Distributed Service Discovery in Mobile IoT Environments Using Hierarchical Bloom Filters

Hyeon-Jun Jo(✉), Jung-Hyun Kwon, and In-Young Ko

School of Computing, Korea Advanced Institute of Science and Technology,
291 Daehak-ro, Yuseong-gu, Daejeon 305-701, South Korea
{hyeonjun.jo,junghyun.kwon,iko}@kaist.ac.kr

Abstract. The Internet of Things (IoT) enables many devices to interact with each other in order to provide services to users. Especially, mobiles devices are becoming more powerful and common in everyday life, and it is essential to utilize their capabilities to provide services in IoT environments. In this paper, we propose a fast and flexible approach to discovering IoT-based services 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 in updating information about available services. Especially, we extended the traditional Bloom filter to give it a hierarchical structure to more efficiently discover and manage information about the capabilities of IoT resources and the services that utilize them. We showed that performance increases and less message traffics in discovering available services by adding hierarchies to Bloom filters.

Keywords: Service discovery · Distributed service registries · Internet of things · Bloom filter

1 Introduction

In Internet of Things (IoT) environments, many real-world objects, from small sensors to vehicles, become smart objects that can be utilized to provide users with information and service capabilities [1] [2]. In addition, mobile devices such as smart phones, 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 call the IoT environment, in which there are IoT devices that have mobility, a *mobile IoT environment*. Mobile Ad-hoc Network (MANET) [4] is being regarded as a promising technology to make connections among mobile IoT devices to make them interact with each other for user tasks. For example, the Thread group [7] is a mesh network, a type of MANET, which connects various household devices in a flexible and scalable manner.

In this mobile IoT environment, because of the mobility characteristic of IoT devices, connectivity among the devices is highly dynamic and spontaneous. Therefore, it is a challenging issue to discover and provide services that utilize the capabilities of mobile IoT devices in a reliable manner. In addition, since the status of services that

P. Cimiano et al. (Eds.): ICWE 2015, LNCS 9114, pp. 498–514, 2015.
DOI: 10.1007/978-3-319-19890-3_32

can be offered utilizing mobile IoT devices is volatile [3], it is necessary to effectively monitor and manage the availability and status of IoT-based services. The characteristics of the mobile IoT environment can be summarized as mobility, temporal connectivity, diversity, and large number of devices [4].

An IoT resource can be used by one or more services. In addition, services that utilize IoT resources work together to perform user tasks. A user task is a composition of services that provides capabilities that are necessary to accomplish a user's goal. Therefore, the challenging issue is to find appropriate services for a task out of the available services in the mobile IoT environment. Unlike service discovery in the traditional Web environment, in an IoT environment it is essential to effectively deal with dynamic changes in service availability and connectivity among resources. Therefore, s*ervice discovery* in mobile IoT environments can be defined as follows: recognizing, configuring, and communicating of IoT resources and the services that utilize them [5]; finding available services in a dynamic manner; and managing status information [6].

In this paper, to meet these needs, we propose distributed service registries that enable fast and flexible service discovery in highly dynamic and spontaneous IoT-based service environments. A *service registry* stores information about the available services in an environment and provides clients with functions to find necessary services and to check the status of the services. To successfully build distributed service registries, it is essential to minimize the overhead of message exchanges between registries in order to find available services based on the IoT resources in a certain region. Especially, it is crucial to reduce the time overhead spent identifying resource capabilities of the available IoT devices, the services that can be provided by utilizing resource capabilities, and the tasks that can be performed by coordinating the available services.

In this work, we applied Bloom filters to the configuration and management of distributed service registries to reduce the configuration cost, and to reduce message exchanges in updating information about available services in dynamic mobile IoT environments. A *Bloom filter* converts a set of elements into an array by mapping each element of the set to a combination of index numbers. Each combination of index numbers is unique, and can later be used to identify an element of the set. For our distributed registry system, we convert each resource capability and service into a combination of integer numbers. Then, we project the combinations into coordinate points on a multi-dimensional coordinate system. Then, a registry can find necessary resource capabilities for a service on the coordinate system using the vector of the resource capabilities. On the coordinate system, each point means that the corresponding resource capability is available in the environment. In this way, availability of a service in a registry can be efficiently checked. Then, by sharing the converted points for available services with other neighboring registries, it is possible to maintain a global view of available services in the IoT environment.

Due to the false-positive problem of Bloom filters, we must control the range of unique index numbers and the number of hash functions to reduce the rate of generation of false-positive errors. The false-positive error rate depends on the total number of elements used. In other words, the higher the number of elements we use, the higher the number of hash functions will be needed. This causes a scalability problem. Therefore, we developed a *hierarchical Bloom filter* for the distributed service registries. We use multiple Bloom filters, each of which is used for a different level of the resource and the service hierarchies. The *resource-capability hierarchy* defines

various types of resource capabilities in a hierarchical manner; *service hierarchy* defines a hierarchy of service capabilities. The two hierarchies are needed to help clients request services at various abstraction levels. With experiments using ns-3 simulator, we show that performance increases and less message traffics in discovering available services in an environment.

The remainder of the paper is organized as follows. In Section 2, we explain 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 of using Bloom filters to build an efficient distributed service registry system. Section 5 presents our experimental results and analyses. Finally, Section 6 concludes the paper with a brief discussion on our contribution and future work.

2 Related Work

Distributed service registries have been studied to overcome disadvantages such as bottleneck and single point of failure problems of centralized registry systems. First, a backbone network based approach was proposed [12] [13]. In this approach, a backbone network is formed using several nodes in an ad-hoc manner, and service registries are installed in the network nodes. This approach makes it possible to discover and manage services in distributed network nodes, but it may produce a lot of network traffic because a query to discover services is broadcasted to all registries in the network. Moreover, whenever there are any changes in the network nodes, the backbone network needs to be reformed to reflect the changes.

Service clustering methods have been developed to overcome the disadvantages of backbone network based approaches [14] [15]. The service clustering methods classify services in a network in different groups based on their types (e.g., node location) and choose a representative network resource to manage services in each group. This approach can reduce network traffic by sending a service-discovery query only to a specific cluster. However, it has the additional overhead of choosing a new representative for a cluster when the network changes. Although it is possible to use a static clustering method, one that does not consider network changes, such a method can still be problematic when the representative of a cluster is broken or not reachable.

There have also been studies on using distributed hash tables (DHT) for service discovery [16] [17]. In such an approach, each service has its unique key and the key and the location information of the service are stored in a hash table. Multiple registries distributed in a network environment are used to manage the table. The location information refers to a point in the entire area formed by all the service registries. Existing research assumes that each registry knows the correct location, which is hard to know because each registry knows location information for its local area, not for the global area.

There has been research on using a Bloom filter for service discovery. Sailhan et al. applied a Bloom filter to service description and reduced the size of service information stored in a registry [18]. This study, however, did not apply a Bloom filter to service-discovery queries exchanged among the service registries. Cheng et al. proposed a Counter Bloom filter to overcome the limitation of the traditional Bloom filter, in which

stored hash values cannot be deleted after they are generated [19]. Using the Counter Bloom filter, 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 caused by the registries exchanging service information.

The aforementioned studies focused on fast information search within a service registry using a Bloom filter. Our study seeks not only to search service information fast inside a registry but also to reduce network traffic exchanged among registries by using a hierarchical Bloom filter, which is explained in Section 4.

3 Mobile IoT-Based Service Environment

The target environment that we consider in this work is an IoT-based service environment, in which various IoT resources and service gateways are deployed in a geographical region. The IoT resources and service gateways have mobility. The IoT resources are various types of IoT devices from tiny sensors to security cameras on streets, smartphones, and other IoT devices. Each IoT resource provides *resource capabilities*, which can be utilized to provide services. For the IoT resources that do not have enough computation and networking power to provide services, there are service gateways that proactively find IoT resources in the surrounding environment and make them accessible 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 IoT resources that are bound to the service. A *user task* is a composition of abstract services that define 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 based on the quality requirements of a user.

Fig. 1 illustrates a mobile IoT-based service environment, including these core components. The *task layer* in Fig.1 is for selecting and running a user task. The *IoT resource layer* is composed of service gateways, service registries, and IoT resources. A *service gateway* discovers nearby IoT resources, and monitors changes in the surrounding environment to update the status information about the available IoT resources and service instances that utilize the IoT resources. The gateway has a connection range in which it can discover IoT resources. Gateways can be connected to each other and exchange information about available IoT resources and services that can be provided in a local environment. Smartphones and portable computing devices that have computing and networking capabilities can be used as gateways.

An IoT resource can be moved into an environment and can be dynamically discovered by nearby service gateways. An IoT resource can be utilized by one or more services. A service gateway usually has a limited *connection capacity* of maintaining connections to IoT resources. In addition, the number of gateways to which an IoT resource is connected and by which it can be monitored can be controlled by overall policy.

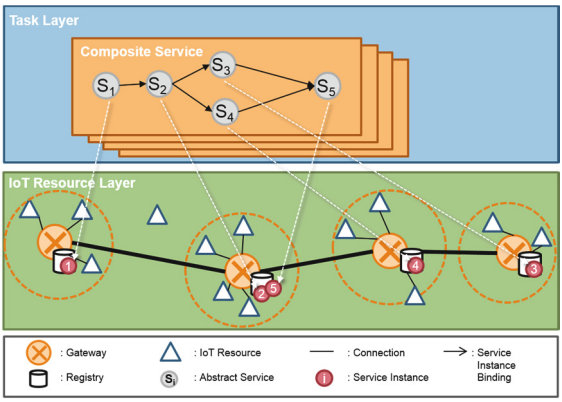


Fig. 1. Mobile IoT-based Service Environment

A service registry is deployed to a service gateway. 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 in a local environment, and maintains information about available services. Service registries also collaborate with each other by exchanging messages to monitor and share the status information about available services in the local environment. Therefore, it is possible to obtain information about available services by querying any of the service registries in the local environment.

4 A Hierarchical Bloom Filter for Distributed Service Registries

4.1 Hierarchies of Resource Capabilities and Services

We designed an ontology model to represent the relationship between the conceptual entities that are related to the resources and services in IoT-based service environments. Fig. 2 shows the model, which is composed of task, service, resource, and capabilities. The tasks, services, and resources provide their own capabilities, and the capabilities are compared against each other by the service discovery engine. A task is a composition of services and requires some service capabilities to provide its capabilities; a service requires certain resource capabilities to deliver its service capability.

To run a service instance, that instance needs to be bound to necessary resource capabilities. In our registry system, the required resource capabilities can be specified at different abstraction levels. The IoT resources that provide capabilities at a certain abstraction level that a service specifies, or more specialized capabilities than those of the abstraction level, are all considered as candidate IoT resources for the service. To support this, the resource capabilities are defined hierarchically by representing the subsumption relations (by using the 'subClassOf' relation) between the capabilities. Services are also defined similarly by representing the subsumption relations 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 to find service instances for an abstract service can be made according to the abstraction levels specified.

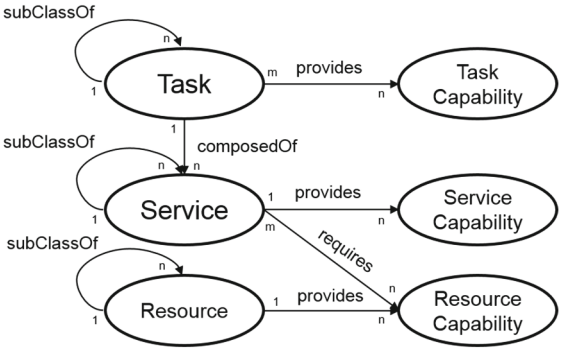


Fig. 2. Ontology Model for Mobile IoT Service Environment

By using the hierarchies of resource capabilities and services, it is also possible to identify services that can be run based on the available resource capabilities in a local environment. In addition, after recognizing the available services, user tasks that can be performed in the environment can be recommended to users.

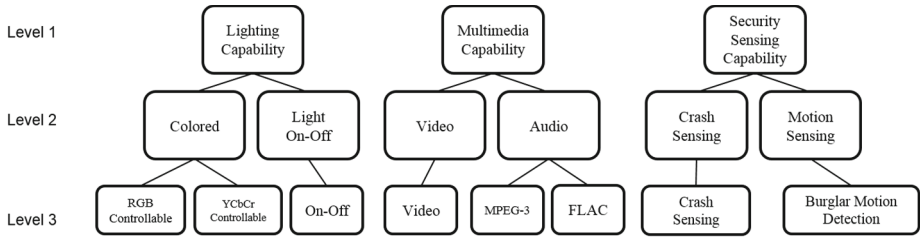


Fig. 3. Resource Capability Hierarchy Example

Fig. 3 shows a partial view of the resource capability hierarchy. The 'Lighting', 'Multimedia', and 'Security Sensing' capabilities are defined as the top-level capabilities, which can be divided into more specialized capabilities. For example, there are the 'Colored' lighting and simple 'Light On-Off' capabilities under the 'Lighting' capability, and 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.

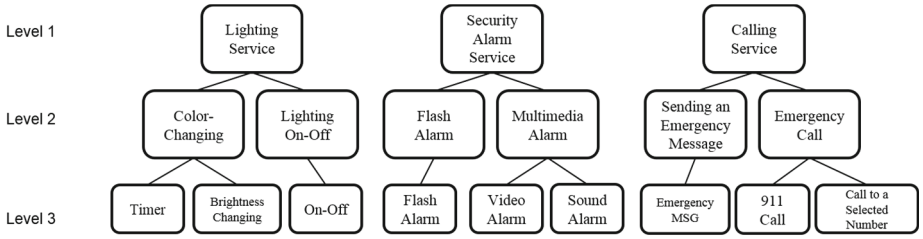


Fig. 4. Service Hierarchy Example

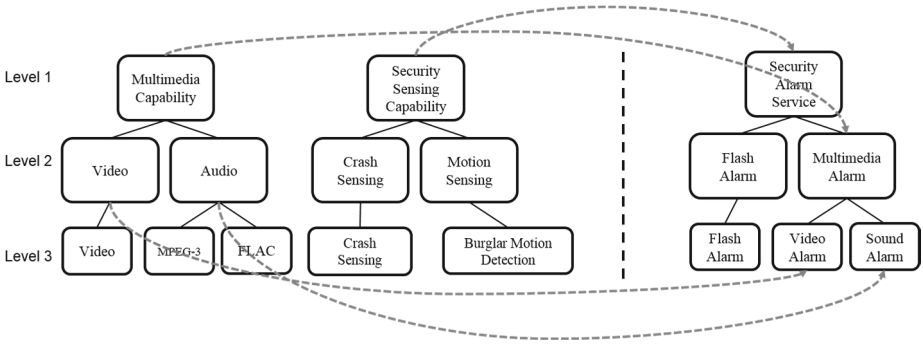


Fig. 5. An Example of Representing Relationship between Resource Capabilities and Services

Fig. 4 shows a part of the service capability hierarchy. The 'Lighting' service can be specialized into the 'Light On-Off' service, and the 'Color-changing' lighting service, which has more specialized services for changing the light colors. If a timer is needed, the 'Timer' service can be chosen. For the 'Security Alarm Service', there are more specialized services including the 'Flash Alarm' and 'Multimedia Alarm' services. If the required resource capabilities are not present, a service cannot be instantiated. For instance, as shown in Fig. 5, in order to instantiate any of the 'Security Alarm Service', both the 'Security Sensing' and 'Multimedia' or 'Lighting' resource capabilities are required.

A user task can be performed if all the abstract services are instantiated by being bound to required IoT resources. For example, the 'Secure a car' task requires two services, the 'Lighting' and 'Security Alarm' services, which can be instantiated by using the 'Lighting', 'Multimedia', and 'Security Sensing' capabilities.

4.2 Hierarchical Bloom Filters

A Bloom filter is a probabilistic data structure used to test if an element is a member of a set or not [8]. A Bloom filter is composed of a bit array and a set of hash functions. Once the bit array is made by performing the hash functions for a set of elements, it is possible to efficiently check if a specific element is in the set or not without searching all the elements. Some lookup mechanisms are necessary to implement a Bloom filter to avoid unnecessary search [9] [10]. The hash functions generate a series of integers for each element. These integers represent the positions (indices) in

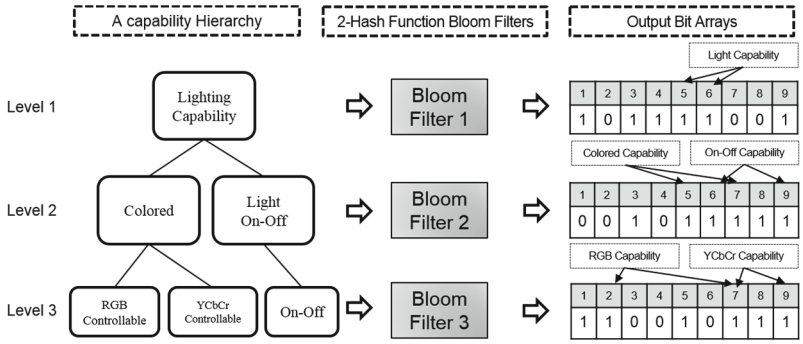


Fig. 6. Bloom Filters Applied to the Resource Capability Hierarchy

the bit array. The number of integers is determined based on 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. We can decide whether an element is in the set or not by checking specific positions of the bit array.

A *hierarchical Bloom filter* is a set of Bloom filters, each of which corresponds to one level of the resource capability hierarchy. Fig. 6 shows an example of the application of a hierarchical Bloom filter to the 'Lighting' capability hierarchy. In this example, since each Bloom filter has two hash functions, each capability is converted into two integers. In the figure, the 'Lighting' capability is represented by the integer numbers, 5 and 6, the 'Colored' lighting capability is converted to 5 and 7, and the 2 and 7 integer numbers are assigned to the 'RGB Controllable' lighting capability.

A Bloom filter has a 100% recall rate, which means that there are no false-negative errors but there may be false-positive errors. If there are false-positive errors, the resource capabilities and/or services that are not actually available may appear as if they are available during the service discovery process. Therefore, it is critical to manage the rate of false-positive errors so that it is very low. We can configure the rate of generating false-positive errors by resizing the number of hash functions and the length of the bit array. Equation (1) shows how to calculate the rate of false-positive errors [11]. In the 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) means that the Bloom filter makes a tradeoff between the length of the bit array and the rate of false-positive errors. An increase of the number of hash functions causes an increase of the dimensions of the coordinate system, and generates more computational overhead in the discovering of services. Therefore, we determine the number of hash functions first, and then set the acceptable rate of false-positive errors by adjusting the length of the bit array.

$$(1 - [1 - \frac{1}{m}]^{kn})^k \approx (1 - e^{-kn/m})^k \quad (1)$$

4.3 Discovery of Resource Capabilities

As Fig. 6 shows, the number of Bloom filters depends on the depth of the resource capability hierarchy. A different Bloom filter is used for each different level of the hierarchy. Because each level of the capability hierarchy may have a different number of capabilities, we determine the number of hash functions based on the maximum number of siblings in the hierarchy. All the Bloom filters have the same number of hash functions and keep the same dimension coordinate system for all levels.

For effective and efficient discovery of resource capabilities, both bottom-up and top-down discovery of resource capabilities and services need to be supported. *Bottom-up discovery* is used to find available services from the available resource capabilities found in a local environment. *Top-down discovery* is used to find necessary resource capabilities for instantiating an abstract service of a user task. There should be also a fast and simple method to reduce the overhead of exchanging messages to find available services based on the resource capabilities that exist in an environment. In addition, since the integer numbers that are generated by Bloom filters do not have any regularity but are rather almost random, we need a method to represent the hierarchical relations between levels of resource capabilities.

Algorithm 1 Convert Resource Capabilities to Vectors

Require: Capability C
Require: Capacity Hierarchy CH
Require: Number of Hash Functions in a Bloom Filter HF
1: Set $capabilityVector$ to HF dimensional vector
2: Set all elements in $capabilityVector$ to 0
3: Set Number of digits to shift per each hash function to $digitsToShift$
4: **while** C in CH **do**
5: $resourceDescription = getResourceDescription(C)$
6: $hashValues = applyBloomfilter(resourceDescription)$
7: **for** $axis = 0, axis < length(hashValues)$ **do**
8: $hashValues[axis] * = power(10, digitsToShift[axis])$
9: $capabilityVector[axis] + = hashValues[axis]$
10: $C = getParentNode(C)$

To meet these requirements, we have developed a method to map all the Bloom filter output values of resource capabilities in a coordinate space. Algorithm 1 shows the procedure of converting a resource capability into a coordinate point by using Equation (2). In the equation, L is the level of the resource capability hierarchy, BF_l is an integer number of a resource capability generated by a Bloom filter of level l, and D_l is the digit of the bit array of the Bloom filter of level l. Using this method, we can represent any resource capability as a numeric vector rather than as a string description. For instance, in Fig. 3, the 'Lighting' capability is represented as $\vec{V}_L(500, 600)$, the 'Light On-off' capability as $\vec{V}_B(560, 690)$, the 'Colored' lighting capability as $\vec{V}_C(550, 670)$, the 'RGB' lighting capability as $\vec{V}_R(552, 677)$, and the 'YCbCr' lighting capability as $\vec{V}_Y(557, 679)$.

$$\text{Resource Capability A}, \vec{V}_A = (\bar{p}_1, \bar{p}_2, \bar{p}_3, \dots, \bar{p}_k)$$

$$\text{Bloom Filter of Level } l, BF_l(A) = \left(i_{l,1}, i_{l,2}, i_{l,3}, \dots, i_{l,k}\right)$$

$$D_l = \begin{cases} l = L, D_l = 0 \\ l < L, D_l = \sum_{j=l+1}^{L} d_j \end{cases}, \; d_j = Digit\ of\ BF_j(A)$$

$$\text{For } \vec{V}_A, \bar{p}_k = \sum_{l=0}^{L} \{i_{l,k} \times 10^{D_l}\} \tag{2}$$

Each service registry shares the vector points of its own resource capabilities with other registries in a certain geographical region. The messages that are exchanged between registries are composed of the vector point information, and the location and name of the service registries. The vector points of the resource capabilities and services that are stored in a service registry can be projected to a coordinate system. Each point represents the location of a resource capability. Since a service may require multiple resource capabilities, we represent the location of a service as the median point of the required resource capabilities. By specifying a resource capability at a certain abstraction level, all the specialized resource capabilities can be included in the area in the coordinate system. Resource capabilities that are siblings belong to different areas in the coordinate system.

Fig. 7 shows the vector convergence of the resource capabilities shown in Fig. 3, and the mappings of the resource capabilities to a coordinate system. Different shadow areas indicate different types of capabilities at different abstraction levels. More specialized resource capabilities are shown in smaller areas in the coordinate system. In addition, in Fig. 7, similar or alternative resource capabilities are shown as close to each other. Therefore, alternative resource capabilities can be found efficiently by searching a small area rather than searching the entire coordinate space. However, an alternative resource capability found in the space may belong to a totally different abstraction level if there is a higher-level boundary to the alternative. Therefore, we need to find alternative resource capabilities at the highest abstraction level.

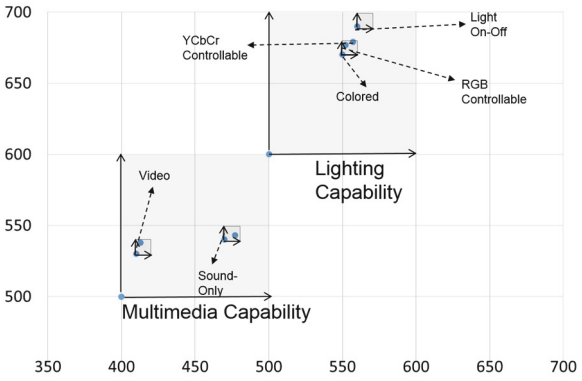


Fig. 7. Vector Conversion of Resource Capabilities

4.4 Discovery of Services

We have designed a *service table* that keeps the status information of the resource capabilities that are necessary to instantiate a service. This table enables both bottom-up and top-down discovery of service instances that are needed to perform a user task. Table 1 is an example of the service table; it includes the names of registered services, the number of resource capabilities required for each service, and the resource capabilities that are necessary for each service. Required resource capabilities of a service can be represented at various abstraction levels in the service table.

Algorithm 2 Discovery of Available Services

Require: Service Table ST
Require: Capability Hierarchy CH
1: $availableService = []$
2: **for** $service$ in ST **do**
3: $requiredResourceCapabilities$=$ST[service]$
4: $matchingResourceCapabilities$=[]
5: **for** rc in $requiredResourceCapabilities$ **do**
6: **if** rc in the terminals of CH **then**
7: //There are exact services providing rc
8: $matchingResourceCapabilities$.append($rc$)
9: **else if** rc in the non-terminals of CH **then**
10: //There are alternative services providing rc
11: $matchingResourceCapabilities$.append($rc$)
12: **if** $requiredResourceCapabilities == matchingResourceCapabilities$ **then**
13: $availableService$.append($service$)

Once the coordinate system for resource capabilities is generated, as Algorithm 2 shows, the availability of a resource capability can be checked efficiently by looking up the vector point of the resource capability in the coordinate system. After comparing service tables with identified resource capability points, available services can be efficiently identified. In addition, when available resource capability points are shared between registries via message exchanges, the location information of the registry in which the resource capability is managed is also stored in service registries.

To find a service instance that is necessary to perform a task in a top-down manner, a registry checks whether all the capabilities that are required by the service are available. While checking the required resource capabilities of the service, the registry considers the acceptable level of abstraction, and finds alternative resource capabilities when there is no resource capability that matches the required capability specified for that abstraction level.

Table 1. A Service Table

Service	No. of capabilities required	Resource Capability 1	Resource Capability 2	Resource Capability 3	…
A	1	(x_1, y_1)	-	-	…
B	3	(x_6, y_6)	(x_2, y_2)	(x_3, y_3)	…
C	2	(x_4, y_4)	(x_5, y_5)	-	…

Another set of Bloom filters is used to discover user tasks by identifying available service instances. The discovery process is similar to the one that we explained in the previous sections. The available services are shared as a point; then, the available tasks are identified from all the shared service points by using a *task table*, which plays a role similar to that of the service table. The difference between a service table and a task table is that a task table includes information about the sequence of composing services.

5 Experiment and Analysis

Experiments are conducted to show how the pointed problems are solved. First, we expect that a Bloom filter itself will have a space advantage in representing each resource capability and service. In addition, by using a Bloom filter we expect that messages between distributed service registries will be generated with shorter lengths. We suggest hierarchical Bloom filters to overcome the problem of computing overhead when identifying available services. The Bloom filters generate a unique sequence of numbers for each resource capability and shares the integer numbers. After sharing, each service registry is able to quickly identify available services by comparing the integer numbers.

Furthermore, hierarchical Bloom filters are suggested for the identification of available services in a more flexible manner. This process reduces the searching space when seeking resource capabilities for each service; the final computing time is reduced more than it is when using non-hierarchical Bloom filters.

5.1 Experiment Settings

In comparison to the traditional service discovery approach, which exchanges service descriptions encoded as Simple Object Access Protocol (SOAP) messages, our approach generates much less network traffic because the Bloom filters convert the full service descriptions into simple integers. In addition, comparisons between bit arrays take much less computing time than does the comparing of SOAP XML messages. Therefore, in this experiment, rather than comparing the Bloom filter based approach and the traditional SOAP-based approach, we focus on evaluating the effectiveness of the two different approaches to applying Bloom filters: the non-hierarchical and the hierarchical Bloom filter approaches.

The dataset that we use for the experiment is generated by constructing hierarchies of resource capabilities and services, and by classifying the instances of IoT resources and services under the hierarchies in a randomized manner. For service instances, we use the Web service dataset,[1] which is composed of practical Web services data. We do not parse all information from the Web service data set; however, we use them as input for the Bloom filters. We generate various hierarchies of resource capabilities and services. The various hierarchies have different maximum numbers of siblings under a parent. When creating the relationships between resources and services, we

[1] http://www.wsdream.net/dataset.html

assign a maximum of five resource capabilities to a service and a maximum of five service capabilities to a user task.

During the experiment, we check the effectiveness of the approaches by changing the number of registries, the number of capabilities per service registry, the total number of resource capabilities and services, the depth of the resource capability and service hierarchies, and the number of hash functions for each Bloom filter. We measure the computing time for a service registry to identify available services based on IoT resources that are available in a local environment. In addition, we measure the amount of message traffic generated while identifying the available services. In the experiment, we check the computing time and message traffics while changing the number of resource capabilities and services from 30 to 400. In addition, we randomly allocate up to 30 resource capabilities for each service registry, and make a service to be associated with maximum 5 resource capabilities in a randomized manner. In addition, we test our approach with changing the depth of the hierarchies from 2 to 4.

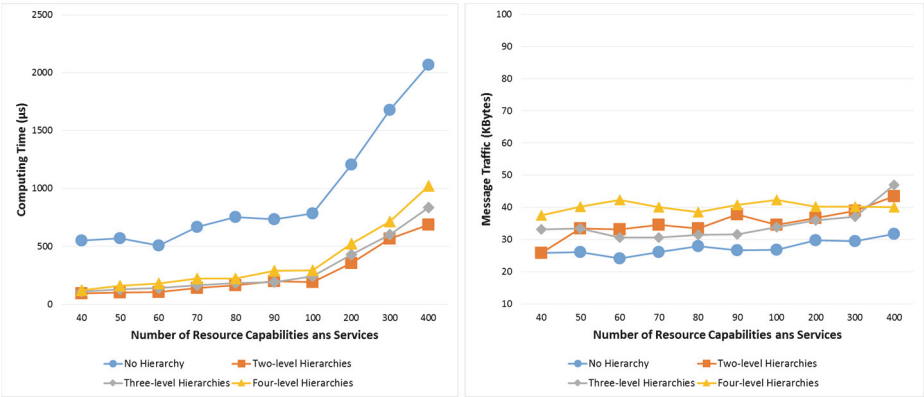
In this experiment, we simulate the mobile IoT environment by using the ns-3[2] simulator, which is commonly used in network simulations. In this simulation environment, we created service gateways, and IoT resources that can be connected via Wi-Fi 802.11b. The simulation is performed by using a desktop computer running Ubuntu 12.04 (32-bit) with Intel Core i7-2600 (3.40GHz) with 16.00GB of RAM.

5.2 Experiment Result and Analysis

Fig. 8 compares the computing time and message traffics between the non-hierarchical Bloom filter and the hierarchical Bloom filters with different depths (with 2 to 4 levels). As shown in Fig. 8 (a), all the cases of using the hierarchical Bloom filter take much less computing time than the case of using the non-hierarchical Bloom filter. In addition, the gap between the non-hierarchical and hierarchical Bloom filter approaches gets wider as the number of resource capabilities and services increases. The difference between the non-hierarchical Bloom filter approach and the hierarchical Bloom filter approach with four levels is about 490μs when there are 40 resource capabilities and services, and the difference reaches up to 1,020μs when there are 400 resource capabilities and services. We expect that there will be more performance improvement of discovering services when there are more number of resource capabilities and services that need to be managed in a local IoT environment by the distributed service registries.

The computing time of the hierarchical Bloom filter approaches increase slightly as the depth of the hierarchies increases. This time overhead is due to the increase of the time to access multiple integers that correspond to the levels in the hierarchies. The simulation is performed by using a high-performance desktop computer. In practical IoT environments, service gateways are usually located in mobile devices that have much less computing power than the desktop computer. Therefore, we expect that the performance gain that we obtained by using the hierarchical Bloom filters will be significant in real IoT-based service environments. In addition, the performance of individual service registries is critical to improve the overall performance of discovering services in a

[2] https://www.nsnam.org



(a) Comparison of Computing Time **(b) Comparison of Message Traffic**

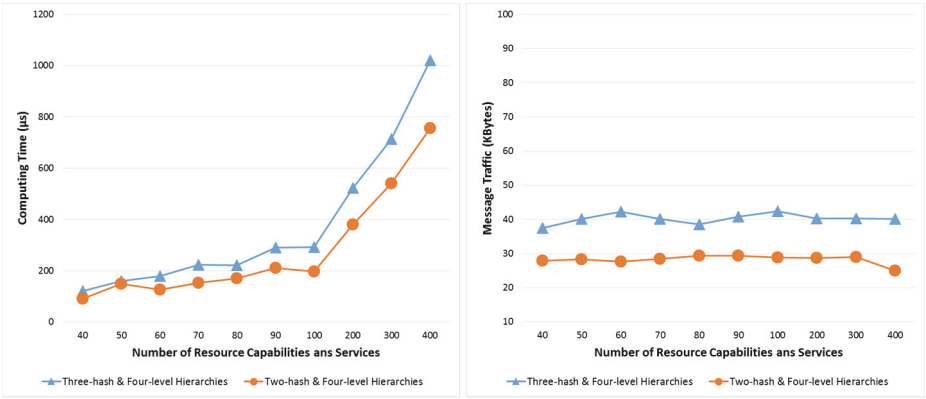
Fig. 8. Comparisons between Non-Hierarchical and Hierarchical Bloom Filters with Three Hash Functions

wide area where there are many resource capabilities and services to manage by a number of service registries.

As shown in Fig. 8 (b), the non-hierarchical Bloom filter generates the least amount of message traffic. Regarding to the hierarchical Bloom filter approaches, the message traffic increases slightly as the depth of the resource capability and service hierarchies increases. This is due to the fact that the hierarchical Bloom filters generate longer bit arrays as the depth of the hierarchies increases. However, the differences between the non-hierarchical and hierarchical Bloom filter approaches are not significant (maximum 18Kbyte difference) in mobile IoT environments where the network bandwidth is usually greater than 10Mbps (with Wi-Fi connections).

As the number of resource capabilities and services increase, it is necessary to increase either the number of hash functions or the depth of the resource capability and service hierarchies. In Fig. 8, we showed that having levels in the hierarchies causes a slight increase in computing time and message traffic. Fig. 9 compares the computing time and message traffics between the cases of using two hash functions and three hash functions for the Bloom filters with four levels in the hierarchies. As can be seen in the figure, having an additional hash function causes some overhead in computing time and message traffic. This is because it is necessary to manipulate and transmit more number of integers, as there is more number of hash functions.

In addition, the computing overhead caused by increasing the number of hash functions is relatively lager than the overhead generated by increasing the depth of the hierarchies. In other words, when there are 400 resource capabilities and services to manage, there is an increase of about 220µs in computing time as we add an additional hash function, whereas there is about 100µs increase in computing time when we increase the level of the hierarchies from 3 to 4. However, there is no significant difference in terms



(a) Comparison of Computing Time **(b) Comparison of Message Traffic**

Fig. 9. Comparisons between Different-Hash Four Level Bloom Filters

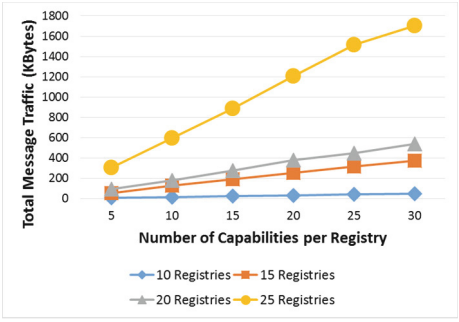


Fig. 10. Message Traffic Generated by Different Number of Registries

of the message traffic increased by having an additional hash function and adding a level in the hierarchies (about 15Kbyte increase vs. about 18Kbyte increase). Therefore, to handle more number of resource capabilities and services, it is more efficient to increase the depth of the hierarchies rather than adding more hash functions to the distributed service discovery system.

Fig. 10 compares the amount of message traffic generated by the service registries as we increase the number of service registries in an IoT environment. For each of the cases of having 10, 15, 20 and 25 service registries, we checked the trend of message traffic increase as we increase the number of resource capabilities that need to be managed by a service registry. All the cases show a linear increase in message traffic. However, as can be seen in the figure, the rate of message-traffic increase gets higher as there is more number of service registries. This is because there should be more interactions among the registries if their number increases in an environment. However, the amount of message-traffic increase is not significant when we increase the number of service registries up to 20 (maximum 500Kbyte increase when the number

of service registries increases from 10 to 20). This result implies that the proposed approach is scalable to the number of service registries. However, to maintain the amount of message traffic in an IoT environment at a certain level, it is necessary to limit the number of service registries in the local area.

6 Conclusion

A mobile IoT environment is the service environment where diverse mobile IoT resources produce various capabilities that can be utilized to provide users with services that are necessary to accomplish their tasks. In this environment, it is essential to efficiently manage and find resource capabilities and services, of which availability and connection status are dynamically changed. In this paper, we have proposed an efficient approach to discovering IoT-based services 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 in updating information about available services in dynamic mobile IoT environments. Especially, we extended the traditional Bloom filter to give it a hierarchical structure to discover more efficiently and manage information about the capabilities of IoT resources and the services that utilize them. We conducted a simulation of discovering services in a highly dynamic mobile IoT environment; our results proved that our approach is much more efficient than existing distributed service discovery approaches.

Our contribution is in that we have designed an ontology model for managing the information about user tasks, services and resource capabilities, and developed the hierarchical Bloom filtering approach that enables the significant reduction of the search space for discovering resource capabilities or services. In addition, we applied the hierarchical Bloom filtering approach to make distributed service registries to efficiently collaborate with each other to find resource capabilities and services that are spread in a mobile IoT environment.

As the future work, we firstly plan to extend the approach to handle more dynamic situations such as addition and/or removal of resource capabilities and services, and join and/or leave of service registries to and/or from an IoT environment. In addition, we will make the hierarchical Bloom filters more extensible and flexible to handle a large number of IoT resources and services, and to reflect the dynamic changes on the structures of the resource-capability and service hierarchies. We will also make the vectors of services and resource capabilities contain additional information such as quality of services 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 to choose from. We will also test our approach in a real test-bed environment that we are currently building in our campus.

Acknowledgement. This work was supported by the Dual Use Technology Program (UM13018RD1).

References

1. Broll, G., Rukzio, E., Paolucci, M., Wagner, M., Schmidt, A., Hussmann, H.: Perci: Pervasive service interaction with the internet of things. Internet Computing **13**(6), 74–81 (2009). IEEE
2. De, S., Barnaghi, P., Bauer, M., Meissner, S.: Service modelling for the Internet of Things. In: 2011 Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 949–955. IEEE (2011)
3. Verma, R., Abhishek, S.: A novel web service directory framework for mobile environments. In: 2014 IEEE International Conference on Web Services (ICWS), pp. 614–621. IEEE (2014)
4. Kozat, U.C., Leandros, T.: Network layer support for service discovery in mobile ad hoc networks. In: INFOCOM 2003, Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies, vol. 3, pp. 1965–1975. IEEE (2003)
5. Zhu, F., Mutka, M.W., Ni, L.M.: Service discovery in pervasive computing environments. IEEE Pervasive computing **4**(4), 81–90 (2005)
6. Bettstetter, C., Renner, C.: 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)
7. Thread Group. http://threadgroup.org
8. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM **13**(7), 422–426 (1970)
9. Dharmapurikar, S., Krishnamurthy, P., Sproull, T., Lockwood, J.: Deep packet inspection using parallel Bloom filters. In: IEEE Symposium on High Performance Interconnects (HotI), Stanford, CA, pp. 44–51 (2003)
10. Dharmapurikar, S., Krishnamurthy, P., Taylor, D.E.: Longest prefix matching using Bloom filters. In: Proceedings of the 2003 Conference on Applications, Technologies, Architectures, And Protocols For Computer Communications, ACM Sigcomm, pp. 201–212 (2003)
11. Mitzenmacher, M., Upfal, E.: Probability and computing: Randomized algorithms and probabilistic analysis, pp. 107–112. Cambridge University Press (2005)
12. Kozat, U.C., Tassiulas, L.: 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)
13. Sailhan, F., Issarny, V.: Scalable service discovery for MANET. In: Third IEEE International Conference Pervasive Computing and Communincations, pp. 8–12 (2005)
14. Klein, M., König-Ries, B., Obreiter, P.: service rings — a semantic overlay for service discovery in Ad Hoc networks. In: Network-Based Information Systems at Database and Expert Systems Applications, Prague, pp. 180–185 (2003)
15. Schiele, G., Becker, C., Rothermel, K.: Energy-efficient cluster-based service discovery for ubiquitous computing. In: 11th ACM SIGOPS European Workshop, Belgium (2004)
16. Sivavakeesar, S., Gonzalez, O.F., Pavlou, G.: Service discovery strategies in ubiquitous communication environments. Communications Magazine **44**(9), 106–113 (2006)
17. Tyan, J., Mahmoud, Q.H.: A comprehensive service discovery solution for mobile ad hoc networks. Mobile Networks and Applications **10**(4), 423–434 (2005)
18. Sailhan, F., Issarny, V.: Scalable service discovery for MANET. In: Third IEEE International Conference on Pervasive Computing and Communications, 2005, pp. 235–244 (2005)
19. Cheng, S., Chang, C.K., Zhang, L.J.: An efficient service discovery algorithm for counting bloom filter-based service registry. In: 2009 IEEE International Conference on Web Services, pp. 157–164 (2009)