electronics

Article

A Novel Approach towards Resource Auto-Registration and Discovery of Embedded Systems Based on DNS

Azimbek Khudoyberdiev, Wenquan Jin and DoHyeun Kim *

Department of Computer Engineering, Jeju National University, Jeju 63243, Korea; azimbekkhudoyberdiev@jejunu.ac.kr (A.K.); wenquan.jin@jejunu.ac.kr (W.J.)
* Correspondence: kimdh@jejunu.ac.kr; Tel.: +82-64-754-3658

Received: 23 March 2019; Accepted: 13 April 2019; Published: 17 April 2019

Abstract: The Internet of Things (IoT) is expected to deliver a whole range of new services to all parts of our society, and improve the way we work and live. The challenges within the Internet of Things are often related to interoperability, device resource constraints, a device to device connection and security. One of the essential elements of identification for each Internet of Things devices is the naming system and addresses. With this naming system, Internet of Things devices can be able to be discoverable by users. In this paper, we propose the IoT resource auto-registration and accessing indoor services based on Domain Name System (DNS) in the Open Connectivity Foundation (OCF) environment. We have used the Internet of Things Platform and DNS server for IoT Resource auto-registration and discovery in the Internet Protocol version 4 (IPv4). An existing system called Domain Name Auto-Registration in Internet Protocol version 6 can be used for Internet of Things devices for auto-registration and resource discovery. However, this system is not acceptable in the existing internet networks, because the highest percentage of the networks on the Internet are configured in Internet Protocol version 4. Through the proposed auto-registration system, clients can be able to discover the resources and access the services in the OCF network. Constrained Application Protocol (CoAP) is utilized for the IoT device auto-registration and accessing the services in the OCF network.

Keywords: Domain Name System (DNS); Internet of Things (IoT); domain names; device auto registration; Internet Protocol (IP); Open Connectivity Foundation (OCF); Constrained Application Protocol (CoAP)

1. Introduction

The Internet of Things (IoT) is the hottest sphere which can impact our digital world. The tools in our daily life will be furnished with raspberry pies, transceivers, microcontrollers, digital communicators and internet protocols which are suitable for stacks that will help them to be able to interconnect with each other, as well as, with their users [1]. Presently, more than 9 billion connected IoT devices are being used, and it is predicted to reach more than 26 billion devices by 2020 [2]. With the IoT concept, the Internet becomes more ubiquitous and immersive. The IoT will promote the development of a number of application scenarios that make use of the potentially substantial quantity of data analyzed by such devices to offer a new type of services for individuals, governments and companies. One of the essential aspects of IoT devices is the development of lightweight communication and interaction with a varied range of IoT devices. These devices include video surveillance cameras, lighting controls, home automation, smart vehicles, smart healthcare devices, actuators and so on. With these devices, users can be able to get the sensing data of the environment using sensors and control the actuators for changing the environment through the Internet or a local network [3]. This model certainly finds applications in a variety of domains, such as industrial and home automation, mobile healthcare, medical aids, greenhouse, energy management, automatization, traffic controlling and many other types of services [4]. Therefore, the communication and management functionalities are required for the development of interoperability of devices utilizing these services, such as auto-registration of devices to the Internet, the environment and device monitoring, device information registration for discovery and accessing indoor services in real-time [5].

Utilization of Application Programming Interfaces (APIs) can provide a user-friendly interface, and request handling service which means users easily send requests to the sensors to get sensing data and control the actuator according to their needs [6]. For IoT applications development, the service-oriented architecture (SOA) is one of the widely used development architectures which can be implemented by Representational State Transfer (REST) architecture. REST architecture-based APIs provide access to the IoT device resources through communication protocols using a Uniform Resource Identifier (URI) on the Internet [7]. Hypertext Transfer Protocol (HTTP) and Constrained Application Protocols (CoAP) can be used for providing efficient and reliable communication between clients and IoT devices [8,9]. One of the essentials among IoT deployments is IoT Platforms [10]. They can integrate several features and capabilities into a solution, such as enabling to deploy IoT applications better, faster, more cost-effective and integrated, as well as can serve as a middleware, bridge and solution. As the number of connected devices increases, several trial versions have been created for providing a common IoT Platform. Thread, IoTivity and AllJoyn are the most highlighted examples of this platform. The IoT Platform provides a set of basic principles, common standards and protocols for the general implementation of IoT applications. IoTivity is an open source project developed by the Open Connectivity Foundation and controlled by Linux Foundation. A fundamental principle of the Open Connectivity Foundation (OCF) is providing diverse vertical domains [11].

The Internet connection of IoT devices is based on different parameters such as default routers, address prefixes, and DNS servers. They can be configurated automatically via Neighbor Discovery in Internet Protocol (IP) Version 6 environment. Stateless Address Autoconfiguration and Router Advertisement Options are used in IPv6 for Domain Name System Configuration [12]. However, this type of configuration system is too complicated in current networks because most of the devices in the current network are based on IP version 4, in this case, the existing configuration of Domain Name System (DNS) names for IoT devices might be inefficient and time-wasting as the quantity of IoT devices increases speedily in a network [13].

With this paper, we have proposed an IoT device auto-registration method for the local DNS server through the OCF network. The IoT Platform can be a bridge for providing the communication between the indoor devices and the clients through the Local Area Network (LAN) such as Wi-Fi network. The proposed system includes IoT DNS, IoT Client, IoT Platform and IoT devices. For experimental environment we have developed an android application based on Java programming language. This application provides an interface for sending a request to the IoT DNS server and describes the response of the IoT DNS, as well as, after getting the IP address successfully from IoT DNS, clients can be able to connect to the IoT device for checking the indoor service. Due to the system simplicity and clear explanation, we consider the Raspberry Pi 3 Model B with a BME 280 sensor in every step. However, the design of the proposed system is flexible and adaptable for working with other home appliances, e.g., humidity, smart lighting, smart fan, smart locks to name a few. When an IoT device is connected to the Internet for the first time, it sends a registration request to the IoT Platform. The OCF based IoT Platform plays a vital role in collecting all the information, including the host, ID, resource, interface, resource type, and policy, about IoT devices [14]. In addition, it makes address registration to the DNS server. When IoT clients want to connect to the IoT devices, clients send discovery request with GET method /oic/res requests to all OCF Nodes through 5683 address port. "rt" query parameter has to be included in the request, in order to access specific indoor resource type (*jnu.rt.temperature*). When the IoT clients send the request with URL name (*device001.jnu*) to the DNS server, the DNS server checks its cache and finds out the equivalent IP address (*192.168.0.2*) according to the URL name, and sends the response with the IP address. After getting the IP address successfully, IoT clients will be able to access home services through this IP address (*coap://192.168.0.2:5683/temperature*) and discover resources through IoT devices.

The rest of the paper includes the following sections; Section 2 includes the related works which consist of DNS server working principles and different types of device to device connection ways, as well as, OCF IoTivity functionalities. Section 3 illustrates the design of our proposed IoT DNS approach and detailed explanation of working principles. Section 4 includes the structure of the device registration, device discovery, and device accessing processes. Section 5 describes the detailed implementation and testing results. Finally, there is a conclusion given in Section 6.

2. Related Works

Domain Name System (DNS) is a central part of Internet connection which allows users to discover resources from the Internet through domain names and IP addresses [15]. Domain names are human-friendly names because they are alphabetic, easy to remember and guess, whereas IP addresses are based on dot-decimal notation, so they are difficult to remember [16]. DNS is used for translating a domain name (for example, www.jnu.ac.kr) or hostname into an IP address (168.131.31.206). DNS originated with the implementation of ARPAnet (a project of the Defense Advanced Research Projects Agency) [17]. It enables individual computers to be identified uniquely to transmit and receive data over an extensive area network. The DNS contains information that allows each computer to be uniquely identified. Each computer on the network was assigned an address, which today is known as an Internet Protocol Address (IP Address). Today, each computer's IP Address consists of a unique string of digits. A domain name consists of a unique string of characters. The DNS maps each unique domain name to its unique IP Address [18]. Nowadays, two types of IP addresses are widely used; IPv4 and IPv6 [19]. The naming system is also different in these IP versions. Internet Protocol version 6 is a new internet addressing system standard which is intended to eventually replace the current IPv4, because IPv4 is available over four billion IP addresses. This means the number of IPv4s is limited [20].

First of all, we need to discuss the IPv6 DNS autoconfiguration system, Neighbor Discovery [21], which is a protocol for using in IPv6 environment, and is used to observe default gateway and operates at the Link Layer addresses of neighbors in the same subnet. NDP includes five Internet Control Message Protocol [22] packet types such as, router advertisement (RA) and solicitation (RS), neighbor solicitation and advertisement, and redirecting network. Once an IPv6 device joins a subnet for the first time, it forwards router solicitation, to a router to ask for a router advertisement. Each router occasionally sends a multicast router advertisement with router information in the same subnet. There is a DNS Search List in the options of the RA [23], which provides the domain suffixes list for domain name construction. This IPv6 system is predicted to achieve a simpler method for controlling a variety of IoT devices [24]. As we have already mentioned above, Neighbor Discovery protocol can be used for DNS autoconfiguration system in an IPv6 environment. However, most of the current Internet networks are configured in an IPv4 environment. Furthermore, the auto-configuration of the DNS naming system for every IoT device in the IPv4 environment may remarkably expand productivity in the DNS configuration in existing networks. With this DNS naming system, users efficiently manage and categorize IoT devices.

The one Machine to Machine(M2M) [25] group creates technical statements for device identifying. An object identifier (OID) is used for the identification method in an M2M device. It was designed by working with globally unique IDs for every device. The Object Identifier (OID) is an identification system created jointly by International Organization for Standardization/International electrotechnical Commission and International Telecommunication Union (ISO/IEC and ITU-T) corporations. OID utilizes a hierarchical tree structure, includes a manufacturer ID, product model ID, serial number ID, and expanded ID, respectively. The ID of the manufacturer describes the node manufacturer. The product model ID describes the node model. The node serial number is described with the serial number ID. The expanded ID means that an optional arc for the legacy device is related to the M2M node. However, a traditional networking system is based on wired networks whose parameters are different from M2M networks. M2M area networks can include not only of many devices acting themselves but also of IoT sensor networking which includes large numbers of nodes cooperating to provide sophisticated solutions. Individual management of each node is impossible in a considerable number of nodes [26]. Electronic Product Code global [27] is a GS1 enterprise to advance industry-driven system standards for the Electronic Product Code (EPC) which is based on RFID technology (radio frequency identification). 96-bit binary is used as a unique number in EPC for the identifiable. However, in order to obtain information, EPC has to be connected to the EOC information servers. In this case, limiting dynamic updates of information is inefficient. For overcoming this issue, EPCglobal proposes an Object Naming Service (ONS), that provides worldwide recovery services because converting is possible to Uniform Resource Locator (URL) [28,29]. We know that when the URL name is associated with an IP address, the information can be retrieved from the object. However, in ONS, the product type of resolution is limited.

Bonjour [30], is a networking service which supports DNS naming and service discovery service, created by Apple company. Bonjour finds and registers an Apple device and informs users what the service is used with their Apple device. mDNS [31] uses each device as a DNS server and resolver. mDNS plays a role as a carrier protocol in DNS-based service discovery, and name resolution does not use a trusted DNS server. mDNS uses a multicasting method to resolve the name of the local network. Though, it is not acceptable for multi-link networks as a result of a large amount of traffic. mDNS is a privacy problem because it can provide a naming system without any security key. Bonjour software provides service discovery with DNS based Service Discovery protocol (DNS-SD) [32]. DNS Service Discovery protocol supports a list of port numbers and service devices for hosts.

In Mobile Ad-hoc Network (MANET) is also used as another name service system architecture [33]. This method creates a unique domain name using the device ID or user ID. However, the drawback of this naming system is that the user cannot easily read when retrieving device information [34,35].

Keuntae Lee at al. [36] introduced the Domain Name System (DNS) Name Autoconfiguration (DNSNA) for IoT devices in IPv4 and IPv6 networks. According to their proposed approach, for IoT DNS name autoconfiguration Neighbor Discovery (ND) protocol and Dynamic Host Configuration Protocol are used in IPv6 and IPv4 networks, respectively. DNSNA includes four main contributions, such as a DNS naming framework, the DNS naming format for the device discovery, physical-contact based Near Field Communication (NFC) for the IoT device authentication, and the service discovery through DNS service resource records. The proposed approach evaluation results describe that in the DNSNA scenario the average number of packets and the packet accumulation volume can be reduced by 60.8% and 97% respectively. However, the system frameworks are too complex and generic which, despite their many positive facets, attribute to slow response time and latency which are the fundamental need of modern IoT networks. Our proposed approach is lightweight and targets modern latency-aware applications for the IoT resource auto-registration and smart service access for indoor services based on OCF IoTivity.

Tomohiro Yanase at al. [37] developed a flexible name autoconfiguration method which can autogenerate Fully Qualified Domain Name (FQDN) based on IoT devices information, and this function can be installed to the home gateway. According to the IPv6 characteristics, the number of the IP addresses in the IPv6 environment is higher than IPv4 addresses, and they are described in hexadecimal string format. Consequently, when clients communicate with the IPv6 address assigned IoT device, this address should be a user-friendly identifier such as the authors introduced FQDN instead of IP addresses. Home Getaway plays a role as a generator for FQDN of IoT devices and registers domain names to the DNS server. When an IoT device is connected to the home network, it automatically generates an IPv6 address. For obtaining information, home getaway sends a multicast device search request and IoT device response with device information such as manufacturer name, model number and IP address. After that home getaway configures FQDN according to the received packet, the FQDN are generated for the IoT device, home getaway sends the registration request to the

DNS server with the IoT devices FQDN, and if the received IoT device FQDN is unique on the Internet and the DNS server, this IoT device is successfully registered to the network. One of the disadvantages of this system, if the generated FQDN is not unique on the Internet, the home getaway has to repeatedly generate a new FQDN for the IoT device until it becomes unique on the Internet. It creates too much overlapping on the network, as well as this system being proposed for the IPv6 environment.

The Open Connectivity Foundation (OCF) is an industry group [38] that is committed to developing specification standards, promoting interoperability guidelines, and providing certification programs for the Internet of Things devices. It has become one of the [39] industry standardization organizations of IoT including Microsoft, Intel, Samsung Electronics, Electrolux and Qualcomm [40].

The IoTivity Framework API provides a framework for developers and can be used with different operating systems and multiple programming languages [41]. The IoTivity framework is based on a constrained application protocol, and it can provide optimized and dedicated protocols for the Internet of Things devices. Resource Encapsulation is one of the essential parts of the IoTivity framework; it is an abstract layer which includes common resource function modules. Developers' work is becoming easier with this module because it can provide functionalities for both the server and client side. More precisely, for the client side, it supports monitoring the presence of resources in the network with Resource cache and Broker functionalities. For the server side, it can provide a direct and simple way for creating the resource and setting the properties and attributes. The OCF IoTivity framework can operate as middleware in different operating systems and platforms; the framework manages four essential functions including [42,43], resource discovery, data transmission, data management and device management. Resource discovery provides various discovery structures for resources and devices via remote and proximity connection. Data Transmission supports exchanging information and controlling according to the stream and message model. Data Management is utilized for storing, collecting, managing and analyzing the data from multiple resources. Device Management provides device configuration, device management and diagnostics of devices.

3. Proposed IoT Device Auto-Registration Based on OCF

In our proposed approach, the IoT devices make auto-registration automatically to the DNS server through the IoT Platform. Figure 1 describes the model of the proposed system architecture. This architecture includes IoT devices, IoT Platform, DNS server and IoT Client. The IoT Platform has a registration resource, discovery resource and data management functions. The IoT Platform plays a role as a bridge between an IoT device and network when an IoT device is connected to the network via an IoT Platform. The IoT Platform collects information of the IoT device, makes auto-registration of the IoT device's IP addresses and URL names, and makes address registration to the DNS server. When the IoT clients send requests, the URL name request is via android application or internet browser to the DNS server. The DNS server validates the URL name and finds out IP address according to the URL name after that DNS sends an acknowledgment with the IP address to the client. After taking the IP address successfully, the client can be able to send a request to the IoT devices for exploring temperature/humidity or other types of information about the condition. For connecting and changing the information we used CoAP [44], which is a network-oriented protocol that has similar characteristics, like HTTP. However, CoAP also enables low overhead and multicasting functions. The REST architecture is used in CoAP, which is a conventional system for accessing Internet resources.

On the one hand, Constrained Application Protocol supports the Uniform Resource Identifier. REST architecture uses GET, PUT, POST and DELETE methods for corresponding to create, read, update and delete operations. On the other hand, CoAP is based on lightweight User Datagram Protocol (UDP) protocol, which supports IP multicast that fulfills group communication for the Internet of Things. For compensating for the reliability of UDP protocols, CoAP defines retransmission mechanisms and supports a resource description for discovery mechanisms [45].

As it can be seen that, in the first step, the IoT device makes auto-registration to the IoT Platform. This registration includes IoT device's parameters, such as "host," "ID," resource type, resource interface, entity handler and resource property. In the second step, the IoT platform collects that parameters in its database and checks them every time with the Resource Directory (RD) registration resource and RD Resource Discovery. Besides that, the IoT Platform makes address registration to the DNS server, and this registration includes only IP addresses and URL names. When a client wants to connect to the IoT device, he or she sends a request with the URL name, and the URL name comes to the DNS server, and the server checks its mapping table and response with equivalent IP address (steps 4 and 5). After taking the IoT device's IP address successfully, the IoT client can be able to connect to the IoT device for taking information about the indoor environment such as temperature, humidity, is led on or off and so on (step 6). At the end of the process, the IoT device sends the response with temperature level to the IoT client.

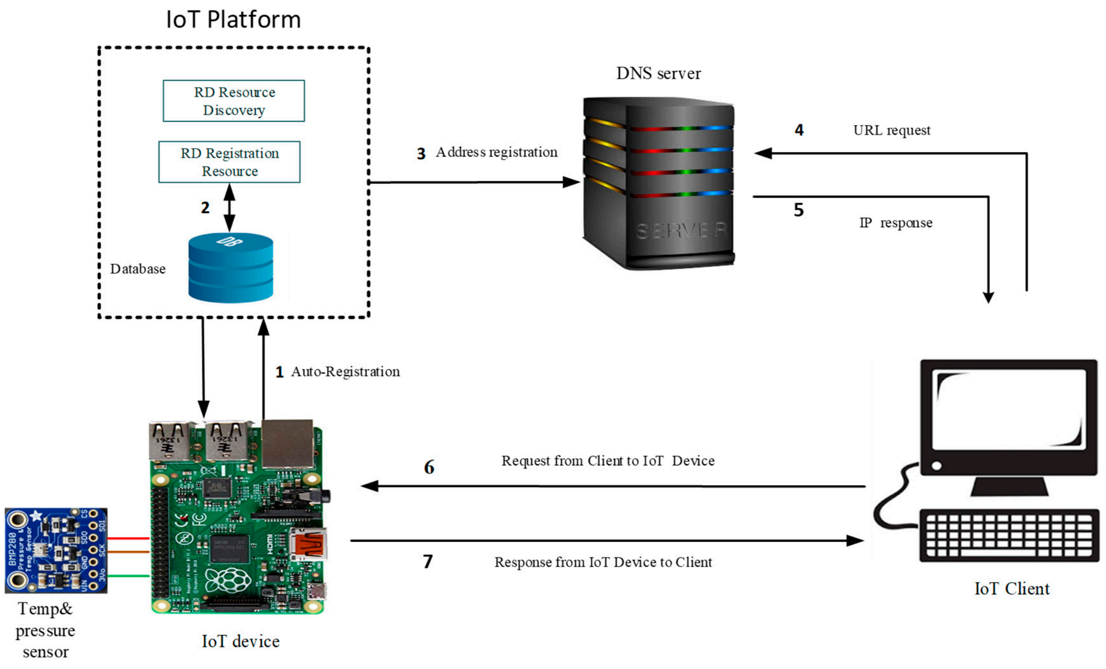


Figure 1. Proposed system architecture for auto-registration and resource discovery.

In general, there are two most common ways that the devices can acquire IP addresses. They are manual address assignment and automatic address assignment. Almost all networks have automatic IP address assignment with Dynamic host configuration protocol, because this protocol is more convenient and reliable. Sometimes there are problems with reaching the DHCP server; in this case, we can set up IP addresses manually. Both address assignment systems are acceptable for our system.

Figure 2 shows a sequence diagram for the proposed system architecture. Firstly, the IoT device local parameters are configured. When the IoT device is connected to the IoT Platform, the device sends IoT device resource values to the IoT Platform, and IoT Platform collects all the information about the IoT device to its database. After that, the IoT platform sends a request for address registration to the DNS server (IP address and URL name). When the client tries to connect to the IoT device, he or she sends a request with the URL name (*device001.jnu*) to the client. Moreover, the request comes to the DNS server, and the DNS server selects a proper IP address (192.168.0.2) from its mapping table according to the URL name, after local IoT DNS sends an acknowledgment with the IP address. After taking the IP address, the IoT client can be able to access the indoor environment, and the IoT device sends the temperature of condition to the client {"temperature": "22,5"}.

Figure 3 describes an example of IoT device registration resources to the IoT Platform. The given properties provide information which is utilized for discovering IoT devices over the Internet by clients. The properties of the structure are based on common properties of the OCF IoTivity core specification that is a unique format for registration of IoT devices. These values consist of "host," "device id," resource list, resource interface list, resource type list and list of methods. As can be seen, “host” is an IoT device IP address and a host number, “id” is a unique identifier of the device, and it is relatively used with “href” in order to get endpoint information. Items in the “resources” play a role in providing a request by IoT clients: “dataType” are derived JavaScript Object Notation (JSON) values. The data types can be adaptable for a specific utilization; for instance, the string length can be changed for a particular condition. The “oic” is utilized for OCF defined interfaces, as well as, it is accepted as the first segment in the Interface property value. The “oic.if.baseline” is applicable in RETRIEVE, UPDATE methods and used for defining a view into all properties of a resource. The “policy” parameter provides diverse rules for reliable accessing to the resource. This parameter is configured by the setting of “key-value” pairs. OCF defined resource types and interfaces are specified using JSON and RESTfull API Modeling Language (RAML), respectively.

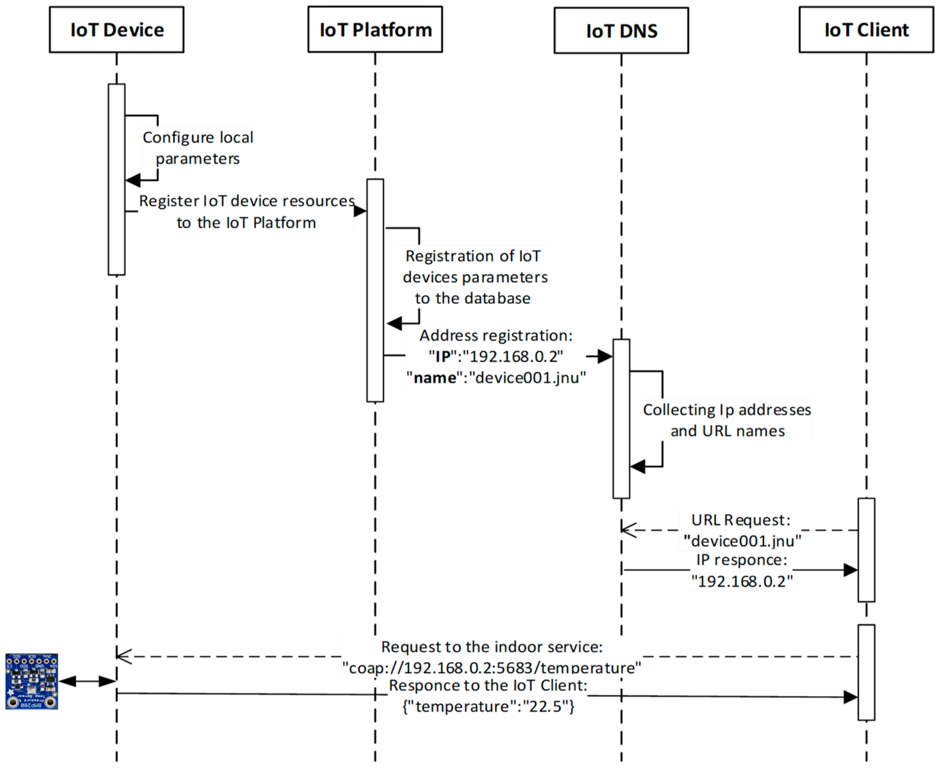
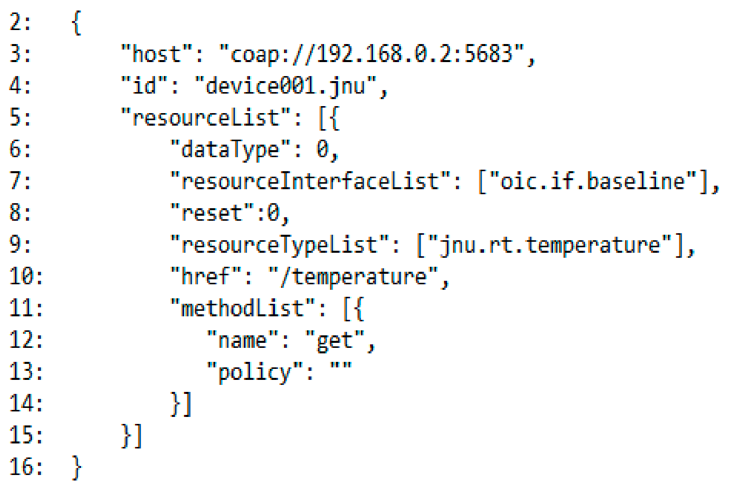

IoT Device
IoT Platform
IoT DNS
IoT Client
Configure local parameters
Register IoT device resources to the IoT Platform
Registration of IoT devices parameters to the database
Address registration: "IP":"192.168.0.2" "name":"device001.jnu"
Collecting Ip addresses and URL names
URL Request: "device001.jnu"
IP responce: "192.168.0.2"
Request to the indoor service: "coap://192.168.0.2:5683/temperature"
Responce to the IoT Client: {"temperature":"22.5"}

Figure 2. Sequence diagram of the proposed Internet of Things (IoT) device auto-registration.



```
{
    "host": "coap://192.168.0.2:5683",
    "id": "device001.jnu",
    "resourceList": [{
        "dataType": 0,
        "resourceInterfaceList": ["oic.if.baseline"],
        "reset":0,
        "resourceTypeList": ["jnu.rt.temperature"],
        "href": "/temperature",
        "methodList": [{
            "name": "get",
            "policy": ""
        }]
    }]
}
```

Figure 3. The IoT device registration resources.

Figure 4 illustrates the overall collaboration of the proposed system architecture. As we can see from the figure, firstly, the IoT device sends a device registration request to the IoT Platform; secondly, the IoT Platform makes Address Registration to the DNS server. When a client wants to access the indoor service, he or she sends a URL name request via a browser or Android application, the URL name request comes to the DNS server, and the DNS server sends a response with the IP address; after taking the IP address successfully, the client is able to send a request to the IoT device. At the end of the process, the IoT device sends a response with temperature level.

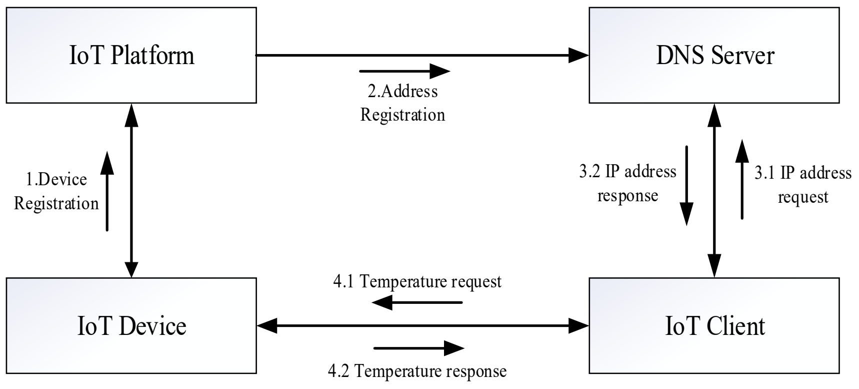


Figure 4. Collaboration diagram of the proposed system.

4. Device Registration, Device Discovery and Sensing Data Processes of the Proposed System

4.1. Device Registration

Figure 5 describes the flowchart of the IoT device registration to the DNS server. First of all, the IoT device is installed to the home, workspace or required place after it is connected to the Internet through a wired or wireless connection. When the device is connected to the internet, the device needs a unique IP address over the network. Usually, IP addresses are chosen dynamically through a DHCP server because a manual method for applying IP addresses is cumbersome, as remembering all the clients and their IP addresses in one host is impossible. After taking the IP address successfully, the IoT device sets up an IP address and URL name. Then, the IoT device sends a registration request with the URL name, IP address, as well as some other parameters of the IoT device such as resources list, resource interface list, resource type list and method list to the IoT Platform.

The IoT Platform collects these parameters to its database and sends to the DNS server for address registration, and the DNS server makes validation of these parameters and, if each given parameter is acceptable and with the correct values, the IoT device is registered successfully to the DNS server. On the other hand, if there is some problem with the IP address or URL name, the DNS Server gives a response with the message "IP address and URL name are invalid." In this case, IoT device installation or parameters have to be rechecked.

The Resource Directory in the OCF Platform can provide a discovery service which allows devices to publish their resource value information to an RD, and updates resource information and allows deleting of resource information from the resource directory [33]. Figure 6 describes the sequence diagram of the device registration process. First of all, the IoT device is installed and connected to the Internet, after connection to the internet router gives an appropriate unique IP address to the IoT device, and this IP address and URL name become related to this IoT device. After that, the IoT device sends a registration request to the IoT Platform with IoT device parameters. These parameters are collected to the RD Resource Discovery, Registration Resource and Database. The IoT Platform plays a role as a bridge; it collects all of the IoT devices in its host. The next step is sending a request to the DNS server for Address Registration. The DNS server validates all IP addresses, URL names and if they are valid in this network the DNS server registers the IoT device successfully. If there are some invalid parameters in the registration process, registration fails and the IoT device should be reinstalled.

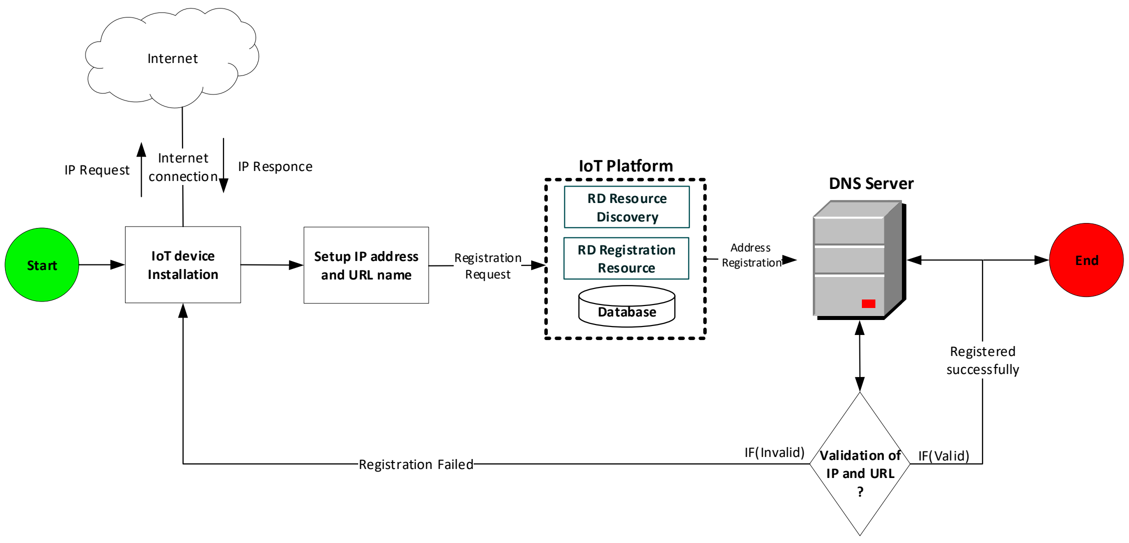


Figure 5. IoT device registration process flowchart of the proposed approach.

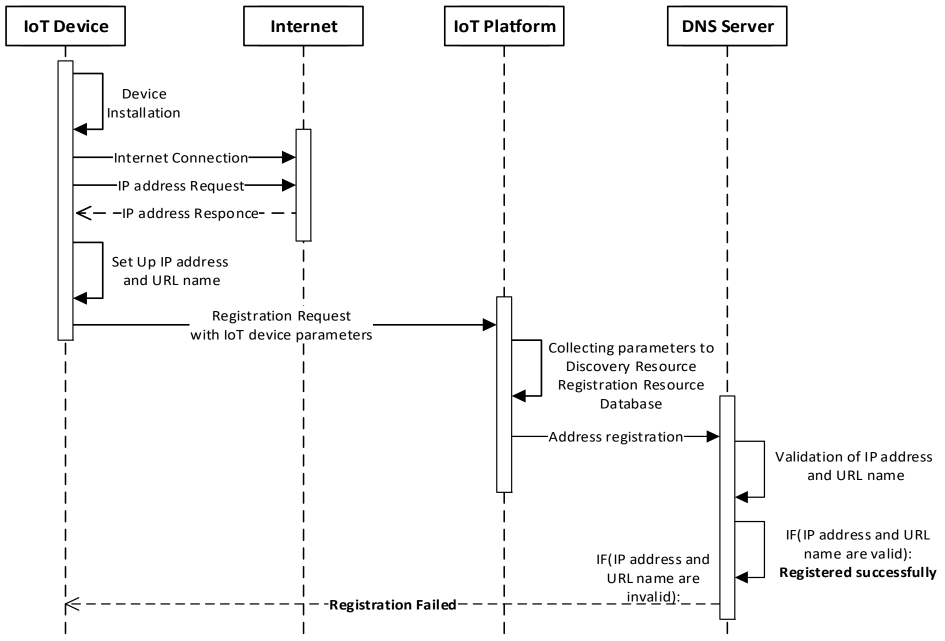

IoT Device
Internet
IoT Platform
DNS Server
Device Installation
Internet Connection
IP address Request
IP address Responce
Set Up IP address and URL name
Registration Request with IoT device parameters
Collecting parameters to Discovery Resource Registration Resource Database
Address registration
Validation of IP address and URL name
IF(IP address and URL name are valid): Registered successfully
IF(IP address and URL name are invalid):
Registration Failed

Figure 6. IoT device registration sequence diagram of the proposed system.

4.2. Device Discovery and Sensing Data

Figure 7 illustrates a general overview of the device discovery and sensing data response process of the proposed system. The figure is divided into two parts: the first part describes the device discovery and taking the IP address from the DNS server; the other part shows the process of the sensing data from the IoT device. As we have explained above, the client sends a request to the IoT DNS server for getting an IP address, and the DNS server checks its Map Table and response with an appropriate IP address according to the URL name. After getting the IP address, the client is able to send a request to the IoT device (coap://192.168.0.2:5683/temperature), and the IoT device sends a response with indoor temperature.

Device and resource discovery are provided by CoAP endpoints. This function is critically essential in the M2M applications, in which there is no people's role in the process, and there is a vulnerability due to static interfaces. CoRe Link Format provides resource discoverable services without fully manual configuration for improving interoperability in a CoRe environment. This process depends on the server in which discoverable resources are used. Figure 8 describes the sequence diagram of the device discovery. Firstly, the client makes a request preparation, he or she opens a browser, enters the URL name to the address bar and clicks enter. In a sequence diagram, the client sends a request with device001.jnu URL name and send it to the DNS server. After that, the DNS server searches this URL name in its Map Table. If there is no equivalent IP address according to the URL name in the Map Table, the DNS server sends a "Not Found" message, this process describes the loop process, and it will continue until the client enters the available URL name. When the DNS server finds a valid IP address according to the URL name, it sends an acknowledgment with an IP address such as 192.168.0.2. When a client reaches the IP address successfully, he or she can send a request to the IoT device. The IoT device sends an acknowledgment with temperature.

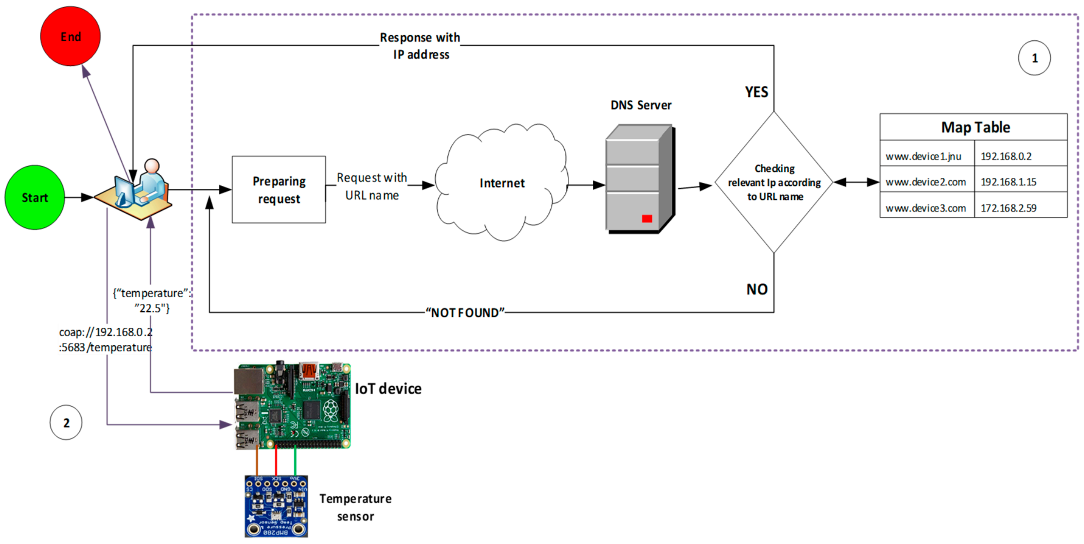


Figure 7. The general overview of the device discovery and sensing data process.

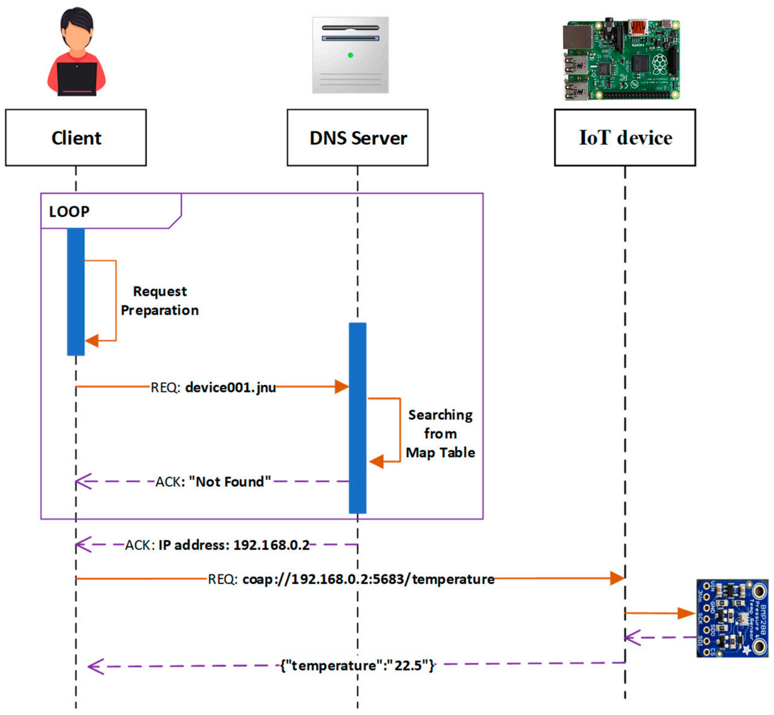


Figure 8. Sequence diagram of the device discovery and sensing data.

5. Implementation Results

5.1. Implementation of IoT DNS Based on OCF IoTivitiy

Implementation of the proposed system consists of IoT devices, an IoT Client, IoT Platform based on OCF network and IoT DNS server. We have IoT devices which can provide an indoor service to the home. The IoT client can access the indoor services through the Internet. The IoT Platform plays a role as a bridge for IoT devices and the IoT DNS server. The proposed system is based on Constrained Application Protocol. The proposed local IoT DNS system provides IoT device auto-registration and smart service access based on the OCF Platform. IoT devices register their information by sending a POST/oic/rd request to the IoT Platform, and the IoT Platform makes address registration to the local IoT DNS. Once the IoT device's IP address and URL name are registered to the DNS server, the user can use the services of the IoT device though taking information from the local IoT DNS server.

Figure 9 presents the environment of the experiment which consists of the OCF Platform and the Internet. For internet connectivity to the system, we can use LAN such as Wi-Fi. Additionally, the system includes the IoT client, IoT DNS and IoT devices, which are connected with each other through OCF protocol in the LAN. In the implementation of the local IoT DNS server; the IoT client can use his or her Android phone which is provided with the User Interface for interacting with the clients. The IoT devices are Raspberry Pies which are connected to the BME 280 temperature sensor, fan, LED and servomotor, which are utilized for providing indoor smart services.

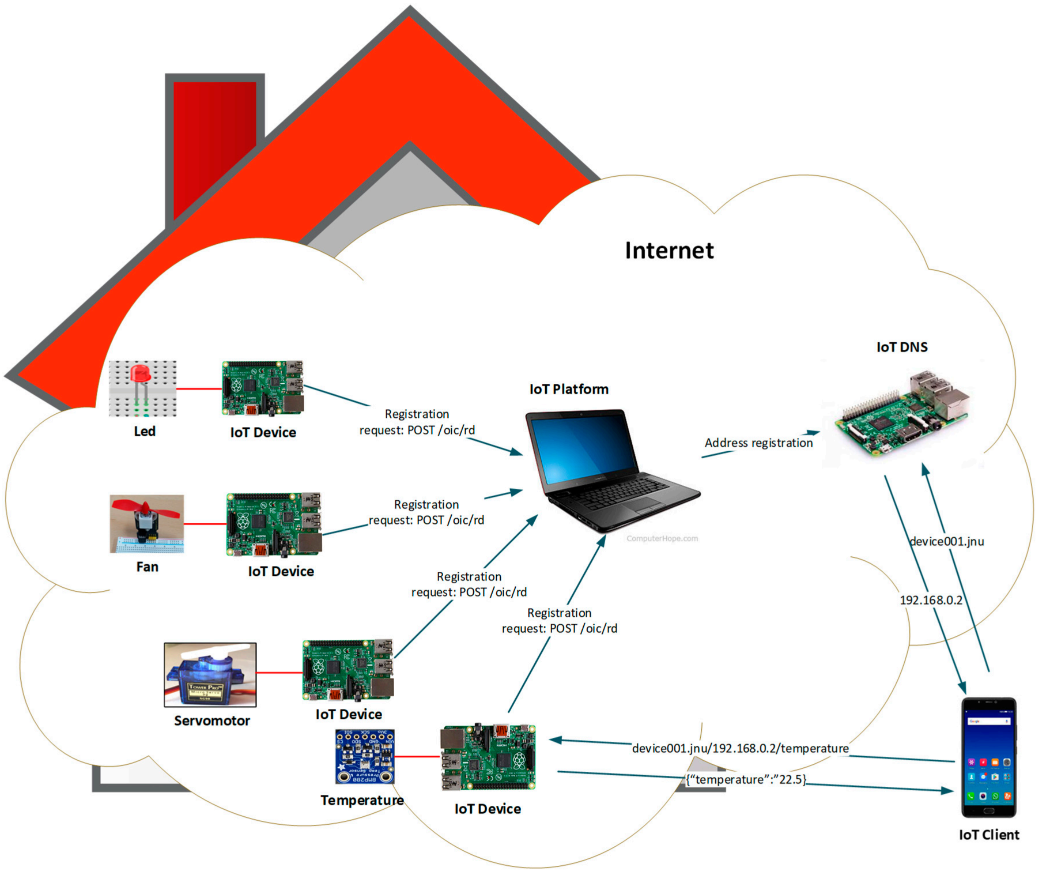

Internet
IoT DNS
IoT Platform
Led
IoT Device
Registration request: POST /oic/rd
Address registration
Fan
IoT Device
Registration request: POST /oic/rd
device001.jnu
192.168.0.2
Registration request: POST /oic/rd
Registration request: POST /oic/rd
Servomotor
IoT Device
Temperature
IoT Device
device001.jnu/192.168.0.2/temperature
{"temperature":"22.5}
IoT Client

Figure 9. Experimental environment.

Table 1 describes the experimental environment tools for the development environment. Android Studio 3.0.1 was used to the application development and performance analysis. Java programming language was utilized for application implementation. We used five Raspberry Pies for IoT devices and the IoT DNS server.

Table 1. Implementation environment.

Entities	Hardware	Platform	Library and Framework
IoT DNS	Raspberry Pi 3 Model B	Android Things 0.8	IoTivity 1.3.1
IoT Client	Samsung Galaxy A9	Android 8.0.0 (API 26)	IoTivity 1.3.1
IoT Devices	Raspberry Pi 3Model B, BME280 sensor Fan Motor Led Servomotor	Android Things 0.8	IoTivity 1.3.1
IoT Platform	PC (CPU: Intel i5-4570)	Ubuntu 14.04	JRE System Library (JavaSE-1.8)

The IoT DNS server runs on Raspberry Pi 3 Model B, and the operating system is Android Things 0.8, and program implemented in Java programming language.

The IoT Device runs on Raspberry Pi 3 Model B which operates Android Things 0.8. The IoT devices are connected to the BME 280 temperature sensor, fan, led and servomotor.

The IoT Client runs on Samsung Galaxy A 9 and operates Android 8.0.0, and IoTivity 1.3.1 was used for development for the OCF client.

The IoT Platform runs on Ubuntu 14.04 (x64) platform, and JRE System Library [JavaSE-1.8] was used for a library, and Java programming language was utilized for implementation of the system.

Figure 10 presents the testbed network environment of the proposed system. The testbed network environment is a campus subnetwork at Jeju National University. As we have already mentioned above, we used five Raspberry Pi 3 Model B as IoT devices and IoT DNS server. All of the devices are connected to the Internet through wired and wireless connection to our Wi-Fi router, and they have power connection.

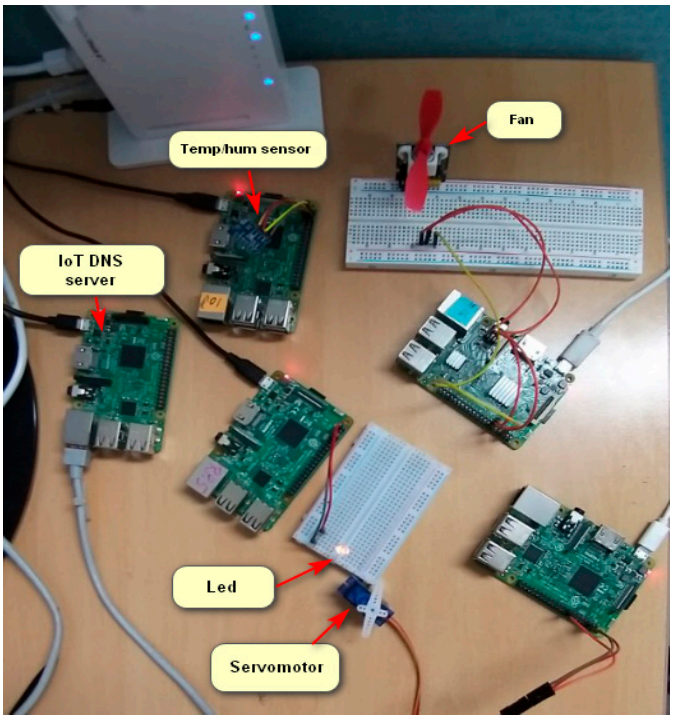
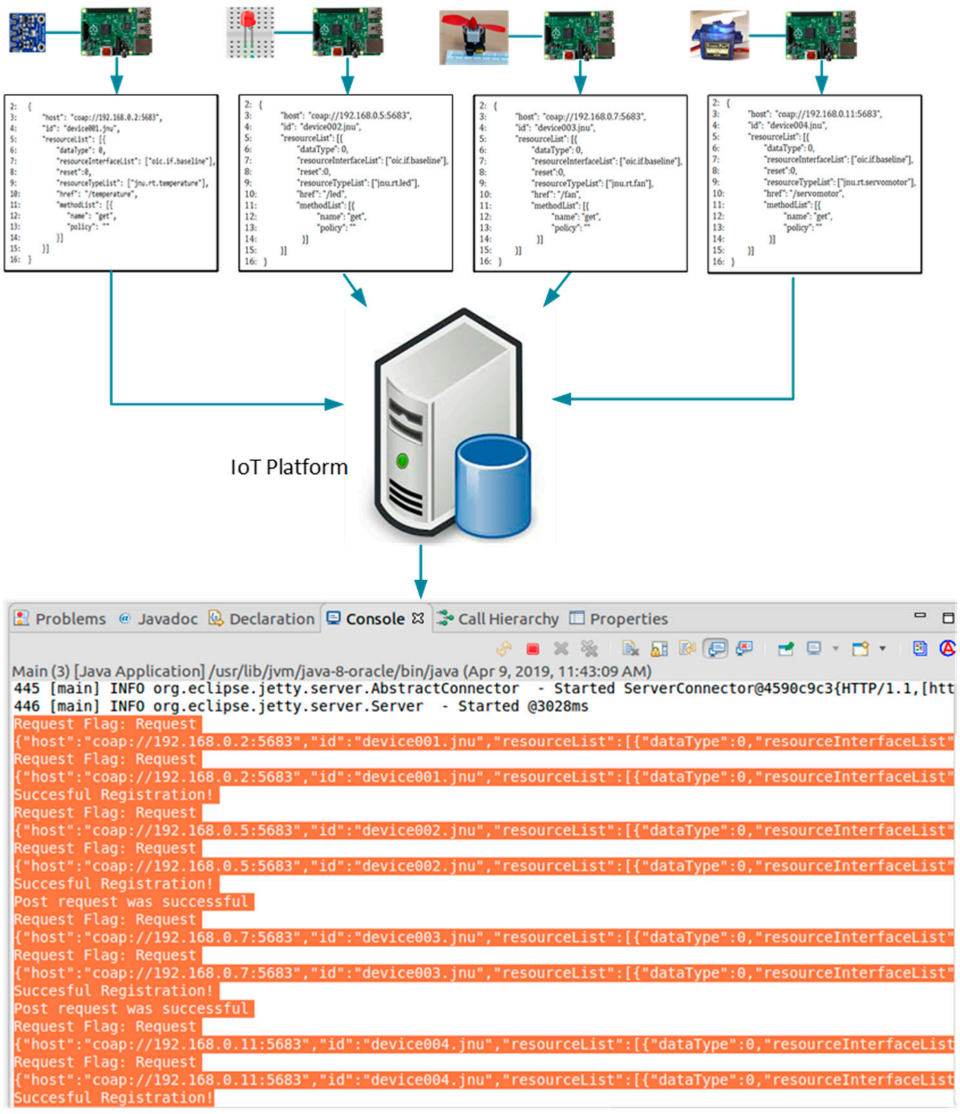


Figure 10. The testbed network environment of the IoT Domain Name System (DNS) and IoT Devices.

As we have already mentioned above for our experimental and testbed environment process, we used four types of IoT devices in order to control the indoor environment. Namely, temperature sensor, fan, led and servomotor. However, the system design is flexible and adaptable for a massive number of IoT devices. IoT device auto-registration results include all of the IoT device registration results for the IoT Platform.

The IoT Platform includes the IoT server and database for the registration of IoT devices. The IoT platform is a high-performance device which has sufficient computing ability to communicate with the IoT DNS server and IoT devices. The database is used for saving the information of the IoT devices and sensing data that is collected from the environment. Figure 11 describes the implementation result of the IoT devices auto-registration to the IoT platform. The parameter values of the IoT devices are sent to the IoT Platform. The resource profile is based on a JSON format that is implemented in the Android Platform. When the IoT devices are connected to the IoT platform, the IoT devices send POST request with its parameters to the IoT Platform, and these parameters automatically register to the IoT platform. As can be seen, the parameters include host, id, resource models such as resource list, resource type list and method list of the IoT device. If we take the first IoT device as an example, this device's "host": "coap://192.168.0.2:5683" is described based on <host>:<port> method. <host> shows the name of the network or endpoint network, and <port> addresses network port number. Id describes device URL name "device001.jnu." In the figure, we have four types of IoT devices and four of them have sent a POST registration request with their parameters. After that, every IoT device is registered automatically to the platform. When the IoT device auto-registration process finishes, the system shows a "Successful registration!" message.



```
{
    "host": "coap://192.168.0.2:5683",
    "id": "device001.jnu",
    "resourceList": [{
        "dataType": 0,
        "resourceInterfaceList": ["oic.if.baseline"],
        "reset":0,
        "resourceTypeList": ["jnu.rt.temperature"],
        "href": "/temperature",
        "methodList": [{
            "name": "get",
            "policy": ""
        }]
    }]
}
```

```
{
    "host": "coap://192.168.0.5:5683",
    "id": "device002.jnu",
    "resourceList": [{
        "dataType": 0,
        "resourceInterfaceList": ["oic.if.baseline"],
        "reset":0,
        "resourceTypeList": ["jnu.rt.led"],
        "href": "/led",
        "methodList": [{
            "name": "get",
            "policy": ""
        }]
    }]
}
```

```
{
    "host": "coap://192.168.0.7:5683",
    "id": "device003.jnu",
    "resourceList": [{
        "dataType": 0,
        "resourceInterfaceList": ["oic.if.baseline"],
        "reset":0,
        "resourceTypeList": ["jnu.rt.fan"],
        "href": "/fan",
        "methodList": [{
            "name": "get",
            "policy": ""
        }]
    }]
}
```

```
{
    "host": "coap://192.168.0.11:5683",
    "id": "device004.jnu",
    "resourceList": [{
        "dataType": 0,
        "resourceInterfaceList": ["oic.if.baseline"],
        "reset":0,
        "resourceTypeList": ["jnu.rt.servomotor"],
        "href": "/servomotor",
        "methodList": [{
            "name": "get",
            "policy": ""
        }]
    }]
}
```

```
Main (3) [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (Apr 9, 2019, 11:43:09 AM)
445 [main] INFO org.eclipse.jetty.server.AbstractConnector  - Started ServerConnector@4590c9c3{HTTP/1.1,[htt
446 [main] INFO org.eclipse.jetty.server.Server  - Started @3028ms
Request Flag: Request
{"host":"coap://192.168.0.2:5683","id":"device001.jnu","resourceList":[{"dataType":0,"resourceInterfaceList"
Request Flag: Request
{"host":"coap://192.168.0.2:5683","id":"device001.jnu","resourceList":[{"dataType":0,"resourceInterfaceList"
Succesful Registration!
Request Flag: Request
{"host":"coap://192.168.0.5:5683","id":"device002.jnu","resourceList":[{"dataType":0,"resourceInterfaceList"
Request Flag: Request
{"host":"coap://192.168.0.5:5683","id":"device002.jnu","resourceList":[{"dataType":0,"resourceInterfaceList"
Succesful Registration!
Post request was successful
Request Flag: Request
{"host":"coap://192.168.0.7:5683","id":"device003.jnu","resourceList":[{"dataType":0,"resourceInterfaceList"
Request Flag: Request
{"host":"coap://192.168.0.7:5683","id":"device003.jnu","resourceList":[{"dataType":0,"resourceInterfaceList"
Succesful Registration!
Post request was successful
Request Flag: Request
{"host":"coap://192.168.0.11:5683","id":"device004.jnu","resourceList":[{"dataType":0,"resourceInterfaceList
Request Flag: Request
{"host":"coap://192.168.0.11:5683","id":"device004.jnu","resourceList":[{"dataType":0,"resourceInterfaceList
Succesful Registration!
```

Figure 11. IoT device auto-registration result.

Figure 12 shows the implementation result of the IP address discovering by the IoT client. The screenshot of our proposed system android application shows obtaining the IP address from IoT DNS. The process includes four main phases. Firstly, the IoT Client is connected to the IoT DNS server through coap://192.168.0.3:5683/dns. Secondly, for getting the IP address, the IoT client has to write a URI name (device001.jnu) to the address bar. Then the client can click Get IP from the DNS button to get the valuable IP address according to the URI name. In the last phase, the IoT DNS sends an acknowledgment with an IP address (192.168.0.2). If the IoT DNS server does not include the IP address requested by an IoT Client, the IoT DNS response with a "Not Found" message.

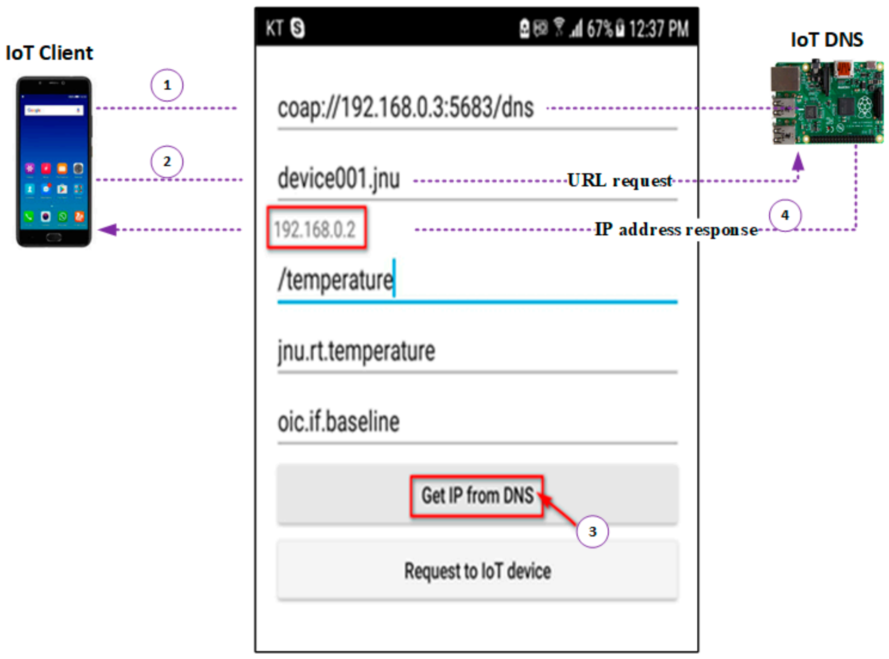


Figure 12. The result of getting an Internet Protocol (IP) address from IoT DNS.

One of the crucial aspects of the OCF standard is an interface. The *oic.if.baseline* interface consists of all the properties of the resource. When an IoT client wants to know all properties of a resource, the "baseline" interface is utilized. The client sends the URI query parameter definition "if=oic.if.baseline" in a retrieve request. When this query parameter definition is added, the server will send the response with a resource representation which consists of all of the implemented properties of a resource, such as, temperature, light, led and so on. The server sends an error message when it is not able to send back all of the resource representations.

Figure 13 describes the implementation result of accessing the indoor service. It is clear that IoT client is connected to the local IoT DNS server with *coap://192.168.0.3:5683/dns*. The IoT client enters the URI name (*device001.jnu*) to the address bar and clicks "Get IP from DNS button." After that, the IoT DNS server sends, in response, the IP address to the client (*192.168.0.2*). After getting tje IP address from the IoT DNS, the IoT client is able to connect with the IoT device which equips with a BME 280 and with "*coap://192.168.0.2/temperature*" request the client clicks the "Request to IoT device" button for accessing the temperature sensing service. Once the temperature sensing service is requested, then a temperature value is returned in JSON format.

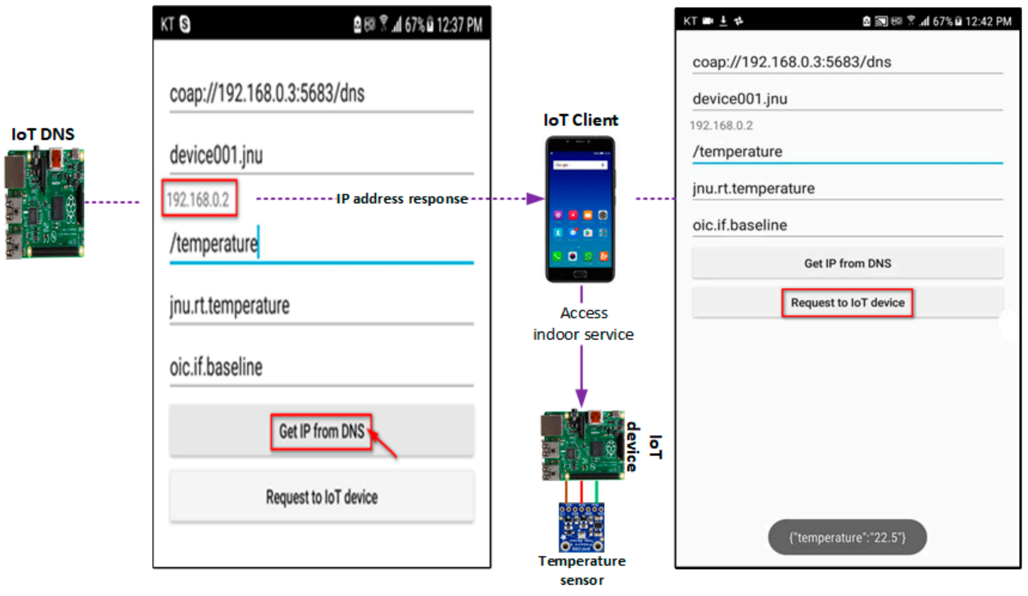


Figure 13. Fragment of IoT resource discovery and accessing sensing data.

5.2. *Performance Evaluation*

For the performance evaluation of the proposed system, we checked the overall system with three steps: IoT device registration, the IP address request-response process, and accessing indoor service. These processes are monitored by using the Android Studio. Additionally, we ran our evaluation according to three groups, in order to analyze the flexibility and adaptability of the propped approach.

5.2.1. IoT Device Registration Performance Evaluation

Figure 14 shows the evaluation results for the IoT device registration Round Trip Time (RTT). In this experiment, we present the evaluation results through the collection of RTT 25 times. The same hardware and network environment do the RTT for each registration. The RTT is collected in the IoT device through the time difference between the time for sending the registration request and the time for receiving the registration response. The results describe the RTTs are taken by between 2990 and 3000 ms, the average is 2996 ms, and the standard deviation is 3.3 ms. According to the results, IoT device registration takes about 3 s in our proposed system. The quantity of the latency for every device registration is approximately 5 ms, which compared to the average registration time latency is very low.

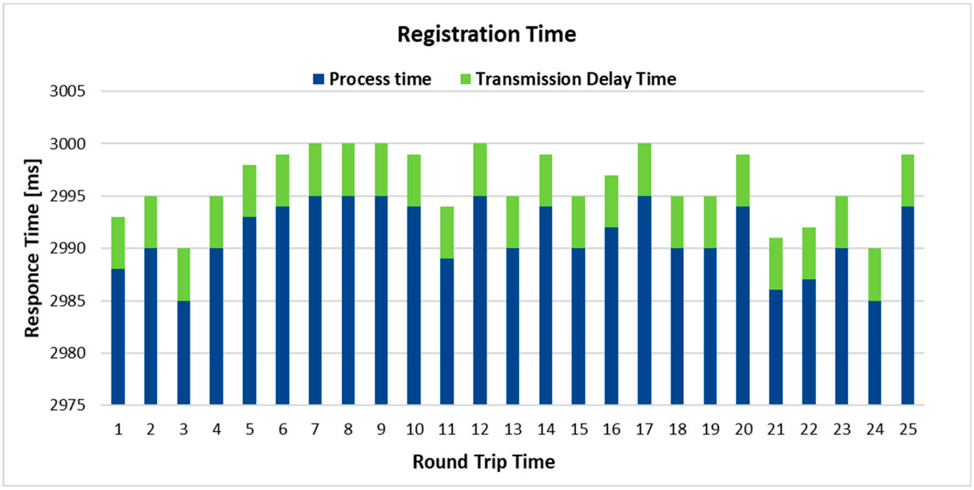


Figure 14. Round Trip Time of the IoT device registration.

Figure 15 includes the comparison among three different numbers of IoT devices registrations in order to investigate the response time of the proposed system. When the new IoT device is connected to the home network, it sends an auto-registration POST request with IoT device's parameters. We simulated for a period of 100 ms. In general, the response time increases as the number of users increase and querying the system at the same time. At the first round, we assume that 200 IoT devices sent the registration request to the IoT DNS server for auto-registration, 500 devices at the second round, and lastly, we evaluated the performance of implementation by increasing the number of IoT devices to 800. Figure 14 shows that when only one IoT device sends a registration request, registration takes approximately 2996 ms, but when 200 IoT devices send registration requests at the same time, it takes between 62,000 and 81,000 ms. The registration time of the system for the first two groups is nearly the same (in only one point the second group spends response time more than 8500 ms). However, when we increase the number of IoT devices to 800, the IoT devices registration time increases significantly compared to the other two groups. More precisely, after increasing the number of IoT devices to 800, registration time reaches 92,000 ms, which means as the number of the devices increases response time also increases.

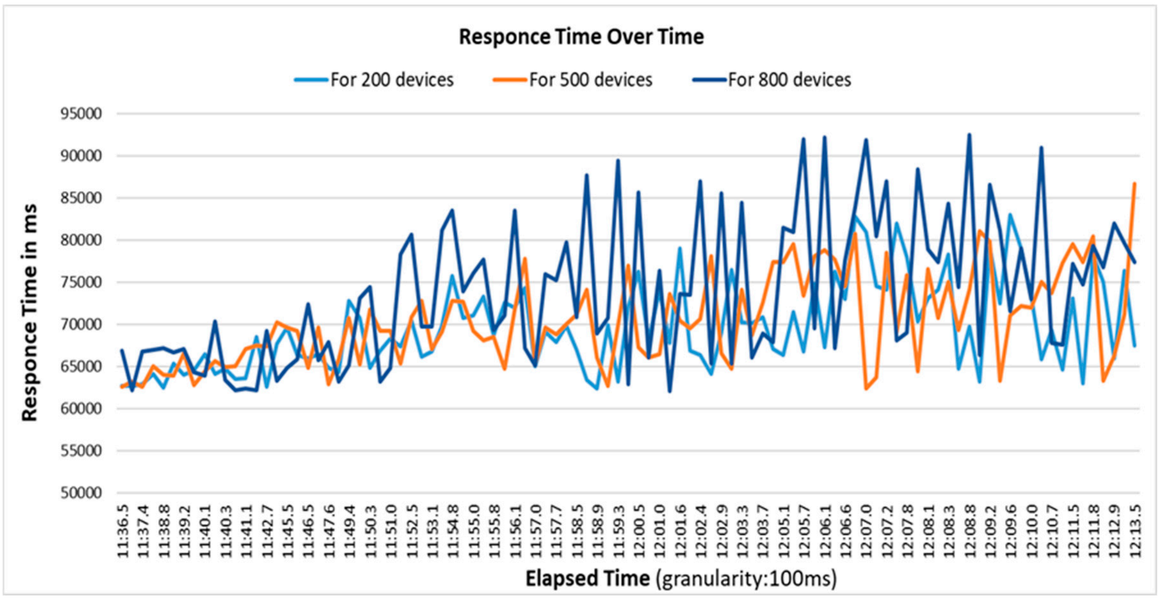


Figure 15. Device registration response time in a different simultaneous request.

5.2.2. IP Address Request-Response Performance Evaluation

Figure 16 describes the evaluation results for the IP address request-IP address response process Round Trip Time (RTT). The evaluation results were tested through the collection of RTT 25 times. Our proposed system's application interface shows the connected available IoT DNS server (coap://192.168.0.3:5683/dns). In order to get the IP address from the IoT DNS server, the client sends a request with the URL name and clicks "Get IP address" button from the Android application and the IoT DNS server sends a response with the IP address of the IoT device, this process which is sending a request and taking response IoT DNS server are described in 1 RTT. The same hardware and network environment are used to the RTT of each request and response. The results illustrate the RTTs are between a minimum of 25 ms and the maximum 49 ms, the average RTT is 37 ms, and the standard deviation is 6.9 ms.

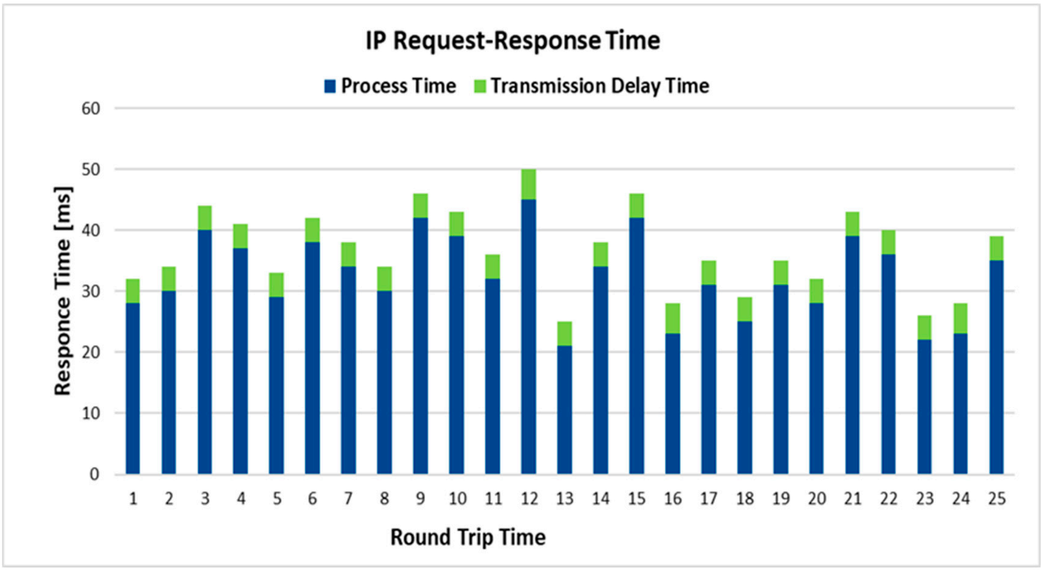


Figure 16. Round Trip Time of the IoT device registration.

Figure 17 describes the IP request-response time results for the 100 ms period. As we have already explained, response time increases as the number of requests increase. We evaluated the results according to the three categories of IP address requests. The first category includes 200 IP address requests; the second group consists of 500 IP address requests, and the last group includes 800 requests at the same time. In the initial ms of the evaluation response time for 200 requests, 500 requests and 800 requests are equal to 63,300, 62,200 and 62,500 ms, respectively. According to the graph, the maximum response time is more than 91,000 ms. However, the response time of the system for the three groups is almost the same; we are not able to see a significant difference among them. Although the number of IP requests increases but the response time of the system remains stable, the results illustrate that the proposed system is flexible and adaptable enough.

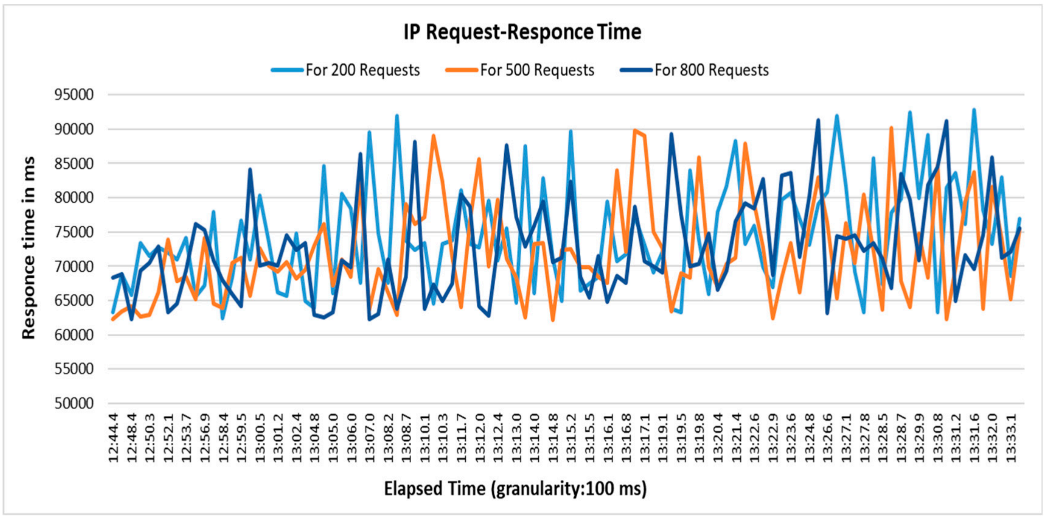


Figure 17. IP address response time for a different simultaneous request.

5.2.3. Performance Evaluation of Accessing Indoor Service

After taking the IP address successfully from the IoT DNS server, the client sends a GET request (coap://192.168.02:5683/temperature) for checking indoor temperature level by clicking "Request to IoT device" button from the application, and the IoT device sends response with temperature level in JSON format such as {"temperature": "22,5"}. Figure 18 illustrates the evaluation results for smart service access to the indoor environment. We also present the evaluation results via the collection of RTT 25 times. For analyzing the smart service access to the indoor environment, the same network and hardware environment are utilized. The RTT is taken based on the difference between the sending request time and the time for receiving the last response. The RTTs are taken between a minimum of 30 ms and maximum of 61 ms, the average response time is equal to 40 ms, and the standard deviation is 7.7 ms.

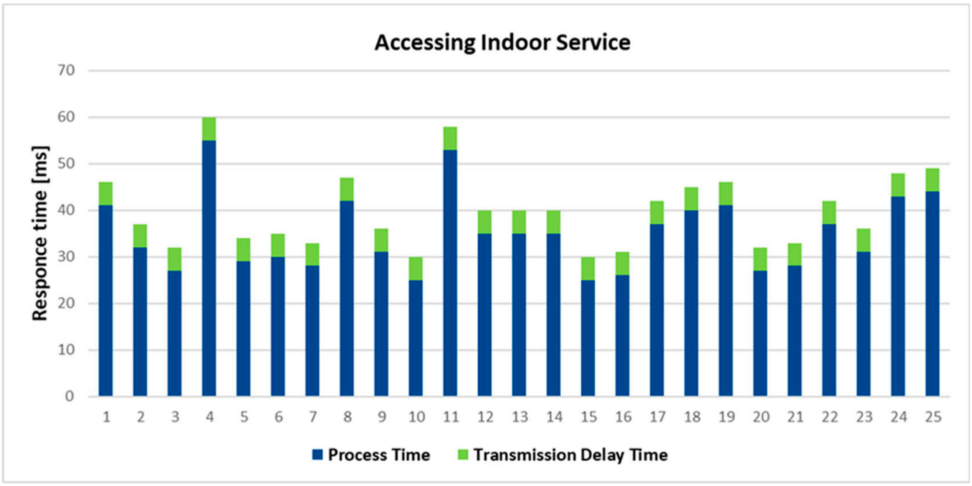


Figure 18. Round Trip Time for accessing indoor service.

For accessing indoor service evaluation performance, we ran the simulation three times according to three various user groups in the 100 ms period. As can be seen from Figure 19, as the amount of the users increase the response time also increases in the proposed system. When 200 users send a request for accessing the indoor service at the same time, minimum and maximum response time is between 61,000 ms, and it is more than 85,000 ms (in only one point), respectively. When user numbers are equal to 500, the response time also increases a little bit. However, it is difficult to make a comparison between the first and the second groups, because the system response time for the first two groups is almost the same. Only in some points, the second group is closer to the 8500 ms response time. The third group simulation results are higher than the other two groups when 800 users send the GET request for checking indoor temperature. With the CoAP network protocol, it takes more than 85,000 ms in some points. However, in some other points, response times are nearly similar to the other two groups.

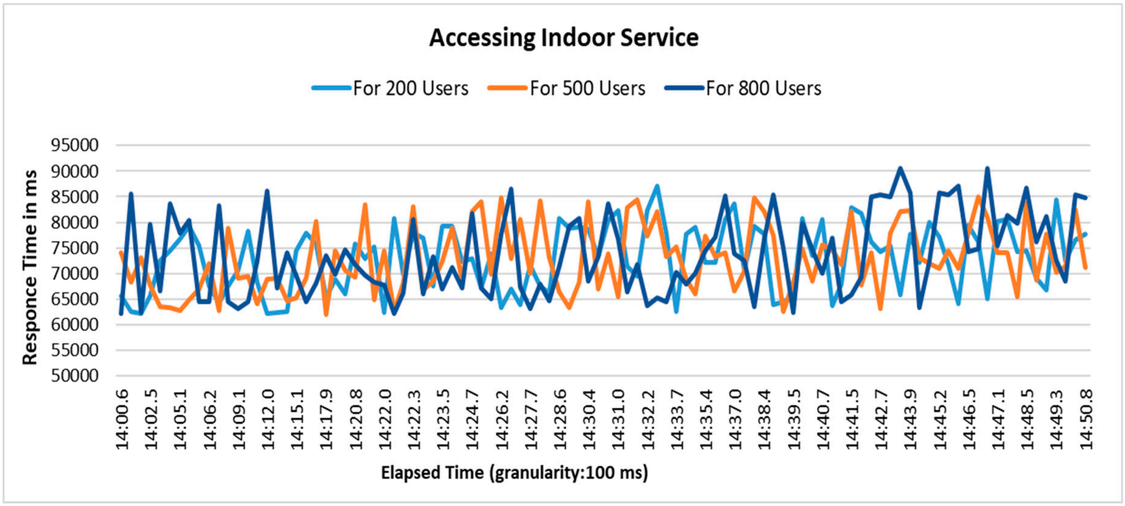


Figure 19. Accessing indoor service response time in the different simultaneous request.

6. Conclusions

We proposed the IoT resource auto-registration and smart service access for indoor services based on OCF IoTivity for IoT devices in this paper. The proposed approach can support the device auto-registration scheme which enables devices to be registered to the local IoT DNS server by itself to be discovered by IoT clients from the Internet in IP version 4 environment. For developing the proposed IoT DNS in the OCF IoTivity, the scenarios of registration, device discovery, service accessing, and the implementation results were explained in detail. With our proposed auto-registration system for Domain Names, Internet of Things devices are able to register their internet protocol addresses and URI names automatically to Domain Name System server through Open Connectivity Foundation's IoTivity framework. Furthermore, we also design the Android application which interacts in real time with clients and IoT devices. Some experiments are carried out in order to test the performance analysis of the proposed system; the interactions are monitored for the auto-registration, getting an IP address and service access. Additionally, the response time of auto-registration, IP request-response process and accessing indoor services are checked in three different groups (200, 500 and 800), in order to check the proposed system's flexibility and adaptability. In the future, we will consider and develop the proposed system's security functions such as securing the communication among users, IoT devices and local IoT DNS server with secure data transmission. Furthermore, we will consider a mechanism to access the IoT device by authorized users in a more robust and secure fashion.

Author Contributions: A.K. and D.H.K. conceived the idea, designed the overall system and wrote the paper. W.J. assisted in model design and implementation of the system. D.H.K. conceived the overall idea of the IoT DNS server, and proof-read the manuscript.

Acknowledgments: This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No.2017-0-00526, Development of Intelligent IoT System capable of cooperation and learning between things in movement-free environment), and this research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2019-2014-1-00743) supervised by the IITP (Institute for Information & communications Technology Planning & Evaluation). Any correspondence related to this paper should be addressed to DoHyeun Kim.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Wilkinson, J.S. Internet of Things (IoT). In *Communication Technology Update and Fundamentals*; Taylor&Francis: New York, NY, USA, 2018.
2. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswami, M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.* **2013**, *29*, 1645–1660. [CrossRef]
3. Mehmood, F.; Ullah, I.; Ahmad, S.; Kim, D. Object detection mechanism based on deep learning algorithm using embedded IoT devices for smart home appliances control in CoT. *J. Ambient Intell. Humaniz. Comput.* **2019**. [CrossRef]
4. Bellavista, P.; Cardone, G.; Corradi, A.; Foschini, L. Convergence of MANET and WSN in IoT Urban Scenarios. *IEEE Sens. J.* **2013**, *13*, 3558–3567. [CrossRef]
5. Jin, W.; Kim, D. Development of Virtual Resource Based IoT Proxy for Bridging Heterogeneous Web Services in IoT Networks. *Sensors* **2018**, *18*, 1721. [CrossRef] [PubMed]
6. Kaplinger, T.E.; Moore, V.S.; Nusbickel, W.L. Self-Documentation for Representational State Transfer (REST) Application Programming Interface (API). U.S. Patent 9,959,363, 1 May 2018.
7. Guinard, D.; Trifa, V.; Karnouskos, S.; Spiess, P.; Savio, D. Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. *IEEE Trans. Serv. Comput.* **2010**, *3*, 223–235. [CrossRef]
8. Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 2347–2376. [CrossRef]
9. Ahmad, S.; Hussain, I.; Fayaz, M.; Kim, D.-H. A Distributed Approach towards Improved Dissemination Protocol for Smooth Handover in MediaSense IoT Platform. *Processes* **2018**, *6*, 46. [CrossRef]

10. SCOOP. The Role of IoT Platforms in an Evolving IoT Business and Technology Context. Available online: https://www.i-scoop.eu/internet-of-things-guide/internet-things-iot-platforms/ (accessed on 5 April 2019).
11. Ahmad, S.; Hang, L.; Kim, D.H. Design and Implementation of Cloud-Centric Configuration Repository for DIY IoT Applications. *Sensors* **2018**, *18*, 474. [CrossRef] [PubMed]
12. Shelby, Z.; Bormann, C. *6LoWPAN: The Wireless Embedded Internet*; John Wiley & Sons: Hoboken, NJ, USA, 2011; Volume 43.
13. Chen, C.Y.; Chao, H.C.; Wu, T.Y.; Fan, C.I.; Chen, J.L.; Chen, Y.S.; Hsu, J.M. IoT-IMS communication platform for future internet. *Int. J. Adapt. Resil. Auton. Syst.* **2011**, *2*, 74–94. [CrossRef]
14. *OIC Core Specification V1.1.0*; Open Connectivity Foundation, Inc.: Beaverton, OR, USA, 2016. Available online: https://openconnectivity.org/specs/OIC_Core_Specification_v1.1.0.pdf (accessed on 10 January 2019).
15. Liao, M.; Yuping, D.; Yongping, D. DNS Extension for Autonomous Internet (AIP). 2017. Available online: https://tools.ietf.org/html/draft-diao-aip-dns-02 (accessed on 2 April 2019).
16. Drako, D. Policy-Managed DNS Server for to Control Network Traffic. U.S. Patent 8,447,856, 21 May 2013.
17. Leiner, B.M.; Cerf, V.G.; Clark, D.D.; Kahn, R.E.; Kleinrock, L.; Lynch, D.C.; Postel, J.; Roberts, L.G.; Wolff, S. A brief history of the Internet. *ACM SIGCOMM Comput. Commun. Rev.* **2009**, *39*, 22–31. [CrossRef]
18. Fellman, B. Internet Domain Name Registration System. U.S. Patent 6,980,990, 27 December 2005.
19. Hinden, R.; Deering, S. *IP Version 6 Addressing Architecture (No. RFC 4291)*; IGI Global: Hershey, PA, USA, 2006.
20. Dainotti, A.; Benson, K.; King, A.; Huffaker, B.; Glatz, E.; Dimitropoulos, X.; Richter, P.; Finamore, A.; Snoeren, A.C. Lost in Space: Improving Inference of IPv4 Address Space Utilization. *IEEE J. Sel. Areas Commun.* **2016**, *34*, 1862–1876. [CrossRef]
21. Narten, T.; Nordmark, E.; Simpson, W.; Soliman, H. *Neighbor Discovery for IP Version 6 (IPv6)*; IETF RFC 4861; John Wiley & Sons: Hoboken, NJ, USA, 2007.
22. Alsadhan, A.A.; Hussain, A.; Alani, M.M. Detecting NDP Distributed Denial of Service Attacks Using Machine Learning Algorithm Based on Flow-Based Representation. In Proceedings of the 2018 11th International Conference on Developments in eSystems Engineering (DeSE), Cambridge, UK, 2–5 September 2018; pp. 134–140.
23. Jeong, J.; Lee, S.J. Method for Naming DNS for IOT Device. U.S. Patent 15/745,665, 25 October 2018.
24. Jara, A.; Ladid, L.; Skarmeta, A. The internet of everything through ipv6: An analysis of challenges solutions and opportunities. *J. Wirel. Mob. Netw. Ubiquitous Comput. Dependable Appl.* **2013**, *4*, 97–118.
25. Kim, J.; Lee, J.; Kim, J.; Yun, J. M2M Service Platforms: Survey, Issues, and Enabling Technologies. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 61–76. [CrossRef]
26. Kim, J.; Jeon, H.; Lee, J. Network management framework and lifetime evaluation method for wireless sensor networks. *Integr. Comput. Aided Eng.* **2012**, *19*, 165–178. [CrossRef]
27. Gorbach, G. IoT Standards Get a Big Push: Meet the Open Connectivity Foundation (OCF). 23 February 2016. Available online: https://www.arcweb.com/blog/iot-standards-get-big-push-meet-open-connectivity-foundation-ocf (accessed on 14 January 2019).
28. Yang, X.; Moore, P.; Chong, S.K. Intelligent products: From lifecycle data acquisition to enabling product-related services. *Comput. Ind.* **2009**, *60*, 184–194. [CrossRef]
29. Laranjo, I.; Macedo, J.; Santos, A. Internet of Things for Medication Control: Service Implementation and Testing. *Procedia Technol.* **2012**, *5*, 777–786. [CrossRef]
30. Boyd, S. Gigaom—New Open Connectivity Foundation combines Open Interconnect Consortium and AllSeen Alliance. 26 February 2016. Available online: https://gigaom.com/2016/02/20/new-open-connectivity-foundation-combines-open-interconnect-consortium-and-allseen-alliance/ (accessed on 14 January 2019).
31. Sawers, P. *Microsoft, Intel, Samsung, & Others Launch IoT Standards Group: Open Connectivity Foundation*; VentureBeat: San Francisco, CA, USA, 2015.
32. IoTivity Project. Available online: https://www.iotivity.org (accessed on 16 January 2019).
33. Park, S. OCF: A New Open IoT Consortium. In Proceedings of the 31st International Conference on IEEE Advanced Information Networking and Applications Workshops (WAINA), Taipei, Taiwan, 27–29 March 2017.
34. Bouhaddi, M.; Radjef, M.S.; Adi, K. An efficient intrusion detection in resource-constrained mobile ad-hoc networks. *Comput. Secur.* **2018**, *76*, 156–177. [CrossRef]
35. Loo, J.; Mauri, J.L.; Ortiz, J.H. *Mobile Ad Hoc Networks. Mobile Ad Hoc Networks: Current Status and Future Trends*; CRC Press: Boca Raton, FL, USA, 2012.

36. Lee, K.; Kim, S.; Jeong, J.; Lee, S.; Kim, H.; Park, J.-S. A framework for DNS naming services for Internet-of-Things devices. *Future Gener. Comput. Syst.* **2019**, *92*, 617–627. [CrossRef]
37. Yanase, T.; Tanaka, H.; Suzuki, H. Flexible Name Autoconfiguration for IoT Devices. In Proceedings of the 2018 Eleventh International Conference on Mobile Computing and Ubiquitous Network (ICMU), Auckland, New Zealand, 5–8 October 2018; pp. 1–6.
38. Singh, V.P.; Dwarakanath, V.T.; Haribabu, P.; Babu, N.C. IoT standardization efforts—An analysis. In Proceedings of the 2017 International Conference on Smart Technologies for Smart Nation (SmartTechCon), Bengaluru, India, 17–19 August 2017; pp. 1083–1088.
39. Shelby, Z.; Sensinode; Hartke, K. Constrained Application Protocol (CoAP). draft-ietf-core-coap-18. 28 June 2013. Available online: http://tools.ietf.org/html/draft-ietf-core-coap-18 (accessed on 20 January 2019).
40. Muhonen, T. *Standardization of Industrial Internet and IoT (IoT–Internet of Things)–Perspective on Condition-Based Maintenance*; University of Oulu: Oulu, Finland, 2015.
41. IoTivity.org. Available online: https://iotivity.org/documentation/architecture-overview (accessed on 23 January 2019).
42. Open Connectivity Foundation. Available online: https://openconnectivity.org/ (accessed on 30 January 2019).
43. Ahmad, S.; Mehmood, F.; Mehmood, A.; Kim, D. Design and Implementation of Decoupled IoT Application Store: A Novel Prototype for Virtual Objects Sharing and Discovery. *Electronics* **2019**, *8*, 285. [CrossRef]
44. Bormann, C.; Castellani, A.; Shelby, X. CoAP: An Application Protocol for Billions of Tiny Internet Nodes. *IEEE Internet Comput.* **2012**, *16*, 62–67. [CrossRef]
45. Shelby, Z.; Hartke, K.; Bormann, C. *The Constrained Application Protocol (CoAP)*; Internet Engineering Task Force (IETF) RFC-7252: Fremont, CA, USA, 2014.