BLEX: Flexible Multi-Connection Scheduling for Bluetooth Low Energy

Eunjeong Park
ejpark@netlab.snu.ac.kr
Department of ECE and INMC
Seoul National University
Seoul, Republic of Korea

Hyung-Sin Kim
hyungkim@snu.ac.kr
Graduate School of Data Science
Seoul National University
Seoul, Republic of Korea

Saewoong Bahk
sbahk@snu.ac.kr
Department of ECE and INMC
Seoul National University
Seoul, Republic of Korea

ABSTRACT

This work investigates a run-time scheduling system for Bluetooth Low Energy (BLE). The Bluetooth specifications propose control of the anchor point (i.e., start time of each connection event) by changing the *WinOffset* parameter of the control protocol data unit (PDU), which can be very useful when the master schedules multiple connections simultaneously. Since we had no access to the controller (i.e., which wraps the physical layer and link layer of Bluetooth), we could not figure out how *WinOffset* works or use it the way we wanted. However, with Zephyr's advent, there is room to modify the controller of BLE, and we can adjust anchor points as we want using *WinOffset*. To the best of our knowledge, this work is the first systematic study of this regime in the research community. We experimentally study how commercial devices actually schedule multiple connections and how this inefficient scheduling degrades BLE's performance. Based on the preliminary study, we propose *BLEX* that dynamically adjusts anchor points of multiple slaves to satisfy quality of service (QoS) requirements without violating the Bluetooth specifications or intervening the BLE host (i.e., software-defined higher layer). Through extensive performance evaluation on off-the-shelf BLE chips, we show that *BLEX* provides stable performance regardless of the traffic requirements of applications and shows significantly reduced QoS failures compared to the state-of-the-art scheduling schemes for BLE.

IPSN' 21, May 18–21, 2021, Nashville, TN, USA

ACM ISBN 978-1-4503-8098-0/21/05…$15.00
https://doi.org/10.1145/3412382.3458271

CCS CONCEPTS

• **Networks → Link-layer protocols**; **Network protocol design**.

KEYWORDS

Internet of Things, Bluetooth Low Energy, resource scheduling

ACM Reference Format:
Eunjeong Park, Hyung-Sin Kim, and Saewoong Bahk. 2021. *BLEX*: Flexible Multi-Connection Scheduling for Bluetooth Low Energy. In *Information Processing in Sensor Networks (IPSN' 21), May 18–21, 2021, Nashville, TN, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3412382.3458271

1 INTRODUCTION

Bluetooth is a representative of low-power wireless technology that is deeply embedded in our daily life today. The Bluetooth special interest group (SIG), the Bluetooth standards organization, expects that 1.14 billion Bluetooth devices will be shipped each year, and 2.1 billion units of Bluetooth-equipped phones, tablets, and PCs will be shipped each year by 2024 [2]. In addition, with the growth of the Internet of Things (IoT), it has been more common to use *multiple* Bluetooth devices simultaneously. For example, anyone using a tablet PC can type with a Bluetooth keyboard and click with a Bluetooth mouse (or Bluetooth trackpad) while listening to music with Bluetooth earphones. People also use smartphones as a Bluetooth hub that connects to multiple Bluetooth peripherals, such as smartwatches and earphones.

Bluetooth technology has significantly evolved over the past two decades. Specifically, Bluetooth specification 4.0, defined in 2010, includes Bluetooth Low Energy (BLE) that enhances low-power characteristics while keeping most advantages of the classic Bluetooth except high throughput. Bluetooth SIG initially designed BLE for low-traffic applications, but has evolved gradually to handle higher traffic. Bluetooth specification 4.2 extended the BLE's packet length to 251 bytes in 2014, and Bluetooth specification 5 increased its data rate up to 2 Mbps in 2016. As BLE's bandwidth improved a lot, its applications have been diversified, including

those that generate *high traffic*. For example, hearing aids included in Android and iOS use BLE [1, 5, 9]. The Bluetooth SIG has recently proposed a BLE-based audio profile [7], which used to be supported by only the classic Bluetooth. A recent study in [21] proposed BLE-based voice delivery.

It becomes more common to use BLE for both low and high traffic applications and to use multiple BLE connections with different applications simultaneously. So it is worth studying systematically whether BLE can efficiently handle multiple high (or low) traffic applications at the link-layer level, which was hided in commercial chipsets. The latest Zephyr [11], an open-source operating system maintained by multiple vendors such as Nordic, Google, and Intel, opens BLE's controller (link-layer) implementation. Therefore it is viable for the research community to investigate BLE's link-layer protocol. Specifically, given that BLE does not include a resource scheduling technique, which is necessary to maintain multiple connections, in its specification, our work investigates *multi-connection resource scheduling* of BLE. To the best of our knowledge, it is the first systematic approach to this topic.

In the Motivation section, we first perform a preliminary study using commercial BLE chipsets with vendor-specific scheduling techniques (closed implementation). Our results show that existing scheduling technology fails to provide reliable service for multiple connections even when communication resources are sufficient. In the worst case, much more communication resources are allocated for low-traffic applications than high-traffic applications. It turns out that none of these techniques consider the actual amount of resources to meet the QoS of each application. Moreover, existing schedulers do not update resource allocation unless a connection is added or removed; once an application has improper resource allocation, it will experience QoS degradation until the end.

To address the issues, it is advisable to reserve resources with useful information related to traffic load and QoS requirements. We need to design a scheduling method that is flexible and reallocates resources for each connection as needed. To enable the scheduler to operate on real BLE chipsets without violating the BLE specification, we need to answer the following four questions.

(1) How to provide application-level QoS information for the controller: The controller (link layer) does not have QoS-related information such as traffic load and latency requirements, which makes proper resource scheduling difficult. To resolve this, the controller need to define and use additional control messages to know the traffic load of applications. Otherwise, the controller must infer the application's traffic load through a new method that has not been presented so far.

(2) How to schedule resources when establishing a new connection: With the QoS requirements of multiple connections, the scheduler should determine a different connection event length for each connection and how often each connection can use its time resource. Specifically in BLE, this ends up with determining the two parameters for each connection: *connection interval* and *anchor point*. This should be done whenever a new connection is established.

(3) How to re-schedule resources: Even if the scheduling scheme establishes a new connection considering its application-level QoS, the required QoS may change over time, or the initial resource allocation may not be optimal. If this is the case, the scheduling scheme should adjust the timing for each connection (i.e., anchor point) to provide just-enough resources to meet its QoS requirements.

(4) When to perform resource scheduling: Each BLE device (e.g., Nordic nrf52840 [8] used in this work) has low computational power, but deals with time-sensitive link-layer tasks. To avoid disturbing other essential BLE operations, the scheduling algorithm should be simple and choose carefully when to run the scheduling.

To answer the questions above, we design *BLEX*, an adaptive resource scheduling system for BLE to meet QoS requirenmts of multiple connections with various applications. For resource scheduling, *BLEX* firstly figures out the amount of traffic for the application through the length of the connection event. Then, *BLEX* adjusts the anchor point of each connection when establishing a new connection. *BLEX* calculates the connection event lengths for ongoing connections and places the new connection's anchor point in an idle time.

Given that only idle time is used for a new connection, *BLEX* does not degrade the performance of existing connections. On the periodic update, *BLEX* squeezes each connection's idle time to maximize the idle time available to future applications. Lastly, *BLEX* performs calculations during the idle time to avoid disrupting existing time-sensitive operations. We implement *BLEX* on Zephyr and extensively evaluate its performance by using Nordic nrf52840. Our results show that *BLEX* performs significantly better than the conventional schemes in various scenarios.

The contributions of this work are threefold.

- Through the preliminary experiments, we observe how existing commercial BLE chipsets perform scheduling, and reveal that *de facto* systems cannot meet various QoS requirements of multiple connections.
- We propose *BLEX*, a scheduling system, that aims to guarantee QoS of multiple connections through anchor point adjustment without additional overhead. The proposed system does not violate the Bluetooth specifications and does not require modifications on the host part, so *BLEX* can be used for any application.

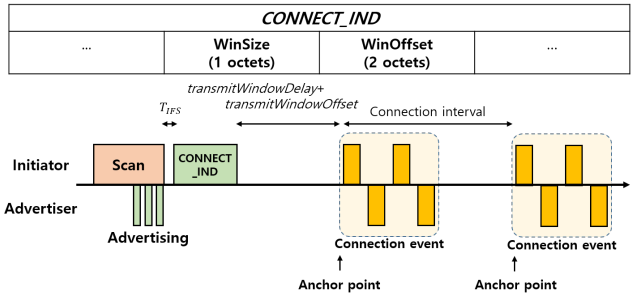


Figure 1: Connection establishment procedures and connection events of BLE link-layer.

- We implement the proposed system prototype, open its source code[1], and evaluate it extensively on a testbed, showing that the proposed system outperforms state-of-the-art scheduling systems.

This paper is organized as follows. We discuss the background and motivation in Section 2 and Section 3, respectively. Section 4 designs the proposed system and we evaluate it in Section 5. We review the related work in Section 6, provide the limitations and future work in Section 7, and then Section 8 concludes the paper.

2 BACKGROUND: BLE

BLE supports two types of communication channels: advertising channels for connection-less broadcasting and data channels for connection-based unicasting. Given that this work focuses on resource scheduling for multiple *connections*, this section describes how a pair of BLE devices establish a connection and exchange data through the established data channel.

2.1 Connection Establishment

Figure 1 illustrates how a pair of BLE devices, an initiator and an advertiser, establish a connection by using three advertising channels (i.e., channels 37, 38, and 39). To establish a connection, the initiator scans through the three advertising channels while the advertiser transmits advertising packets over these channels. When the initiator receives an advertising packet, it sends the advertiser a `CONNECT_IND` protocol data unit (PDU) after T_{IFS} (150 µs).

Once the connection is established, the two devices start communication on the 37 data channels, and the initiator and advertiser are referred to as the master and slave, respectively. The master and slave wake up every *connection interval* simultaneously and exchange packets. The periodic events of packet exchange are called *connection events* and the absolute time at which each connection event starts is called the *anchor point*. Therefore the connection interval is the time interval between two consecutive anchor points. The master starts each connection event by sending a packet at the corresponding anchor point.

[1]https://github.com/ejparkNETLAB/BLEX_Master, https://github.com/ejparkNETLAB/BLEX_Slave

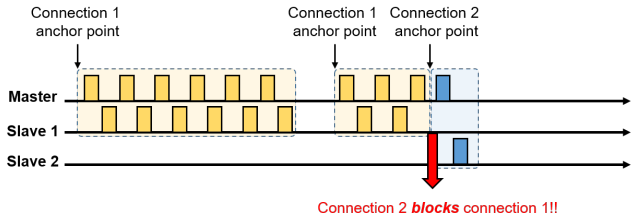


Figure 2: Connection event blocking. The connection event of slave 2 blocks that of slave 1.

The `CONNECT_IND` PDU includes parameters for maintaining a connection, such as *WinOffset* and *WinSize* that affect the connection's anchor point. Specifically, if the `CONNECT_IND` PDU transmission ends at time t_{ind}, the master sets the connection's first anchor point between the two absolute times below:

$$t_{ind} + transmitWindowDelay + transmitWinOffset, \quad (1)$$

$$t_{ind} + transmitWindowDelay + transmitWinOffset + transmitWinSize \quad (2)$$

where

$$transmitWinOffset = WinOffset \times 1.25msec, \quad (3)$$

$$transmitWinSize = WinSize \times 1.25msec \quad (4)$$

and $transmitWindowDelay$ is a constant (1.25 msec). Then each anchor point is at a connection interval after its previous anchor point. Setting *WinSize* to zero allows the new anchor point to be set to a specific value.

2.2 Basics for BLE Resource Scheduling

There are two connection parameters that have a significant impact on the QoS of application: (1) connection interval and (2) anchor point. The connection interval determines how often a connection has connection events (i.e., chance to exchange packets), which directly affects *latency* performance. The anchor point is related to the connection event length, which determines *the amount of traffic* a connection can transmit during a connection event.

Figure 2 shows how anchor points affect the connection event length when the master has multiple connections. Since there must be only one active connection event at a time, BLE forces the current connection event to close when another connection event is started. In Figure 2, for example, slave 1's connection event is terminated even during packet transmission since slave 2's connection event is started. The authors in [15] use the term *blocking* to describe

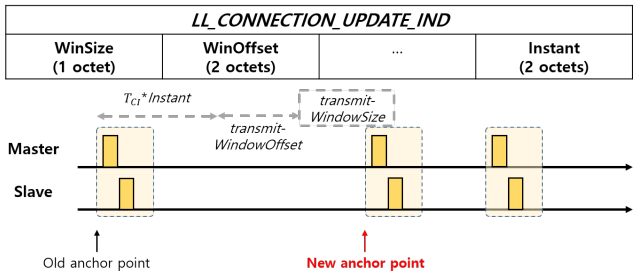


Figure 3: Anchor point update procedures.

this phenomenon that the rear connection *blocks* the previous connection. Due to the connection blocking, when the master handles multiple connections, each connection's anchor point affects each other's event length.

Then BLE resource scheduling ends up with determining the connection interval and anchor point of each connection properly. The BLE specifications let the master's host (upper layer) set the range of the connection interval (minimum and maximum values), and let the master's controller (link layer) select a specific value within the given range. To this end, BLE provides a host-controller interface (HCI) command, called `LE Connection Update`, that is sent from the host to the controller to update the connection interval.

In contrast, BLE enables only the controller to set a connection's anchor point; there is no HCI command that can set/access *WinOffset* and *WinSize*. Figure 3 depicts how a connection's anchor point is updated. To change an existing connection's anchor point, the controller uses not only *WinSize* and *WinOffset* but also *Instant*. The new anchor point is not applied immediately. *Instant* indicates how many connection intervals are needed from the current anchor point until a new anchor point is applied. Therefore, if the update is triggered at a connection event whose anchor point is t_{old} and its connection interval is T_{CI}, the first anchor point with the new parameters is between the two absolute times below:

$$t_{\text{old}} + Instant \times T_{\text{CI}} + transmitWinOffset, \quad (5)$$

$$t_{\text{old}} + Instant \times T_{\text{CI}} + transmitWinOffset + transmitWinSize. \quad (6)$$

By setting *WinSize* to zero, we can set the new anchor point to a specific value.

The host (upper layer) determines each connection's connection interval according to the *latency* requirement of an application. The anchor point, however, is only controlled by the controller (link layer) that does not have any application-layer information. Therefore the resource scheduling scheme should consider how to provide the controller with information about the application's *traffic load*.

2.3 Zephyr Operating System

Lastly, it is essential to mention Zephyr [11], a real-time and open-source operating system, that supports various protocols, such as BLE, Wi-Fi, IEEE 802.15.4, 6LoWPAN, CoAP, IPv4, IPv6, Ethernet, and Thread. Zephyr is maintained by multiple vendors and has become more widely used in the IoT domain. Specifically, it has a significant impact on BLE research because it opens the BLE's controller code that used to be completely hidden. Although Bluetooth has been around for two decades, we believe that there are more research opportunities than ever before since the delivery of a software-defined open controller. There is room to investigate various aspects of the BLE link layer, such as resource scheduling by adjusting anchor points.

3 MOTIVATION

This section performs a preliminary study on existing scheduling systems, examining if these systems can support multiple connections simultaneously. Based on the results, we present a set of design requirements for a resource scheduling system to handle multiple connections efficiently.

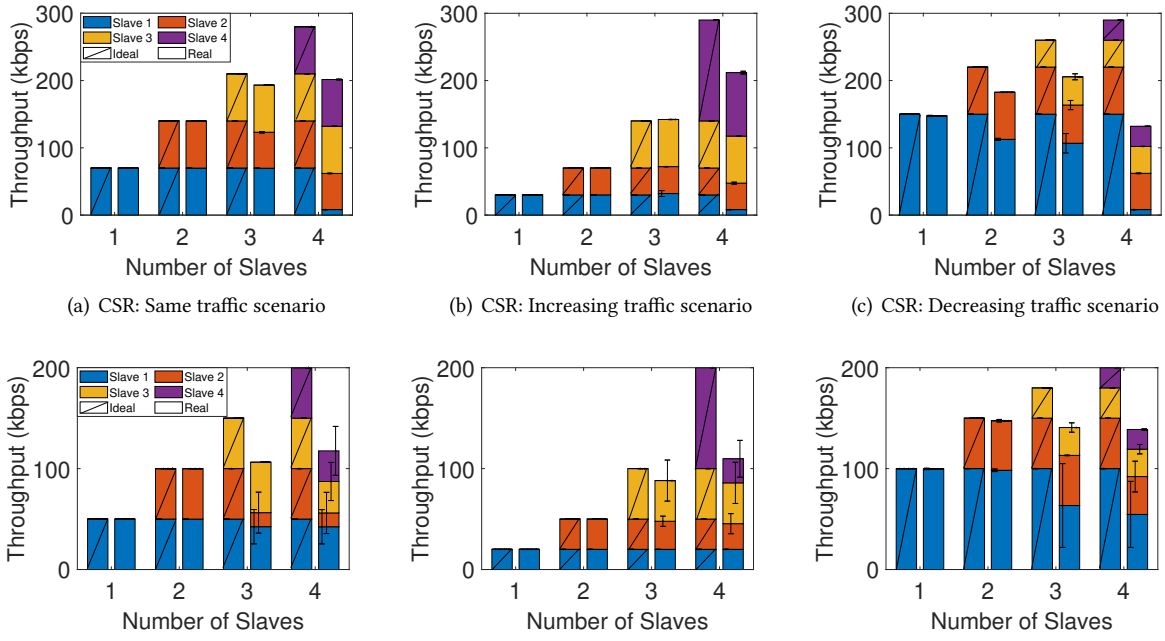
3.1 Preliminary Study

We evaluate two representative commercial BLE chipsets: CSR 4.0 dongle [4] and BCM4356 chipset [3]. CSR4.0 dongle is often used to connect Bluetooth earphones, keyboard, and mouse to PC, and BCM4356 is widely used in laptops, smartphones, and tablet PCs. To maintain a single connection, CSR 4.0 dongle and BCM4356 provide (maximum) throughput of 285 kbps and 185 kbps, respectively.

To see how two devices work when there are multiple connections, we connect up to four slaves to the master with the same connection interval of 40 ms, where slaves 1, 2, 3, and 4 are connected to the master sequentially. We make each slave generate traffic to the master in three different scenarios: (1) same-traffic, (2) increasing-traffic, and (3) decreasing-traffic. In the same-traffic scenario, each slave generates 1/4 of the maximum throughput. In the increasing-traffic scenario, slaves 1, 2, 3, and 4 generate 1/8, 1/8, 1/4, and 1/2 of the maximum throughput, respectively. In the decreasing-traffic scenario, slaves 1, 2, 3, and 4 generate 1/2, 1/4, 1/8, and 1/8 of the maximum throughput, respectively.

Figure 4 shows the results according to the number of slaves. Each case has two stacked bars: one on the left represents the ideal throughput and the other on the right represents the actual throughput. A coarse analysis of the results shows that both CSR dongle 4.0 and BCM4356 experienced substantial performance degradation with the number of connected slaves: throughput loss from 27% to 62%.

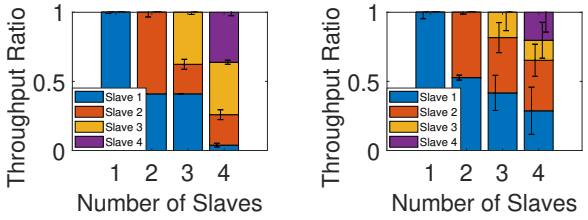
Given that the resource scheduling systems of the two devices are hidden, it is necessary to do reverse engineering



(a) CSR: Same traffic scenario (b) CSR: Increasing traffic scenario (c) CSR: Decreasing traffic scenario

(d) BCM: Same traffic scenario (e) BCM: Increasing traffic scenario (f) BCM: Decreasing traffic scenario

Figure 4: Stacked throughput bar graphs of the CSR 4.0 dongle and BCM4356 chipset when the number of slaves increases from 1 to 4. Figs. 4(a) and 4(d) show when each slave's traffic is the same, Figs. 4(b) and 4(e) show when the slave's traffic is gradually increasing, and Figs. 4(c) and 4(f) show when the slave's traffic is gradually decreasing. The left bars are for ideal scheduling and the right bars are for actual measurements.



(a) CSR dongle (b) BCM4356 chipset

Figure 5: The ratio of each slave's throughput when the number of slaves increases from one to four in the saturated-traffic scenario.

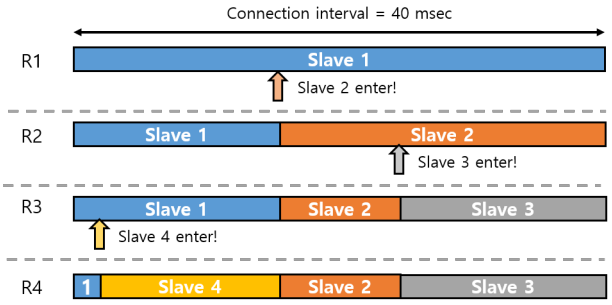


Figure 6: Resource scheduling (anchor point placement) of CSR 4.0 dongle when the number of slaves increases from one to four

to analyze the results of Figure 4 more deeply. To this end, we conduct another experiment under saturated traffic. We sequentially connect four slaves to the master and ensure all of them to generate as much traffic as possible. Figure 5 shows the relative throughput of the slaves according to the number of slaves.

In the case of CSR 4.0 dongle, as shown in Fig. 5(a), we observe that whenever slave 2 newly joins after slave 1, slave 2 takes 60% of slave 1's resources. Given that all the slaves use the same connection interval, the result implies that slave 2's anchor point may be set as (*slave 1's anchor point + 0.4*connection interval*), making a connection event of slave 2 block that of slave 1. When slave 3 joins, it takes 60% of slave 2's resources by blocking its connection event (i.e., slave 1's resources are fully preserved this time), dividing the throughput of the three slaves by 2:1:2 each. Finally, when slave 4 joins, it blocks slave 1 instead of slaves 2 and 3, reducing slave 1's throughput drastically. We can conjecture that slave 4's anchor point is located shortly after slave 1's anchor point. Slave 2, which previously blocked slave 1, now blocks slave 4. Given that the results are reproduced again and again, we conclude that CSR 4.0 dongle has a fixed scheduling strategy, which is depicted in Figure 6.

With the findings above, let us come back to Figures 4(a)-(c), the case of CSR 4.0 dongle. In the same-traffic scenario

(Figure 4(a)), the master receives traffic from slave 1 to slave 3 fairly well. This is because slaves 1 to 3 take 40%, 20%, and 40% of the total resources, respectively, which is similar to or greater than each slave's traffic (i.e., 25% of the maximum throughput). When slave 4 joins, however, it significantly reduces the throughput of slave 1 by blocking. In the increasing-traffic scenario (Figure 4(b)), slaves 2 and 3 deliver their traffic well, but again, slave 1's throughput is reduced as slave 4 joins. In this case, slave 4 runs out of resources since it is blocked by slave 2.

In the decreasing traffic scenario (Figure 4(c)), due to the same reason, slave 1's throughput is first reduced when slave 2 joins and then is more significantly degraded when slave 4 joins. Overall, the reason why CSR 4.0 dongle causes the throughput degradation in the presence of multiple connections is not due to the lack of overall resources but its fixed scheduling strategy that runs regardless of the traffic demand of each connection.

BCM4356 shows different characteristics. As shown in Figure 5(b), if slave 2 enters when there is only slave 1, each of slaves 1 and 2 takes half of the whole resources. However, when slaves 3 and 4 enter, we can see that the throughput ratio of each slave significantly varies (long error bars). This implies that the anchor points of slaves 3 and 4 may be determined randomly. This phenomenon is reproduced in Figures 4(d)-(f). In the three traffic scenarios of BCM5436, it can be seen that slaves 1 and 2 achieve the desired throughput when the number of slaves is two since each of them occupies 50% of the whole resources. However, as slaves 3 and 4 join, overall throughput is significantly degraded, and each slave's throughput significantly varies. It is confirmed that the random scheduling strategy of BCM5436 does not support multiple connections either because it does not consider each connection's traffic demand.

3.2 Design Requirements

The results of the previous section give five requirements for a BLE resource scheduling system to reliably support multiple connections simultaneously as follows:

Protection for existing QoS: Both BCM4356 and CSR 4.0 dongle degrade existing slaves' throughput when a new slave joins, which is not desirable in practice. To avoid this, the master should be able to monitor which slaves do not fully utilize their resources (i.e., have idle resources). When the idle resources are not enough to fully satisfy the new slave, it is better for the new slave to be left unconnected or unsatisfactory than to bother existing slaves.

Guaranteeing new QoS: When establishing a new connection, neither BCM4356 nor CSR 4.0 dongle considers its traffic demand for resource allocation, degrading performance even when there are idle resources. More fundamentally, the master's controller does not know QoS requirements of a new connection in advance. To solve this problem, the master should allocate enough resources for the slave.

Avoiding spread of idle resources: Both BCM4356 and CSR 4.0 dongle spread idle resources over multiple connections. In Figure 4, for example, slaves 2 and 3 have resources more than needed. If idle resources are spread across many connections, it is challenging to allocate enough resources for the new slave without degrading the QoS of existing connections. Instead, a single connection should have idle resources as much as possible while other connections squeeze their resources, just enough to satisfy their QoS.

Adapting to changing QoS: In our experience, both the two devices adjust resource distribution only when establishing a new connection. However, even if a scheduling system satisfies all the requirements above, there are still limitations if it is static. For example, if an application such as FTP changes traffic over time, its resources, which were initially sufficient, can sometimes be insufficient or wasted. To address the problem, a resource scheduling system should adjust resource allocation according to time-varying traffic demands.

Computational Complexity: Given that BLE is a time-sensitive and low-power communication protocol, a scheduling scheme should have low computational complexity not to interfere with other essential operations. This aspect partially explains why both BCM4356 and CSR 4.0 dongle perform primitive and simple scheduling.

4 *BLEX*: FLEXIBLE SCHEDULING FOR BLE

This section presents *BLEX*, a novel resource scheduling system for BLE, that aims to satisfy QoS of multiple connections efficiently. We first give an overview of the whole *BLEX* system and describe the details of its core modules. Given that setting the connection interval for each connection is trivial, easily done at the host by considering an application's latency requirement [32], *BLEX* focuses on a more challenging task: scheduling anchor points for multiple connections at the controller.

4.1 System Overview

Figure 7 summarizes the overall architecture of our *BLEX* system. *BLEX* has four core modules as given below.

- **Connection event length calculator** estimates each connection's application traffic load at the controller without any explicit control message. This module enables *BLEX* to monitor whether each connection's resources are sufficient to accommodate its traffic load.

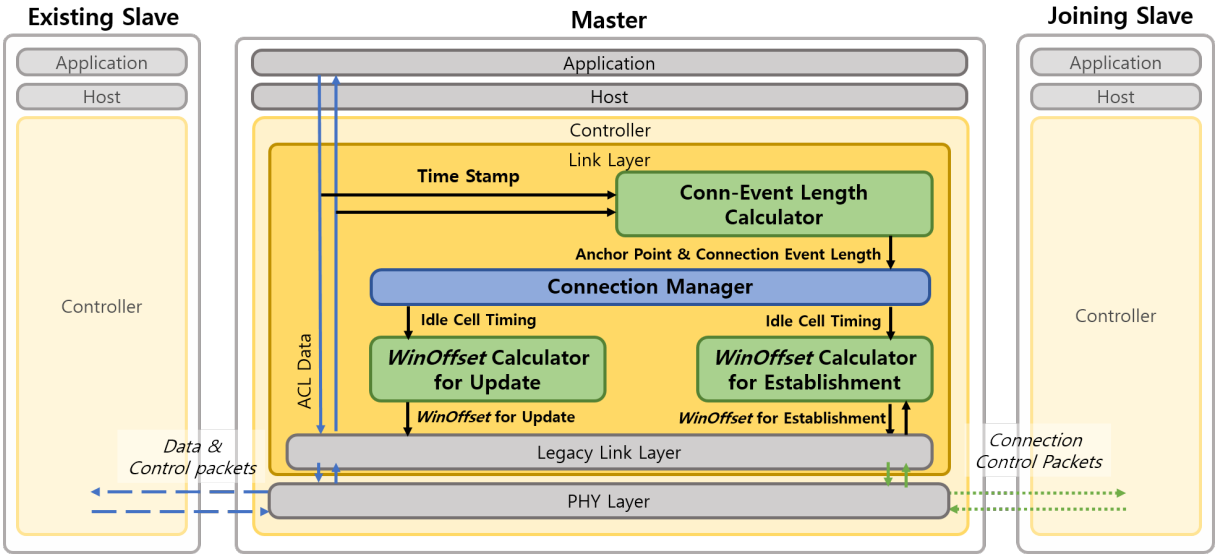


Figure 7: Proposed system architecture.

- **Connection manager** constructs a timetable that holds each connection's event length, connection interval, and anchor point. This module tries to keep the timetable size small to reduce computational complexity.
- ***WinOffset* calculator for connection update** periodically updates *WinOffset* for each existing connection to adjust resource allocation. Its goal is for each connection to have just-enough resources to cope with its traffic and a single connection to have almost all idle resources.
- ***WinOffset* calculator for connection establishment** provides *WinOffset* for a newly joined slave. This module aims to provide an anchor point for the new slave so that it has as much resource as possible without invading the QoS of the existing slaves.

BLEX is a pure link-layer mechanism, fully implemented on Zephyr's controller code. By decoupling its whole system from the host part, *BLEX* can impact on various BLE chipsets. In addition, an application can expect performance improvement without changing its profile and the host operation (e.g., using additional HCI commands).

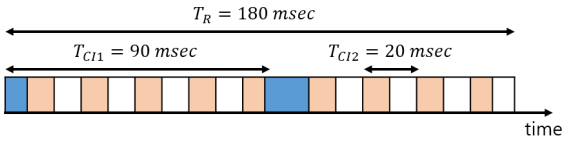
4.2 Connection Event Length Calculator

A challenge to *BLEX* is that the controller does not know the application traffic load. There are a couple of ways to resolve this issue. First, it is possible to define an additional HCI command for the host to give the controller traffic load information. However, this causes an extra overhead and restricts the mechanism's applicability since the host part should be modified together. Second, the controller can check its transmission (TX) queue length. However, this method can be only applied to senders. If a slave sends traffic to the master, the master cannot get the traffic load information since it does not know the slave's TX queue length. To this end, the slave should send the master a new control PDU, including its TX queue length, which is an additional overhead.

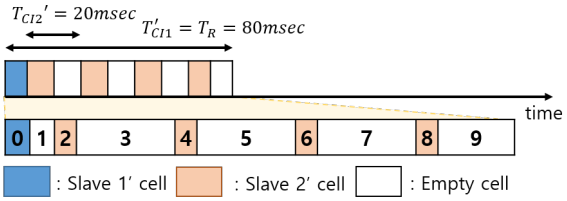
Instead, we find out that a connection's event length can be used to indicate its traffic load. Although the connection has abundant resources, its connection event ends earlier if there is no traffic to deliver; connection event length is proportional to traffic demand. Therefore we measure each connection's event length using the time difference between its start (i.e., anchor point) and end.

However, there are a few things to resolve to use the connection event length for resource scheduling. First, given that a connection's event length varies depending on traffic patterns and link-layer retransmissions, it is important to have a proper value representing the *traffic load*. To this end, we exploit the exponentially weighted moving average (EWMA) filter to average each connection's event lengths. Second, it is essential to note that BLE terminates a connection event early when two consecutive packets cause CRC error at the receiver, even when many packets are waiting to be sent. In this case, the connection event length is short even when the traffic load is high. To alleviate this problem, we ignore the connection event length if the event is terminated due to CRC errors.

Finally, the connection event length calculator sends the measured anchor points and connection event lengths to the connection manager for scheduling.



(a) Repeated schedule unit before conversing connection interval



(b) Repeated schedule unit after conversing connection interval

Figure 8: Example of connection interval conversion when the connection interval of slave 1 is 90 *mesc*, and that of slave 2 is 20 *mesc*.

4.3 Connection Manager

With the information received from the connection event length calculator (i.e., anchor points and connection event lengths), the connection manager constructs a timetable to efficiently manage the resource schedule. To this end, the connection manager should address a challenge: timetable overhead when multiple connections have different connection interval values.

Figure 8(a) illustrates the challenge where two connections have different connection interval values, 90 *mesc* and 20 *mesc*, respectively. In this case, the connection manager should know all the anchor points (and the connection event lengths) at least during a 180 *mesc* interval (lowest common multiple of 90 and 20) to have complete knowledge of the resource schedule: 11 anchor points in total. We refer to the minimum unit that contains the complete resource schedule as the *repeated schedule unit*. The connection manager's timetable should include at least one repeated schedule unit; with the size of the repeated schedule unit, the timetable size of the connection manager and computational overhead increase. Given that the size of the repeated unit is proportional to the lowest common multiple of all the different connection interval values, its overhead becomes problematic, especially when the connection interval values are *mutually prime*.

To resolve the issue, *BLEX* limits the connection interval to $1.25*2^n$ *mesc*. Specifically, when the host gives the range of the connection interval, the controller selects the largest connection interval within the range that can be represented in the form of $1.25*2^n$ *mesc*. Then the lowest common multiple of connection interval values always becomes the longest connection interval, limiting the size of the repeated schedule unit. Figure 8(b) shows the effect of this limitation. slave 1's connection interval becomes 80 *mesc* (=$1.25*2^6$) rather than 90 *mesc* while slave 2's connection interval remains the same (20 = $1.25*2^4$). Since the minimum common multiple of the two is 80 *mesc*, the size of the repeated schedule unit becomes 80 *mesc*, less than half of the previous one (i.e., 180 *mesc*). The number of anchor points to be managed becomes 5, less than half of the previous one (11).

Table 1: Connection management example: Parameters of slave 1 and slave 2

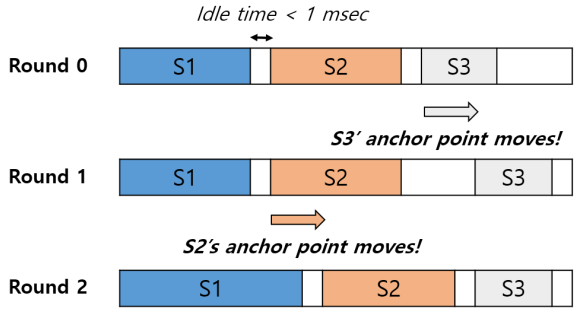
Slave number	Connection interval	Anchor point	Connection event length
Slave 1	90 *mesc*	250 *mesc*	9 *mesc*
Slave 2	20 *mesc*	260 *mesc*	4 *mesc*

Table 2: Connection management example: Timetable

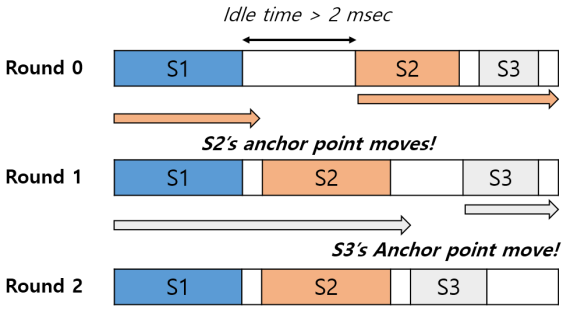
Cell number	Node number	Starting point (anchor point)
0	1	250
1	NULL	259 (=250+9)
2	2	260
3	NULL	264 (=260+4)
4	2	280 (=260+20)
5	NULL	284 (=260+20+4)
6	2	300 (=260+20*2)
7	NULL	304 (=260+20*2+4)
8	2	320 (=260+20*3)
9	NULL	324 (=260+20*3+4)

Next, to construct the timetable, the connection manager divides the repeated schedule unit into cells using each connection's event length, anchor points, and connection interval. The number of cells for each slave is the length of the repeated schedule unit divided by the slave's connection interval. Since the timetable needs to store idle cells between the slaves' active cells, the timetable needs to store up to twice the total number of active cells. Each cell entry in the timetable has its anchor point, and the node number of the slave that uses the cell. If no slave uses the cell (i.e., idle cell), its node number becomes NULL and its anchor point becomes the time when the previous cell ends.

We describe how to construct the timetable by using the example in Figure 8(b). The repeated schedule unit starts from a cell of the firstly joined slave, slave 1 in this case. After that, the connection manager fills in all the entries by using the information it has. If each slave's anchor point is 250 *mesc* and 260 *mesc*, respectively, as shown in Table 1, the timetable includes 10 cells from 250 *mesc* to 330 *mesc*. Table 2 shows the timetable of 10 cells. Since slave 1's connection interval is the same as the repeated schedule unit, it has only one cell, and its anchor point is 250 *mesc*. Slave 2 needs four cells, whose anchor points are 260 *mesc*, 280 *mesc* (i.e., 260 + T_{CI}), 300 *mesc* (i.e., 260 + $2T_{CI}$), and 320 *mesc* (i.e., 260 + $3T_{CI}$), respectively, because slave 2's connection interval is a quarter of the repeated schedule unit. Lastly, there are five idle cells. Each idle cell's anchor point is calculated using its previous cell's anchor point and event length. This method can be applied without losing generality even



(a) Too short idle time



(b) Too long idle time

Figure 9: Connection update procedures.

when the master has more slaves with various connection intervals.

4.4 *WinOffset* for Connection Update

With the timetable, the *WinOffset* calculator for connection update identifies whether the current resource schedule is efficient or not. There are two cases in which the calculator determines that the timetable is inefficient: (1) no (or too short) idle cell between two active cells and (2) too long idle cell between two connection events (i.e., the last idle cell is ignored). In the case of (1), *BLEX* may need more resources but it determines that the rear slave blocks the front slave. In the case of (2), there is room to squeeze the idle cell, which increases the last idle cell for the future connection. Note that it is better to have one large idle cell than many small idle cells for future resource allocation. If the both cases happen, (1) is dealt with first since satisfying QoS is more important than efficiency.

Figure 9 shows how *BLEX* updates the connections of three slaves for the following two cases.

Case of too short idle cell: Figure 9(a) illustrates when the idle cell between the active cells of slave 1 and slave 2 is too short (less than 1 *mesc*, half of 2 *mesc* that is needed to transmit the longest packet with a 1 Mbps PHY rate). In this case, *BLEX* should increase the cell of the front slave (slave 1) to satisfy its QoS. To this end, *BLEX* re-configures the anchor points.

To increase slave 1's active cell size, the idle cell right after the active cell should be increased. Since the master's controller does not know how many resources are required to meet slave 1's QoS, *BLEX* should *maximize* the idle cell size after slave 1's active cell first so that slave 1 can use as many resources as it wants. If too many resources are given to slave 1, case (2) reschedules unnecessary resources.

Maximizing slave 1's resources should be performed carefully since other slaves should not be bothered during this process. To this end, *BLEX* adjusts anchor points sequentially as follows: Since there is no time-space to squeeze the idle cell between slaves 2 and 3, but the last idle cell is long enough, slave 3's anchor point is deferred to reduce the last idle cell. Then the idle cell between slave 2 and slave 3 increases. Next, slave 2's anchor point is deferred to reduce the idle cell, which increases the idle time used by slave 1. As a result, *BLEX* allocates more time resources to slave 1.

Case of too long idle time: Figure 9(b) illustrates this case where the idle cell between the active cells of slave 1 and slave 2 is unnecessarily long (longer than 2 *mesc*). In this case, *BLEX* adjusts anchor points to reduce the idle cell to maximize the last idle cell. Given that *BLEX* allocates the last idle cell for a new connection, it is important to maximize the last idle cell by squeezing other idle cells. Specifically, *BLEX* defers slave 2's anchor point by (*connection interval-the idle space length*) to minimize the idle cell between slaves 1 and 2. Then slave 3's anchor point is deferred to minimize the idle cell between slaves 2 and 3, which maximizes the last idle cell placed after slave 3's active cell. As a result, all idle resources are in the last idle cell.[2]

4.5 *WinOffset* for Connection Establishment

Lastly, *BLEX* allocates resources when a new connection is established. If there is no existing slave connected to the master, a newly joining slave becomes the first slave in the timetable, and its anchor point is not important. However, if there are some existing slaves, the newly joining node should not affect their QoS.

A challenge here is that the master does not know the traffic demand of new slave since it does not have any information about connection event length for it. To address this issue, *BLEX* calculates the time difference between scanning connectable advertising and the anchor point of the largest idle cell, and sends `CONNECT_IND` to the new slave by designating this as *WinOffset*. Then the anchor point of the new slave becomes that of the largest idle cell. In this way, the

[2]Since we can only move the anchor point backward in time, but not forward, we move it backward.

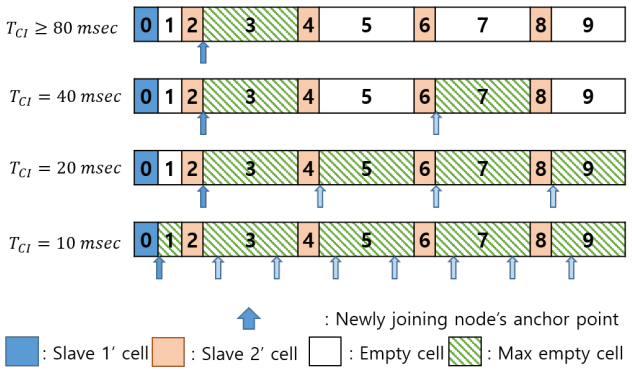


Figure 10: Idle cell's anchor points that can provide the maximum QoS for a new slave according to its connection interval.

new slave can achieve the maximum QoS possible in the first connection event.

As explained in the background, there is only T_{IFS} from when the master scans a connectable advertising packet until the master sends `CONNECT_IND`. However, calculating *WinOffset* requires a long computation time so that it can hinder the transmission of `CONNECT_IND` PDU and, hence, the connection establishment. Therefore, *BLEX* calculates this *WinOffset* in advance so that when a new node tries to join, the master can immediately send this *WinOffset*. However, the problem is that we do not know the connection interval of the slave to be joined in the future, and *BLEX* does not know the number of necessary cells accordingly. Therefore, it is necessary to store the appropriate anchor point for all possible connection interval sets. We can do this very quickly because we have limited the connection interval in the connection manager.

The range of connection intervals defined in the Bluetooth specification ranges from 7.5 *mesc* minimum to 4 sec maximum, so there are only nine values corresponding to $1.25^{*}2^{n}$ *mesc*. Also, if the repeated schedule unit is less than 2560 *mesc*, the number of calculations is further reduced. In the example of Figure 8, if the connection interval is greater than or equal to 80 *mesc*, the number of required cells is only one, so it is possible to calculate at once. Therefore, the number of cases to be calculated in advance is only four: 1) $T_{\text{CI,new}} \geq 80$ *mesc*, 2) $T_{\text{CI,new}} = 40$ *mesc*, 3) $T_{\text{CI,new}} = 20$ *mesc*, and 4) $T_{\text{CI,new}} = 10$ mesc, when $T_{\text{CI,new}}$ is the connection interval of the newly joinging slave.

For each connection interval, we determine an anchor point that (1) satisfies the connection interval, (2) does not overlap with existing connections, and (3) provides the maximum QoS. Figure 10 shows the location of the anchor point to be assigned according to $T_{\text{CI,new}}$. If $T_{\text{CI,new}}$ is greater than or equal to 80 *mesc*, only one cell will be allocated in the current timetable, so *BLEX* sets the start time of the largest idle cell to the new anchor point. When $T_{\text{CI,new}}$ is 40 *mesc*, since the length of the current timetable is 80msec, the new slave needs two cells within this timetable. At this time, considering $T_{\text{CI,new}}$, the set of allocatable cells is {(1,5), (3,7), (5,9)}. According to the previous timetable, the combination (3,7) can provide the most extended time resources to the new slave, so the new anchor points are placed at the start time of cells 3 and 7. Even when $T_{\text{CI,new}}$ is smaller than 40msec, *BLEX* can apply the same method without loss of generality.

5 PERFORMANCE EVALUATION

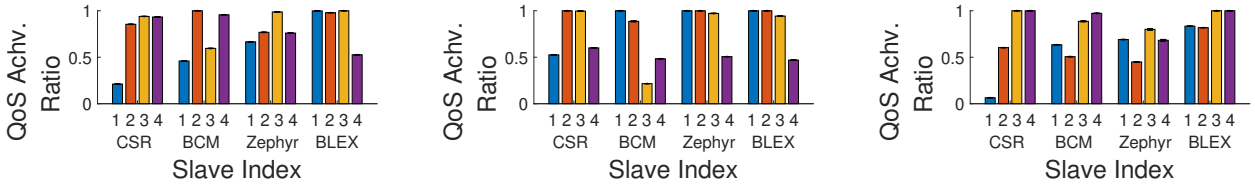
To evaluate *BLEX*'s performance, we implemented it on the Zephyr's controller and build it on Nordic nrf52840 [8]. In the Zephyr's controller, we modified the `ull_master.c`, `ull_conn.c`, and `lll_scan.c` files. Zephyr is compliant with the Bluetooth specification, but the *WinOffset* update parts of the connection establishment and the connection update were not implemented. Accordingly, we modified the `event_conn_upd_prep`, `ctrl_rx`, and `isr_rx_pdu` functions of the master and slave so that devices can perform the *WinOffset* update procedures. To implement *BLEX*, we modified the `ull_llcp_conn` and `ull_llcp_conn_done` functions of `ull_conn.c` in the master side.

In the `ull_llcp_conn` function, we measure the length of connection events and average them. In the `ull_llcp_conn_done` function, we use this connection event length to manage the schedule and calculate the *WinOffset*. When sending `CONNECT_IND` PDU, response to the connectable advertising message received through `isr_rx` in `lll_scan.c`, this *WinOffset* is transmitted to the slave. Also, if the master determines that anchor point update is necessary, it sends *WinOffset* through the modified `llcp_conn_update` and updates the connection. Noridic nrf52840 is a single-board development kit equipped with 64 MHz Arm Cortex-M4 with FPU, 1 MB Flash, and 256 KB RAM, and supports all Bluetooth Low Energy, Bluetooth mesh, Thread [26], and ZigBee. This board has been used in many studies by providing Bluetooth 5 features [12, 13, 33].

For comparison, we implemented the CSR-based and BCM-based schemes, the *de facto* scheduling schemes of CSR 4.0 dongle and BCM4356, on the Zephyr's controller. It was based on the real experiments as in the Motivation section. We also use Zephyr default operation together with CSR-based and BCM-based schemes. Zephyr default operation sets *WinOffset* to 0 and sends `CONNECT_IND` PDU when the master receives a connectable advertising packet. That means the time at which the scan takes place becomes the anchor point. We confirm the effect of *WinOffset* in the connection establishment and connection update when the connection interval is the same (i.e., 40 msec), and then conduct performance evaluation in real multiple application scenarios.

(a) Same traffic scenario (b) Increasing traffic scenario (c) Decreasing traffic scenario

Figure 11: Stacked throughput graphs of *BLEX* and the compared schemes when the number of slaves varies from 1 to 4 in three traffic scenarios.



(a) Same traffic scenario (b) Increasing traffic scenario (c) Decreasing traffic scenario

Figure 12: QoS achievement ratio of each slave in *BLEX* and the comparison schemes when four slaves are connected to one master in three traffic scenarios.

5.1 *WinOffset* in Connection Establishment

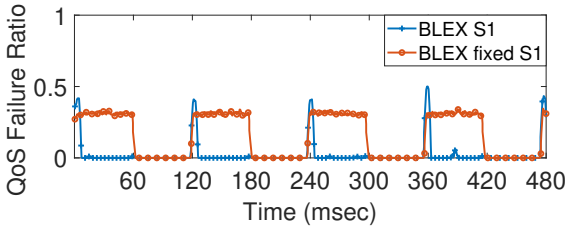
We experimentally study the performance of *BLEX* when multiple connections have the same connection interval. We evaluate each scheme's performance in three simple scenarios to see whether *BLEX* achieves our goal: guaranteeing the maximum QoS of new connection without degrading the QoS of existing connections. The three scenarios cover the case of equal traffic, increasing traffic, and decreasing traffic using four slaves and one master, as in the Motivation section.

In the same traffic scenario, each slave generates 125 kbps toward the master. In the increasing traffic scenario, the master receives 51 kbps from slaves 1 and 2, 175 kbps from slave 3, and 285 kbps from slave 4. In the decreasing traffic scenario, the master receives 285 kbps from slave 1, 175 kbps from slave 2, and 51 kbps from slaves 3 and 4. These values are selected to allow the application to send the traffic with a fixed rate while considering that Nordic nrf52840 running Zephyr has the throughput of up to 600 kbps when one slave occupies all the resources. Figure 11 and 12 show that *BLEX* achieves the highest total throughput while not affecting the QoS of ongoing slaves by changing *WinOffset* when establishing a new connection.

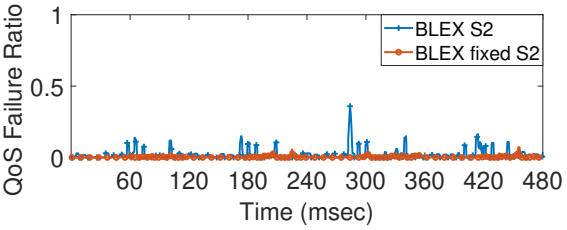
Equal traffic scenario: Figure 11(a) shows the measured throughput, and Figure 12(a) depicts the QoS achievement ratio of each slave in the same traffic scenario. CSR-based and BCM-based schemes cannot protect the QoS of existing slaves, and as a result, when a slave newly joins, these schemes significantly degrade the QoS of the existing slaves. In the case of Zephyr default operation, the time at which the master performs scanning affects the connection's anchor point. Since the scan occurs only during the idle time, it does not significantly affect the QoS of ongoing slaves compared to *de facto* schemes such as BCM-based and CSR-based schemes.

Compared to this, through the connection event length calculator, *BLEX* guarantees as much time as each slave needs (i.e., the averaged connection event length), so even if a slave newly joins, its impact on existing slaves is insignificant. *BLEX* also improves the total throughput by 19%, 16%, and 10% compared to the CSR-based and BCM-based schemes and Zephyr default operation, respectively. This is because *BLEX* allocates time to the ongoing slaves first, then squeezes the remaining time and allocates it to the new slave.

Increasing traffic scenario: When the amount of traffic from existing slaves is low, the length of the connection events they occupy is short, so the probability of being interrupted is relatively small. In the CSR-based scheme, the total throughput is slightly higher than that in the proposed scheme, but when slave 4 enters, it takes up most of slave 1's time, so the QoS of existing slaves is not guaranteed. In the BCM-based scheme and Zephyr default operation, we see that the total throughput is low while not satisfying the QoS of existing slaves. In comparison, *BLEX* guarantees the QoS of ongoing transmissions while allocating the maximum possible time to the new slave, i.e., scheduling runs ideally.



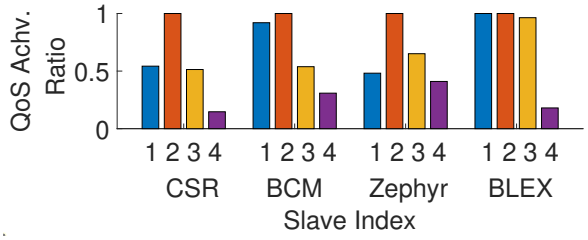
(a) Slave 1' QoS failure



(b) Slave 2' QoS failure

Figure 13: QoS failure ratio changes when the traffic rate for each slave changes in *BLEX* and *BLEX*-fixed.

(a) Achieved throughput



(b) QoS achievement ratio

Figure 14: Throughput and QoS achievement ratio in a practical application scenario.

Decreasing traffic scenario: In this scenario, *BLEX* shows the greatest performance improvement among the three scenarios. The reason is that ongoing slaves are already taking up much time. Accordingly, in the comparison schemes, the probability that the anchor point of the new slave overlaps the connection event of existing slaves is high; thereby, the new slave infringes the QoS of ongoing slaves. In particular, in the CSR-based scheme, since joining of slave 4 with low traffic occupies most of the time of slave 1 with heavy traffic, the total throughput is significantly lowered. Compared to the other schemes, *BLEX* allocates the maximum time to the new slave while guaranteeing the QoS of existing ongoing connections. *BLEX* improves total throughput performance by 113%, 33%, and 38% compared to CSR-based and BCM-based schemes and Zephyr default operation, respectively.

5.2 *WinOffset* in Connection Update

To show the effect of *WinOffset* on the connection update, we connect two slaves sequentially to the master (until slave 2 establishes a connection, slave 1 does not generate traffic) and measure the QoS failure ratio by changing the amount of traffic to the slaves every 60 seconds. For the first 60 seconds, the master sends 220 kbps to slave 1 and 97 kbps to slave 2. Then the master sends 220 kbps to slave 2 and 97 kbps to slave 1 for the next 60 seconds. Traffic changes every 60 seconds.

Figure 13 compares the performance of *BLEX* with and without connection update. In *BLEX* without connection update, *BLEX*-fixed, when the first anchor point is determined, no matter how much traffic changes, the anchor point no longer changes. Therefore, even if there are wasted resources as traffic changes, they are not allocated to the slave that needs more resources. In this case, slave 1 with initially less allocated time experiences periodic QoS failures. However, *BLEX* with connection update determines whether the schedule is inefficient periodically and adjusts the anchor point accordingly. *BLEX* adaptively allocates more time to slaves with heavy traffic, and therefore, it always shows a lower QoS failure ratio than *BLEX*-fixed.

5.3 Multiple QoS Requirements

Until now, we conduct experiments in a straightforward but not practical scenario in which all the slaves use the same connection interval to show the performance of anchor point adjustment. However, when using BLE, each application generates a different amount of traffic and requires different latency. This subsection conducts experiments for the scenario when the throughput and latency requirements are different for each application. In CSR-based and BCM-based schemes, since the connection intervals are different, unlike in the preliminary experiment, additional reverse engineering was performed and reflected in the implementation.

We connect slaves 1 to 4 to the master in order and assume that slaves 1 and 3 use audio streaming applications, slave 2 uses a keyboard application, and slave 4 uses an FTP application such as connecting to a photo printer. According to

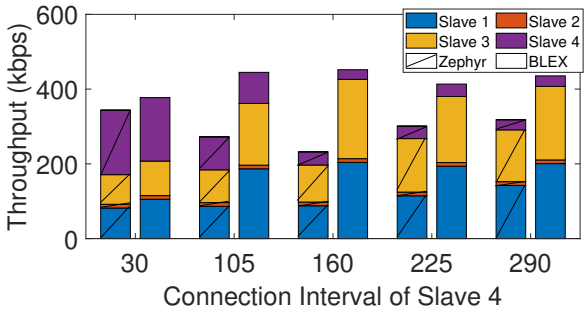


Figure 15: Throughput of *BLEX* in a practical application scenario compared to that of Zephyr according to the connection interval of slave 4.

[6, 10], the throughput required by an audio streaming application is about 192 kbps, and the throughput required by the keyboard application is lower than 10 kbps. FTP application is assumed to need more than 150 kbps, although it depends on the file size. Also, refer to [16] for the latency requirement; the audio streaming application requires the latency lower than 100 msec, the keyboard application requires 50 msec, and FTP application requires 200 msec because latency is not essential for FTP application. Assuming that there is no link loss, we use these latency requirements to determine the maximum connection interval. Figure 14 shows the resulting throughput and QoS achievement ratio for each slave.

In all the comparison schemes, we observe similar or more throughput degradation and decreased QoS achievement ratio, compared to the previous experiment results. It is noteworthy that when the connection interval is the same, slave 2 in the Zephyr default operation does not degrade the performance of slave 1 a lot. But if their connection intervals are different, joining of slave 2 degrades the performance of slave 1 dramatically. The reason is that when the connection intervals are the same, in the Zephyr default operation, the first anchor point always lies within the idle time, and this is the same for the next connection interval. However, when slave 2 uses a connection interval smaller than that of slave 1, even though the scanning point is idle and the first anchor point happens to be in the idle cell, the next anchor point may overlap the ongoing connection event of slave 1. On the contrary, *BLEX* periodically checks the schedule and calculates the appropriate anchor point according to the interval of new connection, so it guarantees the QoS of existing connections regardless of whether the connection intervals are the same or not.

5.4 Connection Interval Conversion

The last thing to look at is the effect of connection interval conversion on throughput performance. Previously, we converted the connection interval into a schedulable value to apply *BLEX* scheduling. However, sometimes, this value might not exist within the possible connection interval range that the host defined. We show how the performance of *BLEX* varies when *BLEX* cannot convert the connection interval value. Figure 15 shows the experimental results according to the connection interval of slave 4 in the above multiple QoS applications scenario. In particular, in this case, we measured the throughput without converting the connection interval value of slave 4 to the value we want.

No matter which value slave 4 uses as the connection interval, *BLEX* outperforms the Zephyr default operation. This is because even if *BLEX* cannot convert the connection interval of slave 4, *BLEX* can schedule the other slaves. In the worst case, even if the connection intervals of all the slaves are not convertible, it can be expected that *BLEX* achieves at least the performance of Zephyr default operation.

6 RELATED WORK

In wireless communication, resource scheduling of multiple devices is a *long-loved* topic because multiple devices share limited frequency and time resources [14, 15, 17, 22–24, 27]. In particular, in low-power and lossy networks (LLNs), resource scheduling is a more critical issue because LLN devices cannot support complex mechanisms or have a centralized server. Even more problematic, the number of devices that make up LLNs is growing rapidly, but resources such as frequency or computation power are limited.

Time slotted channel hopping (TSCH), a low-power medium access control (MAC) protocol, defined in IEEE 802.15.4e, is similar to BLE in that it provides time-synchronization, channel hopping, and periodic wake-up. Therefore, it is meaningful to look at TSCH slot scheduling systems for designing a BLE scheduling system. The latest resource scheduling techniques for TSCH [18, 23, 24, 27, 28] operate in a distributed or autonomous manner, allocating time slots to each node or directional link when the routing topology and/or traffic load vary. The authors in [24] allocate time slots by adaptively adjusting the slotframe, considering the topology and the amount of traffic generated by each node. In [27], the proposed scheme is not traffic aware, but reduces collisions by allocating time-slots for each TX-RX link. The scheme proposed in [23] went further from this and created a binary resource tree to reduce collisions by allocating dedicated time slots for each link while being aware of the traffic amount. However, unlike TSCH, in BLE:

- There is no discrete-time slot. Instead of allocating time slots, we set only a connection event's start time (i.e., anchor point).
- It is necessary to exchange control packets to adjust parameters because BLE is connection-based.

- The default network configuration is a single-hop star topology, not a multi-hop topology. Therefore, the master in BLE does resource scheduling in a centralized manner.

Therefore, BLE scheduling cannot use TSCH scheduling as it is, and has been covered in several recent studies.

While a number of studies on BLE scheduling considered single-connection scenarios [29, 32, 33], there are some prior work that handled multiple connections [14, 15]. The authors in [14] studied a method to reduce power consumption and increase the packet delivery ratio, and proposed a scheduling scheme for beacons, a.k.a. advertising packets. However, since using advertising packets is initially intended to transmit data that does not require high QoS, robust communication is not essential. The authors of [15] considered QoS for scheduling in data channels, but they only adjusted the connection interval without considering the anchor point of the connection event. However, there is a limit to solving the problems with just the connection interval, described in the Motivation section. As far as we know, there are no papers that improve the QoS of BLE data channels by adjusting the anchor points of connection events.

Since Zephyr became a project of the Linux Foundation in 2016, Zephyr has been increasing its share in BLE research through the modifiable BLE controller, which was challenging before. In particular, in [34], M. Spork et al. implemented a scheme that detects interference through the number of retransmissions and adjusts the connection interval, and evaluated its performance through Zephyr. In [35], the authors improved BLE's adaptive frequency hopping performance by using the packet delivery ratio and signal-to-noise ratio for each channel, which has been challenging to use. The authors in [20] proposed a scheme that uses the FSK (-like) signal of BLE as an ultra-low power FSK wake-up radio and implemented it using Zephyr. In [19], Zehpyr was used as an advertiser to test the proposed new scan algorithm. However, there are no studies that use Zephyr to modify the anchor point of BLE for efficient scheduling.

7 DISCUSSION

BLEX is the first resource scheduling scheme that adjusts anchor points flexibly, which does not mean that it closes the research area. In this section, we look at the potential to further enhance the performance of *BLEX*.

About additional HCI commands and control PDU. *BLEX* can be used for various applications. This is because it does not infringe on the Bluetooth specifications and does not require host-level modifications. If we allow the host to specify the traffic to be forwarded during the connection establishment with additional HCI commands and control PDU, we may improve the performance of BLEX further. However, the proposed method of predicting application traffic through connection event length already shows a QoS achievement rate close to 100%. Therefore, the extra gains that can be achieved with additional control are negligible.

On the extension to multi-hop cases. Research on the IEEE 802.15.4 commonly handles multi-hop cases, but there is not much research on BLE for multi-hop [25, 30, 31]. A question arises as to whether we can extend the proposed BLEX to multi-hop. With *BLEX* currently proposed, the master can freely change anchor points of slaves. So if a slave is the master of another node, scheduling the slave's slave causes *chaos*. To overcome this, exchanging additional control PDUs between master and slave is the most intuitive solution. Alternatively, if a slave is also acting as a master, it would be good to deny requests to update anchor points.

About when to update the connection. In our paper, we use 1 msec and 2 msec as the basis for determining whether *BLEX* needs a connection update. However, this may be too short considering the length of one packet, resulting in unnecessary connection updates. Increasing this value so that connection updates occur less frequently will make *BLEX* more stable, but will incur unnecessarily wasted time resource. This value should be well controlled, so there is room for further experiments to find an appropriate value.

8 CONCLUSION

This paper has confirmed that the anchor point of each connection significantly affects the QoS of connections when multiple slaves are connected to one master. After observing that existing scheduling schemes significantly degrade performance despite enough resources, we designed a novel scheduling system, named *BLEX*, to address the problem. *BLEX* adjusts BLE connection's anchor point, aiming to maximize total QoS without sacrificing the QoS of existing connections. We implemented *BLEX* on a real embedded platform and verified that it successfully adjusts the BLE device's anchor point to satisfy a predefined QoS requirement. *BLEX* outperforms the baseline scheme, which uses very naïve anchor point designation. As the first attempt to adapt BLE's anchor point, we believe that this work can be a *stepping stone* for other researchers to go further in this regime.

ACKNOWLEDGMENTS

This research was supported by a grant to Bio-Mimetic Robot Research Center Funded by Defense Acquisition Program Administration, and by Agency for Defense Development (UD190018ID). Saewoong Bahk and Hyung-Sin Kim are the co-corresponding authors. The authors are grateful to Hongchan Kim, Jeongjun Park, and Siyong Choi for their helpful comments and discussions.

REFERENCES

[1] [n.d.]. Android: Hearing aid over BLE. Retrieved May Oct, 2020 from https://source.android.com/devices/bluetooth/asha

[2] [n.d.]. Bluetooth SIG (Special Interest Group). Retrieved May Oct, 2020 from https://www.bluetooth.com/

[3] [n.d.]. Broadcom BCM4356. Retrieved May Oct, 2020 from https://www.broadcom.com/products/wireless/wireless-lan-bluetooth/bcm4356

[4] [n.d.]. CSR8510. Retrieved May Oct, 2020 from https://www.qualcomm.com/products/csr8510

[5] [n.d.]. ios: Hearing aid over BLE. Retrieved May Oct, 2020 from https://developer.apple.com/documentation/avfoundation/avaudiosession/port/1616624-bluetoothle

[6] [n.d.]. LE Audio Specifications: The next generation of Bluetooth® audio. Retrieved May Oct, 2020 from https://www.bluetooth.com/specifications/le-audio/

[7] [n.d.]. The next generation of Bluetooth® audio. Retrieved May Oct, 2020 from https://www.bluetooth.com/specifications/le-audio/

[8] [n.d.]. nRF52840 DK. Retrieved May Oct, 2020 from https://www.nordicsemi.com/Software-and-Tools/Development-Kits/nRF52840-DK

[9] [n.d.]. Signia hearing aids' direct streaming. Retrieved May Oct, 2020 from https://www.signiausa.com/direct-streaming/

[10] [n.d.]. Traditional Profile Specifications: Profiles make Bluetooth technology interoperable. Retrieved May Oct, 2020 from https://www.bluetooth.com/specifications/profiles-overview/

[11] [n.d.]. Zephyr Project. Retrieved May Oct, 2020 from https://zephyrproject.org/

[12] M. A. Ayoub and A. M. Eltawil. 2020. Throughput Characterization for Bluetooth Low Energy with Applications in Body Area Networks. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–4. https://doi.org/10.1109/ISCAS45731.2020.9180727

[13] M. A. Ayoub and A. M. Eltawil. 2020. Throughput Characterization for Bluetooth Low Energy with Applications in Body Area Networks. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–4. https://doi.org/10.1109/ISCAS45731.2020.9180727

[14] D. Chen, Y. Zheng, Y. Chen, and K. Lee. 2020. Online Power Management for Latency-Sensitive Bluetooth Low-Energy Beacons. *IEEE Systems Journal* 14, 2 (2020), 2411–2420.

[15] J. Chen, Y. Chen, and Y. Jiang. 2018. Energy-Efficient Scheduling for Multiple Latency-Sensitive Bluetooth Low Energy Nodes. *IEEE Sensors Journal* 18, 2 (2018), 849–859.

[16] Yan Chen, Toni Farley, and Nong Ye. 2004. QoS Requirements of Network Applications on the Internet. *Systems Management* 4 (01 2004). https://doi.org/10.1145/1030194.1015475

[17] J. Choi, S. Y. Jung, S. Kim, D. M. Kim, and P. Popovski. 2019. User-centric resource allocation with two-dimensional reverse pricing in mobile communication services. *Journal of Communications and Networks* 21, 2 (2019), 148–157.

[18] Atis Elsts, Seohyang Kim, Hyung-Sin Kim, and Chongkwon Kim. 2020. An empirical survey of autonomous scheduling methods for TSCH. *IEEE Access* 8 (2020), 67147–67165.

[19] P. Gavrikov, M. Lai, and T. Wendt. 2019. A Low Power and Low Latency Scan Algorithm for Bluetooth Low Energy Radios with Energy Detection Mechanisms. In *2019 IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob)*. 1–6. https://doi.org/10.1109/APWiMob48441.2019.8964210

[20] P. Gavrikov, P. E. Verboket, T. Ungan, M. Müller, M. Lai, C. Schindelhauer, L. M. Reindl, and T. Wendt. 2018. Using Bluetooth Low Energy to trigger an ultra-low power FSK wake-up receiver. In *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. 781–784. https://doi.org/10.1109/ICECS.2018.8618031

[21] M. Gentili, R. Sannino, and M. Petracca. 2016. BlueVoice: Voice communications over Bluetooth Low Energy in the Internet of Things scenario. *Computer Communications* 89-90 (2016), 51 – 59. https://doi.org/10.1016/j.comcom.2016.03.004 Internet of Things Research challenges and Solutions.

[22] S. G. Hong, J. Park, and S. Bahk. 2020. Subchannel and power allocation for D2D communication in mmWave cellular networks. *Journal of Communications and Networks* 22, 2 (2020), 118–129.

[23] S. Jeong, H.-S. Kim, J. Paek, and S. Bahk. 2020. OST: On-Demand TSCH Scheduling with Traffic-Awareness. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. 69–78.

[24] S. Jeong, J. Paek, H.-S. Kim, and S. Bahk. 2019. TESLA: Traffic-Aware Elastic Slotframe Adjustment in TSCH Networks. *IEEE Access* 7 (2019), 130468–130483.

[25] C. Jung, K. Kim, J. Seo, B. N. Silva, and K. Han. 2017. Topology Configuration and Multihop Routing Protocol for Bluetooth Low Energy Networks. *IEEE Access* 5 (2017), 9587–9598. https://doi.org/10.1109/ACCESS.2017.2707556

[26] Hyung-Sin Kim, Sam Kumar, and David E Culler. 2019. Thread/OpenThread: A compromise in low-power wireless multihop network architecture for the Internet of Things. *IEEE Communications Magazine* 57, 7 (2019), 55–61.

[27] S. Kim, H.-S Kim, and C. Kim. 2019. ALICE: Autonomous Link-based Cell Scheduling for TSCH. In *2019 18th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. 121–132.

[28] S. Kim, H.-S. Kim, and C. Kim. 2021. A3: Adaptive Autonomous Allocation of TSCH Slots. In *2021 20th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*.

[29] Taeseop Lee, Jonghun Han, Myung-Sup Lee, Hyung-Sin Kim, and Saewoong Bahk. 2017. CABLE: Connection interval adaptation for BLE in dynamic wireless environments. In *2017 14th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, 1–9.

[30] T. Lee, M. Lee, H. Kim, and S. Bahk. 2016. A Synergistic Architecture for RPL over BLE. In *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. 1–9. https://doi.org/10.1109/SAHCN.2016.7732968

[31] L. Leonardi, G. Patti, and L. Lo Bello. 2018. Multi-Hop Real-Time Communications Over Bluetooth Low Energy Industrial Wireless Mesh Networks. *IEEE Access* 6 (2018), 26505–26519. https://doi.org/10.1109/ACCESS.2018.2834479

[32] Eunjeong Park, Myung-Sup Lee, Hyung-Sin Kim, and Saewoong Bahk. 2020. AdaptaBLE: Adaptive control of data rate, transmission power, and connection interval in bluetooth low energy. *Computer Networks* 181 (2020), 107520.

[33] Michael Spörk, C. A. Boano, and K. Römer. 2019. Improving the Timeliness of Bluetooth Low Energy in Noisy RF Environments. In *EWSN*.

[34] Michael Spörk, Carlo Alberto Boano, and Kay Römer. 2020. Improving the Timeliness of Bluetooth Low Energy in Dynamic RF Environments. *ACM Trans. Internet Things* 1, 2, Article 8 (April 2020), 32 pages. https://doi.org/10.1145/3375836

[35] Michael Spörk, Jiska Classen, Carlo Alberto Boano, Matthias Hollick, and Kay Römer. 2020. Improving the Reliability of Bluetooth Low Energy Connections *(EWSN '20)*. Junction Publishing, USA, 144–155.