OCF Bridging Techniques for UWB/LoRa IoT Ecosystems

HYUN SEOB OH 1, WHA SOOK JEON 2, (Senior Member, IEEE), AND DONG GEUN JEONG 3, (Senior Member, IEEE)
1Samsung Research, Samsung Electronics, Samsung Seoul R&D Campus, Seoul 06765, South Korea
2Department of Computer Science and Engineering, Seoul National University, Seoul 08826, South Korea
3Department of Electronics Engineering and Applied Communication Research Center, Hankuk University of Foreign Studies, Yongin-si 17035, South Korea
Corresponding author: Wha Sook Jeon (e-mail: wsjeon@snu.ac.kr)
This work was supported by the National Research Foundation of Korea (NRF) Grant funded by the Ministry of Science and ICT (MSIT) of Korea Government under Grant 2020R1F1A1075882 and 2021R1A2C1095552.

ABSTRACT
The Open Connectivity Foundation (OCF) is one of the most powerful IoT standard organizations. The OCF has developed not only its own standard but also an open source software framework called IoTivity for efficient implementation of IoT systems. The OCF has made great efforts to develop the bridging techniques for supporting inter-operability with existing non-OCF IoT systems. In this paper, we develop the OCF-to-UWB (O2U) and OCF-to-LoRa (O2L) bridging techniques that can promote the release of diverse services related to UWB-based short-range and LoRaWAN-based long-range communications (e.g., indoor localization, sensor-based metering). We also suggest a simple and cost-effective bridging technique between different non-OCF IoT systems, which complies with the OCF standard. Our O2U and O2L bridging implementation is checked for consistency with the OCF standard using the OCF conformance test tool. Three example use-cases are implemented for demonstrating the usability of the developed O2U and O2L bridges, which are proximity-based Zigbee device control combined with UWB, LoRa-based metering, and LoRaWAN as backhaul of UWB system. Our work can provide a useful guideline to the development of new bridging techniques for other non-OCF IoT systems.

INDEX TERMS
OCF-to-UWB (O2U) bridging, OCF-to-LoRa (O2L) bridging, OCF-based bridging between non-OCF IoT systems, IoTivity-based implementation

I. INTRODUCTION

The Internet of Things (IoT) has been greatly changing the way people live, as a unified concept including technologies and services. In particular, as the coronavirus disease 2019 (COVID-19) pandemic dramatically shrinks face-to-face contacts and social activities among individuals, the empty space from the reduced interactions is being filled with diverse “untact” activities supported by IoT-related technologies and services. According to [1], the whole number of IoT devices is expected to increase up to 80 billion by 2025, which was the prediction under the normal situation of no COVID-19. However, due to the “untact” trend increased by COVID-19, this growing speed can be further accelerated. Therefore, the way of effectively managing the explosively increasing IoT devices becomes more and more important.

The fast growing IoT industry has promoted the development of IoT standards by global vendors. Several IoT standard organizations including Open Connectivity Foundation (OCF) [2], AllSeen Alliance [3], OneM2M [4], Connectivity Standards Alliance (CSA) [5], and Matter [6] were established. Among them, the OCF, which was organized in Sep. 2014, has become the most influential standard organization with more than 500 members. In the Consumer Electronics Show (CES) 2020 at Las Vegas, the commercialization of the OCF standard as an international IoT standard was officially announced by global vendors [7].

The OCF aims at providing easy and secure communication among IoT devices irrespective of their operating systems, service providers, transport technologies, etc. Accordingly, the OCF releases not only its standard specification but also bridging specification for other IoT systems and a certification program. The OCF also sponsors the development of IoTivity [8] which is an open source software framework for implementing IoT services/products according to the OCF standard. The IoTivity can be mounted on small devices which are operated by several dominant operating systems.
such as Linux, Windows, Android or iOS. This may enable the cost-effective development of various IoT products and services in a short time.

One the other hand, the OCF was very interested from the beginning in developing a bridging standard toward each of influential non-OCF IoT ecosystems[1], because a lot of non-OCF IoT devices were widely deployed already in the IoT market. The overall bridging concept was included in the OCF specification 1.0. The OCF bridging specifications for AllJoyn, OneM2M, and Zigbee IoT ecosystems were published in June 2017, Apr. 2019, and June 2019, respectively. The OCF version 2.1.1 (released in Feb. 2020) included the bridging specifications for Bluetooth low energy (BLE) [9], Z-Wave [10], Uplus [11], and EnOcean [12] at once.

Recently, the ultra wideband (UWB) technology has attracted great attention from global vendors and IoT service developers, because of its highly precise and secure ranging capability. Several chipset and device vendors (e.g., Samsung, NXP, Qorvo, Apple) have participated in IEEE 802.15.4z standardization [13] and organized the FiRa (which stands for “Fine Ranging”) consortium [14] to promote diverse IoT services using UWB technology. The IEEE 802.15.4z-based UWB ranging will be broadly utilized in various use-cases, for example, human/asset tracking systems in factory/office environment and proximity-based services. Accordingly, when considering the potential usability of UWB, the OCF-to-UWB bridging technique is needed but is not developed yet.

Long-range wide area network (LoRaWAN) [15] has also been extensively utilized, due to its feature enabling a large scale deployment of IoT devices over wide area. As an open standard, the LoRaWAN specification 1.1 was released in Oct. 2017 by the LoRa Alliance. The LoRaWAN can be deployed with low cost and is capable of long-range communication up to a few kilometers while consuming a very low power. Owing to such attractive features, according to the LoRa Alliance report in Dec. 2019 [16], there are 143 countries with LoRaWAN deployments, 133 LoRaWAN network operators, and more than 80 million connected devices. When considering the widespread of LoRaWAN-based IoT systems, the bridging between OCF and LoRaWAN is surely an important and urgent work.

On the basis of the above discussion (i.e., the position of OCF as an international IoT standard, the high usability of UWB and LoRaWAN), in this paper, we concentrate on developing the OCF-to-UWB (O2U) and OCF-to-LoRaWAN (O2L) bridging techniques. These techniques enable an OCF client device to control a remote UWB or LoRa device without its direct connectivity to the UWB or LoRa device. In addition, since UWB and LoRaWAN have their intrinsic strengths for IoT services in short communication range and long communication range, respectively, they can complement each other without competition. In particular, if both technologies are used together in a single device, the device can provide more diverse and suitable IoT services without restriction of communication range.

On the other hand, it is obvious that the bridging between different non-OCF IoT systems is also very important. Under the situation that there already exist a lot of non-OCF IoT systems, note that developing a separate bridging technique for a pair of any non-OCF systems can be very inefficient. As a simple and cost-effective solution, we suggest a way to utilize the OCF bridges for “switching” between two non-OCF IoT systems. For example, when implementing the bridging between two non-OCF systems A and B, we can use an OCF-to-A bridge and an OCF-to-B bridge together. The suggested bridging technique combines the OCF-to-A bridge and the OCF-to-B bridge, by letting these two bridges share one OCF-client. Then, this common OCF-client plays a key role in realizing a A-to-B bridge. Through this approach, the bridging between non-OCF IoT systems can be easily realized.

In this paper, we design the O2U and O2L bridging techniques and describe their IoTivity-based implementation details, which can give useful guidelines to comprehending OCF bridging techniques and to developing a new bridging technique for other non-OCF IoT systems. By using the developed O2U/O2L bridges, various IoT services of the combined UWB/LoRaWAN systems can be easily implemented. Also, to support the cost-effective bridging between non-OCF IoT systems, we suggest and implement an efficient OCF-based bridging technique following the OCF standard. The contribution of this paper can be summarized as follows:

1) Development of O2U and O2L bridging techniques
2) Development of a simple and cost-effective bridging technique between non-OCF IoT systems
3) Implementation of potential use-cases using the developed bridging techniques

The remainder of this paper is organized as follows. In Section II, as the background, the brief overview of UWB and LoRaWAN technologies and two potential use-cases of utilizing UWB and LoRaWAN are shortly described. In Section III, we develop the O2U and O2L bridging techniques and suggest the OCF-based bridging method between non-OCF IoT systems. Section IV introduces IoTivity and OCF bridging security features, as a preliminary knowledge of implementation. The IoTivity-based implementation details of O2U and O2L bridges are presented in Section V and three example use-cases are implemented in Section VI. Finally, the paper is concluded with Section VII.

II. BACKGROUND

A. IEEE 802.15.4Z UWB RANGING

The UWB exploits a wide bandwidth over 500 MHz on the frequency bands from 3.1 to 10.6 GHz, and inherently has the beneficial frequency characteristics for precise ranging, such as the robustness against interference and the propagation straightness. For this reason, time-of-flight (ToF) can be

---

1 In this paper, we use the words “IoT ecosystem” and “IoT system” interchangeably.
measured with a high resolution. This fine ranging capability of UWB enables a variety of applications utilizing a highly precise spatial awareness and indoor localization.

The UWB ranging technique was recently standardized as IEEE 802.15.4z [13]. Note that there exist already several standards related to IEEE 802.15.4z (i.e., IEEE 802.15.4a, IEEE 802.15.4f, and IEEE 802.15.4-2015 standards [17]–[19]). The goal of IEEE 802.15.4z standardization is to introduce new capabilities using UWB into the physical (PHY) and medium access control (MAC) layers of these existing 802.15.4 standards. As the PHY enhancement, a scrambled timestamp sequence [14], [20]. In addition, for a highly accurate and efficient ranging, several features to support various ranging scenarios composed of multiple devices are updated as the MAC layer enhancements of IEEE 802.15.4z. That is, the procedures for one-to-many and many-to-many ranging among UWB devices are supported. Also, there are three ranging modes for measuring the ToF between two devices, which are one-way ranging (OWR), single-side two-way ranging (SS-TWR), and double-side two-way ranging (DR-TWR). Refer to [13] for details of UWB standard.

**B. LORAWAN TECHNOLOGY**

The LoRaWAN consists of three entities: a network server, gateways, and end nodes. The messages between end nodes and the network server are usually relayed by gateways. There are three MAC classes of one baseline class (class A) and two optional classes (classes B and C). Three classes are different from each other, merely in the way how the end nodes open their downlink receive windows for receiving acknowledgment or data traffic from the server. The class A nodes open two receive windows only immediately after transmitting a data frame. In class B, the nodes have additional receive windows synchronized with a beacon transmitted by the gateway. In class C, the nodes continuously stay in the reception mode. For the uplink transmission, an end node without respect to its class can transmit a frame to the network server via gateways at any time, according to pure-ALOHA protocol.

In the PHY layer, the physical channels having bandwidth of 125, 250, or 500 kHz on the sub-GHz ISM band are used. The LoRaWAN adopts a proprietary modulation, based on chirp spread spectrum (CSS) exploiting pulses whose frequency increases or decreases linearly over a certain amount of time. The CSS modulation is known as having the robust characteristic against channel noise and multipath. For this reason, the communication range of LoRaWAN is long enough to cover several kilometers. Also, when the gateways are time-synchronized to each other, the location of an end node can be quite accurately calculated, using a time difference of arrival (TDoA) technique [21], [22]. That is, the LoRaWAN has a strength in device localization. The details of LoRaWAN standard are explained in [13] and [23]. Hereafter, we use “LoRa” shortly instead of “LoRaWAN,” if there is no confusion.

**C. SYNERGY OF UWB/LORA INTEGRATION: POTENTIAL USE-CASES**

Let us consider the following two potential use-cases which demonstrate the merits of using both UWB and LoRa technologies together.

1) **UWB/LoRa-Based Indoor Localization**

In the UWB-based localization system, there exist four entities in general: anchor node, tag node, gateway node, and monitoring server. The anchor node is an infrastructure node of which the location is already known. The tag node is a target node which is attached to an object or possessed by a human for the purpose of location tracking/monitoring. The gateway node plays a role of proxy server to support the connection between the UWB anchor/tag nodes and the monitoring server. The tag node calculates its location by measuring the ToF from itself to each of nearby anchor nodes. Then, the location information acquired at the tag node is forwarded to the monitoring server through the anchor and gateway nodes.²

In a small scale network of home/office environments, the communication range of a single UWB gateway can sufficiently cover a few of anchor nodes for providing the location or proximity-based device control services (e.g., light, door). However, to support a considerable number of UWB anchor nodes deployed inside the huge building, several UWB gateways need to be deployed together. Then, the distances between some UWB gateways and the monitoring server may be much longer than the feasible communication range of UWB. As a cost-effective solution of this long distance problem, the LoRa technology can be jointly utilized, where the LoRaWAN provides the long-distance wireless backhaul link up to LoRa gateway connected to the monitoring server, as depicted in Fig. 1. Note that, in Fig. 1, the UWB gateway

²It is also possible that a remote monitoring server calculates the position of the tag node using the ToF values reported by the tag.

**FIGURE 1.** UWB/LoRa-based IoT System.
with LoRa module functions as not only a UWB gateway but also a LoRa end node.

As another promising use-case, both UWB and LoRa technologies may be used together in a single IoT device having several sensors, when not only the measurement data from the sensors within the device but also the localization information of the device need to be reported to a far-away server.

2) UWB/LoRa-Aided UAV for Logistics Services
The management system of unmanned aerial vehicles (UAVs) for logistics services is another promising use-case utilizing the UWB/LoRa technologies. This concept can provide prompt logistics services to customers [24], where an airborne fulfillment center suspended in midair covers a large number of customers within a range of several kilometers. A novel and prompt commodity delivery can be realized by using the big data analytics, since potential items can be supplied to the airborne fulfillment center in advance before actual orders. When an actual order of an online customer is received, a GPS-equipped UAV may deliver the ordered item to the customer within few minutes.

By delivering items to the front door of the customer (in the apartment or office building), further customer-friendly logistics services are possible but this concept additionally requires a precise indoor localization. In the near future, to support this kind of advanced logistics services based on an accurate indoor localization, the UWB anchor nodes may be deployed in the indoor environment. Then, UWB and LoRa technologies can be utilized together in a GPS-equipped UAV for logistics services, where the UWB and LoRa support the indoor localization and long-range wireless backhaul, respectively. Through the LoRa communication between each UAV and the remote airborne fulfillment center, various control messages (e.g., delivery completion or error, return position using GPS) can be effectively exchanged.

III. O2U AND O2L BRIDGING
In this section, we outline the basic concept of OCF bridging described in [26], and then discuss the details of O2U and O2L bridging. Also, a simple approach for supporting the bridging between non-OCF IoT ecosystems is suggested.

A. BASIC CONCEPT OF OCF BRIDGING
The OCF basically prefers to an asymmetric server bridging type exposing non-OCF servers to OCF clients, although the other two types (asymmetric client bridging and symmetric bridging) are additionally defined in [26]. Fig. 2 shows a logical structure of asymmetric server bridging, where an OCF client controls non-OCF servers. Note that most of the OCF bridging developed so far is asymmetric server bridging. In this paper, we also develop O2U and O2L bridges of asymmetric server bridging type.

To realize an asymmetric server bridge, the counterparts of real OCF client and real non-OCF server need to be virtualized. Through the virtualization, a real OCF client and a real non-OCF server can communicate with their respective counterparts within the bridge, called a virtual OCF server and a virtual non-OCF client (see Fig. 2). A virtual OCF server is referred to as a virtual OCF device (VOD).

An OCF device is composed of a collection of resources and each resource can possess multiple properties indicating physical or logical states. Accordingly, for bridging between OCF and non-OCF, the main features of non-OCF devices should be mapped to the corresponding OCF resources and properties. If there are no appropriate OCF resources, new resources and properties for representing the non-OCF features should be defined. That is, the resources/properties and the mapping rules are core elements of OCF bridging toward non-OCF ecosystems (non-OCF bridging functions in Fig. 2). Let us examine the mapping rules and resources/properties for O2U and O2L bridges.

B. MAPPING RULES BETWEEN OCF AND UWB/LORA DEVICES
Since the UWB and LoRaWAN specifications do not define application profiles, there is no pre-defined mapping rule between OCF and UWB/LoRa. Thus, we need to define new OCF resources for UWB and LoRa devices and the mapping rules between them. We will design the bridging essential features, focusing on the representative use-cases of UWB and LoRa illustrated in Section II-C, i.e., localization and sensor-based metering. For other promising use-cases of UWB and LoRa, if necessary, the corresponding resources and mapping rules can be added.

Table 1 summarizes the OCF resource type mapping from UWB and LoRa devices to OCF devices. Since there is no device type representing UWB and LoRa devices in the OCF resource type specification [27], we newly define two device types: ‘oic.d.uwb’ (UWB device) and ‘oic.d.lora’ (LoRa device). Note that the OIC is the past name of the OCF and was already renamed to the OCF, but the ‘oic’ notation is still used in the OCF specifications. Hereafter, to avoid inconsistency with the OCF specification [2], we follow the ‘oic’ notation.
TABLE 1. OCF resource type mapping from UWB or LoRa device

| From UWB (U) or LoRa (L) | To OCF                                                                 |
|--------------------------|------------------------------------------------------------------------|
| (U) Anchor, Tag node     | oic.d.uwb, oic.r.uwb.tag_position, oic.r.uwb.anchor_position,         |
|                           | oic.r.uwb.mac_config, oic.r.uwb.phy_config                              |
| (L) Gateway node         | oic.d.lora, oic.r.lora.gateway                                          |
| (L) End node (having sensor(s)) | oic.d.lora, oic.r.lora.endnode, {oic.r.temperature, oic.r.sensor.atmosphericpressure, oic.r.humidity, oic.r.sensor.acceleration} |

1) Resources for UWB-based Localization

It is obvious that the indoor ranging/localization is a representative use-case of UWB. Note that various types of OCF resources have been well defined in [27] and, in particular, the sensing-related OCF resources can be applied to the O2U and O2L bridging as they are. However, the resources for ranging and localization need to be newly defined.

For the position of a UWB device, we define two resource types ‘oic.r.uwb.tag_position’ (tag node) and ‘oic.r.uwb.anchor_position’ (anchor node). Each of them has a mandatory property which is the \((x, y, z)\) coordinate indicating the current location of the corresponding UWB device. The current location of an anchor node is the position where the node is installed, whereas that of a tag node is the current position of the target object/human equipped with the tag. On the other hand, since a UWB gateway is merely a device for relaying the traffic between anchor/tag nodes and the monitoring server and has no role in ranging or localization, a position of UWB gateway is not treated as a resource.

Since many MAC and PHY parameters related to ranging/localization have been specified in [13], two resource types for MAC and PHY configuration, ‘oic.r.uwb.mac_config’ and ‘oic.r.uwb.phy_config,’ are also newly defined. For the resource type ‘oic.r.uwb.mac_config,’ multiple mandatory properties for indicating the ranging configuration of MAC layer can be assigned (e.g., ‘device type,’ ‘device role,’ ‘multi-node mode,’ ‘ranging method’ properties). The ‘device type’ property presents whether the device is a controller or controllable, and the ‘device role’ property indicates whether the device participates in ranging as an initiator or a responder. The ‘multi-node mode’ property presents whether a ranging operation between devices is one-to-one, one-to-many, or many-to-many; the ‘ranging method’ property indicates the ranging technique, i.e., SS-TWR, DS-TWR, OWR or TDoA. In addition to these properties, several other MAC parameters, if necessary, can be assigned as mandatory properties to the resource type ‘oic.r.uwb.mac_config.’

Similarly, we can define the mandatory properties for the resource type ‘oic.r.uwb.phy_config,’ based on the UWB PHY parameters. Note that the UWB PHY layer has many parameters including PHY frame format (e.g., preamble length, preamble code index, pulse repetition frequency) and calibration (e.g., antenna delay, frequency offset).

Among these MAC/PHY configuration parameters, the parameters expected to be accessed by an OCF client, for IoT application services or specific purposes, need to be defined as mandatory properties of the ‘oic.r.uwb.mac_config’ and ‘oic.r.uwb.phy_config’ resources. Then, the values of the defined mandatory properties might be set, based on the corresponding chipset data sheet.

Remark: In implementation of O2U bridge (as described in Section V), we used the DWM1001 module of Qorvo Corp. [28] as UWB anchor or tag. Note that this module does not provide a firmware application programming interface (API) that enables easy access to any MAC/PHY configuration parameters. By considering that this paper focuses on developing the O2U bridging technique (not the UWB ranging/localization technique itself), although we define the resources ‘oic.r.uwb.mac_config’ and ‘oic.r.uwb.phy_config’ for generality, the OCF client accesses only the ‘oic.r.uwb.tag_position’ and ‘oic.r.uwb.anchor_position’ resources in our implemented O2U bridge.

2) Resources for Localization and Metering Service of LoRa

For a LoRa gateway, the resource type ‘oic.r.lora.gateway’ is defined, which includes three mandatory properties: a list of connected LoRa end nodes, the current location \((x, y, z)\) of the gateway, and the time sync indicator representing whether the LoRa gateways are time-synchronized to each other. The resource type ‘oic.r.lora.endnode’ for a LoRa end node includes the current location of the node as a mandatory property. Note that the current position of gateway can be updated by using the GPS equipped to the gateway.\(^5\) The LoRa end nodes may also be equipped with GPS. Although a LoRa end node is not equipped with GPS, its location can be calculated by using the estimated distances from the end node to adjacent gateways.

As another use-case of LoRa, we consider the information reporting or metering using various sensors. For a LoRa end node with sensor(s), the notation \(\{\cdot\}\) is additionally used in Table 1. When a Lora end node is equipped with sensor(s), the device type ‘oic.d.sensor’ can be optionally included together with the device type ‘oic.d.sensor.’ According to the equipped sensor type, the OCF resource type(s) corresponding to the sensor should be added as the resource of the end node. For example, a LoRa end node with a tem-

\(^5\) The location of a gateway may be essential for the use-case of UAV-based logistics services, when the gateway is deployed at the return position of the UAVs.
perature sensor and a humidity sensor has three resources: ‘oic.r.lora.endnode,’ ‘oic.r.temperature,’ and ‘oic.r.humidity.’ The details of mandatory properties of each resource type on sensors are explained in [27]. Note that Table 1 merely illustrates four resource types, related with temperature, humidity, atmospheric pressure, and acceleration sensors. When other sensor types need to be added, we can additionally define the corresponding resource types.

**C. BRIDGING BETWEEN NON-OCF IOT ECOSYSTEMS**

The cost-effective and sustainable solution for bridging between different non-OCF IoT ecosystems is to exploit the OCF bridges which were developed already for these non-OCF ecosystems. This approach is very effective since (1) the influence of the OCF standard has been greatly extending; (2) the OCF bridging for a variety of non-OCF IoT ecosystems has been actively developing; (3) the development maturity of IoTivity has been rapidly increasing.

While considering that most of the OCF bridging developed so far is asymmetric server bridging, we suggest a simple but efficient way to realize the bridging between non-OCF ecosystems using the OCF bridges of asymmetric server type. Fig. 3 illustrates a logical structure of the proposed bridging technique between two non-OCF IoT ecosystems, where an asymmetric OCF to non-OCF (A) bridge and an asymmetric OCF to non-OCF (B) bridge are combined by an OCF client-based agent. The logical structure in Fig. 3 seems complex, but two OCF to non-OCF bridges have the same structure as that in Fig. 2 and they are simply merged through a common OCF client. As expected, the main advantage of the structure is that each OCF to non-OCF bridge can be reused for bridging to each other. This can prevent developing a separate bridging technique for a pair of any non-OCF systems that may cause huge cost and efforts. For example, we can implement the UWB-to-LoRa (U2L), UWB-to-Zigbee (U2Z), and LoRa-to-Zigbee (L2Z) bridges, using the O2U, O2L, and O2Z bridges without any modification. Since the agent also uses the OCF protocol, these bridges between non-OCF ecosystems operate complying with the OCF standard. In Section VI, we will present an application example for demonstrating the usefulness of the suggested bridging approach.

**IV. PRELIMINARY FOR OCF BRIDGE IMPLEMENTATION**

The IoTivity is an open source software framework for implementing IoT applications/products according to the OCF standard, and it enables efficient implementation of OCF bridging functions. Accordingly, we develop the O2U and O2L bridges using the OCF core API of IoTivity, while obeying the bridging-related security regulations of OCF. For the readers who are not familiar with OCF bridging implementation, we briefly summarize the security features and IoTivity API for OCF bridging device, described in our previous work [29]. Refer to [8] and [30] for the details of IoTivity and OCF security, respectively. The readers already familiar with these subjects can skip this section.

**A. SECURITY FEATURES IN BRIDGING FUNCTIONS**

According to [30], in order to access the resources of a VOD within the bridge, an OCF client should acquire firstly the ownership of the bridge and then the ownership of the VOD. Since a VOD or an OCF bridge is a kind of OCF device, the OCF client gets the ownership of bridge or VOD, according to the ownership acquisition (provisioning) procedure of OCF. The ownership status of VOD is expressed as ‘owned’ or ‘unowned’ for the corresponding client, depending on whether the provisioning between the client and VOD has been completed or not. In general, since an OCF server can be owned by multiple OCF clients, the server manages the access control list to store the status of each client that completed provisioning with itself.

On the other hand, the OCF bridging standard defines a new resource called VOD list, in order to efficiently handle the VODs within a bridge in a secure way. The VOD list resource includes the summarized information of VODs in the owned state. Through this resource, only the OCF clients that possess the ownership of bridge can securely get or renew the information of their owned VODs. Also, the OCF client can easily obtain the whole information of its owned VODs at once.

The O2U and O2L bridges should also be implemented to comply with the OCF security rules, mentioned above.

**B. BRIDGING PLUGIN FUNCTIONS OF IOTIVITY**

Fig. 4 depicts the IoTivity architecture for an OCF bridge device. In principle, the IoTivity prefers to a plugin-based
implementation, where a plugin is a software component used to add a particular feature to the existing software. Thus, developing an OCF bridge for a non-OCF IoT ecosystem is to implement and manage a bridging plugin for the corresponding non-OCF ecosystem.

Let us examine the bridging plugin, which is composed of seven basic functions: pluginCreate, pluginStart, pluginScan, pluginAdd, pluginRemove, pluginReconnect, and pluginDestroy. Fig. 5 illustrates execution sequences of the above functions. The pluginCreate generates a plugin instance and a virtual bridge which is a virtualization of the bridge. The created plugin instance is carried out by the pluginStart function. The pluginStart also generates a VOD list resource for the virtual bridge. The pluginAdd creates a VOD and metadata for each non-OCF device discovered by the pluginScan. Whereas, the pluginRemove removes the VOD and its metadata generated by the pluginAdd. The information of a VOD is inserted into a VOD list resource by using the pluginAdd and is deleted from the list by the pluginRemove function. The unloaded bridging plugin can be afterward reloaded by the pluginReconnect. Then, the removed VODs are automatically re-created using their previous metadata. The pluginDestroy eliminates the plugin instance and the VOD list resource.

These plugin functions are managed by a protocol plugin manager (PPM) which is a management software for controlling several plugins (see Fig. [29]).

V. IMPLEMENTATION OF O2U AND O2L BRIDGES

In this section, we describe the IoTivity-based implementation details of the O2U/O2L bridges and the OCF client-based agent for bridging between non-OCF IoT systems.

A. IMPLEMENTATION ISSUES

As mentioned in Section IV-B, an OCF bridge is developed by implementing the bridging plugin functions and a software for managing them (i.e., PPM). Since the PPM was implemented already in IoTivity [8] using a library ‘mini plugin manager,’ we concentrated on implementing the plugin functions for O2U and O2L bridges. In addition to the plugin functions, two modules were also implemented: the VOD list manager module for handling the list of authenticated VODs, and the communication module between each VOD and a virtual UWB or LoRa client. In other words, the blue boxes in Fig. 6 is our implementation scope.

In implementing the O2U and O2L bridges, we took the following four issues into account. The first two issues are actually related to IoTivity-based implementation of bridging plugin (irrespective of the corresponding non-OCF ecosystem), whereas the last two issues are related to communication with a virtual UWB/LoRa client and a client agent-based bridging between UWB and Zigbee, respectively. Accordingly, note that the issues 1) and 2) were already described in our previous work for O2Z bridge [29].

1) IoTivity Instance per VOD

According to the current IoTivity framework [8], IoTivity instances can be created once at a time and each instance has its unique device ID and security-related resources. Note that, to support multiple devices concurrently, the OCF bridge has to

FIGURE 4. IoTivity architecture for OCF bridge device [29].
create multiple VODs corresponding to these devices. Since a VOD, like a real device, also needs a unique device ID and resources for security, the bridging plugin should be able to create a separate IoTivity instance per VOD. To do this, we implemented the pluginAdd function which generates multiple instances, by applying a fork()-based process creation method under Linux environment. Whenever the pluginAdd function calls the fork(), a new process is created and an IoTivity instance for each VOD is generated in the process. Then, the corresponding VOD (i.e., its IoTivity instance) is identified with its unique device ID and process ID.

2) VOD List Manager
The VOD list manager is an entity for efficiently managing the VODs having the owned state. Since the VOD list manager is responsible for maintaining the VOD list resource in the latest state, it should be able to recognize the change in VOD ownership. In order to know the ownership status of a VOD, the VOD list manager needs to communicate with the VOD. Note that the VOD list manager is an entity within the virtual bridge. Since the virtual bridge and the VOD respectively exist in a separate process, the ownership status information can be exchanged between two processes, by using the inter-process communication (IPC) based on a shared memory. Through this IPC-based information sharing method, the implemented VOD list manager could easily know the change in VOD ownership, and the VOD list resource was correctly maintained using the latest ownership information of VODs.

3) Module for Communication with Virtual Client
As shown in Fig. 3, the OCF bridge consists of a bridging plugin instance, virtual OCF servers, and a virtual non-OCF client. In implementation, all of these three entities may be placed in a single device but it is also feasible that the plugin instance and virtual OCF servers are implemented in a bridge device and the virtual non-OCF client is deployed in another device.

For example, let us consider a typical LoRaWAN deployment that LoRa end nodes are connected to the monitoring server through a LoRa gateway. Then, it is possible that all bridging entities are implemented in the LoRa gateway or the LoRaWAN monitoring server. As another implementation case, it can be considered that the virtual LoRa client is in the gateway, and the bridging plugin instance and virtual OCF servers are in the LoRaWAN server that functions as an O2L bridge.

For various feasible implementation cases, the bridging plugin instance should be able to find a real non-OCF server, by executing the pluginScan to the corresponding virtual non-OCF client. Also, a virtual OCF server in the bridge should be able to send/receive information to/from its virtual non-OCF client. Thus, irrespective of whether a virtual non-OCF client is installed in the bridge or in another device, the virtual non-OCF client has to communicate with the corresponding entity within bridge.

We implemented a virtual LoRa client using Python code, whereas implementation of a virtual UWB client was based on the software supported by Qorvo Corp. [28]. Each virtual client has two communication modules: one is its own communication protocol stack (i.e., UWB or LoRa) for accessing a real non-OCF server; the other is the communication module for forwarding the information to a virtual OCF server or plugin instance. The first module can be implemented by using the firmware API of chip vendor and the second module can be developed using IoT communication protocol for IP network. As shown in Fig. 3, we implemented the communication module between the OCF bridge and its virtual non-OCF client, by using mqtt library. Then, the virtual non-OCF client becomes a mqtt broker and, the bridging plugin instance and VODs are the mqtt clients.

Let us examine how the mqtt module operates, with a simple example. When the pluginScan function is executed, a mqtt client in the plugin instance retrieves the device information published from the virtual non-OCF client. Also, since a mqtt broker in the virtual non-OCF client publishes the status of the topic (e.g., coordinates of UWB tag, sensor information of LoRa end node), a mqtt client in the virtual OCF server that subscribes the designated topic(s) obtains the updated status of non-OCF server. For the details of mqtt protocol and libraries, please refer to [31] and [32].

4) OCF Client-Based Agent
An OCF client-based agent plays a key role in realizing the bridging between two non-OCF ecosystems. As depicted in Fig. 3, the agent relays the information or operations between virtual OCF servers of both bridges, by accessing the virtual OCF servers through the OCF protocol. We implemented the agent by modifying the OCF client code provided by IoTivity, such as ‘occlient.cpp.’

Note that an OCF client can get or update the properties included in the resources of OCF servers, by using the Retrieve, Update, and Notify operations defined in the OCF protocol. The IoTivity provides Get, Put, and Observe functions, corresponding to Retrieve, Update, and Notify, respectively. The Get retrieves the properties of a resource at once, whereas the Observe is used to periodically report the value of a property or notify changes in property value. And, the Put is the function to update the value of a property.

The OCF client-based agent obtains the resource information of virtual OCF server in the first bridge, through the Get or Observe function. Then, the agent can renew the properties of the other virtual OCF server in the second bridge by forwarding the obtained resource information through the Put function. With this simple way, an OCF client-based agent can effectively realize the bridging between two non-OCF IoT ecosystems without violating the OCF standard.

The mqtt protocol supports lightweight and efficient messaging for small IoT devices. In the publish/subscribe architecture of mqtt, when a mqtt client subscribes a topic (e.g., temperature) of a mqtt broker, the mqtt broker publishes the status of the topic (e.g., 24°C).

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
B. DEVELOPMENT ENVIRONMENT

1) O2U Bridge

For the UWB anchor/tag nodes, we used the UWB module (DWM1001 of Qorvo Corp. [28]) adopting IEEE 802.15.4-2011 standard [33]. As a UWB gateway, we utilized a Linux-based Raspberry Pi 3 Model B equipped with the UWB module. All entities for O2U bridging were implemented in the same Raspberry Pi (with IoTivity version 2.0) used as the UWB gateway. That is, in our implementation, a single device acted as both the UWB gateway and the O2U bridge.

2) O2L Bridge

For a LoRa end node, we used a LoRa module (B-L072Z-LRWAN1 of STMicroelectronics Corp. [34]) including LoRa transceiver (SX1276 of Semtech Corp.) and micro controller unit board (STM32L072 of Murata Corp.). To build a sensor-based LoRa end node, a multi-sensor board (X-Nucleo-IKS01A2 of STMicroelectronics Corp.) composed of temperature, humidity, atmospheric pressure, acceleration sensors was combined with the LoRa module. For a LoRa gateway, we used a Linux-based Raspberry Pi 3 Model B equipped with the LoRa module. A Linux-based laptop computer where the IoTivity version 2.0 was installed, was used as a LoRaWAN server. The gateway was connected to the LoRaWAN server through Ethernet link.

To demonstrate various implementation feasibility, the O2L bridge was implemented in two ways. In the LoRa-based metering use-case (Section VI-B), all O2L bridging modules were implemented on the LoRaWAN server. That is, the LoRaWAN server performs the full function of the O2L bridge. Whereas, in another LoRa use-case of Section VI-C, a virtual LoRa client was implemented in the LoRa gateway and, the bridge plugin and the virtual OCF server were in the LoRaWAN server.

3) U2Z Bridge

To realize the proposed bridging concept between two non-OCF systems, a U2Z bridge was implemented by utilizing O2U and O2Z bridges. The O2Z bridge developed in our previous work [29] was exploited. As a Zigbee client, a Zigbee color dimmable light (Zigbee 3.0 smart light bulb of Sengled Corp.) was used and an O2Z bridge was implemented on a Linux-based Raspberry Pi 2 Model B equipped with Zigbee 3.0 module (ZM3588S-USB-LR of Silicon Labs Corp.). The OCF client-based agent for merging O2U and O2Z bridges was implemented within the O2U bridge, and the O2U bridge was connected to the O2Z bridge via Ethernet link.

C. CONFORMANCE TESTS

According to the implementation guideline of OCF, all implementation results should be checked for the consistency with the existing OCF specification, using the OCF conformance test tool (CTT) [35]. Since the conformance tests validate whether implementation results meet the various requirements of OCF specification, if the CTT-based tests are passed, the test target device can interoperate with any type of certified OCF clients complying with the OCF standard.

Fig. 7 depicts the test configuration for checking the validity of the implemented O2U/O2L bridging functions, where the implemented bridge is connected to the computer running the CTT through the network connection and the CTT acts as an OCF client. Even if the details of all test cases are explained in [36], we outline the following key test cases, for readability.

In the test cases of device/resource discovery, the CTT sends discovery messages to a VOD (e.g., oic.d.uwb). When receiving the corresponding response messages from the VOD, the CTT checks whether the device type (e.g., oic.d.uwb) and resource list (e.g., oic.r.uwb.tag_position, security-related common resources) in the response messages are equal to the pre-defined configurations.

In the test of each OCF operation toward resources, several request/response messages are exchanged between CTT and VOD. To check if the Retrieve operation works properly, the CTT sends a Retrieve request message for a property of a specific resource (e.g., coordinates of oic.r.uwb.tag_position) through Get function, and investigates the corresponding response message including the current property value of the resource. To check an Update operation, the CTT sends an Update request message for a property of a resource through Put function. Since a test case of Update operation is implicitly incorporates Retrieve, after receiving the acknowledgement of an Update request, the CTT sends a Retrieve request message to examine whether the prior Update operation has been applied properly. In a test case of Notify operation, the CTT executes an Observe function for a property of a specific resource. To check if the change of the property value is notified in a timely manner, after artificially modifying the property value, the CTT investigates whether the changed value is reported within a pre-defined timeout.

For the security test, the CTT examines whether common security resources exist and work well through the OCF operations. Also, authentication, ownership transfer of a VOD from ‘unowned’ to ‘owned’, secure communication between CTT and VOD, and access control capabilities are investigated.
TABLE 2. Average latency of OCF operations and authentication

| Device Type | Resource Type | Retrieve | Update | Authen. |
|-------------|---------------|----------|--------|---------|
| oic.d.uwb   | oic.r.uwb.tag_position | 0.081 s  | 0.097 s | 0.123 s |
|             | oic.r.uwb.anchor_position |         |        |         |
|             | oic.r.uwb.phy_config      | 0.028 s  | 0.032 s | 0.107 s |
|             | oic.r.temperature         | 0.029 s  | 0.034 s |         |
|             | oic.r.sensor.atmosphericpressure | 0.032 s  |         |         |
|             | oic.r.humidity            | 0.028 s  | 0.038 s |         |

We carried out all required tests for the developed O2U/O2L bridges and VODs, according to the CTT procedures that comply with certification test requirements defined in [36]. Specifically, they were composed of the 97 test cases on each OCF device (because VODs and bridges are regarded as OCF devices) and another test case of 67 steps with 32 checkpoints for correct bridging operation of each bridge. Our O2U and O2L bridges successfully passed all tests.

D. LATENCY PERFORMANCE OF O2U AND O2L BRIDGES

Table 2 summarizes the average latency of OCF operations, measured by the CTT during the conformance test of the O2U and O2L bridges. And, the average authentication time that the CTT (OCF client) takes to acquire the access right for each virtual OCF server is also presented. As mentioned already in Section III-B, since the used UWB module does not provide a firmware command interface which enables access to PHY/MAC configuration parameters, the resource types ‘oic.r.uwb.mac_config’ and ‘oic.r.uwb.phy_config’ could not be implemented, and thus their performance results are not presented.

The latency of an operation is the elapsed time from a sending time of a request message to a receiving time of the corresponding response message. Note that the current CTT measures the latency only for an OCF operation of which it sends a request and receives the response (i.e., Retrieve or Update), by considering that there may be a time synchronization error between any two devices operating on their respective local clocks. Since a Notify operation is applied from a VOD in a bridge to the CTT in another computer, the CTT does not measure the latency of Notify operation.

The latency was measured by using Npcap which is a well-known Windows-based packet capture library [37]. The resources ‘oic.r.temperature’, ‘oic.r.sensor.atmosphericpressure’, and ‘oic.r.humidity’ are for the sensors mounted to a LoRa end node. It is observed from the table that the resources related to sensors have much lower latency than the other resources. This is because the data sizes representing the properties of sensor resources are much smaller than the data size of the ‘coordinates’ property [37] which leads to much shorter transmission/reception and processing time of payload.

VI. DEMONSTRATION WITH THREE EXAMPLE USE-CASES

We demonstrate three use-cases utilizing the implemented O2U, O2L, and U2Z bridges.

A. PROXIMITY-BASED ZIGBEE DEVICE CONTROL

The indoor localization-based Zigbee light control system was implemented, using the O2U and O2Z bridges.

1) Experimental Environment

The layout of this experiment is plotted in Fig. 8. As the O2U part, four UWB anchor nodes and a UWB gateway having a role of O2U bridge device were deployed in an office room and a UWB tag node was attached to a person. In addition, a Zigbee color light device and the O2Z bridge were included as the O2Z part. Fig. 9 shows the O2U bridge and O2Z bridge used in our experiment. By connecting the O2U bridge and the O2Z bridge through Ethernet link, we constructed the U2Z bridge described in Section V-B-3. The OCF client-based agent was implemented in the O2U bridge device. For simplicity, we defined beforehand power-on/off and multiple colors for Zigbee light device, depending on the indoor position of the UWB tag. Also, the localization frequency of the UWB tag was set to 10 Hz, which is a maximum value in the DMW1001 module.

2) Experimental Demo

The operation flow is as follows. The UWB tag calculates its location by measuring the ToFs from itself to nearby anchors, which leads to its location by measuring the ToFs from itself to nearby anchors.
FIGURE 10. Operation tests of Zigbee light control using UWB-based localization.

anchors, ten times per second, and sends it to the O2U bridge. The virtual UWB client in the O2U bridge, whenever receiving the location information (i.e., ten times per second), publishes the topic. However, to mitigate the excessive processing overhead by frequent execution of Notify operation, the virtual OCF server subscribes the location topic only twice per second. As a result, the ‘coordinates’ property of the ‘oic.r.uwb.tag_position’ resource is updated twice per second. Note that the ‘coordinates’ value taken by the virtual OCF server is the latest value reported by a tag. On the other hand, according to IoTivity implementation, if the client calls an Observe function for a property of a resource, the server notifies the corresponding property value to the client whenever the property value is changed. Thus, twice per second, the agent can obtain the most recently updated ‘coordinates’ of the tag. Based on the ‘coordinates’ value, the agent updates the ‘On/Off’ property of ‘oic.r.switch.binary’ resource or the ‘hue plus saturation’ properties of ‘oic.r.colour.hs’ resource within O2Z bridge, through an Update operation. Then, the virtual Zigbee client in the O2Z bridge controls the on/off switch or the color of the Zigbee light (the real Zigbee server). In our real tests, when a person with the tag node entered in the room, the Zigbee light was turned on. The RGB color of the light was changed according to the location of the moving person in the room (see Fig. 10).

The demonstration video of this use-case example is presented in [38]. In the real operation tests, the UWB-based positioning was considerably accurate. Also, through the OCF protocol, the location information of the UWB tag was forwarded well to the OCF client agent. Then, the agent could properly control the Zigbee light according to the location of a person with the UWB tag. On the basis of this experiment result, we expect that various localization-based services or applications can be efficiently realized by utilizing the bridging techniques developed herein.

3) Latency Performance of U2Z Bridge

The main operations of U2Z bridge in this use-case are the Retrieve operation for obtaining the property of the ‘oic.r.uwb.tag_position’ resource in the O2U bridge and the Update operation for updating the properties of ‘oic.r.switch.binary’ or ‘oic.r.colour.hs’ resources within the O2Z bridge. Let us address how long it took for the U2Z bridge to perform these two consecutive OCF operations. To measure the latencies of these operations, we reconstructed the U2Z bridge by merging two conformance test configurations toward O2U and O2Z bridges, where a single CTT acted as the OCF-client agent (refer to Fig. 3 and Fig. 7). The latency of each OCF operation was measured by using Npcap. The average latency that the agent (CTT) took to retrieve the ‘coordinates’ value of the tag in the O2U bridge was 0.081 s. And, the average latency that the agent took to update the ‘On/Off’ property or the ‘hue plus saturation’ properties in the O2Z bridge was measured as 0.031 s. That is, within the U2Z bridge, it takes about 0.112 s for the agent to retrieve the location property of UWB tag and update the property values for controlling the real Zigbee server. This level of latency might be acceptable for this use-case, especially when considering that the tag location information is updated with the period of 0.5 s.

B. OCF-BASED CONTROL OF WIDE-AREA METERING USING LORA

We implemented a prototype of a LoRa-based sensor information metering system by using the developed O2L bridge. This system was composed of a LoRa end node with sensor (real non-OCF server), a LoRa gateway, a LoRaWAN server, an O2L bridge, and an OCF client device. Fig. [11] shows the LoRa end node having multi-sensor board and the LoRa gateway constructed for our experiment. To build the LoRa gateway, a LoRa module and a Raspberry Pi were connected with a micro 5-pin USB cable. A Linux-based laptop

[9] The RGB color of the light was mapped to ‘hue and saturation’ properties.

[10] The LoRa end node was operated with three AAA-type cells, whereas the LoRa gateway used an external power line.
computer played a role of both the LoRaWAN server and the O2L bridge. That is, the full function of the O2L bridge was implemented on the server computer where the IoTivity version 2.0 was installed. And, the CTT which was installed on another notebook computer acted as an OCF client. The LoRa gateway, the LoRaWAN server/O2L bridge, and the CTT were interconnected via Ethernet.

In the operation tests, the sensor information acquired by the LoRa end node was successfully forwarded to the virtual OCF server in the O2L bridge, and the sensor information in the virtual OCF server was consistently updated. And, the average time that the OCF client retrieves the sensor information in the O2U bridge was measured as 0.030 s. Through this example use-case, it is expected that an operator can easily manage the sensor information of virtual OCF server through an OCF client and this facilitates the OCF-based management of the sensors deployed over wide area.

C. LORA AS BACKHAUL FOR INDOOR LOCALIZATION

The developed bridging techniques can be applied to using LoRaWAN as the backhaul links of the UWB-based indoor localization system. To demonstrate the feasibility of this approach, we constructed a bridged server device having both roles of UWB tag and LoRa end node by using Linux-based Raspberry Pi, UWB module, and LoRa module. This bridged server device is a user tag, which can be regarded as a combined UWB/LoRa device to forward the UWB positioning information to the monitoring server through a LoRa gateway. Among three entities for O2L bridging (virtual LoRa client, O2L bridging plugin, VODs), the virtual LoRa client was implemented in the LoRa gateway and the O2L bridging plugin and the virtual OCF servers were implemented in the monitoring server. That is, the gateway and the monitoring server were combined as the O2L bridge. An OCF client was run on the monitoring server for simplicity, although it can be executed on another separate device.

Fig. [12] shows the test layout for this use-case. A single UWB/LoRa server device, six UWB anchor nodes, a LoRa gateway and a monitoring server were deployed in the corridor of the building.

In the combined UWB/LoRa server device (tag), the \((x, y, z)\) coordinates obtained from the UWB module is forwarded to the LoRa module within the same device using the mqtt protocol. Then, the coordinates is delivered into the virtual LoRa client in the LoRa gateway using LoRa protocol, ten time per second. As a mqtt broker, the virtual client publishes the coordinates to the virtual OCF server in the monitoring server, via Ethernet link. The virtual OCF sever which is a mqtt client subscribe the coordinates topic, twice per second. The OCF client in the monitoring server acquires the ‘coordinates’ property of ‘oic.r.uwb.tag_position’ resource by monitoring the virtual OCF server through the *Observe* function.

On the other hand, the OCF client in the monitoring server can deliver control commands to the combined UWB/LoRa server device, by updating the resource properties of the virtual OCF server through an OCF *Update* operation. We expect that this concept can be usefully utilized in the management of UAVs for logistics services.

VII. CONCLUSION

In this paper, we have developed the O2U and O2L bridging techniques and have implemented these bridges, based on IoTivity. We have verified that the proposed O2U/O2L bridging techniques comply with the OCF standard, according to the OCF CTT-based test procedure, and also have confirmed that the implemented bridges can properly manage commercial UWB and LoRa devices. In addition, we have suggested a practical method which can cost-effectively realize the bridging between different non-OCF IoT ecosystems without violation of the OCF standard by utilizing the OCF client-based agent. To demonstrate the usefulness of the developed bridging techniques, we have additionally presented three example use-cases with O2U, O2L, and O2Z bridges.

We expect that the proposed O2U/O2L bridging techniques would be added to the OCF bridging standard like our previous work on the O2Z bridge in [29]. Also, the developed technique can provide guidelines for realizing the bridging between heterogeneous IoT ecosystems. Therefore, the work reported in this paper would promote the convergence to an integrated IoT ecosystem by interconnecting various promising IoT ecosystems.

REFERENCES

[1] Forbes, “152,000 smart devices every minute In 2025: IDC outlines the future of smart things,” Accessed: Mar., 21, 2022. [Online]. https://www.forbes.com/sites/michaelkanellos/2016/03/03/152000-smart-devices-every-minute-in-2025-idc-outlines-the-future-of-smart-things/

[2] Open Connectivity Foundation, “OCF specification 2.2.5,” Jan. 2022. Accessed: Mar., 21, 2022. [Online]. https://openconnectivity.org/developer/specifications

[3] AllSeen Alliance, “AllJoyn open source project,” Accessed: Mar., 21, 2022. [Online]. https://github.com/alljoyn/alljoyn.github.com/wiki

[4] OneM2M, “OneM2M specification,” Accessed: Mar., 21, 2022. [Online]. http://www.onem2m.org/technical/published-drafts

[5] Connectivity Standards Alliance, Accessed: Mar., 21, 2022. [Online]. https://csa-iot.org/

[6] Matter, Accessed: Mar., 21, 2022. [Online]. https://buildwithmatter.com/
[7] Bloomberg, “The Open Connectivity Foundation introduces world’s first international standard for smart home; will demonstrate new OCF unlicensed cloud interface-based smart home solutions at CES 2020,” Accessed: Mar., 21, 2022. [Online]. https://www.bloomberg.com/press-releases/2020-01-02-the-open-connectivity-foundation-introduces-worlds-first-international-standard-for-smart-home-will-demonstrate-new-ocf

[8] Open Connectivity Foundation, “IoTivity project,” Accessed: Mar., 21, 2022. [Online]. https://github.com/iotivity/iotivity

[9] Bluetooth, “Bluetooth specifications,” Accessed: Mar., 21, 2022. [Online]. https://www.bluetooth.com/specifications

[10] STMicroelectronics, “B-L072Z-LRWAN1,” Accessed: Mar., 21, 2022. [Online]. https://www.st.com/en/evaluation-tools/b-l072z-lrwан1.html

[11] HYUN SEOB OH received the B.S. degree in the department of information and computer engineering from Ajou University, Suwon, South Korea, in 2012, the PhD degree from the Department of Computer Science, Seoul National University, South Korea, in 2021.

He is currently with Samsung Research, Samsung Electronics, South Korea. His research interests include radio resource management for wireless and mobile networks, wireless local area networks, enhanced ultra wideband (UWB), and the Internet of Things.

[12] Open Connectivity Foundation, “Certification test requirements (version 2002.1.0),” Accessed: Mar., 21, 2022. [Online]. https://workspace.openconnectivity.org/kws/test_tools/Certification_Test_Requirements_v2002.1.0.pdf

[13] “Packet capture library for Microsoft Windows,” Accessed: Mar., 21, 2022. [Online]. https://npcap.com

[14] “A demonstration of proximity-based Zigbee light control,” Accessed: Mar., 21, 2022. [Online]. https://youtu.be/6Ya2Y4SRVqs

[15] DONG GEUN JEONG (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer engineering from Seoul National University, Seoul, South Korea, in 1983, 1985, and 1989, respectively.

From 1989 to 1999, she was with the Department of Computer Engineering, Hanyang University, South Korea. In 1999, she joined Seoul National University, as a Faculty Member, where she is currently a Professor with the Department of Computer Science and Engineering. Her research interests include resource management for wireless and mobile networks, high-speed networks, communication protocols, network performance evaluation, and the Internet of Things. She served on the Editorial Board of the Journal of Communications and Networks (JCN) from 2002 to 2017.

[16] Forbes, “Amazon’s prime air drone delivery service gets FAA approval,” Aug. 31, 2020. Accessed: Mar., 21, 2022. [Online]. https://www.forbes.com/sites/ilkerkoksal/2020/08/31/amazons-prime-air-drone-delivery-service-gets-faa-approval

[17] Open Connectivity Foundation, “OCF bridging specification (version 2.2.3),” Apr. 2021.

[18] Open Connectivity Foundation, “OCF resource type specification (version 2.2.3),” Apr. 2021.

[19] Orato, “MDEK1001 development kit,” Accessed: Mar., 21, 2022. [Online]. https://www.decawave.com/product/mdek1001-deployment-kit/

[20] H. S. Oh, S. B. Seo, G. T. Lee, W. S. Jeon, and M. G. Lee, “OCF-to-Zigbee standardization for low-rate wireless personal area networks (LR-WPANs),” 2012.

[21] Open Connectivity Foundation, “Certification test requirements (version 2.2.5),” Jan. 2022.

[22] HYUN SEOB OH received the B.S. degree in the department of information and computer engineering from Ajou University, Suwon, South Korea, in 2012, the PhD degree from the Department of Computer Science, Seoul National University, South Korea, in 2021.

He is currently with Samsung Research, Samsung Electronics, South Korea. His research interests include radio resource management for wireless and mobile networks, wireless local area networks, enhanced ultra wideband (UWB), and the Internet of Things.

[23] W. S. Jeon and D. G. Jeong, “Adaptive uplink rate control for confirmed data transmission in LoRa networks,” IEEE Internet of Things Journal, vol. 8, no. 22, pp. 16418–16426, Nov. 2021.

[24] J. Pospisil, R. Fujdiak, K. Mikhaylov, “Investigation of the performance of TDoA-based localization for LoRa networks,” Sensors, vol. 20, no. 19(5464), pp. 1–22, Sept. 2020.

...