RT-BLE: Real-time Multi-Connection Scheduling for Bluetooth Low Energy

Yeming Li, Jiamei Lv, Borui Li, Wei Dong
College of Computer Science, Zhejiang University and Alibaba-Zhejiang University Joint Institute of Frontier Technologies
Email:{liymemnets, lvjm, borui.li, dongw}@zju.edu.cn

IEEE INFOCOM 2023 - IEEE Conference on Computer Communications | 979-8-3503-3414-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/INFOCOM53939.2023.10229006

***Abstract*—Bluetooth Low Energy (BLE) is one of the popular wireless protocols to build IoT applications. However, the BLE suffers from three major issues that make it unable to provide reliable service to time-critical IoT applications. First, the BLE operates in the crowded 2.4GHz frequency band, which can lead to a high packet loss rate. Second, it is common for one device to connect with multiple BLE Peripherals, which can lead to severe collision issue. Third, there is a long delay to re-allocate time resource. In this paper, we propose *RT-BLE:* a real-time multi-connection scheduling scheme for BLE. We first formulate the BLE transmission latency in noisy RF environments considering the BLE retransmission mechanism. With this, *RT-BLE* can get a set of initial connection parameters. Then, *RT-BLE* uses collision tree based time resource scheduling technology to efficiently manage time resource. Finally, we propose a subrating-based fast connection re-scheduling method to update the connection parameters and the position of anchor points. The result shows *RT-BLE* can provide reliable service and the error of our model is less than 0.69%. Compare with existing works, the re-scheduling delay is reduced by up to 86.25% and the capacity is up to 4.33× higher.**

I. Introduction

Bluetooth Low Energy (BLE) has become one of the most popular wireless protocols to implement IoT applications, because of its cheap, low-energy, and wide adaptation nature. According to the ABI research [1], the shipment of BLE devices will reach 1.6 billion in 2023. In addition, with the growth of the Internet of Things (IoT), it is common for one central node to connect with multiple remote peripherals via BLE. For example, in a smart home application [2], a BLE gateway connects with a remote temperature sensor node to remotely control the air-conditioner and heater. Among all these BLE-based IoT applications, some connections have critical latency requirements, e.g., the fire alarms [3], [4], health care [5], [6], and structural health monitoring (SHM) [7], [8].

However, the existing experiment [9], [10] shows native BLE failed to provide real-time and reliable service for those time-critical applications because of packet loss and collisions among multiple connections. Despite their contributions, these works are still far from real-time transmission guarantee in practice, especially in multi-connection scenario. Addressing this problem, however, raises a number of challenging issues.

First, how to build an accurate model for real-time transmission in noisy RF environment. There are existing models for predicting BLE transmission latency [11]–[13]. However, these approaches are usually based on the measurement of average round-trip-time (RTT) for application-layer data transmissions, ignoring the underlying BLE retransmission mechanisms.

Second, how to efficiently schedule multiple connections to avoid collisions and ensure the latency requirements. A full parameter space would inevitably lead to a large computing overhead at the Central node. BLEX [10] proposes an online multi-connection scheduling method. But it is failed to meet real-time latency requirements because it can only manipulate part of the connection parameters.

Third, how to rapidly perform connection re-scheduling. More specifically, changing the position of the anchor point and updating two connection parameters of *connection interval* and the length of *connection event*. This is important to match the traffic dynamics. The existing method suffers from a long mandatory delay, i.e., 6 connection intervals, which hampers the transmission performance in highly dynamic conditions.

To address these three challenges, we present *RT-BLE*, a real-time multi-connection scheduling scheme for BLE. To the best of our knowledge, this is the first work that explicitly considers the problem of real-time transmission over multiple BLE connections. First, we propose an accurate BLE transmission model which precisely presents the underlying BLE retransmission mechanism. The model indicates the worst-case transmission latency occurred when packet losses happened in the last data fragment. With this model, *RT-BLE* can estimate how much time resource is required for each connection. Second, we propose *collision tree based time resource management technology* to efficiently schedule multiple connections. The collision tree shows the relationship of time resource (i.e., collided or not). By searching the collision tree, *RT-BLE* can find the optimal resource allocation for each connection. Finally, we propose a novel *subrating-based fast connection re-scheduling method*. It is designed based on connection update and connection subrating procedures in conjunction, in which the connection subrating allows users to coarsely adjust connection parameters without extra delay.

We implement *RT-BLE* on nRF52840 platform [14] and NimBLE protocol stack [15]. The experiment results show *RT-BLE* can provide reliable service for time-critical IoT applications even with a high packet loss rate, and the error of our model is less than 0.69%. Compared with existing works, *RT-BLE* can reduce up to 86.25% re-scheduling delay and has up to 4.33× higher capacity. Our contributions are three-fold:

- We propose a timeliness model in noisy RF environment

This work is supported by the National Natural Science Foundation of China under Grant No. 62072396 and the National Youth Talent Support Program. Jiamei Lv and Wei Dong are the corresponding authors.

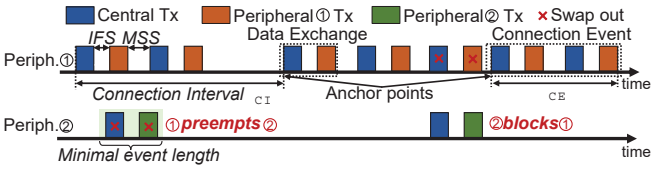


Fig. 1: Timing diagram of BLE link-layer scheduling.

considering the BLE retransmission mechanism. Then it is used to allocate proper time resource to each connection according to their requirements.

- We propose a novel collision tree based resource management technology to efficiently allocate time resource for each connection.
- We have implemented *RT-BLE* using off-the-shelf BLE chip and open source our code[1]. We conduct extensive experiments to evaluate the performance of *RT-BLE*.

II. Related Work

Time resource scheduling is quite popular for wireless protocols. A large number of works has been proposed for different link-layer protocols, such as pTunes [16] for IEEE 802.15.4, AdaptiveLoRa [17] for LoRa protocol, and [18]–[22] for TSCH protocol. For BLE, numerous works has been done to optimize the performance of neighbor discovery communication [23], [24], cross-technology communications [25], and indoor localization [26], [27]. There are already some works that provide models for connection-based BLE communication performance. The BLEach [28] proposed an IPv6-over-BLE stack and formulate the latency in single connection scenario and a multi-hop version is proposed in [29]. Based on this, [9], [11]–[13] introduce a similar BLE transmission model in noisy RF environment. It asks the node to measure the average RTT of an application packet with maximum payload in advance. Then check how many fragments are needed for the current data packet and speculate the latency. However, this method works in application-layer and ignores the underlying details of BLE link-layer. The *RT-BLE* carefully model the transmission latency of BLE with considering the complex retransmission mechanism. Some works focus on managing time resource in single-connection scenario [30], [31] and multi-connection scenario [32], [33]. Park et al. propose BLEX [10] for online multi-connection scheduling. It allocate time resource for each connection according to their historical usage. But BLEX is failed to guarantee real-time transmissions, especially when the application requirements changes frequently. This is because the connection rescheduling of BLEX suffers from a long mandatory delay introduced by Bluetooth specification. Besides, BLEX uses a greedy policy to allocate the time resource, which makes the system capacity low. To solve these, *RT-BLE* proposes subrating-based fast connection rescheduling method and collision-tree based resource management technology.

III. BLE Background

In this section, we introduce the necessary preliminary knowledge of BLE. Sec. III-A introduces the BLE link-layer scheduling. Sec. III-B introduces the connection update and connection subrating procedures.

[1]https://github.com/sada45/RT-BLE

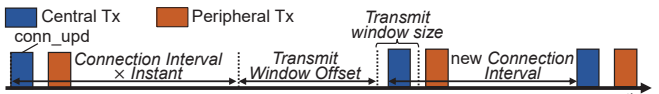


Fig. 2: Connection update procedure.

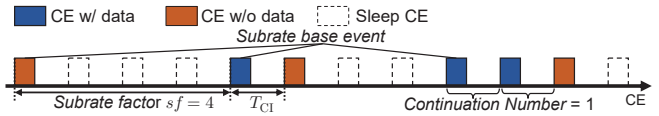


Fig. 3: Connection subrating feature.

A. BLE link-layer scheduling

Devices in a BLE connection have two roles: the Central and Peripheral, or the Master and Slave with old fashion names. Fig. 1 shows an example of BLE link-layer multi-connection scheduling. The Central and Peripheral will synchronize their clock during connection establishment. Therefore, both of them will turn on their radio periodically, and start a *connection event* (CE). The start of each connection event are called *anchor points*, and the interval between two consecutive anchor points is *connection interval* (CI), we denote it as t_{CI}. The CI is an integer multiple of 1.25ms and the value should between 7.5ms and 4000ms. During each CE, the Central and Peripheral will exchange data at least once for data transmission or connection keep-alive. The Central will first sends an empty or data packet to the Peripheral. Then the Peripheral will sends a response after inter-frame size (IFS) time T_{IFS}, and the data exchange will end after minimum subevent space (MSS) time T_{MSS}. The IFS time is 150us and the MSS time should be no less than 150us. Each CE can contain multiple data exchanges depending on how much data the Central and Peripheral want to send, but the length should not exceed the *max connection event length* t_{CE}.

In multi-connection scenario, there are two types of collisions that can occur. First, while scheduling the connections, the BLE link-layer usually reserves a minimal CE length for each CE to ensure it can contain one or more data exchanges. However, the minimal CE length of multiple connections can overlap with each other. A common policy is to schedule the connection with the least last schedule time, and we call this connection preempts the others (e.g., Peripheral 1 preempts Peripheral 2 in Fig. 1). Second, if there are anchor points of other connections positioned between two consecutive anchor points of the current connection, the current CE is blocked and the CE length is shorter (e.g., Peripheral 2 blocks Peripheral 1 in Fig. 1). To identify each CE, the BLE connection has a 16-bit CE counter. It starts from 0, and it should always be plus 1 no matter the CE is normally performed or preempted by other connections.

B. Connection Update and Connection Subrating

The Bluetooth specification designs the connection update procedure for connection re-scheduling (Fig. 2). The Central starts the procedure by sending the conn_upd control packet to the Peripheral, which includes the new connection

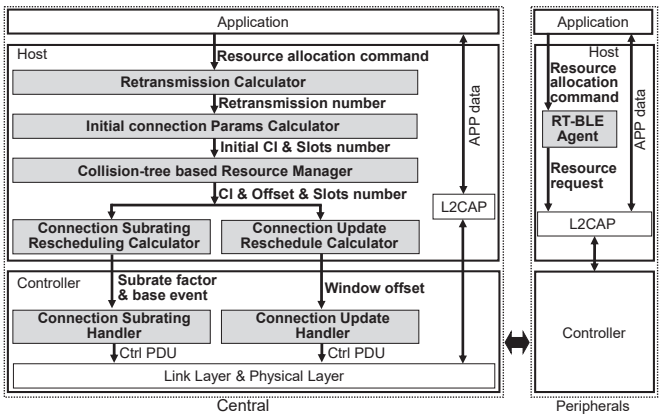


Fig. 4: System overview of RT-BLE.

parameters and the offset and size of transmit window. There is a mandatory delay between the control packet being sent and new parameters are applied, which is *instant* $\times$ t_{CI} and the *instant* should be larger or equal than 6. After that, the Central will delay *transmit window offset* and transmit a packet anywhere during *transmit window size* time. The transmit window offset/size must be an integer multiple of 1.25ms. By tweaking the *transmit window offset/size*, the Central can reposition anchor points. However, its delay is considerable, especially when the CI is large.

The connection subrating is proposed in Bluetooth specification v5.3, which allows the connection to skip some CEs (Fig. 3). The Central and Peripheral always wake up at the *subrate base events* to exchange data or keep-alive packets, and the number of CEs between two consecutive base events is called *subrate factor*. Besides the base event, the nodes can exchange data in other CEs if at least one of the previous *continuation number* CEs has data packets transmitted. To simplify, we call subrate base events as base CEs and other CEs as non-base CEs in the rest of the paper. To change these three subrating parameters, the Central transmits a subrate_ind packet to the Peripheral, and it will reply a link-layer ACK immediately. After that, the new subrating parameters are applied. Although the delay to apply the new subrating parameters is small, the granularity is coarse, especially with a large CI.

IV. System Overview

In this section, we discuss the design goals of *RT-BLE* and overview our system design.

A. Design Goals

- **Latency-aware.** The timeliness is recognized as a critical performance metric of IoT applications and the requirements can change at runtime. *RT-BLE* should allocate enough and proper time resource for each connection.
- **High capacity.** While ensuring the latency requirement of each connection, *RT-BLE* is supposed to connect as many connections as possible.
- **Light-weight.** The computational resource on embedded devices is quite limited. *RT-BLE* should be lightweight enough for modern BLE chips.
- **Fast re-scheduling.** The traditional method suffers from a high delay. *RT-BLE* is supposed to achieve a fast connection re-scheduling method.

B. RT-BLE Architecture

In Fig. 4, we show the birds-eye view of *RT-BLE*'s system architecture. Here, one Central can simultaneously connect with multiple Peripherals. Each Peripheral has two work modes: normal and critical mode. Users can set the percentile worst-case latency requirements for each mode (e.g., the worst-case latency of 95% transmissions is less than 200ms). Generally, the normal mode has less strict latency requirements to save energy. Once there are exceptions in the sensor data, nodes can quickly enter the critical mode, which has more strict latency requirements. The *RT-BLE* has eight modules. Six of them are located in the BLE host of the Central, away from the controller, which includes the link-layer. Therefore, *RT-BLE* will not interfere real-time tasks in the link-layer and normal operations of BLE. The communication of Central and Peripheral is through the Logical Link Control and Adaptation Protocol (L2CAP). Before they start data transmission, the application layer sends a resource allocation command to *RT-BLE*, which gives the basic information of application requirements, including the data packet length and the percentile worst-case latency requirement. *RT-BLE* uses a centralized way to schedule time resource. So, the *RT-BLE* agent on the Peripheral translates the resource allocation command into a resource request to the Central. Then, *RT-BLE* goes through the following modules to allocate resource and apply the corresponding parameters:

Retransmission calculator. Given the percentile latency requirement, this module calculates how many retransmissions are required to cover the specified number of data transmissions. The packet loss rate can be given through prior measurements or dynamically obtained from the link-layer. The details of this module are in Sec. V.

Initial connection parameters calculator. This module estimates the worst-case transmission latency and preliminarily calculates the required CI and the length of the CE. We call this set of parameters as the initial connection parameters set. The initial CI is taken as the largest value that meets the latency requirements, so the initial connection parameters set is the most energy-efficient one. The details are in Sec. VI-B.

Collision tree based resource manager. This module will tweak the initial parameters and checks whether there is enough time resource for the connection. If there are no enough resource, *RT-BLE* has to refuse the resource allocation command or request. Details are introduced in Sec. VI-C.

Connection subrating/update re-scheduling calculator. *RT-BLE* uses connection subrating and connection update to re-schedule the connections, and these two modules calculate the parameters for each procedure. The details of the parameters calculation is presented in Sec. VI-D.

Connection subrating/update handler. These two handlers actually perform connection subrating and connection update

Algorithm 1 Calculating retransmissions number

Input: Packet loss rate P; The percentile of transmissions p; Number of data PDUs n;
Output: Number of retransmissions n_{re};

$p_{\text{re}} \leftarrow (1-P)^n$; $c \leftarrow p_{\text{re}}$; $n_{\text{re}} \leftarrow 0$
while $p_{\text{re}} < p$ **do**
  $n_{\text{re}} \leftarrow n_{\text{re}} + 1$
  $c \leftarrow c \times P \times \frac{n+n_{\text{re}}-1}{n_{\text{re}}}$
  $p_{\text{re}} \leftarrow p_{\text{re}} + c$
end while
return n_{re}

procedures, respectively. After the procedures are finished, the connection re-scheduling is considered as done.

V. BLE Timeliness Modeling

Although there has been some works that formulate the transmission performance of BLE in noisy RF environment, they ignore the retransmission mechanism of the BLE. In these section, we start with the data transmission latency with ideal channel. Then we investigate the BLE retransmission mechanism and introduce the packet loss into our model. After that, since *RT-BLE* uses connection subrating to achieve fast connection re-scheduling, we discuss the impact of connection subrating to the transmission latency.

A. Data transmission time

We set the Central and Peripheral have l_{c} an l_{p} bytes data to transmit, respectively. The L2CAP layer will add a two bytes length field and fragment the data into multiple protocol data units (PDU). The maximum length of each PDU is 247 bytes. We denote the number of PDU of Central and Peripheral as n_{c} and n_{p}, respectively. The length of the last PDU can be less than 247 bytes, we denote them as l_{lc} and l_{lp} for the Central and Peripheral, respectively. The transmission time for each PDU with l bytes payload is denoted as $t_{\text{PDU}}(l)$, which includes the time to transmit the preamble (1B), access address (4B), link-layer-header (2B), L2CAP header (2B), CRC code (3B), and the payload. If $l = 0$, there is no need the transmit the L2CAP header. To ensure each CE can start exactly at the anchor point, the BLE link-layer will start executing the corresponding link-layer task slightly before the anchor point, we call this the start-up time, denoted as T_{s}. It is used for data preprocessing and radio ramping up (e.g., 213us for using NimBLE and nRF52840). Therefore, the total time for data transmission is $t_{\text{data}} = T_{\text{s}} + \max(n_{\text{c}}, n_{\text{p}}) \times (T_{\text{IFS}} + T_{\text{MSS}}) + t_{\text{PDU}}(l_{\text{lc}}) + t_{\text{PDU}}(l_{\text{lp}}) + t_{\text{PDU}}(247) \times (n_{\text{c}} + n_{\text{p}} - 2)$.

B. Packet loss

For BLE, if there is a data packet lost, the link-layer will retransmit the packet until it is successfully received. Therefore, the worst-case transmission latency becomes unbounded with packet loss. The *RT-BLE* allows users to set the percentile worst-case latency of each connection (p, t_{ddl}), where the p is a percentage between 0 and 1. t_{ddl} is the latency requirements. That means p of the data transmissions' worst-case latency is lower than t_{ddl}. To achieve this requirement, we first determine how many retransmissions are needed to cover p data transmissions. We denote the packet loss rate as P. If the Central or Peripheral wants to transmit n PDUs, the possibility that all of them are correctly received without any retransmissions is $(1-P)^n$. If there are n_{re} packet loss occurred before all data is correctly received, the node has totally transmitted $n + n_{\text{re}}$ PDUs since the number of retransmissions should be same as the number of packet losses. The last one should always be correctly transmitted, and the other PDUs can include errors, so the probability of transmitting n PDUs with n_{re} transmissions is $(1-P)^n P^{n_{\text{re}}} \binom{n+n_{\text{re}}-1}{n_{\text{re}}}$, where $\binom{n+n_{\text{re}}-1}{n_{\text{re}}}$ is the combination calculator. Therefore, the percentage of transmissions that include n data PDUs and retransmission number is less or equal to n_{re} is:

$$p_{re}(n_{\text{re}}) = (1-P)^n \sum_{i=0}^{n_{\text{re}}} P^i \binom{n+i-1}{i}. \quad (1)$$

We need to find the minimal n_{re} that satisfy $p_{re}(n_{\text{re}}) >= p$. However, directly getting the numerical solution is too complex for embedded MCUs. The retransmission calculator uses an iterative algorithm. First, we simplify the Eq. 1 to:

$$p_{re}(n_{\text{re}}) = (1-P)^n \sum_{i=0}^{n_{\text{re}}} P^i \frac{(n+i-1)\cdots(n+1)n}{i!}. \quad (2)$$

We denote the part inside the summation as c_i, and c_{i+1} can be presented by c_i iteratively:

$$c_{i+1} = P\frac{n+(i+1)-1}{i+1} c_i. \quad (3)$$

Based on Eq. 3, we propose an iterative algorithm to get the n_{re}. The pseudo-code is shown in Alg. 1. With this algorithm, we can get the number of retransmissions needed by Central (n_{rec}) and Peripheral (n_{rep}).

For now, we have determined the number of retransmissions. However, when these packet losses occur can result in quite different latency. We investigate the retransmission mechanism of BLE and classify all packet loss cases into the following three scenarios: **(1) One loss during CE.** If there is only one packet loss during the whole data transmission process, the transmitter will retransmit the lost PDU at the next data exchange and will not lead to any extra CE to retransmit this PDU. **(2) Two consecutive losses in one CE.** If there are two consecutive packet losses in a same CE, it will be terminated and the remaining data transmissions should wait until the next CE. So, some extra CEs are needed for data retransmissions. The max number of two consecutive packet losses is $\lfloor n_{\text{rec}}/2 \rfloor + \lfloor n_{\text{rep}}/2 \rfloor$. Since there should be at least two data exchanges performed before the CE is terminated by packet loss, the maximum number of extra CEs should be no more than half of the data exchange number. **(3) One or more packet loss at the last data PDU.** Before performing the last data exchange, both the Central and Peripheral have no more data to send. So, they both set the more data flag at the last two link-layer PDU headers to 0 and will turn off their radio after this data exchange. If any of the two PDUs is

lost, it should be retransmitted in the next CE. Therefore, the maximum number of extra CEs caused by the packet loss in the last data exchange is $\max(n_{\text{rec}}, n_{\text{rep}})$, which should always larger or equal than the extra CE number in the former two situations. In conclusion, the worst-case latency occurs when all packet losses are happened on the last data PDU. The time of the data exchange which contains the last retransmission t_{last}, which includes the time for retransmission PDUs and one or zero empty PDU.

C. Impact of connection subrating

The *RT-BLE* uses connection subrating to achieve fast connection re-scheduling, and the continuation number can be set to 0 or 1 according to different requirements. The detail of the subrate-based fast connection re-scheduling is presented in Sec. VI-D. When the continuation number is 0, the maximum number of extra CEs is:

$$n_{\text{eCE}}^{(1)} = sf \times \max(n_{\text{rec}}, n_{\text{rep}}), \tag{4}$$

where the sf is the value of the subrate factor. If the continuation number is 1, the Central and Peripheral can perform data exchanges in n_{lim} CEs (including the base and non-base CE) after each base CE. The value of n_{lim} will be given during the resource allocation, and its detail is presented in Sec. VI-C. During the data transmission, there would be $\max(n_{\text{c}}, n_{\text{p}})$ times data exchanges. The remaining retransmissions happen for the last data PDU in the worst-case. We denote the number of remaining retransmissions of Central and Peripheral in extra CEs as n_{crem} and n_{prem}, respectively. They can be presented as $n_{\text{crem}} = \max(n_{\text{c}} + n_{\text{rec}} - \max(n_{\text{c}}, n_{\text{p}}), 0)$ and $n_{\text{prem}} = \max(n_{\text{p}} + n_{\text{rep}} - \max(n_{\text{c}}, n_{\text{p}}), 0)$.

There are three situations while retransmitting these packets. First, if the $n_{\text{crem}} = n_{\text{prem}}$, both two roles can transmit a packet in the n_{lim} CEs after each base CE. If n_{crem} and n_{prem} are 0, then the number of extra CEs is 0, Otherwise, the number is:

$$n_{\text{eCE}}^{(2)} = sf \times \left(1 + \left\lfloor \frac{n_{\text{crem}} - 1}{n_{\text{lim}}} \right\rfloor\right) + \text{mod}\,(n_{\text{crem}} - 1, n_{\text{lim}}). \tag{5}$$

Second, if $n_{\text{crem}} > n_{\text{prem}}$, the first n_{prem} data exchanges can be preformed in non-base CEs because both the Central and Peripheral have data for transmission. However, the Peripheral will only turn on its radio on base CE after the n_{prem}th CE since it neither has data to transmit nor received a valid data packet from the Central. The Central will still retransmit packets at non-base CEs, although it can not receive the acknowledgment. Therefore, if the last retransmission of Central is at a non-base CE, it has to retransmit the packet one more time at the next base CE. The maximum number of extra CEs in this situation is:

$$n_{\text{eCE}}^{(3)} = sf \times \left(1 + \left\lceil \frac{n_{\text{crem}} - 1}{n_{\text{lim}}} \right\rceil\right). \tag{6}$$

The third situation is when $n_{\text{crem}} < n_{\text{prem}}$. The first n_{crem} data exchanges can be performed at the non-base CE. As for the last $n_{\text{prem}} - n_{\text{crem}}$ PDUs, the Central does not have more data to transmit and has not received valid data PDUs from the Peripheral, so it will only start a data exchange in subsequent base CEs. Therefore, the Peripheral can only retransmit the remaining PDUs in base CEs, and the maximum number of CEs under this situation is:

$$n_{\text{eCE}}^{(4)} = sf \times \left(1 + \left\lfloor \frac{n_{\text{crem}} - 1}{n_{\text{lim}}} \right\rfloor + n_{\text{prem}} - n_{\text{crem}}\right). \tag{7}$$

In conclusion, The number of extra CEs n_{eCE} that required for data retransmission can be calculated from the Eq. 4, and Eq. 5-7 in different situations.

VI. RT-BLE Design

In this section, we first introduce some necessary concepts used in *RT-BLE*. Then, introduce how to use the model above to get the initial connection parameters. After that, we present the collision tree based time resource management technology. Finally, we introduce how to achieve fast connection re-scheduling with connection subrating feature.

A. Basic Concept of RT-BLE

The main purpose of *RT-BLE* is to efficiently manage the time resource of the Central. Existing works usually consider time resource to be continuous, such as the BLEX [10] and BLEach [28]. However, we find that lots of parameters of BLE must be integer multiple of 1.25ms, such as the CI, transmit windows offset. The reason is the Bluetooth specification sets the time for a single Tx/Rx slot to 0.625ms and the minimal time for a Tx/Rx cycle is 1.25ms. Slicing the time resource into slots can improve the efficiency of resource allocation, especially on embedded devices. Furthermore, we find it is unnecessary to schedule the time resource with the 1.25ms slot because it is too short for Central and Peripherals to efficiently transmit any data, we can choose a longer unit to further reduce memory and computational resource overheads. Therefore, *RT-BLE* divides the time resource into *virtual slots*. Each length of virtual slot is an integer multiple of 1.25ms and the length of CI and CE are integer multiples of virtual slots. The length of the virtual slot is a trade-off between the resource management accuracy and computational overhead. We set the length of the virtual slot to four slots (5ms) since it is the minimum length to ensure the Central and the Peripheral can both transmit a PDU with maximum payload. Therefore, at least one data exchange can be guaranteed in each virtual slot. It is a reasonable length and also NimBLE takes the 5ms as the minimal CE length.

To serve the subrating-based fast connection re-scheduling (details in Sec. VI-D), we first fix the CI to 10ms, which is equal to two virtual slots and slightly over the minimal CI (7.5ms). Then, we use the subrate factor sf to "mimic" different CIs . For example, the sf is 4 to mimic 40ms CI. We call the mimicked CI as *equivalent CI*, denoted as t_{eCI}. However, in resource allocation, we can allocate s number of virtual slots to each CE. There would be no problem if the s is less or equal to 2. Otherwise, the length of CE exceeds the 10ms native CI. To solve this, *RT-BLE* will set the continuation number to 1 and $n_{\text{lim}} = \lceil s/2 \rceil$ when $s > 2$. So, the nodes can exchange data in n_{lim} CEs every t_{eCI}.

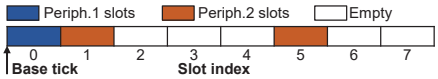


Fig. 5: An example of virtual slot table. Two Peripherals with 20ms and 40ms CI are connected.

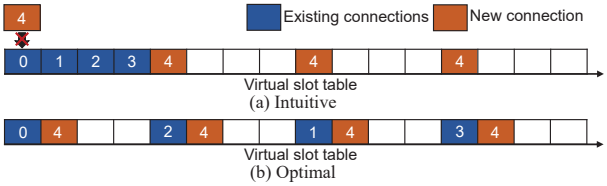


Fig. 6: Different time resource management policy.

To manage multiple connections, *RT-BLE* uses a virtual slot table to store the occupancy of each virtual slot. The length of the virtual slot table is the least common multiple of all connections' equivalent CIs. Therefore, one virtual slot table contains the complete time resource scheduling. However, the length of the virtual slot table will explode if the equivalent CIs are mutually prime. It would be a great threat for resource-constraint devices. To solve this, we borrow the design in BLEX [10]. It specifies that the CI must be $2^n \times 1.25$ms, where n is an integer between the 3 and 11. In *RT-BLE*, we set the subrate factor to $sf = 2^n$, where the $0 \leq n \leq 8, n \in N$. The equivalent CI can cover the range from 10ms to 2560ms. An example of virtual slot table is shown in Fig. 5. The equivalent CI of Peripheral 1 and Peripheral 2 are 40ms and 20ms, respectively. The start time of the first slot is the base tick, and it should be updated once a virtual slot table is finished. The offset is the slot index where the first slot of the connection is, denoted as off (i.e., 0 for Peripheral 1, 1 for Peripheral 2). It is worth noticing that, the length of the virtual slot can be $2^n \times 1.25$ms, $n \in N$ and there is no conceptual change required for our policy (details in Sec. VI-C).

B. Initial Connection Parameters Calculator

With the percentile worst-case latency requirement (p, t_{ddl}), this module calculate the most energy-efficient initial connection parameters. To be more straightforwardly, to find a set of connection parameters with the longest equivalent CI while meeting the latency requirement. The number of extra CEs for retransmission is fixed once given the data packet length and requirement. To increase the equivalent CI, the only thing we can do is to allocate enough virtual slots to the connection so that all the data PDUs can be transmitted in one equivalent CI. Therefore, we set the initial number of virtual slots to $s_{\text{ini}} = \lceil t_{\text{data}}/5\text{ms} \rceil$. The end-to-end latency consists of waiting time, data transmission time, and retransmission time. The waiting time is the time from the data packet enqueued into Tx buffer until it starts to be transmitted, and the maximum value is t_{eCI}. The end-to-end latency should be less or equal to the latency requirement:

$$t_{\text{e2e}} = (sf + n_{\text{eCE}}) \times 10\text{ms} + t_{\text{last}} \leq t_{\text{ddl}}. \quad (8)$$

Here, all the variables except the sf are fixed. The sf is a power number of 2, between 2 and 512. We choose the largest sf that satisfy Eq. 8 as the initial subrate factor sf_{ini}.

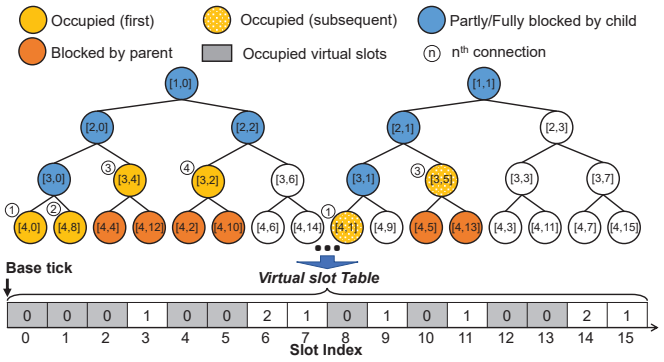


Fig. 7: An example of collision tree. It contains with four connections with different application requirements.

C. Collision-tree based Resource Management Module

An intuitive way to manage time resource is using a greedy policy that merge all connections at the beginning of the virtual slot table, like the BLEX [10] does. However, the capacity is low because this policy does not consider the periodic nature of BLE. Fig. 6(a) gives an example of that. There are already four connections with 80ms equivalent CI. If now we have a new connection with 20ms equivalent CI, there will be a denial of service (DoS) to this connection request. A better way to allocate virtual slot is shown in Fig. 6(b). To find the optimized time resource allocation, we propose collision tree based time resource management technology. An example of the collision tree is shown in Fig. 7. Each node in collision tree represents a resource block and the level of the tree is from 1 to 9. We use $[lv, off]$ to present a resource block, where the lv is the level in the collision tree and the interval between two consecutive virtual slots in it is $2^{lv} \times 5$ms ($sf = 2^{lv-1}$). The off means the offset in virtual slot table, and each connection can occupy one or more resource blocks with continuous offsets. With the collision tree, we can easily find out the relationship of all resource blocks. More specifically, if any of the node's parent or child nodes are occupied, the corresponding resource block collides with others and can not be allocated to any connection.

Although there are some works that use a similar design to manage the time resource in IEEE 802.15.4 [22], there are still two major issues that need to be solved for BLE:

1) How to assign multiple continuous resource-block for one connection: Existing works only assign one resource block to each connection. Therefore, to transmit large data, it has to assign a resource block with a small lv. For BLE, the more energy-efficient way is to allocate s resource blocks with continuous offset and a large lv. For ease of searching resource, the virtual slot table contains the *number of continuous free virtual slots* (NCF) for each virtual slot. In Fig. 7, the gray block in the virtual slot table means this virtual slot is occupied and the NCF should be 0. Otherwise, the NCF shows how many virtual slots are available after the current virtual slot (including the current one). Therefore, to check whether a resource block can be the first one among the s resource blocks allocated for this connection, we just need to confirm that the NCF of all virtual slots in this resource block are greater or equal than s. Once a new virtual slot

Algorithm 2 Find resource in collision tree

Input: The initial resource blocks number s_{ini} and level lv_{ini};
Output: Level in the collision tree lv; Offset in virtual slot table off; The number of resource blocks s;

$s \leftarrow s_{\text{ini}}$
for lv from lv_{ini} **to** 1 **do**
  $\{n_{\text{left}}, n_{\text{right}}\} \leftarrow$ get_free_number(lv)
  if $n_{\text{left}} \geq n_{\text{right}}$ **or** $(n_{\text{right}} - n_{\text{left}} = 1$ **and** $\text{mod}(s, 2) = 1)$ **then**
    $\{lv, s, off\} \leftarrow$ search_from_left_subtree()
  else
    $\{lv, s, off\} \leftarrow$ search_from_right_subtree()
  end if
  if find nodes **then**
    return $\{lv, s, off\}$;
  else
    $s \leftarrow \lceil s/2 \rceil$
    ensure_latency_requirement($lv - 1, s$)
  end if
end for
return No enough resource left;

is occupied, it needs to update the NCF backward until it encounters another occupied virtual slot or reaches the leftmost side of the repeating unit. The new NCF starts from 0, and once moves backward one virtual slot, it should add one.

2) How to find suitable resource blocks: The initial connection parameters calculator gives the initial subrate factor sf and the initial virtual slot number s_{ini}. So, we start searching at the level $lv_{\text{ini}} = sf_{\text{ini}} + 1$ to find s_{ini} continue resource blocks. Because our subrating-based fast connection re-scheduling method (details in Sec. VI-D) can achieve lower delay if the offsets of the current resource block and the target one are both odd or even. The offset of the resource blocks is even in the left subtree and is odd in the right one. When the connection tries to enter the critical mode, we expect that the first one of new resource blocks is in the same subtree as the current resource block. Therefore, we should balance the number of free resource blocks in the two subtrees. To do so, if there are more or equal free resource blocks in the left subtree or the number of free resource blocks in the left subtree is one less and s_{ini} is an odd number, we start searching for free resource from the left subtree. Otherwise, we first search in the right subtree then the left one. Within the subtree, we search for resource blocks in the order they appeared in the tree. For example, in Fig. 7, the connection 1 has $lv_{\text{ini}} = 4$ and $s_{\text{ini}} = 2$, so it take the $[4, 0]$ and $[4, 1]$. The connection 2 has $lv_{\text{ini}} = 4$ and $s_{\text{ini}} = 1$. Since there same number of free resource blocks, we start searching at the left subtree and find the $[4, 8]$. The connection 3 and 4 both search from the left one and takes the $[3, 4]$ and $[3, 2]$ as their first resource blocks, respectively. With this searching order, we can make full use of those virtual slots which are belong to partly occupied resource blocks and reserve as many resource blocks with low lv as possible. If there is no enough resource blocks in lv_{ini}, we will move to the upper level, half the number of virtual slots, and check whether this new resource block can ensure the latency requirement.

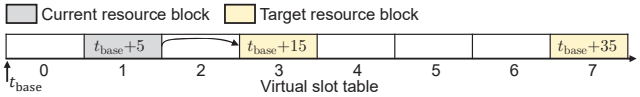


Fig. 8: Subrating-based fast connection re-scheduling to move the anchor point by an even number of virtual slots.

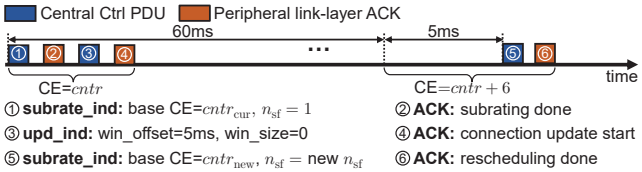


Fig. 9: Subrating-based fast connection re-scheduling to move the anchor point by an odd number of virtual slots.

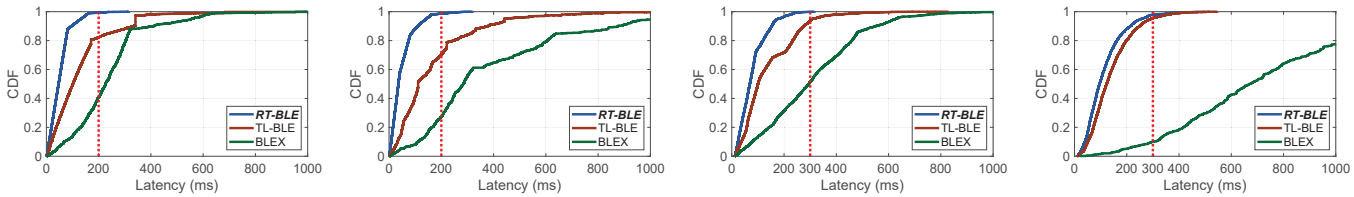
This procedure will repeat until we find resource or get to the top of the tree. The pseudo-code is shown in Alg. 2.

To reduce the memory consumption, *RT-BLE* stores the nodes in each level as an array. The order of nodes in the array is same as the order in which they appear in the tree. Each node is stored as a 8-bit number. Three bits are used to indicate the state of the node (e.g., free, occupied, etc.). The other five bits are used to store the corresponding connection handle. To converse between the offset of the resource block and the corresponding node index in the array We find that there is a reverse-bits relationship between the offset of the resource block and the corresponding node index. For example, the index of node $[3, 6]$ is 011b and the offset of the resource block is 110b. Therefore, we can achieve the conversion with O(1) complexity. The memory consumption for the collision tree is 1,022B. As for the virtual slot table, it stores the 16-bit NCF of each virtual slot and the memory consumption is 1KB. It is worth mentioning that, even if the length of virtual slot is set to the minimal 1.25ms, the total memory consumption is 8,185B, which is totally acceptable for modern BLE chips (e.g., nRF52840 has 256KB RAM).

D. Subrating-based Fast Connection Rescheduling Method

Then, we propose a fast connection re-scheduling method based on connection subrating feature since the connection subrating can update its parameters without extra delay.

If the length of the anchor point movement is an even multiple of virtual slots, we can achieve it by changing the base CE of the connection. To change the base CE, Central should calculate the counter of new base CE and send it in a subrate_ind PDU. The connection management module gives s resource blocks and we take the first one as the target $[lv, off]$. Then, *RT-BLE* calculates the start time of virtual slots in the target resource block and chooses the closest virtual slot after the current base CE as the target virtual slot. After that, it calculates the time difference t_{diff} between current base CE and the start time of the target virtual slot, and the CE counter difference is $cntr_{\text{diff}} = t_{\text{diff}}/5\text{ms}$. Therefore, the new base CE counter is $cntr_{\text{new}} = cntr_{\text{cur}} + cntr_{\text{diff}}$, where the $cntr_{\text{cur}}$ is the counter of current base CE. When the parameters have been calculated, Central will send subrate_ind to the Peripheral, the Peripheral reply a link-layer ACK, and this connection subrate procedure is considered as done. Fig. 8



(a) 8 nodes, 100B data, 10% loss (b) 8 nodes, 100B data, 40% loss (c) 4 nodes, 1024B data, 10% loss (d) 2 nodes, 1024B data, 40% loss

Fig. 10: End-to-end data transmission latency with different packet size and packet loss rate.

gives an example that the connection switch from $[3, 1]$ to $[2, 3]$. The start time of current base CE is t_{base}+5. The closest virtual slot after it is the slot 3 and its start time is $t_{\text{base}}+15$. So, the time difference is 10ms and new counter is $cntr_{\text{cur}}$+1.

If the length of the anchor point movement is an odd multiple of virtual slots, we should first move one virtual slot with connection update procedure and than use the method described above to move even number of virtual slots. However, the connection update procedure suffers from a mandatory delay of $6 \times sf \times t_{\text{CI}}$. The key idea to solve this problem is we change the subrate factor to 1 before starting connection update procedure. Therefore, the mandatory delay is shirked to 60ms, and will not increase with the equivalent CI. Fig. 9 shows the details. At the beginning, the Central will transmit two control PDUs. The first one is subrate_ind, which changes the subrate factor to 1. The second one is upd_ind, which set the transmit window offset to 5ms and the window size is 0. It can move the anchor point to the right for one virtual slot. Since the control packets are relatively short, they can all be transmitted in one CE. The connection update procedure starts after the upd_ind is transmitted. At the last CE of connection update procedure, the Central transmits another subrate_ind PDU to move the anchor point by an even multiple of virtual slots and update subrate factor. The re-scheduling is done after the acknowledgment.

VII. Performance Evaluation

To evaluate the performance of *RT-BLE*, we implement it with RIOT OS [34] and NimBLE protocol stack [15]. The modules in the BLE host are implemented as a system module in RIOT OS. The other two modules in the controller are in ble_ll_conn.c. The experiment platform is the popular Nordic nRF52540DK board with a build-in nRF52840 chip [14]. This chip uses a 32768Hz crystal, so we take 164 ticks (5004us) as the length of virtual slot. The experiment is performed in an office. To make the packet loss rate controllable, we set the transmit power to the max value to cancel as much interference from the real world as possible. Then we manually inject packet loss into the link-layer with the target loss rate.

For comparison, we implement BLEX [10] and Timeliness BLE (TL-BLE) [11] with NimBLE. We make two modifications to the BLEX: (1) For CE length prediction, we only use the history CEs with data transmissions. (2) We use our model with ideal channel to set the CI and the CE length before establishing the connections and make sure the initial CE length is larger than 5ms so that the native NimBLE can establish connections normally. For the TL-BLE, it needs to pre-measure the number of CEs to transmit a theoretically longest packet in one CE. We set the length to 245 bytes since in the worst-case, the native NimBLE only ensures 5ms for each CE and each role can only transmit one L2CAP packet.

TABLE I: Model error with different packet loss rate (‰).

Tx Role	Pkt. size	Ideal	10%	20%	30%	40%
Central	100B	1.697	0.201	0.201	0.202	0.193
	1024B	1.043	0.172	0.180	0.176	0.172
Peripheral	100B	6.913	0.089	0.980	0.757	0.711
	1024B	0.737	0.589	0.116	0.742	0.128

A. Latency Guarantee

We evaluate the latency guarantee of *RT-BLE* with two different traffic rates. The first one has eight Peripherals that transmitting 100B data every 500ms, and the latency requirement is $(95\%, 200\text{ms})$. The second one has up to four Peripherals each transmitting 1KB data every 500ms, which requires three virtual slots. The latency requirement for each connection is $(90\%, 300\text{ms})$.

Fig. 10 shows the end-to-end latency with different traffic rates and packet loss rates. The *RT-BLE* can guarantee latency requirements in all scenarios. The reason is two-fold. First, we propose an accurate latency model in noisy RF environment and use it to allocate proper time resource for each connection. Second, *RT-BLE* uses a collision tree to avoid the connection collisions. To evaluate the accuracy of estimating the worst-case transmission time. We manually inject n_{re} times packet losses to the last PDU and measure the data transmission time, where the n_{re} is calculate by Alg. 1. Tab. I shows the error between the estimation and real transmission time. The error of our model is less than 6.913‰. The BLEX fails to provide reliable service. For small traffic rate, the CE length is relatively short, and BLEX merges the time resource that the time between two anchor points of different connections can be less than 5ms, which are considered as collisions. For the large traffic scenario, the packet loss will cause significant fluctuations in CE length, which makes the CE length prediction harder. Also, the BLEX will not adapt the CI to different packet loss rates, leading to large retransmission latency. The TL-BLE performs better than BLEX. However, the CI of TL-BLE is shorter than *RT-BLE* because its model does not consider the underlying BLE link-layer behavior. For example, in the Fig. 10(d), the CI of TL-BLE is 32.5ms and 40ms for *RT-BLE*. The reason why TL-BLE still has higher latency is TL-BLE is built on native BLE link-layer, the collision issue gets worse when the number of Peripherals and packet length are increased.

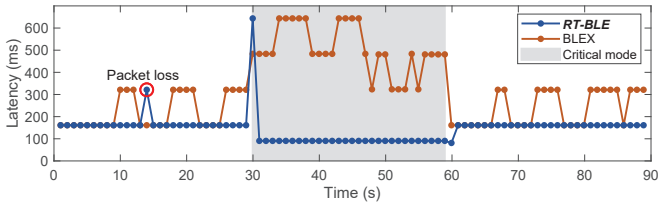


Fig. 11: The worst-case latency during requirement changes.

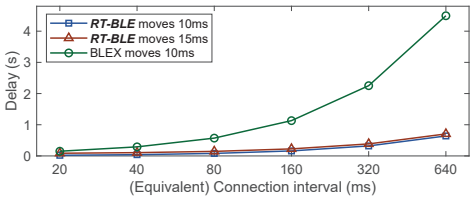


Fig. 12: The connection re-scheduling delay.

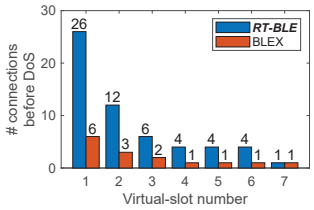


Fig. 13: System capacity.

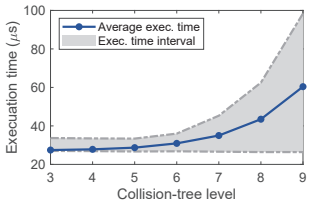


Fig. 14: Execution time.

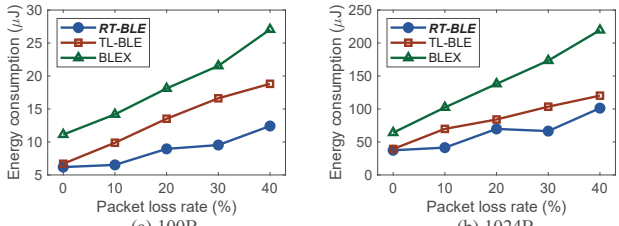


Fig. 15: Average energy consumption to transmit one packet.

B. Online Adaptation

To show *RT-BLE* can quickly adapt to the requirement changes, we connect the Central with eight Peripherals without injecting any packet loss. In the first 30 seconds, all Peripherals work in the normal mode that transmit 100B data every 1s and the worst-case latency is 200ms. During the 30s and 60s, the first Peripheral enters the critical mode to transmit a 1KB packet every 1s, and the worst-case latency is 100ms. After the 60s, the first Peripheral returns to the normal mode.

Fig. 11 shows the timeline of the worst-case latency during 90 seconds. *RT-BLE* can quickly adapt to the latency requirement changes. During normal mode, the transmission at 14s suffers from interference from the real world, so the worst-case latency exceeds the 200ms limitation. Except this one, all the other transmissions satisfy the limitation and the average latency is 166.72ms. During the critical mode, the worst-case latency of the 30th transmission is high, since there are no enough time resource has been allocated to this connection yet and the Central has to transmit control PDUs for connection re-scheduling. After that, *RT-BLE* is adapted to the traffic rate and latency requirement changes. Also, *RT-BLE* can quickly return to the normal mode after 60s. BLEX is failed in this scenario. In normal mode, there are multiple transmissions that violate the latency limitation because the CE length is small and BLEX merges the connections too closely to avoid collisions. While in the critical mode, BLEX seriously violates the latency requirement. The reason is two-fold. First, once the latency requirement is changed, the only thing BLEX can do is extend the CE. However, it is not enough to achieve the 100ms latency. Second, the connection re-scheduling method used by BLEX suffers from a long delay. What's worse, before extending the CE of the first Peripheral, BLEX should first move the remaining seven connections to the right. In Fig. 12, we compare the time to re-schedule one connection. To move 2 virtual slots (10ms), *RT-BLE* can reduce 85.74% to 86.25% delay compared with the traditional method used in BLEX. This because we utilize the connection subrating feature that can update its parameters with minimal delay. If the length of movement is three virtual slots (15ms), *RT-BLE* should first use connection update to move the anchor point 5ms to the right, so there is an extra delay of 65.11ms compared with the one move 10ms. This extra delay is independent of the equivalent CI, since *RT-BLE* will change the subrate factor to 1 before it starts the connection update procedure.

C. Capacity and System Overhead

To evaluate the capacity of *RT-BLE*, assume that there is no packet loss and there is a connection that needs one virtual slot and the worst-case latency is 50ms. Before the *RT-BLE* deny to provide service to this connection, how many connections which has 200ms worst-case latency requirement and different numbers of virtual slots can be connected. Fig. 13 shows the capacity of *RT-BLE* is up to 4.33× higher than BLEX. The reason is *RT-BLE* utilizes a collision tree to optimally allocate time resource but the BLEX uses a greedy policy.

Fig. 14 shows the time to search any node in different level. The average searching time increased from 27.45us to 60.39us with the increase of tree level. The lower bound is relatively fixed, it happens when searching the first node in each level. The upper bound of execution time is 98.40us, which is acceptable for BLE connection establishment. We evaluate the average energy consumption for the Peripherals to complete one data transmission, which is obtained by multiplying the average power and the end-to-end latency. The scenario is same as the Sec. VII-A. In Fig. 15, *RT-BLE* can reduce 7.51% to 55.72% energy consumption for Peripheral transmitting 100B data, and 4.89% to 61.64% for 1KB data.

VIII. Conclusion

This paper proposes *RT-BLE*. First, we introduce a timeliness model which precisely formulate the retransmission mechanism of BLE. *RT-BLE* uses it to calculate how much time resource is needed for each connection. Then, we propose a collision tree based time resource manage technology to optimally manage time resource. Finally, we propose a subrating-based fast connection re-scheduling method without violating the Bluetooth specification. Our experiment result show *RT-BLE* can provide reliable service even with heavy packet loss. The error of our model is less than 0.69%. The re-scheduling delay is reduced by up to 86.25% and the capacity is up to 4.33× higher than existing work.

REFERENCES

[1] A. Research, "Bluetooth Low Energy Market Set to Triple by 2023, Reaching 1.6 Billion Device Shipments," https://www.abiresearch.com/market-research/product/1032422-wireless-connectivity-technology-segmentat/, 2019.

[2] C. Lazaroiu and M. Roscia, "BLE To Improve IoT Connection in the Smart Home," in *Proc. of IEEE ICRERA*, 2021, pp. 282–287.

[3] H.-K. Wu, "Development of A Low Power Temperature Sensing Device Based on BLE Technology," in *Proc. of IEEE ICKII*, 2021, pp. 58–59.

[4] J. Razavi and N. Brennan, "A novel application of Bluetooth technology for detection of forest fires," in *Proc. of IEEE ICWiSE*, 2016, pp. 66–70.

[5] H. Karvonen, K. Mikhaylov, M. Hämäläinen, J. Iinatti, and C. Pomalaza-Ráez, "Interference of wireless technologies on BLE based WBANs in hospital scenarios," in *Proc. of IEEE PIMRC*, 2017, pp. 1–6.

[6] 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, 2018.

[7] P. Paul, N. Dutta, B. A. Biswas, M. Das, S. Biswas, Z. Khalid, and H. N. Saha, "An internet of things (IoT) based system to analyze real-time collapsing probability of structures," in *Proc. of IEEE IEMCON*, 2018, pp. 1070–1075.

[8] M. Bartholmai, S. Johann, and C. Strangfeld, "Embedded wireless sensor systems for long-term SHM and corrosion detection in concrete components," in *Proc. of International Conference on Structural Health Monitoring of Intelligent Infrastructure-Proceedings*, 2017, pp. 1–7.

[9] R. Rondón, M. Gidlund, and K. Landernäs, "Evaluating Bluetooth Low Energy suitability for time-critical industrial IOT applications," *International Journal of Wireless Information Networks*, vol. 24, no. 3, pp. 278–290, 2017.

[10] E. Park, H.-S. Kim, and S. Bahk, "BLEX: Flexible Multi-Connection Scheduling for Bluetooth Low Energy," in *Proc. of ACM IPSN*, 2021, pp. 268–282.

[11] M. Spörk, C. A. Boano, and K. Römer, "Improving the Timeliness of Bluetooth Low Energy in Noisy RF Environments." in *Proc. of EWSN*, 2019, pp. 23–34.

[12] Spörk, Michael and Boano, Carlo Alberto and Römer, Kay, "Improving the timeliness of Bluetooth Low Energy in dynamic RF environments," *ACM Transactions on Internet of Things*, vol. 1, no. 2, pp. 1–32, 2020.

[13] M. Spörk, M. Schuß, C. A. Boano, and K. Römer, "Ensuring End-to-End Dependability Requirements in Cloud-based Bluetooth Low Energy Applications." in *Proc. of EWSN*, 2021, pp. 55–66.

[14] "nRF52840 Objective Product Specification v0.5," https://infocenter.nordicsemi.com/pdf/nRF52840_OPS_v0.5.pdf, 2016.

[15] "Apache NimBLE," https://github.com/apache/mynewt-nimble, 2022.

[16] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele, "ptunes: Runtime parameter adaptation for low-power mac protocols," in *Proc. of ACM IPSN*, 2012, pp. 173–184.

[17] W. Gao, Z. Zhao, and G. Min, "AdapLoRa: Resource adaptation for maximizing network lifetime in LoRa networks," in *Proc. of IEEE ICNP*, 2020, pp. 1–11.

[18] S. Kim, H.-S. Kim, and C. Kim, "ALICE: Autonomous link-based cell scheduling for TSCH," in *Proc. of ACM IPSN*, 2019, pp. 121–132.

[19] S. Kim, H.-S. Kim, and C.-k. Kim, "A3: Adaptive autonomous allocation of TSCH slots," in *Proc. of ACM IPSN*, 2021, pp. 299–314.

[20] O. Tavallaie, J. Taheri, and A. Y. Zomaya, "Throughput maximization in low-power iot networks via tuning the size of the tsch slotframe," in *Proc. of ACM SenSys*, 2021, pp. 401–402.

[21] S. Duquennoy, B. Al Nahas, O. Landsiedel, and T. Watteyne, "Orchestra: Robust mesh networks through autonomously scheduled tsch," in *Proc. of ACM SenSys*, 2015, pp. 337–350.

[22] S. Jeong, H.-S. Kim, J. Paek, and S. Bahk, "OST: On-demand TSCH scheduling with traffic-awareness," in *Proc. of IEEE INFOCOM*, 2020, pp. 69–78.

[23] W. Park, D. Ryoo, C. Joo, and S. Bahk, "BLESS: BLE-aided Swift Wi-Fi Scanning in Multi-protocol IoT Networks," in *Proc. of IEEE INFOCOM*, 2021, pp. 1–10.

[24] C. Julien, C. Liu, A. L. Murphy, and G. P. Picco, "Blend: practical continuous neighbor discovery for Bluetooth Low Energy," in *Proc. of ACM IPSN*, 2017, pp. 105–116.

[25] H.-W. Cho and K. G. Shin, "BlueFi: Bluetooth over WiFi," in *Proc. of ACM SIGCOMM*, 2021, pp. 475–487.

[26] R. Ayyalasomayajula, D. Vasisht, and D. Bharadia, "BLoc: CSI-based accurate localization for BLE tags," in *Proc. of ACM CoNEXT*, 2018, pp. 126–138.

[27] R. Liu, Z. Yin, W. Jiang, and T. He, "WiBeacon: Expanding BLE location-based services via WiFi," in *Proc. of ACM MobiCom*, 2021, pp. 83–96.

[28] M. Spörk, C. A. Boano, M. Zimmerling, and K. Römer, "Bleach: Exploiting the full potential of ipv6 over ble in constrained embedded iot devices," in *Proc. of ACM SenSys*, 2017, pp. 1–14.

[29] H. Petersen, T. C. Schmidt, and M. Wählisch, "Mind the Gap: Multi-hop IPv6 over BLE in the IoT," in *Proc. of ACM CoNEXT*, 2021, pp. 382–396.

[30] P. Kindt, D. Yunge, M. Gopp, and S. Chakraborty, "Adaptive online power-management for Bluetooth Low Energy," in *Proc. of IEEE INFOCOM*, 2015, pp. 2695–2703.

[31] 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. of IEEE SECON*, 2017, pp. 1–9.

[32] J.-H. Chen, Y.-S. Chen, and Y.-L. Jiang, "Energy-efficient scheduling for multiple latency-sensitive Bluetooth Low Energy nodes," *IEEE Sensors Journal*, vol. 18, no. 2, pp. 849–859, 2017.

[33] F. J. Dian and R. Vahidnia, "Formulation of BLE throughput based on node and link parameters," *Canadian Journal of Electrical and Computer Engineering*, vol. 43, no. 4, pp. 261–272, 2020.

[34] "RIOT OS: The friendly Operating System for the Internet of Things," https://www.riot-os.org/, 2022.