Sensors and Materials, Vol. 33, No. 1 (2021) 147–169
MYU Tokyo

S & M 2442

BluMoon: Bluetooth Low Energy Emulator for Software Testing

Tsubasa Yumura,[1,2*] Kunio Akashi,[1,2] Tomoya Inoue,[1,2] and Yasuo Tan[1,2]

[1]National Institute of Information and Communication Technology, 2-12 Asahidai, Nomi, Ishikawa 923-1211, Japan
[2]Japan Advanced Institute of Science and Technology, 1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan

(Received July 21, 2020; accepted November 9, 2020)

Keywords: Bluetooth Low Energy, network virtualization, wireless network, network emulation

In software system testing using Bluetooth Low Energy (BLE), it is necessary to evaluate the system, including the wireless communication. However, it is difficult to build a test environment for testing with physical machines because of installation costs. This problem can be solved by emulation to reproduce BLE communication on computers; however, a BLE emulator is required. In this paper, we propose a BLE emulator called BluMoon for testing software systems using BLE. We impose the following requirements on the BLE emulator: (1) calculating the received signal strength for each frame and (2) imitating radio interference. To satisfy these requirements, we devised a software-implemented BLE controller with a host controller interface as a boundary and devised a data format called the BluMoon frame for sending and receiving data frame by frame. We designed and implemented BluMoon, and performed functional and performance evaluation as well as a comparative experiment with a physical environment. The results revealed that it is possible to implement a BLE emulator that meets the aforementioned requirements.

1. Introduction

Devices connected to a network are not limited to personal computers (PCs) and mobile phones; home appliances, wearable devices, and various other devices also communicate with peripherals and servers. Bluetooth Low Energy (BLE)[(1)] is one of the most commonly used wireless communication protocols for the connection of devices. This protocol is a short-range wireless communication standard that is suitable for installation in embedded devices owing to its low power consumption. It is also used as a beacon for positioning indoors when the Global Positioning System (GPS) cannot be used.

The development of a system using BLE involves the challenge of handling wireless communication. Because BLE involves wireless communication using radio waves, its behavior changes depending on the position of the transceiver and surrounding environment. For example, the received signal strength (RSS) used for proximity detection changes depending on the distance between the transmitter and the receiver. The RSS is also affected by reflection, shielding, and diffraction by surrounding objects, walls, and the ground. The RSS thus

*Corresponding author: e-mail: yumu@yumulab.org
https://doi.org/10.18494/SAM.2021.2986

ISSN 0914-4935

fluctuates owing to subtle changes in the environment. Therefore, when using the RSS for an application, it is necessary to consider this fluctuation. In addition, BLE transmitters other than the target device may cause radio wave interference, resulting in loss of communication.

Typical software development is performed by confirming that the software behaves as expected and by correcting defects. However, it is difficult to verify the behavior of systems using BLE, as this requires physical devices to send and receive radio waves. To use physical devices, time is required for installation. In addition, when changing the layout of transceivers for testing, the equipment must be moved by people or machines. The problem of system operation verification using BLE can be solved using a virtual environment built on a computer for testing wireless communication. If the movement and communication of BLE devices can be performed virtually with computers instead of physical BLE devices, the cost of equipment and installation can be greatly reduced.

There are two methods for constructing a virtual wireless communication environment: network simulation and network emulation. Network simulation is used for investigating network characteristics, such as throughput and delay, by modeling and simulating network traffic. NS-3[(2)] is a typical network simulator. However, simulators cannot be used for tests that run applications the same way as for actual environments or for sending and receiving frame-by-frame data to verify the operation of the system. Network emulation, in contrast, can be used without impairing the network function from the viewpoint of the upper layer. The virtual environment used for testing must run the same software as for physical devices, as the development cost would increase if different software programs were written for the emulator and physical device. It is also possible for different software programs to behave differently. In this study, we adopt the emulator method to construct a virtual environment so that the same software can be run as for physical devices.

The IEEE 802.11 emulation system NETorium[(3)] imitates wireless communication over a wired network using a virtual network device. In NETorium, wireless frames are encapsulated and transmitted via Ethernet. Wireless characteristics, such as delay and packet loss, are injected as parameters, and radio wave interference is imitated by collision judgment by the virtual network device upon the reception of each frame. In addition, the Linux operating system (OS) can handle a virtual interface in the same way as an actual IEEE 802.11 device and can execute the same software as an actual device. However, no BLE emulator exists that can realize the same function as NETorium. Furthermore, it is difficult to develop a BLE emulator by modifying the wireless emulators of other protocols. In BLE, the frame transceiver called the controller has a state, and a state transition is performed according to the stage of connection establishment with other devices. Frame transmission may be performed autonomously. The characteristics of the BLE controller are different from those of IEEE 802.11 and other protocols. In this study, we design and implement a BLE emulator for testing software systems, and impose several requirements on the proposed emulator. BLE applications use the RSS for applications such as proximity detection. Because the RSS varies from frame to frame, it is necessary to use an RSS that matches the given situation when receiving a frame. Therefore, BLE communication must be emulated on a frame-by-frame basis, and the RSS must be calculated for each frame. In addition, radio interference is a typical cause of

inactivity in the field despite working well with small-scale verification. When testing a system that communicates with a large number of devices, it is important to verify the availability of connections according to the device layout and the number of installed devices. Therefore, it is necessary to verify radio wave interference at the test stage in the virtual environment. The following requirements are proposed for BLE emulators:
(1) calculating the RSS for each frame and
(2) imitating radio interference.
These requirements are used to design BluMoon, which aids software system development using BLE emulation.

A workshop paper[4] and Japanese research report[5] were published regarding an initial version of BluMoon. In the present paper, in addition to the summary of the previous papers, we describe the design and implementation of BluMoon for calculating the RSS and simulating radio wave interference. Furthermore, the results of performance evaluation and comparative experiments with physical devices are provided.

2. BLE

Bluetooth is a standard for short-range wireless communication using the 2.4-GHz band, and BLE is a low power consumption protocol adopted by Bluetooth version 4.0. Bluetooth specifications including BLE are defined by the Bluetooth Special Interest Group. This section describes the BLE specifications related to the design of BLE emulators, especially the hierarchical structure and connection establishment.

2.1 Layered architecture of Bluetooth

Bluetooth, including BLE, has a hierarchical structure consisting of an application, a host, and a controller (Fig. 1). Applications using Bluetooth are located at the top layer and often use host functions. The controller is responsible for the function of transmitting and receiving radio waves for communication. Commands, notifications, and data are exchanged between the host and the controller via the host controller interface (HCI).

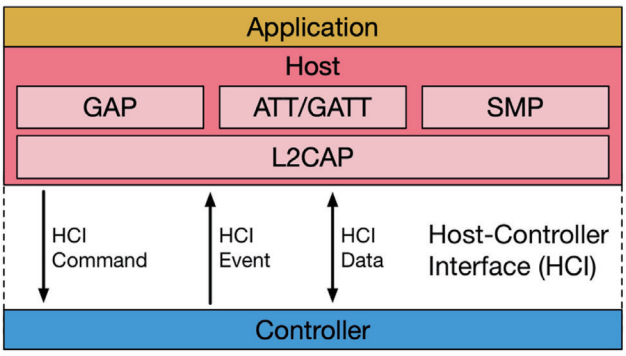

Fig. 1. (Color online) Layered structure of Bluetooth.

To facilitate application development, the host provides various profiles and protocols, such as the Generic Access Profile (GAP), Attribute Protocol (ATT), Generic Attribute Profile (GATT), Security Manager Protocol (SMP), and Logical Link Control and Adaptation Protocol (L2CAP).

The controller sends and receives radio waves to communicate with other controllers. There are several types of controllers. BLE uses the Low Energy (LE) controller. There is another controller called the Basic Rate/Enhanced Data Rate (BR/EDR) controller, which is generally called Classic Bluetooth. BR/EDR and LE are major Bluetooth standards, and most commercially available controllers are implemented with dual stacks that support both BR/EDR and LE. In this paper, *BLE controller* or *controller* refers to an LE controller. There are three types of HCI communication methods: HCI commands that send commands from the host to the controller, HCI events that send responses and notifications from the controller to the host, and HCI data that send and receive data bidirectionally. These are sent and received in the HCI frame format.

2.2 State machine of BLE controller

BLE devices have two roles, namely, central and peripheral. Figure 2 illustrates the sequence of establishing a connection between a central and a peripheral. The BLE controller takes one

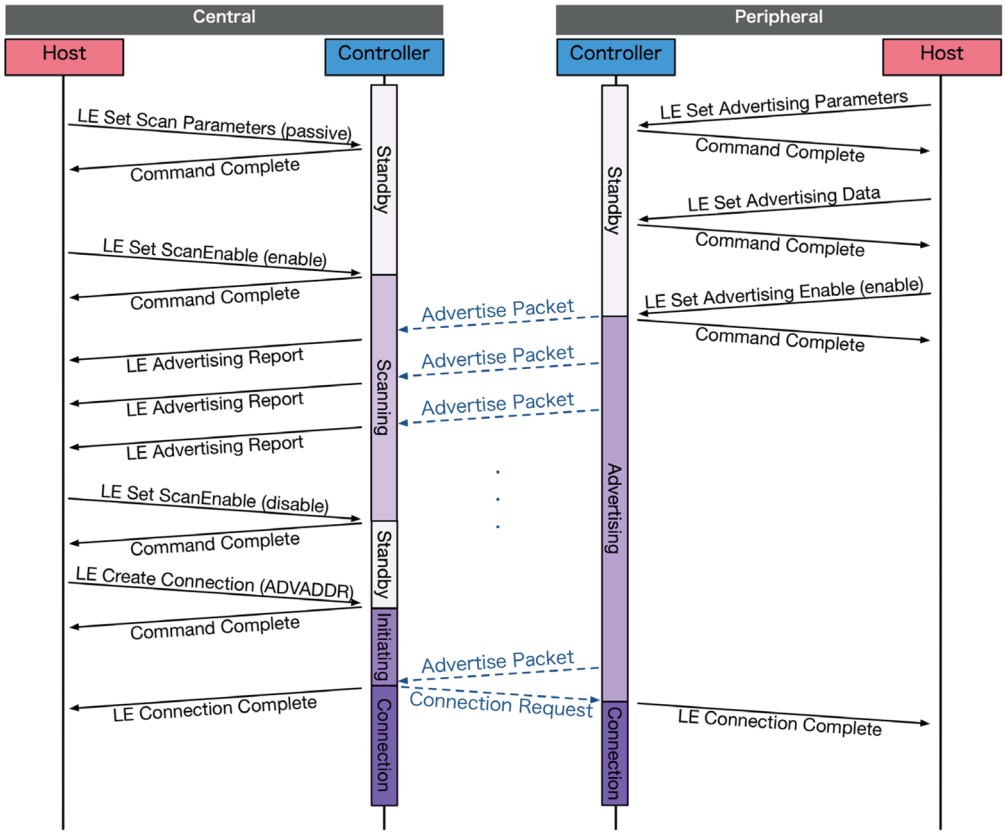

Fig. 2. (Color online) Sequence diagram of HCI command, HCI event, and BLE frame until connection establishment.

of five states: standby, advertising, scanning, initiating, and connection (Fig. 3). A central device takes the following state transition until a connection is established: standby, scanning, initiating, and connection. A peripheral terminal takes the following transition states until a connection is established: standby, advertising, and connection. Standby is the initial state after turning on the controller.

Advertising is the state of regularly sending advertising packets, and the transition to advertising is performed by *LE Set Advertising Enable*, an HCI command. Scanning is the state of waiting for surrounding advertising packets. When receiving an advertising packet, *LE Advertising Report*, an HCI event, is sent from the controller to the host. Transition to scanning is performed by *LE Set Scan Enable*, an HCI command. Initiating is the state of being ready to establish a connection and waiting for surrounding advertising packets from peripherals. Transition to initiating is performed by *LE Create Connection*, an HCI event.

Connection is the state in which a connection is established between a central and a peripheral. The central device transitions to the connection state when it receives an advertising packet in the initiating state. At this time, the central device sends a connection request. When the peripheral receives this connection request, it transitions to the connection state. In the connection state, devices can send and receive data with each other. When the connection is terminated, the controller returns to the standby state. In this paper, we call a data frame (e.g., advertising packets) exchanged between controllers *connection requests* and data transmission a *BLE frame*. Figure 4 presents the structure of the BLE frame. Notably, advertising packets are also used for information dissemination. There are standards, such as iBeacon[6] and Eddystone,[7] that aim to utilize advertising packets for indoor positioning and information transmission.

3. Related Works

3.1 BLE simulators and emulators

This subsection describes existing simulators and emulators used for testing networks and applications related to BLE. Table 1 shows the compliance status of these simulators and emulators with requirements listed in Sect. 1. In addition, Fig. 5 indicates whether the simulators and emulators execute the BLE application, host, controller, and communication virtually or physically. BluMoon proposed in this paper is designed to emulate the controller and communication so that these requirements are satisfied.

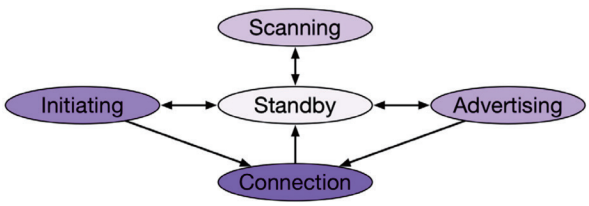

Fig. 3. (Color online) State diagram of BLE controller.

Fig. 4. (Color online) Data format of BLE frame.

Table 1
Adaptation status of existing emulation and simulation method to requirements of BLE emulator in this paper: executing running code required for emulator, calculating RSS for each frame described as requirement 1 in Introduction, and imitating radio interference described as requirement 2 in Introduction.

	Executing running code	Calculating RSS	Imitating radio interference
Network simulator[2]	*2	✔	✔
Btvirt[8]	✔		
Android Emulator[9]	✔		
Wear OS Emulator[9]	✔		
iOS Emulator[10]*1	*3		
Peripheral Simulator[11,12]	✔		
BluMoon (our proposal)	✔	✔	✔

*1Limited in iOS version 5
*2Enable if using Direct Code Execution. Details are described in Sect. 3.3
*3Executing binary built for simulator from same source code as for physical devices

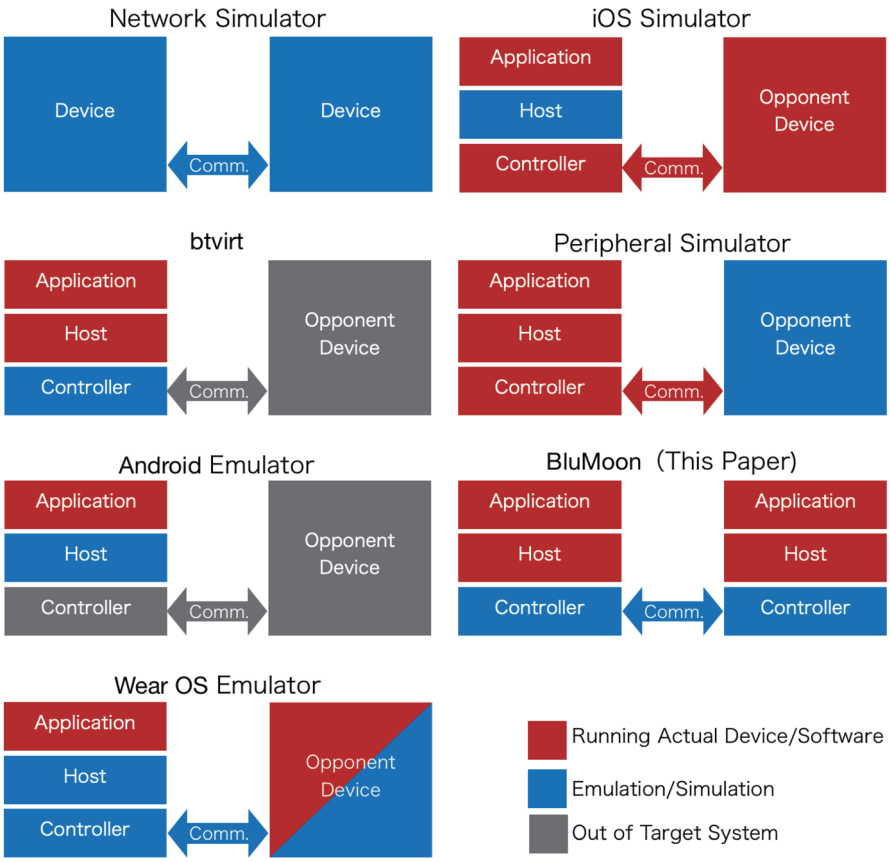

Fig. 5. (Color online) Comparison of simulation targets of existing emulation/simulation methods.

Network simulators are often used to simulate communications, including BLE. A typical example is NS-3. However, network simulators simulate network traffic to investigate throughput and delay, and are thus not appropriate for application testing.

Btvirt is an open source software program included in the tools provided by the Linux Bluetooth stack BlueZ.[8] Btvirt behaves as a virtual Bluetooth controller. It sends and receives HCI frames via the UNIX domain socket, accepts HCI commands, and returns appropriate HCI events. Btvirt can be handled in the same way as a physical controller connected via Universal Serial Bus (USB) or Universal Asynchronous Receiver/Transmitter (UART). However, btvirt does not have a function to communicate with other devices. Therefore, it cannot be used for system tests that involve communication.

In mobile application development, it is common to use the emulator or simulator included in the development environment. For Android, there are many emulators that are included in the development environment Android Studio,[9] as well as other emulators created by third parties. Because these emulators do not have a function for imitating Bluetooth, the Bluetooth operation test is performed with physical devices. Notably, the emulator of Wear OS,[13] an Android-based OS for wristwatch-type devices, can be used as a BLE peripheral. The Wear OS emulator has the function of establishing a connection with a physical Android smartphone or the Android emulator. When pairing with a physical Android smartphone, it should be connected to a PC running a Wear OS emulator with a USB cable. For iOS, the iOS simulator is included in the development environment Xcode.[14] The iOS simulator runs a binary that is built for a simulator on MacOS, and the binary is built from the same source code as that of the physical iOS device.

When running an application that uses Bluetooth on an iOS simulator, the simulator can use the Bluetooth controller on the computer. By preparing the physical device for the peripheral device, the iOS simulator can verify the operation using BLE communication. However, this function has been discontinued in the iOS version 5 simulator and has not been implemented in the later version.[10] The BLE emulator discussed above provides a test environment tailored to specific OSs and devices, and thus cannot be used for general BLE testing.

There are also smartphone applications that imitate the behavior of peripheral devices for the BLE central application test. In this paper, we call these applications peripheral simulators. BLE Peripheral Simulator[11] is an Android application for testing the features of Web Bluetooth. It imitates the behavior of three types of sensor devices: battery, heart rate monitor, and thermometer. Similarly, LightBlue Explorer[12] can launch various peripheral devices as virtual peripherals. Peripheral simulators play the role of peripheral devices using a physical smartphone. These simulators do not emulate BLE communication but transmit physical radio waves with a physical device. Therefore, we cannot use peripheral simulators for BLE system testing using emulation.

3.2 Network virtualization of Bluetooth

BluMoon performs BLE communication emulation frame by frame. For this purpose, we consider virtualization that places a BLE frame on another protocol. The virtualization of Bluetooth communication has been studied from various perspectives. For performing Bluetooth communication between devices in remote areas, several methods have been proposed for transmitting and receiving by encapsulating Bluetooth data. UbiPAN[15] makes it possible to communicate with devices in remote areas via Bluetooth by installing gateways

as a bridge. Tsuda *et al.* [16] proposed a method to transfer HCI messages by incorporating this gateway function into a terminal, and Okada and Suzuki[17] extended this method to support BLE. In these methods, the frames are encapsulated, transferred, and then restored to the Bluetooth wireless frame. The transmitted frame does not contain the characteristic information of wireless communication, such as the RSS indicator (RSSI), and is thus insufficient for use in our BLE emulation.

BlueMonarch[18] uses emulation for Bluetooth testing and is an emulator that imitates Bluetooth OBject EXchange (OBEX) file transmission using service discovery protocol (SDP) communication that does not need to establish pairing. BlueMonarch is used for the evaluation of content distribution systems and imitates Bluetooth using another Bluetooth protocol. It does not use communication protocols other than Bluetooth and thus differs from the emulation of our research.

There are also emulators that use software defined radio (SDR) to Bluetooth testing. Wang *et al.*[19] proposed to perform upper layer protocol emulation with SDR. Liu *et al.*[20] proposed Bluetooth signal emulation performed with field-programmable gate array (FPGA) controlled SDR. These proposed methods can accurately reproduce Bluetooth communication by using SDR. However, our purpose is software testing. In particular, we are targeting running large amounts of application-level software. The methods using SDR have a bottleneck in the installation cost of SDR, and it is difficult to scale the test environment. Our method ensures scalability by using a network emulator that uses Ethernet communication for the computer cluster.

3.3 Wireless network emulators other than Bluetooth

There are many wireless network emulators for wireless network protocols other than Bluetooth. Some wireless network emulators are implemented by software, while others are implemented by dedicated hardware. The software-based wireless network emulator QOMET[21] injects the characteristics of wireless communication into wired network communication. QOMET was designed assuming IEEE 802.11 emulation. On the basis of QOMET, a wireless network emulation testbed QOMB,[22] an IEEE 802.15.4 emulator,[23] and a dynamic network emulation tool DynamiQ[24] have also been proposed. Another emulator for IEEE 802.11 is Meteor.[25] NETorium, a large-scale software-based wireless network emulator, imitates frame collisions using a system called Asteroid. It uses mac80211_hwsim[26] as a virtual interface for emulation.

Tazaki *et al.*[27] proposed Direct Code Execution (DCE), a framework for executing the Linux running code using network simulator NS-3. DCE does not support BLE, but using the DCE concept to imitate BLE communication is an option. However, DCE has some disadvantages. The first is that processing is concentrated on NS-3. It may prevent scalability. This was mentioned in their paper. Second concerns mobility. BLE is used in smartphones and wearable devices, which move with the user and RSS changes. Therefore, it must be possible for the position of the node in the virtual environment to be changed. Because it is assumed that the operation is performed while viewing the result, the node position must be able to be changed

at any timing during the execution of the emulation. However, NS-3 only supports predefined node movements. Therefore, DCE cannot flexibly specify the node position during simulation. The last is about clock. DCE uses a simulated clock to synchronize communication, and the code is executed according to it. However, in the system test including BLE communication, we would like to confirm the operation of the entire system in a wall clock, not in a simulated clock. From these, we did not adopt the concept of DCE, but designed with another architecture described in the next section.

In addition, there are hardware-based wireless emulators that use a FPGA[28] and commercial products, such as Network Emulator II.[29] Hardware-based wireless emulators can inject wireless characteristics with high accuracy; however, it is difficult to build a test environment using many devices owing to the required hardware.

4. Design Policy of BluMoon

In this section, we propose a method of meeting the requirements of the BLE emulator, as described in Sect. 1, and present design policies.

4.1 Emulation of controller divided by HCI

As an emulator, BluMoon must run the same software for physical devices. As described in Sect. 2, in Bluetooth, the host and controller are connected via an HCI. If the behavior of the controller connected to the lower layer of the HCI is emulated, the software for physical devices can also be run on the emulator. The emulated controller can be handled from the host layer via the HCI in the same way as a physical controller. In this paper, we call this emulated controller the BluMoon controller (BM-CTL).

Another option is to emulate including the host layer. However, because the host contains many profiles, it is difficult to implement an emulator that imitates the function of the host. The HCI-bounded method proposed in this paper can use the actual software as the host. Therefore, it is possible to reduce the mounting and easy to follow the Bluetooth specification version.

4.2 Emulation of frame-by-frame communication

To realize the emulation of frame-by-frame communication, the BM-CTL sends and receives data corresponding to BLE frames. We refer to these transmission and reception data as the BluMoon frame (BM-FRM). The BM-FRM format is illustrated in Fig. 6. The BM-FRM stores the actual BLE frame of the BLE specification format. In addition, it stores three types of metadata (channel, TxPower, and location) that are used to calculate the RSS and imitate interference.

Fig. 6. (Color online) Format of BM-FRM.

4.3 Calculation of RSS

Receiving the BLE frame, the BLE controller calculates the RSS for each frame and passes it to the host. The RSS has the characteristic of being attenuated according to distance. Considering the simplest model, the radio propagation loss in free space, $L(d)$, can be represented by

$$L(d) = -20 \log_{10}\left(\frac{4\pi d}{\lambda}\right), \tag{1}$$

where λ is the wavelength and d is the distance between the transmitter and the receiver. Because the received radio wave intensity in a real environment is affected by various factors, such as reflection and diffraction, it is not simple enough to be imitated by free space radio wave propagation loss. However, calculation by detailed modeling requires time and effort for setting conditions and for calculation. BluMoon calculates the RSS from the original transmission power, free space radio wave propagation loss, and additive white Gaussian noise (AWGN) δ.

$$RSS = TxPower - 20 \log_{10}\left(\frac{4\pi d}{\lambda}\right) + \delta \tag{2}$$

AWGN δ is generated by randomizing the probability density function $P(\delta)$ of the normal distribution with mean 0 and standard deviation σ.

$$P(\delta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\delta^2}{2\sigma^2}\right) \tag{3}$$

TxPower for calculating the RSS is included in the BM-FRM header. The distance d is obtained from the positions of the transmitter and receiver. The position of the node is generated outside the controller and sent to each BM-CTL. We refer to the data that describe the location of all nodes as location information (LC-INF) and refer to the module that creates LC-INF as the location generator (LC-GEN). The transmitter position is included in the BM-FRM header when transmitting. The BM-CTL of each receiver records its own position and uses it when calculating the RSS. There are several ways to implement the LC-GEN. One is a scenario-based method that determines the location information in advance, and another is a multi-agent simulator method that moves on the basis of the rule possessed by each node.

4.4 Occurrence of radio interference

BLE divides the band from 2.400 to 2.480 GHz into 0.002 GHz and configures 40 channels that do not overlap. If a BLE frame is transmitted on the same channel from a different node,

the signal cannot be decoded and communication cannot be established. In this paper, this phenomenon is called radio wave interference. BLE has a mechanism to avoid radio wave interference by BLEs and other 2.4 GHz band communication systems. Advertisement uses three channels (37, 38, 39) in a fixed manner; thus, radio wave interference is likely to occur. Bluetooth version 4.2 specifies a frame transmission bit rate of 1 Mbps; in other words, it takes 200 μs to send a 200-bit frame. Radio interference occurs during the transmission of the frame. However, because BluMoon transmits Ethernet frames, the time required for frame transmission is much shorter than that for actual BLE transmission. The actual transmission time required for BLE is then estimated, and a pseudo-transmission state is generated to imitate interference. The design of this interference was proposed by Asteroid, which was included in the wireless network emulator NETorium. Specifically, the time from receiving the BM-FRM is regarded as the time required to receive the BLE frame. If another BM-FRM is received, it is considered that interference has occurred.

The time required to receive a BLE frame is denoted t_{rcv} and is proportional to the length of the frame. t_{rcv} is derived as

$$t_{rcv} = \frac{L}{10^6}. \tag{4}$$

Here, L is the length of the BLE frame, that is, the length of the BM-FRM body. This method of simulating radio wave interference is explained in Fig. 7.

Figure 7 illustrates the flow of time when two BM-FRMs are received sequentially. In Fig. 7(a), there is no interference, while in Fig. 7(b), there is interference. In each subfigure, a circle indicates the time at which the BM-FRM is received; these are referred to as t_0 and t_1. t_{rcv}, which is the time required to receive a BLE frame, is described in detail above. In the figure, the first frame is blue and the second frame is red, and they are independent. If another BM-FRM is not received before t_{rcv} elapses from the time of receiving the BM-FRM, it is considered that there is no radio wave interference, and the BM-FRM is accepted [Fig. 7(a)]. However, if another BM-FRM is received within the t_{rcv} time from the time of receiving the BM-FRM, it is

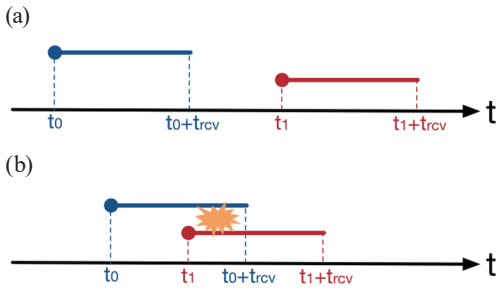

Fig. 7. (Color online) Explanation of radio interference emulation in BluMoon. t_0 and t_1 indicate the arrival time of the BM-FRM, and t_{rcv} indicates the time required for receiving the entire BLE frame. (a) If another BM-FRM is not received before t_{rcv} elapses from the time of receiving the BM-FRM, it is considered that no radio interference has occurred, and the BM-FRM is accepted. (b) When another BM-FRM is received, it is considered that radio interference has occurred, and both BM-FRMs are discarded.

considered that radio wave interference has occurred, and both BM-FRMs are discarded [Fig. 7(b)].

Note that another method for simulating radio wave interference is to generate packet loss with an arbitrary probability. However, in the BLE use case, the surrounding radio wave conditions change markedly. It is more versatile to generate radio wave interference depending on the given situation than to generate packet loss with a fixed probability. It is possible to change the situation by frequently updating the probability of packet loss. However, in this case, an additional mechanism is necessary to collect the surrounding radio wave environment and calculate the packet loss probability. Because BluMoon emulates on a frame-by-frame basis, it is possible to use the received frames to imitate radio interference without collecting additional information. Therefore, we use the proposed method in this study.

5. Design and Implementation of BluMoon

In this section, we describe the specific software design and implementation of BluMoon to realize the design policy presented in Sect. 4. In this study, we implement BluMoon as software running on Linux.

5.1 Overview

BluMoon consists of the BM-CTL and LC-GEN. The BM-CTL emulates the behavior of the controller, while the LC-GEN generates the location information of each device and sends it to all BM-CTLs. We designed the BM-CTL by dividing it into the following three modules:

- BluMoon HCI transceiver (BM-HCI)
- BluMoon manager (BM-MGR)
- BluMoon connector (BM-CNC)

The design of dividing a module into functional units simplifies the implementation, in turn making it easier to follow the Bluetooth standard update. For example, the BM-HCI can be modified to follow updates to the HCI. Similarly, the BM-CNC can be modified to follow updates to the link layer. The LC-INF is required for the calculation of the RSS. The LC-GEN generates the LC-INF and transfers it to each BM-CTL. These configurations are summarized in Fig. 8.

5.2 BM-HCI

The BM-HCI has the function of transmitting and receiving HCI frames with the host layer. When receiving HCI data or HCI commands from the host, the BM-HCI transfers the contents to the BM-MGR. When receiving an HCI event generation or data transmission command from the BM-MGR, the BM-HCI transmits the HCI event or data to the host. Figure 9 illustrates the relationship between the BM-HCI and BlueZ, which is a Bluetooth stack for Linux. BlueZ acts as a host on Linux. In BlueZ, some drivers are connected to BlueZ Core, which has the core functions of the host.

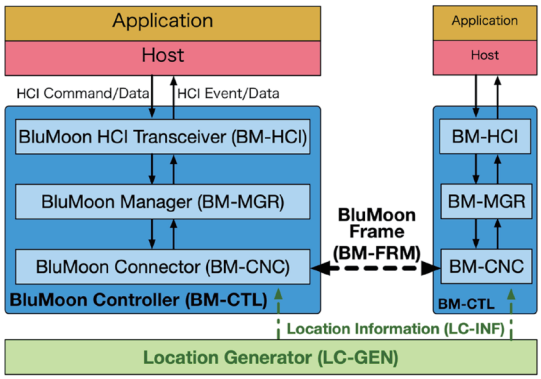

Fig. 8. (Color online) Design overview of BluMoon.

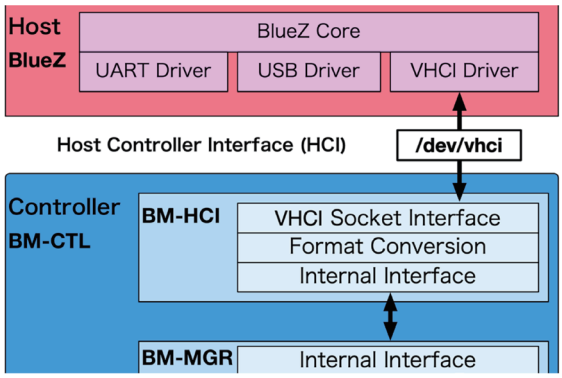

Fig. 9. (Color online) Architecture of BlueZ (Linux Bluetooth stack) and BM-HCI. These are connected with the UNIX domain socket /dev/vhci.

The driver is a Virtual Host Controller Interface (VHCI) driver for connecting to a virtual controller, and a UART driver and USB driver for connecting to the actual device. There are several types of drivers. The UART and USB drivers connect to controllers with their protocols, while the VHCI driver connects to a virtual controller. Because the difference in the controller device is absorbed by the driver, the upper layer from BlueZ Core can handle the controller without depending on the device. Virtual controllers connected to the VHCI driver can be controlled by software for the actual devices. The VHCI driver sends and receives HCI frames via the UNIX domain socket /dev/vhci. To act as a virtual controller for the BM-CTL, it is necessary to launch a socket server that waits for VHCI sockets. We implemented the VHCI socket interface in the BM-HCI by extending btvirt, which is a reference implementation of a virtual controller communicating with BlueZ. The BM-HCI converts the format of the HCI frame received from the VHCI socket interface and passes it to the BM-MGR. When receiving the HCI event generation and data transmission commands from the BM-MGR, the BM-HCI sends the HCI frame corresponding to the command to BlueZ via the VHCI socket interface.

5.3 BM-MGR

The BM-MGR manages the entire BluMoon controller for sending and receiving. The BM-MGR manages the five states (Fig. 3) of standby, advertising, scanning, initiating, and connection. In addition, it controls the transmission of the BM-FRM and the receiving slot. The BM-MGR creates a BM-FRM by adding metadata of the channel and TxPower to the BLE frame format (Fig. 4) and passes it to the BM-CNC. In the advertising state, the BM-MGR has a timer and transmits at regular intervals. The BLE controller can receive the BLE frame of the specified channel during standby, and it often sleeps to reduce power consumption during standby. The BM-MGR is responsible for specifying the standby channel and switching between waking up and sleeping. The BM-MGR collates the channel described in the received BM-FRM metadata with the standby channel of the BM-CTL. If they match, the BM-FRM is accepted; otherwise, the BM-FRM is discarded.

5.4 BM-CNC

The BM-CNC is responsible for the actual transmission and reception of the BM-FRM. The BM-CNC encapsulates the BM-FRM by Generic Network Virtualization Encapsulation (Geneve), stores it in a User Datagram Protocol (UDP) segment (Fig. 10), and broadcasts it to an Internet Protocol (IP) network. The BLE frame, channel, and TxPower for creating the BM-FRM are included in the transmission command from the BM-MGR. The BM-CNC holds the location of its own ID described in the LC-INF sent from the LC-GEN. When the BR-FRM is transmitted, the BM-CNC describe the location into the BM-FRM header.

The receiving BM-CNC analyzes the received BM-FRM, determines the radio wave interference, and calculates the received radio wave intensity. These details are as described in Sect. 4. The BM-CNC adds the RSSI to the received BLE frame and passes it to the BM-MGR.

5.5 LC-GEN

The LC-GEN generates LC-INF and broadcasts it periodically. The LC-INF comprises data that contain the node IDs and the two-dimensional coordinates of all nodes. Figure 11 displays the JSON data format of LC-INF. There are several ways to implement LC-GEN: one method that follows the predetermined position and time scenario, and another method that uses a multi-agent simulator. In this paper, we implement the LC-GEN as a Python script that sends LC-INF according to the predetermined scenario.

6. Evaluation

6.1 Functional evaluation

We verified whether the proposed BluMoon met the functional requirements of the BLE emulator for software testing. There are many HCI commands and events in the Bluetooth standard. In this study, we implemented six HCI commands and three HCI events, as displayed in Table 2. As a result, it was possible to respond to all five states (Fig. 3) of the LE controller presented in Sect. 2. This indicated that the minimum requirements were met. We omitted the implementation related to encryption and communication parameter settings; however, these can be supported by future implementations. Figure 12 presents the execution result of hciconfig,

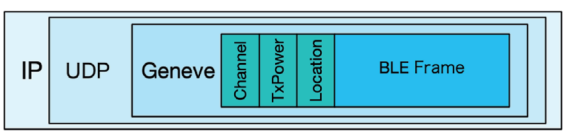

Fig. 10. (Color online) BM-FRM encapsulated in Geneve and stored as UDP segment.

```
{
  "1":{"x":65, "y":75},
  "2":{"x":25, "y":40},
  "3":{"x":25, "y":55},
  "4":{"x":75, "y":90}
}
```

Fig. 11. Example of LC-INF that describes two-dimensional coordinate of node ID 1–4.

Table 2
List of HCI commands and events we implemented in this paper.

Type	Name
Command	LE Set Advertising Parameter
Command	LE Set Advertising Data
Command	LE Set Advertising Enable
Command	LE Set Scan Parameters
Command	LE Set Scan Enable
Command	LE Create Connection
Event	Command Complete
Event	LE Advertising Report
Event	LE Connection Complete

```
$ hciconfig
hci0:   Type: BR/EDR  Bus: VIRTUAL
        BD Address: 00:AA:01:00:00:23  ACL MTU: 192:1  SCO MTU: 0:0
        UP RUNNING
        RX bytes:0 acl:0 sco:0 events:56 errors:0
        TX bytes:1016 acl:0 sco:0 commands:42 errors:0
```

Fig. 12. Result of hciconfig command execution.

which is a command tool for Linux included in BlueZ. Hciconfig verifies the status of the controllers connected by the HCI and control startup and shutdown. The BM-CTL implemented as BluMoon is also displayed by hciconfig and controlled similarly to other controllers. The BM-CTL can be controlled using not only hciconfig, but also programming languages such as C and Node.js. In this way, the emulated BM-CTL can be handled as a Bluetooth controller via the HCI.

6.2 Performance evaluation

To evaluate the BluMoon resource consumption, we measured the central processing unit (CPU) usage during BluMoon execution. The CPU usage was measured at the time of sending and receiving advertisements. We used Group P nodes of the network testbed StarBED[30] for measurement. Each node was connected with 10-Gigabit Ethernet via network switches. Table 3 displays a list of equipment and software used for measurement. In measurement at sending, HCI commands that set and started advertisement were executed. In measurement at receiving, the Node.js script created using noble[31] was executed. Figure 13 presents the measurement results of the CPU usage when sending advertisements. The figure displays the CPU usage according to the number of advertisement transmissions per unit time. Because the BM-HCI, BM-MGR, and BM-CNC were executed as individual processes, the CPU usage is shown for each process.

The CPU usage rates of the BM-MGR and BM-CNC increased as the advertising interval decreased; that is, the transmission frame per unit time increased. In contrast, the BM-HCI only sent the HCI command (*LE Set Advertise Enable*) at the start of the advertisement and

Table 3
Measurement environment.

Chassis	Dell PowerEdge R430
CPU	Intel Xeon E5-2683 v4 (2.1GHz, 16cores) x2
Memory	384 GB
OS	Ubuntu 16.04 Server AMD64
Bluetooth Stack	BlueZ 5.43

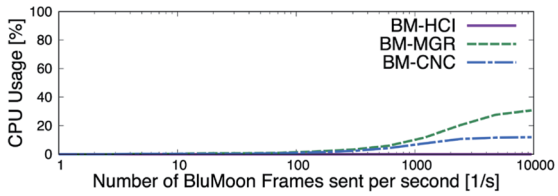

Fig. 13. (Color online) CPU usage when sending advertising packets. The horizontal axis displays the number of advertisements sent per second. The vertical axis represents the CPU utilization. Values are calculated for each process of the BM-HCI, BM-MGR, and BM-CNC.

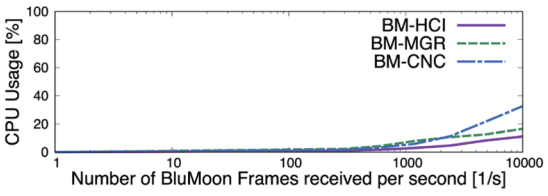

Fig. 14. (Color online) CPU usage rate when receiving advertising packets. The format of the graph is the same as that of Fig. 13.

did not use the CPU during the advertisement transmission. According to the specifications of Bluetooth version 4.2, the minimum advertisement interval was set to 20 ms. In other words, the maximum number of transmission frames per second was 50. Within the specifications, it was possible to execute transmission with low CPU consumption.

Figure 14 presents the measurement results of the CPU usage when receiving advertisements. The figure displays the CPU usage based on the number of received advertisements per unit time. Unlike the case of sending advertisements, the HCI event was generated and sent to the host at every instance of advertisement reception. Therefore, the CPU usage rate increased according to the advertisement reception number for all three processes: BM-HCI, BM-MGR, and BM-CNC. According to the specifications, there is no upper limit on the number of received advertisements. However, the number of advertisement transmitters that exist in the surrounding area is limited in practice. The class of Bluetooth is determined by the maximum radio wave output. Class 1 devices, which have the highest maximum radio wave output, have a reach of approximately 100 m. When advertisement transmitters are arranged every meter in an area of 100×100 m^2, there are approximately 10000 advertisement transmitters. If these advertisement transmitters send an advertisement every 1.28 s, which is the default specification value, the receiver receives 7812.5 advertisements per second. Within this range, it is possible to receive advertisements with the practical consumption of CPU resources.

6.3 RSS

To verify whether BluMoon design requirement 1 was met, we measured the RSS depending on the distance. A transmitter and a receiver were installed to send and receive advertising

packets, respectively, and the RSSI was recorded while changing the distance between the transmitter and the receiver. We measured the RSSI at a transceiver distance of 0.1–30 m, measuring 10 s at each position. The measurement was performed in both the physical and BluMoon emulated environments, and the physical environment was measured in two cases: indoor and outdoor. For the physical environment, we used Raspberry Pi 3 Model B as a transceiver [Fig. 15(a)]. The outdoor measurement was performed in an empty field [Fig. 15(b)], while the indoor measurement was performed in an indoor corridor with a glass wall [Fig. 15(c)].

The measurement results are presented in Fig. 16, which displays the measured value and its logarithmic approximation. Similarly to the physical environment, the BluMoon emulation measurement results also exhibited the characteristics of radio waves, which attenuate according to the distance. The value of BluMoon was approximately halfway between the physical indoor and outdoor measurements. This indicates that the measured values of BluMoon were not unrealistic compared with the physical environment. Note that the results were only for the case of applying the free space radio wave attenuation formula in BluMoon. The radio wave attenuation calculation model used by BluMoon is not limited to this; that is, BluMoon can use other models.

6.4 Radio interference

To verify whether design requirement 2 of BluMoon was satisfied, we measured the reception rate with radio wave interference. To measure the reception rate, the number of received advertising packets transmitted from the transmitter was counted. The advertisement interval of the transmitter was set to 20 ms, which is the shortest interval defined by the Bluetooth 4.2 standard. Ideally, the receiver receives 50 advertising packets per second. Using this as the denominator, the reception rate was calculated from the number of receptions. In addition to the transmitter, up to four interferers were installed. Similarly to the transmitter, these interferers transmitted an advertising packet at 20 ms intervals. The advertising frame

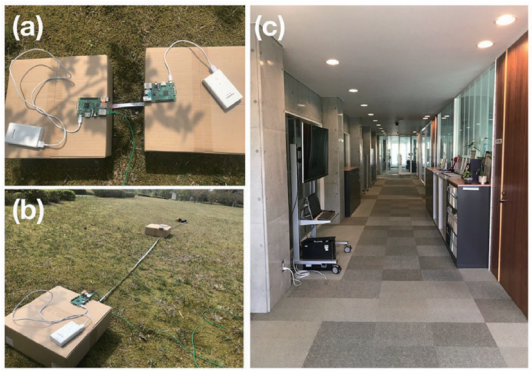

Fig. 15. (Color online) Physical environment of RSSI measurement. (a) Configuration of transmitter and receiver. (b) Outdoor environment. (c) Indoor environment.

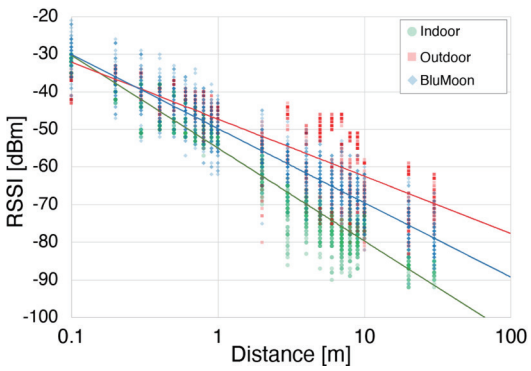

Fig. 16. (Color online) Measurement results of RSSI according to distance. The horizontal axis is logarithmic. The points are measured values, and the lines are their logarithmic approximations.

length is 376 bits. The physical environment was in a steel warehouse that was largely shielded from other radio waves, and Raspberry Pi 3 Model B was used for the transceiver and interferers (Fig. 17). We performed measurements in both the physical environment and the BluMoon emulation environment and compared the results with theoretical values. The theoretical values were derived as follows. When an advertising packet of a certain length was transmitted, the probability of colliding with the advertising packet of one interferometer was

$$P_0 = \frac{2T_{adv}}{T_{ai} + T_d}. \tag{5}$$

Here, T_{adv} is the time required to transmit the advertising packet and can be derived from Eq. (4). T_{ai} is the advertising interval time between sending one advertising packet and the next advertising packet. T_d is the advertising delay, which is a random waiting time inserted after every advertisement transmission. The advertisement delay is used to avoid collisions of advertising packets of the same interval. The advertisement delay is different for each packet transmission, and we set its average value to T_d for use in deriving the theoretical value of the collision probability. The relationship between these variables is illustrated in Fig. 18. T_{adv}, T_{ai}, and T_d have the same values for the transmitter and interferer. When there are N interferers, the probability P_N of colliding with the advertising packet of at least one interferer is the same as that of the complementary event of not colliding with any advertising packet of the interferers. Thus, P_N can be derived as

$$P_N = 1 - (1 - P_0)^N \tag{6}$$

$$= 1 - \left(1 - \frac{2T_{adv}}{T_{ai} + T_d}\right)^N. \tag{7}$$

In this experiment, the advertising frame length is 376 bits. Since the bit rate of BLE is 1 Mbps, T_{adv}, which is the time required to send one advertising frame, is 0.376 ms. The minimum and maximum intervals are set as the advertising parameters. We both set it to 20 ms in this experiment. Therefore, T_{ai} is 20 ms and T_d is 0 ms.

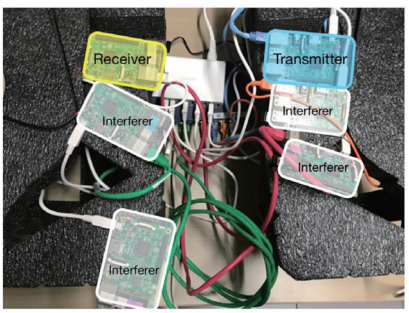

Fig. 17. (Color online) Physical environment of reception rate measurement.

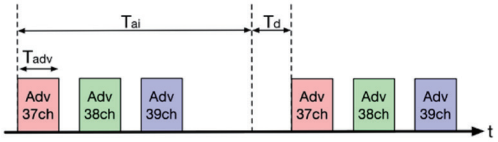

Fig. 18. (Color online) Relationship between T_{adv}, T_{ai}, and T_d in advertising packet transmission.

The measurement results are presented in Fig. 19. Similarly to the theoretical values and physical environment, BluMoon also demonstrated the tendency in which the reception rate decreased as the number of interfering devices increased. The correlation function between the BluMoon measurement and the theoretical value was 0.9799, which indicates that our proposed method successfully imitated radio wave interference. In addition, the correlation between the physical environment measurement and the theoretical value was 0.9241, which was lower than that between the BluMoon measurement and the theoretical value.

The condition of the above experiment is that T_{adv}, T_{ai}, and T_d have the same value for the transmitter and interferer. In the real environment, the value may differ for each interferer. The BluMoon judges interference by monitoring the simultaneous reception of frames in consideration of the frame length. Therefore, the usefulness of our method does not depend on the advertising interval.

In order to demonstrate this, we conducted additional experiments with interferers with different advertising intervals. We performed the experiment with 30 interferers and set their advertising interval to 100, 250, and 1000 ms. These three values are used in Android OS.

In order to compare with the experimental results, we derived the following theoretical formulas. The theoretical value is derived as follows. Let T_{ai} and T_d of the n-th interferer be T_{ai}[n] and $T_d[n]$, respectively. P'_N, which is a probability of colliding with the advertising packet of the n-th interferer, is

$$P'_0[n] = \frac{2T_{adv}}{T_{ai}[n] + T_d[n]}. \tag{8}$$

When there are N interferers with different advertising intervals, the probability P'_N of colliding with the advertising packet can be derived as

$$P'_N = 1 - \prod_{n=1}^{N} \left(1 - P'_0[n]\right). \tag{9}$$

Equation (9) is a general equation for the probability of collision with N interferences. When all interferences have the same advertising interval, Eq. (9) can be simplified to Eq. (7).

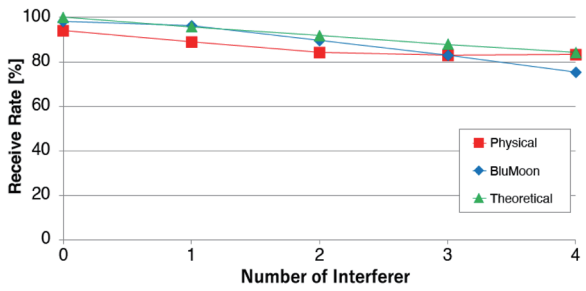

Fig. 19. (Color online) Measurement results of reception rate according to number of interferers.

Table 4 shows a comparison between the experimental results and the theoretical values. As in the previous experiment, the experimental results show that the reception rate decreases as the number of interference frames increases. We speculate that the difference between the theoretical value and the measured value is due to the accuracy of the time handled by the software.

As described in the previous section, the physical environment of radio waves changes owing to various factors, which causes interference fluctuations. Examining how BluMoon emulation handles fluctuations of the physical environment is left for future work.

7. Discussion

7.1 Radio propagation model

In this study, we used a simple model (i.e., free space radio wave propagation loss) to calculate the RSS. Therefore, the environment imitated by BluMoon was an open space with no objects around. However, this is the simplest example. In a physical space, there are various objects, such as the ground, walls, and furniture. Because radio waves reflect from these and take multiple routes, the received radio field intensity fluctuates.[32] By replacing the radio wave propagation model, it is possible to build an emulation environment closer to the physical space. For example, the two-wave reflection model considers the reflected waves on the ground, and the ray tracing method calculates the path from the position of the reflected object.

7.2 Function to avoid interference

In this paper, interference in the BluMoon was evaluated by advertisement. With this mechanism, it is possible to imitate not only advertisement but also communication interference after the establishment of connection. The BLE communication standard has mechanisms to avoid interference such as channel scanning and frequency hopping. However, the BluMoon has not yet implemented these features. This is a future work for the BluMoon.

7.3 Interference other than BLE

In the implementation in this paper, we dealt only with interference between BLEs. However, in the actual environment, there are other 2.4 GHz band radio waves. The principle

Table 4
Measurement results of reception rate with different advertising intervals for interferers.

Number of interferers (100 ms)	Number of interferers (250 ms)	Number of interferers (1000 ms)	Reception rate (theoretical value)	Reception rate (measurement result with BluMoon)
10	10	10	93.46666667	89.303247
5	5	20	95.84444444	93.4400238
5	20	5	94.15555556	90.3251575
20	5	5	90.42222222	84.383995

of interference in BluMoon is to calculate collisions using the frequency band and reception duration. With this information, it is possible to imitate interference with radio waves other than BLE. Interference with other radio waves can be added by defining an interference frame and transmitting it to the BluMoon receiver.

The interference frame requires two types of information: frequency band and duration. In the current BluMoon, the frequency band uses a channel described in the header of BM-FRM, and the duration is calculated from its frame length. However, such calculations cannot be made for interference other than BLE. For example, in the case of Wi-Fi, the channel division method and bit rate are different from BLE. The simplest implementation is to describe the frequency band and duration directly in the interference frame.

There are also radio interference sources other than communication, such as microwave ovens. They become white noise because they interfere over a wide area regardless of the channel. BluMoon can handle such interference by describing that it interferes in the entire band. The detailed design and implementation of interference with radio waves other than BLE shall be future work.

7.4 Deployment to other platforms

In this study, we used Linux and its Bluetooth stack BlueZ to realize our BluMoon concept. The core idea of implementing a virtual controller and emulating with HCI as the interface is applicable not only to Linux and BlueZ but also to other platforms. If the Bluetooth stack is an open source, it can be rewritten to connect to a virtual controller instead of an actual controller. By implementing a virtual controller with the functions introduced in this paper, a BLE emulator can be developed on any platform.

For example, we consider deploying on Android. Versions of Android prior to 4.1 used BlueZ, which is used in our study. However, from 4.2 to the present, another open source Bluetooth stack called Fluoride[33] (formerly Bluedroid) has been used. By rewriting the part of Flouride that is responsible for HCI and implementing a virtual controller, BLE communication can be emulated in Android applications. However, running the Bluetooth emulator on an actual Android device has few advantages. When running the BLE emulator, it will run on the Android emulator. By developing an Android emulator that incorporates customized Fluoride, it will be possible to check the BLE operation of Android applications. In that case, if the host OS of the Android emulator is Linux, The BluMoon Controller we implemented can be utilized.

8. Conclusions

In this paper, we propose BluMoon, a BLE emulator, for software system testing. We designed, implemented, and evaluated BluMoon. For its design, we imposed the following requirements:

(1) calculating the RSS for each frame and
(2) simulating radio interference.

We proposed an emulator design that replaces the controller with the HCI as the boundary. From the viewpoint of layers higher than the HCI, the emulator executes the same software as

for physical BLE devices. To meet requirement 1, we proposed a design in which each controller calculates the RSSI when receiving a BLE frame. The RSSI is derived by calculating the radio wave attenuation from the distance between a transmitter and a receiver. To meet requirement 2, we designed an interference simulation that discards frames if they arrive on the same channel at the same time.

We implemented BluMoon based on this design and evaluated its effectiveness through various measurement experiments. In the functional evaluation, we demonstrated that BluMoon can be used in the same way as a physical BLE controller. In the performance evaluation, we measured the CPU usage and demonstrated that BluMoon can be executed with reasonable resource consumption. In the comparison experiment with physical environments, we demonstrated that it is feasible to use our design to imitate the RSS and radio wave interference.

BluMoon can be used for testing various types of BLE applications, such as proximity detection, sensor data collection, and stamp rally. Various testing methods and theories related to software development have been developed in the field of software engineering. However, for systems that are affected by physical space, testing methods are still evolving. In other research, we used the BluMoon emulation results to reproduce the radio wave environment in a physical box for testing with a physical mobile device.[34] In future work, we plan to further develop the testing method of the system considering the surrounding environment.

Acknowledgments

We are grateful to the staff of NICT StarBED, the member of Tan and Lim laboratory, and Dr. Yuuki Takano for supporting and discussing our research.

References

1 Bluetooth: https://bluetooth.com (accessed July 2020).
2 T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena: SIGCOMM Demonstration **14** (2008) 527.
3 K. Akashi, T. Inoue, S. Yasuda, Y. Takano, and Y. Shinoda: NETorium: Proc. 12th Asian Internet Engineering Conf. (2016) 25–32. https://doi.org/10.1145/3012695.3012699
4 T. Yumura, K. Akashi, and T. Inoue: IPSJ SIG Technical Report Ubiquitous Computing System (UBI) 2017 (2017) 1–7 (in Japanese). http://id.nii.ac.jp/1001/00182996/
5 T. Yumura, K. Akashi, and T. Inoue: 2018 IEEE Int. Conf. Pervasive Computing and Communications Workshops (PerCom Workshops) (2018) 860–865. https://doi.org/10.1109/PERCOMW.2018.8480294
6 Apple: Getting Started with iBeacon Version 1.0 (2014).
7 Google: Eddystone format, https://developers.google.com/beacons/eddystone/ (accessed July 2020).
8 BlueZ: http://www.bluez.org/ (accessed July 2020).
9 Google: Android Studio, https://developer.android.com/studio/ (accessed July 2020).
10 Apple: Technical Note TN2295: Testing Core Bluetooth Applications in the iOS Simulator, https://developer.apple.com/library/archive/technotes/tn2295/index.html (accessed July 2020).
11 BLE Peripheral Simulator: https://github.com/WebBluetoothCG/ble-test-peripheral-android/ (accessed July 2020).
12 LightBlue Explorer: https://itunes.apple.com/jp/app/lightblue-explorer/id557428110 (accessed July 2020).
13 Google: Wear OS by Google, https://wearos.google.com/ (accessed July 2020).
14 Apple: Xcode, https://developer.apple.com/xcode/ (accessed July 2020).
15 J. Albert, T. F. Bissyandé, Y.-D. Bromberg, S. Chaumette, and L. Réveillere: Ubipan: 2010 Int. Conf. Complex, Intelligent and Software Intensive Systems (CISIS), IEEE (2010) 774–778. https://doi.org/10.1109/CISIS.2010.167

16 K. Tsuda, H. Suzuki, K. Asahi, and A. Watanabe: 2014 7th Int. Conf. Mobile Computing and Ubiquitous Networking (ICMU) (2014) 78–79. https://doi.org/10.1109/ICMU.2014.6799065
17 M. Okada and H. Suzuki: Trans. Inf. Process. Soc. Jpn., Consum. Device Syst. (CDS) **8** (2018) 34 (in Japanese). http://id.nii.ac.jp/1001/00189476/
18 T. Smith, S., Saroiu, and A. Wolman: Proc. 7th Int. Conf. Mobile Systems, Applications, and Services (2009) 41–54. https://doi.org/10.1145/1555816.1555822
19 C. Wang, Z. Shao, and M. Fujise: IEEE Int. Symp. Communications and Information Technology (ISCIT) 2004 **2** (2004) 1118. https://doi.org/10.1109/ISCIT.2004.1413893
20 W. Liu, E. De Poorter, J. Hoebeke, E. Tanghe, W. Joseph, P. Willemen, M. Mehari , X. Jiao, and I. Moerman: IEEE Trans. Wireless Commun. **16** (2017) 1755. https://doi.org/10.1109/TWC.2017.2654256
21 R.Beuran, J. Nakata, T. Okada, L. T. Nguyen, Y. Tan, and Y. Shinoda: Proc. Int. Conf. Advanced Information Networking and Applications (AINA) (2008) 223–228. https://doi.org/10.1109/WAINA.2008.111
22 R. Beuran, L. T. Nguyen, T. Miyachi, J. Nakata, K. Chinen, Y. Tan, and Y. Shinoda: Global Telecommunications Conf., 2009, GLOBECOM 2009 (2009) 1–6. https://doi.org/10.1109/GLOCOM.2009.5426182
23 R. Beuran, J. Nakata, Y. Tan, and Y. Shinoda: IEICE Trans. Commun. **95** (2012) 2892. https://doi.org/10.1587/transcom.E95.B.2892
24 R. Beuran, S. Yasuda, T. Inoue, Y. Takano, T. Miyachi, and Y. Shinoda: EAI Endorsed Trans. Self-Adaptive Syst. **1** (2015) e1. https://doi.org/10.4108/icst.tridentcom.2015.259856
25 K. Akashi, T. Inoue, R. Beuran, and Y. Shinoda: IEICE Trans. B **98** (2015) 357 (in Japanese).
26 J. Mailnen: mac8011 hwsim, https://wireless.wiki.kernel.org/en/users/Drivers/mac80211_hwsim (accessed July 2020).
27 H. Tazaki, F. Uarbani, E. Mancini, M. Lacage, D. Camara, T. Turletti, and W. Dabbous: Proc. 9th ACM Conf. Emerging Networking Experiments and Technologies (2013) 217–228. https://doi.org/10.1145/2535372.2535374
28 H. Mano and S. Saruwatari: Trans. Inf. Process. Soc. Jpn. **55** (2014) 1541 (in Japanese). http://id.nii.ac.jp/1001/00101161/
29 Ixia: Network Emulator II, https://www.ixiacom.com/products/network-emulator-ii/ (accessed July 2020).
30 T. Miyachi, T. Nakagawa, K.-I Chinen, S. Miwa, and Y. Shinoda: Int. Conf. Testbeds and Research Infrastructures (Springer, 2011) 43–58.
31 Sandeep Mistry: noble, https://github.com/sandeepmistry/noble (accessed July 2020).
32 J. Neburka, Z. Tlamsa, V. Benes, L. Polak, O. Kaller, L. Bolecek, J. Sebesta, and T. Kratochvil: Radioelektronika (RADIOELEKTRONIKA), 2016 26th Int. Conf. (2016) 121–125. https://doi.org/10.1109/RADIOELEK.2016.7477344
33 Fluoride Bluetooth stack: https://android.googlesource.com/platform/system/bt/+/master#fluoride-bluetooth-stack (accessed October 2020).
34 T. Yumura, M. Enomoto, K. Akashi, F. Hirose, T. Inoue, S. Uda, T. Miyachi, Y. Tan, and Y. Shinoda: Proc. 2018 ACM Int. Joint Conf. and 2018 Int. Symp. Pervasive and Ubiquitous Computing and Wearable Computers, ACM (2018) 476–479. https://doi.org/10.1145/3267305.3267568