Improving the Timeliness of Bluetooth Low Energy in Noisy RF Environments

Michael Spörk
Institute for Technical Informatics
Graz University of Technology
Graz, Austria
michael.spoerk@tugraz.at

Carlo Alberto Boano
Institute for Technical Informatics
Graz University of Technology
Graz, Austria
cboano@tugraz.at

Kay Römer
Institute for Technical Informatics
Graz University of Technology
Graz, Austria
roemer@tugraz.at

Abstract

The ability to communicate within given delay bounds in noisy RF environments is crucial for Bluetooth Low Energy (BLE) applications used in safety-critical application domains. In this work, we experimentally study the latency of BLE communications in the presence of radio interference, and show that applications may incur long and unpredictable transmission delays. To mitigate this problem, we devise a model capturing the timeliness of connection-based BLE communications in noisy RF channels by expressing the impact of radio interference in terms of the number of connection events necessary to complete a successful data transmission (n_{CE}). We show that this quantity can be estimated using the timing information of commands sent over the host controller interface of common BLE devices, hence without additional communication overhead or energy expenditure. We finally show that a BLE application can make use of our model and recent n_{CE} measurements to adapt its connection interval at runtime, improving its performance in the presence of radio interference. Experiments on the popular nRF52840 platform running Zephyr show that a BLE application can effectively increase the timeliness of its communications in noisy RF environments, reducing the number of delayed packets by up to a factor of 40.

Categories and Subject Descriptors

B.8 [**Performance and Reliability**]

General Terms

Design, Measurement, Performance, Reliability.

Keywords

Bluetooth Low Energy, BLE, Dependability, Interference.

1 Introduction

The continuous proliferation of wireless devices leads to an increasing congestion of the RF spectrum; especially in the 2.4 GHz ISM band, where several technologies share the same frequencies [30]. One of these technologies is Bluetooth Low Energy (BLE), which is increasingly used to build Internet of Things (IoT) applications due to its wide adoption in consumer devices such as wearables and smartphones [5].

BLE systems typically need to co-exist with a large number of Wi-Fi devices, which transmit at high data rates, use a significantly higher transmission power, and make use of much wider channel bandwidths (20 or even 40 MHz). Furthermore, Bluetooth-based devices (using either BLE or classic Bluetooth) such as headphones, headsets, smart watches, and fitness trackers, are nowadays becoming ubiquitous, which increases their chances to interfere with each other and experience co-existence issues [1, 18].

Such issues typically manifest in the form of an increased packet loss and a higher amount of re-transmissions, which may affect in turn key performance metrics such as energy efficiency, latency, and throughput [6]. As several BLE-based systems are used in safety-critical application domains such as health care [4, 12] and smart cities [3, 10], it is important to fully understand the impact of radio interference on their performance and to make sure that delay-sensitive applications operate correctly even in noisy RF environments.

Limited number of experimental studies. To date, however, still very little is known about the actual performance of BLE in the presence of interference, especially when it comes to connection-based BLE systems. Existing works focus indeed mostly on BLE discovery [11, 29], or are limited to simulations showing the impact of increasing bit error rates [19, 27]. A few measurement reports carried out using real hardware exist, but are either limited to small-scale experiments in anechoic chambers [24], or only address the interference generated by co-located BLE devices [28].

Unfortunately, the few works available do not allow to get a comprehensive picture of BLE's performance in typical residential and office environments where several wireless networks are co-located. Even worse, some works do not reach the same conclusions: while most simulation works argue that BLE's performance should decrease under interference [19, 27], some of the existing studies do not confirm this [24]. Because of this lack of experimental evidence, the general belief in the community is that BLE is highly reliable also in noisy RF environments by design, thanks to its adaptive frequency hopping (AFH) algorithm [8].

International Conference on Embedded Wireless
Systems and Networks (EWSN) 2019
25–27 February, Beijing, China

ISBN: 978-0-9949886-3-8

No upper bound on latency. The AFH algorithm allows BLE devices to blacklist interfered channels, and to autonomously re-transmit packets on different frequencies until interference is finally avoided. Although this is proven to be an effective method to mitigate co-existence problems [20, 24], it only makes sure that every data packet that is added to the transmission buffer of a BLE radio will *eventually* get transmitted (as long as a connection is not dropped).

As we show in Sect. 3, the presence of interference can introduce significant delays that may affect the performance of a BLE application. To make sure that the communication latency stays within acceptable delay bounds, connection-based BLE applications can *adjust their connection parameters* at runtime [25]. Providing such delay bounds would allow developers to apply theoretical network analysis tools, such as network calculus [21], to their BLE applications.

Unsuitable models. The ability to trim connection parameters at runtime, however, requires proper models capturing the impact of radio interference. Unfortunately, most of the existing models rely on ideal channel conditions [13, 25]. A few models for noisy channels exist [7, 9, 19], but they cannot be used by most BLE devices, as they rely on information that is not available on the BLE host (e.g., bit error rate, employed data channels, and number of CRC errors).

Most BLE controllers are indeed drop-in radio peripherals that hide all communication details to a BLE application running on the host processor. A BLE application may only issue high-level commands, such as adding data to the transmission buffer of the BLE controller. The latter essentially acts as a *black box*, which autonomously handles (re-)transmission and acknowledgment of link-layer packets, buffer management, as well as data channel selection.

Receiving feedback at runtime. Once data is added to the transmission buffer of a BLE controller, the application assumes it is successfully transmitted. The BLE specification [5] does not foresee a standardized way for an application to get information about the number of link-layer retransmissions during a packet exchange, nor specify a link quality indicator. In other words, applications do not receive any feedback from the BLE controller about ongoing link-layer transmissions: neither about loss, nor about latency. Therefore, to be aware of the timeliness of its communications, a BLE application needs to pro-actively exchange application messages to explicitly monitor delays (e.g., by means of round-trip time estimations): an unnecessary communication overhead and an additional energy expenditure.

Contributions. In this paper, we first experimentally study the impact of radio interference on the latency of BLE communications. After showing that the RF noise present in common office environments can significantly decrease the performance of BLE systems, we systematically analyze the timeliness of BLE communications under different interference patterns. Our analysis reveals that, in specific scenarios, BLE's AFH algorithm is unable to cope with the surrounding interference, leading to long delays that may be unacceptable for applications used in safety-critical domains.

To improve the timeliness of BLE in noisy RF environments, we revise the model proposed by Spörk et al. [25], such that it can be used by an application to adapt its connection parameters at runtime. We do so by expressing the impact of interference in terms of *the number of connection events necessary to complete a successful data transmission* (n_{CE}). We show that this quantity can be estimated using the timing information of commands sent over the Host Controller Interface (HCI), the *standardized* interface between host processor and BLE controller. This allows any application compliant to the BLE specification [5] to estimate n_{CE} *without* any extra communication overhead or energy cost.

We experimentally show that the use of HCI timing information allows a more fine-grained and efficient n_{CE} estimation than the exchange of application-level messages to compute the round-trip time. Furthermore, we illustrate how a generic BLE application can efficiently make use of recent n_{CE} estimations to adapt its connection interval at runtime, in order to improve the timeliness of its communications.

Experiments on the popular Nordic Semiconductor `nRF52840` DK [16] using Zephyr [26] confirm that BLE applications estimating the n_{CE} and adapting their connection parameters at runtime following the approach presented in this paper are able to cope with radio interference and to effectively increase their timeliness in noisy RF environments.

After providing the required background information on connection-based BLE communication in Sect. 2, this paper makes the following contributions:

- We experimentally study the latency of BLE communications in the presence of radio interference and show that BLE applications may incur long and unpredictable transmission delays (Sect. 3).
- We revise the timeliness model in [25] by introducing the n_{CE} metric, and show how to estimate its value using only information available to a BLE host (Sect. 4).
- We implement our approach using Zephyr on the `nRF52` radio (Sect. 5), and experimentally evaluate the accuracy and efficiency of the n_{CE} estimation carried out using timing information of HCI commands (Sect. 6).
- We show how an application using recent n_{CE} measurements and our revised timeliness model can adapt its connection interval at runtime (Sect. 7) and increase its timeliness in noisy RF environments (Sect. 8).

After describing related work in Sect. 9, we conclude our paper in Sect. 10, along with a discussion on future work.

2 Connection-based BLE Communication

Compared to the simpler *connection-less* communication mode making use of 3 advertisement channels (37, 38, and 39) to broadcast short data packets, *connection-based* BLE provides bidirectional data transfer between a slave and a master. After an initial setup phase using connection-less primitives, connection-based communication takes place during *connection events* (N_0 ... N_i), as shown in Fig. 1.

The time between the start of two consecutive connection events is defined by the *connection interval* (`conn_int`). During a single connection event, master and slave exchange link-layer packets that may carry application data (yellow). In case no data needs to be sent, master and slave simply exchange link-layer keep-alive packets (dark blue), which only carry the mandatory link-layer header.

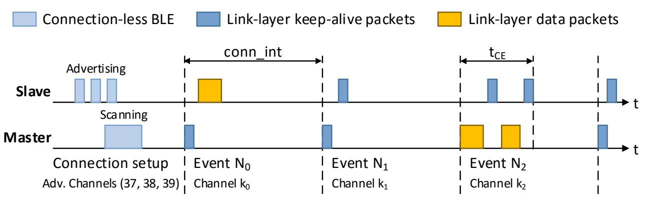


Figure 1. BLE connection between slave and master.

The duration of a connection event depends on the number and the size of exchanged link-layer packets and is limited by the *maximum connection event length* (t_{CE}). Every connection event starts with a transmission from the master, to which the slave responds. Master and slave keep exchanging link-layer data packets until they have all been successfully sent or until t_{CE} is reached. The last link-layer packet during a connection event is always sent from the slave to the master, after which both devices turn off their radio and resume communication at the next connection event.

In the example shown in Fig. 1, during connection event N_0, the master starts the connection event by sending a keep-alive packet to the slave. The slave has data to transmit and therefore responds with a link-layer data packet. Because the slave sends data instead of only a keep-alive packet, its transmission time is longer than the master's. During connection event N_1, master and slave have no data to send and therefore only exchange the mandatory keep-alive packets. In connection event N_2, the master transmits data by sending a data packet. Because the transmission data of the master exceeds the maximum link-layer data packet length, the master waits for a link-layer packet from the slave before completing its data transmission. The slave responds with a link-layer keep-alive packet within the same connection event.

Using connection-based BLE, the link layer automatically handles the *acknowledgment* (ACK) of packets and link-layer flow control using a 1-bit ACK field and a 1-bit sequence number in the header of every link-layer packet (both keep-alive and data packets). In case a link-layer packet is not successfully received, it is automatically re-transmitted without any notification to the BLE application.

Data channel selection. At the beginning of every connection event, one out of 37 possible BLE data channels is selected by the adaptive frequency hopping (AFH) algorithm. A new channel is chosen for every connection event and is used by master and slave to transmit/receive all packets during the event. All 37 possible BLE data channels are located in the unlicensed 2.4 GHz ISM band. The latter, however, is also used by other wireless technologies, such as Wi-Fi, Classic Bluetooth, and IEEE 802.15.4, that may interfere with BLE communications, leading to link-layer packet loss and re-transmissions. The AFH algorithm may choose only a subset of the 37 data channels, defined by the *channel map* (C_{map}) set by the BLE master during connection setup.

To mitigate the effect of co-located wireless applications or multi-path fading, the AFH algorithm may *blacklist* any BLE data channel with poor link quality by updating the C_{map} of the BLE connection at runtime. A data channel disabled in the C_{map} will not be used for communication, but may be *whitelisted* again. Both black- and whitelisting of BLE data channels is performed using standardized BLE commands and may only be initiated by a BLE master.

The BLE specification [5] defines a mandatory delay of at least six connection events between a slave receiving the C_{map}, and the latter being used for actual communication. A slave is required to use the updated channel map, but cannot impose nor suggest changes in C_{map} to the master in a standardized way. This can lead to long transmission delays in case a source of interference is located near the slave and is not detected by the master, as we show in Sect. 3.

Transmission latency. Several models capturing the transmission latency of application data sent over a BLE connection exist [9, 13, 19, 25]. However, only the model proposed by Spörk et al. [25] uses information that is typically available from a BLE radio and can hence be directly used by an application to adapt its connection parameters at runtime. According to this model [25], the *upper bound on transmission latency* of application data sent over a BLE connection on an ideal channel can be computed as:

$$t_{max} = \lceil D/F \rceil \cdot conn\_int + t_{CE}, \qquad (1)$$

where D is the data length in bytes, F is the maximum number of bytes that may be transmitted during a single connection event, *conn_int* is the length of the connection interval, and t_{CE} is the maximum length of a connection event [25].

As we show in Sect. 3, an application cannot rely on this model to compute an upper bound on its end-to-end latency in noisy environments. Radio interference, indeed, causes several link-layer re-transmissions that introduce delays up to four times higher than the ones predicted by this model.

3 BLE Latency in Noisy RF Environments

To demonstrate the impact of radio interference on a BLE connection, we experimentally show that RF noise in a common office environment leads to high transmission latencies over BLE connections (Sect. 3.1). We use a testbed with 9 BLE nodes (Sect. 3.2) to measure the latency of individual data packets in detail and investigate how BLE's AFH algorithm adapts the data channel map over time (Sect. 3.3). Based on our results, we highlight the specific scenarios in which the AFH algorithm is unable to cope with the surrounding interference, leading to long delays (Sect. 3.4).

3.1 Latency in a Common Office Environment

We start by evaluating the transmission latency of a BLE application running in a common office environment for 48 hours. We use a `nRF52840` DK node as slave and connect it via IPv6-over-BLE to a Raspberry Pi 3 (RPi3) master. Master and slave have a distance of approx. 2 meters with direct line of sight. After the IPv6-over-BLE connection is set up, the slave transmits a 29-bytes long UDP packet (resulting in 80 bytes of link-layer payload) to the master once every second. For each UDP packet, we measure the transmission latency ($t_{latency}$) as the time difference between the slave's application issuing the send command to the BLE radio and the master's application successfully receiving the packet.

The two BLE nodes can send F = 128 bytes during a single connection event and have a maximum connection event length of 10 ms. When using the model shown in Eq. 1 with

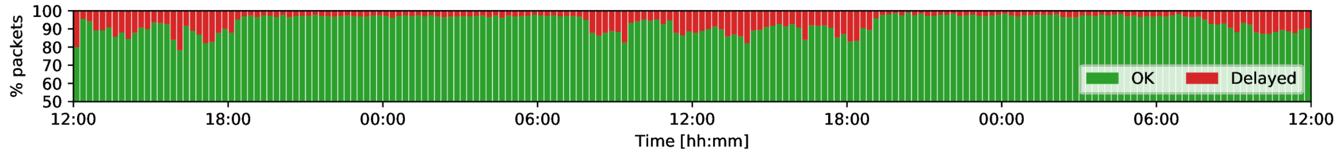


Figure 2. Percentage of packets exceeding the expected upper bound on transmission delay across 48 hours in a common office environment. During daytime, up to 22% of the transmitted packets are delayed due to surrounding interference.

$conn\_int$ = 250 ms, an application would expect a maximum transmission latency $t_{max} = 260$ ms for each UDP packet.

Fig. 2 shows the percentage of data packets that exceed this upper bound on transmission latency over the 48 hours. Each bar refers to 15 minutes, i.e., 900 UDP transmissions. During daytime, when the office is busy, up to 21.74% of UDP packets sent within 15 minutes experience a transmission latency above t_{max}. Several packets even experienced a latency above 1000 ms. During nighttime, when the office is vacant, almost no packets have a latency above 260 ms.

These results show that the RF noise present in a common office environment can have a significant impact on the transmission latency of connection-based BLE. To get a deeper understanding of the impact of different sources of radio interference on the BLE transmission delay, we investigate next the performance of BLE in a systematic way.

3.2 Experimental Setup

We perform our experiments on a testbed allowing us to control the RF noise experienced by each BLE node.

Testbed facility. The testbed consists of nine RPi3 equally distributed over a University lab (6x10 meters) that is kept vacant during our experiments. Each RPi3 runs the Raspbian OS and is connected via USB to one BLE node (`nRF52840` DK device). All RPi3 are connected via Ethernet and use NTP for time synchronization, providing us with the same notion of time across the testbed. Each RPi3 is augmented with a D-Cube board [22] allowing us to accurately measure the power consumption of each BLE node over time.

Generating interference. All RPi3 in the testbed are used to re-program the BLE nodes and to monitor the status of each experiment by logging data in persistent memory. We further use the RPi3s in the testbed to generate Bluetooth and Wi-Fi interference using their on-board Broadcom `bcm43438` radio chip. To generate Bluetooth interference, we configure each RPi3 to create a point-to-point Bluetooth connection with another RPi3, and to transmit RFCOMM packets with a length of 1000 bytes every 11.034 ms, resulting in a RFCOMM data rate of 725 kbits/s. To create Wi-Fi interference, we let each RPi3 generate IEEE 802.11 b/g/n packets of configurable length and rate on a given channel and with a transmission power of 30 mW, using the approach followed by Schuss et al. [23].

BLE master. We use one of the RPi3 as BLE master for all our tests. This RPi3 uses its on-board BLE radio to scan for and connect to nearby IPv6-over-BLE slaves. When an IPv6-over-BLE connection is established, the master starts a UDP server that waits for incoming UDP packets. Every time a UDP packet is received, payload and reception time are logged locally via the serial interface of the node.

BLE slave. We use each of the `nRF52840` devices (only one at a time to avoid self interference) as BLE slave. Each slave waits for the BLE master to initiate an IPv6-over-BLE connection and sends a UDP message to the master every second, once the connection is established. Each UDP message has a length of 29 bytes (resulting in a BLE link-layer packet length of 80 bytes) and carries an 8-digit sequence number in its payload. Whenever the slave sends a UDP message, transmission time and sequence number are logged locally via the serial interface of the node. The slave application uses the Zephyr OS [26] and its existing IPv6-over-BLE stack.

3.3 Experimental Results

Using our testbed setup, we experimentally investigate the loss induced by radio interference on link-layer data packets and their resulting transmission latency. Both master and slave make use of $conn\_int = 250\,ms$, $F = 128\,bytes$, and $t_{CE} = 10\,ms$. As discussed in Sect. 3.1, we expect the upper bound on each transmission t_{max} to be 260 ms (see Eq. 1).

3.3.1 Bluetooth Interference

We first analyze the latency of data transmissions in the presence of classic Bluetooth interference. Similar to BLE, Bluetooth also uses the 2.4 GHz ISM band and makes use of frequency hopping to mitigate external interference by hopping to a new channel every 625 μs [5]. As described in Sect. 3.2, we use the RPi3 in the testbed to create three simultaneous Bluetooth connections, each transmitting with an RFCOMM bandwidth of 725 kbits/s. We further measure the packet latency ($t_{latency}$) as the time difference between the slave issuing the send command and the master being notified about packet reception, as described in Sect. 3.1.

Fig. 3 shows $t_{latency}$ (top) and the data channel map (bottom) of a BLE connection between a master and a slave communicating at a distance of 10 meters with direct line of sight. Data packets exceeding $t_{max} = 260\,ms$ (shown as horizontal dashed line), are marked as *delayed*. After initializing the IPv6-over-BLE connection, we wait 60 seconds for the system to be stable before we simultaneously start interfering on all three Bluetooth connections (time 0 in Fig. 3).

Our results show that every UDP packet is, *eventually*, successfully received. However, Bluetooth interference causes several UDP messages to sustain a $t_{latency} \geq 2 \cdot t_{max}$.

We further see that the AFH algorithm is trying to update the data channel map to mitigate the effect of the Bluetooth interference on the BLE connection. However, the AFH algorithm is not able to predict the frequencies used by Bluetooth and its blacklisting strategy does not help in mitigating the impact of Bluetooth interference on the BLE connection.

Note that the same effect shown in Fig. 3 is experienced by every BLE slave in our testbed, even those that are only 3 meters away from the master and have direct line of sight.

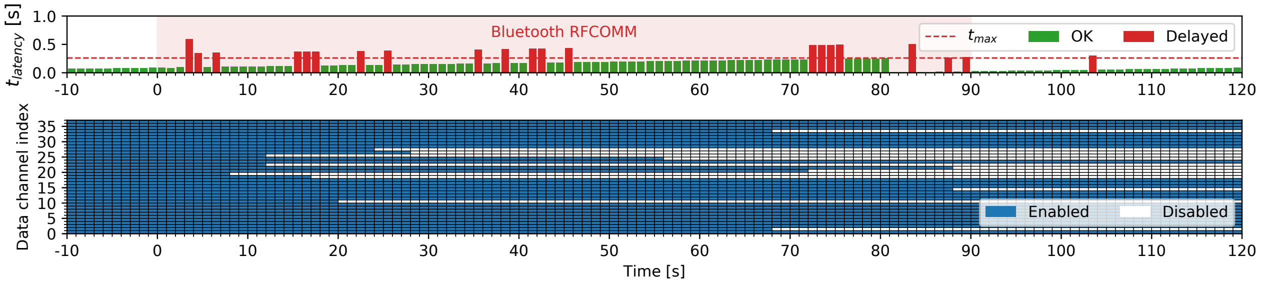


Figure 3. Packet latency ($t_{latency}$) and data channel map of a BLE connection under heavy Bluetooth interference.

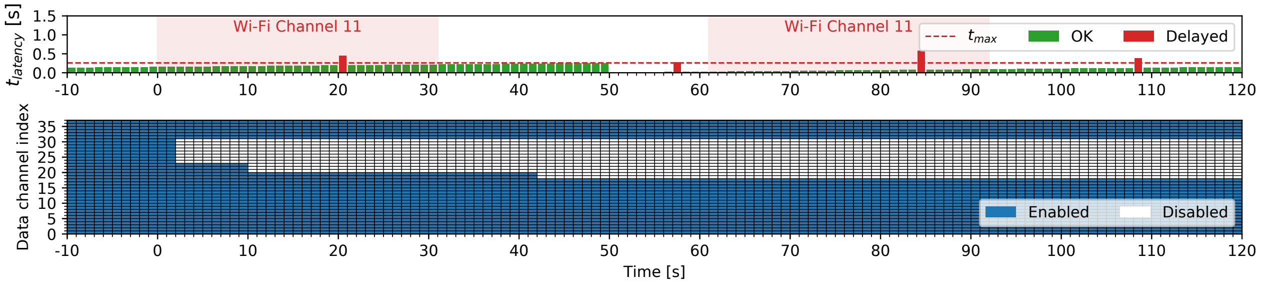


Figure 4. Packet latency ($t_{latency}$) and data channel map of a BLE connection under Wi-Fi interference near the master.

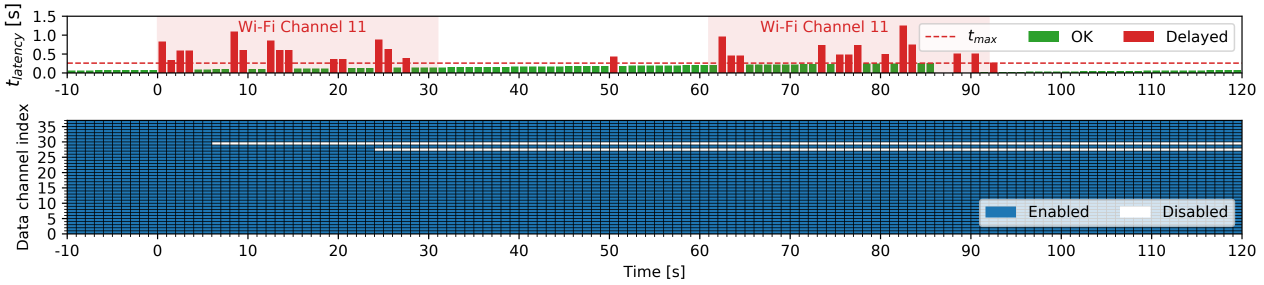


Figure 5. Packet latency ($t_{latency}$) and data channel map of a BLE connection under Wi-Fi interference near the slave.

3.3.2 Wi-Fi Interference

We investigate next the impact of Wi-Fi interference on an active BLE connection. Following the setup described in Sect. 3.2, we first generate Wi-Fi interference in proximity of the BLE master, and then in proximity of a BLE slave.

Wi-Fi interference near the master. Fig. 4 shows the measured $t_{latency}$ (top) and used data channel map (bottom) of a BLE connection in the presence of Wi-Fi interference located near the master. Also in this case, the master and slave are at a distance of 10 meters with direct line of sight. After an initial delay of 60 seconds to let the IPv6-over-BLE connection being set up, we let a RPi3 near the BLE master generate Wi-Fi traffic on channel 11 in bursts of 30 seconds.

Our results show that every UDP packet is *eventually* received. We further see that the AFH algorithm is able to detect the Wi-Fi interference and mitigate its effects on the BLE connection. Despite the heavy traffic generated on Wi-Fi channel 11, indeed, the affected BLE data channels (18 to 31) are blacklisted as soon as the master detects their poor link quality, resulting in only a few delayed packets.

Wi-Fi interference near the slave. Using the aforementioned setup, we now let the RPi3 near the slave generate the same Wi-Fi pattern. Fig. 5 shows the measured $t_{latency}$ (top) and used data channel map (bottom) of a BLE connection in the presence of Wi-Fi interference near the slave.

Once again, every UDP packet is, *eventually*, successfully received. However, compared to the experiments shown in Fig. 4, this time the AFH algorithm is unable to mitigate the effects of Wi-Fi interference on the BLE connection. During Wi-Fi bursts, several UDP messages are significantly delayed, some even with $t_{latency} \geq 5 \cdot t_{max}$. Note that we have observed the same behavior on all slaves located at more than 7 meters distance from the master. The reason for this is the inability of the BLE master to detect the Wi-Fi interference, which does not update the data channel map as shown in Fig. 4. The BLE slave would be able to detect the poor quality of the data channels affected by Wi-Fi traffic. However, it cannot update the data channel map in a standardized way according to the BLE specification, as discussed in Sect. 2.

3.4 Lessons Learned

Our experiments show that BLE connections are *eventually* able to successfully transmit *all* data packets, even under heavy Wi-Fi or Bluetooth interference, hence confirming BLE's high reliability highlighted by [24]. Although no data packet is lost, however, we have observed that the transmission latency significantly increases under interference.

Inefficiency of AFH. In particular, our experiments highlight that, in *two* situations, the AFH algorithm used by BLE is unable to cope with the surrounding interference, leading to long delays. First, the AFH algorithm loses its efficacy in the presence of interference generated by other radio technologies making use of frequency hopping, such as Classic Bluetooth. Second, in the presence of Wi-Fi interference located close to the slave, the master is unable to detect the RF noise and does not update the list of blacklisted channels. In these situations, the number of re-transmissions performed by a BLE connection drastically increases, leading to high latencies that may be unacceptable for safety-critical BLE applications such as health care monitoring [4, 12].

Need for runtime adaptation. In order to avoid such long latencies, delay-sensitive BLE applications need to adjust the connection parameters of their ongoing connections, e.g., by lowering the connection interval according to changes in the link-quality. However, this task is complicated by the fact that the BLE specification [5] does not provide a standardized way for an application to directly get feedback about ongoing link-layer (re-)transmissions or about the quality of a BLE connection. As a consequence, to be aware about the timeliness of its communications, a BLE application needs to pro-actively let the communicating nodes exchange application-level messages to explicitly monitor delays, e.g., by means of round-trip time estimations. Pro-actively exchanging application messages, however, is an unnecessary communication overhead and an additional energy expenditure that is undesirable for resource-constrained BLE nodes.

In the next section, we show that any application compliant to the BLE specification [5] can estimate the impact of interference on an ongoing connection by estimating the number of connection events necessary to complete a successful data transmission. We show how this quantity can be measured *without* any extra communication overhead or energy cost using the timing information of HCI commands.

4 Measuring and Modeling BLE Latency

In this section, we first revise the model shown in Eq. 1 in order to capture the n_{CE}, i.e., the number of connection events necessary to complete a successful data transmission (Sect. 4.1). After discussing the unavailability of link-layer information on standard-compliant BLE host devices (Sect. 4.2), we show how a BLE application can estimate n_{CE} autonomously in two ways. First, we show how to relate n_{CE} to the round-trip time measured by introducing application-layer ACKs (Sect. 4.3). As this method is inaccurate and increases the communication overhead as well as the energy expenditure of BLE devices, we propose a second way to estimate n_{CE} that makes use of the timing information of commands sent over the standardized *Host Controller Interface* between host processor and BLE controller (Sect. 4.4).

4.1 Revising the BLE Timeliness Model

We start by revising the timeliness model from [25] shown in Eq. 1. The latter describes how an application data packet of length D (bytes) is split into data fragments with a maximum size F (bytes), each of which is transmitted on a separate connection event. As discussed in [25], the model neglects the effects of link-layer packet loss on the transmission delay of the individual fragments.

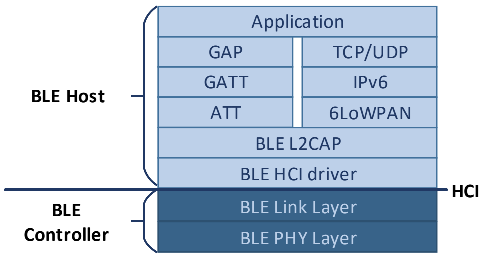


Figure 6. Standard BLE and IPv6-over-BLE stack.

To model the effects of link-layer packet loss and retransmissions, we introduce the n_{CE} metric, which expresses *the number of connection events necessary to successfully transmit individual data fragments*, into the model as:

$$t_{max} = \left(\sum_{f=1}^{\lceil D/F \rceil} n_{CE_f} \cdot conn\_int \right) + t_{CE}, \qquad (2)$$

where $\lceil D/F \rceil$ captures the fragmentation of data with length D into one or multiple data fragments of length F, and n_{CE_f} is the n_{CE} of a *single* data fragment f.

By knowing the n_{CE} of *each* fragment, Eq. 2 now captures the impact of RF noise on the quality of a BLE connection, and is hence able to provide an upper bound on transmission delay. This, however, requires a precise n_{CE} measurement.

4.2 Challenges in Measuring n_{CE}

The main challenge in measuring n_{CE} on a standard-compliant host device is the nature of the BLE communication stack. The latter is split into two separate parts, a *BLE controller* and a *BLE host* [5], that exchange commands via a standardized *Host Controller Interface (HCI)* (see Fig. 6). To simplify the development of BLE applications, the controller implements the physical and link layer – practically acting as a *black box* to the host running the application.

The controller, indeed, provides all services needed for connection-based BLE communication, such as autonomous link-layer retransmissions and acknowledgments, timing of connection events, and data channel selection (including blacklisting) using the AFH algorithm. Controllers are often separate chips that are closed-source and cannot be accessed or modified by developers. The only way for a host to interact with a controller is to provide high-level parameters and listen for HCI events. No info about the BLE connection, such as the number of retransmissions, is passed to the host.

The BLE host implements the upper communication layers of the BLE stack, including the L2CAP layer, the ATT/GATT protocols, and support for IPv6 communication.

Due to the nature of the BLE stack, several challenges arise when measuring n_{CE} on a host, which we discuss next.

Packet transmission. The controller autonomously handles the scheduling of transmissions and the ACK of packets in its transmission buffer. The host can use HCI commands to add a new data packet to the transmission buffer of the controller, but has no implicit control over its timing and no information about when it has actually been sent. The host hence assumes that each packet will be sent, *eventually*, as long as the underlying BLE connection is not dropped.

Buffer management. The controller implements its own management of both reception and transmission buffer. The

BLE host (and hence the application developer) has no direct control over the controller's buffers and can only request the number and length of available buffers in the controller.

Channel selection. At connection setup, the link layer of the BLE master provides the data channel map to the slave. During an active connection, the controller of both slave and master autonomously handles BLE data channel selection. The BLE host, however, has no control or info over the data channel used in current or upcoming connection events.

Link quality information. The BLE specification [5] does not provide any standardized primitive allowing a host to retrieve link quality information about an ongoing connection. Any information about the received signal strength (RSS), the signal-to-noise ratio (SNR), or the number of retransmissions on a BLE data channel is limited to the link layer of the BLE controller. The BLE host is hence unable to retrieve any of these low-level measurements in a standardized way.

Due to these challenges, directly measuring the n_{CE} of an ongoing connection from the BLE host is *not* possible. A host may, however, *estimate* the n_{CE} using application-layer acknowledgments or HCI information, as we show next.

4.3 Estimating n_{CE} using Round-Trip Time

An application can estimate the number of connection events necessary to successfully transmit individual data fragments by using application-layer ACKs and by measuring the round-trip time (t_{RTT}): we refer to this form of n_{CE} estimation as `RTT-based` n_{CE}. When carrying out an `RTT-based` n_{CE} estimation, every data transmission initiated by an application (master or slave) is confirmed by an ACK from the other party's application, as shown in Fig. 7.

An application measures t_{RTT} as the time between the instant in which it adds a data packet P to the transmission buffer of the controller, and the time in which it receives the application-layer ACK A in its reception buffer. The measured t_{RTT} can be expressed as the sum of t_P and t_A:

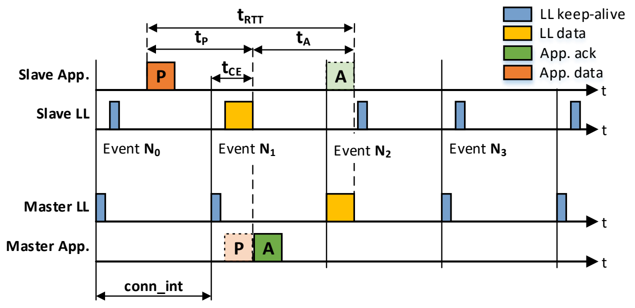
$$t_{RTT} = t_P + t_A,$$

where t_P is the time it takes between P being added to the controller's transmission buffer and being received in the other party's reception buffer. t_A, instead, captures the time between P being received into the receiving buffer, and the subsequent application-layer ACK being received by the application that originally sent P. Fig. 7 shows an example in which a slave sends a data packet consisting of a *single* fragment, and a master replies with an application-level ACK.

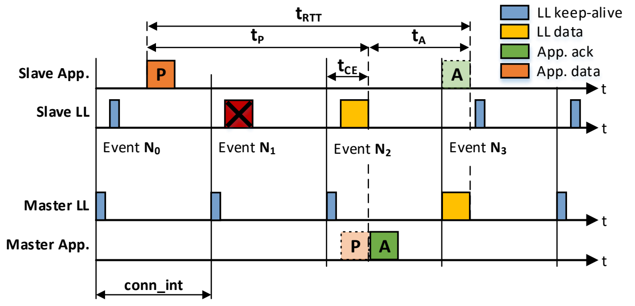
Both data exchanges (actual packet and application-layer ACK) can be modeled as individual data transmissions, each with an upper latency bound t_{max} that is calculated using Eq. 2. For our model, we assume that data packet and ACK have the same length D. Furthermore, because an application has no insight about the performance of each individual fragment, it can only derive a n_{CE_f} that is the same for all fragments involved in the data exchange (data packet and ACK). Following these assumptions, we calculate t_{RTT} as:

$$t_{RTT} \leq 2 \cdot t_{max} \quad or \quad t_{RTT} \leq 2 \cdot \lceil D/F \rceil \cdot n_{CE_f} \cdot conn\_int + 2 \cdot t_{CE}.$$

By measuring t_{RTT}, an application can hence estimate the *average* n_{CE_f} for all fragments in the data exchange as:



(a) No link-layer errors (n_{CE_f} = 1).



(b) One link-layer error during data packet transmission (n_{CE_f} = 2).

Figure 7. `RTT-based` n_{CE} estimation for a slave transmitting a data packet (P) and receiving an ACK (A).

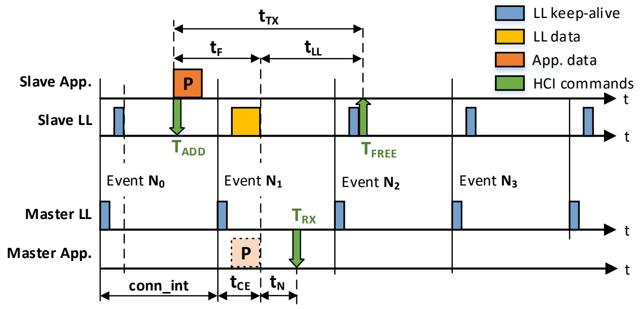
$$n_{CE_f} = \left\lceil \frac{t_{RTT} - 2 \cdot t_{CE}}{2 \cdot \lceil D/F \rceil \cdot conn\_int} \right\rceil. \tag{3}$$

Limitations. A basic requirement to be able to carry out `RTT-based` n_{CE} estimation is that the developer has full control over the application running on both master and slave. This may not necessarily be the case, for example, when a slave acting as IPv6-over-BLE node transmits IPv6 messages to a router (master). Although a developer could force a round-trip time measurement using L2CAP ping messages, `RTT-based` n_{CE} estimation might not be suitable for energy-constrained slaves. The same observation applies when introducing application-layer ACKs, as they increase communication overhead and hence cause an additional power consumption, as we show in Sect. 6.2.

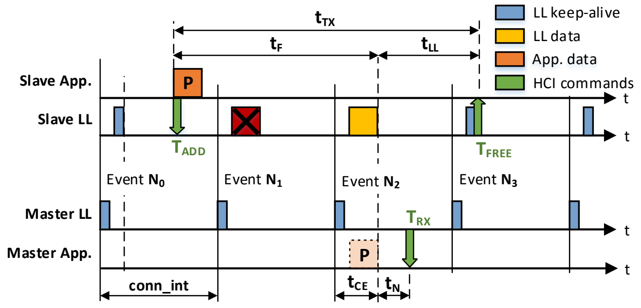
Another limitation of `RTT-based` n_{CE} estimation is that it assumes the *same* n_{CE_f} for all fragments involved in the data exchange. On the one hand, this assumes the link to be symmetric, which may lead to an *underestimation* of n_{CE_f} in case the data packet is retransmitted for several connection events, but the ACK is received immediately. On the other hand, by estimating an average n_{CE_f} for all fragments, `RTT-based` n_{CE} estimation cannot capture the case in which interference leads to a high n_{CE_f} for specific fragments.

4.4 Estimating n_{CE} using HCI Timing Info

In order to tackle the limitations of `RTT-based` n_{CE} estimations, we present another approach that estimates the number of connection events necessary to successfully transmit a data fragment by using HCI timing information. We refer to this form of n_{CE} estimation as `HCI-based` n_{CE} estimation. As HCI commands and events are standardized, *any* BLE-compliant host can make use of this approach. Dif-



(a) No link-layer errors (n_{CE_f} = 1).



(b) One link-layer error during link-layer data transmission (n_{CE_f} = 2).

Figure 8. `HCI-based` n_{CE} estimation for a BLE slave transmitting a packet (P) consisting of one data fragment.

ferent from `RTT-based` n_{CE} estimation, the `HCI-based` approach can discern the n_{CE_f} of each *individual* fragment, and its computation varies for masters and slaves.

4.4.1 Estimating n_{CE} on a BLE slave

Fig. 8 shows the inner working of `HCI-based` n_{CE} estimation for a BLE slave transmitting a packet P consisting of a single data fragment to the master. Compared to Fig. 7, one can notice that the master is no longer sending application-layer ACKs after receiving a packet. Fig. 8 also highlights a number of time-stamps (T_{ADD}, T_{FREE}, and T_{RX}) that can be retrieved from the communication exchanges on the HCI.

Whenever an application needs to transmit data over the BLE connection, it uses the `HCI ACL data packet` command to add data to the transmission buffer of the controller. We define this instant T_{ADD} and measure it in the HCI driver of the host. We also define T_{FREE} as the instant in which the buffer of the controller changes state and measure it by listening for `HCI_Number_Of_Completed_Packets` events. The latter are issued from the controller when a transmission buffer is freed, due to successful data transmission.

Both in the absence (Fig. 8(a)) and in the presence (Fig. 8(b)) of link-layer errors, the only available timing info that can be derived by the slave via the HCI is the time t_{TX} elapsed between the data being added to the controller's transmission buffer (T_{ADD}) and the buffer being actually freed (T_{FREE}):

$$t_{TX} = T_{FREE} - T_{ADD}. \tag{4}$$

According to Fig. 8, we can observe that t_{TX} can be expressed as the sum of two components t_F and t_{LL}:

$$t_{TX} = t_F + t_{LL}, \tag{5}$$

where t_F is the latency of a *single* data fragment (which may carry up to F bytes) into the master's reception buffer,

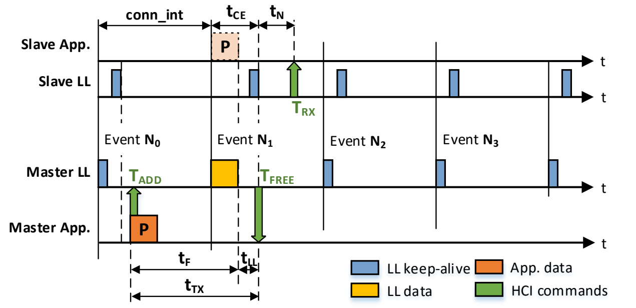


Figure 9. `HCI-based` n_{CE} estimation for a BLE master transmitting a single data fragment (n_{CE_f} = 1).

whereas t_{LL} captures the time between the reception of the data fragment into the master's reception buffer and the slave receiving the link-layer ACK and freeing the buffer (T_{FREE}).

The latency of a single data fragment t_F can be derived from Eq. 2 by setting $D = F$, hence deriving:

$$t_F \leq n_{CE_f} \cdot conn\_int + t_{CE}. \tag{6}$$

Compared to the data fragment that may have a length of up to 255 bytes according to the BLE specification [5], the link-layer acknowledgment only has a length of 16-bits. We therefore assume that the link-layer acknowledgment is successfully transmitted within the first transmission attempt and neglect its duration, resulting in $t_{LL} = conn\_int$. With this assumption, we can calculate t_{TX} (using Eq. 5) as:

$$t_{TX} \leq (1 + n_{CE_f}) \cdot conn\_int + t_{CE}. \tag{7}$$

A BLE application using the HCI communication to measure t_{TX} (using Eq. 7) can hence estimate the current n_{CE_f} as:

$$n_{CE_f} = \left\lceil \frac{t_{TX} - t_{CE}}{conn\_int} \right\rceil - 1. \tag{8}$$

4.4.2 Estimating n_{CE} on a BLE master

`HCI-based` n_{CE} estimation can be used on a master device using the same approach and HCI timing information described in Sect. 4.4.1 (i.e., $t_{TX} = T_{FREE} - T_{ADD}$). The main difference compared to `HCI-based` n_{CE} estimation on a slave is that the link-layer ACK for the data fragment sent by the master comes within the same connection event, as shown in Fig. 9. Therefore, t_{LL} is already captured by the maximum connection event length t_{CE} value in Eq. 6. This allows us to calculate t_{TX} as:

$$t_{TX} \leq n_{CE_f} \cdot conn\_int + t_{CE}. \tag{9}$$

An application running on the BLE master can hence estimate the current n_{CE_f} value using the measured t_{TX} as:

$$n_{CE_f} = \left\lceil \frac{t_{TX} - t_{CE}}{conn\_int} \right\rceil. \tag{10}$$

We describe next the implementation of `RTT-based` or `HCI-based` n_{CE} estimation on the `nRF52840` DK platform using the Zephyr operating system (OS).

5 Implementing n_{CE} Estimation on BLE Hosts

In this section, we present the implementation of a BLE slave using `RTT-based` or `HCI-based` n_{CE} estimation on the Nordic Semiconductor nRF52840 DK platform [16]. The latter embeds an ARM Cortex-M4F application processor, a `nRF52840` chip with 1024 kB of flash and 256 kB of memory, as well as a radio supporting BLE communication up to version 5. Note that the same implementation can be used out-of-the-box on all `nRF52` variants.

Since we use only standardized BLE functionality, our implementation can be ported on every hardware platform that is compliant to the BLE specifications v4.1 and above. Even devices using a proprietary communication interface between BLE host and controller such as the TI CC26xx platform can use our approaches with just minor adaptations.

The Zephyr OS used for our implementation already includes a fully compliant BLE stack (including IPv6-over-BLE support) that uses the standardized HCI to exchange information between the BLE controller and the BLE host.

5.1 `RTT-based` n_{CE} Estimation

We start by implementing the `RTT-based` n_{CE} estimation approach in the slave application described in Sect. 3.2. For every UDP data message (with a UDP length of 29 bytes) sent by the slave, the master responds with an 8-byte long UDP acknowledgment. We measure the transmission time of every UDP message right before it is added to the transmission buffer of the controller. The reception time is measured immediately after the application was notified about the incoming application-layer acknowledgment from the master. Both timestamps measure the current system uptime in milliseconds, which we retrieve by calling `k_uptime_get()`.

After every successful data transmission, the BLE application calculates the round-trip time t_{RTT} of the recent data exchange and estimates the current n_{CE_f} value using Eq. 3.

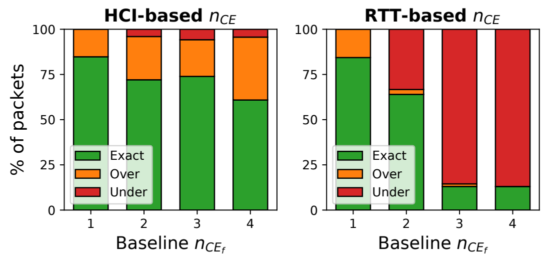
5.2 `HCI-based` n_{CE} Estimation

We next implement `HCI-based` n_{CE} estimation reusing the slave application from Sect. 3.2 and adding the n_{CE} measurements to the BLE host in the HCI driver layer (`hci_core`). Every time the host sends a `HCI ACL Data Packet` command to the BLE controller in order to transmit application data, we store the current system uptime as T_{ADD}. When the BLE controller issues an `HCI_Number_Of_Completed_Packets` event to notify the host about the successful data transmission, we store T_{FREE} as the current system uptime. T_{ADD} and T_{FREE} are retrieved using `k_uptime_get()` and measured in milliseconds.

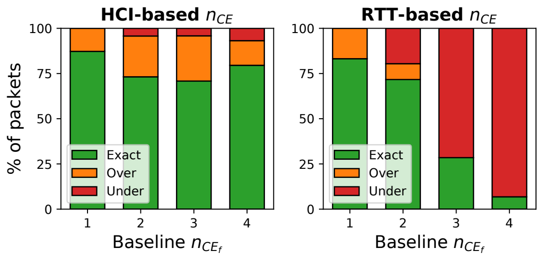
The n_{CE} estimation is performed in the HCI driver layer on the host using Eq. 8 each time the controller has successfully transmitted a data fragment. To provide BLE applications with the possibility to retrieve the current n_{CE_f} when using `HCI-based` n_{CE} estimation, we extend the HCI driver with the function `bt_hci_get_nce()`, which returns the most recent n_{CE_f} estimate. This function is a custom addition to the BLE stack and can be added to any BLE controller driver, independently of the type of communication used between BLE host and controller (HCI or proprietary).

6 Evaluating the Accuracy and Efficiency of n_{CE} Estimation

We experimentally evaluate the accuracy (Sect. 6.1) and efficiency (Sect. 6.2) of `RTT-based` or `HCI-based` n_{CE} estimation. We focus our evaluation on the BLE slave device, since (i) it is typically more constrained in its energy budget and processing power than the BLE master, and since (ii) the `HCI-based` n_{CE} estimation on a slave is by design less accurate than on a master, as discussed in Sect. 4.



(a) Connection interval = 62.5 ms



(b) Connection interval = 125 ms

Figure 10. Accuracy of `HCI-based` and `RTT-based` n_{CE} estimation for two connection intervals.

6.1 Accuracy

We make use of the same experimental setup described in Sect. 3.2. We run one estimation approach at a time and measure the n_{CE_f} for each data fragment by computing the end-to-end latency from slave to master $t_{latency}$ as follows:

$$t_{latency} = T_{RX} - T_{ADD}.$$

We then compute our baseline n_{CE_f} for each data fragment based on the measured $t_{latency}$ as:

$$n_{CE_f} = \left\lceil \frac{t_{latency} - t_{CE}}{\lceil D/F \rceil \cdot conn\_int} \right\rceil.$$

Note that, as described in Sect. 3.2, the RPi3 nodes connected to the master and the slave are NTP-synchronized, giving us the same notion of time across the two nodes.

For both `RTT-based` and `HCI-based` n_{CE} estimation, slave and master exchange 600 UDP packets consisting of a single fragment using two connection intervals (62.5 and 125 ms) in the presence of Wi-Fi interference near the slave. We repeat our measurements ten times for each setting.

Fig. 10 plots the percentage of UDP packets for which the resulting fragment's n_{CE_f} has been correctly estimated, overestimated, or underestimated (green, orange, and red, respectively). We can clearly see that the number of correctly-estimated n_{CE_f} values is higher when using `HCI-based` n_{CE} estimation, especially in the presence of highly unreliable BLE connections (bars with a high baseline n_{CE_f} value).

Overall, `HCI-based` n_{CE} estimation outperforms the `RTT-based` one by 0.42, 8.06, 60.87, and 47.82% for a n_{CE_f} of 1, 2, 3, and 4, respectively, in estimating the exact n_{CE_f} value (*conn_int*=62.5 ms). Furthermore, Fig. 10 also hints that `HCI-based` estimation is far less likely to *underestimate* the n_{CE_f} value of a fragment than `RTT-based` estimation. `HCI-based` n_{CE} estimation reduces the number of underestimations by 29, 80, and 83% for a n_{CE_f} of 2, 3, and 4, respectively (*conn_int*=62.5 ms). Similar trends are observed when using a *conn_int*=125 ms (Fig. 10(b)).

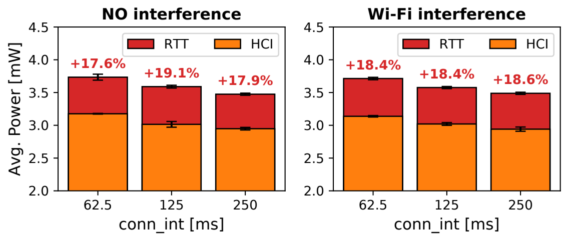


Figure 11. Average power consumption of the n_{CE} estimators for different connection intervals and interference.

The few cases in which `HCI-based` n_{CE} estimation underestimates the baseline n_{CE_f} value ($\leq$ 0.9% of all cases) are caused by uncontrollable delays in the notification introduced by the OS on the BLE master (shown as t_N in Fig. 8).

6.2 Power Consumption

We measure the average power consumption of both `RTT-based` and `HCI-based` n_{CE} estimation under different interference patterns, following the same experimental setup described in Sect. 3.2. We measure the power consumption of a `nRF52840` slave using the D-Cube board [22].

Fig. 11 shows the average power consumption for different connection intervals in absence and in the presence of Wi-Fi interference. We can observe that, regardless of the connection interval uses, and of the presence of Wi-Fi interference, the `RTT-based` n_{CE} estimation adds an extra 18% power consumption on the slave. This higher power consumption is due to the communication overhead introduced by the exchange (unnecessary when using `HCI-based` n_{CE} estimation) of application-level acknowledgments.

7 Increasing the Timeliness of BLE using n_{CE}

To increase the timeliness of BLE applications in noisy RF environments, we can use n_{CE} information to adapt the BLE connection interval at runtime in order to mitigate the presence of interference while minimizing energy consumption (Sect. 7.1). Towards this goal, we can use a series of recent n_{CE_f} estimates to predict the n_{CE_f} of upcoming data fragment transmissions (Sect. 7.2).

7.1 Adapting the BLE Connection at Runtime

Following Eq. 2, a delay-sensitive BLE application is able to compute the maximum connection interval allowing its communications to sustain an upper bound on the transmission delay t_{max} despite the presence of surrounding interference. From Eq. 2 we can indeed derive:

$$conn\_int_{max} \leq \frac{t_{max} - t_{CE}}{\lceil D/F \rceil \cdot n_{CE_{f\star}}}. \tag{11}$$

where $n_{CE_{f\star}}$ is the expected number of connection events necessary to successfully transmit *upcoming* data fragments.

Depending on the $conn\_int_{max}$ computed using Eq. 11, the slave application can request a new connection interval from the master[1]. $conn\_int_{max}$ represents the *most energy-efficient* connection interval to be used in order to sustain the upper bound on transmission delay t_{max} — provided that $n_{CE_{f\star}}$ correctly captures the expected number of connection events necessary to successfully transmit *upcoming* data fragments. We discuss in the next section how an application can make use of the recent n_{CE_f} estimates to predict this value.

[1]To this end, the slave can use the standardized `L2CAP CONNECTION PARAMETER UPDATE REQUEST`. A master may change the connection interval by issuing a `LL CONNECTION UPDATE` command. According to the BLE specification, there is a fixed delay of at least six connection events between the slave receiving the new connection interval, and the latter being used.

7.2 Predicting Future n_{CE_f} Values

Using a series of recent n_{CE_f} measurements, we can predict the expected number of connection events necessary to successfully transmit *upcoming* data fragments ($n_{CE_{f\star}}$). This allows us to find the most efficient connection interval $conn\_int_{max}$ and adapt the BLE connection so that future data transmissions do not exceed the maximum transmission delay t_{max}. To achieve this goal, we use a simple filtering approach that calculates the maximum n_{CE_f} value out of a given observation window of L fragments. Research on IEEE 802.15.4 communication has shown that such a maximum filtering approach can be used to select suitable communication parameters that minimize the number of packets exceeding an upper latency bound [14]. Using such a filtering approach, we can predict $n_{CE_{f\star}}$ as:

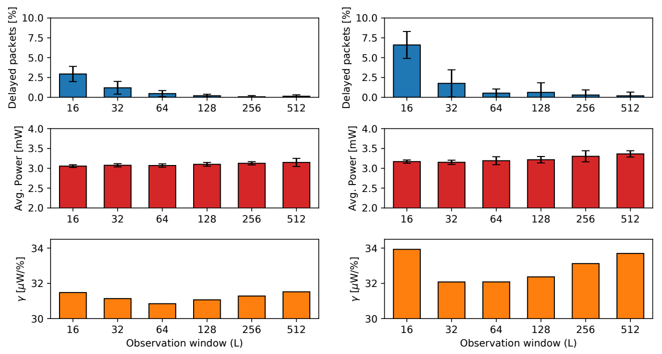
$$n_{CE_{f\star}} = max[n_{CE_f}(t), \ldots, n_{CE_f}(t-L)], \tag{12}$$

where $n_{CE_{f\star}}$ is the predicted number of connection events necessary to successfully transmit upcoming data fragments, whereas n_{CE_f} (t) to n_{CE_f} (t - L) are the latest n_{CE_f} estimates obtained following the approach explained in Sect. 4.

Finding an optimal ***L*****.** We next experimentally investigate a suitable observation window length L. We consider six different lengths (16, 32, 64, 128, 256, and 512) and compute $n_{CE_{f\star}}$ according to Eq. 12. We then instruct a BLE slave to adapt its connection interval according to Eq. 11 and experimentally measure (i) the number of delayed packets (i.e., the number of packets whose latency exceeds the expected upper bound t_{max}), and (ii) the energy consumption of the slave over time. We make use of the same setup described in Sect. 3.2, i.e., a slave and a master communicating using t_{max} = 260 ms, t_{CE} = 10 ms, and F=128 bytes in the presence of Bluetooth and Wi-Fi interference near the slave.

In principle, we expect the number of delayed packets to be high when using a short observation window. When using a short L, indeed, the limited information about the amount of interference affecting the channel in the recent past translates in an optimistic prediction (higher $conn\_int$). At the same time, we also expect that, when using a longer L, at least one of the observed n_{CE_f} values captures a burst of interference and hence results in a pessimistic prediction (lower $conn\_int$), leading to a higher radio activity and, therefore, a higher energy consumption of the system.

Fig. 12 shows the results of our evaluation. As expected, the percentage of delayed packets decreases for larger observation windows, whilst the average power consumption of the system increases. To find the optimal L, we calculate the power consumption necessary to transmit a timely packet γ $[\mu W/\%]$ for the different values of L. Fig. 12 (bottom) shows that selecting L=64 offers a good trade-off between energy-efficiency and timeliness of BLE transmissions under both Bluetooth (Fig. 12(a)) and Wi-Fi (Fig. 12(b)) interference. With this setting, only 0.6% of all data transmissions exceed the latency bound, even under Wi-Fi interference.



(a) Bluetooth interference (b) Wi-Fi interference

Figure 12. Percent of delayed packets (top), average power consumption (middle), and power cost (bottom) for different observation windows under interference.

8 Evaluating the Timeliness of BLE in Noisy RF Environments when using n_{CE}

We finally evaluate the timeliness of BLE applications adapting their connection parameters using the approach proposed in Sect. 7 systematically (Sect. 8.1) and in common office environments rich of RF noise (Sect. 8.2).

8.1 Systematic Evaluation

We compare the performance of a slave S_{fixed} running an application using a fixed connection interval to a slave S_{adapt} using the approach shown in this paper. S_{fixed} selects its connection interval statically according to Eq. 1 in order to sustain a maximum transmission delay $t_{max} = 260ms$. S_{adapt}, instead, uses Eq. 11 and an observation window length L=64 to adapt its connection interval as described in Sect. 7.

Setup. Using the same setup described in Sect. 3.2, we let each of the two slaves transmit 500 UDP packets to a master located at 10 meters distance with direct line of sight. We run only one slave at a time and repeat each experiment ten times. We analyze the performance of S_{fixed} and S_{adapt} in absence of RF noise, in the presence of Bluetooth interference, and with Wi-Fi interference located close to the slave.

Results. Fig. 13 shows the percentage of delayed packets and the average power consumption of S_{fixed} (orange) and S_{adapt} (red). We can clearly see that, whilst S_{fixed} experiences an amount of delayed packets between 6.8 and 24.6%, almost the entirety of packets transmitted by S_{adapt} (at least 99.45%) are within the expected delay bounds. Adapting the connection interval at runtime to mitigate the effects of surrounding radio interference comes, as expected, at the cost of an increased energy consumption. Our experiments show that S_{adapt} incurs an additional power consumption of 7.42% in absence of interference, and of 9.51 and 17.96% in the presence of Bluetooth and Wi-Fi interference, respectively.

8.2 Common Office Environments

To finally prove the efficacy of our proposed method, we re-run the same application described in Sect. 3.1 in the same office environment populated with employees (same location of nodes). We configure the BLE slave to adapt its connection interval at runtime as described in Sect. 7, and make use of an observation window of length L=64.

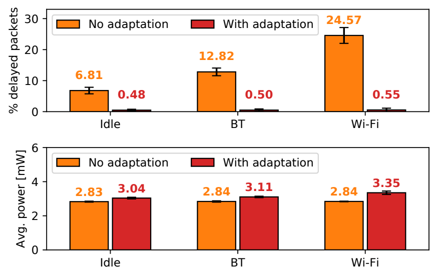


Figure 13. Delayed packets and power consumption of a slave with and without connection interval adaptation.

Fig. 14 shows the number of delayed packets and the adaptation of the connection interval across 48 hours. At most 1.34% of the UDP packets sent within 15 minutes exceed the latency bounds. In Fig. 2, the number of delayed packets was up to 21.74%. The average number of packets delayed is 0.54% (compared to 6.18% obtained in Fig. 2). These results show the effectiveness of our approach, which allows BLE applications to significantly increase the timeliness of their communications in noisy RF environments.

9 Related Work

Several studies have investigated the performance of low-power wireless technologies under interference [2, 17]. While these works mostly focus on IEEE 802.15.4, only a few studies investigate the performance of BLE under interference or the latency of its communications.

BLE performance under interference. Most of the works studying the performance of BLE in the presence of interference carry out *analytic* investigations. Existing works focus either on the performance of *device discovery* [11, 27, 29], or of *BLE connections* [9, 13, 19, 25]. Only a few works actually measure the performance of BLE under interference experimentally [15, 24, 28]. These studies, however, lack practicality, as they are performed in a small anechoic chamber [24], or artificially constrain the performance of BLE's AFH algorithm by disabling channel blacklisting [15, 28].

In this paper, to the best of our knowledge, we provide the first comprehensive study investigating the performance of BLE connections under different interference patterns. We carry out not only experiments in common office environments, but also a systematic evaluation in testbeds.

Modeling BLE latency. In this paper we also develop the first model capturing the timeliness of connection-based BLE communications in noisy environments *that can be used on any BLE host device*. Existing works, indeed, model the latency of BLE connections using information that is not available to the application, such as bit error rate, number of CRC errors, or data transmission probability [9, 13, 19]. Differently from these works, we only embed in our model quantities that a standard host device is able to measure. Other timeliness models either focus on *device discovery* [11, 29], or assume *perfect channel* conditions [7, 25].

10 Conclusions and Future Work

In this work, we experimentally study the latency of BLE communications in the presence of radio interference, and show that BLE applications may incur long and unpre-

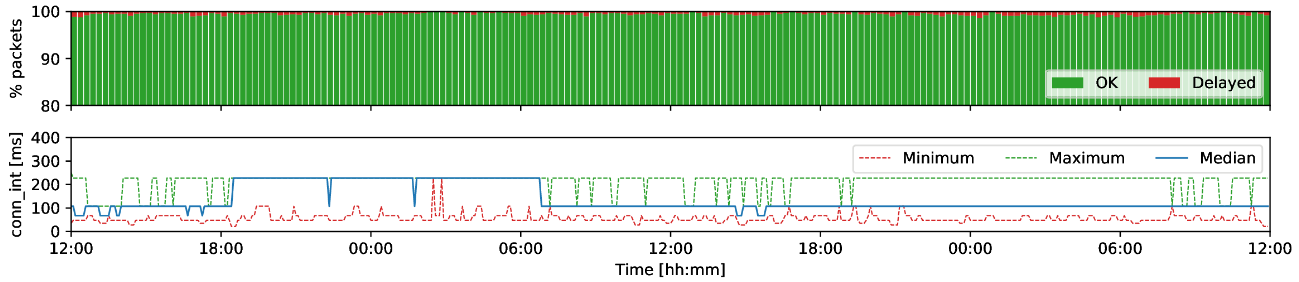


Figure 14. When adapting its connection interval at runtime using an observation window of 64 fragments, a slave is able to significantly increase the timeliness of its communications, resulting in at most 1.34% of UDP packets delayed.

dictable transmission delays. To mitigate this problem, we devise a model capturing the timeliness of connection-based BLE communications in noisy RF environments that can be used on any BLE host device. We do so by expressing the impact of interference in terms of the number of connection events necessary to successfully transmit individual data fragments (n_{CE}), a quantity that can be measured – among others – by using the timing information of commands sent over the HCI interface between host processor and BLE controller. This allows any BLE application to adapt its connection parameters at runtime without additional communication overhead, and to increase its timeliness also in noisy environments. Hence, our work paves the way towards the use of Bluetooth Low Energy for real-time IoT applications.

Future work includes the refinement of n_{CE} estimation, as well as the improvement of the performance of BLE's adaptive frequency hopping algorithm under interference. Both of these potential improvements, however, require control over the inner workings of the BLE controller black box.

11 Acknowledgments

This work has been performed within the LEAD project "Dependable Internet of Things in Adverse Environments" funded by Graz University of Technology. This work was also partially funded by the SCOTT project. SCOTT (`http://www.scott-project.eu`) has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 737422. This joint undertaking receives support from the European Unions Horizon 2020 research and innovation programme and Austria, Spain, Finland, Ireland, Sweden, Germany, Poland, Portugal, Netherlands, Belgium, Norway. SCOTT is also funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between May 2017 and April 2020. More information at `https://iktderzukunft.at/en/`.

12 References

[1] N. Amanquah and J. Dunlop. Improved Throughput by Interference Avoidance in Co-Located Bluetooth Networks. In *Proc. of the 5th EPMCC Conf.*, 2003.

[2] N. Baccour et al. Radio Link Quality Estimation in Wireless Sensor Networks: a Survey. *ACM Trans. on Sensor Networks*, 8(4), 2012.

[3] A. K. Bhattacharjee et al. Extending Bluetooth Low Energy PANs to Smart City Scenarios. In *Proc. of the SMARTCOMP Conf.*, 2017.

[4] BLE Home. iAlert Sensing Motion: Quick Start Guide, 2017.

[5] Bluetooth SIG. Bluetooth Core Specification v5.0, 2018.

[6] C. A. Boano and K. Römer. External radio interference. In *Radio Link Quality Estimation in Low-Power Wireless Networks*. 2013.

[7] K. Cho et al. Analysis of Latency Performance of Bluetooth Low Energy (BLE) Networks. *Sensors*, 15(1), 2014.

[8] M. Collotta and G. Pau. A Solution Based on Bluetooth Low Energy for Smart Home Energy Management. *Energies*, 8, 2015.

[9] M. H. Dwijaksara, W. S. Jeon, and D. G. Jeong. A Channel Access Scheme for Bluetooth Low Energy to Support Delay-Sensitive Applications. In *Proc. of the 27th PIMRC Symp.*, 2016.

[10] B. Islam et al. Rethinking Ranging of Unmodified BLE Peripherals in Smart City Infrastructure. In *Proc. of the 9th MMSys Conf.*, 2018.

[11] W. S. Jeon, M. H. Dwijaksara, and D. G. Jeong. Performance Analysis of Neighbor Discovery Process in Bluetooth Low-Energy Networks. *IEEE Transactions on Vehicular Technology*, 66(2), 2017.

[12] H. Karvonen et al. Interference of Wireless Technologies on BLE Based WBANs in Hospitals. In *Proc. of the 28th PIMRC Symp.*, 2017.

[13] P. Kindt et al. Adaptive Online Power-Management for Bluetooth Low Energy. In *Proc. of the IEEE INFOCOM Conference*, 2015.

[14] S. Munir et al. Addressing Burstiness for Reliable Communication and Latency Bound Generation in Wireless Sensor Networks. In *Proc. of the 9th IPSN Conf.*, 2010.

[15] R. Natarajan, P. Zand, and M. Nabi. Analysis of Coexistence between IEEE 802.15. 4, BLE and IEEE 802.11 in the 2.4 GHz ISM band. In *Proc. of the 42nd IECON Conf.*, 2016.

[16] Nordic Semiconductors. nRF52 Series SoC, 2018.

[17] M. Petrova, J. Riihijarvi, P. Mahonen, and S. Labella. Performance study of IEEE 802.15. 4 using measurements and simulations. In *IEEE Wireless communications and Networking Conf.*, 2006.

[18] P. Popovski et al. Adaptive Mitigation of Self-Interference in Bluetooth Scatternets. In *Proc. of the 7th WPMC Conf.*, 2004.

[19] R. Rondón et al. An Analytical Model of the Effective Delay Performance for BLE. In *Proc. of the 27th PIMRC Symp.*, 2016.

[20] R. Rondón et al. Evaluating Bluetooth Low Energy Suitability for Time-Critical Industrial IoT Applications. *Intl. Journal of Wireless Information Networks*, 24(3), 2017.

[21] J. B. Schmitt and U. Roedig. Sensor network calculus–a framework for worst case analysis. In *Proc. of DCOSS '05 Conf.* Springer, 2005.

[22] M. Schuß, C. Boano, M. Weber, and K. Römer. A Competition to Push the Dependability of Low-Power Wireless Protocols to the Edge. In *Proc. of the 14th EWSN Conf.*, 2017.

[23] M. Schuß et al. JamLab-NG: Benchmarking Low-Power Wireless Protocols under Controlable and Repeatable Wi-Fi Interference. In *Proc. of the 16th EWSN Conf.* Junction Publishing, 2019.

[24] S. Silva et al. Coexistence and interference tests on a Bluetooth Low Energy front-end. In *Proc. of the SAI Conf.*, 2014.

[25] M. Spörk, C. Boano, M. Zimmerling, and K. Römer. BLEach: Exploiting the Full Potential of IPv6 over BLE in Constrained Embedded IoT Devices. In *Proc. of the 15th ACM SenSys Conf.*, 2017.

[26] The Zephyr Project. Zephyr OS: An RTOS for IoT, 2018.

[27] J. Tosi et al. Evaluating Bluetooth Low Energy Suitability for Time-Critical Industrial IoT Applications. *Sensors*, 17(12), 2017.

[28] J. J. Treurniet et al. Energy Consumption and Latency in BLE Devices under Mutual Interference: An Experimental Study. In *Proc. of the 3rd FiCloud Conf.*, 2015.

[29] J. Wyffels et al. Influence of Bluetooth Low Energy on Wi-Fi Communications and Vice versa. In *Proc. of the ECUMICT Conf.* 2014.

[30] G. Zhou, J. A. Stankovic, and S. H. Son. Crowded Spectrum in Wireless Sensor Networks. In *Proc. of the 3rd EmNets Worksh.*, 2006.