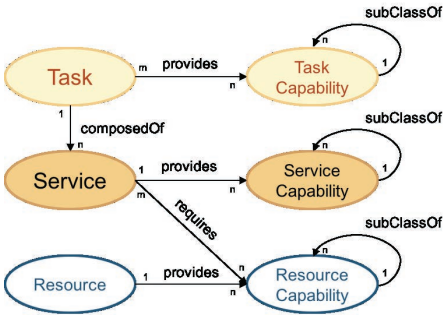


Figure 2 An ontology model for representing the relationships among IoT resources, services, and tasks.

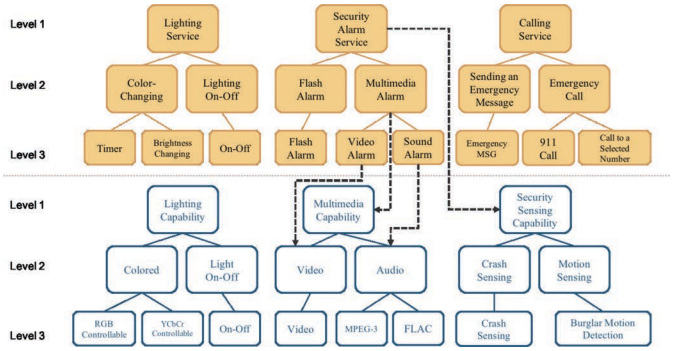


Figure 3 Example hierarchies of service capabilities and IoT-resource capabilities along with the associations between them.

relationships (using the "subClassOf" relationship) between the capabilities in the ontology model.

Services are also similarly defined by representing the subsumption relationship between service capabilities. The abstract services of a user task define their necessary service capabilities, which can be represented at various abstraction levels, and a query for finding service instances for an abstract service can be carried out according to the specified abstraction levels. With these types of hierarchies of IoT-resource capabilities and services, it is also possible to identify services that can be run on the basis of the IoT-resource capabilities in an available MANET environment. In addition, after recognizing the available services, the user tasks that can be performed in the MANET environment can be identified.

Figure 3 shows an example of the IoT-resource and service capability hierarchies and the associations between them. In real world environment, we expect that a domain expert can build the hierarchy of service and IoT-resource capability. There are three hierarchies of service capabilities at the top. The "Lighting" service can be specialized into the "Light On—Off" and "Color-changing" services, which provide more specialized capabilities for controlling the light colors. If a light needs to be controlled with a timer, the "Timer" service can be selected. The "Flash Alarm" and "Multimedia Alarm" services are more specialized services (defined with their alarm method) than the "Security Alarm Service," which defines general alarming services.

The lower part of Figure 3 shows the three different hierarchies of IoT-resource capabilities: the "Lighting," "Multimedia," and "Security Sensing" capabilities. Similar to the service-capability hierarchies, these IoT-resource

hierarchies have multiple levels of abstraction. For example, there are the "Colored" lighting and simple "Light On–Off" capabilities under the "Lighting" capability; the "Colored" lighting capability can be further specialized into the "RGB Controllable" and "YCbCr Controllable" capabilities. If a service requires the "Colored" lighting capability, any of the IoT resources that provide either of these specialized capabilities can be selected and bound to the service.

If the required IoT-resource capabilities are not present, a service cannot be instantiated. For example, as shown in Figure 3, in order to instantiate a "Security Alarm Service," both the "Security Sensing" and "Multimedia" or "Lighting" resource capabilities need to be available in the IoT environment.

4.2 Hierarchical Bloom Filters

In our previous work, we defined a *hierarchical Bloom filter (HBF)* as a set of Bloom filters, each of which corresponds to one level of a capability hierarchy [11]. Figure 4 shows an example of the application an HBF to the "Lighting" capability hierarchy discussed in the previous subsection. In this example, we use two hash functions for each Bloom filter; therefore, each capability is converted into two integers. As shown in the figure, the "Lighting" capability is represented by the integers 5 and 6, the "Colored" lighting capability is converted to 5 and 7, and the integers 2 and 7 are assigned to the "RGB Controllable" lighting capability.

Although a Bloom filter has a 100% recall rate, which means that there are no false-negative errors, there may be false-positive errors. If there are false-positive errors, the IoT-resource capabilities and/or services that are not actually available may appear as if they are available during the service discovery process. We can configure the rate at which false-positive errors

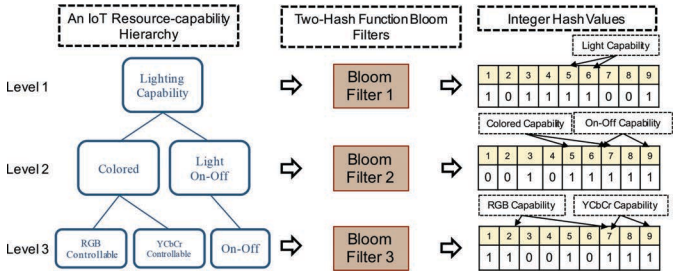


Figure 4 Application of hierarchical Bloom filters to a resource-capability hierarchy [11].

are generated by resizing the number of hash functions and the length of the bit array.

$$\left(1 - \left[1 - \frac{1}{m}\right]^{kn}\right)^{k} \approx \left(1 - e^{-\frac{kn}{m}}\right)^{k} \tag{1}$$

Equation (1) explains how to calculate the rate of false-positive errors [16]. In this equation, n is the number of elements managed by a Bloom filter, k is the number of hash functions, and m is the length of the bit array. Equation (1) indicates that the use of the Bloom filter involves a trade-off between the length of the bit array and the rate of false-positive errors. An increase in the number of hash functions causes an increase in the dimensions of the search space and generates more computational overhead in the discovery of services. Therefore, for our service discovery approach, we first determine the number of hash functions for a Bloom filter and then adjust the length of the bit array to set the acceptable rate of false-positive errors.

4.3 Search Space Serialization

In our approach, the number of Bloom filters depends on the depth of a capability hierarchy. A different Bloom filter is used for each level of the hierarchy. Because each level of the hierarchy may have a different number of capabilities, we determine the number of hash functions on the basis of the maximum number of siblings in the hierarchy. All the Bloom filters have the same number of hash functions and the same number of dimensions for the search space for all levels.

We use a bottom-up approach to find available services from the available IoT-resource capabilities discovered in a MANET environment. To realize this approach in a highly dynamic mobile IoT environment, it is critical to reduce the overhead of exchanging and processing information about the available capabilities across service registries. In addition, because the integers that are generated by the Bloom filters do not have any regularity but are mostly random, we need a method to represent the hierarchical relationships between levels of capabilities. To meet these requirements, we have developed a method that maps the hash values from Bloom filters to a coordinate space, defining a search space for discovering IoT-resources and service capabilities.

The search space can be encoded using a specific unit such as the digits of numbers (e.g., the tens place of 712 for the second level in a hierarchy and the hundreds place for the first level). In this case, the search space will be sparsely filled with encoded capabilities, as a fixed number of spaces

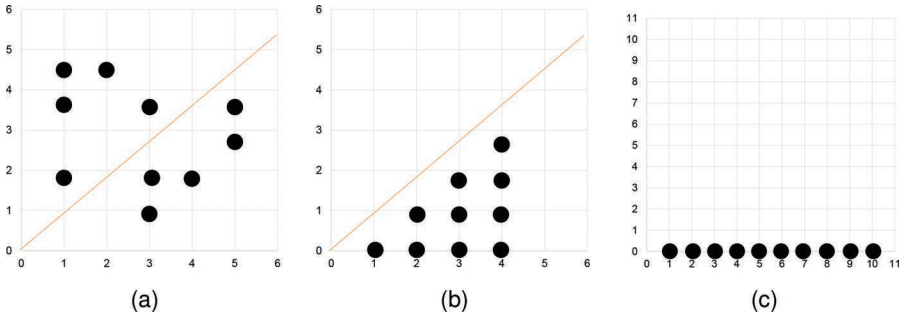


Figure 5 The process of serializing hash values from Bloom filters. The two-dimensional search space that contains hash values (a) is reduced to (b) by flipping the values in the upper triangular part of the space. Then, with Equation (2), the space (b) is flattened to a linear space (c).

depending on the unit is set regardless of the number of capabilities, as shown in Figure 5(a). As a result, much of the space will be unused.

Therefore, we developed a serialization method to make the multidimensional capability representation more compact but still efficiently searchable. Figure 5 shows the serialization process. Figure 5(a) shows the space allocated for a capability hierarchy using a two-hash Bloom filter. Because the two-hash Bloom filter does not produce duplicated values, there is no space allocated on the diagonal of the search space (coordinate system). In addition, given that the result of a Bloom filter is order-independent, the allocated space is not symmetrical with respect to the diagonal. Thus, despite the fact that we flip spaces to one side of the diagonal, as shown in Figure 5(b), there will be no conflict. Then, when *h1* and *h2* are the results of Bloom filters, by using Equation (2), we can serialize the allocated spaces into a linear space, as Figure 5(c) shows.

$$Serialize(h1, h2) = \begin{cases} \sum_{i=1}^{h2-1} (N - i) + (h1 - h2), & h1 > h2 \\ \sum_{i=1}^{h1-1} (N - i) + (h2 - h1), & h1 \leq h2 \end{cases}, \qquad (2)$$

Here, N is the maximum number of capability classes at each level of a hierarchy.

The serialization process is applied to all levels of a capability hierarchy. As a result, each capability is converted into the same number of integers as the number of levels in the hierarchy.

To solve the sparsity problem in search spaces and to deal with the dynamic changes in the availability of IoT-resource and service capabilities, we developed a method that efficiently represents the information of the available capabilities while retaining their hierarchy. We use mixed-numerical encoding, which dynamically calculates search spaces for each level in a hierarchy.

First, from the result of the serialization process performed in the previous stage, we obtain the maximum integer value of each level of a capability hierarchy. The maximum integer value refers to the maximum number of capability classes at each level of the hierarchy.

Second, we define a *key number* for each level of the hierarchy by adding 1 to the maximum integer value for an *abstract indicator*. A key number is the number of search spaces necessary for each level of the hierarchies. Because it is needed to allow users to search for services by representing the required capabilities, we use *abstract indicators*, which are reserved spaces, to represent the abstract capabilities (the capabilities in a certain class). Equation (3) is the formula for generating the key numbers (k_l) for a hierarchy.

$$k_l = \begin{cases} 1, & l = 1 \\ k_{l-1} \times m_{l-1}, & l > 1 \end{cases}, \tag{3}$$

where *l* is the level in the hierarchy, and m_{l-1} is the maximum hash value at the l_{th} level of the hierarchy plus one.

Last, to convert the integer values of the capabilities at a certain level of a capability hierarchy into a unique hash value for the capability (x_c), all of the integer values are added after multiplying each integer value by the key number of the higher level of the hierarchy, as Equation (4) shows.

$$x_c = \sum_{l=0}^{L} k_l * h_{l,c}, \tag{4}$$

where *l* is the level in the hierarchy, *L* is the depth of the hierarchy, and $h_{l,c}$ is the serialized hash value for the level *l* of the capability *c*.

By doing this, we can set the boundary of the search space for each level dynamically, as Figure 6 illustrates. For instance, we multiply the integer values at the second level by the key number at the third level.

Algorithm 1 explains how to convert resource capabilities into vectors. First, the resource descriptions are hashed by Bloom filters (Line 5). Second, the results of the Bloom filters are serialized (Line 6). Last, the serialized hash values are converted into a one-dimensional vector (Line 7).

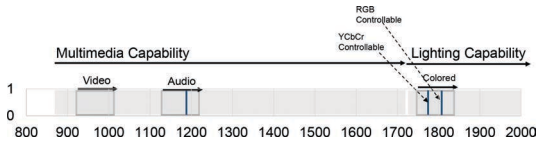


Figure 6 A serialized search space.

Algorithm 1 Convert resource capabilities into a hash value

Require: Capability *c*
Require: Capacity Hierarchy *ch*
Require: KeyNumbers *keynums*
1: *capNum* = 0 ▷ capability number
2: **while** *c* is not the root of *ch* **do**
3: *level* = *getHierarchicalLevel(c)*
4: *resourceDescription*=*getResourceDescription(c)*
5: *h1, h2* = *bloomFilter(resourceDescription)*
6: *serializedHash* = *serialize(h1,h2)*
7: *capNum*+=*serializedHash* * *keynums[level]*
8: *c* = *getParent(c)*
9: **return** *capNum*

Figure 6 shows an example of how a three-level capability hierarchy is represented in a serialized space. In the example, the maximum numbers of the three levels in the hierarchy are 3, 78, and 10, and the key values of each corresponding level are 869, 11, and 1, respectively. Figure 6 shows the spaces allocated for the levels in the hierarchy, where a higher-level space is divided into the spaces for the lower levels.

5 Service Identification and Monitoring

5.1 Service Identification

Distributed service registries play three main roles. First, each service registry maintains connections to its neighboring service registries. Second, a service registry discovers available IoT resources that are directly connectable to the corresponding service gateway, subsequently sharing the information about the available capabilities of the IoT resources with the neighboring service registries. Third, a service registry stores the information of deployed service instances and provides the service information to service clients.

To enable registries to discover available services using the spaces allocated for the capabilities of IoT resources, each service registry maintains

an identical space. The search space is initially filled with zeros to represent the "not discovered" state. The service registry then increases the count by one to indicate the IoT-resource capabilities that are discovered locally by the registry. If the IoT resource is disconnected, the service registry decreases the count by one for the space that is allocated for the corresponding resource capability. Then, the registry shares the updated information with neighboring registries. When update messages are sent to other registries, the service registry decodes the message. If a message includes the update information of the capability of its sender, the service registry increases the corresponding count by one to indicate that the capability information has been received. The registry also keeps the sender information so that the location of the IoT-resource capability is recorded. By exchanging update messages, all service registries can maintain consistent information about the available IoT-resource and service capabilities in an IoT environment.

When a service registry monitors the changes in the information stored in the spaces allocated for IoT-resource capabilities, it activates the service identification process. As described in Algorithm 2, a service registry identifies available services on the basis of the capabilities that can be generated by using the discovered IoT resources. This is done in a bottom-up manner, and it requires predefined information about the IoT-resource capabilities that are necessary to provide services. The predefined information is managed in the service table. To determine whether a service is available, the service registry obtains the required resource capabilities from the service table (Line 3). Then, the service registry determines whether or not each required resource capability is available by finding the leaf nodes in the capability hierarchy that match the encoded value of each resource capability (Lines 6 and 7). If all required capabilities are available, the service registry can confirm that the service is available.

Table 1 is a service table that stores the information about the required IoT-resource capabilities for services. By comparing the capabilities of the

Table 1 A service table that provides the information about the required IoT-resource capabilities for each service

Services	IoT-Resource Capability 1	IoT-Resource Capability 2	IoT-Resource Capability 3	...
A	$x1$	$x4$	$x6$	...
B	$x6$	$x3$		...
C	$x15$			...

IoT resources that are discovered in a local environment against the IoT-resource capabilities that are stored in the table, a service registry can identify the services that can be provided in the environment.

Algorithm 2 Find available services

```
Require: Service Table st
Require: Capability Hierarchy ch
availableServices = []
for service in st do
    capabilities = getRequiredResourceCapabilities (st[service])
    matchedResourceCapabilities = []
    for rc in capabilities do
        if rc is a leaf node of ch then
            matchedResourceCapabilities.append(rc)
    if capabilities == matchedResourceCapabilities then
        availableServices.append(service)
return availableServices
```

5.2 Service Monitoring

When a service instance is deployed to a service gateway, the service registry that resides in the gateway or a neighboring gateway converts the service-capability information into a set of vectors using an HBF. The service vectors are then stored in a certain space using the serialization process explained in the previous section. Finally, the service registry sends an update message to neighboring registries.

By exchanging the update message of a service, all distributed service registries can identify the location of the service. Status changes in the IoT resources, such as their availability and location, can also cause changes in the types and locations of service instances in the IoT environment.

Lastly, on the basis of the available services, user tasks that can be performed by coordinating the services are identified. Using vector conversion for the services and tasks, we reduce the message traffic and computation time for identifying available services and tasks. This enables quick updates of the information about IoT resources, available services, and tasks across gateways in a MANET environment. In addition, the distributed registries can constantly keep track of the statuses of all IoT resources, services, and tasks.

6 Experiments and Analysis

Experiments have been conducted to demonstrate the effectiveness of the distributed service registries that manage and exchange information about IoT resources and services using HBFs. First, we compare our approach against a baseline approach that uses the Efficient XML Interchange (EXI) format, which is a standard (W3C recommendation[2]) binary XML (BXML) format. In the baseline approach, service registries exchange resource capability descriptions in the EXI format and parse the descriptions to update their information about available services and tasks. In particular, we compare the latency of synchronizing distributed registries when there are changes in an IoT environment. In other words, when a service registry detects an update of the availability and status of IoT resources and services, it broadcasts the update information (a list of resource capability descriptions) to its neighboring registries.

We compare our HBF approach against the baseline approach in terms of the service identification time and the number of messages exchanged among service registries. In addition, to show the effectiveness of adding "hierarchies" of IoT-resource and service capabilities when representing and searching them, we compare the efficiency of the service discovery process between the nonhierarchical and hierarchical Bloom-filter approaches. We also conduct an experiment to assess the effectiveness of the approaches in terms of how they deal with mobility in IoT environments by changing the velocity of the mobile IoT resources and service gateways.

Table 2 Evaluation criteria

Criteria	Meaning	Unit
Service Identification Time	Time to identify available services using the discovered IoT-resource capabilities	Milliseconds (ms)
Message Traffic	Total amount of message traffic generated to share information among registries	Kilobytes (KB)
Update Latency	Time to update all service registries in an environment with updated information	Milliseconds (ms)

[2]Efficient XML Interchange (https://www.w3.org/TR/exi/)

6.1 Experimental Setting

In our experiments, we use a real-world web-services dataset that is composed of 3,738 web services collected from WS-DREAM[3] [25]. However, because this dataset does not include information about the capability hierarchies of services, we extract the operation information from each service description (written in WSDL) and regard the operations as necessary IoT-resource capabilities. The hierarchical structure of the resource capabilities is randomly generated in our experiments to avoid the bias introduced by manually creating the capability hierarchies. A service instance is considered as available when all necessary IoT-resource capabilities for performing the operations of the service instance are found in a MANET IoT-environment.

During the experiments, we check the effectiveness of the approaches while changing the number of IoT-resource capabilities, the number of services, the depth of the IoT-resource and service capability hierarchies, and the velocity of the mobile devices. We check the computing time and the amount of message traffic while changing the number of services from 10 to 100 and setting the average number of IoT resources to 50. In addition, we randomly allocate up to 100 IoT resources in an area with dimensions of 500 m $\times$ 500 m with 10 service registries.

We also test our approach while changing the depth of the hierarchies of the IoT-resource capabilities and services from one to five. In this test, the nonhierarchical Bloom-filter approach searches IoT-resource and service capabilities starting from a randomly selected point in a bit array, while the other hierarchical Bloom-filter approach searches the search space to which the required resource capability belongs.

Five trials are carried out for each experiment independently. Each trial is performed for 140s with the same set of settings. The velocity of the mobile devices is set to range from 0m/s to 10m/s to simulate the dynamicity of the IoT environments. This setting is similar to the simulation setting used in other studies [3, 23].

In the experiments, we use a C++ Bloom-filter library[4] to implement the HBFs. The false-positive probability of the Bloom filter is set to 0.01, which means that the possibility of generating the same encoded value for different IoT-resource or service capabilities at a certain level of a capability hierarchy is 1%. This setting is acceptable for non-mission-critical IoT environments.

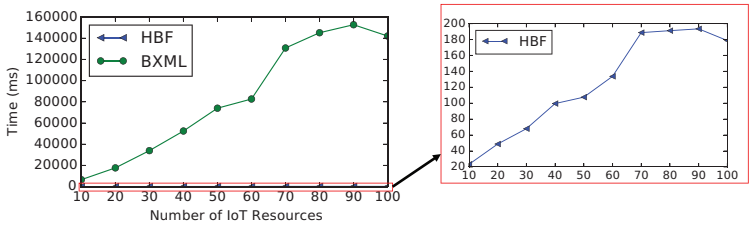
[3] https://wsdream.github.io/

[4] https://github.com/arashpartow/bloom

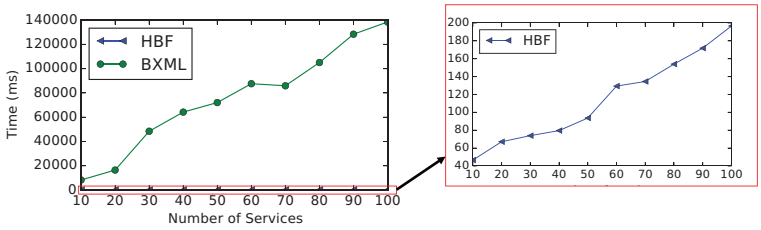
As explained earlier, this false-positive probability can be determined according to the domain requirement of an IoT environment. In addition, we simulate a mobile IoT environment with the ns-3 simulator,[5] which is commonly used in network simulations. In this simulation environment, we created service gateways and IoT resources that can be connected via WiFi 802.11b. The simulation is performed using a virtual desktop computer running Ubuntu 14.05 (64-bit) with an Intel Core i7-3770 chip (3.40 GHz, dual-core) with 4.00 GB of RAM.

6.2 Service Identification Time and Message Traffic

Figures 7(a) and 7(b) show the average time required to identify available services by the service registries. In Figure 7(a), the service identification time of the baseline approach increases as the number of IoT resources increases. When the number of IoT resources is increased to 90, the time spent by the baseline approach (BXML) reaches 152 s, whereas our approach involving the use of HBFs requires much less time (194 ms at maximum) to discover services. For our approach, there is an increase in the time (156 ms)



(a) Service identification time as the number of IoT resources changes



(b) Service identification time as the number of service changes

Figure 7 Service identification time and message traffic.

[5]http://www.nsnam.org

as the number of IoT resources changes from 10 to 100. However, the increase in the time is about 868 times smaller than that of the baseline approach (135528 ms).

As shown in Figure 7(b), the service identification time of the baseline approach also increases as the number of defined services increases. The result of the baseline approach increases by 138 s, while the result for the HBF approach is still less than 197 ms. Although the service identification time of the HBF approach also increases as the number of services increases, the time required by the baseline approach is always larger than Bloom-filter-based approach, and the time difference increases to 138251 ms.

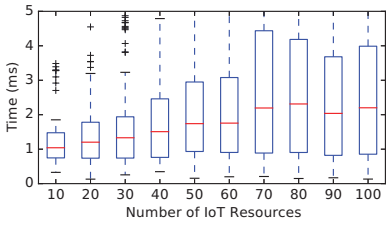
The service identification time shown in Figures 7(a) and 7(b) is the average time required to identify services and tasks in a registry. These simulation results demonstrate that our approach of using HBFs is effective for reducing the service discovery time, especially in dynamic IoT environments.

When an environmental change occurs (i.e., an IoT resource becomes unreachable or a new IoT resource becomes available), the service registry that detects the change shares the updated information of the IoT resources and services with other service registries by exchanging messages. We measure the latency for updating the information in the service registries and synchronizing them with consistent information about IoT resources and services.

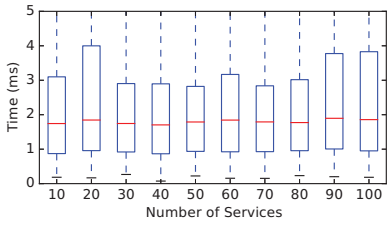
Figure 8(a) shows the changes in the latency time as the number of IoT resources increases. As the graph shows, the median latency when updating service registries is steady at around 2 ms, regardless of the increase in the number of IoT resources. However, the variation in the latency time fluctuates. This is caused by the random assignment of IoT-resource capabilities to resources and by the random movement of the IoT devices during the experiment. If many changes occur at the same time in an IoT environment, the latency for updating all service registries in the environment increases dramatically. However, the maximum variation in the latency time remains less than 5 ms. Because the latency time includes the service identification time, the increase in the service identification time may be the main cause of the time increase.

Figure 8(b) shows the latency when updating the service registries. As the figure shows, the median latency when updating the service registries is also mostly steady at around 2 ms, regardless of the number of services.

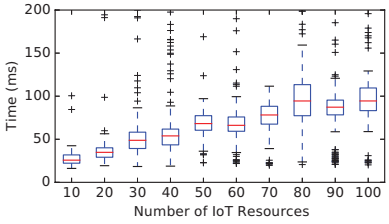
For the baseline approach, as shown in Figure 8(c), an increasing trend for the latency time as the number of IoT resources increases is observed. The median latency when updating the registries also increases almost linearly



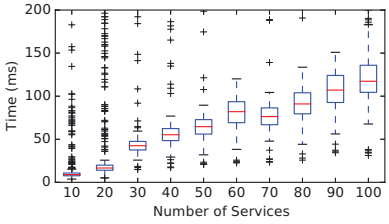
(a) Latency time according to the number of IoT resources (HBF)



(b) Latency time according to the number of services (HBF)



(c) Latency time according to the number of IoT resources (BXML)

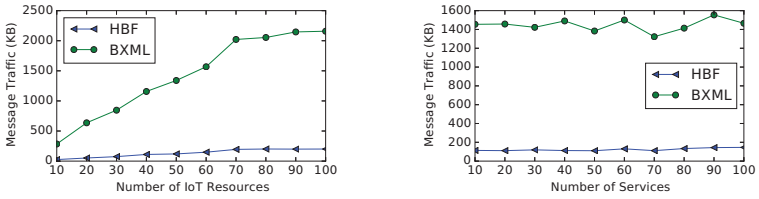


(d) Latency time according to the number of services (BXML)

Figure 8 Latency when updating service registries.

as the number of services increases, as can be seen in Figure 8(d). In other words, the baseline approach of exchanging BXML messages requires much more time to synchronize distributed service registries when there are changes in an IoT environment. In comparison to the HBF approach, the baseline approach requires more time by nearly 63-fold when there are 100 services in an IoT environment. Changes in the statuses of IoT resources cause service registries to spend more time updating their information because the registries need to identify services that can be provided by utilizing alternative or new IoT resources.

The results of this experiment indicate that in a highly dynamic IoT environment, the baseline approach is not suitable because environmental changes cannot be handled rapidly enough to synchronize the registries to utilize the IoT resources to perform user tasks, as the latency increases. On the other hand, the HBF approach can deal with environmental changes almost immediately, even when there are large numbers of services and IoT resources to be managed by the registries.



(a) Message traffic according to the number of IoT resources

(b) Message traffic according to the number of services

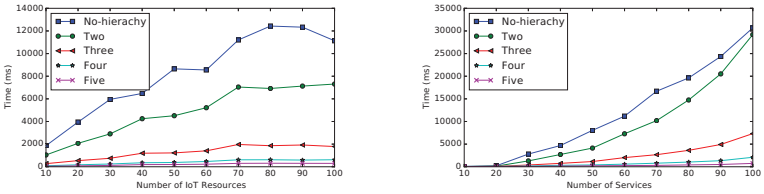
Figure 9 Message traffic generated by registries to share the information of IoT resources and services.

Figure 9(a) shows the amount of network traffic generated during message exchanges between service registries for synchronizing their information according to the changes in IoT-resource statuses. The baseline approach generates around 284–2158 KB of message traffic, whereas the HBF approach generates approximately 25–200 KB of message traffic. In addition, Figure 9(b) shows the message traffic according to the number of services. Both approaches show stable message traffic regardless of the number of services. However, the difference between the message traffic generated by the baseline approach and that generated by the HBF approach is around 1411 KB (when the number of services is 90). This demonstrates that our approach is more efficient than the baseline approach in terms of reducing the message traffic among distributed service registries.

6.3 Effectiveness of Having Hierarchies of IoT Resources and Services

To check the effectiveness of adding hierarchies to Bloom filters based on the capabilities of IoT resources and services, we conducted experiments while setting the depth of the hierarchies at different levels. In these experiments, we measured the size of the search space created by the capability hierarchies and the amount of message traffic generated among service registries while changing the numbers of IoT resources and services.

As explained previously, if an IoT-resource capability required by a service becomes unavailable, the registries search higher levels of the capability hierarchy and find alternative IoT resources that have similar resource capabilities. Figure 10(a) shows the difference in the service identification times of the nonhierarchical and hierarchical Bloom-filter approaches. All



(a) Service identification time according to the number of resources

(b) Service identification time according to the number of services

Figure 10 Service identification times when using hierarchical Bloom-filter approaches.

graphs show a similar trend, but the maximum time difference between the nonhierarchical approach and hierarchical approach is around 12 s when the number of IoT resources reaches 80. Figure 10(a) also shows the service identification time required by the HBF approaches. When the hierarchy level increases to three, the identification time decreases. However, with a four-level HBF approach, the decrease in the identification time becomes small.

In the graph shown in Figure 10(b), as the number of services increases, the service identification time increases more rapidly when using no hierarchy or a one-level hierarchy of capabilities than using hierarchies of two levels or greater. The time difference between no hierarchy and the five-level hierarchy reaches around 30 s when the number of services is 100. As shown in the figure, having more than two levels in the hierarchies is more effective than having only two levels. With 100 services, the HBF approach with three levels requires nearly 7.3 s—four times less than the approach with two levels. However, the addition of more than three hierarchical levels does not contribute to the same amount of improvement in the performance as the hierarchical level changes from no hierarchy to two or from two to three. The reason is that the overhead incurred when using more Bloom filters cancels out the performance improvement that is realized with more levels in the hierarchies. Therefore, we think that the use of three levels in the IoT resource and service hierarchies is the most effective for reducing the service identification time.

We also measure the changes in the message traffic among service registries as we increase the depth of the IoT resource and service hierarchies. As shown in Figure 11, the message traffic generally increases as the number of IoT resources increases. The rate of increase in the traffic becomes higher as the depth of the hierarchies increases. When there are 100 IoT resources

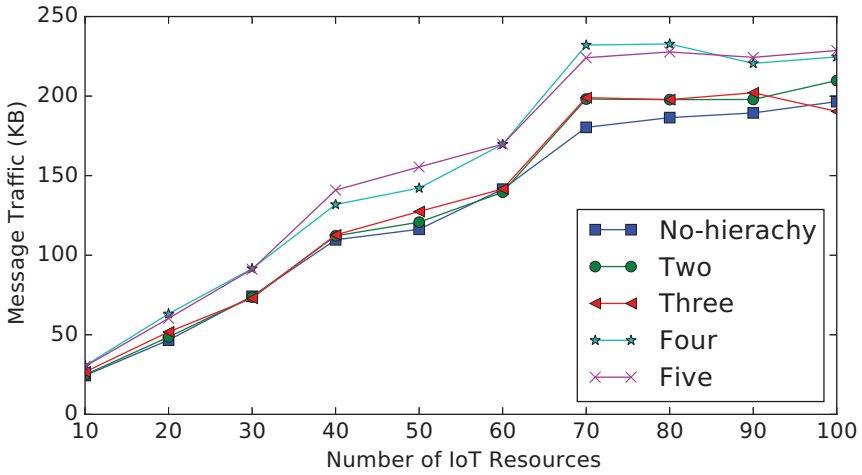


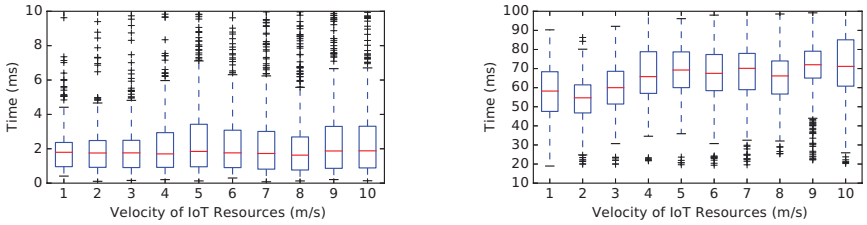
Figure 11 Message traffic generated by service registries for sharing the information of the IoT resources.

to manage, the HBF approach with a five-level hierarchy produces 40 KB more traffic on average than the non-hierarchical Bloom-filter approach. In Figure 11, the four-level and five-level Bloom-filter approaches generate the most network traffic of all cases. This reflects the overhead of using a longer integer vector to keep and exchange multiple hash values when there are multiple levels in the capability hierarchies. Therefore, it is necessary to select an appropriate depth for the capability hierarchies.

6.4 Sensitivity to the Velocity of IoT Devices

We also conducted an experiment to observe the effect of increasing the velocity of IoT devices in a MANET environment. In a mobile IoT environment, keeping track of the locations of the IoT devices is essential for providing services in a stable manner. We measure the latency associated with updating the distributed service registries while changing the velocity of the IoT devices from 1 to 10 m/s in a 500 m $\times$ 500 m area.

As shown in Figure 12(a), the median latency when updating service registries with the HBF approach is quite steady (around 1.8 ms). The median latency when the velocity is 10 m/s is slightly higher than that when the velocity is 1 m/s. However, the difference is not significant (less than 0.1 ms). This result demonstrates that our approach is sufficiently efficient to deal with mobile IoT devices that move at walking or running speeds. As shown in



(a) Latency incurred when updating service registries by using the hierarchical Bloom-filter approach

(b) Latency incurred when updating service registries by using the baseline approach

Figure 12 Latency incurred when updating service registries when there are IoT devices operating at different velocities.

Figure 12(b), the baseline approach generates about 37 times greater latency on average than the proposed approach when updating service registries. The latency time generally increases as the velocity of the IoT devices increases.

These experimental results demonstrate that the HBF approach is effective and efficient when used to deal with dynamic changes in mobile IoT environments, as it can successfully manage distributed service registries.

6.5 Threats to the Validity

6.5.1 Threats to the external validity

The first threat to the external validity of our experiment is that we regard the operations defined in the WSDL description of a service as the IoT-resource capabilities that are necessary for the service. In a real IoT environment, an operation of a service may not be the same as an IoT-resource capability. However, because service operations need to be tightly associated with the functionality of IoT resources, we believe that this assumption does not invalidate the results of our experiments.

The second threat to the external validity is that the hierarchies of services and IoT-resource capabilities are generated randomly for our experiments. Different IoT environments usually have different hierarchical structures of services and IoT-resource capabilities. Our approach, however, can be applied to any type of capability hierarchy. According to the characteristics (depth and shape) of the hierarchies, our approach reduces the search space to different degrees. As we showed earlier in Section 6.3, having more levels in a capability hierarchy results in a reduction in the search space but incurs more overhead for maintaining the Bloom filters for the levels.

6.5.2 Threats to the internal validity

A threat to the internal validity is that we simulated a mobile IoT environment in our experiments. We use ns-3, which is widely used for simulating real-world network environments. Therefore, the distributed service registries implemented in the ns-3 simulator reflect the characteristics of practical MANET environments.

7 Conclusion and Future Work

A mobile IoT environment is a service environment in which diverse mobile IoT resources that have various capabilities can be utilized to provide users with services that are necessary to accomplish their tasks. In this environment, it is essential to manage and find resource capabilities and services efficiently, even when the availability and connection statuses dynamically change. In this research, we proposed an efficient approach for discovering distributed IoT resources and identifying available services by utilizing the discovered IoT resources in mobile IoT environments.

We applied Bloom filters to the configuration and management of distributed service registries to reduce the configuration cost and to reduce the number of message exchanges between registries when updating the information about available IoT resources in dynamic mobile IoT environments. Specifically, we extended the use of traditional Bloom filters and gave them a hierarchical structure for more efficient discovery and management of the information about the capabilities of IoT resources and the services that utilize them. We also developed an approach for serializing the encoding space of the hash values generated by the Bloom filters and for improving the efficiency of representing and searching the information of the IoT-resource and service capabilities.

We conducted a simulation for discovering services in a highly dynamic mobile IoT environment in which IoT resources are continuously moving in the environment at various velocities. Our results demonstrated that the proposed approach is much more efficient than the baseline approach (binary XML) used in the comparison. The effect of the HBFs was also demonstrated with various structures of the resource capability hierarchies. We attempted to solve the network overhead issue as well, which backbone network approaches do not target, while also attempting to reduce the configuration cost, which is also not targeted by cluster-based approaches. Finally, we addressed the mobility issue, whereas other Bloom-filter approaches do not take this into account. In addition, we conducted experiments using the ns-3

simulator without any environmental assumptions, unlike the DHT-based approaches.

Our contribution is the development of hierarchical Bloom filtering and search-space serialization approaches that enable a significant reduction in the search space for discovering IoT-resource capabilities or services in environments where IoT resources are continuously moving. In addition, we applied the HBF approach to create distributed service registries that efficiently collaborate with each other to find resource capabilities and services that are distributed in a mobile IoT environment. Then, we extended our approach to handle more dynamic situations, such as the additions and/or removals of resource capabilities and services, with service registries joining and/or leaving to and/or from an IoT environment. In addition, we made the HBFs more extensible and flexible to handle large numbers of IoT resources and services and to reflect the dynamic changes in the structures of the resource capability and service hierarchies.

In our future work, we plan to create vectors of services and resource capabilities that contain additional information, such as the quality of service (QoS), so that users can select a service or resource capability by considering more specific quality requirements and/or constraints when there are the same or similar services and resource capabilities from which to choose. We will also extend our approach to deploy dynamic service instances following predefined policies and track them using distributed service registries. In addition, we will test our approach in an actual test-bed environment that we are currently building in our campus laboratory.

Acknowledgements

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT (NRF-2017M3C4A7066210).

References

[1] Christian Bettstetter and Christoph Renner. A comparison of service discovery protocols and implementation of the service location protocol. In *Proceedings of the 6th EUNICE Open European Summer School: Innovative Internet Applications*, 2000.

[2] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[3] Safdar Hussain Bouk, Iwao Sasase, Syed Hassan Ahmed, and Nadeem Javaid. Gateway discovery algorithm based on multiple qos path parameters between mobile node and gateway node. *Journal of Communications and Networks*, 14(4):434–442, 2012.

[4] Gregor Broll, Enrico Rukzio, Massimo Paolucci, Matthias Wagner, Albrecht Schmidt, and Heinrich Hussmann. Perci: Pervasive service interaction with the internet of things. *IEEE Internet Computing*, 13(6):74–81, 2009.

[5] Shuxing Cheng, Carl K. Chang, and Liang-Jie Zhang. An efficient service discovery algorithm for counting bloom filter-based service registry. In *Proceedings of the 2009 IEEE International Conference on Web Services (ICWS)*, pages 157–164. IEEE, 2009.

[6] Marco Conti, Silvia Giordano, Martin May, and Andrea Passarella. From opportunistic networks to opportunistic computing. *IEEE Communications Magazine*, 48(9):126–139, 2010.

[7] Suparna De, Payam Barnaghi, Martin Bauer, and Stefan Meissner. Service modelling for the internet of things. In *Proceedings of the Computer Science and Information Systems (FedCSIS)*, pages 949–955. IEEE, 2011.

[8] Sarang Dharmapurikar, Praveen Krishnamurthy, Todd Sproull, and John Lockwood. Deep packet inspection using parallel bloom filters. In *Proceedings of the 11th symposium on High performance interconnects*, pages 44–51. IEEE, 2003.

[9] Sarang Dharmapurikar, Praveen Krishnamurthy, and David E. Taylor. Longest prefix matching using bloom filters. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 201–212, New York, NY, USA, 2003. ACM.

[10] Mohammadmajid Hormati, Fatna Belqasmi, Roch Glitho, and Ferhat Khendek. A DNS protocol-based service discovery architecture for disaster response systems. In *Proceedings of the 2013 IEEE Symposium on Computers and Communications (ISCC)*, pages 000366–000371. IEEE, 2013.

[11] Hyeon-Jun Jo, Jung-Hyun Kwon, and In-Young Ko. Distributed service discovery in mobile IoT environments using hierarchical bloom filters. In *Proceedings of the International Conference on Web Engineering*, pages 498–514. Springer, 2015.

[12] Ronny Klauck and Michael Kirsche. Bonjour contiki: A case study of a DNS-based discovery service for the internet of things. In *Proceedings of the International Conference on Ad-Hoc Networks and Wireless*, pages 316–329. Springer, 2012.

[13] Michael Klein, Birgitta Konig-Ries, and Philipp Obreiter. Service rings-a semantic overlay for service discovery in ad hoc networks. In *Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, pages 180–185. IEEE, 2003.

[14] Ulas C. Kozat and Leandros Tassiulas. Network layer support for service discovery in mobile ad hoc networks. In *Proceedings of the Twenty-Second Annual Joint Conference of the IEEE Computer and Communications (INFOCOM)*, volume 3, pages 1965–1975. IEEE, 2003.

[15] Ulaş C. Kozat and Leandros Tassiulas. Service discovery in mobile ad hoc networks: an overall perspective on architectural choices and network layer support issues. *Ad Hoc Networks*, 2(1):23–44, 2004.

[16] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[17] Mikko Pitkänen, Teemu Kärkkäinen, and Jörg Ott. Mobility and service discovery in opportunistic networks. In *Proceedings of the 2012 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 204–210. IEEE, 2012.

[18] Francoise Sailhan and Valerie Issarny. Scalable service discovery for manet. In *Proceedings of the third IEEE International Conference on Pervasive Computing and Communications*, pages 235–244. IEEE, 2005.

[19] Gregor Schiele, Christian Becker, and Kurt Rothermel. Energy-efficient cluster-based service discovery for ubiquitous computing. In *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, page 14. ACM, 2004.

[20] Siva Sivavakeesar, Oscar F. Gonzalez, and George Pavlou. Service discovery strategies in ubiquitous communication environments. *IEEE Communications Magazine*, 44(9):106–113, 2006.

[21] Jerry Tyan and Qusay H. Mahmoud. A comprehensive service discovery solution for mobile ad hoc networks. *Mobile Networks and Applications*, 10(4):423–434, 2005.

[22] Rohit Verma and Abhishek Srivastava. A novel web service directory framework for mobile environments. In *Proceedings of the 2014 IEEE*

International Conference on Web Services (ICWS), pages 614–621, IEEE, 2014.

[23] Yonghang Yan, Linlin Ci, Zhiming Wang, and Wei He. Qos-based gateway selection in manet with internet connectivity. In *Proceedings of the 2013 15th International Conference on Advanced Communication Technology (ICACT)*, pages 195–199, IEEE, 2013.

[24] Jia Zhang, Runyu Shi, Weiyi Wang, Shenggu Lu, Yuanchen Bai, Qihao Bao, Tsengdar J. Lee, and Kiran Nagaraja. A bloom filter-powered technique supporting scalable semantic service discovery in service networks. In *Proceedings of the 2016 IEEE International Conference on Web Services (ICWS)*, pages 81–90, IEEE, 2016.

[25] Yilei Zhang, Zibin Zheng, and Michael R. Lyu. Wsexpress: A qos-aware search engine for web services. In *Proceedings of the 2010 IEEE International Conference on Web Services (ICWS)*, pages 91–98. IEEE, 2010.

[26] Fen Zhu, Matt W. Mutka, and Lionel M. Ni. Service discovery in pervasive computing environments. *IEEE Pervasive computing*, 4(4):81–90, 2005.

Biographies

Hyeon-Jun Jo received his M.S. in computer science from Korea Advanced Institute of Science and Technology, KAIST. His research interests is distributed service discovery in IoT environments.

Jung-Hyun Kwon is a Ph.D. student in school of engineering at the Korea Advanced Institute of Science and Technology, KAIST. His research interests include web applications, software testing. He received his M.S in the Division of Web Science and Technology in 2014 from KAIST, Republic of Korea.

MinHyeop Kim is a Ph.D. student in School of Computing at the Korea Advanced Institute of Science and Technology, KAIST. His research interests include service composition and resource allocation in IoT environment. He received his M.S in computer science in 2015 from KAIST, Republic of Korea.

In-Young Ko is an associate professor in School of Computing at the Korea Advanced Institute of Science and Technology, KAIST. His research interests include Web engineering, self-adaptive systems, and service computing. He received his Ph.D. in computer science in 2003 from University of Southern California, USA.