Improving the Timeliness of Bluetooth Low Energy in Dynamic RF Environments

MICHAEL SPÖRK, CARLO ALBERTO BOANO, and KAY RÖMER, Graz University of Technology

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, such as health care and smart cities. 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 further show that a BLE device can make use of our BLE timeliness model and recent n_{CE} measurements to adapt its BLE communication parameters at runtime, thereby improving its performance in the presence of dynamic radio interference. We implement such an adaptive scheme on the popular nRF52840 platform and perform an extensive experimental study in multiple indoor environments using three different BLE platforms. Our results show that a BLE application can, indeed, make use of the proposed model and recent n_{CE} measurements to adapt its connection interval at runtime to increase the timeliness of its communications, reducing the number of delayed packets in noisy RF environments by up to a factor of 40.

CCS Concepts: • **Networks** → **Network performance analysis**; • **Computer systems organization** → *Embedded systems*; *Real-time systems*; *Reliability*;

Additional Key Words and Phrases: Bluetooth low energy, BLE, dependability, interference

ACM Reference format:
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

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 Union's Horizon 2020 research and innovation program and Austria, Spain, Finland, Ireland, Sweden, Germany, Poland, Portugal, Netherlands, Belgium, and 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/.

Authors' addresses: M. Spörk, C. A. Boano, and K. Römer, Graz University of Technology, Graz, Austria; emails: {michael.spoerk, cboano, roemer}@tugraz.at.

2577-6207/2020/04-ART8 $15.00
https://doi.org/10.1145/3375836

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 [37]. 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, tablets, laptops, 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, hearing aids, smart watches, and fitness trackers, are becoming ubiquitous, which increases their chances to interfere with each other and experience co-existence issues [1, 21].

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 under interference. 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 mostly on BLE discovery [11, 36] or are limited to simulations showing the impact of increasing bit error rates [25, 34]. A few measurement reports carried out using real hardware exist but are either limited to small-scale experiments in anechoic chambers [30] or only address the interference generated by co-located BLE devices [35].

Unfortunately, the 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 [25, 34], some of the existing studies do not confirm this [30]. Because of this lack of experimental evidence, the general belief in the community is that BLE is highly reliable under radio interference by design, thanks to its autonomous packet re-transmission and its adaptive frequency hopping (AFH) mechanism [8, 15, 29].

No upper bound on latency. By using autonomous retransmissions and AFH, BLE connections re-transmit packets on different frequencies until interference is finally avoided and the packet is successfully sent. Although this is proven to be an effective method to mitigate co-existence problems [24, 30], 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 Section 3, the presence of interference can introduce significant delays that may affect the performance of a BLE application. To minimize the number of data transmissions exceeding a given delay bound, connection-based BLE applications can *adjust their connection parameters* at runtime [31] to increase timeliness at the cost of additional power consumption.

Unsuitable latency models. The ability to adjust 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, 31]. A few models for noisy channels exist [7, 9, 25], 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 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 are 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 as shown in References [14, 15]): an unnecessary communication overhead leading to an additional energy expenditure. Besides the missing feedback on data transmissions, BLE applications are also unable to retrieve any link-quality information of a BLE connection from the BLE black box in a standard-compliant way.

Different AFH behavior. The link-quality information of a BLE connection is indeed monitored internally by the BLE controller as part of the AFH mechanism. As mentioned above, the AFH mechanism of BLE autonomously classifies the available portions of the RF spectrum into BLE channels of good and bad link quality. Once classified as bad, a channel may be blacklisted (disabled) at runtime and therefore not be used by a BLE connection until it is whitelisted (re-enabled) again. Although the primitives to black- and whitelist BLE data channels are standardized by the BLE specification [5], how to measure a channel's link quality and when to actually blacklist a channel is not defined and is left up to the vendor of the BLE platform.

This causes different BLE platforms to have vastly diverse performance, especially in the presence of external radio interference, as we show in Section 3. This, as a consequence, exacerbates the problem of achieving timely BLE communication even further.

Contributions. In this article, 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, state-of-the-art implementations of BLE's AFH mechanism are 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. [31], 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 applications compliant to the BLE specification [5] to estimate n_{CE} *without* introducing any extra communication overhead or additional 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

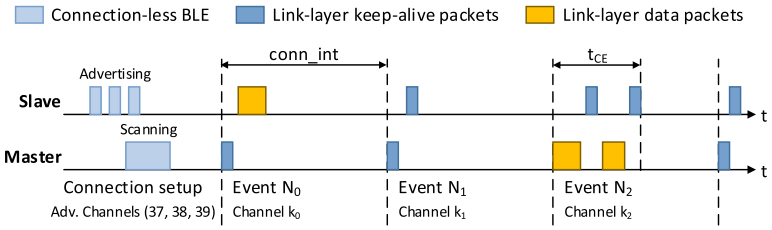


Fig. 1. BLE connection between slave and master.

recent n_{CE} estimations to adapt its connection interval at runtime to improve the timeliness of its communications in noisy RF environments.

Experiments on popular BLE platforms, such as the Nordic Semiconductor `nRF52840` DK [18], the Broadcom BCM43439 [23], the Qualcomm CSR8510 A10 [22], and the Panasonic PAN1762 [19], confirm that BLE applications estimating the n_{CE} and adapting their connection parameters at runtime following the approach presented in this article 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 Section 2, this article makes the following contributions:

- We experimentally study the latency of BLE communications, using three popular BLE platforms with different AFH implementations in the presence of radio interference, and show that BLE applications may incur long and unpredictable transmission delays (Section 3).
- We revise the timeliness model in Reference [31] by introducing the n_{CE} metric and show how to estimate its value using only information available to a BLE host (Section 4).
- We implement our approach using Zephyr on the `nRF52` radio (Section 5) and experimentally evaluate the accuracy and efficiency of the n_{CE} estimation carried out using timing information of HCI commands (Section 6).
- We show how an application using recent n_{CE} measurements and our revised timeliness model can adapt its connection interval at runtime (Section 7) and increase its timeliness in different noisy RF environments, independently of the used AFH implementation (Section 8).

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

This is an extended version of Reference [32], which includes a more detailed investigation of BLE's transmission latencies in different environments using multiple BLE platforms and an extensive evaluation of the dynamic behavior of our proposed adaptation scheme in noisy RF environments.

2 CONNECTION-BASED BLE COMMUNICATION

Compared to the simpler *connection-less* communication mode making use of three advertisement channels 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* ($N_0 \ldots N_i$), as shown in Figure 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 and are used to keep the connection active.

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 Figure 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 exceed 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 by the BLE link layer without any notification to the upper BLE stack layers or the application.

Data channel selection. At the beginning of every connection event, 1 out of 37 possible BLE data channels is selected by the adaptive frequency hopping (AFH) mechanism. A new channel is chosen for every connection event and is used by master and slave to transmit and receive all packets until the end of the ongoing connection event. All 37 possible BLE data channels (0 to 36) are located in the unlicensed 2.4 GHz ISM band. The latter, however, is also used by other wireless communication technologies, such as Wi-Fi, Classic Bluetooth, and IEEE 802.15.4, that may interfere with ongoing BLE communications, leading to link-layer packet loss and re-transmissions. The AFH mechanism 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, implementations of the AFH mechanism 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 by updating the connection's channel map. Both black- and whitelisting of BLE data channels is performed using standardized BLE commands and may only be initiated by a BLE master. Although these commands are standardized, how to measure the link quality of the BLE data channels and when to black- and whitelist individual channels is not specified by the Bluetooth specification. This means that BLE devices are likely to implement the AFH mechanism differently while still being standard-compliant. This may lead to divergent performance of the BLE connection under external radio interference, depending on which BLE radio platform, and therefore AFH implementation is used as 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 Section 3.

Transmission latency. Several models capturing the transmission latency of application data sent over a BLE connection exist [9, 13, 25, 31]. However, only the model proposed by Spörk et al. [31] 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 [31], the *upper bound on transmission latency* of application data sent over a BLE connection on an ideal channel can be computed as:

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

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 [31].

As we show in Section 3, an application cannot rely on this model to compute an upper bound on its end-to-end latency in noisy environments. In our experiments, interference causes several link-layer re-transmissions leading to transmission delays of up to 1,657 ms, which is 537% higher than the maximum expected transmission latency (t_{max} = 260 ms) 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 (Section 3.1). We use a testbed with nine BLE nodes (Section 3.2) to measure the latency of individual data packets in detail. Furthermore, we perform our tests with three popular BLE platforms acting as BLE master to investigate how different implementations of BLE's AFH mechanisms adapt the data channel map over time (Section 3.3). Based on our results, we highlight the specific scenarios in which the tested AFH implementations are unable to cope with the surrounding interference, leading to long delays (Section 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 an `nRF52840` DK [18] node as slave and connect it via IPv6-over-BLE to a Raspberry Pi 3 (RPi3) master [23] that uses its on-board Broadcom BCM43439 radio for BLE communication. Master and slave have a distance of approximately two 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 being notified about the successful reception of the packet from the slave.

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 Equation (1) with $conn\_int$ = 250 ms, an application would expect a maximum transmission latency t_{max} = 260 ms for each UDP packet.

Figure 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 populated with employees, up to 21.74% of the UDP packets sent within 15 minutes experience a transmission latency higher than t_{max}. Several packets even experienced a latency above 1000 ms, i.e., four times higher than t_{max}. During nighttime, instead, when the office is at its quietest, only a minimal number of packets exhibit 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, despite the use of the AFH

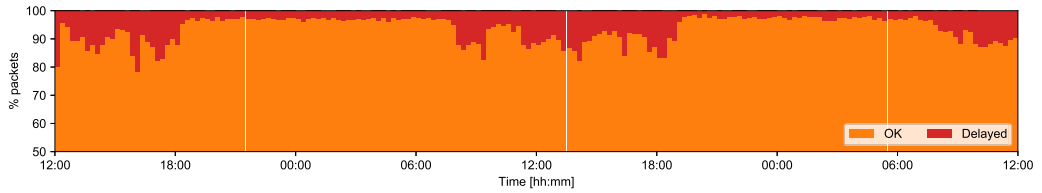


Fig. 2. Percentage of data packets exceeding t_{max} across 48 hours in a common office environment when using a Raspberry Pi 3 with its on-board Broadcom BCM43439 radio as BLE master. During daytime, up to 21.8% of the transmitted packets are delayed due to surrounding interference.

mechanism. 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

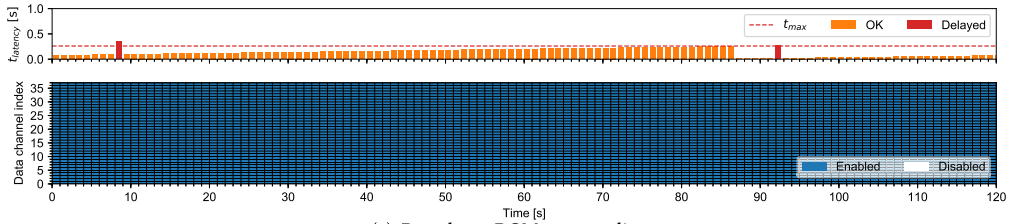
We perform our experiments on our testbed facility, which allows us to have fine-grained control over the RF noise experienced by each BLE node in our testbed.

Testbed facility. The testbed consists of nine RPi3 equally distributed over a University lab (6×10 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 also augmented with the open-source D-Cube board [26, 27], which allows us to accurately measure the power consumption of each `nRF52840` DK device over time.

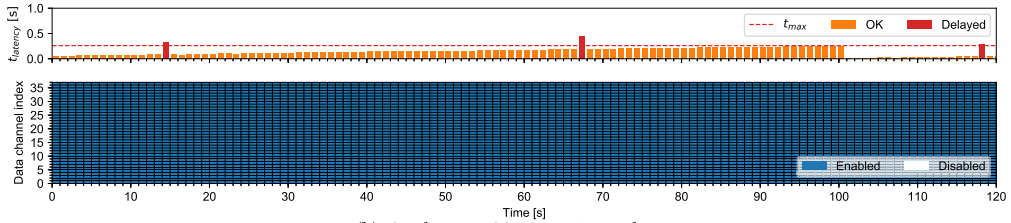
Generating interference. All RPi3 in the testbed are used to re-program the BLE nodes and to monitor the status of each experiment by logging data in persistent memory. We further use the RPi3s in the testbed to generate Bluetooth and Wi-Fi interference using their on-board Broadcom BCM43439 radio chip [23]. 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 an RFCOMM data rate of 725 kbits/s. To create Wi-Fi interference, we let each RPi3 generate IEEE 802.11 b packets of configurable length and configurable rate on a given Wi-Fi channel and with a transmission power of 30 mW using JamLab-NG, an open-source tool to generate repeatable and reproducible Wi-Fi interference [28].

BLE master. We use one of the RPi3 as BLE master for all our tests. In addition to the nRF52840 DK node, this RPi3 is connected to three additional BLE devices: (i) the RPi3's on-board Broadcom BCM43439 radio [23], (ii) a Qualcomm CSR8510 A10 USB-BLE dongle [22], and (iii) a Panasonic PAN1762 USB-BLE dongle [19]. For every experiment, the RPi3 selects one of these three connected radios to 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, its payload and reception time are logged locally via the serial interface of the node.

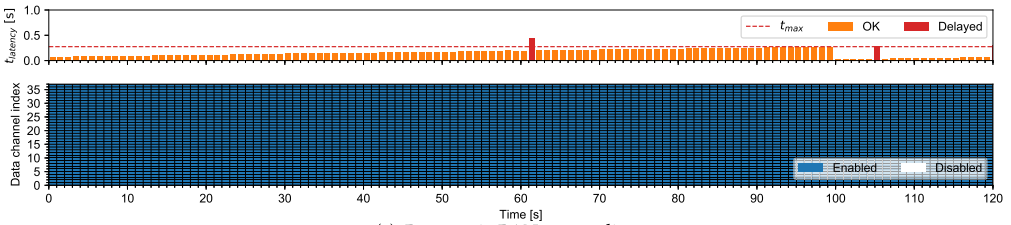
BLE slave. We use each of the `nRF52840` devices (except the one connected to the RPi3 acting as a master) as BLE slave. Each slave waits for the BLE master to initiate an IPv6-over-BLE connection and sends a UDP message to the master every second once the connection is established. Each UDP message has a length of 29 bytes (resulting in a BLE link-layer packet length of 80 bytes) and carries an eight-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



(a) Broadcom BCM43439 radio.



(b) Qualcomm CSR8510 A10 radio.



(c) Panasonic PAN1762 radio.

Fig. 3. Packet latency ($t_{latency}$) and data channel map of a BLE connection under *no external interference* for three different BLE radios used at the Raspberry Pi 3 acting as master.

slave application sits on top of the Zephyr OS [33] and uses its existing IPv6-over-BLE stack. Note that only one slave is used in an experiment to avoid self-interference.

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 ($t_{latency}$) of individual data packets sent from slave to master. For every experiment, the RPi3 uses one of its three BLE radios. Both master and slave make use of $conn\_int = 250\,ms$, $F = 128\,bytes$, and $t_{CE} = 10\,ms$. As discussed in Section 3.1, we expect the upper bound on each transmission t_{max} to be 260 ms (see Equation (1)).

3.3.1 No External Interference. We first analyze the latency of data transmissions in the presence of no external interference in our testbed. We measure the packet latency ($t_{latency}$) of every data transmission as the time difference between the slave issuing the send command and the master being notified about the successful packet reception, as described in Section 3.1.

Each plot in Figure 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 30 seconds for the system to be stable before we start to analyze the data transmission latencies (time 0 in Figure 3).

Our results show that every data packet is successfully transmitted, but some transmissions occasionally exceed the threshold $t_{max} = 260\,ms$, even though no external radio interference is being generated artificially. These delays are likely caused by packet loss resulting from multipath

Table 1. Performance of the Three BLE Radios Acting as a BLE Master under *No External Interference*

Radio	DELAYED [%]	AVG [ms]	MED [ms]	90% [ms]	99% [ms]	MAX [ms]
Broadcom	6.33	158.8	165.0	248.0	467.0	629.0
CSR	2.82	150.7	153.0	233.0	400.0	501.0
Panasonic	10.14	173.2	179.0	261.0	419.0	519.0

The table shows the percentage of delayed packets (DELAYED), the average (AVG), median (MED), 90 and 99 percentile (90% and 99%), and maximum experienced transmission delay (MAX) over 10 test runs per radio.

fading in our testbed or beaconing activities of nearby Wi-Fi access points. As Figure 3 shows, these delays occur for each of the three BLE platforms used. Table 1 summarizes the results obtained after performing this experiment 10 times for each employed BLE platform. We can observe that, depending on the used BLE master radio, a fraction of the data transmissions ($\leq$10%) exceed t_{max}.

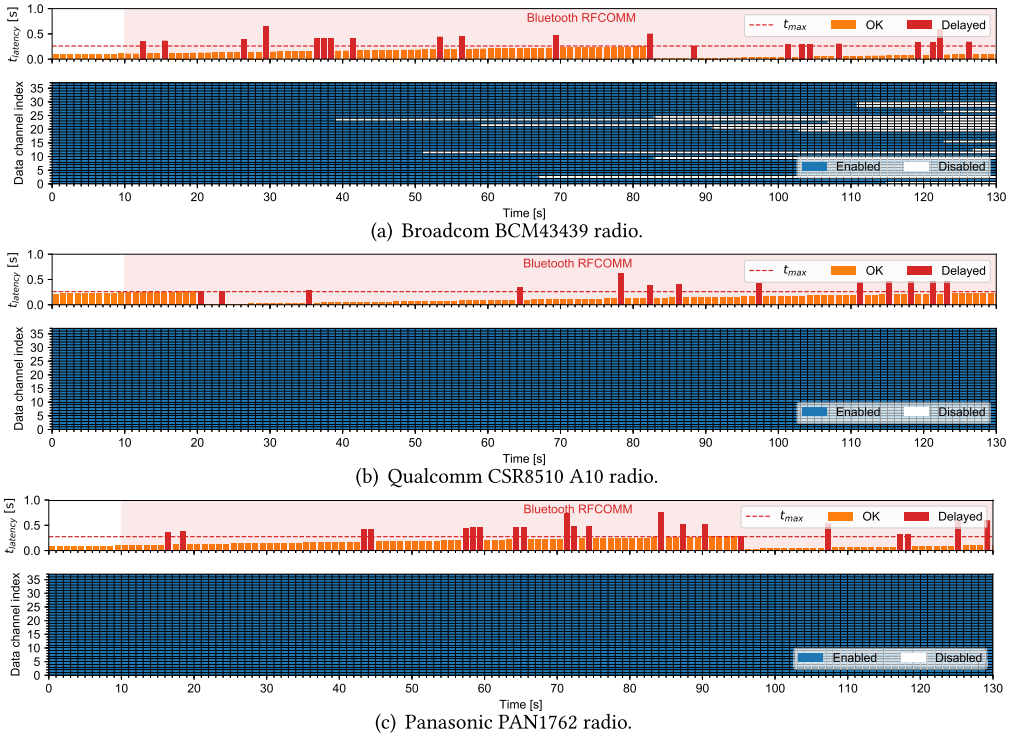
Furthermore, we see that non-delayed data transmissions (marked as OK in Figure 3) experience a $t_{latency}$ between t_{CE} and t_{max}. This is caused by the unsynchronized schedules of BLE application and BLE connection. As discussed in Section 2, an application can issue a data transmission at any time, but the data will actually be sent during the next upcoming connection event. In our experiments, the application issues data transmissions slightly faster than the schedule of the BLE connection. This causes the time between the application issuing and the BLE connection actually transmitting a packet (shown as t_F in Figures 9 and 10) to rise, which results in a linearly increasing $t_{latency}$. When the time between issuing and transmitting a packet gets higher than t_{max}, the packet is sent one connection event earlier, which results in a $t_{latency}$ of approximately t_{CE} in such a case.

3.3.2 Bluetooth Interference. Next, we 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 Section 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. Similar to the experiments in Section 3.3.1, we 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.

Each plot in Figure 4 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. Again, packets exceeding $t_{max} = 260\,ms$ (shown as horizontal dashed line) are marked as *delayed*. After initializing the IPv6-over-BLE connection, we wait 30 seconds for the system to be stable before we simultaneously start interfering on all three Bluetooth connections (time 10 in Figure 4).

Our results show that every UDP packet is, *eventually*, successfully received. However, Bluetooth interference causes between 10% and 15% of all transmissions to be delayed, as shown in Table 2.

Furthermore, Figure 4(a) shows that the AFH implementation of the Broadcom BCM43439 is trying to update the data channel map to mitigate the effect of the Bluetooth interference on the BLE connection. However, the master is not able to accurately predict the frequencies used by Bluetooth and its blacklisting strategy does not help in mitigating the impact of Bluetooth interference on the BLE connection. The Qualcomm CSR8510 A10 radio only occasionally updates the data channel map, because of the nearby Bluetooth interference, leading to 1.55% more packets being delayed compared to the Broadcom radio. Figure 4(b) shows a test run where the Qualcomm radio does not update its data channel map under Bluetooth interference, the observed behavior of this platform in our tests. The Panasonic PAN1762, shown in Figure 4(c), does not update the



(a) Broadcom BCM43439 radio.

(b) Qualcomm CSR8510 A10 radio.

(c) Panasonic PAN1762 radio.

Fig. 4. Packet latency ($t_{latency}$) and data channel map of a BLE connection under *Bluetooth RFCOMM interference* for three different BLE radios acting as a BLE master.

Table 2. Performance of the Three BLE Radios Acting as a BLE Master under *Bluetooth RFCOMM Interference*

Radio	DELAYED [%]	AVG [ms]	MED [ms]	90% [ms]	99% [ms]	MAX [ms]
Broadcom	10.62	166.9	163.0	265.0	502.0	732.0
CSR	12.17	160.8	147.0	266.0	510.0	825.0
Panasonic	14.88	184.3	179.0	271.0	519.0	755.0

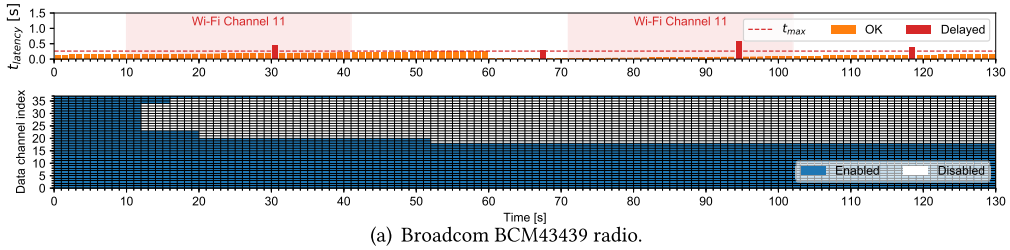
The table shows the percentage of delayed packets (DELAYED), the average (AVG), median (MED), 90 and 99 percentile (90% and 99%), and maximum experienced transmission delay (MAX) over 10 test runs per radio.

channel map, which leads to almost 15% of all data transmissions being delayed, as summarized in Table 2.

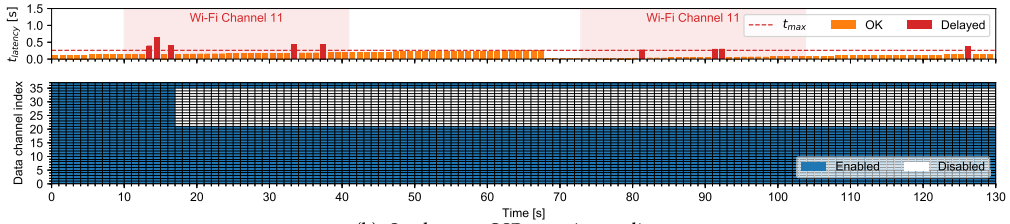
Note that the same effect shown in Figure 4 is experienced by every BLE slave in our testbed, even those that are only three meters away from the master and have direct line-of-sight.

3.3.3 Wi-Fi Interference. We investigate next the impact of Wi-Fi interference on a BLE connection. Following the setup described in Section 3.2, we generate Wi-Fi interference near master and slave.

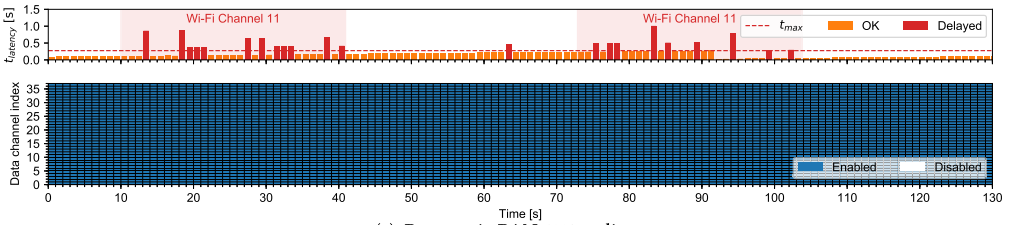
Wi-Fi interference near the master. The plots in Figure 5 show the measured $t_{latency}$ (top) and used data channel map (bottom) of a BLE connection in the presence of Wi-Fi interference located near the master. Also in this case, master and slave are at a distance of 10 meters with direct line-of-sight. After an initial delay of 30 seconds to let the IPv6-over-BLE connection set up,



(a) Broadcom BCM43439 radio.



(b) Qualcomm CSR8510 A10 radio.



(c) Panasonic PAN1762 radio.

Fig. 5. Packet latency ($t_{latency}$) and data channel map of a BLE connection under *Wi-Fi interference near the master* for three different BLE radios acting as a BLE master.

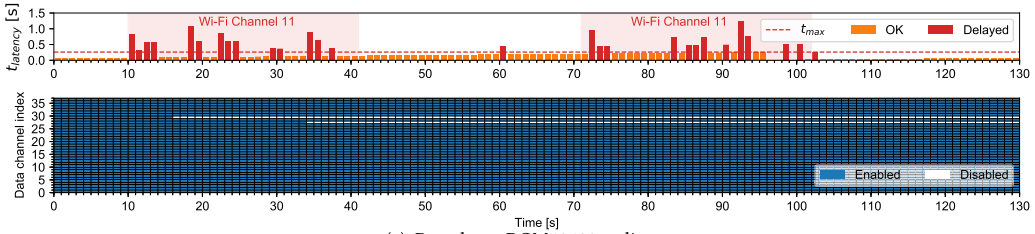
Table 3. Performance of the Three BLE Radios Acting as a BLE Master under *Wi-Fi Interference Near the Master*

Device	DELAYED [%]	AVG [ms]	MED [ms]	90% [ms]	99% [ms]	MAX [ms]
Broadcom	12.42	175.9	171.0	361.0	629.0	881.0
CSR	8.09	155.1	144.0	253.0	483.0	949.0
Panasonic	27.22	247.2	205.0	502.0	1,011.0	1,657.0

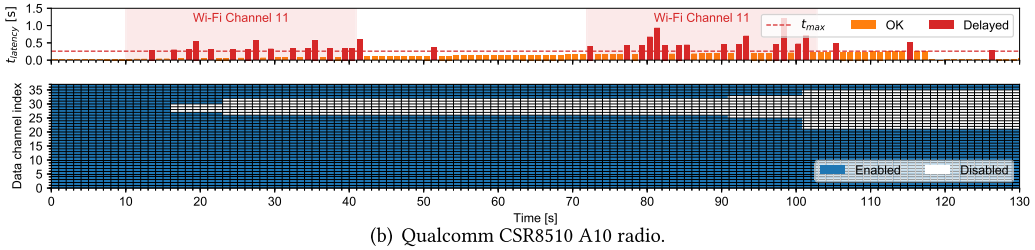
The table shows the percentage of delayed packets (DELAYED), the average (AVG), median (MED), 90 and 99 percentile (90% and 99%), and maximum experienced transmission delay (MAX) over 10 test runs per radio.

we let an RPi3 near the BLE master generate Wi-Fi traffic on channel 11 in bursts of 30 seconds (time 10 to 41 s). We pause the Wi-Fi interference for 30 seconds before starting to interfere again for approximately 30 seconds (time from 70 to 100 s). This inference pattern mimics a rate-limited Wi-Fi device downloading two large files from the Internet, with a pause between the two files.

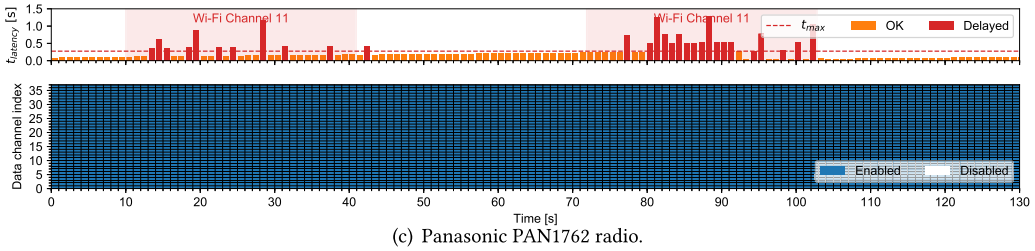
Similar to the previous experiments, our results show that every UDP packet is *eventually* received. We further see that the AFH implementations of the Broadcom and Qualcomm radios, shown in Figure 5(a) and Figure 5(b), respectively, are successfully able to detect the Wi-Fi interference and mitigate its effects on the BLE connection. Despite the heavy traffic generated on Wi-Fi channel 11, both BLE radios blacklist the affected BLE data channels (18 to 31) as soon as they detect their poor link quality. As Table 3 shows, this results in only 13% and 8% of all transmissions being delayed for the Broadcom BCM43439 and Qualcomm CSR8510 A10, respectively.



(a) Broadcom BCM43439 radio.



(b) Qualcomm CSR8510 A10 radio.



(c) Panasonic PAN1762 radio.

Fig. 6. Packet latency ($t_{latency}$) and data channel map of a BLE connection under *Wi-Fi interference near the slave* for three different BLE radios acting as a BLE master.

The AFH implementation of the Panasonic PAN1762, however, does not seem to update the data channel map according to the experienced Wi-Fi interference and is therefore not able to mitigate its effect on the BLE connection. This leads to 27% of transmissions being delayed and a maximum delay of 1657 ms, which is approximately 6.4 times t_{max}.

Wi-Fi interference near the slave. Using the above setup, we now let the RPi3 near the slave generate the same Wi-Fi pattern. The plots in Figure 6 show 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 Figure 5, this time all three AFH implementations are mostly 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 $t_{latency} \geq 6 \cdot t_{max}$, independent from the used BLE radio.

As Figure 6(a) shows, the Broadcom BCM43439 radio does not effectively detect the Wi-Fi interference and therefore only blacklists a subset of the BLE data channels affected by Wi-Fi. The Qualcomm CSR8510 A10 is able to successfully blacklist more of the BLE data channels experiencing Wi-Fi interference[1] after a longer delay of approximately 90 seconds between the beginning

[1]Our measurements suggests that the Broadcom BCM43439 and the Qualcomm CSR8510 A10 use the signal-to-noise value of each BLE data channel to estimate its link quality. Further, our data indicate that the Qualcomm radio is using a more sensitive signal-to-noise threshold and therefore blacklists BLE data channels more aggressively, as shown in Figure 6.

Table 4. Performance of the Three BLE Radios Acting as a BLE Master under *Wi-Fi Interference Near the Slave*

Device	DELAYED [%]	AVG [ms]	MED [ms]	90% [ms]	99% [ms]	MAX [ms]
Broadcom	28.7	289.5	236.0	923.0	1871.0	1921.0
CSR	26.7	255.6	198.0	578.0	1114.0	1168.0
Panasonic	30.2	307.4	208.0	661.0	1496.0	1624.0

The table shows the percentage of delayed packets (DELAYED), the average (AVG), median (MED), 90 and 99 percentile (90% and 99%), and maximum experienced transmission delay (MAX) over 10 test runs per radio.

of Wi-Fi interference and the data channel map being updated, as shown at time 91 in Figure 6(b). This delayed data channel map adaptation, however, does not significantly improve the performance of the Qualcomm radio in this scenario. Similar to the previous experiments, the Panasonic PAN1762 does not update BLE data channel map under Wi-Fi interference near the slave, as shown in Figure 6(c).

As Table 4 summarizes, between 26% and 30% of all data transmissions are delayed in this experiment with maximum transmission delays between 1,168 ms and 1,921 ms. All three BLE radios used as master fail to mitigate the effect of Wi-Fi interference near the slave on the BLE connection.

The reason for this outcome lies in the inability of the BLE radios to effectively detect the Wi-Fi interference and the lack of a subsequent data channel map update. 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 Section 2.

3.4 Lessons Learned

Our experiments show that, regardless of the platform used, 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 Reference [30]. Although no data packet is lost, however, we have observed that the transmission latency significantly increases under interference, even up to a value of almost two seconds, i.e., eight times t_{max}, as shown in Table 4.

Inefficiency of AFH implementations. In particular, our experiments highlight that, in *two* situations, the implementations of the AFH algorithm used by the three tested BLE radio platforms are unable to cope with surrounding interference, leading to long delays. First, the AFH mechanism 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 mostly unable to efficiently detect the RF noise and mitigate its effects by updating the list of blacklisted channels. We expect this to be the case also for surrounding networks making use of channel hopping (e.g., networks based on TSCH).

In all 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]. To get any information about transmission latencies, a BLE application needs to explicitly monitor the connection, e.g., using application acknowledgments.

Diversity of BLE radios. Furthermore, the three used BLE radios, each having its specific—yet standard-compliant—implementation of the AFH mechanism, behave differently under the tested interference scenarios. On the one hand, the Broadcom BCM43439 and Qualcomm CSR8510 A10 platforms are able to effectively and rapidly mitigate the effects of Wi-Fi interference near the

master on the BLE connection, as shown in Section 3.3. Indeed, in this scenario, the Broadcom and Qualcomm radios are both able to sustain a rate of delayed packets below 12.5%. On the other hand, the Panasonic PAN1762 radio never updates the BLE data channel map in all of our experimental settings shown above. This leads to a significantly higher percentage of delayed packets, between 10% and 31% in our four interference scenarios, when using the Panasonic PAN1762 compared to the other two BLE radio platforms. Such high rates of delayed packets, however, may be unacceptable for real-time applications with stringent bounds on data transmission latencies [3, 4, 10, 12].

This diverse behavior of various BLE radios makes it almost impossible to statically select suitable BLE communication parameters depending on the application's latency requirements. During development of a BLE slave application, for example, a developer needs to first predict the noise in the RF environment of the future application to choose the right connection parameters, which is often not possible. Furthermore, the AFH behavior of the BLE master, to which the slave will connect to, needs to be anticipated to choose connection settings that are able to sustain the maximum latency. Failing to select suitable connection parameters will likely result in several transmissions with increased latencies and calls for runtime adaptation.

Need for runtime adaptation. 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. This, however, only *hints* to an application whether there is a need to adjust its connection parameters (e.g., select a lower connection interval to decrease the latency), but not *how* these parameters should be modified. Without a model supporting this decision, an application can only try to significantly lower the connection interval (at the cost of a higher energy expenditure) or slightly lower the connection interval (preserving its energy budget, but at the risk of suffering poor performance).

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. To adapt the connection interval to the BLE host, however, models such as References [9] and [25] are not usable, because they require information that is only available on the BLE controller. A model needs to only use information available on BLE hosts to be usable by most BLE applications.

4 MEASURING AND MODELING BLE LATENCY

In this section, we first revise the model shown in Equation (1) to capture the n_{CE}, i.e., the number of connection events necessary to complete a successful data transmission (Section 4.1). After discussing the unavailability of link-layer information on standard-compliant BLE host devices (Section 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 (Section 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

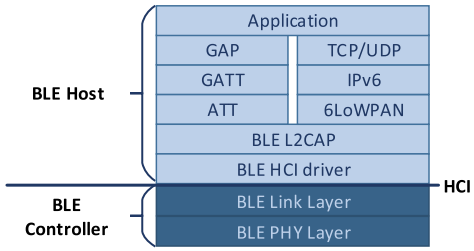


Fig. 7. Standard BLE and IPv6-over-BLE stack.

the timing information of commands sent over the standardized *Host Controller Interface* between host processor and BLE controller (Section 4.4).

4.1 Revising the BLE Timeliness Model

We start by revising the timeliness model from Reference [31] shown in Equation (1). The latter describes how an application data packet of length D (bytes) is split into data fragments with a maximum size of F (bytes), where each fragment is transmitted during a separate connection event. As discussed in Reference [31], the model relies on ideal channel conditions and neglects the effects of link-layer packet loss on the transmission delay of the individual fragments.

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:

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

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.

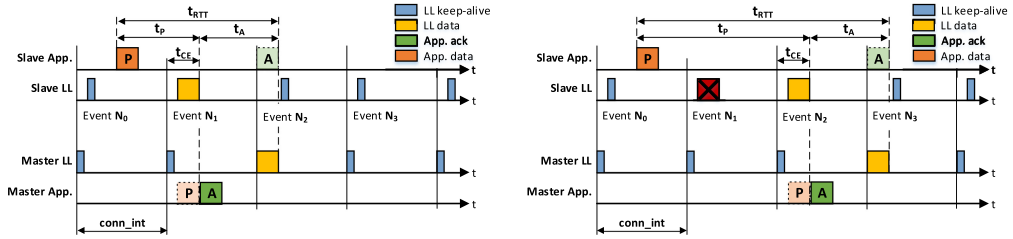
By knowing the n_{CE} of *each* fragment, Equation (2) now captures the impact of RF noise on the quality of a BLE connection and is hence able to provide an upper bound on transmission delay. This, however, requires a precise n_{CE} measurement.

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 Figure 7). 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 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 mechanism. 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, like 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. The BLE HCI driver provides all upper host layers with the functionality to interact with the BLE controller by



(a) No link-layer errors ($n_{CE_f} = 1$). (b) Error during the link-layer data transmission ($n_{CE_f} = 2$).

Fig. 8. RTT-based n_{CE} estimation for a slave transmitting a data packet (P) and receiving an ACK (A).

exchanging HCI commands and events. In contrast to the closed BLE controller, the open-source host provides access to all the upper BLE stack layers, allowing developers to extend the controller to add new functionality that may be needed.

Due to the nature of the BLE stack, several challenges arise when measuring n_{CE} on a host:

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 it 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 available number of reception and transmission buffers in the controller and their individual buffer length.

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, including the blacklisting of data channels with poor link quality. The BLE host, however, has no control or information over the data channel used in the current or the 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 BLE 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 Figure 8.

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. Figure 8 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 Equation (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 an n_{CE_f} that is the same for all fragments involved in the data exchange (data packet and ACK). For our model, we assume that data packet and ACK have the same length D and can hence 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:

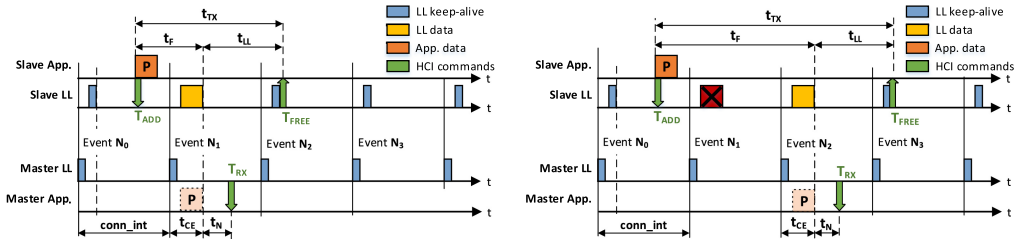
$$n_{CE_f} = \left\lceil \frac{t_{RTT} - 2 \cdot t_{CE}}{2 \cdot \lceil D/F \rceil \cdot conn\_int} \right\rceil. \tag{3}$$

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 (to generate the ACK and to measure the round-trip time). This may not necessarily be the case, for example, when a slave acting as IPv6-over-BLE node transmits IPv6 messages to an IPv6-over-BLE router (master). Although a developer could force a round-trip time measurement using L2CAP ping messages as in References [14, 15], using `RTT-based` n_{CE} estimation might not be suitable for energy-constrained slaves that need to limit the overall communication overhead. 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 Section 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. For example, data consisting of three segments is sent and fragments 1, 2 experience an n_{CE} of 1 and fragment 3 an n_{CE} of 4. This approach estimates an n_{CE} of 2 and therefore overestimates the quality of the BLE connection.

4.4 Estimating n_{CE} Using HCI Timing Info

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. Unlike `RTT-based` n_{CE} estimation, which calculates a single n_{CE} value over the whole data transmission of D bytes, `HCI-based` n_{CE} estimates the n_{CE} of every *individual* data fragment sent during the data exchange.



(a) No link-layer errors ($n_{CE_f} = 1$). (b) Error during the link-layer data transmission ($n_{CE_f} = 2$).

Fig. 9. HCI-based n_{CE} estimation for a BLE slave transmitting a packet (P) consisting of one data fragment.

4.4.1 Estimating n_{CE} on a BLE Slave. Figure 9 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 Figure 8, one can notice that the master is no longer sending application-layer ACKs after receiving a packet. Figure 9 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 (Figure 9(a)) and in the presence (Figure 9(b)) of link-layer errors, the only available timing information that can be derived by the slave via the HCI is the time t_{TX} elapsed between the data packet 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 Figure 9, 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, 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 Equation (2) by setting $D = F$ as:

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

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 $t_{LL} = conn\_int$. In cases where the data are successfully transmitted but the link-layer acknowledgment is interfered, HCI-n_{CE} overestimates the n_{CE} value of the packet transmission. With this assumption, we can calculate t_{TX} (using Equation (5)) as:

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

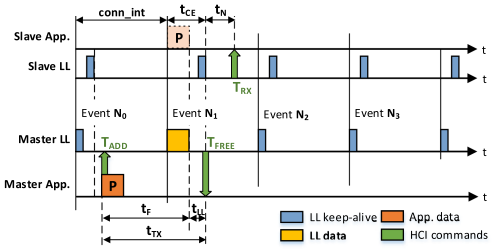


Fig. 10. HCI-based n_{CE} estimation for a BLE master transmitting a single data fragment ($n_{CE_f} = 1$).

A BLE application using the HCI communication to measure t_{TX} (using Equation (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 Section 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 Figure 10. Therefore, t_{LL} is already captured by the maximum connection event length t_{CE} value in Equation (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:

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

Compared to Equation (8), Equation (10) does not need to account for the delayed link-layer acknowledgment received by the BLE slave (modeled by decreasing n_{CE_f} by 1 in Equation (8)).

We now have described two approaches, RTT-based n_{CE} and HCI-based n_{CE}, that allow BLE applications to estimate the timeliness of their communication in a standard-compliant way and do not need any link-layer information limited to the BLE controller. 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 [18]. The latter embeds an ARM Cortex-M4F application processor, an nRF52840 chip with 1,024 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, including the nRF52832 with 512 kB of flash and 64 kB of RAM or even the nRF52810 with 192 kB of flash and 24 kB of RAM.

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.

We use the Zephyr operating system [33] for implementing the BLE slave using our estimation approaches. The Zephyr OS used for our implementation already includes a BLE communication

stack (including IPv6-over-BLE support) that is fully compliant to the BLE specification. Furthermore, the Zephyr BLE stack on the nRF52 platform uses the standardized HCI to exchange information between the BLE controller and the BLE host.

For our work, we focus on estimating the n_{CE} on BLE slave devices, which are usually much more constrained in their energy budget and processing power than BLE masters. By showing that a constrained slave is able to accurately estimate n_{CE} values, we also show that the more powerful BLE master is able to do so. Furthermore, the latter receives link-layer acknowledgments for data transmissions within the same connection event, as discussed in Section 4.4. This means that a master is able to estimate n_{CE} more accurately, because the master's link-layer receives feedback almost immediately after the data was successfully sent and does not need to wait for the link-layer ACK until the next connection event, which may also be interfered, leading to an overestimation of n_{CE}.

5.1 RTT-based n_{CE} Estimation

We start by implementing the RTT-based n_{CE} estimation approach in the slave application described in Section 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 Equation (3).

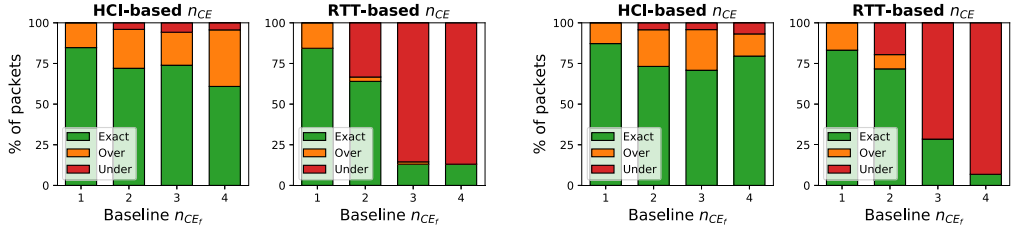
5.2 HCI-based n_{CE} Estimation

We next implement HCI-based n_{CE} estimation reusing the slave application from Section 3.2 and adding the n_{CE} measurements to the BLE host in the HCI driver layer (`hci_core`). Every time the host sends an `HCI ACL Data Packet` command to the BLE controller 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 are measured in milliseconds.

The n_{CE} estimation is performed in the HCI driver layer on the host using Equation (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 platform, 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 estimation accuracy (Section 6.1) and energy efficiency (Section 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 due to the delayed link-layer ACK, as discussed in Section 4. In the experiments of this section, we configure the BLE slave running on an nRF52 device to use one estimation approach at a time and connect it to a Pi 3 master using its on-board Broadcom BCM43439 radio for communication.



(a) Connection interval = 62.5 ms (b) Connection interval = 125 ms

Fig. 11. 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 Section 3.2. We use the Broadcom BCM43439 radio as BLE master, 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:

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

Note that, as described in Section 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 10 times for each setting.

Figure 11 plots the percentage of UDP packets for which the n_{CE_f} has been correctly estimated, overestimated, or underestimated (marked in 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 (baseline $n_{CE_f} \geq 2$).

Overall, HCI-based n_{CE} estimation outperforms the RTT-based one by 0.42, 8.06, 60.87, and 47.82% for an n_{CE_f} of 1, 2, 3, and 4, respectively, in estimating the exact n_{CE_f} value ($conn\_int$ = 62.5 ms). Furthermore, Figure 11 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 an 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 (Figure 11(b)). 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 notification delays introduced by the OS on the master (shown as t_N in Figure 9).

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 Section 3.2. We measure the power consumption of an nRF52840 slave using the D-Cube board [27].

Figure 12 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

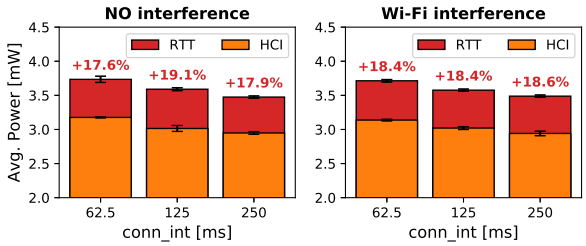


Fig. 12. Average power consumption of n_{CE} estimators for different connection intervals and interference.

an extra 18% power consumption on the slave. This higher power consumption is due to the additional exchange of application-level acknowledgments, which is unnecessary for `HCI-based` n_{CE} estimation.

When comparing the two n_{CE} estimation approaches, we can clearly see that the `HCI-based` n_{CE} estimation is (i) more accurate (see Figure 11) and (ii) more power-efficient (see Figure 12) than the `RTT-based` n_{CE} estimation approach. Other existing BLE link-quality estimation approaches [14, 15] use application-level round-trip-time measurements and work similar to our `RTT-based` n_{CE} estimation approach, i.e., experience the same increased power consumption and less accurate n_{CE} values compared to `HCI-based` n_{CE} estimation. For this reason, we will use only the `HCI-based` n_{CE} estimation approach to increase BLE timeliness for the remainder of this work.

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 to mitigate the presence of interference while minimizing energy consumption (Section 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 (Section 7.2).

7.1 Adapting the BLE Connection at Runtime

Following Equation (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 Equation (2), we can 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 Equation (11), the slave application can request a new connection interval from the master.[2] $conn\_int_{max}$ represents the *most energy-efficient* connection interval to be used 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.

[2]To this end, the slave can use the standardized `L2CAP CONNECTION PARAMETER UPDATE REQUEST`. A master may change the connection interval by issuing an `LL CONNECTION UPDATE` command. According to the BLE specification, there is a 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 future data transmissions do not exceed the maximum transmission delay t_{max}. To achieve this goal, we use a 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 communication parameters that minimize the number of packets exceeding an upper latency bound [16]. 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 Section 4.

As we show in Section 8.3, our simple and aggressive filtering approach is able to efficiently and accurately detect changes in the RF environment and triggers a BLE connection parameter adaptation accordingly. By using such a simple filtering approach, we can run our prediction mechanism even on platforms with a very constrained energy budget and limited processing capabilities. Using a more complex filtering approach, such as linear regression, may provide similar or slightly more accurate n_{CE_f} predictions at the cost of additional processing overhead, leading to an increased power consumption, which is undesirable for constrained IoT devices.

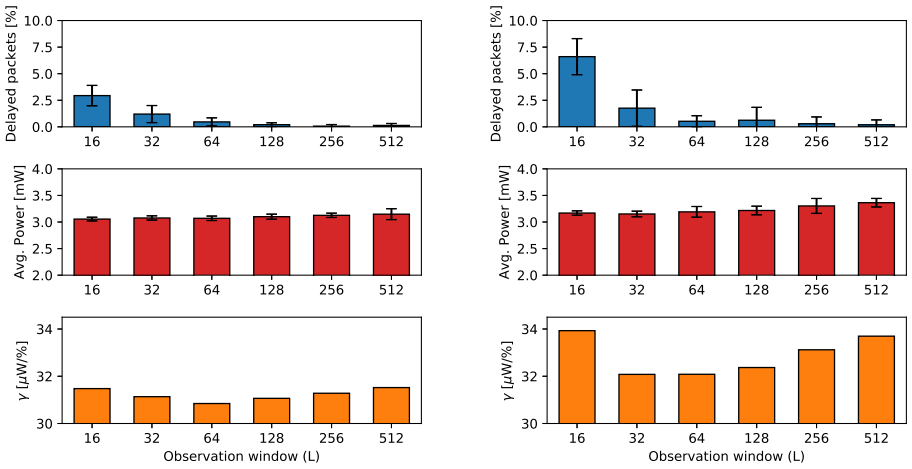
Finding an optimal *L*. We next experimentally investigate a suitable observation window length L. We consider six different lengths (16, 32, 64, 128, 256, and 512) and compute $n_{CE_f\star}$ according to Equation (12). We then instruct a BLE slave to adapt its connection interval according to Equation (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 Section 3.2, i.e., a slave and a master (using the Broadcom BCM43439 BLE platform) 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, the limited information about the amount of interference affecting the channel in the recent past translates to 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.

Figure 13 shows the results of our evaluation. As expected, the percentage of delayed packets decreases for larger observation windows, while 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. Figure 13 (bottom) shows that selecting $L = 64$ offers a good trade-off between energy-efficiency and timeliness of BLE transmissions under both Bluetooth (Figure 13(a)) and Wi-Fi (Figure 13(b)) interference. With this setting, only 0.6% of all data transmissions exceed the latency bound, even under Wi-Fi interference.

8 EVALUATING BLE TIMELINESS IN NOISY RF ENVIRONMENTS WHEN USING n_{CE}

In this section, we evaluate the adaptive scheme that we proposed in Section 7 in terms of the percentage of data packets exceeding the maximum packet latency. First, we systematically compare



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

Fig. 13. Percent of delayed packets (top), average power consumption (middle), and power cost (bottom) for different observation windows under Bluetooth and Wi-Fi interference.

a BLE slave using the proposed adaptation approach to a BLE slave using no connection parameter adaptation (Section 8.1). Second, we evaluate the adaptation approach in two different environments over 48 hours using three different BLE platforms (Section 8.2). Third, we investigate in detail how the adaptation approach reacts dynamically to changes in the RF environment (Section 8.3).

8.1 Systematic Evaluation

We compare the performance of a slave S_{fixed} running an application using a fixed connection interval to that of a slave S_{adapt} employing the adaptation approach proposed in this article. S_{fixed} selects its connection interval statically according to Equation (1) to sustain a maximum transmission delay $t_{max} = 260ms$. S_{adapt}, instead, makes use of Equation (11) and an observation window length $L = 64$ to adapt its connection interval as described in Section 7.

Setup. Using the same setup described in Section 3.2, we let each of the two slaves transmit 500 UDP packets to a master (using the Broadcom BCM43439 radio) located at 10 meters' distance with direct line-of-sight. We run only one slave at a time and repeat each experiment 10 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. Figure 14 shows the percentage of delayed packets and the average power consumption of S_{fixed} (orange) and S_{adapt} (red). We can clearly see that, while 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. This increased energy consumption is caused by using shorter connection interval settings during periods of high RF noise, which lead to a higher power draw.

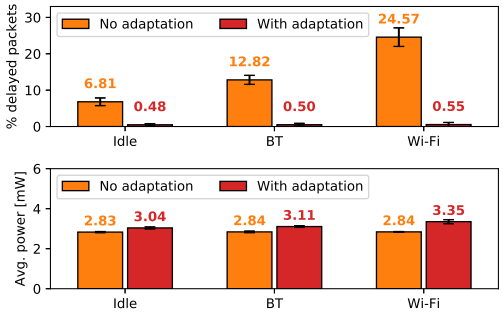


Fig. 14. Delayed packets and power consumption of a slave with and without connection interval adaptation.

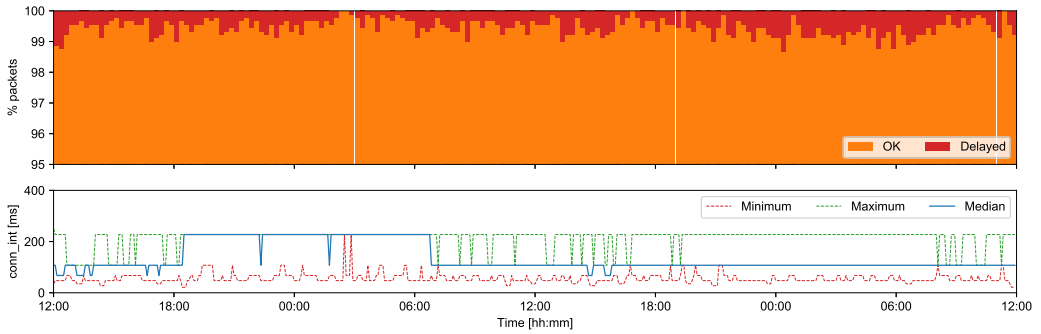


Fig. 15. When adapting its connection interval at runtime using an observation window of 64 fragments, a slave connected to a Broadcom BCM43439 master is able to significantly increase the timeliness of its BLE communications. Note the more granular y-axis of the top plot in this figure compared to Figure 2.

8.2 Long-term Evaluation

To prove the efficacy of our proposed method, we run the same application described in Section 3.1 in different indoor environments on three different BLE platforms.

8.2.1 Common Office Environments. We start by re-running the application in the office environment shown in Section 3.1 populated with employees and use the same location of the nodes. We configure the BLE slave to adapt its connection interval at runtime as described in Section 7 and make use of an observation window of length $L = 64$ and an upper latency bound $t_{max} = 260\ ms$.

Figure 15 shows the number of delayed packets and the adaptation of the connection interval at runtime across 48 hours. It is quite remarkable how at most 1.34% of the UDP packets sent within 15 minutes exceed the maximum latency bounds. In Figure 2, the number of delayed packets was up to 21.74%. The average number of packets delayed in this experiment is 0.54% (compared to an average of 6.18% obtained in Figure 2 when using no adaptation of BLE connection parameters). This shows an improvement of a factor of 11.5 for the average number of packets delayed.

8.2.2 Student Laboratory. Next, we run our experiments in a student laboratory (as described in Section 3.2) and compare the performance of a slave S_{fixed} using a fixed connection interval to a slave S_{adapt} using the proposed adaptation approach. Similar to the systematic evaluation in Section 8.1, both S_{fixed} and S_{adapt} try to sustain a maximum latency bound $t_{max} = 260\ ms$ for their data transmissions. While S_{fixed} uses Equation (1) to select a fixed connection interval, S_{adapt} uses Equation (11) and a window length $L = 64$ to adapt its connection interval at runtime as described in Section 7.

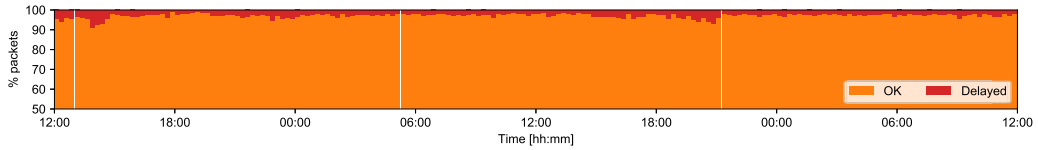


Fig. 16. Percentage of packets exceeding t_{max} across 48 hours in a student laboratory when *using a fixed connection interval* and a *Broadcom BCM43439 radio* as BLE master.

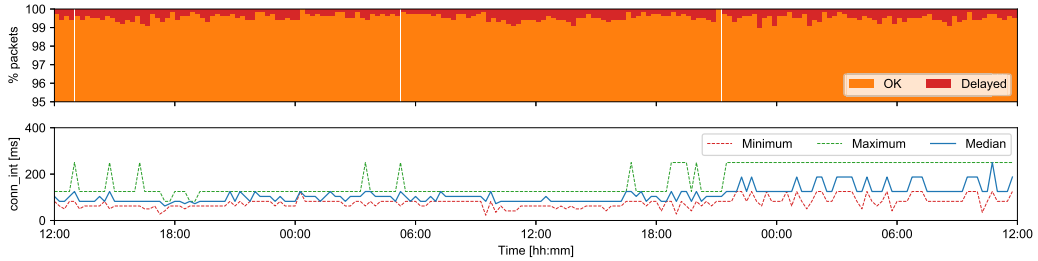


Fig. 17. Percentage of delayed packets (top) and adapted connection interval *conn_int* (bottom) over 48 hours in a student lab when using our *adaptive approach* and a *Broadcom BCM43439* as BLE master. Please note the more granular y-axis of the top plot in this figure compared to Figure 16.

For these experiments, however, we concurrently run S_{fixed} and S_{adapt} in our student laboratory over the same 48-hour periods and connect each slave to a separate master. To ensure that both slaves experience similar changes in the RF environment, we position both slaves next to each other and put both masters at approximately the same location. Each slave has a distance of approximately 10 meters and direct line-of-sight to their BLE master. As both BLE connections use all 37 data channels and exchange data infrequently (sending a packet every second), the interference from one BLE connection on the other is insignificant compared to the RF noise present in the student lab.

In contrast to the experiments shown in Section 3, the lab is used by different student groups during our experiments. We repeat this experiment for all three BLE platforms connected to our masters.

Broadcom BCM43439 radio. For our first long-term lab experiment, each slave connects to one of the Broadcom BCM43439 radios acting as master. Figure 16 shows the percentage of delayed packets of S_{fixed} over 48 hours. Over this period, on average, 2.7% of all transmissions exceed t_{max}, with a maximum of 8.65% of packets being delayed within a 15-minute period. Figure 17 shows the percentage of delayed packets (top) and the used BLE connection interval (bottom) over the same two-day period. We see that the connection interval is successfully adapted, resulting in an average of 0.42% of all packets and a maximum of 0.99% of packets within 15 minutes being delayed.

Qualcomm CSR8510 A10 radio. Next, we perform the same experiment, but connect each slave to a Qualcomm CSR8510 A10 radio acting as master. Figure 18 shows the percentage of delayed data transmissions when using S_{fixed}. We see that overall 7.23% of all packets are classified as delayed, with a maximum of 14.32% packets being delayed within a 15-minute period. The reason for this higher number of delayed packets, compared to the previous experiment, is that the lab was used by students (and their Wi-Fi devices) during daytime.

Nevertheless, we see that the slave using our adaptation approach (shown in Figure 19) is able to cope with this increased RF noise. Over the same period, only 0.62% of all packets and a maximum of 1.22% packets within a 15-minute period are delayed when using our approach.

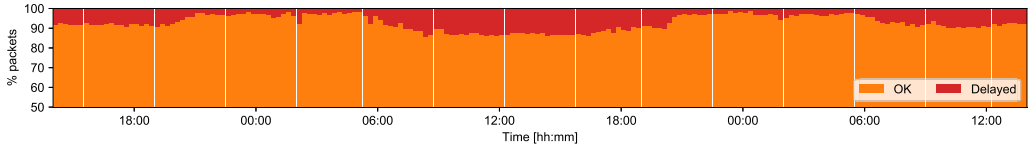


Fig. 18. Percentage of packets exceeding t_{max} across 48 hours in a student laboratory when *using a fixed connection interval* and a *Qualcomm CSR8510 A10 radio* as BLE master.

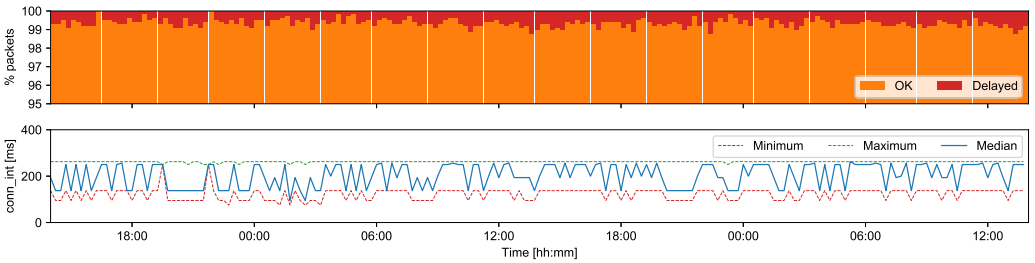


Fig. 19. Percentage of delayed packets (top) and adapted connection interval *conn_int* (bottom) over 48 hours in a student lab when using our *adaptive approach* and a *Qualcomm CSR8510 A10* as BLE master. Please note the more granular y-axis of the top plot in this figure compared to Figure 18.

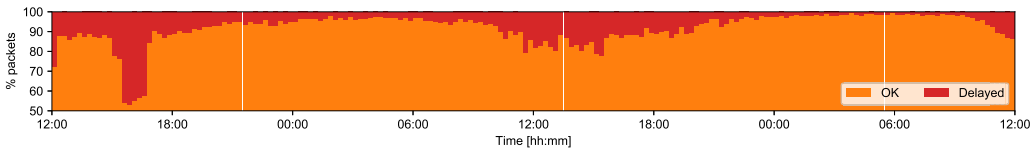


Fig. 20. Percentage of packets exceeding t_{max} across 48 hours in a student laboratory when *using a fixed connection interval* and a *Panasonic PAN1762 radio* as BLE master.

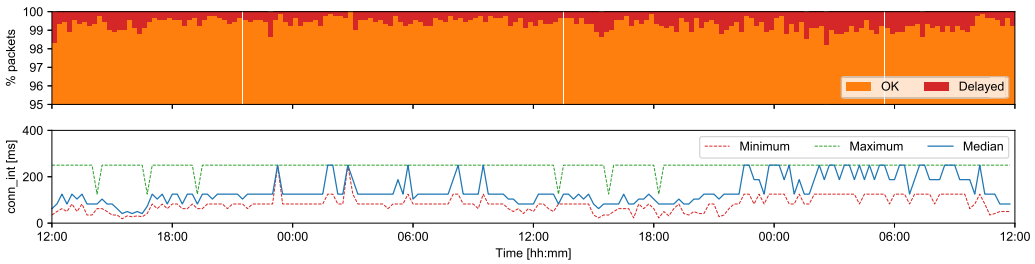
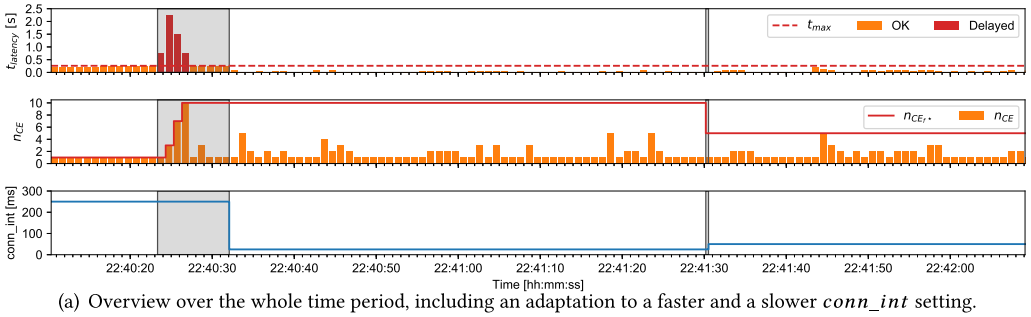
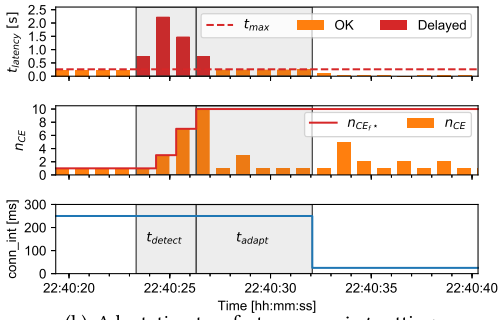


Fig. 21. Percentage of delayed packets (top) and adapted connection interval *conn_int* (bottom) over 48 hours in a student lab when using our *adaptive approach* and a *Panasonic PAN1762* as BLE master. Please note the more granular y-axis of the top plot in this figure compared to Figure 20.

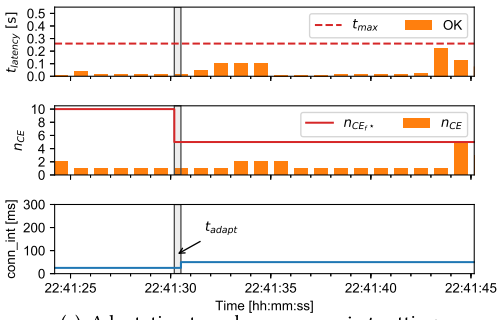
Panasonic PAN1762 radio. We repeat the experiment using two Panasonic PAN1762 radios as BLE masters. Figure 20 shows the percentage of delayed packets of S_{fixed} over two days, where the student lab was extensively used during daytime. Overall, 8.09% of all packets exceed t_{max} with a maximum of 46.67% of delayed transmissions in a 15-minute period. This high rate of transmission delays is caused by (i) an increased student activity during the test period and (ii) the fact that the Panasonic PAN1762 did not adapt the BLE channel map to changes in RF noise during this test.



(a) Overview over the whole time period, including an adaptation to a faster and a slower $conn\_int$ setting.



(b) Adaptation to a faster $conn\_int$ setting.



(c) Adaptation to a slower $conn\_int$ setting.

Fig. 22. Transmissions latency ($t_{latency}$), their corresponding n_{CE} estimations, and the adapted connection interval ($conn\_int$) during a heavy and sudden change in RF noise using a Broadcom BCM43439 master.

Nonetheless, Figure 21 shows that, when using our adaptive approach, only 0.65% of the total data transmissions and at most 1.77% of the packets sent during 15 minutes are classified as delayed.

Overall, our experiments show that our adaptation approach significantly improves the timeliness of BLE connections independently of the used BLE platform and the type of co-located RF noise.

8.3 Dynamic Behavior

To investigate the dynamic behavior of our adaptation approach, we use the long-term measurements from Section 8.2.2 and evaluate how long it takes to (i) detect RF noise changes using `HCI-based` n_{CE} estimation and (ii) to adapt the connection interval to these changes in the RF environment.

Figure 22(a) shows an exemplary time period, during which the RF noise suddenly changes due to a co-located Wi-Fi device transmitting data. This leads to multiple long packet transmission delays (marked as red bars) at time 22:40:23. Figure 22(b) shows this specific time period in more detail and highlights two phases of our n_{CE} estimation and parameter adaptation approach: (i) the time necessary to detect that the link quality has changed (t_{detect}) and (ii) the time needed to adapt the connection interval of the BLE connection to this change (t_{adapt}).

t_{detect} measures the time difference between the instant of time in which a data transmission is issued and the instant in which the corresponding BLE connection update request is sent by the slave. We see in Figure 22(b) that the n_{CE} value increases in steps over time until reaching the actual n_{CE} value of 10. This behavior can be explained by the implementation of our n_{CE} estimation (see Section 5) and parameter adaptation (see Section 7), where before every new data packet transmission, our application reads the most recent n_{CE} value from our `HCI-based` n_{CE} estimator.

Table 5. Measured Timing Values for t_{detect} and t_{adapt} during the Long-term Tests from Section 8.2.2

	Broadcom BCM43439			Qualcomm CSR8510 A10			Panasonic PAN1762		
Timings	**AVG**	**90%**	**MAX**	**AVG**	**90%**	**MAX**	**AVG**	**90%**	**MAX**
t_{detect} $[ms]$	1,007.9	986.0	4,977.0	1,184.8	987.0	4,987.0	1,116.5	986.0	4,979.0
t_{adapt} $[ms]$	2,092.4	2,853.0	6,208.0	1,927.1	2,107.4	5,117.0	3,469.2	4,948.0	5,226.0

The table shows the average (AVG), 90 percentile (90%), and maximum (MAX) measured timing value in milliseconds for the three different BLE radio platforms used over 48 hours.

At time 22:40:24, the application adds the next data packet to the transmission buffer, detects that the previous packet experienced an n_{CE} of at least 3, and, therefore, issues a BLE connection parameter update request to adapt the connection interval. This step is repeated two additional times, until the initial delayed packet (issued at approximately 22:40:23) is successfully transmitted and its actual n_{CE} value becomes available. Note that every subsequent BLE connection parameter update request overrides the previous ones. In the example in Figure 22(a), the detection phase takes $t_{detect} = 2{,}983\,ms$.

t_{adapt} measures the time between issuing the latest BLE connection update request and the new BLE parameters actually being used. As described in Section 7.2, the BLE specification requires a delay of at least six connection events between the slave receiving the new connection interval and the latter being used. In this example, adapting the connection parameters takes $t_{adapt} = 5{,}758\,ms$. The reason for this long adaptation time is that the slave can only request new parameters, but the master may ignore this request. Therefore, the application repeatedly requests new parameters until the master approves, which may lead to t_{adapt} greater than six times the connection interval.

After successfully handling the initial burst of RF noise, the subsequent data transmissions experience an n_{CE} of at most 5. Therefore, a new BLE connection parameter request is issued at 22:41:30 (approximately L fragments after the initial burst) and the connection interval is updated accordingly, as shown in Figure 22(c). In this case, the filtering of our adaptation approach immediately detects the change in link quality ($t_{detect} = 0$) and the adaptation takes $t_{adapt} = 329\,ms$. This makes adapting to a slower connection interval much faster, as during this adaptation a fast connection interval is used and, therefore, the mandatory delay of at least six connection events is shorter.

Table 5 summarizes the timing values of t_{detect} and t_{adapt} measured in the experiments performed in Section 8.2.2. As the data show, detecting a change in the RF environment takes on average between 1,007.9 and 1,184.8 ms, with a maximum duration of about 4,987.0 ms. The subsequent parameter adaptation is performed within 1,927.1 and 3,469.2 ms on average and takes at most 6,208.0 ms.

During our 48-hour test, at most four subsequent data transmissions were delayed when using the Broadcom BCM43439 or the Qualcomm CSR8510 A10 as BLE master radios. When using the Panasonic PAN1762, a maximum of 13 subsequent data transmissions were delayed.

9 RELATED WORK

Several studies have investigated the performance of low-power wireless technologies under interference [2, 20]. 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, 34, 36] or of *BLE connections* [9, 13, 25, 31]. Only few works actually measure the performance of BLE under interference experimentally [17, 30, 35]. These

studies, however, lack practicality, as they are performed in a small anechoic chamber [30] or artificially constrain the performance of BLE's AFH by disabling channel blacklisting [17, 35].

In this article, 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 article, we also develop the first model capturing the timeliness of connection-based BLE communications in noisy environments *that can be used on BLE host devices*. 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, 25]. 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, 36] or assume *perfect channel* conditions [7, 31].

Estimating BLE link quality. A few works have investigated how to estimate the link quality of BLE connections. Lee et al. [15] use round-trip time measurements of periodic L2CAP ping messages on Linux-based devices to capture the link quality of a BLE connection and dynamically change the routing topology of RPL over BLE. The authors estimate the connections' link quality every 10 seconds, which they show to be a suitable period to detect changes in the routing topology (i.e., selecting a new parent) and conclude that BLE is reliable due to its AFH mechanism (even under Wi-Fi interference). Lee et al. [14] investigate the energy consumption and the stability of a BLE connection in environments with variable link quality (e.g., due to the presence or absence of line-of-sight caused by doors opening and closing). The authors measure the round-trip time of frequent L2CAP ping messages and adapt the connection interval to keep the connection alive (not triggering the BLE supervision timeout) while minimizing energy consumption.

Differently from these works, we estimate the link quality of a BLE connection without introducing any additional communication overhead. We further use our link quality estimation scheme to dynamically adapt the connection parameters of a BLE connection and provide an upper latency bound on individual data packet transmissions.

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 unpredictable 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 improvement of the performance of BLE's adaptive frequency hopping mechanism and its channel blacklisting under interference. This, however, requires control over the inner-workings of the BLE controller.

ACKNOWLEDGMENTS

We thank Usman Raza and Toshiba Research Europe Limited for providing us with the Panasonic PAN1762 platforms used in our experiments.

REFERENCES

[1] N. Amanquah and J. Dunlop. 2003. Improved throughput by interference avoidance in co-located Bluetooth networks. In *Proceedings of the 5th European Personal Mobile Communications Conference (EPMCC'03)*.

[2] N. Baccour, A. Koubâa, L. Mottola, M. A. Zúñiga, H. Youssef, C. A. Boano, and M. Alves. 2012. Radio link quality estimation in wireless sensor networks: A survey. *ACM Trans. Sens. Netw.* 8, 4 (2012).

[3] A. K. Bhattacharjee, D. Bruneo, S. Distefano, F. Longo, G. Merlino, and A. Puliafito. 2017. Extending Bluetooth Low Energy PANs to smart city scenarios. In *Proceedings of the Conference on Smart Computing (SMARTCOMP'17)*.

[4] BLE Home. 2017. iAlert Sensing Motion: Quick Start Guide. Retrieved from http://www.blehome.com/ialert.htm.

[5] Bluetooth SIG. 2018. Bluetooth Core Specification v5.0. Retrieved from https://www.bluetooth.com/specifications/bluetooth-core-specification.

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

[7] K. Cho, W. Park, M. Hong, G. Park, W. Cho, J. Seo, and K. Han. 2014. Analysis of latency performance of Bluetooth Low Energy (BLE) networks. *Sensors* 15, 1 (2014).

[8] M. Collotta and G. Pau. 2015. 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. 2016. A channel access scheme for Bluetooth Low Energy to support delay-sensitive applications. In *Proceedings of the 27th Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC'16)*.

[10] B. Islam, M. Uddin, S. Mukherjee, and S. Nirjon. 2018. Rethinking ranging of unmodified BLE peripherals in smart city infrastructure. In *Proceedings of the 9th ACM Multimedia Systems Conference (MMSys'18)*.

[11] W. S. Jeon, M. H. Dwijaksara, and D. G. Jeong. 2017. Performance analysis of neighbor discovery process in Bluetooth Low Energy networks. *IEEE Trans. Vehic. Technol.* 66, 2 (2017).

[12] H. Karvonen, K. Mikhaylov, M. Hämäläinen, J. Iinatti, and C. Pomalaza-Ráez. 2017. Interference of wireless technologies on BLE based WBANs in hospitals. In *Proceedings of the Symposium on Personal, Indoor, and Mobile Radio Communications*.

[13] P. Kindt, D. Yunge, M. Gopp, and S. Chakraborty. 2015. Adaptive online power-management for Bluetooth Low Energy. In *Proceedings of the Conference on Computer Communications (INFOCOM'15)*.

[14] T. Lee, J. Han, M.-S. Lee, H.-S. Kim, and S. Bahk. 2017. CABLE: Connection interval adaptation for BLE in dynamic wireless environments. In *Proceedings of the 14th IEEE Conference on Sensing, Communication, and Networking (SECON'17)*.

[15] T. Lee, M.-S. Lee, H.-S. Kim, and Saewoong S. Bahk. 2016. A synergistic architecture for RPL over BLE. In *Proceedings of the 13th IEEE Conference on Sensing, Communication, and Networking (SECON'16)*.

[16] S. Munir, S. Lin, E. Hoque, S. Nirjon, J. Stankovic, and K. Whitehouse. 2010. Addressing burstiness for reliable communication and latency bound generation in wireless sensor networks. In *Proceedings of the 9th ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN'10)*.

[17] R. Natarajan, P. Zand, and M. Nabi. 2016. Analysis of coexistence between IEEE 802.15. 4, BLE and IEEE 802.11 in the 2.4 GHz ISM band. In *Proceedings of the 42nd IEEE Conference of the Industrial Electronics Society (IECON'16)*.

[18] Nordic Semiconductors. 2018. nRF52840 Specifications. Retrieved from https://www.nordicsemi.com/eng/Products/nRF52840.

[19] Panasonic Industrial Devices Europe GmbH. 2019. PAN1762 - Bluetooth 5.0 Low Energy Module. Retrieved from https://pideu.panasonic.de/products/bluetooth/pan1762-bluetooth-50-low-energy-module.html.

[20] M. Petrova, J. Riihijarvi, P. Mahonen, and S. Labella. 2006. Performance study of IEEE 802.15. 4 using measurements and simulations. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC'06)*.

[21] P. Popovski, H. Yomo, S. Guarracino, and R. Prasad. 2004. Adaptive mitigation of self-interference in Bluetooth scatternets. In *Proceedings of the 7th Conference on Wireless Personal Multimedia Communications (WPMC'04)*.

[22] Qualcomm. 2019. CSR8510 Chipset. Retrieved from https://www.qualcomm.com/products/csr8510.

[23] Raspberry Pi Foundation. 2018. Raspberry Pi 3 Model B. Retrieved from https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[24] R. Rondón, M. Gidlund, and K. Landernäs. 2017. Evaluating Bluetooth Low Energy suitability for time-critical industrial IoT applications. *J. Wirel. Inf. Netw.* 24, 3 (2017).

[25] R. Rondón, K. Landernäs, and M. Gidlund. 2016. An analytical model of the effective delay performance for BLE. In *Proceedings of the 27th Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC'16)*.

[26] M. Schuß, C.A. Boano, and K. Römer. 2018. Moving beyond competitions: Extending D-cube to seamlessly benchmark low-power wireless systems. In *Proceedings of the Workshop on Benchmarking Cyber-Physical Networks and Systems*.

[27] M. Schuß, C. A. Boano, M. Weber, and K. Römer. 2017. A competition to push the dependability of low-power wireless protocols to the edge. In *Proceedings of the 14th Conference on Embedded Wireless Systems and Networks (EWSN'17)*.

[28] M. Schuß, C. A. Boano, M. Weber, M. Schulz, M. Hollick, and K. Römer. 2019. JamLab-NG: Benchmarking low-power wireless protocols under controllable and repeatable Wi-Fi interference. In *Proceedings of the 16th Conference on Embedded Wireless Systems and Networks (EWSN'19)*.

[29] Silicon Labs. 2018. Application Development Fundamentals: Bluetooth Smart Technology. Retrieved from https://www.silabs.com/documents/login/user-guides/ug103-14-fundamentals-ble.pdf.

[30] S. Silva, S. Soares, T. Fernandes, A. Valente, and A. Moreira. 2014. Coexistence and interference tests on a Bluetooth Low Energy front-end. In *Proceedings of the Science and Information Conference (SAI'14)*.

[31] M. Spörk, C. A. Boano, M. Zimmerling, and K. Römer. 2017. BLEach: Exploiting the full potential of IPv6 over BLE in constrained embedded IoT devices. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems (SenSys'17)*.

[32] M. Spörk, C. A. Boano, and K. Römer. 2019. Improving the timeliness of Bluetooth Low Energy in noisy RF environments. In *Proceedings of the 16th Conference on Embedded Wireless Systems and Networks (EWSN'19)*.

[33] The Zephyr Project. 2018. Zephyr OS: An RTOS for IoT. Retrieved from https://www.zephyrproject.org/.

[34] J. Tosi, F. Taffoni, M. Santacatterina, R. Sannino, and D. Formica. 2017. Evaluating Bluetooth Low Energy suitability for time-critical industrial IoT applications. *Sensors* 17, 12 (2017).

[35] J. J. Treurniet, C. Sarkar, R. V. Prasad, and W. de Boer. 2015. Energy consumption and latency in BLE devices under mutual interference: An experimental study. In *Proceedings of the 3rd Conference on Future Internet of Things and Cloud*.

[36] J. Wyffels, J.-P. Goemaere, B. Nauwelaers, and L. De Strycker. 2014. Influence of Bluetooth Low Energy on Wi-Fi communications and vice versa. In *Proceedings of the European Conference on the Use of Modern Information and Communication Technologies (ECUMICT'14)*.

[37] G. Zhou, J. A. Stankovic, and S. H. Son. 2006. Crowded spectrum in wireless sensor networks. In *Proceedings of the 3rd Workshop on Embedded Networked Sensors (EmNets'06)*.

Received July 2019; revised October 2019; accepted November 2019