Multiconnection Scheduling With Fair Resource Management for Scalable Bluetooth Low-Energy Networks

Moonbeom Kim and Jeongyeup Paek, *Senior Member, IEEE*

***Abstract*—Bluetooth low energy (BLE) is a key wireless technology for the Internet of Things (IoT), supporting low-power, multidevice communication. However, managing and scheduling multiple BLE connections under tight timing constraints remain significant challenges. The Bluetooth specification lacks explicit strategies for efficient multidevice coordination, often leading to unfair resource allocation and connection event interruptions due to *resource overlap* among peripherals. To address this problem, we propose enhanced multi-BLE management (EMBLEM), a novel scheduling scheme that treats multiple connections as a unified resource list and dynamically adjusts allocations based on traffic intensity. By leveraging event preemption and connection subrating, *EMBLEM* enables flexible resource allocation that accommodates more peripherals while ensuring fairness. To further improve efficiency, we introduce a bitmap-based state tree for memory-efficient resource tracking and also connection event extension to reduce resource wastage and improve throughput. We evaluate *EMBLEM* on a 51-node testbed against state-of-the-art scheduling schemes and popular BLE stacks. Experiment results show that *EMBLEM* eliminates resource overlap, improves stability, and significantly enhances system fairness, throughput, and scalability under diverse operating conditions.**

***Index Terms*—Bluetooth low energy (BLE), fairness, Internet of Things (IoT), multiconnection scheduling.**

I. Introduction

BLUETOOTH low energy (BLE) has established itself as one of the key wireless technologies for the Internet of Things (IoT), owing to its support for multidevice connectivity, low power consumption, broad compatibility with diverse hardware platforms, and ease of use. BLE has been adopted widely across various domains, from personal devices (*e.g.,* smartphones, desktops, tablets, wearables, and home appliances) [3] to diverse systems such as smart infrastructure [4], [5], [6], [7], healthcare [8], [9], vehicular [10], [11], environment sensing/monitoring [12], [13], and localization or tracking [14], [15], [16]. Consequently, BLE-enabled devices are expected to reach 7.7 billion units in 2029 [3].

As BLE is becoming deeply pervasive in our everyday life and its applications are being diversified, the Bluetooth Special Interest Group (SIG) [17] is actively working to enhance the capabilities of BLE and expand it into large-scale networks. For instance, Bluetooth SIG has continually introduced a range of features, such as additional PHYs (LE 2M and coded 500k/125k) to increase the speed and range of communication, extended and periodic advertising to reduce congestion and improve throughput, and connection subrating to reduce connection update delays while maintaining power efficiency, as well as LE audio to provide high-quality audio streaming over BLE. Moreover, they recently embarked on new projects to support data rates of up to 8 Mb/s and the use of unlicensed midband spectrum (6 GHz) to achieve lower latency.

Despite these ongoing efforts to advance BLE technology, the Bluetooth core specification [18] only defines the constraints and procedures for establishing a connection with a single device and does not specify how multiple devices should be scheduled and managed. Scheduling multiple connections without conflict and ensuring resource fairness are essential for efficiently sharing limited wireless resources to enable reliable communication and improve connectivity. Their importance becomes particularly evident in large-scale networks such as smart homes, smart buildings, and industrial systems. However, systematically managing multiple schedules with precise time synchronization is highly challenging in the BLE link layer, which involves time-critical operations. Furthermore, reallocating resources and rescheduling to ensure quality of service (QoS) under dynamic network conditions adds complexity that has not been addressed yet.

Most commercial BLE devices implement their own connection scheduling schemes and policies due to the absence of an explicit standard for multidevice coordination in the specification. For instance, representative BLE stacks such as NimBLE [19], Nordic [20], and Zephyr [21] commonly adopt simple mechanisms for determining the *anchor point* (*i.e.,* connection starting point) that prioritizes simplifying the process and rapidly establishing the connections over optimal resource allocation. Specifically, when a new device is discovered, they calculate the anchor point to establish the

Received 8 September 2025; revised 4 November 2025; accepted 30 November 2025. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MSIT) (No. RS-2024- 00359450), and also by the Institute of Information & Communications Technology Planning & Evaluation(IITP)-ITRC(Information Technology Research Center) grant funded by the Korea Government (MSIT) (IITP-2025-RS-2022- 00156353). An earlier version of this work was presented in part at ACM SenSys'23 [DOI: 10.1145/3625687.3628411] and ICTC Workshop on Big Data in 2023 [DOI: 10.1109/ICTC58733.2023.10392966]. *(Corresponding author: Jeongyeup Paek.)*

The authors are with the Department of Computer Science and Engineering, Chung-Ang University, Seoul 06974, Republic of Korea (e-mail: mbkim@cau.ac.kr; jpaek@cau.ac.kr).

Digital Object Identifier 10.1109/JIOT.2025.3639570

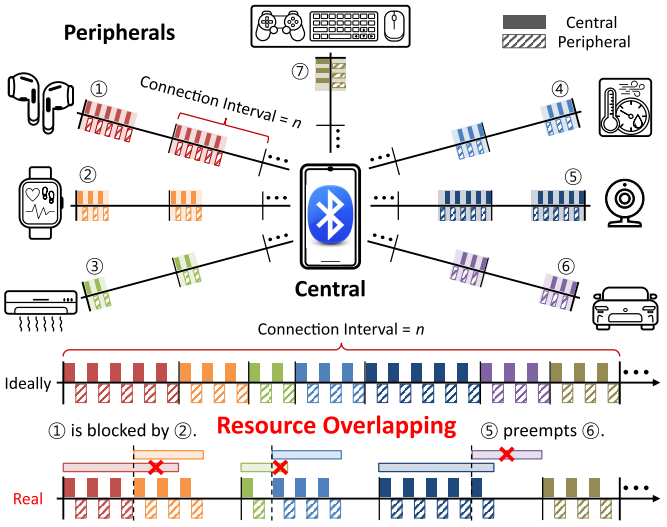

Fig. 1. Resource overlap problem causing resource unfairness, performance instability, QoS degradation, and disconnection.

connection as soon as possible (with a minimum delay of 1.25 ms) without considering the schedules and requirements of already and newly connected devices.

Aforementioned approach commonly used by most, if not all, commercial devices today can result in a significant problem in multiconnection scenarios due to *resource overlap* among peripherals. For example, consider the scenario illustrated in Fig. 1, where multiple peripherals are connected to a central and exchange data with the same connection interval n. We generally expect that all peripherals are fairly allocated the resources they require and are accurately scheduled without resource interference or contention. However, as shown at the bottom of Fig. 1, some connection events can be interrupted either by the start of others (*e.g.,* connection event of ① is blocked by ②) or by the current ongoing ones (*e.g.,* ⑤ preempts the resources for ⑥). These lead to unfair resource usage, which may cause performance degradation or instability, QoS failure, and even disconnections in the worst cases. Moreover, this problem becomes more critical as the amount of transmitted data and the number of connected devices increase.

Although there are many prior studies that aim to enhance BLE capabilities, efforts to address the resource overlap problem remain significantly limited despite its severity. Most existing works have primarily focused on energy efficiency [22], [23], interference avoidance and congestion control [24], [25], optimal connection parameter decision [26], [27], and specific applications or systems [28], [29], [30]. While a few studies have attempted to mitigate the resource overlap problem [31], [32], [33], they continue to face resource unfairness and low QoS, and these issues intensify as network size and/or load increase, limiting the practicality and scalability of the system.

To address these problems, we propose an enhanced multi-BLE management (*EMBLEM*) scheme. *EMBLEM* eliminates resource overlap by managing multiple connection schedules as a *unified resource list* with precise time synchronization and responds to changes in traffic intensity by dynamically adjusting resource allocation. Notably, *EMBLEM* adopts the *connection subrating* feature first introduced in Bluetooth 5.3 [34] for efficient scheduling and fast connection, and also leverages *connection event preemption* to enable flexible resource utilization. Using these features, we devise a new resource allocation scheme, *cyclic scheduling*, and *zigzag allocation with resource segmentation*. These further improve utilization and schedulability by mitigating resource fragmentation and allow the accommodation of more peripherals while ensuring resource fairness. For efficient resource tracking, we propose a novel *bitmap-based resource state tree* that effectively reduces memory usage on constrained embedded devices. This enables *EMBLEM* to manage resources without collision while identifying the optimal resources for each connection. In addition, we propose *connection event extension* to reduce resource wastage and improve transmission efficiency. With these approaches, we aim to make *EMBLEM* adaptable to various systems while enhancing scalability for large-scale BLE networks.

The contributions of this work are summarized as follows.

1) Propose *EMBLEM*, a novel BLE scheduling scheme that eliminates resource overlap while ensuring fairness. It contains cyclic scheduling and zigzag allocation with segmentation to enhance resource utilization and schedulability, a bitmap-based state tree for efficient management, and connection event extension to improve performance.
2) Implement *EMBLEM* on real embedded devices, refine the incomplete connection subrating feature of NimBLE to comply with the specification, and open our source code.[1]
3) Conduct extensive evaluation on a large-scale testbed with one central and 50 peripherals, demonstrating that *EMBLEM* guarantees fairness across all peripherals and provides scalability while outperforming other approaches.

The remainder of this article is organized as follows. We first provide background on BLE in Section II. Then, we describe the problem that we are addressing in Section III and discuss related work inspiring this study in Section IV. We present the design of the *EMBLEM* in Section V and evaluate *EMBLEM* in Section VI. Finally, we conclude in Section VII.

II. Background

In this section, we provide a brief background on key BLE features essential for understanding the proposed *EMBLEM*.

A. Connection Establishment and Data Communication

BLE devices are divided into *central* and *peripheral* roles.[2] The central is primarily responsible for managing connection procedures and settings, and controlling key link layer operations to optimize communication efficiency and resource utilization. These include initiating, maintaining, and updating connections, as well as determining their parameters, all of

[1] https://github.com/moonbeom-kim/EMBLEM

[2] As of Bluetooth v5.3, the Bluetooth SIG replaced the legacy role-based device names "master" and "slave" with "central" and "peripheral" [17], [34].

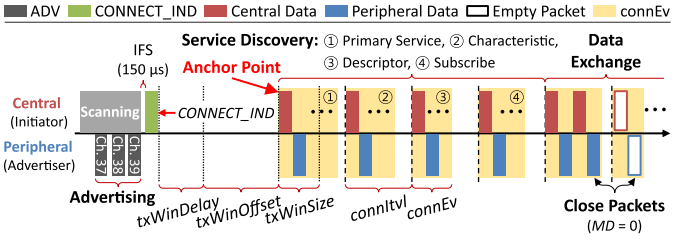

Fig. 2. Procedure from connection initiation to data exchange.

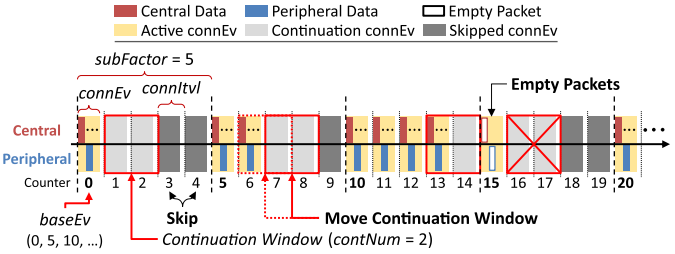

Fig. 3. Operation procedures of connection subrating.

which are closely related to connection scheduling. Fig. 2 illustrates the procedure from connection establishment to data exchange.

1) Connection Establishment: To establish a connection between BLE devices, an *advertiser* periodically broadcasts an advertising (ADV) packet using three advertising channels (*i.e.,* ch 37, 38, and 39), and an *initiator* scans the three channels sequentially during a scanning interval to receive them. When the initiator receives an ADV packet, it transmits a CONNECT_IND packet as a connection request to the advertiser after an inter frame space (IFS) of 150 μs. Once the advertiser accepts the connection request, the initiator and the advertiser are referred to as *central* and *peripheral*, respectively.

The CONNECT_IND contains essential information for establishing and maintaining the connection, such as the "transmit window" and "connection interval (connItvl)." Particularly, the "transmit window" allows the central to transmit the first data channel packet, which determines the *anchor point* for future "connection events (connEv)." After the completion of the CONNECT_IND, the transmission of the first packet starts within the "transmit window size (txWinSize)" following a delay equal to the sum of the "transmit window delay (txWinDelay)" and the "transmit window offset (txWinOffset)," as shown in Fig. 2. The txWinDelay is typically set to 1.25 ms, and the txWinOffset shall be a multiple of 1.25 ms within the range from 0 ms to connItvl. When the peripheral receives a packet from the central for the first time, it adopts the reception time as the *anchor point*.

2) Communication and Service Discovery: Once a connection is established, the central and peripheral wake up periodically at every connItvl starting from the anchor point and exchange packets during the connEv. Every connEv begins with a transmission of either data or an empty packet from the central, and the reception time of this packet serves as the reference for determining the timing of the next connEv on the peripheral side. The peripheral has to transmit a response after IFS if it receives a packet from the central. In addition, to provide meaningful service to users, the central commonly should discover and subscribe to the services available through the peripheral (*e.g.,* heart rate monitoring for healthcare).

The packet header includes the "more data (MD)" field, which indicates whether the device has more data to transmit. If either device sets the MD or the other device receives it, the central may transmit another packet to maintain the connEv, and the peripheral should wait to receive the next packet after transmitting its response. If neither device has set the MD, the last packet from the peripheral closes the connEv. In addition, if a packet is not received or a CRC error is detected twice consecutively within a single connEv, the event is closed.

B. Connection Subrating

The connection subrate feature in Bluetooth 5.3 and later [34] allows the users to rapidly switch the radio duty cycle based on traffic intensity changes while saving power. There are three key parameters: the "subrate factor (subFactor)" allows the central and peripheral to skip some connEv and communicate only at specific connEv, and the "subrate base event (baseEv)" indicates the starting point of connection subrating. It can reduce energy consumption because the two devices do not use the radio during other connEvs except baseEv. The "continuation number (contNum)" enables both devices to wake up for data exchange in the specified number of subsequent connEv if either device receives a data packet (non-empty packet) during the current connEv. This allows them to quickly adapt to variable packet rates.

For example, suppose that the initial baseEv, subFactor, and contNum are 0, 5, and 2, respectively, in Fig. 3. The central and peripheral wake up at specific baseEvs (*i.e.,* 0, 5, 10,...), with a cycle of 5 based on the event counter 0, to exchange data or empty packets. If contNum is zero, all connEv except baseEv are skipped. Otherwise, the connection subrating creates a "continuation window" of size contNum (*e.g.,* red box around connEv 1 and 2). At connEv 0, one or more data packets are exchanged, so the next two connEvs (1 and 2) are designated as continuation events. If no data packets are exchanged during these two connEvs, the continuation window is closed at connEv 2. On the other hand, when at least one data packet is transmitted within the continuation window (*e.g.,* connEv 6), the window shifts by one event ($6 \rightarrow 7$). This is repeated until the next baseEv (*e.g.,* connEv 10–14). When only empty packets are exchanged in the baseEv (*i.e.,* neither device has data to transmit, as in connEv 15), the continuation window becomes inactive (16–17), and all subsequent events until the next baseEv are skipped.

C. Basic Connection and Connection Subrate Updates

1) Basic Connection Update: Central can adjust the anchor point through connection parameter updates to reschedule connections. As shown in Fig. 4(a), the central starts the update procedure by transmitting an LL_CONNECTION_UPDATE_IND to the peripheral, including new parameters (*i.e.,* txWinSize, txWinOffset, connItvl, and so on) and an "*instant*" value. The "instant"

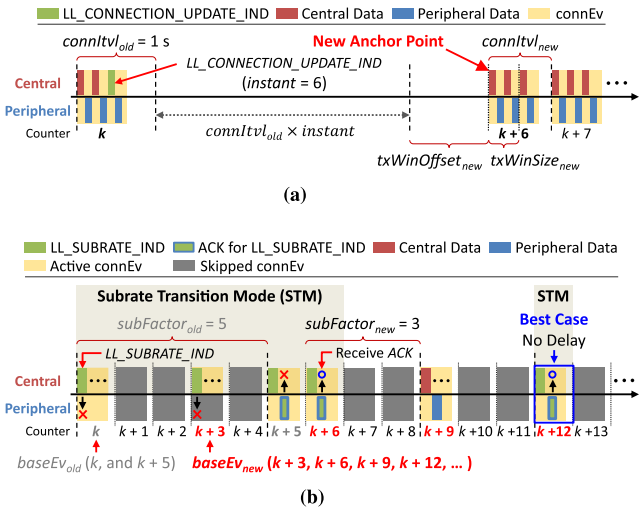

Fig. 4. Update procedures for connection rescheduling. (a) Anchor point adjustment through basic connection update. (b) Base event change through connection subrate update.

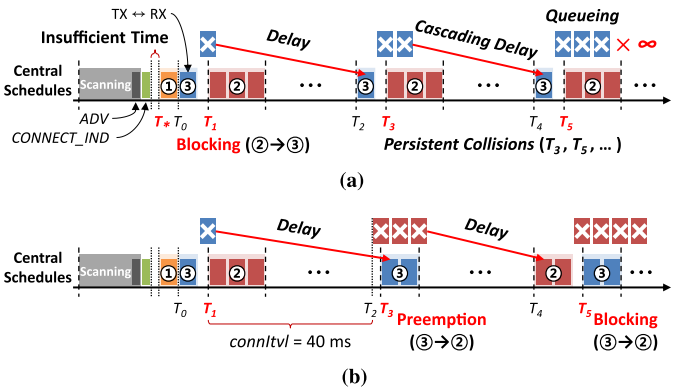

Fig. 5. Examples of resource overlap in a multiconnection scenario. (a) connEv blocking. (b) Resource preemption.

indicates the number of connItvl between the current connEv and the time when the updated values take effect (*i.e.,* $connItvl_{old} \times instant$), with a minimum of 6. At the connEv corresponding to the instant, the central transmits a packet to set a *new anchor point* within the $\text{txWinSize}_{\text{new}}$ after the $\text{txWinOffset}_{\text{new}}$, while the peripheral waits to receive it. However, the larger the current connItvl, the longer both devices may have to wait before the updated values take effect. For example, if $\text{connItvl}_{\text{old}}$ is 1 s, completing the update requires a minimum mandatory delay of 6 s.

2) Connection Subrate Update: Fig. 4(b) illustrates an example of the subrate update procedure. This starts with the central transmitting an LL_SUBRATE_IND to the peripheral, including new parameters (*i.e.,* subFactor, baseEv, and so on). Then, the central enters *subrate transition mode* (*e.g.,* connEv k) and retransmits the LL_SUBRATE_IND for all connEvs corresponding to the old and new subrate parameters (*e.g.,* $k+3$, $k+5$, and $k+6$). This minimizes the possibility of failure caused by update request or acknowledgment (ACK) loss (*e.g.,* k, $k+3$, or $k+5$), ensuring a successful update. Once the peripheral receives the update packet, it responds with an ACK to the central and switches to the new subrate parameters. The central, upon receiving a response from the peripheral, leaves subrate transition mode (*e.g.,* $k+6$) and discards events associated with the old parameters. After that, the central and the peripheral wake up at $\text{baseEv}_{\text{new}}$ (*e.g.,* $k+9$) based on the $\text{subFactor}_{\text{new}}$ of 3 and exchange data packets. This can mitigate the update delay because both devices immediately apply the update values after receiving the update request or response, without the instant. For example, in the best case (*e.g.,* $k+12$), if the update request and response are exchanged on the first attempt, delay does not occur. Functional attributes and advantages of subrating provide potential for new approaches to BLE connection (re)scheduling.

III. Problem and Motivation

Lack of multiconnection management strategies in the specification leads to a significant problem of *resource overlap*, which can degrade performance and stability of BLE systems. In this section, we investigate how the existing mechanisms cause resource overlap, which motivates our work.

A. Resource Overlap Problems

Commercial BLE devices employ their own connection mechanisms and scheduling policies. For instance, popular BLE stacks NimBLE [19], Nordic [20], and Zephyr [21] basically adopt a simple mechanism focusing on rapid connection establishment, as mentioned in Section I, without considering the next schedule of other connections. They manage multiple connections based on the following *scheduling rules*.

1) A connection should be guaranteed the minimum resources to exchange one pair of maximum-size packets.
2) A new connection must not interfere with any already scheduled connections.
3) If two connEv schedules for different connections collide, the one with the oldest last connEv is given higher priority.

Although this approach enables fast connection establishment without disrupting other schedules and aims to provide (roughly) fair access opportunities for all connections, resource unfairness arises due to *connEv blocking* or *resource preemption* caused by overlapping connection schedules. This leads to serious issues that adversely affect system performance.

For example, Fig. 5 illustrates the resource overlap problems from the scheduling perspective of the central when a new connection joins in a multiple devices scenario. Suppose that all connections have the same connItvl of 50 ms, and the central is currently connected to two peripherals ① and ②.

1) ConnEv Blocking: When the central discovers a new peripheral ③ by receiving an ADV, it attempts to establish a connection with the target device ③) as soon as possible. However, after CONNECT_IND, there may be insufficient time to wait for txWinDelay + txWinOffset (minimum delay of 1.25 ms) or to guarantee the minimum resources due to the subsequent connection schedule [*i.e.,* ③ cannot start at T_* due to ①, as shown in Fig. 5(a)]. To avoid collision with connection ① under *scheduling rules 1 and 2*, the central schedules the new connection ③ at the earliest available idle

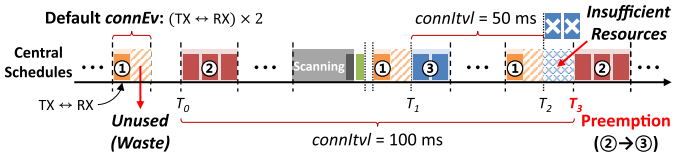

Fig. 6. Example of resource overlap in connEv length configuration.

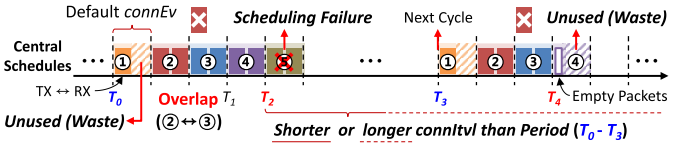

Fig. 7. Example of CSS failure.

time (*i.e.,* T_0) following it. Nevertheless, if connection ③ has multiple packets to transmit beyond the minimum resources guarantee, it can experience a collision with another adjacent schedule (*e.g.,* connection ②). In such a case, *connEv of connection ③ is blocked by the start of connection ②* at T_1, and the remaining data is delayed until the next connEv.

Upon the end of a connEv, the central schedules the next one according to connItvl of the corresponding connection. For example in Fig. 5(a), after connection ③ is blocked at T_1, its next connEv is immediately scheduled for time T_2. However, *persistent collisions* occur between connections ② and ③ in the subsequent schedules (*e.g.,* T_3 and T_5) due to their identical connItvl, consistently blocking connection ③. This results in *cascading delays* in the following packets of connection ③ and accumulation in the transmission queue, potentially leading to buffer overflow in the worst case.

2) Resource Preemption: In the same scenario as before, consider that connection ② has a connItvl of 40 ms, as shown in Fig. 5(b). After connection ③ is blocked at T_1, its next connEv is scheduled for time T_3. Then, connection ② transmits successfully at T_1 with sufficient resources, and the central attempts to schedule its next connEv to start at time T_2. However, it collides with connEv schedule of connection ③. If connection ③ prevents the minimum resource guarantee of connection ②, the central prioritizes the connection whose last connEv was scheduled the longest time ago in accordance with *scheduling rule 3*. In this example [see Fig. 5(b)], the last execution time of connection ③ is T_0, which is earlier than that of connection ② at T_1. Accordingly, connection ③ is given higher priority even though its next connEv time (T_3) is later than connection ② (T_2), while connection ② is deferred until its next time (T_4). Although both delayed and new packets in connection ② can be transmitted in a batch at T_4, this poses potential risks of collision with other schedules (*e.g.,* T_5) and *cascading delays due to persistent collisions*.

B. Handling Resource Overlap in Representative BLE Stacks

BLE stacks such as NimBLE, Nordic, and Zephyr introduce some features to mitigate the resource overlap problem.

1) Minimum connEv Length Configuration: They support a configuration that sets the minimum connEv length, ensuring that all connections are guaranteed at least the resource corresponding to the specified connEv length. However, when connections operate with different connItvls, resource overlap can occur. In Fig. 6, consider a scenario where the minimum connEv length is set to allow the exchange of at least two data packet pairs, and connItvl of connection ② is 100 ms (*e.g.,* T_0–T_3), while connections ① and ③ operate with connItvl of 50 ms. Connection ③ can be established at time T_1 and successfully transmits all data without interference from other connections. After completing communication with connection ③, the central attempts to schedule its next connEv for time T_2. However, the central fails to schedule it and cannot ensure the minimum resources for connection ③ since connection ② preempts the resources under the *scheduling rule 3*.

2) Connection Strict Scheduling: Particularly, NimBLE provides a scheduling option for multiple devices, called connection strict scheduling (CSS). In the CSS mode, a predefined period (*e.g.,* [T_0–T_3] in Fig. 7) is divided into several time slots, and connections are scheduled *sequentially* within that period. CCS mandatorily sets the connItvl of all connections to the period and places an anchor point for each connection at a slot boundary. It allocates a fixed number of slots per connection (*i.e.,* the minimum connEv length) to ensure uniform resource allocation across all connections. Ideally, this enables CSS to schedule as many connections as the total slot count divided by the per-connection slot allocation, without resource overlap. However, enforcing a fixed connItvl for all connections can either result in insufficient resources for a connection to satisfy its QoS requirement or exchanges of unnecessary empty packets and resource waste due to a mismatch between the actual data rate requirement and the enforced connItvl (*e.g.,* connection ④ at T_4), especially when the minimum connEv length is large. Moreover, if the connItvl range (min/max) of a newly discovered peripheral does not include the scheduling period (*e.g.,* connection ⑤ at T_2), that is, if the scheduling period is not within the connItvl range of the new device, CSS rejects the connection with the device.

Aforementioned two approaches aim to provide fair and sufficient resources for all connections by specifying the minimum connEv length, but they still remain vulnerable to the resource overlap problem. Moreover, they apply the same connEv length to all connections and do not allow runtime adjustments to configurations (*i.e.,* the minimum connEv length and the slot count/duration in the scheduling period). This makes it difficult to accommodate diverse services with their respective requirements and maintain performance under network dynamics. If the data volume to be exchanged exceeds the preassigned resources (*e.g.,* connection ② in Fig. 7), resource overlap can still occur. Conversely, if the data volume is small, connEv may be closed early once the transmission is complete, resulting in unnecessary resource waste (*e.g.,* connection ① in Figs. 6 and 7).

These problems motivate us to propose *EMBLEM*, a multiconnection management scheme designed to eliminate the resource overlap problem completely.

IV. Related Work

The resource overlap problem in BLE is a critical challenge that must be solved to guarantee stable performance while

enabling the expansion of BLE into large-scale networks. A few prior studies in the literature have attempted to mitigate or address the resource overlap problem in BLE networks.

A. Connection Parameter Tuning Schemes

Initial efforts were made to assess the severity of connEv interruptions caused by the overlap and to mitigate this problem. Chen et al. [23] propose a method that adjusts connItvl to meet service requirements while balancing the tradeoff between data exchange delays and energy consumption, both of which result from event interruptions. This aims to reduce the incidence of event interruptions, but adjusting connItvl alone cannot resolve the problem without anchor point control. Moreover, they focus on minimizing energy consumption rather than solving the resource overlap problem. Dian et al. [31] propose a simple approach to prevent periodic collisions between connEv schedules at the least common multiple (LCM) of the connItvls by adding a unique delay to the anchor points of each connection. This can eliminate event interruptions at the LCM, but it neither completely resolves resource overlap nor considers resource fairness. In addition, the available anchor points become limited as the number of connections increases, hindering the scalability of the systems.

B. Dynamic Connection Scheduling Schemes

Recently, a few prior works have attempted to address the resource overlap problem while dynamically adapting to changes in network conditions or service requirements. Park et al. [32] proposed BLEX that estimates the traffic load of each peripheral based on its historical link usage and reallocates resources accordingly by adjusting anchor points through connection updates when the idle time between consecutive connEv is under 1 or over 2 ms. However, this approach can trigger frequent update procedures in a dynamic network environment and require a significant time for update completion due to the mandatory delay constraint in Bluetooth. This delay becomes more severe as the connItvl increases, making it difficult to rapidly respond to changes in traffic load, which can lead to significant performance degradation. The work most closely related to ours is a study by Li et al. [33], which proposed RT-BLE that allocates resources for each connection by considering its expected retransmissions. It also uses collision tree-based scheduling to manage connections and adopts the connection subrating feature to achieve fast resource update. However, to precalculate the required resources and expected retransmissions, it must have prior knowledge of the configuration for each connection, such as packet details (*i.e.,* size, number, and loss rate) between connected devices and service requirements (*i.e.,* end-to-end latency and worst case latency percentile). These constraints prevent RT-BLE from being applied to services that are unknown a priori, limiting the diversity of applications and the scalability of networked systems.

C. Limitations of Prior Work and Distinction to EMBLEM

All of the aforementioned prior works remain vulnerable to the resource overlap problem and suffer from performance loss, unfair resource allocation, and lack of scalability or flexibility. Furthermore, they are evaluated in constrained settings, either on a small-scale testbed with up to eight BLE peripheral devices or through simulations under ideal conditions. This makes it difficult to assess the performance and applicability in real environments with a large number of connected devices. In contrast, *EMBLEM* eliminates the resource overlap problem and efficiently manages multiple connections in large-scale networks while ensuring resource fairness by dynamically adapting to varying network conditions and service requirements without relying on predefined information. Furthermore, *EMBLEM* is evaluated on a 51-node testbed through an extensive set of real experiments to demonstrate its effectiveness.

V. EMBLEM Design

This section presents *EMBLEM*, an enhanced multiconnection BLE management scheme for scalable BLE systems. We first provide an overview of *EMBLEM* and then elaborate on the key components for *multiconnection scheduling* and *fair resource management*. Finally, we explain a method to prevent anchor point drift for precise synchronization in a unified schedule and also the connection event extension mechanism to improve efficiency and performance.

A. Overview

EMBLEM eliminates resource overlap to achieve resource fairness and adapts to network dynamics while guaranteeing QoS. It is implemented in, and operates only within, the link layer of the central. There is no additional communication overhead such as exchanging extra packets to share status (*e.g.,* link condition) between the connected devices, nor does it require any newly defined specific host controller interface (HCI) commands or functions to deliver application-level QoS requirements to the lower layer. By removing interlayer and interdevice dependencies, *EMBLEM* can accommodate various devices that provide different services.

Fig. 8 illustrates the architecture of our proposed *EMBLEM*, which consists of seven main modules.

1) *Connection initiator* estimates the threshold of connItvl for guaranteeing the requirements of services and emulates the behavior of connItvl using subFactor.
2) *Resource manager* periodically measures the fluctuation in connEv duration of each connection, estimates the necessary resources using the exponentially weighted moving average (EWMA), and then dynamically reallocates resources.
3) *Multiconnection scheduler* introduces *cyclic scheduling* and *zigzag allocation with resource segmentation* to improve schedulability and resource utilization while minimizing resource waste caused by fragmentation.
4) *Bitmap-based resource state tree* allows not only to efficiently search and manage resources but also to effectively reduce memory usage for resource management.
5) *Anchor point and subrating calculator* computes key parameters, such as txWinOffset and baseEv, to assign and maintain a connection at precise scheduling points

based on the resources identified in the "bitmap-based state tree."

6) *Connection update manager* adjusts resources or reschedules connections via "connection" and/or "subrate" updates based on the resources to be newly assigned.
7) *Connection event controller* incorporates *connEv extension* to optimize the transmission and improve the performance.[3]

B. Connection Interval Emulation for Connection Initiation

To easily manage repeated schedules or resources for multiple connections, the most intuitive and effective approach is to organize them into a list or table based on the *hyperperiod*, which is calculated as the LCM of their transmission or connection intervals (*i.e.,* connItvl in BLE) [23], [31], [32], [33], [35], [36]. However, when connections have diverse connItvls, the hyperperiod increases, leading to higher memory and computational overhead for scheduling, with the effect more pronounced for coprime connItvls. To avoid this issue, we refer to an idea in BLEX [32] and adapt it to suit *EMBLEM*.

EMBLEM fixes the connItvl of all connections to 7.5 ms (minimum value in the specification) and emulates the original interval ($\text{connItvl}_{\text{orig}}$) required by each connection using subFactor. To this end, although it needs to know the precise value of $\text{connItvl}_{\text{orig}}$, obtaining it at the link layer is difficult because it is set as minimum and maximum values at the application layer and passed to the link layer via an HCI command. However, considering that these values are typically represented as the demanded latency and that most users generally rely on one of the following two approaches when setting the connItvl to ensure service requirements, the threshold of connItvl for certain services at the link layer can be estimated.

1) Set the minimum connItvl to the best-case and the maximum connItvl to the worst case.
2) Set both minimum and maximum values of connItvl to either the best-case or worst case.

When *EMBLEM* receives an ADV packet from a peripheral, the "connection initiator" first sets the maximum connItvl delivered by the upper layer to $\text{connItvl}_{\text{orig}}$ and calculates the *factor* (f) as (1). Then, as shown in (2), subFactor is determined as 2^n, which makes $7.5\,\text{ms} \times subFactor$ the closest to $\text{connItvl}_{\text{orig}}$

$$f = \lceil connItvl_{orig} \,/\, 7.5\,ms \rceil. \tag{1}$$

$$subFactor(f) = \begin{cases} f, & f = 2^n \\ 2^{\lfloor \log_2^f \rfloor}, & f \neq 2^n \text{ and } f - 2^{\lfloor \log_2^f \rfloor} \leq 2^{\lceil \log_2^f \rceil} - f \\ 2^{\lceil \log_2^f \rceil}, & f \neq 2^n \text{ and } f - 2^{\lfloor \log_2^f \rfloor} > 2^{\lceil \log_2^f \rceil} - f. \end{cases} \tag{2}$$

For example, if $\text{connItvl}_{\text{orig}}$ is 250 ms, the central and peripheral wake up and interact at 240-ms intervals based on a connItvl 7.5 ms and subFactor 32. We set the range of n from 0 to 9, which supports up to 512 subFactor[4]. This enables *EMBLEM* to provide connItvl coverage from 7.5 to 3840 ms, ensuring the requirements of all connections across the entire connItvl range (7.5–4000 ms) defined in the specification [18].

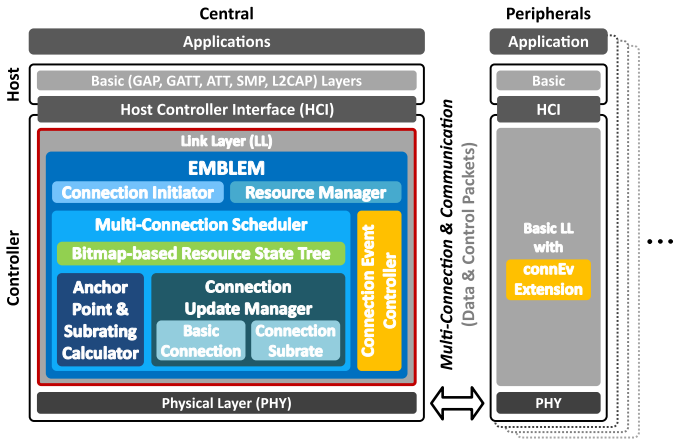

Fig. 8. Overview of the proposed *EMBLEM*.

C. Bitmap-Based Resource State Tree

Resources for connection are allocated as multiple time-slotted units, referred to as *resource slots*. In multiconnection scheduling, however, the resource slot size entails a tradeoff between system performance and overhead. While a smaller resource unit enables fine-grained control of resources and potentially less wasted resources, it requires more memory space for management and leads to higher computational overhead. Conversely, a larger resource unit reduces memory usage and bookkeeping overhead but at the cost of more resource wastage. Appropriately balancing them is an important challenge, especially in constrained embedded systems. To this end, we propose a *bitmap-based resource state tree* that enables fine-grained resource handling with minimal memory usage while supporting efficient resource searching and management.

Fig. 9 illustrates an example of the bitmap-based resource state tree. The resource state tree is structured as a perfect binary tree with ten levels and 1023 nodes, and is implemented as an array. In the array, the root node is located at the first element (index 0), and the elements at indices 511–1022 represent the leaf nodes. Each node occupies 1 byte, which represents a "resource block" containing six "time resource slots." The lower 6 bits of 1 byte represent a bitmap indicating their allocation states, where a bit value of 1 denotes an allocatable slot (white box), and 0 is an allocated slot (gray box). Depending on the occupancy of slots, the nodes have one of three states: empty (green, all bits are 1), full (red, all bits are 0), or partial (orange, mixed bits). We set the time resource slot size to 1.25 ms, which corresponds to the minimum unit used for both the connItvl and the tx/rx slot

[3]To further enhance BLE performance, we add only the "connEv extension" to the peripheral, without modifying any part of its legacy link layer. All *EMBLEM* operations are performed exclusively on the central side.

[4]At the maximum subFactor, the central operates with 512 and the peripheral with 256 to maintain scheduling without violating the specification and compromising compatibility.

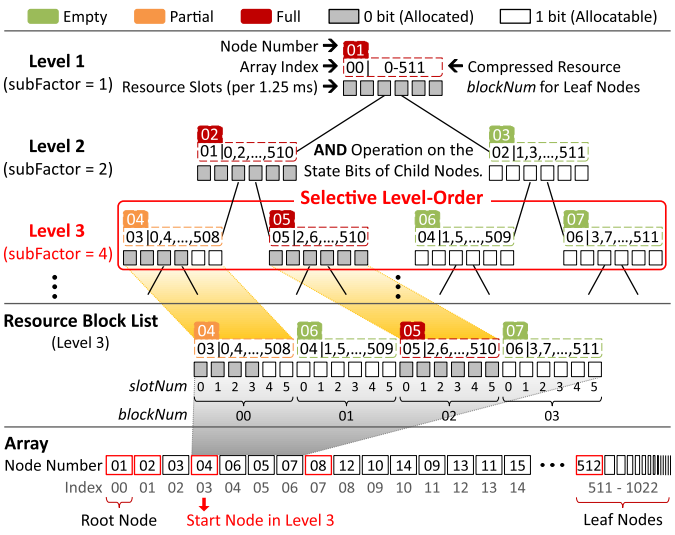

Fig. 9. Example of a *bitmap-based resource state tree.*

pair in BLE[5]. We also adopt it as the minimum unit for resource allocation and connection scheduling in *EMBLEM*, enabling fine-grained resource management and maximizing utilization with minimal waste. Furthermore, the bitmap-based resource state tree requires only 1023 bytes of memory for resource management, which is sufficiently small considering the capacity of commercial BLE devices (*e.g.,* Nordic nRF52840 DK has 1-MB flash and 256-kB RAM [37]).

The resource state tree is hierarchically organized, with each level mapped to a specific subFactor (level $= 1 + \log_2 subFactor$). Nodes at each level indicate the availability of resources that repeat periodically at intervals defined by that subFactor across the entire resource space. That is, the array elements corresponding to the nodes at each level form a "resource block list" for a specific subFactor, where blocks are virtually numbered (*blockNum*) from 0 to 511. For instance, the leaf nodes at level 10 (array indices 511–1022) represent the complete list of resource blocks (blockNum 0–511), which corresponds to consecutive time resources for the maximum hyperperiod of 3840 ms (*i.e.,* 1.25 ms $\times$ *six slots* $\times$ *subFactor of 512*) in *EMBLEM*.

At upper-level nodes, the resource states of the leaf node blocks are compressed based on the subFactor corresponding to each level by performing "bitwise AND operations" on the child nodes. In the array, the offset between the resource blocks of child nodes (left and right) is equal to the subFactor of their parent node, and their indices can be calculated as follows:

$$\begin{aligned} \text{index}_{\text{left}} &= \text{index}_{\text{parent}} + subFactor \\ \text{index}_{\text{right}} &= \text{index}_{\text{parent}} + (subFactor \times 2)\,. \end{aligned} \tag{3}$$

For example, node 2 ($\text{index}_{\text{parent}} = 1$) at level 2 indicates the compressed resource states of blocks spaced at intervals of 2 across all time resources (*i.e.,* leaf node blocks 0, 2,..., 510). Its child nodes 4 ($\text{index}_{\text{left}} = 3$) and 5 ($\text{index}_{\text{right}} = 5$) cover the leaf node blocks 0, 4,..., 508 and 2, 6,..., 510, respectively. As shown in Fig. 9, node 4 has a "partial" state with some slots allocated (bits 000011), while node 5 is in a "full" state, indicating that no resource slots are available (000000). The bitwise AND operation between the two child states (000011 and 000000) results in parent node 2 being in a full state (000000). If both child nodes (*e.g.,* 6 and 7) are in an "empty" state (111111) with all slots unallocated, their parent node (*e.g.,* 3) is also in an empty state.

This hierarchical aggregation of resource state from the leaf nodes to the root based on the subFactor enables *EMBLEM* to allocate resources efficiently and quickly by searching only the nodes at the level corresponding to the target subFactor.

1) Selective Level-Order Resource Search: The "multiconnection scheduler" in *EMBLEM* allocates and manages resources for each connection based on the subFactor emulated by the connection initiator. For example, suppose that the given subFactor for some connection is 4. EMBLEM first identifies the resource block list corresponding to this subFactor in the array. For the resource block list, the start index ($\text{index}_{\text{start}}$) and the end index ($\text{index}_{\text{end}}$) are given by the following equation:

$$\begin{aligned} \text{index}_{\text{start}} &= subFactor - 1 \\ \text{index}_{\text{end}} &= \text{index}_{\text{start}} \times 2. \end{aligned} \tag{4}$$

[5]The basic physical channel (LE 1M PHY) in Bluetooth is divided into time slots for tx/rx, each 0.625 ms in length [18].

Algorithm 1 Bitmap-Based Resource State Tree Update

Input: *subFactor, blockNum*$_{\text{start}}$*, slotNum, and consSlots*

/* Update the leaf nodes (level 10)*/
$slotNum_{\max} \leftarrow 6$
$subFactor_{level} \leftarrow 2^{level_{\max}-1}$
$\text{index}_{\text{start}}, \text{index}_{\text{end}} \leftarrow getResourceBlockList(subFactor_{\text{level}})$
repeatCount
$\leftarrow subFactor_{\text{level}} \,/\, getRepeatCount(subFactor_{\text{level}}, subFactor)$
for $0 \le count <$ repeatCount **do**
  offset $\leftarrow subFactor \times count$
  $\text{index}_{\text{leaf}} \leftarrow \text{index}_{\text{start}} + blockNum_{\text{start}} +$ offset
  $slotNum_{\text{start}} \leftarrow slotNum$
  $slots \leftarrow consSlots$
  while *slots* >0 **do**
    $slotNum_{\text{end}} \leftarrow min(slotNum_{\text{start}} + slots, slotNum_{\max})$
    slotMask $\leftarrow 2^{slotNum}_{\max} - slotNum_{\text{start}} - 2^{slotNum}_{\max} - slotNum_{\text{end}}$
    bitwiseXOROperation ($\text{index}_{\text{leaf}}$, slotMask)
    $\text{index}_{\text{leaf}} \leftarrow getNextIndex(\text{index}_{\text{leaf}}, \text{index}_{\text{start}}, \text{index}_{\text{end}})$
    $slots \leftarrow slots - (slotNum_{\text{end}} - slotNum_{\text{start}})$
    $slotNum_{\text{start}} \leftarrow 0$
  end while
end for

/* Update the internal nodes (level 9–1)*/
while $subFactor_{\text{level}} > 1$ **do**
  $subFactor_{\text{level}} \leftarrow subFactor_{\text{level}} \,/\, 2$
  $\text{index}_{\text{start}}, \text{index}_{\text{end}} \leftarrow getResourceBlockList(subFactor_{\text{level}})$
  $consBlocks \leftarrow min(\lceil (slotNum + consSlots) \,/\, slotNum_{\max} \rceil, subFactor_{\text{level}})$
  repeatCount $\leftarrow subFactor_{\text{level}} \,/\, getRepeatCount(subFactor_{\text{level}}, subFactor)$
  for $0 \le block < consBlocks$ **do**
    offset $\leftarrow subFactor \times count$
    $\text{index}_{\text{parent}} \leftarrow \text{index}_{\text{start}} +$
    $(blockNum_{\text{start}} \bmod subFactor_{\text{level}})$ + offset
    for $0 \le block < consBlocks$ **do**
      $\text{index}_{\text{left}}, \text{index}_{\text{right}} \leftarrow getChilds(\text{index}_{\text{parent}}, subFactor_{\text{level}})$
      bitwiseADNOperation ($\text{index}_{\text{left}}$, $\text{index}_{\text{right}}$, $\text{index}_{\text{parent}}$)
      $\text{index}_{\text{parent}} \leftarrow getNextIndex(\text{index}_{\text{parent}}, \text{index}_{\text{start}}, \text{index}_{\text{end}})$
    end for
  end for
end while

Accordingly, the range of the resource block list with subFactor 4 is from $\text{index}_{\text{start}}$ 3 to $\text{index}_{\text{end}}$ 6, as shown at the bottom of Fig. 9. *EMBLEM* then sequentially searches the resource blocks within the identified list (the middle part of Fig. 9) to find the largest region of consecutively allocatable resources. For example, it identifies whether a block state is empty or full by performing a bitwise AND operation using a slotMask whose bit value is "111111." If the block is neither empty nor full (*i.e.,* partial state), a "bit-shift operation" counts the number of consecutive allocatable slots within it. When the search procedure is complete, the starting block number ($blockNum_{\text{start}}$) of the found resources, the number of the first allocatable slot (*slotNum*) in that block, and the total number of available consecutive slots (*consSlots*) are returned. In this example scenario, the return values for $\text{blockNum}_{\text{start}}$, slotNum, and consSlots are 0, 4, and 8, respectively. If there are multiple regions of the same length, the first one found is selected.

2) Partial Path Propagation: Algorithm 1 presents the pseudocode describing the bitmap-based resource state tree update procedure from the leaf nodes to the root node. The update is performed partially along the paths of nodes whose resource states will be changed due to allocation or deallocation, rather than updating the entire tree.

To facilitate understanding, suppose that the resource state tree has four levels, and a connection with subFactor of 4 is allocated four consSlots starting from slotNum 0 within $\text{blockNum}_{\text{start}}$ 0, such as node 4 at level 3 in Fig. 9. We first set the initial values for the deepest level to update the resources state in a bottom-up manner (Lines 2–5), $slotNum_{\text{max}}$ denotes the number of slots per block (*i.e.,* 6), and $subFactor_{\text{level}}$ represents the subFactor at the current level being updated. The corresponding resource block list ($\text{index}_{\text{start}}$ to $\text{index}_{\text{end}}$) can be derived using (4). repeatCount is the number of times the resources should be updated in the block list according to the subFactor, and it is given by $\lceil subFactor_{\text{level}} / subFactor \rceil$.

In accordance with repeatCount, only the target leaf nodes are updated sequentially throughout the entire resource space (Lines 6–19). At the beginning of each repetition, the target leaf node index (*i.e.,* $\text{index}_{\text{leaf}}$) is determined using an offset based on subFactor (Lines 7 and 8). Updates to the consecutive *slots* are performed on a block basis using a "bitwise XOR operation" with slotMask (Lines 11–18). If the number of slots to be updated starting at $slotNum_{\text{start}}$ exceeds the block boundary (*i.e.,* $\text{slotNum}_{\text{max}}$), $slotNum_{\text{end}}$ is limited to $\text{slotNum}_{\text{max}}$ to prevent the update range from exceeding that (Line 12). Based on the current update range ($\text{slotNum}_{\text{start}}$ to $\text{slotNum}_{\text{end}}$), slotMask is generated (*e.g.,* slotNum 0–4: bits 111100), and a bitwise XOR operation is performed with the target leaf node to update the resource state (Lines 13 and 14). $\text{index}_{\text{leaf}}$ is then incremented to update the remaining slots in the current repetition (Lines 15–17). Note that, if the *next index* exceeds the range of the block list (*i.e.,* $\text{index}_{\text{end}}$), it is wrapped around to $\text{index}_{\text{start}}$ (Line 15–33) for our proposed *cyclic scheduling* scheme (more details in Section V-D).

The changed resource state is propagated toward the root one level at a time, with $\text{subFactor}_{\text{level}}$ being halved at each step (Lines 22–36). At each level, *consBlocks* denotes the number of consecutive blocks (*i.e.,* parent nodes) that require updating due to resource state changes in the lower-level nodes (Line 25). Since the block list at the lower level is folded in half at the upper level, the relative position of $\text{index}_{\text{parent}}$ can be derived by applying the modulo to $\text{blockNum}_{\text{start}}$ with respect to $\text{subFactor}_{\text{level}}$, along with offset determined by each repetition count (Line 29). The indices of child nodes ($\text{index}_{\text{left}}$ and $\text{index}_{\text{right}}$) of $\text{index}_{\text{parent}}$ can be calculated according to (3). A "bitwise AND operation" is then performed between child nodes to compress their resource state into the parent node (Line 32).

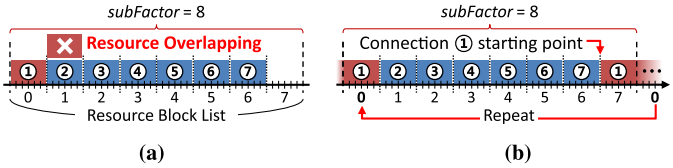

Fig. 10. *Cyclic scheduling* to eliminate the limitation of linearity. (a) Limitation of linearity. (b) Cyclic scheduling.

Through the above update procedure, the resource states allocated to the given connection (*e.g.,* node 4) in this example are propagated sequentially from the leaf nodes to the root node along the path of nodes "$8 \rightarrow 9 \rightarrow 4 \rightarrow 2 \rightarrow 1$." This enables partial updates to be performed only on nodes where state changes are required. Furthermore, this ensures consistency of resource states throughout the tree, thereby enabling efficient tracking of resources per subFactor across the entire resource space.

D. Multiconnection Scheduling and Resource Allocation

Since the resource requirements for each connection defined at the application layer cannot be known at the link layer [18], [32], "multiconnection scheduler" in *EMBLEM* assigns an initial time resource of 7.5 ms to a newly connected peripheral, including the *guard slot* (*GS*). The GS mitigates the impact of unexpected extra packets caused by retransmissions or sudden increases in data volume. Its size is set to 2.5 ms, which is the time required to transmit a max-sized packet (261 bytes) and receive a link layer ACK (minimum 10 bytes) with the default 1-Mb/s PHY of Bluetooth [18]. If the size of allocatable resources found through the bitmap-based resource state tree is equal to or larger than the initial resource size, the scheduler assigns 7.5 ms to the new connection. Otherwise, it gradually reduces the initial resource in 1.25-ms increments until it either finds sufficient resources or its size falls below 2.5 ms.

The most intuitive approach for scheduling multiple connections with assigned resources is to define a linear scheduling space based on the LCM of the connItvls [23], [31], [32], [33] and to allocate resources sequentially in connection order. However, this approach has limitations due to the linearity of the resource space. The sequential allocation without considering the diversity of connItvls can lead to scheduling or resource update failures resulting in resource contention, which, in turn, causes performance degradation or unintended disconnections. For example, in Fig. 10(a), suppose that all seven connections have subFactor 8, resources for each of them are sequentially

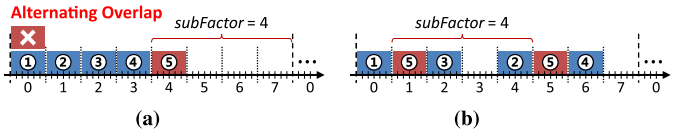

Fig. 11. Alternated overlapping problem and *zigzag allocation*. (a) Alternated overlapping. (b) Zigzag without segmentation.

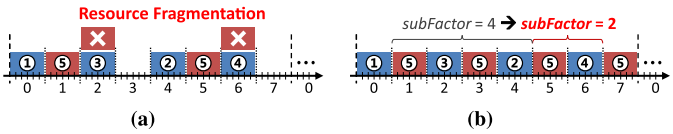

Fig. 12. Fragmentation problem and *resource segmentation*. (a) Resource fragmentation. (b) Zigzag with segmentation.

allocated in the linear space based on the connection order from ① to ⑦, and connection ① requires an update for additional resource allocation to transmit another packet. However, due to the sequential allocation, connection ② immediately follows ①, resulting in no available continuous resources for additional allocation. Moreover, even when sufficient available resources (*e.g.,* blockNum 7) are present in the resource block list, the resource update fails due to the linearity limitation of the scheduling space.

To address these problems, we propose a new resource allocation scheme, *cyclic scheduling* with *zigzag allocation and resource segmentation*.

1) Cyclic Scheduling: To overcome the linearity limitation of the legacy scheduling space, we propose a *cyclic scheduling* scheme that improves resource utilization and schedulability. It treats the resource list as a circular structure by leveraging the periodic nature of connEv, which regularly repeats based on the subFactor. As shown in Fig. 10(b), the scheduler adjusts the start time of ① to the last blockNum 7 and allocates additional resources from blockNum 0, thereby ensuring sufficient resources for ①. Cyclic scheduling enables flexible allocation by creating new available resource space for peripherals and also provides an opportunity for new connections.

Fig. 11(a) illustrates another problem that can occur in a sequential allocation approach, disregarding the diversity of connItvl. In the same scenario as before, consider that the central attempts to establish a connection to the peripheral ⑤ using a subFactor of 4 instead of 8. However, due to the LCM relationship between the subFactors of the existing and new connections, the schedule for ⑤ alternately collides with that of connection ①, which can result in connection establishment failure despite the availability of sufficient resources (*e.g.,* blockNum 5–7). Even if the connection is successfully established, peripherals ① and ⑤ can experience performance degradation due to resource contention, and one of them may be disconnected in the worst case. Consequently, unless a request with a higher subFactor than the current one is received, these issues persist, leading to a restriction of schedulability and a reduction in network connectivity.

2) Zigzag Allocation: To address this problem, we propose the *zigzag resource allocation* in the circular resource list. It allocates the resources to a connection starting from the first resource block if the block list corresponding to the target subFactor is empty [e.g., ① in Fig. 11(b)]. Otherwise, based on the size and position of the available space found through the resource state tree, it calculates the position (p) of resources to be allocated to the new connection as follows:

$$\begin{aligned} p &= 2^{\lfloor \log_2(consSlots\ /slotNum_{\max}) \rfloor} \times slotNum_{\max} \\ &= \lceil (blockNum_{\text{start}} \times slotNum_{\max} + slotNum\ )/p \rceil \times p \\ &= p \bmod (subFactor \times slotNum_{\max}) \end{aligned} \tag{5}$$

from which the $\text{blockNum}_{\text{start}}$ ($\lfloor p/slotNum_{\max} \rfloor$) and slotNum ($p$ mod $slotNum_{\max}$) are derived. This arranges the connections in a zigzag manner so that the interval between n connections becomes $subFactor\ /2^{\lceil \log_2 n \rceil}$, thereby securing space for future connections with a smaller subFactor than the current one. For example in Fig. 11(b), connection ② is placed at a subFactor of 4 from ①, and connection ③ is positioned midway between ① and ② with a subFactor of 2 from each. As a result, it can resolve the issue caused by the LCM (*e.g.,* ① and ⑤) and accommodate more connections with different connItvls. If the available space is smaller than twice the required size, the resources are allocated from the beginning of the space without division.

However, the zigzag allocation can fragment a large space consisting of contiguous resources into smaller units, leading to external fragmentation. In this context, we can consider a scenario where peripheral ⑤ needs additional resources, as shown in Fig. 12(a). It fails to reallocate resources and ultimately causes contention with connections ③ and ④. This can lead to wasted fragmented resources and overall system performance degradation and disconnections.

3) Resource Segmentation: We extend the zigzag allocation by introducing *resource segmentation* to mitigate potential problems caused by resource fragmentation. If suitable contiguous resources are not found, the current subFactor is halved, and the required resources are segmented and allocated to the available spaces, repeating the process until resources are found or the subFactor reaches zero. As shown in Fig. 12(b), resource segmentation enables successful allocation within the fragmented resource space, improving resource utilization and schedulability while reducing waste. Note that resource segmentation is applied only for resource updates to avoid the overhead of unnecessary segmentation since the required resources for a new connection cannot be known in advance.

E. Connection Establishment and Schedule Synchronization

Once resources for the connection are assigned, *EMBLEM* performs the scheduling procedure in two steps: *anchor point calculation* and *baseEv determination*. The schedule is synchronized based on the client's tick (T_{base}) at which it starts.

For example in Fig. 13, suppose that the $\text{blockNum}_{\text{start}}$, slotNum, and consSlots of the resources assigned to a connection with subFactor 4 are 0, 0, and 6, respectively. In the first step, *EMBLEM* calculates the *txWinOffset* to align the *anchor point* with the *slotNum* 0 (*i.e.,* resource starting point) of the resources assigned to the connection, as shown in Fig. 13. To do this, it calculates the starting point of the txWinOffset (T_{off}) depending on the ADV reception time (T_{ADV}) from the target

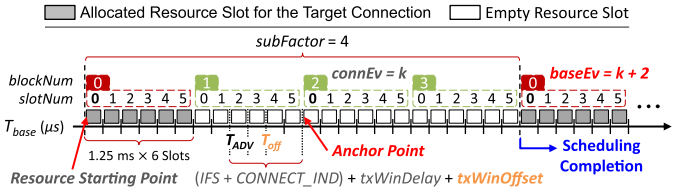

Fig. 13. Connection establishment and synchronization.

peripherals, mapping this time onto the scheduling timeline with T_{base}. Specifically, T_{off} is given by $(\lceil (T_{\text{ADV}} + \text{IFS} + \textit{CONNECT\_IND} + \textit{txWinDelay}) / 1.25\,\text{ms} \rceil \times 1.25\,\text{ms}) \bmod 7.5\,\text{ms}$, where under the 1-Mb/s PHY, the transmission delay of the CONNECT_IND is 352 μs, the txWinDelay is fixed at 1.25 ms, and 1.25 and 7.5 ms are the time capacities of a resource slot and a block, respectively. The txWinOffset is obtained by subtracting T_{off} from *slotNum* × 1.25 ms, and 7.5 ms is added if the result value is negative. Then, *EMBLEM* establishes a connection to the target peripheral by transmitting a CONNECT_IND that includes the calculated txWinOffset.

When *EMBLEM* receives a connection subrate command from the application, it calculates the *baseEv* based on the *current connEv count* (k). If blockNum_k corresponding to k is greater than $\text{blockNum}_{\text{start}}$, baseEv is adjusted by adding the offset ($\text{blockNum}_{\text{start}} + \text{subFactor} - \text{blockNum}_k$) to blockNum_k (*e.g.,* $k+2$ in Fig. 13). In addition, when subFactor is greater than 1, *EMBLEM* sets *contNum* to 1, allowing continuous data transmission by temporarily utilizing the empty resources following the allocated ones (*e.g.,* all slots except the gray shaded ones in Fig. 13), thereby enabling rapid adaptation to sudden traffic increases caused by higher data volume or retransmissions. Then, *EMBLEM* transmits an LL_SUBRATE_IND to establish the connection subrating with the peripheral, completing the connection scheduling.

1) Prevention of Scheduling Distortion: The connection schedule can be distorted as a result of crystal oscillator inaccuracy in embedded devices. For the nRF52840 DK operating with a 32 768-Hz crystal, the connItvl of 100 ms corresponds to 3276.8 ticks, resulting in a deviation of 0.8 ticks per connItvl. Over time, the accumulation of timing deviations may cause the anchor point to drift, potentially collapsing the entire schedule. This effect becomes more critical as the number of connections and the density of the schedule increase. To prevent the distortion and maintain stable schedules, we leverage the *window widening* feature in Bluetooth [18]. Window widening is the extension of the reception window around the anchor point to compensate for clock skew between the central and the peripheral, which is achieved by the peripheral waking up slightly earlier and remaining in listening mode a little longer. By leveraging this feature, *EMBLEM* adjusts the anchor point for next connItvl with a resolution of *1 tick* (≈31 μs) when it deviates from the expected time, enabling precise scheduling. We do not discuss the details of window widening in this work. However, given that the oscillator accuracy of the nRF52840 DK is ± 500 ppm according to the product specification [38], and that the bound of deviation in Bluetooth is 16 μs [18], the minimum range of the window widening calculated over the entire connItvl is 47 μs, which is sufficient for our proposed scheme.

F. Adaptive Resource Management and Update

To dynamically adjust resource allocation based on traffic intensity, the *resource manager* in *EMBLEM* periodically tracks the fluctuation in connEv duration of each connection (*i.e.,* resource usage) and estimates the resource requirement using the exponentially weighted moving average (EWMA). In BLE, when two consecutive CRC errors occur, the link layer immediately terminates the ongoing connEv even if more packets remain to be transmitted. As this causes inaccuracies in resource estimation, connEv that is closed early due to CRC errors is excluded from measurements.

When the resource usage exceeds the allocated resources (including GS), the resource manager searches the bitmap-based resource state tree in three phases to allocate additional resources corresponding to the excess amount plus GS.

1) The manager first checks whether there is a contiguous allocatable region after the resources allocated to a connection. If a sufficient idle region is available, the manager allocates the additional resources solely by updating the resource state tree without connection rescheduling.
2) Otherwise, the manager releases the allocated resources and searches the block list corresponding to subFactor of the connection to find the largest available region.
3) If the size of that region is insufficient for the required resources, the manager repeats the search at a higher-level block list through resource segmentation until an allocatable resource is found or subFactor reaches zero.

EMBLEM reschedules the connection to reflect the updated resource allocation using a *basic connection* and *connection subrate updates*. If slotNum of the updated resources differs from the existing one, *EMBLEM* performs the *basic connection update* to adjust the anchor point to the new slotNum. To minimize a mandatory delay for update due to the *instant* (as mentioned in Section II-C) and achieve a fast connection update, we apply the update procedure recommended in the Bluetooth specification [18]. The specification allows the central to temporarily change the subrate parameter before a connection update through the *connection subrate update*. Accordingly, *EMBLEM* first changes subFactor and contNum of the connection to 1 and 0 by transmitting the LL_SUBRATE_IND, which enables the reduction of the mandatory delay to as low as 45 ms[6]. In the same manner as described in Section V-E, *EMBLEM* then calculates the txWinOffset to align the anchor point with the new slotNum and determines the new baseEv corresponding to $\text{blockNum}_{\text{start}}$ of the updated resources. Finally, it completes the rescheduling for the resource update by sequentially transmitting LL_CONNECTION_UPDATE_IND and LL_SUBRATE_IND, which include txWinOffset and new baseEv (with subFactor and/or contNum when changed), respectively. When only blockNum is changed, while slotNum

[6]When BLE uses the connection subrating feature, the mandatory delay is given by $\textit{connItvl} \times \textit{subFactor} \times \textit{instant}$ (*minimum of 6*).

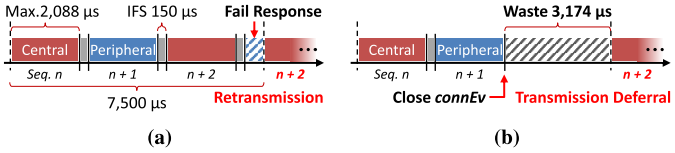

Fig. 14. Problems that degrade transmission efficiency. (a) Redundant retransmission in BLE. (b) Transmission deferral in NimBLE.

is unchanged, *EMBLEM* determines the new baseEv and completes the connection rescheduling only through subrate update for changed parameters.

Conversely, when the resource usage is less than the allocated resources excluding the GS, *EMBLEM* releases the resources by updating the state tree. If the connection resources are segmented, it reduces them while gradually increasing subFactor back to its original value.

G. Connection Event Extension

In Bluetooth, the peripheral shall always respond to a packet from the central, while the central may transmit upon receiving a packet from the peripheral [18]. If the remaining time in connEv is insufficient to transmit the next packet, the central defers the transmission to the next connEv and closes the current one. However, as shown in Fig. 14(a), if the remaining time in connEv after the central's packet (*e.g., seq.* $n + 2$) is insufficient to receive a response, the central misses the response and retransmits the packet in the next connEv. This redundant retransmission degrades transmission efficiency. To prevent this, the central in NimBLE estimates the duration of the packet exchange and calculates the remaining time until the end of connEv [19]. If this time is insufficient to complete the packet exchange, it closes the current connEv early and defers the transmission to the next one, as shown in Fig. 14(b). However, this approach can lead to significant resource waste and QoS degradation due to the premature termination of connEv and transmission deferral.

To improve transmission efficiency, we propose the *connection event extension* (*connEv extension*), which draws inspiration from the *multislot packets* feature in the Bluetooth specification [18]. In the same scenario as above, connEv extension calculates the idle time from now to the start of the next connection schedule to assess whether additional packet exchange is possible. If there is sufficient time for the exchange, it maintains the current connEv until either no more data remains to be transmitted or the other schedule begins. The connEv extension neither expands nor reschedules the actual schedule but instead prolongs the end time of the connEv without any impact on existing schedules. This allows the central and peripheral to treat consecutive connEvs as a single extended connEv, reducing the overhead of repeatedly scheduling connEvs and mitigating performance degradation and resource inefficiency caused by redundant retransmission or transmission deferral. When either a previously scheduled connection starts or a new schedule is introduced, connEv extension is terminated through connEv preemption.

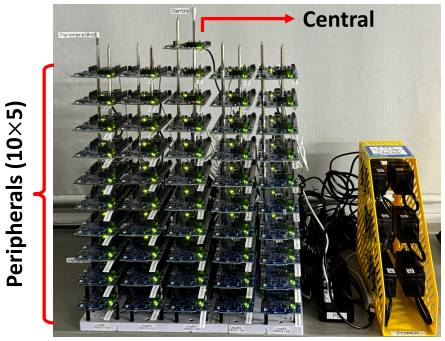

Fig. 15. 51-node BLE testbed with one central and 50 peripherals.

VI. Evaluation

In this section, we evaluate the performance of *EMBLEM* against the state-of-the-art scheduling schemes BLEX [32] and RT-BLE [33], as well as representative BLE stacks NimBLE [19], Nordic [20], and Zephyr [21].

A. Experiment Setup

We implement *EMBLEM* on Nordic nRF52840 DK [37] using Mynewt OS 1.10.0 [39] and NimBLE 1.5.0 while using the open-source code of the comparison schemes without any modifications[7]. Experiments are conducted on a 51-node testbed consisting of one central and 50 peripheral BLE devices, as shown in Fig. 15. To the best of our knowledge, this is the largest-scale BLE experiments conducted among comparable related works.

All BLE stacks support connections with more than 50 peripherals, except Nordic, which is constrained to a maximum of 20 connections. Given this setup, we assess the connection setup latency, schedulability, fairness, scalability, and adaptability based on the number of connections, QoS satisfaction, throughput, and Jain's fairness index (JFI). QoS satisfaction is measured as the actual number of packets received in each connection compared to the expected count based on application requirement (*i.e.,* notification count and connItvl). JFI is calculated as $\left(\sum_{n=1}^{N}(t_n/t_n^*)\right)^2 / \left(N \cdot \sum_{n=1}^{N}(t_n/t_n^*)^2\right)$, where the actual throughput t_n of each connection n is normalized by its expected value t_n^* to evaluate fairness among N connections with different requirements.

EMBLEM aims to eliminate resource overlap while achieving fairness, reliable connectivity/schedulability, rapid adaptability, and scalability.

B. Connection Setup Latency

We first evaluate the connection setup latency under various connItvl (10–4000 ms) to demonstrate the fast connection establishment capability of *EMBLEM*. Connection setup latency is defined as the elapsed time from the transmission of

[7]BLEX [32] https://github.com/ejparkNETLAB/BLEX_Master, *RT-BLE* [33] https://github.com/sada45/RT-BLE, *NimBLE* [19] https://github.com/apache/mynewt-nimble, *Nordic* [20] https://www.nordicsemi.com/Products/Development-software/nRF-Connect-SDK, and *Zephyr* [21] https://github.com/zephyrproject-rtos/zephyr

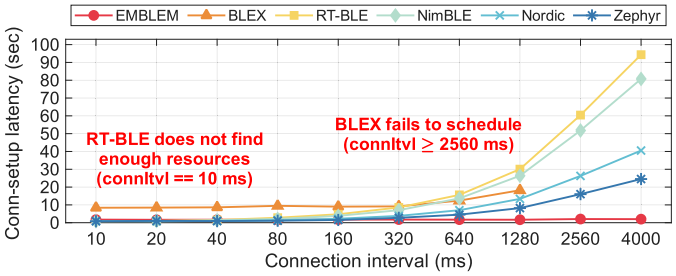

Fig. 16. Average connection setup latency of each scheduling scheme, from the start of CONNECT_IND to a service subscription. *EMBLEM* has the lowest connection setup latency, significantly lower than other schemes when the connection interval is long.

the CONNECT_IND to the reception of the service subscription response and is measured ten times for each connItvl in a one-to-one connection scenario.

Fig. 16 plots the average latency of connection setup for all compared schemes. Except for *EMBLEM*, most schemes exhibit an increase in connection setup latency proportional to connItvl. This is because the discovery process for subscribing to services (as shown in Fig. 2) is carried out according to connItvl after a connection has been established, which is a natural phenomenon. However, particularly when connItvl is 4000 ms, the connection setup latency, depending on the BLE stack, ranges from at least ≈24.52 s (Zephyr) to as long as ≈94.34 s (RT-BLE), which can degrade the user experience due to prolonged waiting time. Furthermore, RT-BLE fails to find enough resources at connItvl of 10 ms, resulting in connection failures, and BLEX triggers system halts when connItvl ≥ 2560 ms.

In contrast, *EMBLEM completes the connection setup in 1.56–2 s across all connItvl (10–4000 ms)*. When connItvl is 4000 ms, the maximum setup latency of *EMBLEM* is ≈2 s, achieving a performance that is ≈12–47 times faster than comparison schemes. This is because *EMBLEM* fixes the connItvl of a new connection to 7.5 ms and completes scheduling after service subscription through a subrating command, thereby enabling faster service discovery by temporarily using unused resources at the minimum connItvl regardless of the $\text{connItvl}_{\text{orig}}$ and subFactor.

C. Main Performance Results

To verify the resource overlap elimination, fairness, connectivity, and scalability of *EMBLEM*, we conduct extensive experiments on the 51-node testbed with various connItvl (20–1280 ms) and compare with other schemes. In this set of experiments, a central attempts to connect sequentially to a maximum of 50 peripherals using the same connItvl. Upon completion of connection setup, each peripheral periodically transmits one data packet with maximum size "attribute value" (*i.e.,* BLE application payload, 244 bytes) to the central using a Bluetooth notification attribute [18]. The packet transmission interval is set identically to the connItvl as described in Section V-B. Each experiment ran for five minutes, and repeat five times for each scenario.

Fig. 17(a) plots the minimum, maximum, and average number of connections established with the central for each connItvl, and Fig. 17(b) and (c) plots the number and ratio of peripherals that satisfy its QoS requirement. *EMBLEM guarantees QoS of more connections compared to other schemes [see Fig. 17(b)] and maintains QoS satisfaction of 100% for all connections [see Fig. 17(c)]*, providing stable and consistent performance across all connItvl scenarios. This is because *EMBLEM* eliminates resource overlap through sufficient resource allocation for each connection and precise scheduling that separates connections. In contrast, the comparison schemes achieve lower QoS satisfaction relative to the total number of connections and show unstable performance with noticeable fluctuations.

For example, when the connItvl is 160 ms [see Fig. 17(a)], *EMBLEM* can connect to 24 peripherals, whereas BLEX, RT-BLE, NimBLE, Nordic, and Zephyr average 6.2, 42.6, 47.6, 13, and 49.4 connections, respectively; It seems that RT-BLE, NimBLE, and Zephyr can connect to approximately twice as many peripherals as *EMBLEM*. However, their QoS-satisfied "good" connections average 4.2, 0.8, 13.6, 2.2, and 0, corresponding to only 69.05%, 2.02%, 28.51%, 16.96%, and 0% of their connections, respectively. Consequently, *EMBLEM* outperforms the comparison schemes in terms of "good" connections by approximately 30.95%, 97.98%, 71.49%, 83.04%, and 100% [see Fig. 17(b) and (c)]. At a connItvl of 1280 ms where there is a sufficient time gap to accommodate 50 or more peripherals, the comparison schemes still fail to support 50 connections and achieve good (near 100%) QoS satisfaction due to resource overlap. *On the other hand, EMBLEM maintains 100% QoS satisfaction for all 50 connections.*

Furthermore, *EMBLEM achieves uniform throughput across all connections [see Fig. 17(d)] and consistently attains a JFI of 1 in all scenarios [see Fig. 17(e)],* demonstrating its fairness in resource allocation. For instance, when connItvl is 20 ms, *EMBLEM* achieves a throughput of ≈97.607 kb/s per connection, exactly matching the theoretical expected throughput of 97.6 kb/s calculated based on connItvl, notification count, and packet size. In contrast, comparison schemes show severe throughput imbalance across connections. For example, the minimum values for BLEX and RT-BLE are ≈20.751 and ≈47.021 kb/s, which are ≈78.74% and ≈51.82% lower than the expected throughput (or *EMBLEM*). This implies that ≈40 and ≈26 data packets per second are queued or delayed due to *resource preemption* or *connEv blocking* among connections, demonstrating that BLEX and RT-BLE are still susceptible to overlap problems.

In addition, Fig. 17(f) shows that *EMBLEM* consistently achieves improved performance over the whole legal connItvl range of 7.5–3840 ms, complementing the previous results and demonstrating *the robustness of EMBLEM across the entire connItvl range specified in Bluetooth standard*. On the other hand, BLEX supports only one connection at 15–1920 ms and does not work at 7.5 and 3840 ms, showing a discrepancy between its design to select an appropriate connItvl within 1.25×2^n ms (*i.e.,* 10, 20,..., 2560 ms) and its actual operation.

Overall, these results confirm that *EMBLEM* successfully eliminates resource overlap, ensures fairness among connections, and provides reliable connectivity to over 50 peripherals

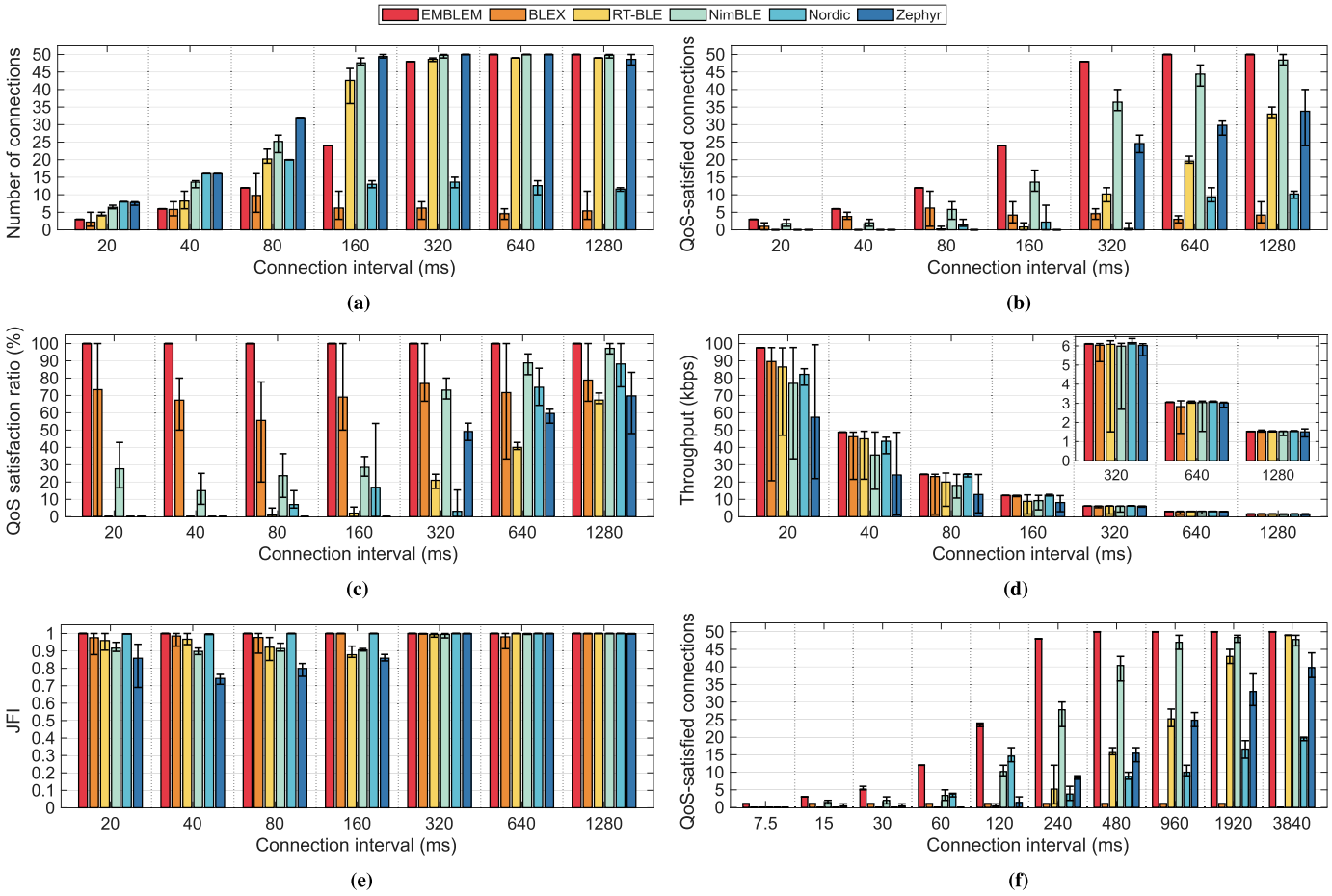

Fig. 17. Main performance results for *EMBLEM* compared to two state-of-the-art scheduling schemes (BLEX and RT-BLE) and three representative BLE stacks (NimBLE, Nordic, and Zephyr) under various connection intervals. (a) Number of maximum connections. (b) Number of QoS-satisfied connections. (c) QoS-satisfied connection ratio. (d) Throughput per connection. (e) JFI. (f) Number of QoS-satisfied connections (7.5–3840 ms).

with higher and perfectly fair throughput, thereby enabling scalability for large-scale BLE networks.

D. Various Resource (Throughput) Requirements

To verify the effectiveness of *EMBLEM* under various resource requirements (*i.e.,* application wishes to transmit more number of packets per unit time), we conduct experiments with varying numbers of notifications per connItvl when the connItvl set to 1280 ms. As in the previous experiment setup, we set the data size to the maximum (244 bytes) and the transmission period to the connItvl (1280 ms), and conduct each scenario for five minutes with five repetitions.[8] The packets are transmitted consecutively within a period.

We first measure the aggregate throughput at the central with multiple connections to understand its capacity for our experiments. Fig. 18(a) plots the average aggregate throughput at the central for all compared schemes under different notification counts per connItvl. Despite differences in scheduling schemes, most exhibit an increasing trend in aggregate throughput as the number of notifications increases.

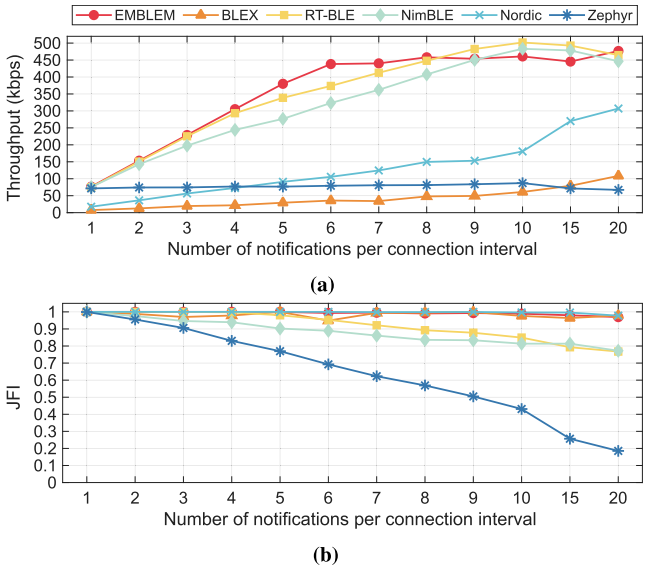

Fig. 18. Performance results for *EMBLEM* under different notification counts (1–20) per connection interval (1280 ms). (a) Average aggregate throughput. (b) Average JFI.

EMBLEM, RT-BLE, and NimBLE achieve higher throughput than others but saturate at a capacity of ≈ 477–502 kb/s for

[8]In this work, we do not consider the data segmentation/reassembly feature of the L2CAP layer in Bluetooth since data of various services may be transmitted in multiple packets on a single connection.

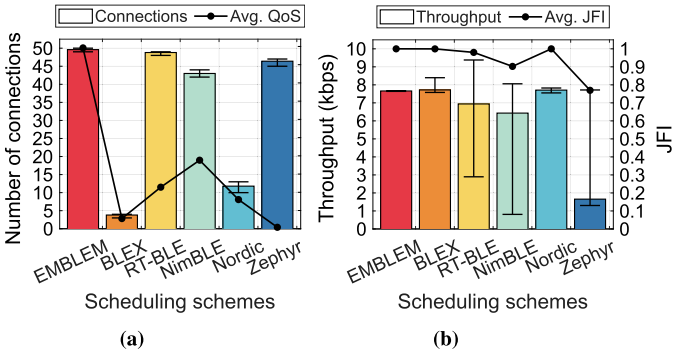

Fig. 19. Performance results with five notifications per 1280 ms connection interval for all 50 peripherals. (a) Number of connections and average QoS satisfaction. (b) Throughput per connection and average JFI.

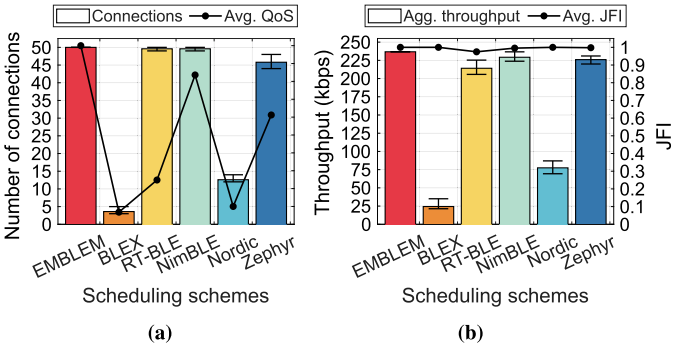

Fig. 20. Performance results under different connection intervals. (a) Number of connections and average QoS satisfaction. (b) Aggregate throughput per connection and average JFI.

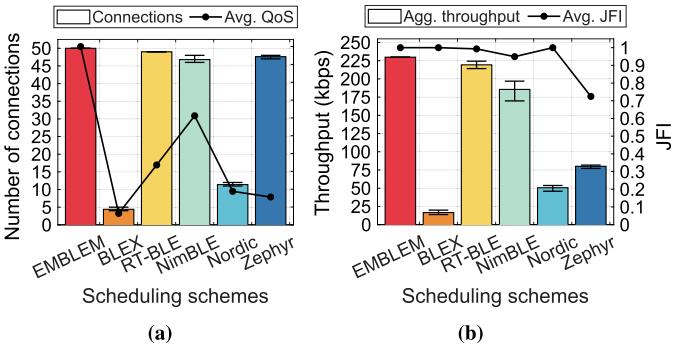

Fig. 21. Performance results under different notification counts. (a) Number of connections and average QoS satisfaction. (b) Aggregate throughput per connection and average JFI.

notification counts above 6–10. BLEX and Nordic continue to increase steadily, but their capacities are significantly lower at ≈ 109 and ≈ 307 kb/s, respectively. The major reason for these results is that Nordic limits the maximum number of connections to 20, and BLEX does not fully demonstrate its capability despite supporting over 50 connections. On the other hand, Zephyr achieves a much lower capacity of ≈ 87 kb/s compared to the others, regardless of notification count. This is because subsequent connections block earlier ones (*i.e., connItvl blocking*), thereby limiting their transmission, as described in Section III-A. Moreover, Fig. 18(b) shows that *EMBLEM*, BLEX, and Nordic maintain high JFI across all scenarios, while RT-BLE, NimBLE, and Zephyr decline with increasing notification counts and fail to ensure fairness.

Fig. 19 plots a representative performance result among the experiments within Fig. 18 when the number of notifications is five per connItvl. As shown in Fig. 19(a), *EMBLEM* can connect to 49.6 peripherals on average (maximum 50 and minimum 49) while all connections satisfy QoS of 100%. Compared to BLEX, RT-BLE, NimBLE, Nordic, and Zephyr, *EMBLEM* supports up to 47, 2, 8, 40, and 5 more connections and improves QoS satisfaction by 50%, 83.67%, 62.79%, 54.55%, and 100%, respectively. In addition, *EMBLEM* achieves uniform throughput and ensures fairness with a JFI of 1 [see Fig. 19(b)], demonstrating its effectiveness.

E. Diverse and Mixed Peripheral Requirements

Diverse requirements of the concurrently connected peripherals make both connection scheduling and resource allocation for multiple devices more difficult, particularly in large-scale networks. Accordingly, to verify the schedulability and connectivity of *EMBLEM* under diverse requirements, we conduct experiments in two scenarios: 1) one with different connItvl among connections and 2) the other with different notification counts across peripherals. In the first scenario, the notification count of all peripherals is fixed to 1, and various connItvls 160, 320, 640, 1280, and 2480 ms are evenly assigned to connections in a round-robin manner. For the second scenario, the connItvl is fixed to 1280 ms, and the notification counts ranging from 1 to 5 are assigned to peripherals in the same manner.

Figs. 20 and 21 confirm that for all scenarios, *EMBLEM* achieves a JFI of 1 and 100% QoS guarantee for all 50 connections with high aggregate throughput of 236.75 and 229.822 kb/s, respectively. This result demonstrates that *EMBLEM* attains high schedulability and provides stable connectivity with fairness, thereby ensuring diverse services and application requirements in large-scale BLE networks. Specifically, in each scenario, *EMBLEM* outperforms the comparison schemes, supporting 1–47 and 1–46 more connections, achieving 20%–91.67% and 25%–85.11% higher QoS satisfaction, and ≈ 13–215 and ≈ 16–216 kb/s higher aggregate throughput for the different connItvl and notification count scenarios, respectively. In contrast, the comparison schemes still exhibit inconsistent performance and unstable connectivity, achieving low QoS satisfaction and schedulability.

F. Adaptability to Network Dynamics

In dynamic wireless network environments, the ability to rapidly respond to traffic load variations and to swiftly stabilize performance is of critical importance. To demonstrate the adaptability and robustness of *EMBLEM*, we conduct experiments under fluctuating traffic loads. In this scenario, we connect five peripherals to the central with connItvl of 160 ms, each of which transmits a maximum-sized notification

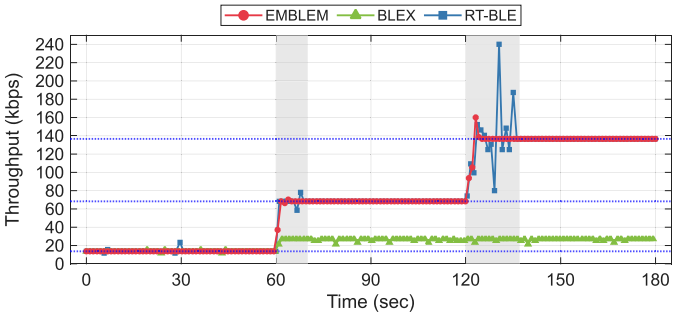

Fig. 22. Adaptability of *EMBLEM* to resource requirement variations under increasing traffic load.

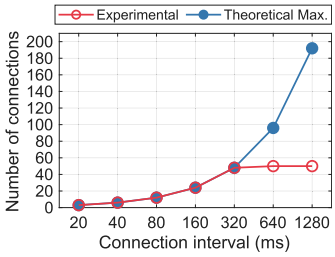

Fig. 23. Theoretical maximum connection bound of *EMBLEM*.

packet (244 bytes) to the central at every connItvl. Once all connections are established, we gradually raise the number of transmission packets for the first connection to 1, 5, and 10 at 60-s intervals and measure the throughput at 1-s intervals over 180 s. Nimble, Nordic, and Zephyr are excluded from this experiment, as they do not support adaptive resource updates.

Fig. 22 plots a representative result from 30 repeated experiments, where the blue horizontal dotted lines denote the expected throughput for each notification packet count. *EMBLEM* rapidly adapts to traffic load fluctuations by continuously tracking resource usage for each connection, enabling the stabilization of throughput within a short convergence time and sustained reliable performance. When the packet count is raised to 10 at $t = 120$ s, *EMBLEM* requires ≈ 4 s ($t = 120$–124 s) to converge and exhibits a transient rise in throughput just before convergence (at $t = 123$ s). This is because of the rescheduling required to allocate a new resource space to the connection, followed by the burst transmission of previously delayed packets caused by insufficient resources before the resource update. In contrast, RT-BLE requires a considerably longer time to stabilize performance. Particularly when the packet count is 10, severe throughput jitter is observed for 15 s ($t = 120$–135 s), resulting in a convergence time of ≈ 16 s ($t = 136$ s). This arises because RT-BLE requires an additional process to respond to changes in data volume of the peripheral, in which the "RT-BLE agent" incorporated into the L2CAP layer on the peripheral side detects a change in application demand, transmits a resource update request to the central, and then performs update and rescheduling between the two devices. On the other hand, BLEX supports dynamic resource updates but fails to perform them. The reason is that the connection schedules in BLEX are arranged consecutively with a short time gap, and a resource update requires adjusting the anchor points of all other connections except the one being updated. The adjustment process requires considerable time and may even cause the entire schedule to collapse if some connection updates fail.

This result confirms that *EMBLEM* ensures performance stability, practicality, and robustness under dynamic network environments while outperforming the comparison schemes by achieving adaptation four times faster than RT-BLE.

G. Schedulability and Energy Consumption

To investigate the theoretical maximum number of connections of *EMBLEM*, we consider the same scenario described in Section VI-C, where the peripherals are connected to the central with the same connItvl and transmit one maximum-sized data packet (244 bytes) once every connItvl. Under this scenario, assuming that the central device has unlimited memory capacity, and the wireless channel experiences neither interference nor packet loss, the theoretical upper bound of *EMBLEM* is given by the ratio of the emulated connItvl ($7.5\,\text{ms} \times subFactor$) derived from (1) and (2) to the sum of the resource slots required for exchanging one packet pair and the GS ($2500\,\mu$s). The number of required slots can be calculated as $\lceil (T_{\text{tx}} + T_{\text{rx}} + \text{IFS} * 2) \,/\, 1250\,\mu s \rceil \times 1250\,\mu s$, where T_{tx} and T_{rx} denote the time required for transmitting the maximum packet ($2088\,\mu$s) and for receiving the link layer ACK ($80\,\mu$s), respectively. The IFS is specified as $150\,\mu$s in Bluetooth [18], and $1250\,\mu$s is the time capacity of one resource slot in *EMBLEM*.

As shown in Fig. 23, *EMBLEM* can theoretically support more than 192 concurrent connections when connItvl is equal to or greater than 1280 ms. Notably, the experimental results obtained from the 51-node testbed coincide with the theoretical upper bound up to connItvl of 320 ms, beyond which *EMBLEM* connects all 50 peripherals in the testbed. This result suggests that *EMBLEM* is expected to maintain performance consistent with the theoretical upper bound even beyond the tested range, implying that it is capable of accommodating more than 50 connections.

Furthermore, we evaluate the average energy consumption of peripherals for transmitting one maximum-sized packet under the same scenario as before. This can be calculated by $V \times (I_{\text{tx}} \times T_{\text{tx}} + I_{\text{rx}} \times T_{\text{rx}}) \times \bar{N}_{\text{tx}}$. According to the nRF52840 product specification, when the supply voltage is 3 V and the transmission power is 0 dBm, the currents for radio transmission and reception (I_{tx} and I_{rx}) are 4.8 and 4.6 mA, respectively [38]. T_{tx} and T_{rx} are the same as defined above, and $\bar{N}_{\text{tx}}$ denotes the average number of transmission attempts, including retransmissions, to successfully transmit one packet.

Fig. 24 shows that *EMBLEM* achieves the average energy consumption of $31.7\,\mu$J across all connItvl, significantly reducing it by up to 33% and 38% compared with BLEX and

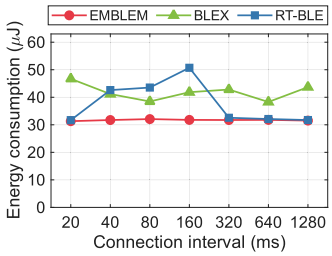

Fig. 24. Average energy consumption to transmit one packet.

RT-BLE, respectively. In contrast, RT-BLE shows a sharp increase in energy consumption when connItvl is 40–160 ms, after which it gradually stabilizes. This arises because it may accept new connections even when resources are insufficient for scheduling, which intensifies resource overlap and contention between connections, leading to more retransmissions and consequently higher energy use. We observe that RT-BLE establishes approximately 1.3–1.5 times more connections than its theoretical upper bound per connItvl. When the connItvl exceeds 320 ms, the overconnection issue is eliminated, and resource contention is reduced, as RT-BLE can theoretically support at least 64 connections. On the other hand, BLEX exhibits an irregular pattern due to resource overlap caused by unstable connection scheduling and resource management.

These results confirm that *EMBLEM* can achieve stable and high connectivity, as well as schedulability, while consuming less energy than state-of-the-art scheduling schemes.

VII. Conclusion

We proposed *EMBLEM* that eliminates the *resource overlap problem* and guarantees fairness while supporting a larger number of connections in BLE networks. We introduced novel mechanisms for fine-grained and precise resource management with high efficiency that adapts rapidly to traffic variations. Our extensive evaluation on a 51-node BLE testbed against state-of-the-art schemes (BLEX and RT-BLE) and representative BLE stacks (Nimble, Nordic, and Zephyr) validates that *EMBLEM* can support 50 concurrent connections with 100% QoS satisfaction ratio, achieves 47× faster connection setup latency and 17× higher throughput, adapts 4× faster to traffic variations, reduces energy consumption by 38%, and ensures perfect fairness, thereby providing improved connectivity, scalability, robustness, practicality, and efficiency. As future work, we plan to extend *EMBLEM* with novel adaptive frequency hopping to investigate scheduling solutions in interference-prone environments.

References

[1] M. Kim and J. Paek, "Poster abstract: Multi-connection scheduling based on connection subrating for fair resource allocation in Bluetooth low energy networks," in *Proc. 21st ACM Conf. Embedded Networked Sensor Syst.*, Nov. 2023, pp. 550–551.

[2] M. Kim and J. Paek, "Multi-connection scheduling for resource fairness in Bluetooth low energy networks," in *Proc. 14th Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2023, pp. 533–535.

[3] (2024). *2024 Bluetooth Market Update*. Accessed: Sep. 2025. [Online]. Available: https://www.bluetooth.com/2025-market-update/

[4] M. Collotta and G. Pau, "A novel energy management approach for smart homes using Bluetooth low energy," *IEEE J. Sel. Areas Commun.*, vol. 33, no. 12, pp. 2988–2996, Dec. 2015.

[5] M. Collotta and G. Pau, "An innovative approach for forecasting of energy requirements to improve a smart home management system based on BLE," *IEEE Trans. Green Commun. Netw.*, vol. 1, no. 1, pp. 112–120, Mar. 2017.

[6] C. Zunino, A. Valenzano, R. Obermaisser, and S. Petersen, "Factory communications at the dawn of the fourth industrial revolution," *Comput. Standards Interfaces*, vol. 71, Aug. 2020, Art. no. 103433.

[7] S. Alletto et al., "An indoor location-aware system for an IoT-based smart museum," *IEEE Internet Things J.*, vol. 3, no. 2, pp. 244–253, Apr. 2016.

[8] N. Y. Philip, J. J. P. C. Rodrigues, H. Wang, S. J. Fong, and J. Chen, "Internet of Things for in-home health monitoring systems: Current advances, challenges and future directions," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 2, pp. 300–310, Feb. 2021.

[9] G. Alfian, M. Syafrudin, M. F. Ijaz, M. A. Syaekhoni, N. L. Fitriyani, and J. Rhee, "A personalized healthcare monitoring system for diabetic patients by utilizing BLE-based sensors and real-time data processing," *Sensors*, vol. 18, no. 7, p. 2183, Jul. 2018.

[10] J.-r. Lin, T. Talty, and O. K. Tonguz, "On the potential of Bluetooth low energy technology for vehicular applications," *IEEE Commun. Mag.*, vol. 53, no. 1, pp. 267–275, Jan. 2015.

[11] W. Bronzi, R. Frank, G. Castignani, and T. Engel, "Bluetooth low energy performance and robustness analysis for inter-vehicular communications," *Ad Hoc Netw.*, vol. 37, pp. 76–86, Feb. 2016.

[12] R. L. Rosa, C. Dehollain, A. Burg, M. Costanza, and P. Livreri, "An energy-autonomous wireless sensor with simultaneous energy harvesting and ambient light sensing," *IEEE Sensors J.*, vol. 21, no. 12, pp. 13744–13752, Jun. 2021.

[13] M. Magno, F. Vultier, B. Szebedy, H. Yamahachi, R. H. R. Hahnloser, and L. Benini, "A Bluetooth-low-energy sensor node for acoustic monitoring of small birds," *IEEE Sensors J.*, vol. 20, no. 1, pp. 425–433, Jan. 2020.

[14] M. Kim, J. Lee, and J. Paek, "Neutralizing BLE beacon-based electronic attendance system using signal imitation attack," *IEEE Access*, vol. 6, pp. 77921–77930, 2018.

[15] Y. Li, H. Lin, J. Lv, Y. Gao, and W. Dong, "BLE location tracking attacks by exploiting frequency synthesizer imperfection," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Sep. 2024, pp. 1860–1869.

[16] P. Davidson and R. Piché, "A survey of selected indoor positioning methods for smartphones," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 2, pp. 1347–1370, 2nd Quart., 2017.

[17] Bluetooth Special Interest Group (SIG). (2025). *Bluetooth Special Interest Group (SIG)*. Accessed: Sep. 2025. [Online]. Available: https://www.bluetooth.com/

[18] Bluetooth Special Interest Group. (2025). *Bluetooth Core Specification*. Accessed: Oct. 2025. [Online]. Available: https://www.bluetooth.com/specifications/specs/core-specification-6-1/

[19] (2025). *Apache Mynewt-NimBLE*. Accessed: Sep. 2025. [Online]. Available: https://mynewt.apache.org/latest/network/index.html

[20] (2025). *Nordic Semiconductor*. Accessed: Sep. 2025. [Online]. Available: https://www.nordicsemi.com/

[21] (2025). *Zephyr Project*. Accessed: Sep. 2025. [Online]. Available: https://www.zephyrproject.org/

[22] P. Kindt, D. Yunge, M. Gopp, and S. Chakraborty, "Adaptive online power-management for Bluetooth low energy," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 2695–2703.

[23] J.-H. Chen, Y.-S. Chen, and Y.-L. Jiang, "Energy-efficient scheduling for multiple latency-sensitive Bluetooth low energy nodes," *IEEE Sensors J.*, vol. 18, no. 2, pp. 849–859, Jan. 2018.

[24] M. Spörk, C. A. Boano, and K. Römer, "Improving the timeliness of Bluetooth low energy in dynamic RF environments," *ACM Trans. Internet Things*, vol. 1, no. 2, pp. 1–32, Apr. 2020.

[25] W. Sun et al., "BlueCoDE: Bluetooth coordination in dense environment for better coexistence," in *Proc. IEEE 25th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2017, pp. 1–10.

[26] T. Lee, J. Han, M.-S. Lee, H.-S. Kim, and S. Bahk, "CABLE: Connection interval adaptation for BLE in dynamic wireless environments," in *Proc. 14th Annu. IEEE Int. Conf. Sens., Commun., Netw. (SECON)*, Jun. 2017, pp. 1–9.

[27] E. Park, M.-S. Lee, and S. Bahk, "AdaptaBLE: Data rate and transmission power adaptation for Bluetooth low energy," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2019, pp. 1–6.

[28] K. E. Jeon, J. She, P. Soonsawad, and P. C. Ng, "BLE beacons for Internet of Things applications: Survey, challenges, and opportunities," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 811–828, Apr. 2018.

[29] R. Rondón, M. Gidlund, and K. Landernäs, "Evaluating Bluetooth low energy suitability for time-critical industrial IoT applications," *Int. J. Wireless Inf. Netw.*, vol. 24, no. 3, pp. 278–290, Sep. 2017.
[30] D. Ryoo, Y. Yoo, J. Paek, and S. Bahk, "SPADE: Secure periodic advertising using coded time-channel rendezvous for BLE audio," in *Proc. Int. Conf. Distrib. Comput. Smart Syst. Internet Things*, vol. 20, 2023, pp. 39–46.
[31] F. J. Dian, A. Yousefi, and S. Lim, "Time scheduling of central BLE for connection events," *IEEE Inf. Technol.*, vol. 12, pp. 763–767, 2018.
[32] E. Park, H.-S. Kim, and S. Bahk, "BLEX: Flexible multi-connection scheduling for Bluetooth low energy," in *Proc. 20th Int. Conf. Inf. Process. Sensor Netw.*, 2021, pp. 268–282.
[33] Y. Li, J. Lv, B. Li, and W. Dong, "RT-BLE: Real-time multi-connection scheduling for Bluetooth low energy," in *Proc. IEEE Conf. Comput. Commun.*, May 2023, pp. 1–10.
[34] Bluetooth Special Interest Group. (2021). *Bluetooth Core Specification*. Accessed: Sep. 2025. [Online]. Available: https://www.bluetooth.com/specifications/specs/core-specification-5-3/
[35] M. Kim, D. Hyeon, and J. Paek, "ETAS: Enhanced time-aware shaper for supporting nonisochronous emergency traffic in time-sensitive networks," *IEEE Internet Things J.*, vol. 9, no. 13, pp. 10480–10491, Jul. 2022.
[36] J. Min, Y. Kim, M. Kim, J. Paek, and R. Govindan, "Reinforcement learning based routing for time-aware shaper scheduling in time-sensitive networks," *Comput. Netw.*, vol. 235, Nov. 2023, Art. no. 109983.
[37] (2025). *NRF52840DK*. Accessed: Sep. 2025. [Online]. Available: https://www.nordicsemi.com/Products/Development-hardware/nRF52840-DK
[38] (2025). *NRF52840 Product Specification*. Accessed: Sep. 2025. [Online]. Available: https://docs.nordicsemi.com/bundle/ps_nrf52840/page/keyfeatures_html5.html
[39] (2025). *Apache Mynewt*. Accessed: Sep. 2025. [Online]. Available: https://mynewt.apache.org/

Moonbeom Kim received the B.S. degree in computer and information communications engineering from Hongik University, Seoul, South Korea, in 2017, and the M.S. degree in computer science and engineering from Chung-Ang University, Seoul, in 2020, where he is currently pursuing the Ph.D. degree.

He is also a Research Assistant at the Networked Systems Laboratory (NSL), Seoul, led by Dr. Jeongyeup Paek, with research interests in wireless networking, localization, time-sensitive networking, and radio resource management and scheduling.

Jeongyeup Paek (Senior Member, IEEE) received the B.S. degree from Seoul National University, Seoul, South Korea, in 2003, and the M.S. degree from the University of Southern California (USC), Los Angeles, CA, USA, in 2005, both in electrical engineering, and the Ph.D. degree in computer science from USC in 2010.

He worked at Deutsche Telekom Inc. Research and Development Labs, Los Altos, CA, USA, as a Research Intern in 2010, and then joined Cisco Systems Inc., San Jose, CA, USA, in 2011, where he was a Technical Leader at the Internet of Things Group. In 2014, he was with the Department of Computer Information Communication, Hongik University, Seoul, as an Assistant Professor. He has been a Tenured Professor at the Department of Computer Science and Engineering, Chung-Ang University, Seoul, since 2015.