Dipl.-Ing. Michael Spörk, BSc

Enabling Time-Critical Internet of Things Applications based on Bluetooth Low Energy

DOCTORAL THESIS

to achieve the university degree of

Doktor der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Inform. Dr.sc.ETH Kay Uwe Römer

Institute of Technical Informatics

Co-Supervisor
Assoc.Prof. Dott. Dott. mag. Dr.techn. Carlo Alberto Boano, MSc

Graz, December 2021

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present doctoral thesis.

Date	Signature

Acknowledgements

During my studies, I was fortunate enough to get to know many great colleagues and friends that accompanied and supported me on my academic journey so far. Even though I cannot thank everyone personally here, I hereby want to thank a few special people.

I was fortunate enough to work on this dissertation while being at the Institut of Technical Informatics at Graz University of Technology. Therefore, I first and foremost would like to thank Prof. Kay Römer for this possibility. Thank you very much for the excellent support, great advice, and helpful feedback that I have received from you over the last years. Furthermore, I want to particularly thank my advisor and co-supervisor Assoc.Prof. Carlo Alberto Boano for his active and intense support during my studies. Without your patience and persistence, your advice, and your outstanding support, especially in the days and nights before an approaching paper deadline, this dissertation would probably not exist. Thank you both for your mentoring and support that allowed me to grow academically and personally. I also would like to thank my external examiner, Prof. Christine Julien, for her helpful feedback on this dissertation. It is a great honor to have three such renowned examiners for my doctoral thesis.

During my studies, I had the fortune to collaborate with outstanding people. Thanks a lot to Bernhard Vacarescu, Ko Odreitz, and Stephan Gailer for your wonderful support during teaching. Thanks to my coauthors Dr. Marco Zimmerling, Dr. Jiska Classen, and Prof. Matthias Hollick for the interesting and instructional collaboration. I especially thank our guru Markus Schuß for his technical support. Most of my experiments would not have been possible without his help or his testbed.

A different kind of thanks goes out to my colleagues at the Institut of Technical Informatics, especially the people of the extended ”Zimmer“: Rainer Hofmann, Lukas Gressl, Martin Erb, Alexander Rech, Fikret Basic, Michael Krisper, Thomas Ulz, and Philipp Stelzer. Thanks for the technical and also non-technical hours that I could spend with you so far. I hope there are many more to come. Thanks to Christian Steger, Engelbert Meissl, Florian Dietrich, Georg Macher, Felix Warmer, Markus Feldbacher, Maximilian Schuh, Hannah Brunner, and Elisabeth Salomon for the nice get-togethers in the kitchen and on the balcony.

Last, but not least, I want to thank my family, especially my mother and father. Without you, I would not be the person that I am today and without your support, I am confident that I would not be writing this dissertation. I cannot thank you enough! Finally, I want to thank my dearest Verena, who had no other choice than to experience the day-to-day ups and downs of my PhD studies with me. Thank you very much for lightening up my days with your sunny spirit.

Thank you all for your support in these interesting and challenging times of my life!

Graz, December 2021
Michael Spörk

Danksagung

Während meines Doktoratsstudiums hatte ich das Glück viele großartige Kolleg*innen und Freunde kennenzulernen, die mich auf meiner akademischen Reise begleitet und unterstützt haben. Auch wenn ich hier nicht alle persönlich erwähnen kann, möchte ich mich hierbei bei einigen besonderen Personen bedanken.

Ich hatte das Glück meine Dissertation am Institut für Technische Informatik an der Technischen Universität Graz durchzuführen. Deswegen möchte ich mich zuallererst bei Prof. Kay Römer für diese Möglichkeit bedanken. Danke für die hervorragende Unterstützung, die guten Ratschläge und das hilfreiche Feedback, welches ich in den letzten Jahren bekommen habe. Ich möchte auch besonders meinem Betreuer und Ko-Supervisor Assoc.Prof. Carlo Alberto Boano für seine tatkräftige und intensive Unterstützung danken. Ohne Carlo's Geduld und Ausdauer, seine Betreuung und seine überragende Mithilfe, vor allem in Tagen und Nächten vor Paper Deadlines, würde diese Dissertation wohl nicht existieren. Vielen Dank an Kay Römer und Carlo Alberto Boano für die herausragende Betreuung und Unterstützung, durch die ich mich fachlich und auch persönlich weiterentwickeln konnte. Ich möchte mich auch bei meiner externen Prüferin Prof. Christine Julien für ihr hilfreiches Feedback zu dieser Dissertation bedanken. Es ist mir eine Ehre, dass ich drei solch hochkarätige Prüfer meiner Dissertation habe.

Während meines Doktoratsstudiums hatte ich das Glück mit hervorragenden Personen zusammenzuarbeiten. Herzlichen Dank an Bernhard Vacarescu, Ko Odreitz und Stephan Gailer für die wunderbare Unterstützung bei der Lehre. Vielen Dank auch an meine Koautoren Dr. Marco Zimmerling, Dr. Jiska Classen und Prof. Matthias Hollick für die interessante und lehrreiche Zusammenarbeit. Besonders möchte ich mich noch bei unserem Guru Markus Schuß für seine technische Unterstützung bedanken. Ohne seine Hilfe und sein Testbed wären meine Experimente nicht möglich gewesen.

Ein Dank der anderen Art gilt auch meinen Kolleg*innen am Institut für Technische Informatik, speziell den Personen des erweiterten Zimmers: Rainer Hofmann, Lukas Gressl, Martin Erb, Alexander Rech, Fikret Basic, Michael Krisper, Thomas Ulz und Philipp Stelzer. Vielen Dank für die fachlichen und vor allem nicht-fachlichen Stunden, die ich mit euch bis jetzt verbringen durfte. Ich hoffe es werden noch einige folgen. Danke auch an Christian Steger, Engelbert Meissl, Florian Dietrich, Georg Macher, Felix Warmer, Markus Feldbacher, Maximilian Schuh, Hannah Brunner und Elisabeth Salomon für die netten Runden in der Küche und am Balkon.

Ich möchte mich auch bei meiner Familie, vor allem bei meiner Mutter und meinem Vater, bedanken. Ohne euch wäre ich sicher nicht der Mensch, der ich heute bin, und ohne eure Unterstützung wäre es auch nie dazu gekommen, dass ich gerade an meiner Dissertation schreibe. Ich kann euch nicht genug danken! Speziell möchte ich noch meiner liebsten Verena danken, die keine andere Wahl hatte, als mit mir tagtäglich die Höhen und Tiefen meines Doktoratsstudiums zu durchleben. Danke, dass du meine Tage durch dein sonniges Gemüt erhellt hast.

Vielen Dank an alle für die Unterstützung in dieser interessanten und herausfordernden Zeit!

Graz, Dezember 2021
Michael Spörk

Abstract

Connecting smart devices to the Internet of Things (IoT) enables a wide range of innovative applications, such as industrial monitoring and automation, smart home applications, or smart city deployments. For many IoT applications, smart devices are required to be compact and mobile, which means they need to operate on small batteries and wirelessly exchange data with other devices on the Internet. One of the most popular wireless technologies for such applications is Bluetooth Low Energy (BLE), which provides energy-efficient radio communication and is already widely available in consumer devices.

As BLE technology is further optimized, BLE devices are increasingly used in time- and safety-critical IoT applications, such as industrial control systems or safety-critical healthcare applications. In these applications, BLE devices need to communicate with other IoT devices within given end-to-end reliability and latency bounds while operating on a single battery charge for an extended time period. Unfortunately, devices employed in real-world settings are likely to experience unpredictable packet loss and delay caused by external interference from co-located radio devices, narrow-band fading effects, or effects on the path across the Internet. How BLE devices can meet given end-to-end reliability requirements, such as a maximum delay or a minimum reliability, remains an open question in real-world applications.

In this thesis, we focus on enabling constrained BLE devices to reliably exchange time-critical data with peer devices. We investigate in detail how BLE-based IoT devices are affected by real-world communication issues occurring in the local BLE environment or on the Internet and show how BLE devices can cope with these problems by dynamically adapting their communication parameters to network changes. Towards this goal, we make the following three contributions:

First, we present three novel BLE adaptation mechanisms that significantly improve the performance of connection-based BLE communication. Our BLE channel management passively monitors the quality of used BLE data channels and dynamically adapts the list of used data channels to improve communication reliability. Our BLE PHY mode adaptation dynamically chooses the most suitable BLE PHY mode to sustain a given communication reliability while limiting unnecessary power consumption. Our BLE connection parameter adaptation monitors the transmission delay of BLE data packets and adapts the BLE connection parameters at runtime to allow time-critical data exchange over BLE connections. All three mechanisms are fully compliant to the BLE specification and can be used on off-the-shelf BLE devices.

Second, we present BLEach, the first full-fledged, open-source IPv6-over-BLE communication stack for constrained and low-power devices. We show that low-power BLE devices can use BLEach to directly exchange IPv6 packets with other IPv6 devices on the Internet. Furthermore, we go beyond the IPv6-over-BLE specification and extend BLEach to support multiple IPv6 traffic flows with different QoS classes over a single IPv6-over-BLE connection. This allows BLE devices to dynamically prioritize certain safety-critical IPv6 packets over other IPv6 traffic.

Third, we show how low-power BLE devices are able to efficiently monitor the communication latency and reliability across the whole network path, *i.e.*, from a BLE node to a cloud server. Using our network estimation and novel BLE end-to-end models, devices are successfully able to meet a given end-to-end reliability and latency bound when communicating with a cloud server on the Internet, while limiting their power consumption.

Kurzfassung

Die Anbindung intelligenter Geräte an das Internet der Dinge (IoT) ermöglicht eine breite Palette innovativer Anwendungen, wie z. B. industrielle Überwachungs- und Automatisierungsapplikationen, Smart-Home-Anwendungen oder Smart-City-Lösungen. In vielen dieser Anwendungen müssen die Geräte kompakt und mobil sein, was bedeutet, dass sie mit kleinen Batterien betrieben werden müssen und Daten nur mittels Funkübertragung mit anderen Geräten austauschen können. Eine der beliebtesten Funktechnologien für solche Anwendungen ist Bluetooth Low Energy (BLE), welche eine überaus energieeffiziente Funkkommunikation ermöglicht und in Smartphones und Laptops bereits weit verbreitet ist.

Durch die ständige Verbesserung von BLE finden BLE-Geräte zunehmend Einsatz in zeit- und sicherheitskritischen IoT-Anwendungen, wie z.B. in industriellen Steuerungssystemen oder in sicherheitskritischen Anwendungen im Gesundheitswesen. In solchen Anwendungen müssen BLE-Geräte innerhalb definierter maximaler Ende-zu-Ende-Latenzzeiten und mit definierter minimaler Ende-zu-Ende-Zuverlässigkeit Daten austauschen und gleichzeitig mit einer einzigen Batterieladung über längere Zeiträume auskommen. Leider kommt es bei Geräten, welche in realen Umgebungen eingesetzt werden, immer wieder zu unvorhersehbaren Paketverlusten und -verzögerungen. Der Grund für diese Verluste und Verzögerungen sind meist Störungen durch benachbarte Funkgeräte, physikalische Fading-Effekte oder Probleme im Internetpfad. Aktuell ist unklar, wie BLE-Geräte in realen Anwendungen bestimmte Anforderungen, wie z. B. eine definierte maximale Ende-zu-Ende-Latenzzeit oder eine definierte minimale Ende-zu-Ende-Zuverlässigkeit, erfüllen können.

Diese Dissertation behandelt die Frage, wie batteriebetriebene BLE-Geräte zeitkritische Daten zuverlässig übertragen können. Wir untersuchen im Detail, wie BLE-basierte IoT-Geräte von realen Kommunikationsproblemen betroffen sind, wenn in der lokalen BLE-Umgebung oder im Internet Störungen auftreten. Weiters zeigen wir, wie BLE-Geräte diese Kommunikationsprobleme bewältigen können, indem sie ihre Kommunikationsparameter dynamisch an Netzwerkänderungen anpassen. Um dieses Ziel zu erreichen, leistet diese Dissertation die folgenden drei Beiträge:

Wir entwickeln drei neuartige BLE-Anpassungsmechanismen, welche die Leistung von BLE-Verbindungen deutlich verbessern. Unser BLE-Kanalmanagement überwacht passiv die Qualität der verwendeten BLE-Frequenzen und wählt dynamisch die besten Frequenzen zur Datenübertragung aus, um die Zuverlässigkeit der Kommunikation deutlich zu verbessern. Unsere Anpassung des BLE-PHY-Modus wählt automatisch den geeignetsten BLE-PHY-Modus aus, der eine gegebene Kommunikationszuverlässigkeit unterstützt und gleichzeitig den Stromverbrauch minimiert. Unsere Anpassung der BLE-Verbindungsparameter überwacht die Latenzzeit von BLE-Datenpaketen und passt die BLE-Verbindungsparameter dynamisch an, um zeitkritischen Datenaustausch über BLE-Verbindungen zu ermöglichen. Alle drei Mechanismen sind vollständig konform mit der BLE-Spezifikation und können daher ohne Probleme auf handelsüblichen BLE-Geräten verwendet werden.

Wir präsentieren BLEach, den ersten kompletten, quelloffenen IPv6-over-BLE-Kommunikationsstack für batteriebetriebene Geräte. Wir zeigen weiters, dass batteriebetriebene BLE-Geräte BLEach einfach nutzen können, um IPv6-Pakete direkt mit anderen IPv6-Geräten

im Internet auszutauschen. Zudem erweitern wir BLEach über die IPv6-over-BLE-Spezifikation hinaus, um mehrere IPv6-Verkehrsströme mit unterschiedlichen QoS-Klassen über eine einzige IPv6-over-BLE-Verbindung zu unterstützen. Dadurch ist es BLE-Geräten möglich bestimmte sicherheitskritische IPv6-Pakete dynamisch gegenüber anderem IPv6-Verkehr zu priorisieren.

Wir zeigen, wie batteriebetriebene BLE-Geräte in der Lage sind, die Kommunikationslatenz und -zuverlässigkeit über den gesamten Netzwerkpfad, also von BLE-Gerät bis Cloud-Server, effizient zu überwachen. Mit dieser Technik und unseren neuartigen BLE-Modellen sind Geräte erstmals erfolgreich in der Lage, bei der Kommunikation mit einem Cloud-Server im Internet eine bestimmte Ende-zu-Ende-Zuverlässigkeits- und Latenzgrenze einzuhalten und gleichzeitig ihren Stromverbrauch zu minimieren.

Contents

List of Figures

List of Tables

List of Abbreviations

6LoWPAN	IPv6 over Low-Power Wireless Personal Area Networks
ACK	acknowledgment
AFH	Adaptive Frequency Hopping
AP	access point
ARQ	Adaptive Retransmission Request
ATT	Attribute Protocol
AWS	Amazon Web Service
BLE	Bluetooth Low Energy
CCA	clear channel assessment
CHIP	Connected Home over IP
CRC	cyclic redundancy check
CSA	channel selection algorithm
DBCA	Dynamic Bandwidth Channel Access
DetNet	Deterministic Networking
DNS	Domain Name System
ECG	electrocardiogram
ETX	expected transmission count
EWMA	exponentially weighted moving average
FEC	forward error correction
GAP	Generic Access Profile
GATT	Generic Attribute Profile
LAN	Local Area Network
L2CAP	Logical Link Control and Adaptation Protocol
LQI	link quality indicator
MAC	medium access control
ML	machine learning
NACK	negative acknowledgment
NB	narrow band
NOMA	nonorthogonal multiple access
NTP	Network Time Protocol
HCI	Host Controller Interface
ICU	intensive care units
IETF	Internet Engineering Task Force

IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISM	industrial, scientific and medical
ISO	isochronous
OFDMA	orthogonal frequency-division multiple access
OS	Operating System
PDR	packet delivery ratio
Pi3	Raspberry Pi 3B
PHY	physical layer
QoS	Quality of Service
RDC	radio duty cycle
RQ	research question
RSSI	received signal strength indicator
RTP	Real-Time Transport Protocol
RTT	round-trip time
SDN	Software-Defined Networking
SIG	Special Interest Group
SM	Security Manager
SNR	signal-to-noise ratio
SCO	Synchronous Connection Oriented
SoC	system on chip
TCP	Transmission Control Protocol
TSN	Time Sensitive Network
URLLC	ultra-reliable low-latency communication

1 Introduction

Over the last decade, the Internet of Things (IoT) has been one of the key enabling technologies for creating a wide range of smart and innovative applications, where vast numbers of smart "things" connect the physical world to the Internet [222]. For example, using IoT technology for industrial monitoring and automation, manufacturers are able to improve plant productivity, decrease manufacturing costs, and minimize production waste. IoT devices in smart healthcare and assisted living applications enable preventive medicine, real-time detection of emergencies, and more accurate patient care. Smart home, smart city, and smart grid applications create sustainable living environments by optimizing resource allocation while improving their inhabitants' quality of life. With these social benefits and an estimated net profit of up to $14.4 trillion between 2013 and 2022 created by IoT applications [218], it is not surprising that the number of devices connected to the IoT is steadily increasing and is expected to hit 75 billion by 2025 [212].

Many IoT applications, such as smart health and fitness trackers [59], smart home products [160], or smart agriculture applications [198], require that IoT devices are cost-efficient, compact, and mobile. This means that so-called IoT nodes have constrained memory and processing resources, need to operate on small batteries, and require wireless communication capabilities [222]. One of the most popular technologies to connect low-power nodes wirelessly to the Internet is Bluetooth Low Energy (BLE), which fits all the requirements above. BLE devices have a small form factor and are widely and cheaply available. BLE devices are also very energy efficient, outperforming Wi-Fi and other low-power wireless technologies such as IEEE 802.15.4 or ANT [55, 72, 190], and may operate on single coin cell batteries for multiple years. In contrast to cellular technologies, BLE communication uses the popular 2.4 GHz ISM (industrial, scientific and medical) frequency band, which allows BLE devices to operate globally without requiring any license fees [24]. Moreover, BLE is already widely available in most consumer electronic devices, such as wearables, smartphones, tablets, and laptops, which means that BLE-based nodes can directly interact with users via their existing devices and can use these devices to connect to the Internet [192].

These advantages make BLE the technology of choice for a wide range of different IoT applications, where IoT nodes need to reliably exchange messages with other devices while operating on small batteries for a prolonged period. Smart grid and smart city applications use BLE devices to improve the energy efficiency of communities [49], track appliances [100], or monitor water quality [83]. Smart healthcare solutions use BLE to monitor vital signs of patients [84, 93] or to locate doctors, nurses, and medical equipment [112]. Smart home and office applications use BLE technology for access control [160, 175], smart lighting solutions [199], or building condition monitoring [13, 191]. Moreover, Project Connected Home over IP (CHIP), a consortium of well-known companies like Google, Amazon, Apple, and IKEA, specifies BLE as one of the standard

wireless technologies for future smart home solutions [235].

As the number of existing BLE devices is steadily increasing and has already reached over 8 billion in 2020 [211], the BLE specification is continuously improved and extended to allow an even wider range of applications using BLE communication, like high-fidelity audio streaming [192].

1.1 Time- & Safety-critical IoT Applications based on BLE

As research on BLE technology advances and BLE's performance is increasingly optimized, more and more BLE-based IoT nodes operate in application domains where they are required to reliably exchange data packets with peer devices within given latency bounds while operating on a limited power supply. Some of these time- and safety-critical IoT applications encompass, but are not limited to, the following application domains:

Professional sports tracking. BLE devices can be embedded into professional sports equipment to allow accurate training progress tracking, data-based referee decisions, or injury detection in real-time. In football or hockey games, for example, BLE devices embedded in the ball or puck and worn by individual players allow to track their individual location in real-time during a game [71]. BLE-enabled equipment allows to track fine-grained statistics during training or competition and almost real-time data annotation in the cloud [186, 208]. Coaches have instant access to their teams' statistics to continuously track performance or quickly shift strategy and athletes may even use instant feedback to improve performance during competition. Besides real-time data tracking, BLE-enabled equipment is also used to prevent or immediately detect sports injuries, such as concussions in American Football [194].

Industrial monitoring and control systems. Industrial monitoring applications use BLE technology in use cases where devices need to reliably transfer time-critical data to a central server. For example, BLE nodes continuously monitor critical processes, *e.g.*, to immediately detect leaks of hazardous fluids, or to perform predictive maintenance of important equipment, *e.g.*, to power it down before it breaks [97]. BLE-based structural monitoring devices, such as fire detection systems, operate for years on small batteries while still reliably transmitting critical alarm messages to a server within seconds to improve building safety [37]. Furthermore, BLE is used to coordinate human-machine interaction in industrial settings, such as autonomous vehicles sharing the warehouse floor with workers [119].

Safety-critical healthcare. Healthcare applications use BLE-enabled tracking devices to continuously monitor patients to identify emergencies and alarm medical personnel. Assisted living facilities employ BLE-enabled wearables to measure daily activity and detect falls of elderly or disabled people [3, 171]. Whenever a fall or another emergency event occurs, the BLE device immediately issues an alarm message to call for help. BLE-based sensors are used in even more safety-critical medical environments, such as hospital beds or intensive care units (ICU)s, to continuously monitor patients' vital signs [93]. In case critical metrics, such as blood oxygen, blood pressure, or electrocardiogram (ECG) readings, indicate a medical emergency, the monitoring BLE device notifies the medical staff to prevent life-threatening situations.

On the one hand, it is important for all the above applications that BLE nodes are able to exchange data packets with a server within hard end-to-end delay and reliability bounds. For example, industrial monitoring systems require an end-to-end communication reliability above 99.9%

and end-to-end communication latencies below 50 ms [132]. Similarly, smart healthcare applications demand an end-to-end reliability above 99% and end-to-end latency bounds of multiple seconds [93]. More safety-critical healthcare applications, such as monitoring patients in ICUs, require an end-to-end reliability of 99.9% and a maximum end-to-end latency of 100 ms [5]. If data packets between node and server are unpredictably lost or exceed the specified maximum delay, costly equipment may be damaged, undetected injuries may cause long-term health problems, or even lives may be lost. On the other hand, it is also important that the battery-powered, wireless nodes used in these applications are able to successfully operate for a given period before being recharged or replaced. If a critical event occurs, but the node has run out of battery, similar harms as above may occur. As the wireless communication of a node accounts for a significant portion of its energy consumption, BLE devices cannot simply use the fastest, and therefore most energy-consuming, BLE communication parameters [55]. Instead, battery-powered BLE devices in time-critical application domains need to find suitable BLE parameters that allow to sustain time-critical data exchange while limiting the devices' power consumption.

1.2 Problem Statement

In this thesis, we focus on enabling resource- and energy-constrained BLE devices to reliably exchange time-critical data with peer devices in the local BLE network and on the Internet. Towards this goal, we investigate how BLE-based IoT devices are affected by and can cope with real-world communication issues occurring in the local BLE environment or over the Internet.

BLE communication in real-world environments. Low-power wireless technologies such as BLE may experience packet loss and delays when deployed in real-world settings [15]. First, BLE transmissions may be interfered by co-located wireless devices that also use the popular 2.4 GHz ISM frequency band, such as Wi-Fi, IEEE 802.15.4, or Classic Bluetooth [30, 31]. Such external radio interference causes BLE transmission to be lost and, therefore, significantly impacts key performance metrics of BLE communication, such as energy efficiency, data throughput, and transmission latency. Second, BLE communication may also be affected by multipath fading, where narrow-band radio signals are reflected by nearby obstacles (*e.g.*, walls or persons) causing destructive self-interference, *i.e.*, a receiver does not detect or cannot decode any signal from the transmitter [226]. Similar to external interference, multipath fading also causes packet loss, which decreases the performance of BLE communication. Third, BLE uses a relatively low transmission power, the default transmission power of BLE devices is 1 mW, to decrease the energy consumption of transmitting devices and to comply with ISM band regulations [24]. Although this leads to very energy-efficient communication, it may cause problems when BLE devices need to communicate over long distances, as packets with a low received signal strength cannot be successfully decoded at the receiver [15].

To cope with these problems, the BLE specification foresees multiple mechanisms that BLE devices may use at runtime to tune the performance of an active BLE connection, namely:

BLE channel management. BLE connections make use of Adaptive Frequency Hopping (AFH) to limit the negative effects of individual BLE data channels that have poor quality. To exclude poor-performing channels from being used for data exchange and hence mitigate the problems due to poor data channels even further, the BLE specification allows BLE devices to adaptively change the list of channels used for frequency hopping at runtime.

BLE PHY mode adaptation. BLE devices can choose one out of four possible physical layer (PHY) modes to exchange data with a peer device. By adapting the used PHY mode of a BLE connection at runtime, BLE devices can trade a longer communication range and high robustness (due to symbol coding and forward error correction) for a higher data rate (caused by a higher physical symbol modulation).

BLE connection parameter adaptation. The timing of data exchange over BLE connections is defined by multiple BLE connection parameters. BLE devices can choose and dynamically change these connection parameters due to changes in the BLE network or updated application requirements. For example, BLE devices may use fast connection parameters when a low transmission latency is required or may use energy-efficient connection parameters to conserve energy.

Techniques related to these three BLE mechanisms have been shown to significantly improve communication performance of other wireless technologies. For example, adaptive frequency hopping techniques have been shown to effectively mitigate the effects of external radio interference and multipath fading in other low-power wireless technologies [225, 226]. Dynamically changing the used PHY mode to changing environmental conditions is an effective technique to sustain given application requirements, as shown for LTE [123], Wi-Fi [125], LoRa [197], and UWB [88]. Furthermore, adapting communication timing has already been successfully used by other low-power radio devices to sustain specific application needs [102,236]. Unfortunately, the BLE specification does neither define nor indicate how these mechanisms should be used to improve the performance of BLE communication. Manufacturers of BLE devices may implement their own proprietary solutions of these mechanisms or may choose not to use these mechanisms at all. To make things even more complicated, the low-level communication behavior of a BLE device is usually implemented by a dedicated BLE radio peripheral, which autonomously handles communication timing and buffer management. These BLE radios act as "black boxes" to the BLE application, as they hide all low-level communication details (*e.g.*, number of retransmissions, BLE signal strength, etc.). Developers have no explicit information or control over ongoing BLE packet transmissions and can only interact with the BLE radio via high-level commands. Therefore, very little is known about how BLE devices in real-world deployments running a single IoT application can use these three mechanisms at runtime to meet given application requirements, such as latency, reliability, and power consumption. To design and implement such effective and efficient mechanisms that cooperatively improve the performance of BLE, a detailed understanding of the behavior of BLE communication under different environmental conditions is necessary. This leads to our first research question (RQ):

RQ 1: *How can the BLE adaptation mechanisms be effectively used in real-world BLE applications to sustain given latency and reliability bounds while minimizing power consumption?*

BLE devices on the Internet. Typical IoT nodes need to communicate with cloud services, *e.g.*, to send and store measurement data, check for firmware and configuration updates, or deliver critical alarm messages. BLE-based IoT applications usually use a gateway, such as a smartphone or a laptop with custom applications installed, to translate standard GATT-based BLE packets into Internet Protocol (IP) packets that are sent to a predefined server on the Internet. This gateway-based data exchange, however, means that BLE devices cannot seamlessly communicate with other devices on the Internet, causing interoperability, scalability, and evolvability issues [43,232].

To combat these problems, the Internet Engineering Task Force (IETF) released the RFC 7668 defining how BLE devices can directly exchange IPv6 packets with other IPv6-enabled devices on

the Internet [156]. This so-called IPv6-over-BLE communication allows BLE nodes to seamlessly participate in the IoT without requiring application-specific translation between plain GATT-based BLE data and IP packets at a gateway. Although experimental studies show the potential of IPv6 over BLE [92, 152, 209], how resource- and energy-constrained devices can efficiently use IPv6-over-BLE communication is still an open question.

Moreover, IPv6-over-BLE devices are likely to experience different kinds of IPv6 traffic, such as exchanging critical data with a cloud server or sporadic ICMPv6 traffic from other devices on the Internet. It is important that IPv6-over-BLE devices have the ability to support different QoS levels of IPv6 packets and prioritize high-priority traffic (*e.g.*, time-critical data packets) over other data exchange. Unfortunately, how IPv6-over-BLE devices can support different QoS levels on top of IPv6 is unknown so far. This gap in research leads to our second research question:

RQ 2: *How can constrained BLE devices use IPv6 over BLE to efficiently connect to the Internet and allow reliable exchange of time-critical data?*

BLE nodes meeting end-to-end requirements. When BLE nodes exchange data with a cloud server, their communication may not only experience problems due to interference or multipath fading, but may also experience loss and delay on the external network path, *i.e.*, the network path outside the local BLE subnet. Although answering our RQ 1 and RQ 2 may significantly improve the communication performance of BLE nodes connected to the Internet, the resulting solutions may not be enough to meet stringent end-to-end requirements across the Internet, as they do not account for any loss and delay across the external network path. To successfully meet given end-to-end communication requirements, like a given reliability and latency, BLE nodes need to also capture and adapt to all loss and delay across the entire network path.

Several studies investigate how to capture and adapt to changes in loss and delay over the Internet [1,2,21,69,79,124,180], but they either focus on devices that use high-bandwidth Internet connections and are not constrained in their processing capabilities and power supply [1,2,69,79,180] or solely investigate how to increase the throughput of low-power nodes [21, 124]. Another large body of research has investigated how to sustain latency and reliability bounds in low-power wireless networks [80, 102], but they do not account for communication that extends across the local radio network. Unfortunately, even after more than a decade of research on IPv6-based low-power radio networks, such as *e.g.*, IPv6 over IEEE 802.15.4, little is known about low-power nodes capturing loss and delay across the whole network or meeting end-to-end requirements, such as a given latency or reliability. This leads to our third research question:

RQ 3: *How can low-power wireless nodes, such as BLE nodes, sustain a given end-to-end reliability and latency over the Internet while operating on a constrained energy budget?*

1.3 Contributions

In this thesis, we answer our three research questions and enable reliable and time-critical data exchange in BLE-based IoT applications. Towards this goal, we make the following scientific contributions:

Supporting time-critical data exchange over BLE connections. In the first contribution of this thesis, we answer our first research question (*RQ 1*) and allow BLE devices to reliably exchange data within a given transmission latency bound over a BLE connection. To enable such time-

critical communication, we investigate in detail the real-world performance of BLE communications under different environmental conditions and design three novel BLE adaptation mechanisms that are fully compliant to the BLE specification and allow off-the-shelf BLE devices to optimize their communication performance at runtime. All three BLE adaptation mechanisms can be used on off-the-shelf BLE devices to optimize the performance of BLE connections in dynamic, real-world application environments.

BLE channel management. We investigate how the overall BLE link-layer reliability and the reliability of individual BLE data channels are affected by different link-layer problems caused by external radio interference or fading effects. Based on our experimental results, we design an effective and hardware-independent BLE channel estimation and management mechanism that passively monitors the packet delivery ratio (PDR) of individual BLE data channels and dynamically adapts the list of used data channels to improve link-layer reliability. We implement our BLE channel estimation and management mechanism on two popular BLE hardware platforms, the Raspberry Pi 3 as well as the Nordic Semiconductor nRF52 platform, and experimentally show that our BLE channel management improves the link-layer reliability of a BLE connection by up to 22% without introducing additional power consumption.

Our solutions can potentially be applied to other wireless technologies using link-layer acknowledgments to estimate the quality of individual data channels without introducing any unnecessary energy consumption caused by probing packets. Such detailed channel knowledge may be utilized to efficiently manage the used channels of a wireless connection, as we show in our work, or to passively monitor the overall communication quality, *e.g.*, to choose the best out of multiple available routing devices.

BLE PHY mode adaptation. We extensively evaluate the performance of all four BLE PHY modes available for connection-based BLE communication. Using our findings, we design an effective BLE PHY mode adaptation mechanism that uses SNR measurements to dynamically choose the most suitable PHY mode to sustain a specified link-layer reliability and limit the power consumption of BLE devices. Our PHY adaptation mechanism, indeed, is able to sustain a link-layer reliability of 99% while minimizing the power consumption of BLE devices when possible. Furthermore, our PHY mode adaptation and our channel management successfully work in parallel to cooperatively improve the reliability of BLE connections.

Our effective PHY adaptation mechanism can potentially be used by other wireless devices, which support different PHY modes, to choose the most suitable PHY mode to sustain a given reliability while minimizing energy waste. Especially low-power wireless technologies that simultaneously use channel management and PHY mode adaptation may use our mechanisms to ensure that these two mechanisms work cooperatively and do not interfere with each other.

BLE connection parameter adaptation. We present how off-the-shelf BLE devices can monitor and control the delay of individual data transmissions over a BLE connection. Towards this goal, we show how BLE devices can passively monitor BLE transmission delays by using timing information on the standardized BLE Host Controller Interface (HCI). BLE devices can use these delay estimates in combination with our novel BLE latency models to dynamically adapt their BLE connection parameters to sustain a given transmission latency while minimizing their power consumption. An evaluation using four different BLE hardware platforms shows that our parameter adaptation mechanism reduces the number of packets exceeding the given transmission latency by up to a factor of 40.

Our passive delay monitoring may be used on top of other wired or wireless communication technologies, where the individual transmission and retransmission behavior is hidden to developers (*e.g.*, it is implemented by a separate closed-source communication peripheral), and upper communication layers simply interact with lower layers over packet buffers. By using our monitoring mechanism, developers are potentially able to estimate and control the transmission delay over such communication technologies to create time-critical applications.

Connecting BLE devices to the IoT using IPv6. The second contribution of this thesis is BLEach, the first full-fledged, open-source IPv6-over-BLE communication stack for constrained, low-power devices. With our work on BLEach, we answer our research question 2 (*RQ 2*) and allow BLE nodes to exchange IPv6 data packets with any other IP-compliant device on the Internet. This enables BLE nodes to directly talk to any IPv6 device on the Internet without requiring a gateway to translate standard GATT-based BLE data into IP packets. BLEach exposes the key parameters of IPv6-over-BLE communication as tuning knobs, allowing IPv6-over-BLE performance optimization at runtime. Using these exposed parameters, we go beyond the IPv6-over-BLE specification and add support for multiple simultaneous QoS levels to IPv6 over BLE, by exploiting its credit-based flow control. Experiments with BLEach show that it is fully interoperable with other IPv6 devices, has a minimal processing and memory demand, and that IPv6-over-BLE communication is more energy-efficient than IPv6 over IEEE 802.15.4. Furthermore, our proposed IPv6-over-BLE QoS support successfully allows nodes to dynamically prioritize certain IPv6 traffic flows over others.

Our QoS support can potentially be used by other communication technologies for constrained devices that use credits to control packet flow and avoid buffer congestion. With our approach, constrained nodes may prioritize certain IPv6 traffic flows over others and may even change the priority of individual traffic flows at runtime, *e.g.*, due to changing application requirements.

Meeting end-to-end requirements in BLE-based IoT applications. In our third contribution, we answer our third research question (RQ 3) and show how BLE devices can seamlessly communicate with a cloud server on the Internet within given reliability and latency bounds. Towards this goal, we extend our BLE latency model to capture the effect of the local BLE subnet, as well as any packet loss or delays that are introduced by the external network path, *i.e.*, the network path between the routing device and the cloud server. We show how low-power wireless nodes, such as BLE nodes, can accurately and efficiently estimate the communication latency and reliability across the entire network path by using short and infrequent probing bursts. Our estimation approach is fully compliant to the end-to-end principle of IP, *i.e.*, it does not require any changes to routing devices on the network path. Using our novel end-to-end model and latency estimation, low-power BLE nodes are able to meet given end-to-end reliability and latency requirements by adapting their BLE connection parameters. An experimental evaluation of our approaches shows that BLE nodes successfully meet a given end-to-end reliability of 99% and a given end-to-end latency of 1000 ms when communicating with a server on the Internet. This holds true even when the BLE connection is experiencing heavy link-layer problems and a cellular Internet connection with long communication delays is used.

Since our solution fully adheres to the IP end-to-end principle, it may be used by other IP-based radio nodes that need to estimate loss and delay across the entire network path while operating on a limited energy budget. Following our proposed solution, low-power nodes are potentially able to meet given end-to-end requirements for communication that spans across different networks.

1.4 Thesis Structure

The remainder of this doctoral thesis is structured as follows. Chapter 2 presents the technical foundations of BLE communication and the BLE communication stack. Chapter 3 summarizes related work on BLE, discusses state-of-the-art techniques for parameter adaptation in other wireless technologies, and provides an overview of studies on time-critical Internet communication. Chapter 4 focuses on optimizing data exchange over a BLE connection, by designing and evaluating efficient BLE channel management, BLE PHY mode adaptation, and BLE connection parameter adaptation mechanisms. In Chapter 5, we discuss how BLE devices can seamlessly exchange IPv6 packets with other devices on the Internet and present BLEach, our IPv6-over-BLE communication stack for constrained devices. Moreover, we show how IPv6-over-BLE devices can support different QoS traffic classes and prioritize individual IPv6 traffic flows over others. Chapter 6 shows how BLE devices can use our BLE end-to-end latency model and a novel network latency probing mechanism to sustain a given end-to-end reliability and latency while limiting the required power consumption of the BLE device. Chapter 7 concludes this dissertation with a summary of the contributions and a discussion of future research. Finally, a list of the scientific work that was published during this dissertation is attached in Chapter 8.

2 Foundations

This chapter presents the technical foundations of this dissertation. Section 2.1 discusses the technical background of Bluetooth Low Energy (BLE) technology and Section 2.2 summarizes the fundamental concepts of parameter adaptation approaches used in wireless technologies.

2.1 Bluetooth Low Energy (BLE) Technology

This section discusses the foundations of BLE technology. Section 2.1.1 explains the origins of BLE and discusses the difference between BLE and Classic Bluetooth. Section 2.1.2 summarizes the BLE communication stack. After that, this chapter discusses the two modes of BLE communication: connection-less BLE (Section 2.1.3) and connection-based BLE (Section 2.1.4). Finally, we summarize how BLE devices can support IPv6-over-BLE communication in Section 2.1.5.

2.1.1 A Short History of BLE

Classic Bluetooth communication was invented in 1994 at Ericsson to replace serial cables with short-range radio communication in the 2.4 GHz industrial, scientific and medical (ISM) band [11]. After the official release of the first Bluetooth specification in 1999 and the first Bluetooth-enabled mobile phone in 2001, Bluetooth communication has gained increased popularity and has been used for a wide range of use cases, such as audio streaming or printer and keyboard connectivity [29]. Unfortunately, due to the different requirements of individual applications, the Bluetooth communication stack was getting increasingly complex and bloated. This complexity resulted in Classic Bluetooth communication having a worse performance than other low-power wireless technologies, such as IEEE 802.15.4 [34, 130].

To address the shortcomings of Classic Bluetooth, the Bluetooth Special Interest Group (SIG) released Bluetooth version 4.0 in 2010, which introduced Bluetooth Low Energy (BLE) as a second communication technology into its specification [22]. BLE is a complete redesign of Classic Bluetooth technology with the objective to significantly reduce hardware complexity, code size, and energy consumption. Since Bluetooth version 4.0, the Bluetooth specification contains two different communication technologies, Classic Bluetooth and BLE, which share some communication stack layers and the Bluetooth brand but are not interoperable with each other.

Nowadays, almost all consumer devices with Bluetooth support are able to communicate over Classic Bluetooth and BLE [192]. However, it is clear that BLE will replace Classic Bluetooth communication in the foreseeable future, as new versions of the Bluetooth specification solely focus on improving and extending BLE functionality. For example, since Bluetooth specifica-

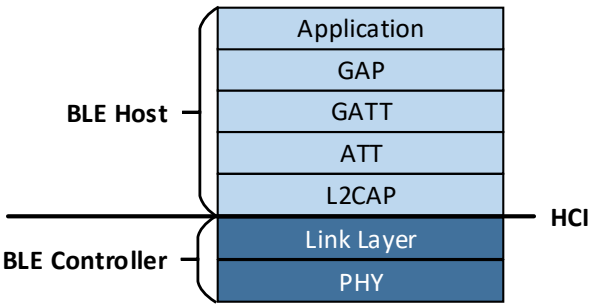


Figure 2.1: BLE communication stack consisting of a BLE controller connected via the BLE HCI to a BLE host.

tion version 5.0, BLE communication has the possibility to choose one out of four possible PHY modes, which allows to either increase the communication range of BLE communication or allows for higher data throughput [24]. Furthermore, BLE 5.0 devices can make use of extended advertising, which allows for longer broadcast packets and more frequency diversity in connection-less BLE communication. As of Bluetooth version 5.1, BLE devices can make use of angle-of-arrival and angle-of-departure techniques in combination with antenna arrays to accurately locate individual BLE devices [25]. With Bluetooth version 5.2, new BLE devices will have support for high-fidelity audio streaming, the last stronghold of Classic Bluetooth communication [26]. Finally, the newest BLE version 5.3 proposes multiple improvements, such as periodic advertising enhancement, faster BLE connection parameter adaptation, and channel classification enhancement, which will further improve the real-time capabilities of connection-based BLE [27].

2.1.2 BLE Communication Stack

Figure 2.1 shows the standard BLE communication stack with its stack layers. As the figure shows, the communication stack consists of two main components: the BLE controller and the BLE host. Host and controller are connected via the standardized BLE Host Controller Interface (HCI).

2.1.2.1 BLE Controller

The BLE controller is usually a dedicated, proprietary communication system on chip (SoC) implementing the BLE physical layer (PHY) and link layer. The BLE PHY layer specifies 40 different channels in the unlicensed 2.4 GHz ISM band that are used for BLE communication (shown in Figure 2.2). Furthermore, the PHY layer also defines four different PHY modes that BLE devices can use for data exchange. The link layer handles BLE device addressing, schedules packet transmissions and receptions, as well as manages incoming and outgoing packet buffers. Moreover, the link layer is responsible for BLE channel management, packet acknowledgment and flow control, as well as synchronizing data exchange with peer devices.

To simplify the design and implementation of the BLE host, the BLE controller autonomously handles all low-level behavior of BLE communication and acts as a “black box” to the upper stack layers. The BLE host simply uses standardized HCI commands to establish a connection or add packets to the outgoing packet buffer of the BLE controller to send data to a peer device. Whenever the BLE controller receives data from a peer device, it notifies the host via standardized

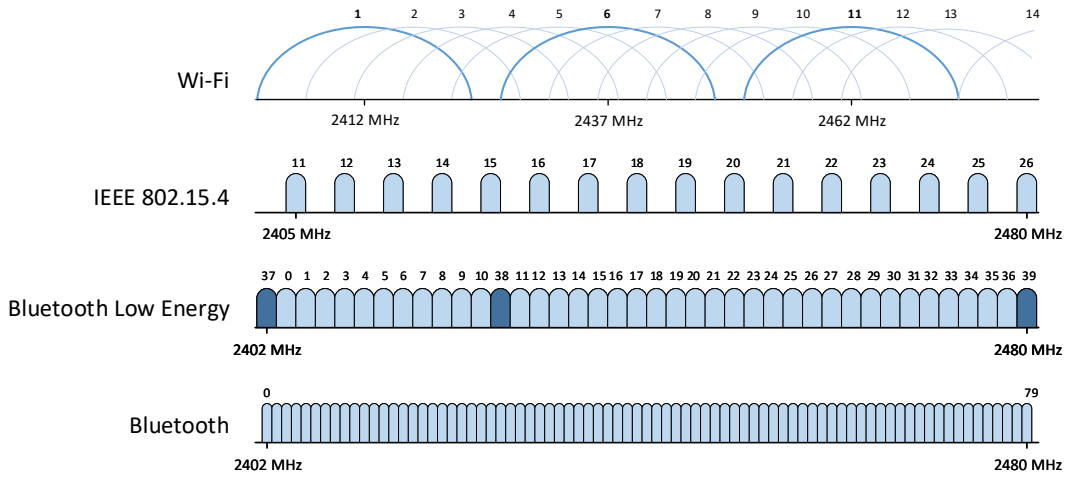


Figure 2.2: Frequency usage of the most popular wireless technologies in the 2.4 GHz ISM band (adapted from [96]). The BLE legacy advertising channels (channels 37, 38, and 39) are shown in dark blue while the BLE data channels (channels 0 to 36) are shown in light blue.

HCI events. All low-level communication details, such as link quality, used BLE channel, or the number of necessary (re-)transmissions, are hidden to the BLE host. The host can only issue high-level HCI commands to adapt the parameters of the BLE controller to change the communication timing, power consumption, or achievable data rate of BLE communication.

Although the Bluetooth specification clearly defines how BLE controllers can interoperably exchange data with other BLE devices, the specification mandates only key details in the link layer behavior that ensure interoperability. Other design decisions, such as the default communication parameters and the behavior of the adaptive BLE channel management, are not specified or discussed by the Bluetooth SIG. This leads to some manufacturers implementing effective channel management strategies in their BLE controllers, while other controllers only implement the mandatory behavior as specified. Therefore, BLE controllers from different manufacturers may significantly differ in terms of their behavior, especially when experiencing link-layer problems, as we show in Section 4.1.

2.1.2.2 BLE Host

The BLE host is located on the main application processor of a BLE device and implements the upper BLE stack layers and the BLE application.

One important layer for connection-based BLE is the Logical Link Control and Adaptation Protocol (L2CAP) layer, which is the interface between the higher layer protocols and the BLE controller. The L2CAP layer is responsible for fragmenting large data packets into smaller fragments that are individually sent over BLE connections and reassembled at the peer device. Furthermore, this layer also implements multiplexing and demultiplexing of the higher layer protocols, so that these protocols can operate simultaneously over a BLE connection.

The Attribute Protocol (ATT) defines a standardized way for BLE devices to discover, read, and write attributes from peer devices. The ATT follows a client-server model, where a server

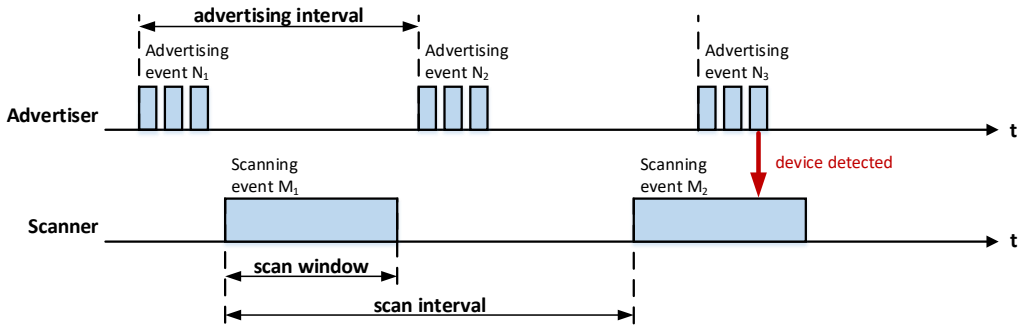


Figure 2.3: Connection-less BLE communication between an advertiser and a scanner. The advertiser uses all three BLE legacy advertising channels in every advertising event and is successfully detected by the scanner in scanning event M_2.

exposes different attributes that a connected client can read and write. The Generic Attribute Profile (GATT) uses ATT primitives and defines a standardized protocol that allows devices to interoperably expose GATT services and characteristics to other BLE devices. This GATT-based data exchange is the standard way for BLE devices to interact with BLE-enabled smartphones, tablets, and laptops. For example, a BLE-based heartrate monitor uses ATT and GATT to expose a heartrate service containing a heartrate measurement attribute. Nearby smartphones can detect this standardized heart rate service and can read the current heart rate measurement using ATT and GATT primitives.

The Security Manager (SM) layer provides BLE devices with security and privacy features that are required for state-of-the-art applications. Therefore, the SM layer defines procedures for device pairing, data encryption, authentification, or random device addresses, as well as allows key generation and storage. Finally, the Generic Access Profile (GAP) defines all mandatory functionality of BLE devices and ensures that BLE operations, such as device discovery, connection setup, and data exchange, work interoperably between devices from different manufacturers [91].

In Section 2.1.5, we will summarize how the BLE host can be extended to allow BLE devices to directly exchange IPv6 packets instead of GATT-based data exchanges. Furthermore, we will discuss how constrained BLE devices can implement this so-called IPv6-over-BLE communication in Chapter 5 of this dissertation. Next, we discuss the two modes of BLE communication.

2.1.3 Connection-less BLE Communication

In the simple connection-less communication mode, also known as BLE advertising, a BLE device acts as either advertiser or scanner. An advertiser periodically broadcasts short unidirectional data packets to nearby devices and a scanner listens for such advertising packets. A typical use case for connection-less BLE is device discovery, where an advertiser broadcasts its BLE capabilities (*e.g.*, a BLE heart rate wearable advertises its heartrate measurement capabilities). If a scanner receives a fitting advertising packet, it may read additional characteristics via a BLE scan request or may initiate a BLE connection to enable bidirectional data exchange. Another use case for connection-less BLE communication is BLE Mesh, where BLE devices use connection-less primitives (*i.e.*, advertising and scanning) to exchange data within a multi-hop mesh network [28].

A BLE advertiser periodically broadcasts data during advertising events, which are non-overlapping timeslots equally spaced out over time. The time between the start of two consecutive advertising events is defined by the BLE advertising interval, as shown in Figure 2.3. During an advertising event, the advertiser uses up to three dedicated channels, so-called BLE legacy advertising channels (shown in dark blue in Figure 2.2), to broadcast messages with a maximum length of 31 bytes. Similarly, a BLE scanner periodically performs scanning events, during which the scanner listens to one of the three legacy advertising channels for advertising packets. The timing of these scanning events is defined by two parameters: the scan interval defining the time between the start of two consecutive scanning events and the scan window defining the time the radio scans during an individual scanning event. Whenever an advertiser broadcasts on the same channel a scanner is currently listening to, an advertising packet can successfully be received. In case a scanner has successfully received an advertising packet, the scanner may perform one of three possible actions: (i) continue listening for other advertising packets, (ii) sending a scan request to the advertiser asking for additional data, or (iii) initiating a connection with the advertiser.

Figure 2.3 shows an exemplary BLE connection-less data exchange. The advertiser is periodically broadcasting short advertising packets on all three legacy advertising channels during every advertising event. The scanner is periodically scanning for nearby advertising packets. In scanning event M_2, the scanner is listening to one of the legacy advertising channels where the advertiser is simultaneously broadcasting data. Therefore, the scanner successfully receives an advertising packet from the advertiser and may initiate a BLE connection.

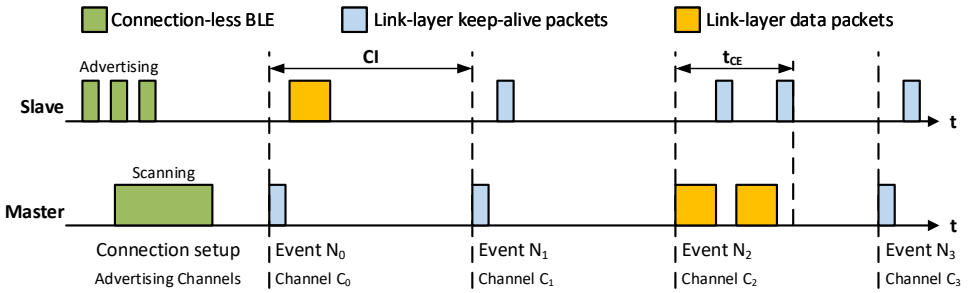
Since BLE version 5, devices may use additional connection-less features that are grouped under the term extended advertising [24]. Using extended advertising, devices may use all 40 available BLE channels for advertising/scanning and can use broadcast packets with a maximum length of 254 bytes. Nevertheless, connection-less BLE only allows unidirectional data exchange from an advertiser to one or multiple scanners. Furthermore, connection-less BLE is inherently unreliable when experiencing link-layer problems (*e.g.*, caused by Wi-Fi interference), as connection-less BLE does not use any link-layer acknowledgments or autonomous retransmissions. To allow a reliable and bidirectional data exchange, BLE devices need to use the connection-based communication mode of BLE, which we discuss next.

2.1.4 Connection-based BLE Communication

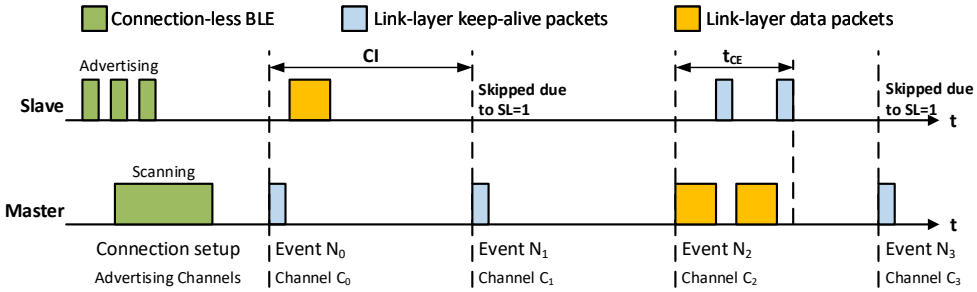
Whenever two BLE devices want to bidirectionally exchange data, they need to establish a BLE connection using connection-less primitives. In the created BLE connection, the former advertiser acts as BLE slave and the former scanner acts as BLE master.[1]

Communication over a BLE connection happens during connection events, which are non-overlapping timeslots where master and slave take turns in sending packets to each other. The time between the start of two consecutive connection events of a BLE connection is defined by the BLE *connection interval (CI)* parameter, as shown in Figure 2.4. A small CI means that both BLE devices wake up frequently to exchange BLE link-layer packets, leading to a lower trans-

[1] Please note that the terms master and slave may be controversial due to their historic associations and that the newest BLE specification version 5.3 [27] released in 2021 changed these terms to Central and Peripheral. To avoid any technical confusion, however, we follow the nomenclature used in the official BLE specifications before version 5.3 and our previously released publications and use the technical terms master and slave in some places to indicate the role of a device in connection-based BLE.



(a) $SL = 0$.



(b) $SL = 1$.

Figure 2.4: Connection setup and data exchange over a BLE connection between a BLE master and slave. The subfigures show how the slave latency (SL) affects the slave's behavior. Adapted from Publication B.

mission delay and a higher maximum data rate, but also causes a significant increase in the power consumption on both BLE devices. During a connection event, both BLE devices bidirectionally exchange link-layer packets until both devices have no more data to send or until the maximum duration of a connection event (t_{CE}) has been reached. Even if both devices have no data to send, they exchange short link-layer keep-alive packets, which only consist of the mandatory link-layer header, to maintain the connection. As these keep-alive messages cause unnecessary power consumption on BLE devices, the BLE specification foresees the BLE *slave latency (SL)* parameter, which allows the BLE slave device to skip up to SL connection events (as shown in Figure 2.4(b)).

Figure 2.4 shows two examples of a master and slave communicating over a BLE connection using different SL configurations. In the example shown in Figure 2.4(a), master and slave use connection-less BLE primitives to establish a BLE connection. The master starts connection event N_0 by sending a keep-alive packet (shown in blue) to the slave and the slave responds with a link-layer data packet (shown in yellow) carrying application data. During connection event N_1, both master and slave have no data to transmit and therefore only exchange the mandatory keep-alive packets to maintain the BLE connection. During connection event N_2, the master starts with sending data to the slave. As the data exceeds the maximum BLE link-layer packet length of 251 bytes, the master splits the data into two link-layer data packets that are both acknowledged via keep-alive packets by the slave. The example shown in Figure 2.4(b) is similar to the example in Figure 2.4(a), but the BLE slave is using a SL = 1 to skip unnecessary connection events, where only keep-alive messages would be exchanged. The different SL behavior can be seen when comparing connection events N_1 and N_3 in Figure 2.4(a) and Figure 2.4(b). When $SL = 0$, the slave needs to wake up during every connection event to exchange the mandatory keep-alive

packets, as shown in Figure 2.4(a). In case $SL > 0$, the slave may skip up to SL connection events to avoid unnecessary energy consumption, as shown in Figure 2.4(b).

Data channel management. At the beginning of every connection event, the Adaptive Frequency Hopping (AFH) mechanism of connection-based BLE selects one out of 37 BLE data channels (shown in light blue in Figure 2.2)[2]. The selected channel is then used for all link-layer packet exchanges during the whole connection event, however, a new channel is chosen for every connection event. Since all 37 BLE data channels are located in the 2.4 GHz ISM band, individual channels may have poor quality due to fading effects or external interference. To mitigate the effects of poor-quality channels on the overall BLE connection, BLE devices may adaptively *blacklist* individual data channels by dynamically updating the *channel map (C_{map})* of the BLE connection. The C_{map} of a connection defines which BLE data channel may be selected by the AFH mechanism, *i.e.*, if a data channel is blacklisted in the C_{map}, it will not be used for communication until being *whitelisted* again.

Although the BLE specification defines primitives for updating the C_{map} at runtime, it does not define how to detect channels with poor quality and when to blacklist them. However, in Section 4.1.2 of this dissertation, we present an effective and efficient BLE channel management approach for off-the-shelf BLE devices that black- and whitelists individual BLE data channels.

Link-layer acknowledgment and flow control. As mentioned above, the BLE link layer automatically handles packet acknowledgments (ACK) and flow control using a 1-bit ACK field and a 1-bit sequence number in the link-layer header of every packet. If a link-layer packet is not successfully acknowledged by the peer device, the BLE link layer autonomously re-transmits this packet until a corresponding acknowledgment is received. This means that the link layer ensures that link-layer packets are reliably sent to the peer in the correct order. The BLE host simply adds data to the outgoing link-layer packet buffer to transmit data to its peer. Furthermore, the host is notified by the link layer if any data packet has been successfully received. All the low-level behavior is implemented by the link layer and hidden to the BLE host.

PHY mode. Since BLE version 5.0, BLE devices are able to select one out of four possible physical layer (PHY) modes for transmitting link-layer packets: the 1M PHY, the 2M PHY, the Coded S2 PHY, and the Coded S8 PHY. The *1M PHY* is the original mode of BLE communication and is the only available PHY mode on BLE devices with a version below BLE 5.0. The 1M stands for the physical modulation of 1 Msym/s (Megasymbols per second) used by this PHY. The 1M PHY does not use any symbol coding or forward error correction (FEC) to increase its communication robustness. The *2M PHY* enables BLE devices to exchange data with approximately twice the data rate compared to the 1M PHY. The 2M PHY achieves this high data rate by using a physical modulation of 2 Msym/s and no symbol coding or FEC. The Coded S2 PHY and the Coded S8 PHY both use symbol coding and FEC to reconstruct flipped bits in received packets. Using these techniques, both Coded PHY modes achieve a more robust communication, which leads to significantly higher communication ranges. Like the 1M PHY mode, both Coded PHYs use a physical modulation of 1 Msym/s. As the name suggests, the Coded S2 PHY uses a symbol coding of 2, which means that every data bit is coded into 2 symbols over the air, resulting in a maximum physical data rate of 500 kb/s. Likewise, the Coded S8 PHY uses a symbol coding of 8, leading to a maximum physical data rate of 125 kb/s.

[2] Since BLE version 5, the BLE master can make use of two possible channel selection algorithms (CSAs). The used CSA, however, does not significantly affect the link-layer reliability of a BLE connection, as we show in Chapter 4.

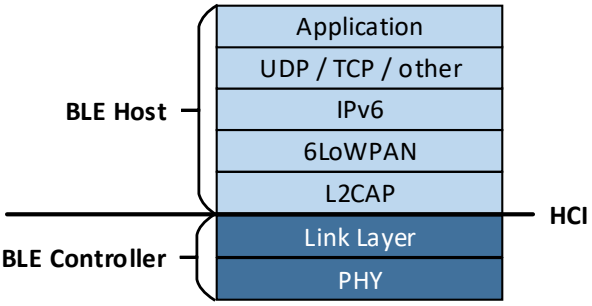


Figure 2.5: IPv6-over-BLE communication stack according to the RFC 7668 [156].

Parameter adaptation. During the connection setup, the BLE master sets the initial parameters (*i.e.*, CI, SL, PHY mode, and C_{map}) of the BLE connection. However, the parameters of the BLE connection may be adapted during runtime due to changing application requirements or environmental conditions. The BLE master may update any of the above parameters by exchanging standardized BLE link-layer commands with the BLE slave. For example, the BLE master may update the used CI and SL by sending a link-layer connection parameter update (`LL_CONNECTION_PARAM_IND`) containing the new connection parameters. Similarly, the master may issue a link-layer PHY update request (`LL_PHY_UPDATE_IND`) to adapt the used PHY mode or send a link-layer channel map update (`LL_CHANNEL_MAP_IND`) to change the used C_{map}, respectively. The BLE specification defines a mandatory delay of at least six connection events between a master issuing a parameter update and the new parameters being used[3].

Unlike the BLE master that can adapt the parameters at any point in time, the BLE slave may only request changes to the BLE connection parameters or the used PHY mode from the master. For example, the slave may request new CI and SL parameters by sending a link-layer connection parameter update request (`LL_CONNECTION_PARAM_REQ`) to the master. Likewise, the slave may request a different PHY mode by issuing a link-layer PHY update request (`LL_PHY_UPDATE_REQ`). After the master has received any of these requests, it may decide to either decline the request and respond with a negative acknowledgment (NACK) or accept the request and send an acknowledgment (ACK) followed by a parameter update as described above.

In Chapter 4 and Chapter 6 of this dissertation, we will show how BLE devices can use these parameter adaptation mechanisms to sustain specific communication requirements, such as a given reliability or latency while limiting the power consumption of BLE devices.

2.1.5 IPv6 over BLE

In this section, we show how BLE devices can make use of IPv6-over-BLE communication, as standardized by the RFC 7668 [156], to directly exchange Internet Protocol version 6 (IPv6) packets with other IP-enabled devices on the Internet.

Figure 2.5 shows the architecture of the IPv6-over-BLE communication stack as specified by

[3] With the release of BLE specification version 5.3 in July 2021 [27], BLE devices gain the ability to subrate BLE connections, which allows a faster BLE connection parameter adaptation under certain conditions. Because this feature is not yet supported in existing BLE hardware platforms, this dissertation sticks to the mandatory delay of at least six connection events when updating BLE parameters.

the RFC 7668 [156]. IPv6 over BLE reuses major parts of the standard BLE stack, such as the whole BLE controller and the BLE L2CAP layer. However, instead of using the ATT, GATT, and GAP protocol, IPv6 over BLE makes use of 6LoWPAN, a bundle of techniques that have been developed to connect IEEE 802.15.4 devices via IPv6 to the Internet, to exchange IPv6 data over BLE connections. Specifically, IPv6 over BLE requires the following stack layers:

IPv6-over-BLE communication reuses the existing BLE controller and the BLE HCI. IPv6-over-BLE nodes use BLE advertisements to broadcast their capabilities and wait for an IPv6-over-BLE router to establish a BLE connection. After a successful connection establishment, node and router use only connection-based BLE communication to bidirectionally exchange data.

IPv6 over BLE makes use of the existing L2CAP layer to handle fragmentation and reassembly of large IPv6 packets as well as buffer overflow prevention. One difference to the GATT-based BLE communication, however, is that exchanging IPv6 data requires the use of connection-oriented L2CAP channels in LE credit-based flow control mode [156]. This credit-based flow control mode allows BLE devices to exchange large IPv6 packets over constrained BLE connections by splitting large packets into multiple smaller L2CAP fragments. To exchange such L2CAP fragments, the L2CAP layer establishes a logical channel between the two IPv6-over-BLE devices during connection setup. Once such a channel is established, L2CAP uses credits to control the flow of individual fragments. During setup, each device grants its peer a number of initial credits. Furthermore, a device may grant its peer additional credits anytime using a dedicated L2CAP signaling channel. Sending one L2CAP fragment costs one credit and if a device has no more credits left, it cannot send any more fragments.

The 6LoWPAN layer sits on top of the L2CAP layer and is responsible for improving the efficiency of IPv6 communication by performing IPv6 header compression. The 6LoWPAN layer for IPv6 over BLE is based on the 6LoWPAN for IPv6 over IEEE 802.15.4, as specified by the RFC 6282 [101], but does not handle fragmentation of IPv6 packets, as this is already done by the L2CAP layer below. Using 6LoWPAN header compression, the 40-byte long IPv6 packet header may reduce down to 3 bytes. IPv6 over BLE uses ordinary IPv6 packets in the network layer, which makes IPv6 over BLE fully interoperable with any other IPv6 device. Furthermore, any transport layer capable of transporting IPv6 packets, such as UDP or TCP, can be used to exchange application data.

In Chapter 5, we will show in detail how constrained BLE nodes can make use of IPv6 over BLE. We further discuss how plain IPv6-over-BLE communication can be extended to support different QoS levels.

2.2 Foundations of Wireless Parameter Adaptation

In this section, we discuss the most common parameter adaptation techniques in wireless technologies that are used to improve communication performance or sustain a given QoS. First, we list the most common communication parameters of wireless technologies in Section 2.2.1. Second, we discuss the different approaches for estimating the quality of wireless links in Section 2.2.2. Third, we summarize different kinds of models of wireless communication that may be used to find suitable wireless parameters in Section 2.2.3. Forth, we describe different parameter adaptation approaches used in wireless technologies in Section 2.2.4.

2.2.1 Communication Parameters

Wireless communication systems have multiple parameters that affect key performance metrics, such as latency, reliability, throughput, communication range, or energy efficiency. Devices may adapt one or multiple of these parameters at runtime, *e.g.*, to improve reliability or decrease power draw. The most common parameters are:

Transmission power. The transmission power of wireless transmissions significantly affects the communication range, transmission reliability, and power consumption of wireless devices. A high transmission power leads to successful packet transmissions over long communication ranges, as the transmitted packet can still be successfully decoded at the receiving device. Unfortunately, higher transmission powers also lead to a significantly higher power consumption of the transmitting devices. A low transmission power, in contrast, leads to a more energy-efficient wireless data exchange at the cost of reduced communication range and reliability. Because the used transmission power affects communication reliability and energy efficiency, multiple radio technologies dynamically adapt their transmission power at runtime. For example, IEEE 802.15.4 [134], LoRa [35], Wi-Fi [114], cellular [19, 123], and UWB [88] dynamically change their transmission power (amongst other parameters) to improve communication performance.

Communication channel. The used communication channel, *i.e.*, the frequency band used for wireless data exchange, significantly affects the communication performance of wireless technologies. For example, the communication channel width impacts the maximum achievable data rate of wireless communication, *i.e.*, a wide channel allows a higher data rate than a narrow channel. Some wireless technologies, like BLE and IEEE 802.15.4, use only narrow communication channels with fixed channel width. Other wireless technologies, which are typically less energy-constrained, however, can dynamically change the used channel width to make efficient use of the available frequency spectrum and improve communication performance. For example, 4G and 5G devices dynamically adapt the channel width of an upcoming packet transmission depending on the current RF environment [157, 228]. Similarly, state-of-the-art Wi-Fi devices adapt the used channel width at runtime to improve the efficiency of their channel usage [125, 164].

If a communication channel has a bad quality, *e.g.*, caused by external interference or fading effects, its communication reliability drops significantly, which in turn negatively affects communication latency, throughput, and energy efficiency. Therefore, most wireless technologies offer the ability to switch the used wireless channel to optimize their communication performance. Some technologies, such as Wi-Fi [139], allow changing the used channel reactively, *e.g.*, when the link quality turns bad. Other technologies, such as BLE, proactively use frequency hopping techniques to mitigate the impact of individual bad channels on the overall communication performance, as we discuss in detail in Section 4.1.2 of this dissertation.

Modulation. The modulation of wireless signals defines how a stream of digital data is transformed into a continuous-time signal that can be transmitted over the air [126]. Typical parameters of a modulation scheme are the physical modulation rate, the information per exchanged symbol, and which aspect of a wave signal is used to encode information (frequency, amplitude, or phase). As the modulation significantly impacts communication reliability, energy efficiency, and achievable data rate, some wireless technologies allow to dynamically adapt their modulation scheme. For example, 4G devices may change their modulation scheme at runtime to improve achievable throughput and reliability [123].

Coding. Coding defines how application data is translated into symbols sent over the air, and vice versa [126]. The used coding scheme impacts maximum data throughput, communication range and reliability, transmission latency, as well as the energy efficiency of wireless communication. Robust coding schemes typically use redundancy and error correction techniques that allow packets to still be decoded/recovered at the receiving device, at the price of longer radio-on times and therefore higher power consumption per packet. Lightweight coding schemes may use only error detection and no redundancy to provide very energy-efficient coding at the cost of lower communication reliability. Similar to modulation, modern wireless technologies may dynamically adapt their coding scheme to optimize their communication performance. For example, 4G [123], LoRa [197], and UWB [88] devices may change the coding scheme at runtime. In Section 4.1.3, we show how BLE devices can adapt aspects of their coding scheme at runtime to sustain a given communication reliability.

Packet length. The used data packet length affects the communication reliability, latency, and data throughput. The overhead per packet (*i.e.*, packet header) usually does not depend on the actual data length of a packet. Therefore, longer packets lead to relatively less communication overhead and higher achievable data throughput. However, longer packets are more likely to be interfered and have generally a higher probability of containing at least one bit error and may hence need to be retransmitted, leading to a poorer communication performance in noisy environments. Wireless technologies may dynamically adapt their maximum packet lengths depending on the transmission reliability to improve communication performance. For example, IEEE 802.15.4 networks may adapt the used packet length depending on the current noise in the environment to improve channel utilization [61] or energy efficiency [62].

Packet timing. The timing of packet transmissions affects the communication latency, data throughput, and energy efficiency of the wireless data exchange. This is especially true for radio technologies, where individual devices can only exchange data within dedicated periodic communication slots. If a device has very frequent communication slots, it can achieve high data throughput and low communication latency at the price of increased energy consumption. If communication slots are infrequent, the wireless device can conserve energy, but can also achieve only low throughput and experience high transmission latencies. For example, the timing of IEEE 802.15.4 transmissions may be adapted to improve network performance [67, 236, 237]. In Chapters 4 and 6 of this thesis, we show how the timing of packets sent over a BLE connection can be adapted at runtime to sustain given latency bounds while minimizing power consumption.

Redundant transmissions. Some wireless technologies may even transmit parts of a packet or whole packets multiple times to introduce redundancy and therefore improve communication reliability at the cost of increased power consumption and decreased achievable throughput. Other than coding, where the redundancy may be contained within individual packets, using redundant transmissions spreads the redundancy across multiple data packets. For example, IEEE 802.15.4 devices may use redundant transmissions of the packet header to improve reliability under interference [133] or transmit whole packets multiple times to improve communication performance [76]. 5G devices may also use redundant packet transmissions to achieve ultra-reliable low-latency communication (URLLC) [187].

2.2.2 Link Quality Estimation

Estimating the quality of a wireless link is vital for monitoring and controlling the performance of wireless data exchanges. Having good estimates of the underlying radio link quality is an important building block of adaptation mechanisms that try to mitigate link-layer problems. According to Baccour et al. [15], the process of link quality estimation of a wireless link consists of three subsequent steps: link monitoring, link measurement, and metric evaluation.

2.2.2.1 Link Monitoring

Link monitoring defines the strategy of how individual link measurements are retrieved. In general, there are three different kinds of link monitoring: active, passive, and hybrid link monitoring.

Active link monitoring. Using active link monitoring, wireless devices use dedicated probing packets to monitor the quality of individual channels. Such probing packets are typically sent periodically and in addition to ordinary data exchange, which introduces additional communication overhead (and therefore energy consumption). The benefit of dedicated probing packets, however, is that they can be used to also probe frequencies that are not regularly used for data exchange. Active monitoring is often used in wireless networks to measure the quality of radio channels. For example, IEEE 802.15.4 devices may actively probe individual channels to detect if a channel is occupied by co-located radio devices [63, 74, 215]. Likewise, Wi-Fi devices may actively measure the noise floor on all channels to select the best Wi-Fi channel for communication [20, 114].

Passive link monitoring. In contrast to active link monitoring, passive link monitoring approaches use existing data exchanges to monitor the link. Passive link monitoring approaches are therefore more energy-efficient, as they do not introduce any additional data exchange for monitoring purposes. Such passive approaches, however, are only able to probe channels used for data exchange. Channels that are not used for communication are not probed using passive link monitoring. Passive monitoring is a popular technique used by low-power wireless devices to monitor the quality of their wireless links. Many IEEE 802.15.4 devices passively estimate the link quality to choose a single suitable channel [225] out of a set of channels [70, 131] for communication. In Section 4.1 of this dissertation, we also make use of passive link monitoring to improve the reliability of BLE connections via effective channel management.

Hybrid link monitoring. Some link monitoring approaches combine both active and passive monitoring techniques. These hybrid approaches usually provide a good tradeoff between energy efficiency and periodic link measurements. For example, low-power IEEE 802.15.4 devices in multihop networks combine active probing with passive estimation techniques to sustain time-critical data exchange in industrial settings [86, 215].

2.2.2.2 Link Measurement

Link quality estimators may measure one or multiple link metrics that can be gathered on the sending or receiving device.

Wireless devices can use information about packet transmissions to measure the link. For example, devices may investigate packet sequence numbers, number of retransmissions, or transmission

timestamps to calculate the current packet delivery ratio (PDR) [70, 188] or the expected transmission count (ETX) [122, 188] over a link. Furthermore, devices can use information about received packets or acknowledgments to extract link measurements. Such information may include the received signal strength indicator (RSSI) [86, 189], the signal-to-noise ratio (SNR) [137], the checksum status, or a dedicated link quality indicator (LQI) [120] provided by the radio hardware.

Section 4.1 of this thesis investigates the available BLE link metrics in detail. We list and discuss the BLE link metrics available on off-the-shelf BLE devices and show how the PDR and the SNR of BLE data packets can be used for effective BLE channel management and PHY mode adaptation, respectively.

2.2.2.3 Link Metric Evaluation

To get an estimate of the link quality of a channel, devices need to combine recent link measurements with a link quality estimation approach.

Some devices use simple estimation techniques, such as low-pass filters, simple averaging, exponentially weighted moving average (EWMA), or counting how many measurements exceed a given threshold, for link metric evaluation [63, 70, 121, 137, 215]. These techniques require minimal memory and computing resources and, therefore, fit on almost any device. Simultaneously, these techniques provide estimates that are accurate enough for most applications.

If a more sophisticated link estimation is required by an application, devices may use more advanced filtering techniques, ranging from Kalman filters [105] over probabilistic models [69] to powerful machine learning (ML)-based estimation approaches [180]. These estimation techniques, however, may require significantly more memory and computing resources, making them often not suitable for very constrained devices.

In Chapter 4 and 6, we show how BLE devices can use simple estimation techniques to accurately estimate loss and delay over BLE connections.

2.2.3 Modelling Approaches

Models of wireless communications are often used to predict the performance of different communication parameters or to choose suitable communication parameters for parameter adaptation.

Analytical model. One common way of modeling wireless communication is to use purely analytical models. Deriving an analytical model of a specific wireless technology, however, may be tricky, as it requires detailed knowledge of the underlying communication mechanisms of the technology. One benefit of analytical models is that they have a closed form, *i.e.*, they can be mathematically solved to get suitable communication parameters for given requirements. Analytical models are very popular in wireless networks, such as IEEE 802.15.4 and BLE networks, to model and optimize key performance metrics, like energy consumption, latency, and reliability [33,68,116,118,179,190]. In this dissertation, we derive such analytical models for connection-based BLE communication and use these models to sustain given end-to-end requirements.

Data-driven model. In contrast to analytical models, data-driven models use ML techniques to derive a communication model for a given wireless technology. These data-driven models do not require intricate knowledge of the communication mechanisms of the wireless technology,

however, they usually require a vast number of previously recorded training data to accurately train the wireless communication model. Such data-driven models are, for example, used in wireless networks to model and control communication reliability [137, 231] or latency [180].

Mixed model. Some techniques combine the above approaches to derive suitable models of wireless communication. One possible way to mix these approaches is to generate a suitable base analytical communication model and use data-driven modeling to calibrate the model parameters. For example, such an approach is used by Fernandez et al. [74] to derive several analytical reliability models of IEEE 802.15.4 links and use recorded data for model calibration.

2.2.4 Parameter Adaptation

To optimize communication performance and sustain given application requirements, wireless devices may dynamically adapt one or multiple communication parameters at runtime. To adapt communication parameters, devices may use the following adaptation approaches:

Rule-based adaptation. The simplest way to adapt communication parameters at runtime is to use predefined adaptation rules. Using such adaptation rules, a device may change communication parameters whenever a measured link quality metric reaches a given threshold. Depending on the application requirements, such a threshold may be fixed [70, 131] or adaptive [121]. For example, a radio device estimates the communication reliability of a radio link via PDR measurements. Whenever the measured PDR drops below a given reliability threshold, the device adapts its communication parameters to a more reliable setting to achieve a higher reliability. Such adaptation approaches are successfully used in IEEE 802.15.4 [70, 131], LoRa [197], UWB [88], and 4G networks [123] to optimize performance.

A drawback of rule-based adaptation is that devices may continuously oscillate between two different parameter settings under certain environmental conditions, leading to insufficient communication performance. To combat such oscillation, most rule-based adaptation approaches use an adaptation hysteresis, *i.e.*, the device uses two different thresholds for parameter switching. For example, a device uses rule-based adaptation to sustain a PDR of 90%, but due to environmental conditions, the device oscillates between two parameter sets. To mitigate oscillation, the device switches to more robust communication parameters when the PDR drops below 90%, but only switches to less robust (but more energy efficient) settings when it experiences a PDR of 95%.

In Section 4.1 of this thesis, we use rule-based adaptation to design an efficient BLE channel management and an efficient BLE PHY mode adaptation to improve the reliability of BLE.

Model-based adaptation. Another parameter adaptation approach is to make use of detailed models (as discussed in Section 2.2.3) to select a suitable set of new communication parameters. Using the current environmental state (*e.g.*, link-layer loss due to interference) and application requirements, devices can calculate the best communication parameters for the current conditions. As discussed in Section 2.2.3, devices using closed-form models can analytically calculate the most suitable communication parameters [116, 179]. More complex models, however, require devices to use network calculus [80] or optimization techniques (*e.g.*, mixed integer nonlinear programming) [138, 237]. Model-based adaptation approaches are successfully used in IEEE 802.15.4 [75, 102], 4G [195], 5G[19] and Wi-Fi networks [114] to optimize performance.

In Section 4.2 and Chapter 6, we use our analytic BLE models to adapt the BLE connection

parameters at runtime to sustain given application requirements.

Control-based adaptation. Wireless devices can make use of feedback control to adapt their used communication parameters to sustain given application requirements. Using such adaptation approaches, a device continuously monitors communication feedback and gradually changes one or multiple communication parameters until the desired communication performance is reached. For example, 5G devices continuously monitor channel state information feedback from peer devices to control their modulation and coding scheme [213].

In the next chapter, we discuss how BLE and other wireless technologies apply the above parameter adaptation techniques to optimize their communication performance.

3 Related Work

In this chapter, we summarize research related to the topics of this dissertation. First, we discuss research on BLE in Section 3.1. Next, we show how other wireless technologies adapt communication parameters to optimize performance in Section 3.2. Finally, we list different approaches used on the Internet to achieve reliable and time-critical data exchange in Section 3.3.

3.1 Research on BLE Communication

This section discusses the state-of-the-art research on BLE communication related to this thesis. In every subsection, we highlight how the work of this dissertation differs from existing research.

3.1.1 Connection-less BLE Communication

Connection-less BLE communication has been the focus of many research studies. Based on these works, we know how to design BLE-based indoor localization [73, 110], locality-based authorization [82], or group management [85].

Several models and experimental studies of connection-less BLE allow developers to improve the BLE device discovery process. For example, the model presented by Liu et al. [135, 136] shows the effects of the BLE connection-less parameters on the power consumption of BLE devices during device discovery. Jeon et al. [106] show the trade-off between discovery latency and energy consumption during device discovery for different connection-less BLE parameters. The works of Kindt et al. [115, 117, 118] present accurate models of connection-less BLE communication and discuss how to optimize BLE device discovery and even provide an upper bound on discovery latency. Due to the work of Cho et al. [47], we know that an increasing number of BLE devices performing device discovery may lead to exponentially increasing discovery latencies. Furthermore, the work of Julien et al. [107] shows how to find suitable connection-less BLE parameters for the device discovery process to minimize packet collisions. Devices may adapt their BLE parameters based on the time of day to optimize device discovery latency while conserving energy [175]. Moreover, BLE devices can make use of the extended advertising functionality introduced in BLE version 5 to optimize device discovery [95] and can make use of the mechanism proposed by Mikhaylov [146] to optimize the connection establishment process.

In addition to device discovery and connection setup, connection-less BLE communication is also used to create mesh networks [53, 54, 98, 148, 149] and the Bluetooth SIG has even specified how short packets can be transmitted over connection-less BLE in mesh networks [28]. More recently, connection-less BLE communication has been successfully used for efficient concurrent

transmissions in multi-hop networks [9, 16].

In contrast to the above research, this dissertation focuses on the connection-based BLE communication mode, as it supports reliable and bidirectional data exchange. Nevertheless, applications using the work in this dissertation can also make use of the research work on connection-less BLE, *e.g.*, to speed up device discovery or locate individual BLE devices.

3.1.2 Communication Reliability of BLE Connections

Multiple research works have investigated how BLE connections are affected by co-located radio devices using the 2.4 GHz ISM band. These studies show that application data exchange over a BLE connection sustains a reliability of 100%, even when interfered by co-located BLE, IEEE 802.15.4, Classic Bluetooth, and Wi-Fi devices [38, 143, 193, 220]. The reasons for this reliability are BLE's autonomous retransmission of unsuccessfully transmitted link-layer packets and the Adaptive Frequency Hopping (AFH) mechanism of connection-based BLE. Every BLE link-layer packet is eventually successfully received by the peer device, however, the communication latency over a BLE connection experiencing external interference significantly increases [38, 220]. Likewise, simulations of BLE's AFH mechanism show that frequency hopping, even without adaptive BLE channel management, improves BLE's performance under radio interference [6–8]. Some studies even propose cooperative solutions in the time and frequency domains between co-located wireless networks to improve the performance of BLE connections [42, 154]. However, no research has investigated how BLE devices can reduce the number of interfered link-layer packets by applying an effective BLE channel management, *i.e.*, adaptively blacklisting BLE data channels with poor quality.

A few works study performance metrics, *i.e.*, maximum data throughput, energy consumption, and communication range, of the different BLE PHY modes analytically [33] or experimentally [9, 50, 113, 166]. Unfortunately, these studies either focus on the connection-less mode of BLE [9, 50, 166, 177] or do not investigate all 4 available PHY modes [113]. Furthermore, none of the works investigate the robustness of the different PHYs under external interference.

To the best of our knowledge, the work in this dissertation provides the first extensive experimental study on BLE data channel management and adaptive PHY mode selection in connection-based BLE. After experimentally investigating the BLE link-layer behavior under different conditions in detail, we design an effective and efficient BLE channel management and a PHY mode adaptation approach, which both cooperatively improve the BLE link-layer reliability while limiting unnecessary power consumption. Our work on the BLE PHYs inspired further research on this topic by Sheikh et al. [189].

3.1.3 Communication Latency over BLE Connections

The communication latency of Classic Bluetooth applications has been studied by several research works [44, 173, 181]. For example, Chen et al. [44] use the Synchronous Connection Oriented (SCO) links of Classic Bluetooth and adapt the Adaptive Retransmission Request (ARQ) timeout to sustain an upper latency bound on transmissions. Likewise, Razavi et al. [173] adapt the ARQ timeout of Classic Bluetooth's SCO links to support basic video streaming. Sattar et al. [181] show that the Bluetooth v2.1 Enhanced Data Rate mode and the Bluetooth v3 High Speed mode can be

used to stream audio and video data by using sophisticated audio/video codecs. All these works on Classic Bluetooth, however, only optimize for throughput or latency and completely neglect any additional power consumption, making these approaches unsuitable for low power devices. Furthermore, these studies only use Classic Bluetooth, which is completely different from BLE (as stated in Section 2.1.1), and therefore cannot be reused for BLE communication.

As stated above, individual application data exchanges over a BLE connection may be delayed due to external interference [38, 193]. Therefore, there exist a number of different approaches to analytically model the communication latency of a BLE connection [68, 116, 179]. Although these works account for link-layer packet loss during data transmission, they all require low-level BLE link-layer information, such as bit error rate, number of cyclic redundancy check (CRC) errors, or data transmission probability. These low-level metrics, however, are hidden in the BLE controller and, therefore, cannot be used by adaptation mechanisms on the BLE host to estimate or control the latency of data transmissions. Furthermore, these models only investigate the latency of transmissions from the BLE slave to the BLE master. How a BLE slave can estimate and control the latency when receiving packets from the BLE master is not investigated. Only the work by Lee et al. [127, 128] uses proactive probing packets to capture link-layer packet loss on the BLE host. However, the authors use their packet loss estimates to select the best of multiple peer connections for data exchange [128] or adapt only the BLE connection interval, one of the two BLE connection parameters, to keep the BLE connection alive under harsh conditions [127]. They do not use their models to sustain a given transmission latency over a BLE connection.

Some works investigate how the standard BLE link-layer behavior can be modified to support real-time transmissions over a BLE connection. Marinoni et al. [144] create a custom real-time protocol that shares the BLE radio with ordinary BLE communication. Rondon et al. [178] propose three novel link-layer retransmission schemes, which limit the maximum number of retransmission attempts of certain link-layer packets, to improve the real-time capabilities of connection-based BLE. Likewise, Agnoletto et al. [4] present a time-slot-based transmission scheme that allows packet prioritization over BLE connections to support real-time communication. Unfortunately, all these works require significant changes to the link-layer behavior of BLE devices and, therefore, cannot be used by existing BLE devices. Furthermore, the works by Rondon et al. and Agnoletto et al. only show their real-time capabilities in computer simulations [4, 178].

One major step towards real-time data exchange over BLE connections is the release of Bluetooth version 5.2, which specifies how BLE devices can support high-fidelity audio data exchange over connection-based BLE [26]. With this new BLE version, BLE devices can make use of isochronous (ISO) channels to exchange time-sensitive data with peer devices. These ISO channels are a complete redesign of the SCO links of Classic Bluetooth focussing on low energy consumption as well as real-time communication capabilities. As of today, however, only a very limited number of devices support BLE version 5.2 with its ISO channels and no study has investigated how these new BLE features can be used to sustain given latencies over BLE connections.

In contrast to the existing research, the work in this dissertation shows a novel BLE latency model that captures the effect of both BLE connection parameters, the connection interval and the slave latency, on the communication latency of traffic from and to a BLE device. Our model can be used on off-the-shelf BLE devices to capture link-layer packet loss, as it uses only standard information available on the BLE host. Furthermore, our end-to-end BLE model allows BLE devices to capture loss and delay across multiple networks, *i.e.*, the Internet, and can be used on BLE devices to sustain given end-to-end latency and reliability bounds. As our work does not

require any changes to the BLE controller, it has already been used for adaptive transmission power control [162] and may be used by future BLE devices supporting BLE version 5.2 and above to sustain time-critical application data exchange.

3.1.4 Energy Efficiency of Connection-based BLE

Several works have experimentally studied the power consumption of devices using connection-based BLE communication. Gomez et al. [87] have measured the transmission delay and energy consumption for exchanging ATT packets with a length of 37 bytes over a BLE connection. Dementyev et al. [55] have investigated the energy consumption of BLE slave devices using connection-based BLE when periodically transmitting 8-byte data packets. Similarly, Siekkinen et al. [190] have studied the energy consumption of connection-less and connection-based BLE communication. The latter two studies both experimentally show that connection-based BLE communication is significantly more energy-efficient than the competing low-power wireless technologies ANT and IEEE 802.15.4.

A few works accurately model the energy consumption of BLE devices exchanging data over BLE connections [118, 190]. Other work by Kindt et al. [116] proposes an adaptation mechanism running on the BLE master to dynamically change the BLE connection interval depending on the traffic load over the BLE connection to increase energy efficiency.

In this thesis, we investigate the energy efficiency of connection-based BLE when exchanging IPv6 data. Therefore, we provide a detailed experimental comparison of IPv6 over BLE and IPv6 over IEEE 802.15.4 on the same hardware platform using a wide range of different IPv6 packet lengths. We conclude that standard IPv6-over-BLE communication is significantly more energy-efficient than IPv6 over IEEE 802.15.4 under comparable application traffic and show different extensions to IPv6 over BLE that boost its performance even further. Furthermore, we provide the first experimental evaluation of the different BLE PHY modes when used in connection-based BLE and show how the used PHY mode can be adapted at runtime to optimize energy efficiency.

3.1.5 BLE-based IoT Applications

There exist a plethora of IoT applications that use connection-based BLE to exchange data with cloud servers, such as smart city deployments [83], smart grid devices [49], and smart healthcare applications [10,93,109,219]. These applications use BLE gateways, usually smartphones, to convert GATT-based BLE packets into IP packets that can be exchanged over the Internet [200]. As shown by Zachariah et al. [232], having dedicated gateway devices results in multiple architectural problems, such as decreased usability, availability, security, privacy, and deployability.

To address these problems and create dependable IoT applications based on BLE, the research community proposed IPv6-over-BLE communication, where BLE devices directly exchange IPv6 packets with any other IP-based device [156], as we summarize in Section 2.1.5. The potential of IPv6 over BLE in smart home applications has been the focus of Chang et al. [43]. Similarly, Nieminen et al. [155] discuss the performance of IPv6-over-BLE communication and show its feasibility in a simple experimental campaign. Furthermore, Lee et al. [128] use connection-less and connection-based BLE communication to create mesh networks exchanging IPv6 packets.

Nowadays, IPv6 over BLE is part of several operating systems for constrained nodes, such as

Mynewt [12], the Nordic Semiconductor nRF5 SDK [158], and the Zephyr RTOS [217]. Unfortunately, the IPv6-over-BLE stack of these operating systems either only supports the node role [12, 217], or does not allow full access to the BLE controller [158].

In this dissertation, we present the first fully open-source IPv6-over-BLE communication stack that allows BLE devices to act as nodes and routers and allows full control over the BLE controller. Our IPv6-over-BLE stack is officially part of the popular Contiki Operating System (OS) and has been used to create IPv6-mesh over BLE networks [41]. Besides designing and implementing plain IPv6-over-BLE support, we have also exposed the key parameters of IPv6-over-BLE communication allowing optimization of the IPv6-based data exchange at runtime. Using these exposed parameters, we have created multiple extensions to plain IPv6-over-BLE, most notably adding support for different QoS levels over a single IPv6-over-BLE link.

3.1.6 Other Relevant BLE Research

Connection-based BLE has been used to create custom mesh network solutions, where multiple BLE connections form multihop mesh networks [46, 129, 165, 234]. Some works have even investigated how to schedule parallel BLE connections on individual devices to decrease the communication latency across a mesh network [129, 165].

Hussain et al. [99] enable mobility of BLE slave devices, as they support seamless handover of active BLE connections from one BLE master to another BLE master closer to the BLE slave. Reich et al. [174] and Park et al. [163] focus on the scalability aspects of multiple parallel BLE connections on a single BLE master. Their experimental evaluations show that the communication reliability of individual BLE connections drops when the BLE master has to sustain multiple BLE connections. By efficiently scheduling the individual BLE connections on the master, however, the number of failed BLE transmissions can be significantly reduced [163].

Parts of our research may be used in all of these applications to improve their reliability, scalability, and latency even further. For example, by combining our work with the work from Hussain et al. [99], one could support highly scalable and mobile IPv6-over-BLE applications.

3.2 Adaptation Techniques in Wireless Technologies

In this section, we discuss state-of-the-art parameter adaptation techniques used by other wireless technologies, such as other low-power radio technologies (Section 3.2.1), Wi-Fi communication (Section 3.2.2), or cellular networks (Section 3.2.3).

3.2.1 Low-power Wireless Technologies

This sections summarizes related research on low-power radio technologies to improve their communication reliability (Section 3.2.1.1) and to sustain time-critical data exchange (Section 3.2.1.2).

3.2.1.1 Communication Reliability

One effective way to improve communication reliability in low-power wireless networks is to use channel hopping. For example, channel hopping has been shown to mitigate the effects of external interference and multipath fading on the communication performance of IEEE 802.15.4 [225,226]. One way to improve the communication reliability even further is to adapt the used communication channels to only use channels with good link quality for communication [225]. Accurately estimating the link quality of a low-power channel (*e.g.*, an IEEE 802.15.4 channel), however, is challenging, as low-power wireless links are lossy and experience dynamic changes [15].

Some research works actively measure the noise floor of IEEE 802.15.4 channels to detect if an individual channel is used by other co-located wireless devices [63, 74, 215]. By investigating the statistical properties (*e.g.*, average, standard deviation, number of samples exceeding a threshold) of noise floor measurements on an individual channel, external interference can be detected and the affected channel can be excluded from further data exchange [74]. Other studies use PDR measurements to passively estimate the link quality of individual IEEE 802.15.4 links [70, 121, 122, 131, 188, 225], as PDR-based estimation approaches typically provide a more accurate estimation than approaches solely based on RSSI, noise floor, or IEEE 802.15.4's LQI [120]. For example, the long-term average PDR can then be used to statically choose only the best performing channels for data exchange [225]. Recent PDR measurements can be used in combination with a fixed [70, 131] or an adaptive threshold [121] to detect and blacklist bad channels. Furthermore, multiple link metrics may be combined to estimate the link quality of individual IEEE 802.15.4 channels [86,137,215]. For example, Gomes et al. [86] measure the RSSI of received packets, the noise floor, and the number of duplicate packets to estimate the link quality of IEEE 802.15.4 channels in industrial environments. Tavakoli et al. [215] use noise floor measurements in combination with clear channel assessment (CCA) and the packet reception status to sense if a channel is bad. Moreover, Liu et al. [137] use multiple link metrics in combination with regression models to calculate the success probability of delivering the next packet on a channel.

Another approach to increase communication reliability of low-power wireless technologies, which support different PHY configurations, is to choose a more robust PHY setting. For example, LoRa devices make use of SNR values to calculate the average packet loss caused by fading or interference [197, 231]. Recent SNR measurements can be used in combination with a neural network to calculate the most suitable LoRA PHY transmission rate [231]. The average SNR over recent packet exchanges may be used to adapt the LoRa PHY transmission rate and power incrementally to sustain a given communication reliability while limiting unnecessary energy consumption [197]. Such a step-by-step adaptation of PHY parameters is also used in UWB communication to achieve an energy-efficient and reliable data exchange [88]. Liang et al. [133] even extend IEEE 802.15.4, which typically does not support different symbol coding schemes, in the MAC layer by dynamically using redundant header transmissions and forward error correction on the packet data to improve communication reliability in the presence of Wi-Fi interference.

In this dissertation, we investigate how both of these techniques, adaptive channel management and PHY mode adaptation, can be used in connection-based BLE. First, we design an effective channel management mechanism that only uses link-layer information available in standard BLE radio devices. Hence, our mechanism can be used to talk to any standard-compliant BLE device, as it does not require any additional probing packet exchanges that would violate the BLE specification. Our mechanism passively monitors the PDR of individual BLE data channels and

promptly blacklists bad data channels with an average PDR below a given threshold to improve communication reliability. Second, we design an effective PHY mode adaptation mechanism for BLE that uses recent SNR values to dynamically choose the most suitable PHY setting of the BLE connection. Third, we experimentally show that both of our adaptation mechanisms successfully work in parallel to cooperatively improve the communication performance of a BLE connection while limiting unnecessary energy consumption, as we show in Section 4.1.

3.2.1.2 Communication Latency

There exists a vast number of research works on low-power radio technologies investigating how to sustain a reliable and time-critical data exchange while minimizing the power consumption of embedded devices. For example, research on WirelessHART shows how adapting the task scheduling on the IEEE 802.15.4 radio improves real-time capabilities [58, 138] and can provide probabilistic real-time guarantees [45]. Other research on IEEE 802.15.4 allows to sustain a given maximum communication delay while minimizing the power consumption of devices in single- and multi-hop networks [39, 75, 80, 102, 236, 237]. Unfortunately, most of these works rely on powerful network coordinator devices that monitor the whole network state and centrally optimize the network's communication schedule. For example, Ma et al. [138] use link quality predictions on the network coordinator in combination with nonlinear integer programming to optimize the central transmission schedule of multi-loop control systems. Franchino et al. [80] use network calculus on a central gateway to sustain a given QoS level and system lifetime. Zimmerling et al. [237] optimize the performance of multi-hop networks by using a network-wide performance model in combination with mixed-integer nonlinear programming. All of the works above, however, only focus on sustaining latency and reliability bounds within the low-power network and cannot be used to exchange data with a cloud server within given latency and reliability requirements.

Only some low-power radio studies have investigated communication across multiple networks. Betzler et al. [21] design an adaptive retransmission timeout estimator for CoAP that can cope with congestion in the IEEE 802.15.4 network. Nodes using their approach are able to increase data throughput and reduce the time to process traffic bursts in a network compared to non-adaptive CoAP. Kumar et al. [124] show how to fine-tune the behavior of TCP so that low-power nodes can make use of TCP when talking to cloud servers to achieve efficient data exchange. Unfortunately, these works do not investigate how low-power nodes can sustain a given end-to-end reliability and latency when communicating with devices on the Internet.

In this thesis, we fill this research gap and show how individual BLE nodes can autonomously adapt their parameters to exchange data with other devices within given end-to-end reliability and latency bounds while minimizing their energy consumption. Our estimation and adaptation approach runs directly on the BLE nodes and does not require any central network coordinator, making our work fully compliant with the end-to-end principle of IP. Off-the-shelf BLE nodes can make use of our work to sustain end-to-end dependability bounds when communicating with peer devices in the local BLE network (*e.g.*, a smartphone collecting data) or outside the BLE subnet (*e.g.*, a cloud server on the Internet), as we show in Chapter 6.

3.2.2 Wi-Fi Technology

Wi-Fi, as specified by IEEE 802.11, is the most popular wireless Local Area Network (LAN) technology and, like BLE, also uses the 2.4 GHz (in combination with other frequencies) ISM band for data exchange. Compared to low-power wireless technologies, Wi-Fi devices have a drastically higher data throughput and lower latency at the cost of a significantly higher power consumption [81].

In Wi-Fi networks, one or multiple Wi-Fi clients talk to a central Wi-Fi access point (AP) using a single Wi-Fi channel. Typically, Wi-Fi devices operate in environments, such as urban housing or conference venues, where a vast number of different Wi-Fi APs share the same frequencies and need to sustain high throughput and low latency. To optimize the performance of individual Wi-Fi networks, co-located APs need to coordinate their individual channel selection to mitigate interfering with each other. In settings, where APs are centrally managed, *e.g.*, sports stadiums, conferences, and public transport, centralized channel coordination provides the best achievable data throughput [17]. In other situations where such centralized channel allocation is not possible, *e.g.*, in housing environments where every apartment has its own AP, Wi-Fi devices need to use their own channel selection algorithm, *e.g.*, based on noise floor measurements across all available Wi-Fi channels, to select the least congested channel [139]. Furthermore, Wi-Fi devices use recent noise floor and RSSI measurements to dynamically adapt their transmission power and CCA thresholds to minimize interference amongst co-locates Wi-Fi networks [20, 114].

Since the release of IEEE 802.11ac, Wi-Fi devices make use of Dynamic Bandwidth Channel Access (DBCA) to optimize their spectrum usage [20]. Using DBCA, Wi-Fi devices actively measure the noise floor of the used channel before every transmission and adapt the used packet rate accordingly [125, 164]. For example, a Wi-Fi device may detect that only 40 MHz of bandwidth on the used channel is currently unoccupied, *i.e.*, has a noise floor level below the current CCA threshold. In this case, the Wi-Fi device sends the next packet with a rate of only 40 MHz, filling the current spectrum gap, although it may support a rate of up to 160 MHz.

To improve the spectrum usage of Wi-Fi even further, IEEE 802.11ax specifies that devices make use of orthogonal frequency-division multiple access (OFDMA), a modulation technique widely used in cellular networks [20, 56, 114]. OFDMA dynamically splits the available Wi-Fi channel width into multiple narrow channels of variable width. These narrow channels are used by the AP in combination with multiple AP antennas to communicate with different Wi-Fi clients in parallel to increase overall throughput.

To summarize, state-of-the-art Wi-Fi devices use powerful modulation techniques in combination with channel management and rate adaptation to optimize their data throughput and communication latency. Furthermore, Wi-Fi devices can change their PHY parameters (*e.g.*, the used transmission data rate) on a per-packet basis. Unlike Wi-Fi, connection-based BLE uses fixed-width channels and requires that parameters are negotiated between two connected devices, which introduces significant adaptation delays (see Section 2.1.4). The adaptation techniques of Wi-Fi can, therefore, not be directly applied to connection-based BLE. However, we discuss that our BLE channel management using passive PDR measurements provides a more reliable BLE connection compared to a Wi-Fi-like approach using noise floor measurements, as we show in Chapter 4.

3.2.3 Cellular Technologies

Another very popular approach to connect embedded devices to the Internet is using cellular technologies, especially 4G and the new 5G broadband cellular network standards. Compared to low-power radio technologies, cellular devices need existing infrastructure (*e.g.*, cellular towers) to function, are not license-free, and have a significantly higher power consumption [5, 161].

In cellular networks, clients (*i.e.*, smartphones) use the regulated frequency bands to communicate with base stations (*i.e.*, cellular towers). Similar to modern Wi-Fi networks, 4G networks make use of OFDMA to increase the spectrum efficiency of their networks [157]. 5G networks use the even more advanced nonorthogonal multiple access (NOMA) approach to take full advantage of the available frequency spectrum. Using NOMA, 5G devices split the available frequency band into subchannels and use power-domain and code-domain multiplexing to allow multiple devices to share the same subchannels for communication simultaneously [228].

As cellular networks operate on regulated frequencies, cellular devices usually do not have problems with external radio interference by other co-located radio technologies. However, cellular devices may experience co-channel interference at the boundaries of cellular cells [151]. Co-channel interference happens when a client is located between two cellular towers that share the same frequencies. In this setting, the two towers may interfere with each other and the client is not able to successfully receive data [223]. Such co-channel interference is one of the major issues of large-scale 4G and 5G systems [151, 168]. One technique to mitigate co-channel interference is to use effective centralized cell planning and selection if multiple frequencies are available. A telecommunication provider may use multiple frequencies and centrally assign them to minimize the chance of two nearby cells using the same frequency [223]. Other effective techniques to mitigate co-channel interference are dynamically changing the used transmission power, adapting the used modulation and coding scheme, coordinating packet transmission scheduling, and using beamforming [19, 123].

Cellular devices can dynamically change their communication parameters at runtime to improve communication performance. For example, 4G devices make use of the received signal strength and analytical path loss models to dynamically choose a suitable transmission power [195]. 5G devices may use SNR measurements in combination with sophisticated 3D-spatial channel models to adapt the used transmission power [19]. 4G devices use complex channel information, consisting of a channel quality indicator, a precoding matrix indicator, and a rank indicator, to adaptively control the modulation and coding scheme on a per-channel and per-packet basis to improve communication reliability and data throughput [123]. Likewise, 5G devices may use sophisticated channel quality information passed through a low-pass infinite impulse response filter to estimate link quality [168]. Cellular devices may use channel quality information in combination with analytical models and complex numerical optimization techniques to adapt the PHY parameters of upcoming packet transmissions [18]. Furthermore, 4G devices incrementally increase the used error correction information of packet transmissions when packets are not successfully acknowledged by the peer device [123].

Moreover, URLLC allows 5G devices to exchange small payloads with an end-to-end latency below 1ms and a success probability above 99.999% [169]. Exemplary use cases of URLLC are wireless industrial control and automation, safety-critical vehicle-to-vehicle communication, and real-time tactile Internet services. To achieve low latencies, 5G devices adapt the flexible frame structure of 5G packets and use advanced transmission scheduling policies. High reliability is

achieved by using enhanced dynamic retransmission schemes and by having multiple antennas on the transmitter and receiver to simultaneously send on different channels [132]. Furthermore, 5G base stations cooperatively use data duplication and redundant packet exchanges to achieve the goals of URLLC [187], even when sharing the same channel with other 5G traffic [168, 170].

Narrowband IoT (NB-IoT) is a fairly new cellular technology that focuses on connecting low-power IoT devices wirelessly via the cellular network to the Internet. NB-IoT is an extension of the existing LTE cellular standard, but optimizes aspects of its specification to support IoT devices that may experience poor cellular coverage, infrequently exchange small packets, and do not have a continuous power supply [145]. Due to these optimizations, NB-IoT devices that infrequently transmit short data packets (*e.g.*, 50 bytes every 2 hours) may run for several years on a single battery charge [172]. The drawbacks of NB-IoT, however, are its maximum achievable data rate of approximately 90 Kbits per second [141] and its highly variable communication delay, *i.e.*, NB-IoT packets may experience delays of several minutes [145].

Overall, state-of-the-art cellular networks use sophisticated channel management and link adaptation approaches in combination with powerful base station antenna arrays and modulation schemes. These techniques, together with licensed frequencies, allow an effective and efficient use of the available frequency spectrum and lead to high data throughput and low transmission latencies. In this dissertation, we show that BLE devices can use simple link quality estimation techniques for effective channel management and PHY mode adaptation to sustain a given communication reliability in the crowded 2.4 GHz ISM band. Our approach does not require multiple BLE antennas, sophisticated channel models, or duplicate and redundant transmissions to multiple BLE routers. Nevertheless, some of the above techniques, such as an adaptive error correction or advanced transmission scheduling, may further improve the performance of BLE communication.

3.3 Time-critical and Reliable Internet Communication

Finally, we summarize research on reliable and time-critical Internet communication. Similar to the sections above, we highlight the differences between our work and existing research.

3.3.1 Estimating Loss and Delay on the Internet

How devices can capture and mitigate loss and delay across the Internet path has been researched in a large body of work. For example, the round trip time of TCP messages over wireless links can be estimated using multiple nested Kalman filters [105]. Loss and delay of UDP traffic across the Internet can be modeled in real-time using machine learning [180] or via two-level Markov models [69]. Furthermore, Domain Name System (DNS) exchanges may be used to estimate loss and delay between two hosts on the Internet [90, 224].

The work in this dissertation follows another approach and adapts the round-trip time (RTT) estimation of TCP proposed by Jacobson [103, 167] to the particular requirements of low-power wireless communication. In Chapter 6, we show that our simple approach using infrequent RTT measurements is able to efficiently and accurately estimate the maximum end-to-end loss and delay across the Internet. Our simple estimation approach is very energy-efficient and requires only little processing and memory resource, making it applicable to almost any constrained device.

3.3.2 Quality of Service (QoS)

Typically, IP networks provide best-effort data exchange, where no packets are prioritized over others. To allow for differential treatment of time-critical IP packets, Internet Protocol version 4 (IPv4) and IPv6 foresee different QoS mechanisms that allow for packet prioritization [214]. The two most popular QoS approaches in IP networks are Integrated Services (IntServ) and Differentiated Services (DiffServ) [140]. Using IntServ, a sender uses signaling packets to reserve and maintain network resources, *e.g.*, a given data rate or latency, on a per-flow basis across the whole network path to the receiver. However, IntServ has scalability issues, as it requires all routers to understand its signaling packets and maintain the state of every QoS flow, and is therefore seldomly used for traffic across the Internet [78]. DiffServ, in contrast, requires no specific end-to-end connection between sender and receiver but defines a QoS class in the header of every IP packet. The QoS class is mapped to specific per-hop behaviors on the individual link layer technologies, such as different packet transmission priorities in Wi-Fi [56, 153], to achieve the desired QoS [140]. This makes DiffServ scalable, flexible, and interoperable, but also means that it cannot guarantee a specific end-to-end QoS across the Internet.

More recently, Software-Defined Networking (SDN) is used in IP networks to sustain specific QoS requirements. With SDN, the network's control and data planes are decoupled and the network state and intelligence are centrally bundled and controlled to dynamically optimize the network performance [111]. Therefore, networks may use SDN to provide end-to-end QoS in small and large-scale networks [89, 176]. To use SDN, however, one needs to have control over all routers in a network, making SDN not feasible for sustaining QoS over the Internet.

In this dissertation, we show how a BLE link layer can achieve different QoS levels when exchanging IPv6 packets with devices on the Internet. Furthermore, our work on sustaining end-to-end loss and delay is very flexible, as it does not rely on any specific QoS mechanism of IPv6. Using a specific QoS mechanism in combination with the work in this thesis will improve communication performance even further.

3.3.3 Real-time Protocols

Sustaining real-time communication is important for many popular applications on the Internet, such as audio and video streaming. Therefore, there exists an abundance of different protocols that allow a reliable and timely transfer of data over the Internet. For example, application protocols, such as Real-Time Transport Protocol (RTP), are successfully used to stream audio and video over the Internet [182, 221]. Advanced transport protocols, such as Sprout [227], Verus [233], and PCC [60] reduce network congestion and allow high data rates and low latency in wired and cellular links [229]. More recently, the real-time protocols Time Sensitive Network (TSN) and Deterministic Networking (DetNet) have been released to achieve a deterministic end-to-end latency in best-effort IP networks [230]. The TSN standards solely focus on extending link layer technologies, such as Ethernet, by advanced flow synchronization, management, and control to create reliable and low-latency communication [153]. The DetNet standards build on top of TSN and define how the network layer can effectively route packets to achieve a reliable data exchange with a deterministic latency [57].

These protocols are able to sustain real-time data exchange across the Internet, but require powerful processing capabilities on every end point [182, 221, 227]. Some of these approaches even

require changing the behavior of intermediate routers [57, 153]. This makes these protocols not suitable for low-power and resource-constrained embedded nodes, such as BLE devices. Therefore, in this dissertation, we focus on how constrained devices can sustain end-to-end requirements while operating on a limited energy budget.

3.3.4 Tactile Internet

Recently, research on the Tactile Internet focuses on sustaining minimal end-to-end latency and maximum reliability on the Internet [77]. These works mostly use 5G and URLLC to reliably reach round trip latencies below 1 ms to enable tactile and visual feedback control across the Internet. In addition to the innovations in the 5G link layer, Tactile Internet applications also require changes to other communication layers and the employed infrastructure to sustain the given requirements. For example, Tactile Internet applications require new transport and application protocols, in combination with novel QoS approaches, to achieve low communication latency [108]. The core network of such applications further needs to be adapted to reduce the protocol overhead of individual packets and to support moving the used cloud server physically near to the application [14, 196]. Furthermore, sophisticated data reduction and compression algorithms in combination with AI-based content caching improve communication performance while optimizing the required data rate [196].

Contrary to these works, the work in this dissertation focuses on sustaining less stringent end-to-end dependability requirements on resource-constrained nodes, while minimizing the power consumption of the constrained node devices.

4

Improving the Performance of BLE Connections

In this chapter, we answer our first research question (RQ 1) and show how the parameters of connection-based BLE can be dynamically adapted to optimize communication performance. In Section 4.1, we discuss how to improve the link-layer reliability of connection-based BLE with an effective BLE channel management and PHY mode adaptation. In Section 4.2, we present how off-the-shelf BLE devices can estimate and control the communication latency of individual data transmissions. This chapter is based on the Publications B, C, and D included in this dissertation.

4.1 Increasing the Reliability of BLE Connections

In this section, we focus on increasing the link-layer reliability of a BLE connection while minimizing the power consumption of the BLE slave. To improve the link-layer reliability, the BLE specification foresees two different approaches: channel management and PHY mode adaptation. In this dissertation, we focus on implementing these two approaches on off-the-shelf BLE devices.

4.1.1 Experimental Study of BLE Link-Layer Reliability

This section summarizes the results of our extensive experimental study of the reliability of the BLE link-layer under different conditions. In our work, we estimate the link quality of individual BLE data channels on the BLE master, which has several advantages over measuring the link quality on the BLE slave. First, the master is usually less power-constrained and could afford to actively exchange probing packets. Second, other than the BLE slave, the master receives acknowledgments to link-layer packets within the same connection event (see Section 2.1.4). Third, for all BLE devices supporting BLE version 5.2 and below only the master can adapt the used data channel map C_{map} of a BLE connection[4].

4.1.1.1 BLE Link Quality Metrics

In our experimental study, we focus on link quality metrics that are available on standard BLE radios allowing link-layer access and we only make use of passive link monitoring (as described

[4] The most recent BLE specification version 5.3 [27] released in July 2021 also allows BLE slave devices to adapt the data channel map of a BLE connection. Because this feature is not yet supported by existing BLE devices, this thesis only shows how a BLE master device can perform effective BLE channel management.

in Section 2.2.2), *i.e.*, we investigate only link-layer information that is available through existing BLE link-layer packets. Introducing additional active probing packets is not suitable for our approach, as this would violate the BLE specification [26] and would break interoperability with other off-the-shelf BLE devices. Based on these constraints, we consider the following metrics:

Noise floor. We measure the noise floor of a specific BLE data channel by reading the RSSI when the radio does not exchange packets. The noise floor provides us with an indication if the data channel is also used by other nearby wireless technologies, such as Wi-Fi.

Signal-to-noise ratio (SNR). We measure the SNR of every successfully-received link-layer packet, by reading a packet's RSSI and subtracting the current noise floor on the used data channel. This metric provides us with information on the packet signal strength on individual data channels.

Packet delivery ratio (PDR). We further measure the PDR of every link-layer packet exchange. Therefore, we adapt the work in [94] to be used in BLE connections. Because we are unable to arbitrarily probe individual channels in connection-based BLE, we use existing link-layer header fields to calculate the PDR of link-layer exchanges. The PDR measures the round-trip reliability of individual link-layer transmissions issued by a master and is computed as:

$$PDR = \frac{\#ACK(S \rightarrow M)}{\#TX(M \rightarrow S)}, \tag{4.1}$$

where $\#TX(M \rightarrow S)$ is the number of issued link-layer transmissions from master to slave and $\#ACK(S \rightarrow M)$ is the number of received valid link-layer acknowledgments from the slave. Every link-layer transmission is either successful ($PDR = 100\%$) or unsuccessful ($PDR = 0\%$).

4.1.1.2 Measuring BLE Link Quality

We experimentally study the above link quality metrics in multiple scenarios to investigate the link-quality information that each metric can provide. We perform our study in a wireless testbed located in a university laboratory, where we have full control over the RF environment. For the experiments, we use one Nordic Semiconductor nRF52 device [159] running the Zephyr RTOS [217] as BLE master and connect it to another nRF52 device running the Zephyr RTOS acting as BLE slave. The slave periodically sends GATT notifications with a length of 27 bytes to the master using a BLE connection interval of 50 ms and a BLE slave latency of 0. In these experiments, master and slave have a free line of sight and are approximately 10 m apart. The master measures and logs the PDR and SNR of every link-layer exchange as well as the noise floor of every BLE data channel after every connection event. Furthermore, the master uses the 1M BLE PHY mode and does not make use of channel blacklisting, which is the default behavior of Zephyr running on an nRF52 device. A detailed description of our experimental setup can be found in Publication D.

The subfigures in Figure 4.1 show the behavior of the BLE link quality metrics under changing antenna attenuation, Classic Bluetooth interference, and Wi-Fi interference. The noise floor shown in Figure 4.1 is the maximum noise floor value recorded on each channel within every second. Similarly, the SNR plots show the average SNR value within a second on every channel. However, when the master does not successfully receive a link-layer acknowledgment from the slave, the SNR of this packet exchange is discarded (marked in brown). In addition to the SNR values on each channel, we also calculate the average SNR (Avg. SNR) across all used BLE channels within

a second. The PDR values shown in Figure 4.1 show the average PDR within a second per channel. Following the approach by Srinivasan et al. [210], we classify channels with a PDR below 10% as bad, channels with a PDR between 10% and 90% as intermediate, and channels with a PDR of 90% or above as good. Whenever a data channel is not used within a second, its SNR and PDR are marked in white in Figure 4.1.

Changing antenna attenuation. First, we measure how the link quality of the BLE data channels is affected when the radio signal is attenuated, *e.g.*, due to obstacles blocking the line of sight or an increasing communication distance. To mimic changing signal attenuation, we use a programmable attenuator [147] connected to the slave's antenna that allows fine-grained control over the antenna attenuation on the slave. Figure 4.1(a) shows the measured noise floor, the average SNR across all used data channels, the SNR per data channel, and the PDR per data channel under gradually changing antenna attenuation. The antenna attenuation on the slave starts at 0 dB and, between the time of 0 to 30 seconds, linearly increases to 10 dB. The antenna attenuation stays at 10 dB for 120 seconds and then linearly goes back to 0 dB. This change in attenuation can clearly be seen in the average SNR and the PDR of some data channels in Figure 4.1(a).

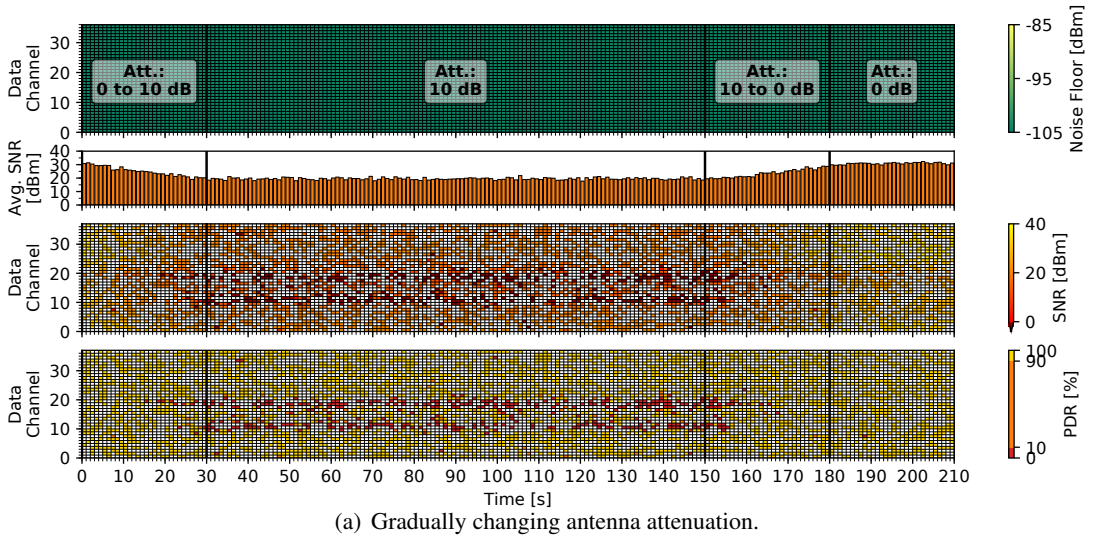
Figure 4.1(a) shows that noise floor measurements are not able to detect link-layer problems due to a weak signal strength. The average and individual SNR values detect the changing signal strength and individually unsuccessful packet exchanges but do not detect when a valid link-layer packet was received, but its checksum is wrong. Only the PDR is successfully able to capture all link-layer packet loss caused by weak signal strength.

Classic Bluetooth interference. Next, we measure the link quality of BLE data channels under co-located Classic Bluetooth interference, which also uses the 2.4 GHz ISM band and employs frequency hopping. To generate Bluetooth interference, we use two pairs of Raspberry Pi 3B (Pi3) devices in our wireless testbed and let each Pi3 pair exchange Bluetooth RFCOMM packets at a data rate of approximately 725 kbps. As shown in Figure 4.1(b), we start the Bluetooth interference 30 seconds after the beginning of our experiment and interfere for roughly 120 seconds. In this experiment, we use a fixed antenna attenuation of 0 dB.

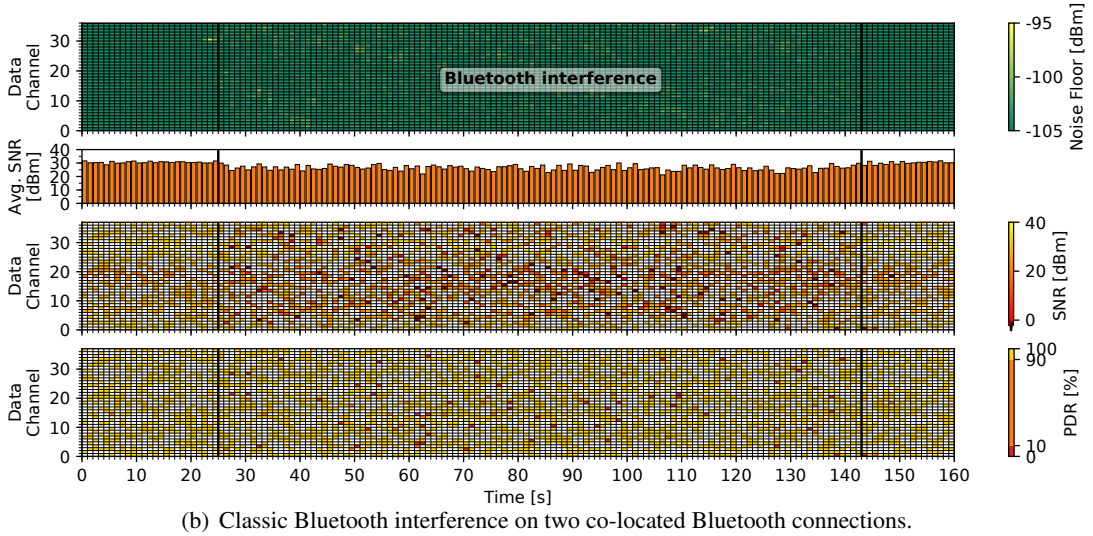
Figure 4.1(b) shows that the noise floor measurements barely detect the external interference. The average and individual SNR measurements are able to detect the Bluetooth interference, but may underestimate the link quality, as several SNR values of successful packet exchanges are very low ($SNR < 5dBm$). As in the previous scenario, only the PDR is able to accurately capture link-layer packet loss caused by co-located Classic Bluetooth communication.

Wi-Fi interference. We measure the link quality of BLE data channels under Wi-Fi interference. To generate Wi-Fi interference, we run JamLab-NG [185] on one of the Pi3s in our wireless testbed, which is located near the BLE slave. We generate Wi-Fi packets with a length of 1500 bytes on Wi-Fi channel 11 every 10 ms and use a transmission power of 30 mW. Like in the previous scenario, we use a constant antenna attenuation of 0 dB and start the interference 30 seconds after the beginning of the experiment and interfere for approximately 120 seconds.

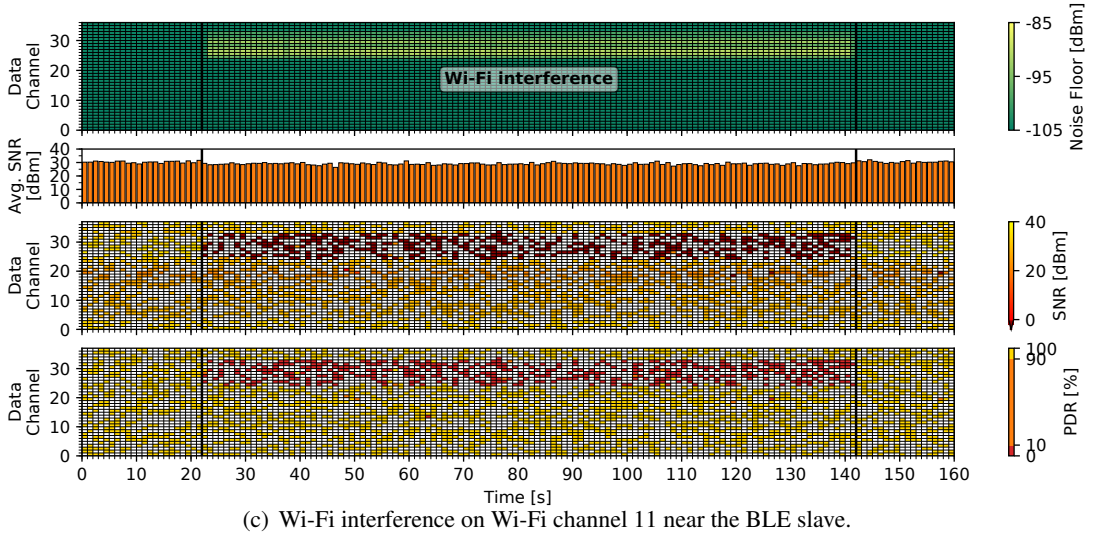
Figure 4.1(c) shows that the Wi-Fi interference significantly decreases the link-layer reliability of the BLE connection. In this case, the noise floor measurements are able to detect the Wi-Fi interference. Similar to Classic Bluetooth interference, the SNR measurements detect unsuccessfully received link-layer packets, but do not detect packets with an invalid checksum. Again, only PDR measurements accurately detect all link-layer errors caused by Wi-Fi interference.



(a) Gradually changing antenna attenuation.

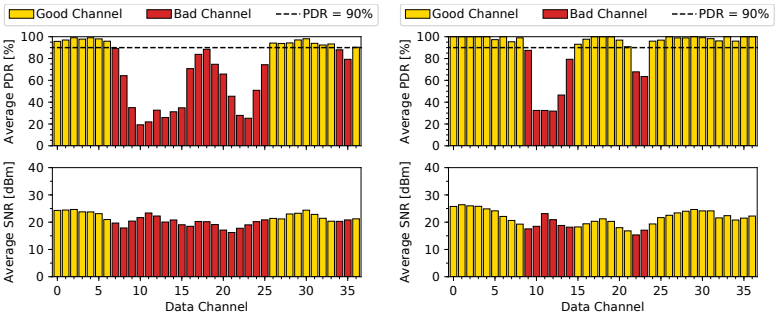


(b) Classic Bluetooth interference on two co-located Bluetooth connections.

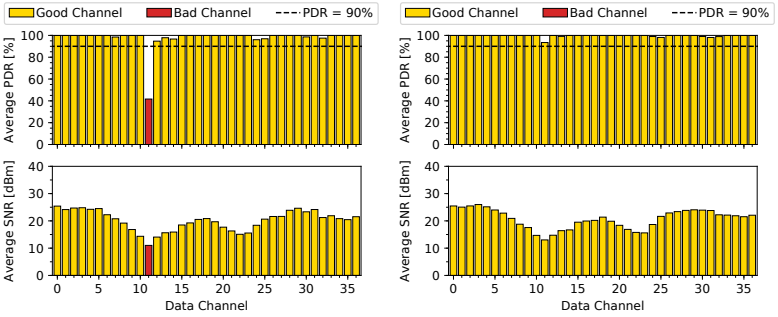


(c) Wi-Fi interference on Wi-Fi channel 11 near the BLE slave.

Figure 4.1: Link quality of a BLE connection under three different conditions. From top to bottom, the figures show the noise floor per channel, the average SNR across all used channels, the SNR per channel, and the PDR per channel. Adapted from Publication D.

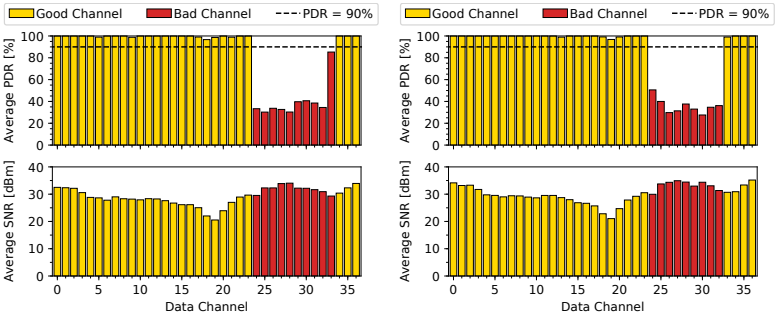


(a) 2M PHY Mode. (b) 1M PHY Mode.

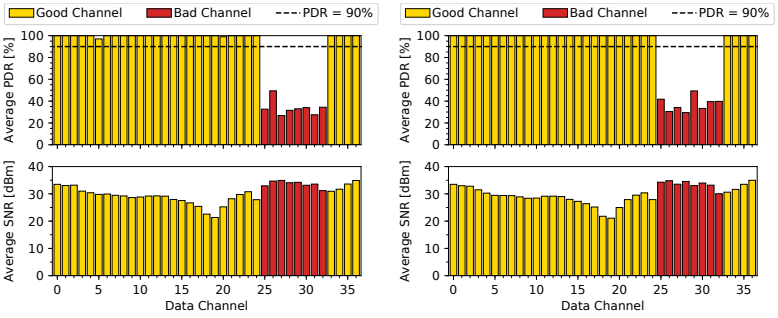


(c) Coded PHY (S2) Mode. (d) Coded PHY (S8) Mode.

Figure 4.2: Average PDR and SNR of all data channels using different PHYs and an antenna attenuation of 10 dB. We see that the used PHY mode of the BLE connection significantly affects the overall link-layer reliability in this scenario. Adapted from Publication D.



(a) 2M PHY Mode. (b) 1M PHY Mode.



(c) Coded PHY (S2) Mode. (d) Coded PHY (S8) Mode.

Figure 4.3: Average PDR and SNR of all data channels using different PHYs under Wi-Fi interference on Wi-Fi channel 11. We see that the used PHY mode of the BLE connection does not significantly affect the overall link-layer reliability in this scenario. Adapted from Publication D.

4.1.1.3 Using Different PHY Modes

After evaluating how individual BLE data channels are affected by different scenarios, we investigate how the used PHY mode, which defines the modulation and coding parameter of BLE communication (see Section 2.2.1), affects the reliability of connection-based BLE. Therefore, we repeat the experiments above with all four PHY modes of BLE 5 and evaluate their performance.

Antenna attenuation. Figure 4.2 shows the average PDR and average SNR of each BLE data channel for the four different PHY modes, while experiencing an antenna attenuation of 10 dB. Channels with an average PDR below 90% are classified as bad (shown in red) and channels with an average PDR of 90% or above are classified as good (shown in yellow). If the BLE connection experiences a weak signal strength, the used PHY mode has a significant effect on the overall link-layer reliability. Because the Coded S2 and S8 PHYs make use of FEC and symbol coding, they are able to sustain a high PDR under weak signal strength, even for channels with a low average SNR. For example, all 37 BLE data channels have a good link quality when using the most robust Coded S8 PHY compared to only 15 good channels when using the 2M PHY mode. The data in Figure 4.2 suggest that switching to a more robust PHY mode significantly increases link-layer reliability when experiencing a low signal strength.

Radio interference. Figure 4.3 shows the average PDR and average SNR of each BLE data channel when Wi-Fi interference is generated near the BLE slave. Unlike the previous experiment, the PHY mode used by the BLE connection does not significantly improve the link-layer reliability under external interference. All four PHY modes experience similar link quality problems on the data channels that overlap with the Wi-Fi interference. The 2M PHY provides an average PDR of 83%, while the most robust Coded S8 PHY provides an average PDR of 86% across the whole BLE connection. Switching from the least robust to the most robust PHY would only provide the BLE connection with a 3% improvement on its average PDR.

4.1.1.4 Lessons Learned

Based on our experimental results, we draw the following conclusions that inform the design of our BLE channel management and BLE PHY mode adaptation. To the best of our knowledge, our experimental study is the first investigating the performance of different BLE link quality metrics in real-world scenarios.

Noise floor. As expected, noise floor measurements on BLE channels are able to successfully detect link-layer loss caused by external interference, *e.g.*, caused by Classic Bluetooth or Wi-Fi. Nevertheless, noise floor measurements are not able to detect link-layer loss caused by weak signal strength, *e.g.*, due to a large communication distance or low transmission power, or fading effects.

Signal-to-noise ratio (SNR). Individual SNR values detect loss on BLE channels when a link-layer packet has not been successfully received, however, the SNR does not detect link problems indicated by invalid checksums of successfully received packets. Moreover, individual SNR values are not a good link quality indicator, as successful link-layer exchanges may have a low SNR value.

As discussed in Section 4.1.1.2, whenever the BLE master does not successfully receive a link-layer acknowledgment, the SNR value of this link-layer packet exchange is discarded. The average SNR of recent data exchanges across all used BLE channels, therefore, only includes the SNR

values of successful link-layer exchanges. This average SNR accurately measures the BLE signal strength and may be used to detect link-layer problems caused by a weak BLE signal and is a suitable metric for detecting when the used PHY mode needs to be adapted.

Packet delivery ratio. The PDR values are accurately able to detect any link-layer packet loss across all of our experimental scenarios. This makes the PDR the most suitable metric to detect poor data channels to blacklist them.

Next, we show how to improve the BLE link-layer reliability by using these lessons.

4.1.2 Channel Management

In this section, we present our effective BLE channel management mechanism that is fully compliant to the BLE specification and significantly improves the link-layer reliability of BLE connections, as we show in Section 4.2.4. Using the findings from Section 4.1.1.4, our channel management passively monitors the PDR of individual BLE data channels by observing ongoing data exchanges between the BLE peer devices. Based on recent PDR measurements, every BLE data channel is classified into either good or bad (see Section 4.1.2.1). Whenever a channel is classified as bad, our channel management immediately blacklists the channel to exclude it from further communication (see Section 4.1.2.2). In case a large portion of BLE data channels has been blacklisted, we whitelist (re-enable) all available data channels to probe their link quality using ordinary data exchanges and get a fresh number of good channels (see Section 4.1.2.3).

Our simple and effective channel management mechanism can be adapted to other wireless technologies that provide PDR measurements of individual data exchanges. With our channel management, devices can passively, *i.e.*, without additional energy consumption, monitor the used data channels to promptly detect bad channels. As our channel management mechanism only relies on PDR measurements, it successfully works in parallel to other communication parameter adaptation mechanisms, as we discuss in Section 4.1.1.4 and experimentally show in Section 4.2.4.

4.1.2.1 Detecting Bad Channels at Runtime

First, we need to find the most effective approach to detect bad BLE data channels at runtime. In our work, we focus on a simple rule-based channel classification approach (see Section 2.2.4), where we mark an individual channel as bad whenever its link quality drops below a fixed threshold. Such rule-based approaches are common in other wireless technologies because they require very few memory and computing resources while being very effective. In contrast to other work, however, we investigate which individual link-layer metric or combination of metrics provide the best channel classification, which we discuss in detail in Publication D.

We reuse the experimental traces from Section 4.1.1 to find the most suitable channel classification approach. For every experimental scenario, we simulate the performance of different channel classification approaches, based on one or multiple BLE link quality metrics, and calculate the resulting overall PDR (PDR_{conn}) and the number of active channels (C_{active}) at the end of each experiment. We step through every measured trace to calculate the overall PDR of the BLE connection as the average of the individual PDR measurements. Simultaneously, we simulate the behavior of different channel classification approaches. Whenever the current classification approach detects that a channel is bad, we mark this channel as blacklisted and do not include any

subsequent link-layer exchanges into our overall PDR calculation.

To find the most suitable channel classification approach, we test six different classification approaches in three different scenarios. We start by evaluating three classification approaches that use individual PDR measurements of packet transmissions and blacklist a channel if the channel's average PDR drops below a given threshold. Based on our data, we see that blacklisting a BLE channel when its average PDR across the 20 most recent packet exchanges drops below 95% provides the best trade-off between PDR_{conn} and C_{active}. In addition to the first PDR-based classification approaches, we also evaluate if additional noise floor or SNR information about a channel improves channel classification. However, our simulations show that blacklisting a channel when its average PDR drops below 95% is the most suitable BLE channel classification approach across all three scenarios and that additional information does not improve channel classification.

4.1.2.2 Blacklisting Data Channels

In our approach, the BLE master immediately updates the used BLE channel map (C_{map}) of the BLE connection by issuing a `LL_CHANNEL_MAP_IND` request to the slave when a bad channel is detected (see Section 2.1.4). The new C_{map} defines if a channel is active, *i.e.*, will be used for further connection events or is blacklisted, *i.e.*, won't be used for any upcoming connection events until being re-enabled again. Updating the C_{map} of a BLE connection, however, requires a mandatory delay of at least six connection events (as discussed in Section 2.1.4), which means that a blacklisted channel may be used in another connection event before the new C_{map} takes effect. If a recent C_{map} update is still pending, *i.e.*, has not been acknowledged by the slave, the master waits for the update to finish and then immediately issues a new C_{map} update.

Whenever a channel is blacklisted, it won't be used for packet exchanges, which means that we are not able to further estimate its link quality following our approach. Hence, any existing information of a blacklisted channel is immediately cleared, as it may have already expired. Only when a channel is whitelisted again, we collect fresh information to estimate its quality.

4.1.2.3 Whitelisting Data channels

As our channel management uses passive link monitoring to classify BLE data channels into good and bad, we do not get valid link quality information of channels that have been blacklisted (as discussed in Section 2.2.2.1). This means that we cannot detect if a previously blacklisted channel has improved in link quality and can be whitelisted again. Using additional probing packets to measure the link quality of blacklisted channels is infeasible, as it breaks interoperability with other BLE devices. Whitelisting data channels after some timeout is also not practical, as the timeout significantly depends on the current RF environment. Therefore, we follow a rule-based approach with a fixed threshold (as discussed in Section 2.2.4) to whitelist all available BLE data channels. This approach allows us to always have a diverse set of data channels for communication, while still requiring only passive link measurements.

In our approach, we trigger a full whitelisting process whenever the number of active BLE channels of a BLE connection drops below the threshold C_{min}. Whenever the BLE master detects that the number of active BLE channels drops below C_{min}, the master runs through the following whitelisting process. First, the master temporarily changes the BLE connection interval to a faster

setting to allow faster channel probing and to avoid significantly impacting the latency of application traffic. Second, the master re-enables all 37 BLE data channels and probes their individual average PDR using regular BLE connection events. This probing phase lasts for t_{probe}, in which every data channel is probed approximately $S_{channel}$ times. After probing, the master calculates the average PDR of each channel, blacklists any channels with poor quality, and reverts to the original BLE connection interval. At the end of the whitelisting process, the BLE connection has a new channel map with only good channels enabled.

4.1.3 PHY Mode Adaptation

Our BLE PHY mode adaptation mechanism presented in this section allows BLE devices to sustain a specified link-layer reliability while minimizing their power consumption. Based on our experimental results in Section 4.1.1, our PHY mode adaptation mechanism passively monitors the SNR values of recent packet exchanges and uses a rule-based adaptation mechanism to find the most suitable PHY mode. Using the SNR for PHY mode adaptation has multiple benefits. First, the average SNR across all used BLE data channels accurately measures the BLE signal strength of the BLE connection. Second, the average SNR (which includes only individual SNR values from successful link-layer exchanges, as discussed in Section 4.1.1.4) is not significantly affected by external interference, which means that our PHY mode adaptation is independent of our BLE channel management presented above. BLE devices can, therefore, use only one of these two mechanisms or can use both mechanisms in parallel to cooperatively improve the link-layer reliability of connection-based BLE.

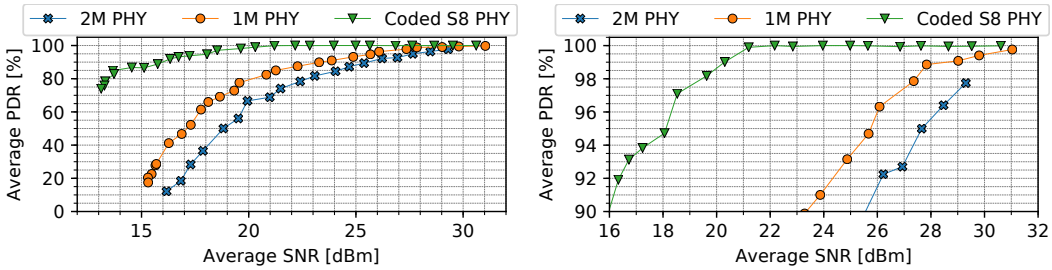
Other wireless technologies may use our PHY mode adaptation mechanism to sustain a given link-layer reliability while minimizing energy consumption. Especially wireless technologies that are able to adapt multiple communication parameters (see Section 2.2.1) may use our PHY mode adaptation, as it does not interfere with other adaptation approaches.

4.1.3.1 Filtering SNR Measurements

To predict the SNR of future link-layer packet exchanges, we make use of a moving average filter (as discussed in Section 2.2.2.2) on recent SNR values on all used BLE data channels of the BLE connection. We reuse the experimental setup from Section 4.1.1 and record the link-layer reliability of the different PHY modes for two different antenna attenuation settings. We step through the recorded data to evaluate which window length (W_{SNR}) provides the best predictions of the average SNR of future link-layer exchanges. Our data suggest that a W_{SNR} = 25 provides the most accurate predictions of the SNR of upcoming 100 link-layer packets.

4.1.3.2 Choosing the Most Suitable PHY

Based on the filtered SNR measurements, a BLE device can select the most suitable PHY mode. Our approach selects the most energy-efficient PHY mode that allows to sustain a given link-layer reliability (PDR_{min}). We reuse the data from Section 4.1.3.1 to find the relationship between the average measured SNR and the PDR of a BLE connection, shown in Figure 4.4. Note that we only investigate three different PHY modes in our evaluation, because the symbol coding of the Coded



(a) Whole PDR range.

(b) PDR range between 90 % and 100 %.

Figure 4.4: Relationship between average PDR and the average SNR of a BLE connection for different BLE PHY modes. Adapted from Publication D.

PHY (either S2 or S8) is statically decided by a BLE device and cannot be changed at runtime. To support the highest possible reliability, we statically choose the Coded S8 PHY for our devices.

Figure 4.4 shows that the Coded S8 PHY, indeed, sustains the highest PDR for any given average SNR. This increased reliability comes with the cost of significantly higher power consumption caused by longer radio times. A slave using the Coded S8 PHY consumes on average $581.79\mu A$, while a slave in the same conditions but using the 1M or the 2M PHY consumes $407.93\mu A$ and $397.07\mu A$, respectively. When combining our current measurements with the data in Figure 4.4, we can see that the 2M PHY has a significantly lower link-layer reliability, but only a slightly higher current consumption, compared to the 1M PHY mode. Hence, we argue that the 2M PHY should not be used to sustain a given PDR_{min}, when data throughput is not an issue. Therefore, our BLE PHY mode adaptation only makes use of the Coded S8 PHY when the conditions are harsh and otherwise uses the 1M PHY for communication.

4.1.3.3 Adapting the PHY Mode

Our approach uses rule-based adaptation (as discussed in Section 2.2.4) to find the most suitable PHY mode of the BLE connection. Based on the data in Figure 4.4 and the user-specified PDR_{min}, our PHY mode adaptation selects a SNR threshold (SNR_{PHY}) for PHY mode adaptation. Whenever the average SNR of a BLE connection is greater or equal to SNR_{PHY}, our PHY mode adaptation mechanism chooses the 1M PHY for communication to conserve energy. If the average SNR drops below SNR_{PHY}, our PHY mode adaptation switches to the Coded S8 PHY to sustain the required link-layer reliability PDR_{min}. To initiate a PHY mode switch, the BLE master sends a link-layer PHY update request (`LL_PHY_UPDATE_IND`) to the slave, as discussed in Section 2.1.4.

One potential problem of rule-based parameter adaptation is oscillation, as we discuss in Section 2.2.4. For example, if a device operates at an average SNR close to the SNR_{PHY}, it may continuously switch between the two available PHY modes. To combat such oscillation behavior, we use a hysteresis and switch to the Coded S8 PHY whenever the average SNR drops below SNR_{PHY}, but we only switch to the 1M PHY when the average SNR $\geq SNR_{PHY}+SNR_{offset}$. Our experiments show that even an offset of $SNR_{offset} = 1dBm$ successfully mitigates such unwanted switching behavior.

4.1.4 Evaluation

In this section, we evaluate the performance of our two proposed adaptation mechanisms: our BLE channel management and our PHY mode adaptation. Our BLE channel management (described in Section 4.1.2) mitigates the effects of external interference and fading on a BLE connection by passively monitoring the quality of individual BLE data channels and adaptively selecting only good channels for communication at runtime. Our PHY mode adaptation (described in Section 4.1.3) works in parallel to our BLE channel management and mitigates the effects of a weak signal strength by dynamically adapting the used BLY PHY mode to reach a specified link-layer reliability while minimizing power consumption. To show that our proposed mechanisms do not depend on any specific hardware, we implement our solution on two popular BLE platforms: the Nordic Semiconductor nRF52 and the Raspberry Pi 3B (Pi3).

Nordic Semiconductor nRF52. On the nRF52, we use the Zephyr RTOS [217], which provides a fully standard-compliant BLE communication stack that allows link-layer access and supports all BLE PHY modes, to implement our channel management and our PHY mode adaptation. Per default, an nRF52 device running the Zephyr RTOS does not make use of any BLE channel management or PHY mode adaptation. In our evaluation, we use the `nRF52840` chip to study our approaches, however, our code runs on all chips that are part of the nRF52 series.

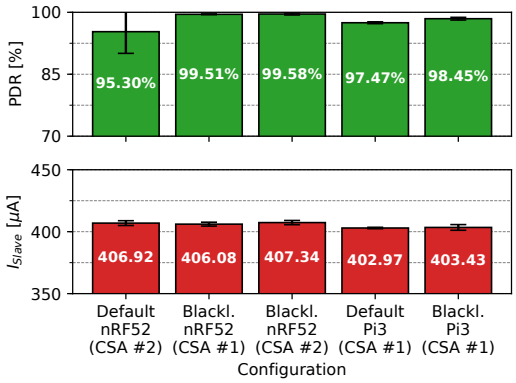
Raspberry Pi 3B (Pi3). On the Pi3, we run Raspbian and the tool InternalBlue [142] to implement the BLE channel management on the onboard Broadcom `BCM43430A1` radio chip of the Pi3. The Broadcom radio on the Pi3 already makes use of basic autonomous BLE channel management. Our BLE channel management, therefore, runs in parallel to Broadcom's existing channel management and extends its performance. The BLE radio of the Pi3 only supports the 1M PHY and the BLE channel selection algorithm #1 (CSA #1).

We experimentally study the performance of our two approaches in our wireless testbed (as described in Section 4.1.1 and in detail in Publication D). We measure the slave's average current draw (I_{Slave}) using our D-Cube testbed facility [183]. Furthermore, we measure the overall link-layer reliability (PDR) of the BLE connection by parsing the link-layer logs recorded on the BLE master device. Following the methodology described in Section 4.1.1, we study our approaches under external interference and under changing antenna attenuation. In all experimental runs, we start with an antenna attenuation of 0 dBm and without any external interference and either gradually change the antenna attenuation or introduce external interference.

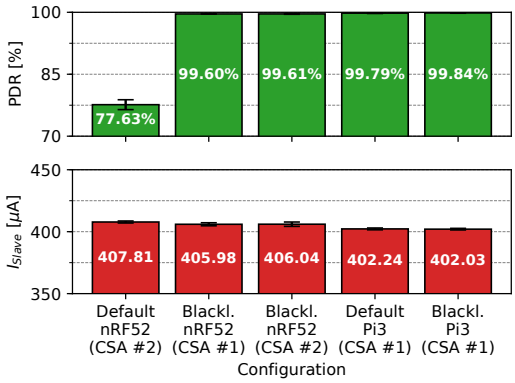
4.1.4.1 BLE Channel Management

First, we evaluate our proposed BLE channel management in three different experimental scenarios. During these experiments, we disable our PHY mode adaptation and fix the used PHY mode to the 1M PHY. Figure 4.5 shows the average PDR and I_{Slave} for five different BLE channel management approaches in three different scenarios. The default behavior of the nRF52 and the Pi3 is shown as *Default nRF52* and *Default Pi3*, respectively. The performance of our BLE channel management on the nRF52 is shown as *Blackl. nRF52 (CSA #1)* and *Blackl. nRF52 (CSA #2)*, where CSA indicates which BLE channel selection algorithm was used. The bar named *Blackl. Pi3* shows the performance of our channel management on the Pi3.

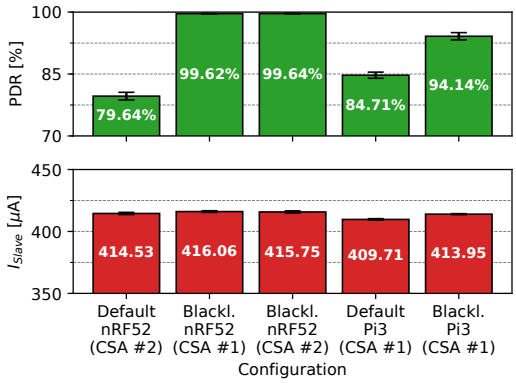
The data in Figure 4.5 show that our proposed BLE channel management significantly improves



(a) Attenuation = 10 dB.



(b) Wi-Fi near the BLE master.



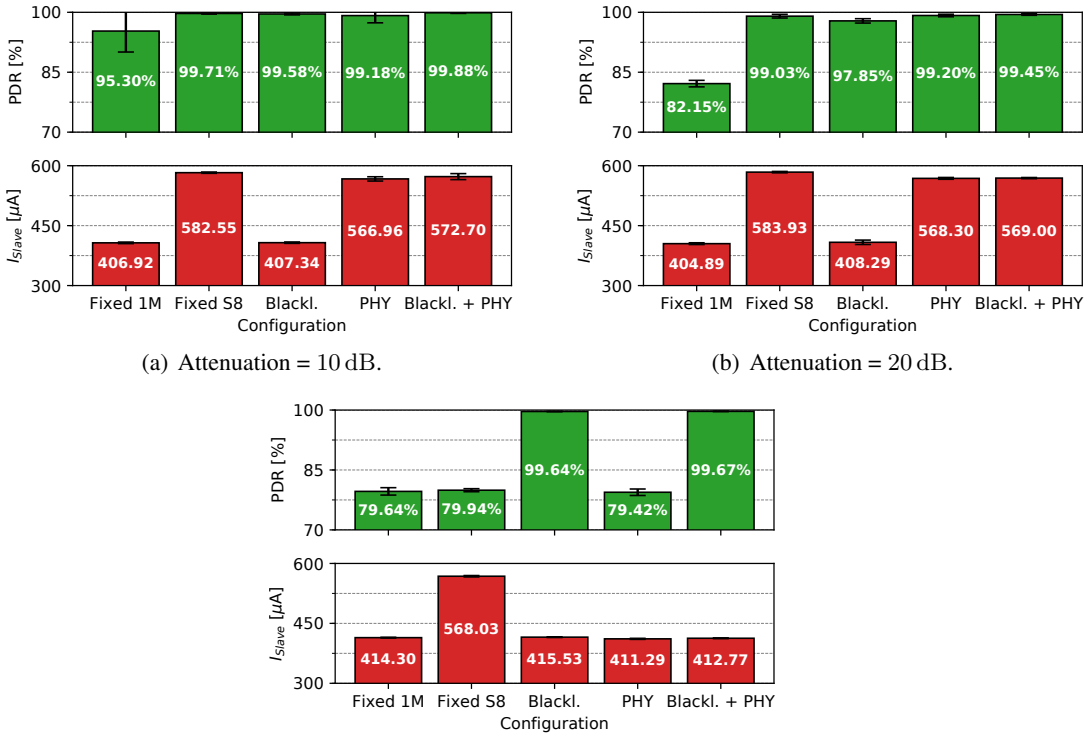
(c) Wi-Fi near the BLE slave.

Figure 4.5: Link-layer reliability (PDR) and average current consumption of the slave (I_{Slave}) for different blacklisting mechanisms in three scenarios. The connection was either using the BLE channel selection algorithm #1 (CSA #1) or CSA #2. Adapted from Publication D.

the link-layer reliability (PDR) of a BLE connection without increasing the power consumption on the BLE slave (I_{Slave}). We further see that our channel management improves the link-layer reliability by up to +22% on the nRF52 and +10% on the Pi3 compared to the default behavior of the platforms. Furthermore, the data indicate that the used CSA does not significantly affect the link-layer reliability of a BLE connection. Overall, our BLE channel management is able to sustain a PDR above 99% in all three experimental scenarios.

4.1.4.2 BLE PHY mode adaptation

Next, we evaluate the link-layer performance of a BLE connection when our BLE channel management and our PHY mode adaptation cooperatively improve link-layer reliability. Therefore, we re-run the experiments from Section 4.1.4.1 with five different configurations: without channel management and a fixed 1M PHY (*Fixed 1M*), without channel management and a fixed Coded S8 PHY (*Fixed S8*), with our proposed channel management and a fixed 1M PHY (*Blackl.*), without channel management and only using our proposed PHY mode adaptation (*PHY*), and finally using our proposed channel management and PHY mode adaptation in parallel (*Blackl. + PHY*). In all



(a) Attenuation = 10 dB.

(b) Attenuation = 20 dB.

(c) Wi-Fi near the BLE slave.

Figure 4.6: Link-layer reliability (PDR) and average current consumption of the slave (I_{Slave}) for five different configurations. Running both mechanisms in parallel (Blackl. + PHY) provides a PDR $> 99\%$ while minimizing power consumption. Adapted from Publication D.

experimental scenarios, we configure the PHY mode adaptation to sustain a PDR above 99%.

The data in Figure 4.6 show that both of our approaches successfully run in parallel to improve the link-layer reliability of connection-based BLE. We can see that our BLE channel management alone (shown as *Blackl.*) significantly improves the link-layer reliability in all experiments shown in Figure 4.6. However, when the BLE signal strength drops significantly, our BLE channel management alone is not able to sustain the specified minimum PDR_{min} of 99% (as shown in Figure 4.6(b)). In case of a weak BLE signal strength, our PHY mode adaptation dynamically improves the PDR of the BLE connection at the price of an increased power draw. Overall, we can see that using our BLE channel management in combination with our PHY mode adaptation is able to sustain the specified minimum PDR_{min} of 99% while minimizing the necessary power consumption of the BLE slave. Overall, we see that both of our proposed approaches successfully run in parallel and are able to sustain a PDR above 99% in all of our experimental scenarios while minimizing the required power consumption on the BLE devices.

4.2 Controlling the Latency over BLE Connections

In this section, we present how off-the-shelf BLE devices can monitor and control the latency of individual data transmissions over a BLE connection. We start by experimentally investigating how the communication latency of individual BLE packets behaves in real-world environments with external interference (Section 4.2.1). To the best of our knowledge, we are the first experimental study highlighting that connection-based BLE communication may experience significant delays in the face of radio interference. Then, we show how BLE devices can make use of existing HCI traffic to passively monitor the latency of BLE packets (Section 4.2.2) and how BLE devices can sustain a given transmission latency by adapting their BLE connection parameters (Section 4.2.3). Finally, we experimentally show that devices using our approach can successfully sustain a given latency bound, even in harsh RF environments (Section 4.2.4).

4.2.1 BLE Latency in Noisy RF Environments

We start by investigating how the latency of data packets exchanged over a BLE connection behaves in a real-world environment. Based on our work in Publication A, we can calculate the upper bound on transmission latency over a BLE connection under ideal channel conditions as:

$$t_{max} = \left\lceil \frac{D}{F} \right\rceil \cdot CI + t_{CE}, \tag{4.2}$$

where D is the data packet length in bytes, F is the maximum number of bytes that can be transmitted during a single BLE connection event, CI is the BLE connection interval, and t_{CE} is the maximum duration of a BLE connection event.

Experimental setup. To check if the calculated t_{max} holds in real-world settings where external interference may be present, we experimentally measure the latency of individual data packets over a BLE connection under different conditions. We use a Nordic Semiconductor `nRF52840 DK` [159] running Zephyr RTOS [217] as a BLE slave and connect it to a Raspberry Pi 3B (Pi3), using its on-board Broadcom BCM43439 BLE radio, acting as BLE master. The slave periodically sends a packet with a link-layer packet length of 80 bytes to the master once every second. We measure the transmission latency ($t_{latency}$) of every slave transmission using our D-Cube testbed facility [183, 184] as the time difference between the slave's application issuing the packet transmission and the master's application successfully receiving the packet from the slave. The BLE connection in our experiments uses a $CI = 250\,ms$, supports an $F = 128\,bytes$, and has a $t_{CE} = 10\,ms$. Using Equation 4.2, we therefore expect a maximum transmission latency $t_{max} = 260\,ms$ for our data packet exchanges from BLE slave to master.

4.2.1.1 Latency in a Common Office Environment

First, we measure the BLE transmission latency in a common office environment over 48 hours. Figure 4.7 shows the latency of the BLE transmissions from slave to master in our common office environment measured experimentally. Each bar in Figure 4.7 shows the percentage of data packets that exceed t_{max} within a period of 15 minutes. We can clearly see that a significant percentage of packets (up to 21.74%) exceed t_{max}, especially during daytime when the office is crowded and external interference is present.

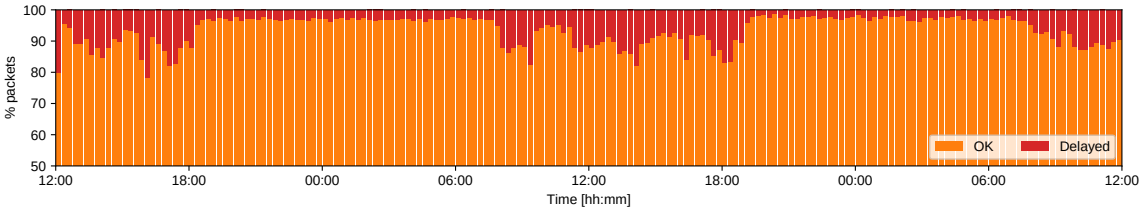


Figure 4.7: Percentage of data packets exceeding the expected maximum transmission latency (t_{max}) in a common office environment across 48 hours. When people are present in the office, up to 22% of all transmissions are delayed. Adapted from Publication B.

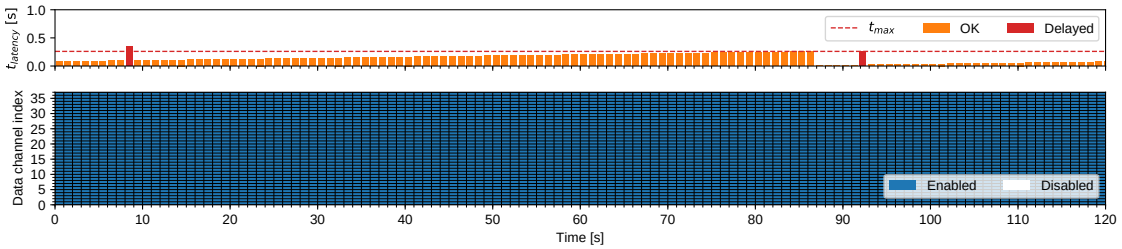
4.2.1.2 Systematic Latency Study

To study the latency of BLE transmissions more systematically, we use the same experimental setup in a vacant university laboratory, where we have fine-grained control over the interference experienced by our BLE devices. Figure 4.8 shows the results of our experiments performed in our testbed facility. Each subplot in Figure 4.8 shows the $t_{latency}$ (top) and the used BLE data channel map (bottom) of the BLE connection. In this experiment, master and slave have a direct line of sight and a distance of approximately 10 meters. Packets with a latency exceeding the calculated $t_{max} = 260\,ms$ (indicated as the dashed horizontal line), are marked as delayed.

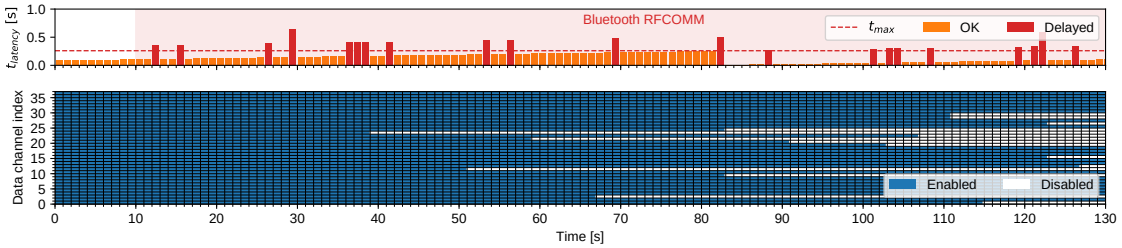
No interference. In Figure 4.8(a), we see that even when we do not create any radio interference in our testbed, individual packets are delayed. These delays are likely caused by link-layer packet loss due to multipath fading or beaconing activities by nearby Wi-Fi devices. On average, 6.33% of all packet transmissions in this scenario exceed t_{max}.

Classic Bluetooth interference. Figure 4.8(b) shows that co-located Classic Bluetooth communication - we are creating Bluetooth RFCOMM traffic with a bandwidth of 725 kbits/s on three parallel Classic Bluetooth connections - affects the communication latency of the BLE connection. Every data packet is eventually successfully received, but between 10% and 15% of packets experience a latency above t_{max}. We further see that the BLE channel management of the Broadcom radio blacklists BLE data channels to mitigate the effects of co-located Bluetooth interference. Classic Bluetooth, however, also makes use of frequency hopping, which means that the Broadcom BLE master is not able to mitigate the effects of Classic Bluetooth interference on the performance of the BLE connection.

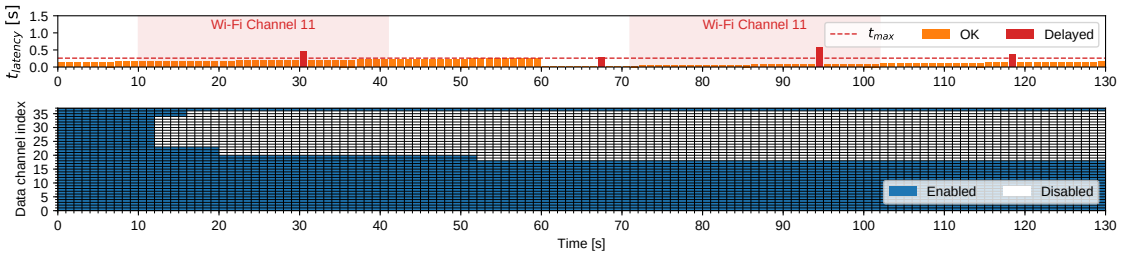
Wi-Fi interference. Figure 4.8(c) and 4.8(d) show the effects of co-located Wi-Fi interference on the performance of the BLE connection. In these scenarios we use JamLab-NG [185] to create Wi-Fi packets on Wi-Fi channel 11 with a transmission power of 30 mW near the BLE master (Figure 4.8(c)) or the BLE slave (Figure 4.8(d)). In both scenarios, all data packets are eventually successfully received by the BLE master, but some transmissions are significantly delayed. When the Wi-Fi interference is near the BLE master, the BLE channel management of the Broadcom radio successfully detects and mitigates the effects of Wi-Fi interference on the BLE connection. As a result, on average, only 12.42% of the received BLE packets exceed t_{max}. However, when the Wi-Fi interference is near the BLE slave, the Broadcom radio does not detect all BLE data channels affected by the Wi-Fi interference and, therefore, does not mitigate the effects of Wi-Fi interference on the BLE connection. Therefore, in this scenario 28.7% of data packets are delayed on average.



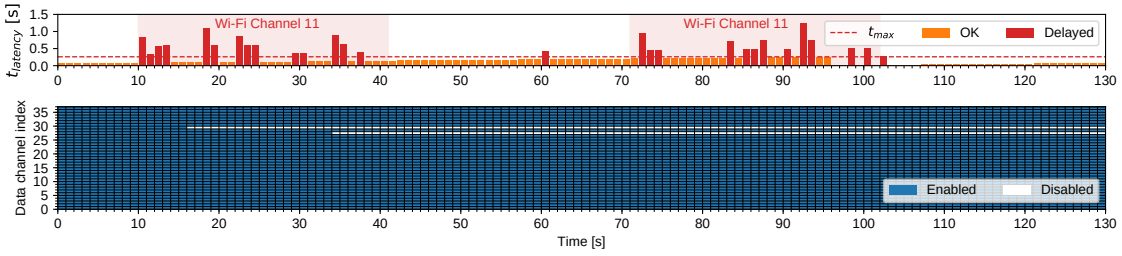
(a) No interference



(b) Classic Bluetooth interference



(c) Wi-Fi interference near the BLE master



(d) Wi-Fi interference near the BLE slave

Figure 4.8: Packet latency ($t_{latency}$) and BLE data channel map of a BLE connection under different interference scenarios. Adapted from Publication B.

4.2.1.3 Lessons Learned

Our experiments confirm that BLE connections are successfully able to transmit all data packets, even under heavy interference, as highlighted in [193]. Nevertheless, external interference may cause link-layer packet loss (as we show in Section 4.1), which in turn leads to *significant transmission delays of BLE data exchanges*. This holds true when other BLE radio platforms are used as BLE master. In our work (shown in detail in Publication B), we repeat our experiments with two other popular BLE radio platforms (the Qualcomm CSR8510 A10 platform and the Panasonic PAN1762 platform) acting as BLE master. Although the BLE channel management of the BLE

platforms significantly differs – the channel management of the Broadcom and Qualcomm platform work rather effectively, while the Panasonic does not make use of channel management in our experiments – all three used BLE platforms lead to 25% of BLE transmissions being delayed in some cases. Unfortunately, state-of-the-art BLE devices do not have a standardized way for an application to monitor its transmission latencies or detect if its data transmissions are delayed. Only with the release of BLE version 5.2 [26], future BLE devices using the new isochronous BLE channels will be able to capture link-layer loss. The mechanism used by BLE 5.2 to monitor communication latencies, however, closely resembles our approach presented next.

4.2.2 Measuring BLE Latency

To allow standard BLE devices to monitor and predict the latency of their transmissions, we analytically model the transmission latency over BLE connections (see Section 2.2.3). Therefore, we update the BLE latency model for ideal channel conditions in Equation 4.2 by introducing the n_{CE} metric, which captures the effects of link-layer packet loss and resulting retransmissions. The n_{CE} metric expresses the number of connection events necessary to successfully transmit individual data fragments over a BLE connection. Our revised model calculates t_{max} as:

$$t_{max} = \left(\sum_{f=1}^{\lceil D/F \rceil} n_{CE_f} \cdot CI \right) + t_{CE}, \tag{4.3}$$

where the bound $\lceil D/F \rceil$ captures the fragmentation of data with length D into one or multiple data fragments of length F, and n_{CE_f} expresses the n_{CE} of an individual data fragment f. CI is the BLE connection interval and t_{CE} is the maximum duration of a single BLE connection event.

Equation 4.3 can be used by off-the-shelf BLE devices to monitor and control their communication latency. To use our updated model, however, BLE devices need to accurately measure the n_{CE} of their transmissions. Measuring the n_{CE} of a BLE connection, however, is difficult due to the design of the BLE communication stack (see Section 2.1.2). As mentioned above, the BLE controller autonomously handles all link-layer functionality and hides its inner workings from the BLE host. An application running on the BLE host, therefore, cannot directly measure the n_{CE} of ongoing transmissions but needs to find a way to estimate the n_{CE} using application-layer acknowledgments or HCI information, which we discuss next.

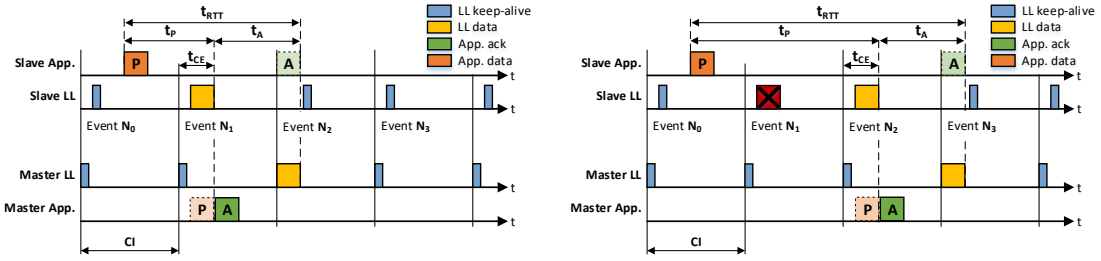
4.2.2.1 Estimating n_{CE} using Round-trip Time

One approach to estimate the n_{CE} of individual data transmissions, is to use application-layer ACKs and measure the round-trip time (RTT), which we call `RTT-based` n_{CE} estimation. Following this approach, every data packet sent by an application is confirmed by an ACK from the application running on the peer device, as shown in Figure 4.9.

An application can measure t_{RTT} as the time between issuing the transmission of a data packet P and receiving the application-layer ACK A. The measured t_{RTT} consists of two parts:

$$t_{RTT} = t_P + t_A, \tag{4.4}$$

where t_P captures the time between issuing and successfully transmitting the packet P, and t_A



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

Figure 4.9: `RTT-based` n_{CE} estimation for a slave transmitting a data packet (P) and receiving an ACK (A). The figure shows the behavior of the application (App.) and link layer (LL) on both BLE devices. Adapted from Publication B.

captures the time between the peer device issuing and successfully transmitting the ACK A. Both packet exchanges, data packet P and ACK A, can be modeled as individual transmissions using Equation 4.3. By assuming that P and A have a packet length of D, we can calculate t_{RTT} as:

$$t_{RTT} \leq 2 \cdot t_{max} = 2 \cdot \left\lceil \frac{D}{F} \right\rceil \cdot n_{CE_f} \cdot CI + 2 \cdot t_{CE}. \tag{4.5}$$

Hence, an application can use t_{RTT} measurements to estimate the average n_{CE_f} as:

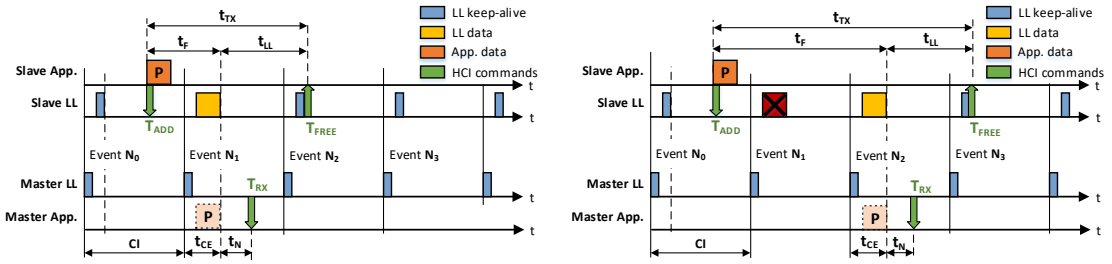
$$n_{CE_f} = \left\lceil \frac{t_{RTT} - 2 \cdot t_{CE}}{2 \cdot \lceil D/F \rceil \cdot CI} \right\rceil. \tag{4.6}$$

This `RTT-based` n_{CE} estimation, however, has several limitations. First, developers need full control over the applications running on the master and the slave to introduce application-level ACKs and RTT measurements. Second, application-level ACKs introduce additional energy consumption, which may not be suitable for low-power applications. Third, this estimation approach can only calculate the average n_{CE_f} across all fragments of a data exchange, which leads to an underestimation of n_{CE_f} values in certain scenarios, as we show in Section 4.2.2.3.

4.2.2.2 Estimating n_{CE} using HCI Timing Information

To address the limitations of `RTT-based` n_{CE} estimation, we present our novel `HCI-based` n_{CE} estimation, which passively monitors standardized HCI commands and events to accurately estimate the n_{CE_f} of individual data fragment transmissions. Since our `HCI-based` n_{CE} approach only uses standardized HCI information, it can be used on off-the-shelf BLE devices.

Figure 4.10 shows how a BLE slave can use `HCI-based` n_{CE} estimation without requiring application-layer ACKs from the master. T_{ADD}, T_{FREE}, and T_{RX} are specific timestamps extracted from passively monitoring the existing HCI communication on the BLE slave. T_{ADD} is the timestamp when the application issues the data packet transmission, indicated as `ACL data packet` command on the HCI. T_{FREE} is the timestamp when the BLE controller indicates that it has freed its outgoing packet buffer by sending an `HCI_Number_Of_Completed_Packets`



(a) No link-layer errors (n_{CE_f} = 1). (b) One link-layer error during transmission (n_{CE_f} = 2).

Figure 4.10: `HCI-based` n_{CE} estimation for a slave transmitting a data packet (P) consisting of one data fragment. The figure shows the behavior of the application (App.) and link layer (LL) on both BLE devices. Adapted from Publication B.

event to the BLE host. T_{RX} is the timestamp when the master's host is notified that the packet P has been successfully received from the slave.

As shown in Figure 4.10, the slave can measure T_{ADD} and T_{FREE} and calculate t_{TX}. The measured t_{TX} consists of two components:

$$t_{TX} = t_F + t_{LL}, \tag{4.7}$$

where t_F is the latency of successfully transmitting a single data fragment into the master's reception buffer and t_{LL} captures the time required at the slave to successfully receive the link-layer ACK and freeing the outgoing packet buffer in the slave's BLE controller.

The transmission latency of a single data fragment (t_F) is calculated by using $D = F$ and Equation 4.3, as:

$$t_F \leq n_{CE_f} \cdot CI + t_{CE}. \tag{4.8}$$

We assume that the short link-layer acknowledgment is successfully transmitted within the first transmission attempt and neglect its duration, resulting in:

$$t_{LL} = CI. \tag{4.9}$$

With this assumption and using Equation 4.7, we calculate t_{TX} as:

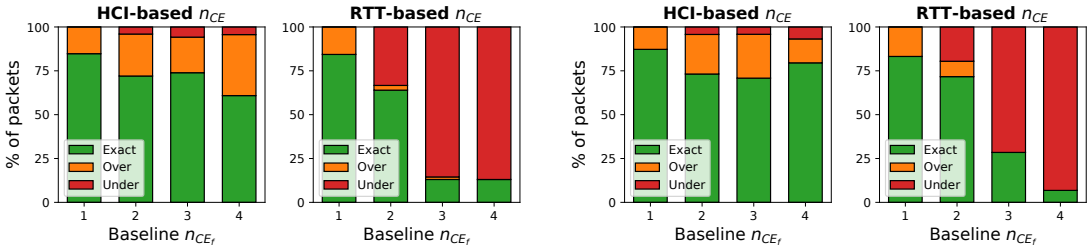
$$t_{TX} \leq (1 + n_{CE_f}) \cdot CI + t_{CE}. \tag{4.10}$$

Therefore, a BLE slave can calculate the n_{CE_f} of individual fragments as:

$$n_{CE_f} = \left\lceil \frac{t_{TX} - t_{CE}}{CI} \right\rceil - 1. \tag{4.11}$$

4.2.2.3 Accuracy and Power Consumption of n_{CE} Estimation

To compare the accuracy and energy-efficiency of the two proposed n_{CE} estimation approaches, we implement both approaches on the Nordic Semiconductor `nRF52840 DK` [159] using the Zephyr RTOS [217]. Because both approaches only require standardized BLE functionality, they can be implemented on every standard-compliant BLE platform. Even devices that using a proprietary interface between host and controller can use our approaches with only minor adaptations.



(a) Connection interval = 62.5 ms.

(b) Connection interval = 125 ms.

Figure 4.11: Accuracy of `HCI-based` and `RTT-based` n_{CE} estimation for two connection intervals. Adapted from Publication B.

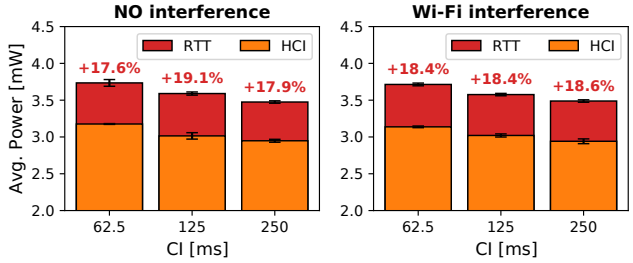


Figure 4.12: Average power consumption of a BLE slave using the proposed n_{CE} estimators for different connection intervals (CI) and interference. Adapted from Publication B.

Accuracy. We reuse the experimental setup from Section 4.2.1 and use our time-synchronized testbed facility to measure the actual n_{CE_f} for every transmitted data fragment from slave to master. We then compare these baseline n_{CE_f} measurement with the estimated n_{CE_f} values of our `RTT-based` and `HCI-based` n_{CE} estimation approaches.

Figure 4.11 shows the percentage of n_{CE_f} values that are correctly estimated, overestimated, or underestimated (shown in green, orange, and red). The data clearly show that the proposed `HCI-based` n_{CE} estimation is more accurate in estimating the n_{CE_f} of packet transmissions, especially for transmissions experiencing high link-layer loss indicated by a high baseline n_{CE_f}.

Power consumption. Using our D-Cube testbed facility [183], we measure the average power consumption of a BLE slave using both n_{CE} estimation approaches under different RF noise.

Figure 4.12 shows the average power consumption of the BLE slave for different BLE connection interval configurations and different types of interference. As expected, the `HCI-based` n_{CE} estimation approach consumes significantly less power than `RTT-based` n_{CE} estimation. The additional application-layer ACKs, which are required by `RTT-based` n_{CE} estimation, result in an approximately 18% higher power consumption of the BLE slave, independent of the type of interference and the used CI.

Overall, `HCI-based` n_{CE} estimation clearly outperforms `RTT-based` n_{CE} estimation in both accuracy and power consumption. Therefore, we only use the `HCI-based` n_{CE} estimation approach to control the transmission latency of BLE applications, which we show next.

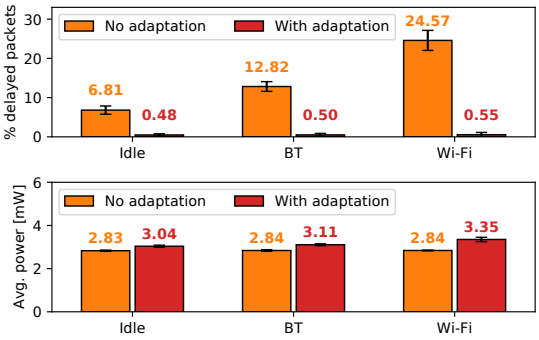


Figure 4.13: Percentage of delayed packets and average power consumption of a slave with and without connection interval adaptation in three different environments. Adapted from Publication B.

4.2.3 Controlling BLE Latency

We can use recent n_{CE} estimates to monitor the latency of packet transmissions over a BLE connection and adapt the BLE connection parameters, if necessary, to sustain a given upper latency bound (as discussed in Section 2.2.4). Using Equation 4.3, we can compute a maximum connection interval (CI_{max}) as:

$$CI_{max} \leq \frac{t_{max} - t_{CE}}{\lceil D/F \rceil \cdot n_{CE_{f\star}}}, \tag{4.12}$$

where $n_{CE_{f\star}}$ is the estimated n_{CE} value for upcoming data fragments and CI_{max} is the most energy-efficient BLE connection interval that is able to sustain the upper latency bound t_{max}. An application running on a BLE slave can use Equation 4.12 in combination with $n_{CE_{f\star}}$ estimation to calculate the most suitable BLE connection interval. If the slave detects that the BLE connection interval needs to be changed, it updates the BLE connection parameter of the BLE connection following the standardized BLE parameter update, as discussed in Section 2.1.4.

As we show in Publication B, using the maximum n_{CE_f} value across the 64 most recent packet transmissions as $n_{CE_{f\star}}$ is an efficient and effective way to estimate $n_{CE_{f\star}}$.

4.2.4 Evaluation

We implement our adaptation approach on a BLE slave using the `nRF52840 DK` and the Zephyr RTOS and connect it to the Pi3 with its onboard Broadcom radio. We use our testbed facility (see Section 4.2.2) to measure the latency of individual data transmissions.

4.2.4.1 Systematic Evaluation

We compare the behavior of a slave using a fixed connection interval (S_{fixed}) to that of a slave using our adaptation approach (S_{adapt}). S_{fixed} calculates the fixed connection interval using Equation 4.2 and S_{adapt} uses the implemented adaptation described in Section 4.2.3. Both slaves try to sustain a maximum transmission latency of $t_{max} = 260\,ms$.

Figure 4.13 shows the performance of S_{fixed} and S_{adapt} in three different RF environments. On top of Figure 4.13, we see the percentage of packets that exceed t_{max}, and on the bottom, we see

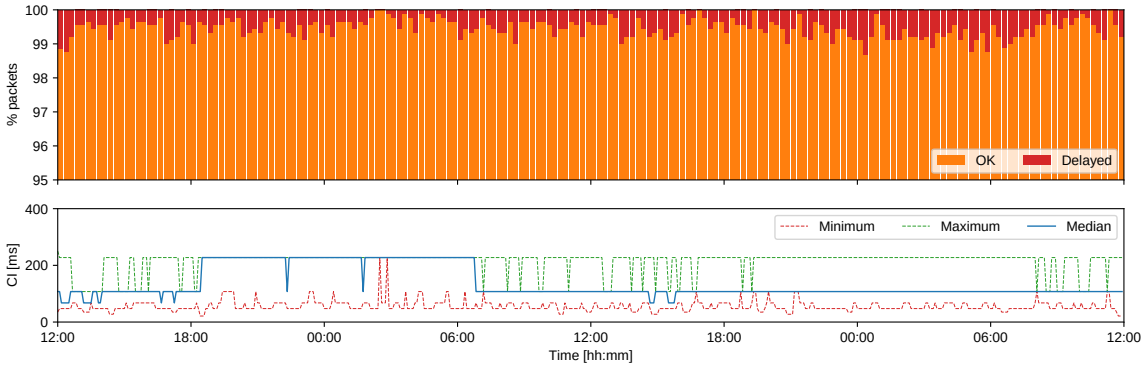


Figure 4.14: When adapting its connection interval (CI) at runtime, a slave is able to significantly increase the timeliness of its BLE communications. Adapted from Publication B.

the average power consumption of the slaves. The data clearly show that our adaptation approach (shown in red) is successfully able to control the latency of its packet transmissions, even under heavy interference. At most 0.55% of transmissions from S_{adapt} are delayed in our experiments, which comes at the cost of below 18% of additional energy consumption compared to S_{fixed}.

4.2.4.2 Long-term Evaluation

We re-run the experiment in our common office environment (discussed in Section 4.2.1.1 and shown in Figure 4.7) with our adaptation approach over 48 hours. Figure 4.14 shows the number of delayed packets and the adaptation of the BLE connection interval in this experiment. Compared to Figure 4.7, where up 22% of packets within 15 minutes are delayed, our adaptive approach results in at most 1.34% of delayed packets within 15 minutes. On average, only 0.54% of all packets are delayed using our adaptive approach (compared to an average of 6.18% of delayed packets without adaptation).

These improvements in transmission latency are consistent across different BLE platforms acting as BLE master and different environments, as we show in Publication B. In some scenarios, our adaptive approach reduces the number of delayed packets by a factor of 40.

4.3 Summary

In this chapter, we have shown how to improve the performance of connection-based BLE communication on off-the-shelf devices by adapting different BLE communication parameters at runtime to changes in the local RF environment. Our proposed BLE channel management effectively increases the link-layer reliability of a BLE connection by dynamically selecting only high-quality data channels for data exchange. Our BLE PHY mode adaptation successfully sustains a given minimum link-layer reliability, while avoiding unnecessary energy consumption on BLE devices. Our BLE connection parameter adaptation enables time-critical data exchange over a BLE connection by monitoring and controlling the delay of individual BLE transmissions.

Because our improvements fully adhere to the BLE specification, they can easily be included into existing and future BLE applications. Upcoming BLE devices supporting the latest BLE

versions, such as BLE version 5.2 [26] and 5.3 [27], may also use our solutions to improve their communication performance. For example, BLE devices may use BLE ISO channels (introduced by BLE version 5.2) in combination with our BLE parameter adaptation to improve their real-time capabilities and energy efficiency. Furthermore, slave devices supporting BLE version 5.3 or above may use our BLE channel management to further increase link-layer reliability.

5

Connecting BLE Devices to the Internet

This chapter focuses on answering our second research question (RQ 2) and presents how constrained devices can use IPv6-over-BLE communication to connect to the IoT and exchange IPv6 data. Section 5.1 shows how IPv6 over BLE, according to the RFC 7668, can be used on off-the-shelf hardware and how IPv6-over-BLE communication compares against IPv6 over IEEE 802.15.4. Section 5.2 presents how IPv6-over-BLE devices can support different IPv6 traffic flows on constrained devices. This chapter is based on Publication A included in this thesis.

5.1 BLEach: Enabling IPv6 over BLE on Constrained Devices

As mentioned in Section 1.2, IoT devices typically need to exchange data with other devices on the Internet, *e.g.*, to send measurement data, receive commands, or check for firmware updates. In most existing BLE-based IoT applications, Internet connectivity is provided by a dedicated gateway device, such as a smartphone or a laptop, that runs a custom application translating standard GATT-based BLE packets into IP packets that can be sent to a predefined server on the Internet. Using such gateway devices, however, creates major interoperability, scalability, and evolvability problems in the overall system [43, 222, 232]. Next, we briefly discuss the main problems of applications using gateways and show how using IPv6-over-BLE communication according to the RFC 7668 [156] can fix them.

First, IoT nodes typically need to communicate with other IoT devices or existing network infrastructure to achieve their application goal. Unfortunately, nodes exchanging standard GATT-based BLE packets are not directly interoperable with other IoT devices. Any packet from such a node needs to be translated by the gateway into IP packets that can be sent to other devices. Implementing such customized protocol translation, however, introduces significant development and maintenance overhead. By exchanging IPv6 packets over BLE connections, *i.e.*, using IPv6 over BLE, BLE-based nodes are fully interoperable with other IPv6 devices and can directly exchange data without the need of any custom gateway translation functionality.

Second, many IoT applications, such as smart city or industrial automation applications, consist of a vast number of nodes. Managing such a large number of devices, *e.g.*, setting the correct device address and configuration in every node, is hard, especially when designing custom device management solutions based on standard GATT-based BLE. Furthermore, creating a custom GATT-based solution for routing packets between many IoT devices is also difficult and error-prone. Fortunately, IPv6 already provides well-established device management and routing functionality for large-scale applications, such as DHCP or DNS. Applications using IPv6 over BLE can, therefore, easily be deployed and managed via using established tools and protocols.

Third, the functionality of individual nodes is usually evolving over time, *i.e.*, new features are added or existing functionality is updated. In GATT-based BLE applications, a developer cannot simply update the firmware of a node to add a new feature or change an existing one, but also needs to update the custom gateway to support the changes. Again, this makes maintaining and updating the overall system error-prone and cumbersome. By using IP and its end-to-end principle, IPv6-over-BLE applications are fully evolvable and can easily be changed and upgraded without the need to also update the gateway logic, which significantly reduces errors and development time.

In this section, we show how constrained and low-power devices with BLE support can use IPv6-over-BLE communication to exchange IPv6-packets with other devices on the Internet. As mentioned above, using IPv6 over BLE communication in BLE-based IoT applications makes the overall system significantly more interoperable, scalable, and evolvable. Towards this goal, we present BLEach, our IPv6-over-BLE communication stack for low-power devices that is part of the Contiki OS [66]. BLEach is the first IPv6-over-BLE stack that is fully open-source (`http://www.iti.tugraz.at/BLEach`) and supports both the IPv6-over-BLE node and router roles.

In contrast to our work presented in the other chapters of this dissertation, where we use the Zephyr RTOS [217] on the Nordic Semiconductor nRF52 platform [159], this chapter uses the Contiki OS [66] on the Texas Instruments CC2650 platform [216]. The main reason for this is that at the time of our research, the Texas Instruments CC2650 in combination with Contiki was the only platform allowing full access to all communication stack layers, including the BLE radio. Furthermore, by using this hardware/software combination we were able to compare our IPv6-over-BLE communication stack to the widely used Contiki IPv6-over-IEEE 802.15.4 communication stack on the same hardware platform (as we show in Section 5.1.5).

5.1.1 Requirements

During the design and implementation of BLEach, we adhere to the following requirements:

Interoperability. BLEach needs to fully adhere to the RFC 7668 [156] and the BLE specification [24] to allow full interoperability with any other standard-compliant IPv6-over-BLE device. This way IPv6-over-BLE nodes running BLEach can follow the standardized primitives to establish an IPv6-over-BLE connection with routers supporting the RFC 7668 and use this connection to communicate with any other IPv6-enabled IoT device on the Internet.

Full-fledged support for IPv6 over BLE. BLEach should support both IPv6-over-BLE node and router roles. This allows BLEach to not only run on battery-powered node devices that stream sensor readings to a cloud server, but also to run on router devices to provide Internet access to one or multiple nodes. Having support for both device roles not only allows BLEach to be self-contained (*i.e.*, it does not require a separate communication stack on the router), but also allows for detailed experiments and further optimizations of IPv6-over-BLE communication, as we have full control over both IPv6-over-BLE devices.

Exposing IPv6-over-BLE tuning knobs. To optimize and fine-tune the performance of individual IPv6-over-BLE devices, we need to have full control over all key parameters of an IPv6-over-BLE connection. Therefore, BLEach needs to expose all IPv6-over-BLE *tuning knobs* and allow to dynamically change one or multiple of these parameters at runtime. By tuning these knobs, a device can sustain given application requirements (*e.g.*, latency, throughput, or reliability) even in

harsh environments with dynamic environmental changes.

Minimal processing and memory overhead. To support even very constrained devices, BLEach needs to add IPv6-over-BLE support while limiting its computational overhead, *e.g.*, caused by IPv6 header compression or IPv6 neighbor table lookups. Furthermore, BLEach should require minimal memory resources in RAM and ROM, as memory may be very limited on some constrained devices.

Fully modular and extendable design. To allow further optimizations to IPv6-over-BLE communication, BLEach's design should be fully modular. Such a modular stack design allows BLEach to be easily extendable by switching individual stack layer implementations to introduce new functionality or improve existing stack layers.

Hardware agnostic. BLEach should be agnostic to the used hardware platform and should support a wide range of different hardware platforms (*e.g.*, single-core and dual-core platforms) by only requiring minimal changes to the lower stack layers. An application should be able to use BLEach without requiring to have detailed knowledge about the actually used hardware platform.

5.1.2 The BLEach Communication Stack

Figure 5.1 shows the architecture of our modular BLEach IPv6-over-BLE communication stack. One of the advantages of IPv6-over-BLE communication according to RFC 7668 is that it does not require any changes to the network (IPv6) or transport layers of existing communication stacks. Therefore, we can use the existing network and transportation layers of Contiki for IPv6 over BLE and only need to design the lowest four layers to support a BLE link layer.

One key challenge in designing BLEach, however, is the black-box nature of the BLE controller, which hides all low-level functionality from the upper stack layers and temporally decouples radio processing from higher-layer protocols (see Section 2.1.2). This behavior of the BLE controller leads to a number of fundamental differences in the communication stack layers compared to existing communication stacks for IoT devices, such as the IPv6-over-IEEE 802.15.4 stack in Contiki (shown in Figure 5.1). The latter typically foresees a radio duty cycling layer scheduling transmissions and directly controlling the on-time of transceivers, as well as a MAC layer taking care of collision avoidance and retransmission of packets [65, 150]. However, all these tasks are already accomplished by the BLE controller, and the challenge for an IPv6-over-BLE stack is to indirectly control all these operations from the above layers in order to fine-tune application performance.

BLE link and PHY. The BLE link and PHY sits at the bottom of the BLEach communication stack, implements all necessary services provided by a BLE controller, and exposes these services to the upper stack layers. As shown in Section 2.1.2, the services exposed by the BLE controller include creating a BLE connection to a peer device, appending packets to the outgoing packet buffer, and notifying upper stack layers about incoming packets. This layer is the only hardware-specific layer within BLEach and needs to support both closed-source, proprietary BLE controllers and open BLE controllers.

Closed-source BLE controllers, such as the Nordic Semiconductor nRF52 SoftDevice [52], autonomously implement the necessary BLE services and hide their implementation details behind the standardized BLE HCI from developers. Open BLE controllers, such as the one provided by

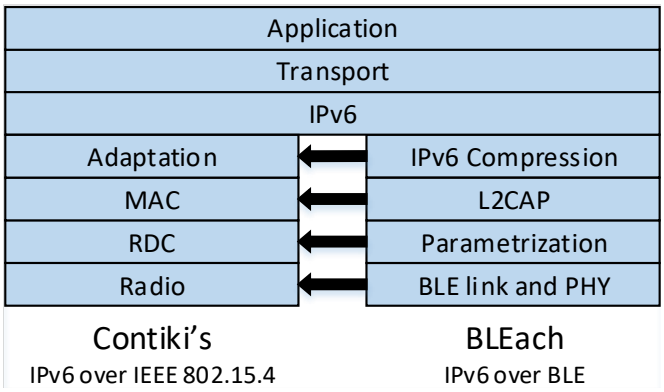


Figure 5.1: Architecture of Contiki's IPv6-over-IEEE 802.15.4 stack (left) and the corresponding layers of BLEach (right). Adapted from Publication A.

the Texas Instruments CC2650 platform [51], require developers to implement (parts of) the BLE services using vendor-specific radio APIs. Therefore, implementing the BLE link and PHY on open BLE controllers is more complex, but also allows to accurately fine-tune and improve the BLE radio functionality. One example for such an improvement in the BLE radio is an effective BLE channel management and PHY mode adaptation as we show in Section 4.1 of this thesis.

Parametrization layer. The parametrization layer sits above the BLE link and PHY and is responsible for selecting and dynamically adapting the tuning knobs of the BLE connection. In contrast to parametrization layers for IPv6-over-IEEE 802.15.4 communication stacks, which directly adapt the radio duty cycle by enabling and disabling the radio [65, 150], the parametrization layer of BLEach *indirectly* changes the BLE radio duty cycle by adapting the BLE connection parameters (the connection interval and slave latency) via standardized commands.

As the used connection parameters significantly affect system and communication performance, which we discuss in Section 2.2.1, this parametrization layer allows developers to dynamically change the BLE connection parameters to influence application metrics such as latency, throughput, or energy efficiency. In the basic version of BLEach, the parametrization layer simply configures suitable BLE connection parameters after connection establishment and does not perform any parameter adaptation. However, due to its modular and extendable design, different parametrization layer versions may be used depending on individual application needs, which we show in Publication A. For example, the parametrization layer may monitor and control the latency of BLE transmissions following the approach we present in Section 4.2. Furthermore, this layer could also temporarily terminate a BLE connection and re-establish it at a negotiated point in time to improve the energy efficiency of mostly-off sensing devices [32, 40].

L2CAP layer. The L2CAP layer in the IPv6-over-BLE communication stack has two main functions, as specified in the RFC 7668 and summarized in Section 2.1.5. First, this layer handles the fragmentation and reassembly of IPv6 packets, which makes it possible to exchange large IPv6 packets over constrained BLE connections. Using L2CAP fragmentation/reassembly, large IPv6 packets are fragmented into smaller L2CAP fragments that are subsequently transmitted to the peer device. Second, the L2CAP layer creates a logical channel between the IPv6-over-BLE node and router, and uses credits to control the flow of individual fragments to avoid buffer overflows,

as we explain in detail in Section 2.1.5.

IPv6-over-BLE devices may adapt this credit-based flow control mode to create a QoS mechanism that allows to prioritize different IPv6 packet flows over others, which we discuss in detail in Section 5.2 of this dissertation.

IPv6 compression layer. As mentioned in Section 2.1.5, the RFC 7668 foresees the use of IPv6 header compression as specified by RFC 6282 [101] over IPv6 over BLE. This header compression mechanism significantly compresses the header of IPv6 packets and, therefore, improves the energy efficiency of IPv6-over-BLE communication.

In contrast to IPv6-over-IEEE 802.15.4 communication, the IPv6 compression layer of BLEach only performs header compression. Packet fragmentation and reassembly, which is typically performed in the same 6LoWPAN layer as IPv6 header compression, is not part of BLEach's IPv6 compression layer, as fragmentation is already handled by L2CAP.

Network and transport layers. As discussed above, IPv6 over BLE does not require any changes to the IPv6 network layer and reuses the standard IPv6 addressing scheme, neighbor discovery, and packet format. Moreover, IPv6 over BLE supports any transport layer on top of IPv6. Therefore, BLEach can reuse any IPv6 implementation for constrained devices such as Contiki's uIP layer [64] and supports TCP, UDP, or any other upper layer running on top.

5.1.3 Integrating BLEach into Contiki

We integrate BLEach into the Contiki OS and reuse its IPv6 and UDP support. Furthermore, we map each of the four lowest layers of BLEach to an existing layer in Contiki's IPv6-over-IEEE 802.15.4 communication stack, as shown in Figure 5.1. This allows developers to use the same application code on either IPv6 over IEEE 802.15.4 or IPv6 over BLE by simply changing the application's configuration file at compile time, which we show in Section 5.1.5.

Because of its generic architecture, BLEach can easily be ported to an arbitrary BLE platform by adapting only the BLE link and PHY implementation. All other stack layers can remain unchanged. Moreover, the buffer sizes of the layers of BLEach as well as the maximum number of simultaneously supported BLE connections are fully configurable, which allows developers to optimize the stack for the hardware platform and the application at hand.

5.1.4 Implementing BLEach

We implement BLEach on the Texas Instruments CC2650 platform [216], which uses an ARM Cortex-M3 application core and a separate ARM Cortex-M0 radio core with IEEE 802.15.4 and BLE support. The basic stack layers of BLEach are implemented as follows:

BLE link and PHY. This layer implements the BLE link and PHY on the CC2650 platform and supports both the BLE master and slave role according to the BLE specification v4.1 [23]. In slave mode, the device may be connected to a single master at a time. In master mode, the device is able to maintain multiple simultaneous connections, whose maximum number depends on the buffer configuration of the BLEach layers. Per default, our implementation can support up to 4 simultaneous BLE connections and allows a BLE connection interval (CI) in the range from 20 ms to 4000 ms as well as a BLE slave latency (SL) between 0 and 500. The CC2650 platform features

an open BLE controller with an API based on shared memory and hardware handshakes. To use BLEach on the CC2650 hardware platform, we implement all necessary BLE services (connection scheduling, buffer management, incoming packet notifications, data channel selection) in Contiki.

Parametrization layer. In the minimum configuration of BLEach, the parametrization layer uses default BLE connection parameters and does not perform any parameter adaptation at runtime. In Publication A, however, we show how this layer can be extended to perform adaptive BLE radio duty cycling based on the current application traffic to improve energy efficiency.

L2CAP layer. The standard L2CAP layer of BLEach supports IPv6 packets with a maximum length of 1280 bytes and splits them into 256-byte L2CAP fragments. After node and router have established a BLE connection, the router creates a single L2CAP LE credit-based flow control channel to the node (as discussed in Section 2.1.5). In the standard L2CAP layer, we use a simple threshold-based flow control mechanism, which grants additional credits to a peer device whenever the peer's credits drop below a fixed threshold. In our implementation, we grant 4 additional credits to the peer when it's credit count drops below 2. This simple flow control mechanism ensures that the two peer devices can exchange at least one L2CAP fragment at any given time.

IPv6 compression layer and above. We adapt the `sicslowpan` layer of Contiki to create the IPv6 compression layer of BLEach. Therefore, we remove the IPv6 fragmentation functionality of `sicslowpan` and only use its IPv6 header compression functionality. Furthermore, we use Contiki's `uip` [64] as BLEach's network and transport layers.

5.1.5 Evaluation

In this section, we experimentally measure the performance of our BLEach communication stack to investigate if BLEach fits all requirements stated in Section 5.1.1. Moreover, we compare the performance of BLEach to that of Contiki's IPv6-over-IEEE 802.15.4 communication stack on the same hardware platform and under the same application configuration. A more detailed evaluation of BLEach can be found in Publication A.

5.1.5.1 Interoperability

One important requirement of BLEach is its interoperability with other IPv6-over-BLE devices that adhere to the RFC 7668 [156]. To show the interoperability of BLEach, we deploy BLEach to a Texas Instruments CC2650 device acting as IPv6-over-BLE node and subsequently connect it to three different IPv6-over-BLE routers: (i) a Raspberry Pi 1 Model B with a LogiLink BZ0015 BLE-USB dongle, (ii) a Raspberry Pi 3 with its onboard Cypress Semiconductor BCM43438 BLE radio, and (iii) a Texas Instruments CC2650 device running BLEach in IPv6-over-BLE router mode. To ensure a fair comparison, we configure all three router devices to use a maximum fragmentation size F = 128 bytes, a BLE connection interval CI = 125 ms, and a BLE slave latency SL = 0. Our BLEach node is connected to one of the three router devices at a time and exchanges IPv6 packets with a length of 256 bytes with the router every second.

Table 5.1 shows the results of our interoperability study. We see that the BLEach node is fully interoperable with all three IPv6-over-BLE router devices. Furthermore, we see that the energy cost on the node, *i.e.*, the consumed energy on the BLEach node for exchanging IPv6 packets with an IPv6-over-BLE router, does not depend on the used IPv6-over-BLE router.

Table 5.1: Interoperability of a BLEach node with three different IPv6-over-BLE router devices.

IPv6-over-BLE router device	Interoperable?	Energy cost node
TI CC2650	✓	103.796 ± 0.659 mJ
Raspberry Pi 3	✓	103.821 ± 0.895 mJ
LogiLink BZ0015	✓	104.597 ± 0.791 mJ

Table 5.2: Memory footprint of BLEach when supporting a maximum IPv6 packet length of 512 bytes.

IPv6-over-BLE role	RAM usage [kB]	ROM usage [kB]
Node	3.318	10.941
Router (1 slave)	3.318	12.938
Router (2 slaves)	5.771	13.023
Router (4 slaves)	10.678	13.023

5.1.5.2 Memory Footprint

Next, we measure the memory consumption of BLEach on the Texas Instruments CC2650 platform. For this analysis, we use default BLEach, which supports a maximum IPv6 packet length of 512 bytes and at most 4 simultaneous IPv6-over-BLE node connections.

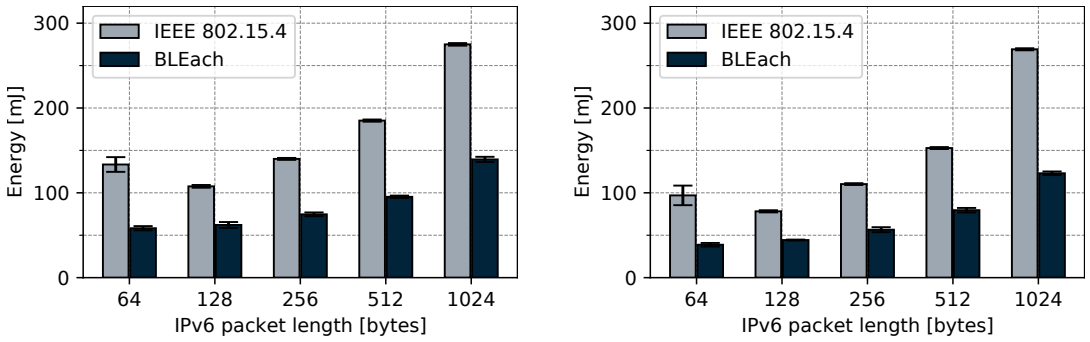
Table 5.2 shows the result of our memory footprint analysis. BLEach requires 3.318 kB of RAM and 10.941 kB of ROM when configured as IPv6-over-BLE node device. Moreover, when operating as IPv6-over-BLE router, BLEach requires 3.318 kB of RAM and 12.938 kB of ROM when connected to a single node device. A BLEach router can support up to four node devices simultaneously, which results in 10.678 kB of RAM and 13.023 kB of ROM usage.

Overall, BLEach is very lightweight and can fit on very constrained hardware platforms that employ only limited RAM and ROM space. Furthermore, as the buffer sizes of BLEach are fully configurable, BLEach can significantly lower its RAM usage. For example, by limiting the maximum IPv6 packet length to 64 bytes, a BLEach node requires only 1.53 kB of RAM.

5.1.5.3 Comparing BLEach to IPv6 over IEEE 802.15.4

Next, we compare the energy efficiency of BLEach to the energy efficiency of Contiki's standard IPv6-over-IEEE 802.15.4 communication stack on the same Texas Instruments CC2650 platform using the same application configuration. For this experiment, we use one CC2650 device as a node device that periodically exchanges UDP packets with the server once every second. Because BLEach fully adheres to the Contiki network stack architecture, we can reuse the same application in our experimental runs and only need to change the used link-layer technology via the project configuration files. When using IPv6-over-IEEE 802.15.4, we enable ContikiMAC's phase-lock optimization and use a wake-up interval of 62.5 ms and 125 ms. Similarly, when using IPv6-over-BLE, we configure BLEach to use a BLE connection interval of 62.5 ms or 125 ms, respectively, to ensure a fair comparison between the two wireless technologies.

Figure 5.2 shows the energy consumption of the CC2650 node device for different link-layer technologies (either BLE or IEEE 802.15.4), IPv6 packet lengths, and wake-up/connection in-



(a) Wake-up/connection interval of 62.5 ms

(b) Wake-up/connection interval of 125 ms

Figure 5.2: Average energy consumption of a Texas Instruments CC2650 node device running either BLEach or Contiki's default IPv6-over-IEEE 802.15.4 communication stack. Adapted from Publication A.

terval configurations. Each experimental run measures the energy consumption of the node for exchanging 60 UDP request/response pairs with the router. The x-axis in Figure 5.2 shows the overall IPv6 packet length, including the IPv6 header, the UDP header, and the UDP payload. Overall, we can clearly see that BLEach consumes approximately 50% less energy than Contiki's IPv6-over-IEEE 802.15.4 stack, independent of the used IPv6 packet length or the wake-up/connection interval. As we show in detail in Publication A, the higher energy efficiency of BLEach can be explained by its lower processing and radio times. First, IEEE 802.15.4 uses a physical data rate of 250 kbit/s, which is 4 times lower than that provided by the BLE 1M PHY used in this experiment. This results in longer radio times for IEEE 802.15.4 when exchanging packets of equal length. Second, IEEE 802.15.4 has a lower maximum fragment length than BLE, which results in large IPv6 packets being fragmented into multiple fragments, each introducing additional link-layer overhead, that are subsequently sent over the wireless link.

Although our experiments show that the BLE link layer is more energy-efficient than IEEE 802.15.4, the BLE 1M PHY results in a lower maximum achievable communication range than IEEE 802.15.4. In our experiments shown in Publication A, IEEE 802.15.4 achieves a maximum communication range of 90 meters in free line-of-sight. The 1M PHY of BLE only achieves a maximum communication range of 75 meters under the same conditions. Nevertheless, a BLE node may use our effective BLE PHY mode adaptation to drastically improve its communication range at the cost of a higher energy consumption, as we show in Section 4.1.3.

5.2 Supporting different IPv6 traffic flows

Most IPv6-over-BLE nodes experience different kinds of IPv6 traffic flows, *e.g.*, time- and safety-critical data exchanges with a cloud server, non-critical firmware update checks, or even unwanted ICMPv6 traffic from other devices on the Internet. These different IPv6 traffic flows, however, are not equally important to the reliable function of the node device (*e.g.*, unwanted traffic may even drain the battery of the node) and should be treated with different priorities. Moreover, the priority of individual IPv6 traffic flows may change at runtime depending on the application state. For example, if the node detects that a critical firmware update needs to be performed, the IPv6

traffic flow downloading the new firmware should receive the highest priority.

In this section, we show how to extend basic BLEach (as presented in Section 5.1) to support different IPv6 packet flows each having its own QoS class. Using our QoS support, BLE nodes can support multiple IPv6 packet flows and can dynamically prioritize certain QoS classes over others. Furthermore, our novel QoS support may be used in IoT applications to add DiffServ support (see Section 3.3.2) to IPv6-over-BLE links.

5.2.1 Adding Traffic Prioritization and Multiplexing to BLEach

To support different QoS traffic classes, we extend the basic L2CAP layer of the BLEach network stack (shown in Figure 5.1) by adding IPv6 traffic prioritization and multiplexing capabilities.

In its basic version, the L2CAP layer of the IPv6-over-BLE communication stack uses a single L2CAP channel in LE credit-based flow control mode between node and router to exchange IPv6 packets. As discussed in Section 2.1.5 and 5.1.2, the credit-based flow control mode handles fragmentation and reassembly of large IPv6 packets and uses its credits to prevent buffer overflows on router and node. Each peer device has a number of credits, which is set at L2CAP channel setup and may be increased by additional L2CAP signaling packets. Sending an L2CAP fragment to the peer device increases the device's credit count by one. If a device has no more credits left, it cannot send any L2CAP fragments until it receives additional credits from its peer.

To add support for different IPv6 traffic flows to IPv6 over BLE, we establish multiple L2CAP LE credit-based flow control channels between router and node at connection setup. Each L2CAP channel has its own fragmentation buffer and credit count and transports a different IPv6 traffic flow with a distinct QoS class. We adapt the L2CAP transmission behavior to prioritize the transmission of L2CAP fragments on the channel with the highest credit count. By granting a different amount of credits to each L2CAP channel, a device can prioritize a specific IPv6 traffic flow over others. For example, a node uses three different L2CAP channels (A, B, and C) to exchange three different IPv6 traffic flows with the router. Channel A carries critical application traffic, channel B is used for firmware update checks, and channel C carries ICMPv6 echo request/response traffic. To prioritize the critical application traffic, the node ensures that the router always has at least 5 L2CAP credits on channel A in comparison to 2 credits for channel B and 1 credit for channel C. This way, the router always transmits application traffic on channel A with the highest priority to the node. If channel A has no more data to send, the router transmits the data on channel B to the node. Only if channel A and B have no more data to send, traffic on channel C is exchanged.

5.2.2 Implementing QoS-enabled L2CAP

We implement our IPv6 traffic prioritization and multiplexing on the Texas Instruments CC2650 platform by extending the basic L2CAP module of BLEach. First, we create a separate L2CAP channel in LE credit-based flow control mode during IPv6-over-BLE connection setup for every supported IPv6 traffic flow. This allows us to multiplex different IPv6 traffic flows over a single BLE connection between node and router. Second, we adapt the fragmentation of the L2CAP layer such that it prioritizes the transmissions of the channel with the highest credit count. Furthermore, we allow IPv6-over-BLE devices to change the priority of incoming IPv6 traffic dynamically by using standardized L2CAP LE flow control credit messages. Although multiple L2CAP channels

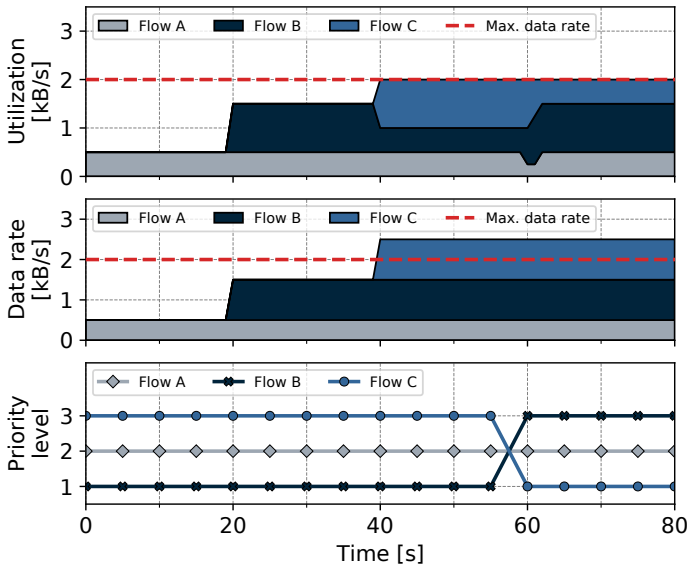


Figure 5.3: IPv6-over-BLE communication between a node and router supporting three different IPv6 traffic classes with different QoS levels. Adapted from Publication A.

for IPv6 data are currently not foreseen by the RFC 7668, our approach does not violate any BLE specification and uses only standardized primitives. The BLE specification only permits devices to grant additional L2CAP credits to the peer and does not foresee that peers can reduce the L2CAP credit count at runtime. As our implementation fully adheres to the BLE specification, changing the priority of IPv6 traffic flows may result in brief adaptation periods, which we show next.

5.2.3 Evaluation

We evaluate our IPv6 traffic prioritization and multiplexing mechanism by running an exemplary IPv6-over-BLE system consisting of a router and a node employing three different IPv6 traffic flows A, B, and C. In this experiment, flow A transports critical application traffic collected by the node, flow B carries ICMPv6 traffic, and flow C embeds commands issued by the node to other nodes in the network. Each IPv6 traffic class is assigned its own L2CAP channel and the channel priorities are adjusted by the router at runtime using the standardized L2CAP credit messages. We investigate how our prioritization and multiplexing mechanism affects the performance of the overall system. In our experiments, we use a BLE connection interval CI = 125 ms and configure the BLE connection to carry at most 256 bytes during a single connection event. The BLE link layer, therefore, can send at a maximum data rate of 2 kB/s using eight connection events per second carrying 256 bytes each.

Figure 5.3 shows the results of our QoS experiment. The top figure shows the utilization of the BLE link, *i.e.*, the actually achieved data rate of each of the three IPv6 traffic flows, and the maximum data rate of the IPv6-over-BLE connection (indicated as red dashed line). The middle of Figure 5.3 shows the data rate of each IPv6 traffic flow under ideal conditions, *i.e.*, without the restrictions imposed by the actual IPv6-over-BLE connection. The bottom of the figure shows the priority level of each of the traffic flows, where a high priority level indicates a high priority.

At time 0, the node periodically transmits 512 bytes/s of traffic class A over the IPv6-over-BLE connection. After 20 seconds, the node additionally generates 1024 bytes/s of traffic class B. At time 40 seconds, the node generates 1024 bytes/s of traffic class C. The sum of the three traffic flows exceeds the maximum achievable data rate over the BLE link and, therefore, the higher-priority traffic flow C gets scheduled first followed by traffic flow A and only a portion of the lower-priority traffic flow B is served. At time 60 seconds, the router switches the prioritization of traffic flow A and C by changing the granted credit count on these channels. Because each flow needs to consume its current credits before the new priorities take effect, the actual priority change requires a brief adaptation period (shown by the utilization in Figure 5.3 from time 60 to 62 seconds).

5.3 Summary

This chapter introduced BLEach, the first full-fledged, open-source IPv6-over-BLE communication stack for constrained and low-power devices. BLEach has minimal processing and memory overhead, a modular and extendable design, and can easily be used by low-power BLE nodes to directly exchange IPv6 data with other IoT devices on the Internet. Furthermore, BLEach exposes the key parameters of IPv6-over-BLE communication as tuning knobs, which enables further optimizations, such as supporting different IPv6 traffic flows with dynamic QoS levels.

Although BLEach uses BLE version 4.1 for data exchange, it significantly outperforms existing IPv6-over-IEEE 802.15.4 solutions, as we show in our experimental evaluation. Porting BLEach to more recent BLE versions, such as BLE version 5.2 [26], would even increase its energy efficiency, reliability, and throughput. For example, BLEach could make use of the different BLE PHY modes to achieve a longer communication range or more throughput (as discussed in Section 4.1). Furthermore, BLEach may use BLE ISO channels and our proposed BLE connection parameter adaptation (as shown in Section 4.2) to reduce its communication delay.

In our work on traffic prioritization over IPv6 over BLE, we went beyond the existing BLE specification and established multiple L2CAP channels in LE credit-based flow control mode between BLE node and router. With the release of the BLE specification version 5.2 [26] in 2020, having multiple of these L2CAP channels is officially supported by off-the-shelf BLE devices. Upcoming BLE devices with support for BLE version 5.2 or above, therefore, can directly make use of our QoS support for IPv6 over BLE to handle multiple IPv6 traffic flows with different QoS classes.

6

Meeting End-to-End Requirements in BLE-based IoT Applications

In this chapter, we answer our third research question (RQ 3) and show how BLE nodes can meet given end-to-end requirements, such as a given maximum end-to-end latency and a given minimum end-to-end reliability, when communicating with devices outside the local BLE subnet, *e.g.*, a cloud server on the Internet. We identify problems of existing cloud-based BLE applications (Section 6.1) and discuss how BLE nodes can model end-to-end communication latency and reliability (Section 6.2). We further investigate how nodes can estimate and meet end-to-end metrics using these models (Section 6.3 and 6.4), implement our solutions on off-the-shelf hardware (Section 6.5), and experimentally evaluate our improvements (Section 6.6). This chapter is based on Publication E of this thesis.

6.1 Investigating Cloud-based BLE Applications

We start by experimentally investigating the end-to-end behavior, *i.e.*, the end-to-end communication latency and reliability, of BLE nodes exchanging data with a cloud server on the Internet.

Figure 6.1 shows the network topology of our initial experiment, where an IPv6-over-BLE node is connected to an IPv6-over-BLE router providing Internet access. Node and router exchange IPv6 packets according to the RFC 7668 [156], as discussed in Chapter 5. This network topology is commonly used in popular IoT protocols, such as CoAP, MQTT, or MQTT-SN. As it is typical for low-power IoT applications, our node uses the lightweight UDP transport layer to communicate with the cloud server, instead of the heavyweight TCP transport layer. This allows a low power and memory consumption on our node at the cost of potential packet loss on the network path [21].

As discussed in Chapter 4, BLE connections use adaptive frequency hopping and autonomous packet retransmissions to achieve a reliability of 100% within the BLE subnet. Therefore, communication *within the BLE subnet* does not experience packet loss, but may experience packet delays due to link-layer effects such as external radio interference or multipath fading. Packet exchanges *across the external network path*, instead, may experience both packet loss and delay. These cannot be controlled by the individual nodes, and their severity strongly depends on the technology used to provide Internet access (*e.g.*, Ethernet, 3G, or 4G).

Experimental setup. We measure the end-to-end latency and reliability of IPv6 packet exchanges across the whole network topology (shown in Figure 6.1) in our wireless testbed powered by D-Cube nodes [183] located in a vacant laboratory, as described in detail in Publication E.

We simultaneously use four Nordic Semiconductor `nRF52840 DK` devices [159] to commu-

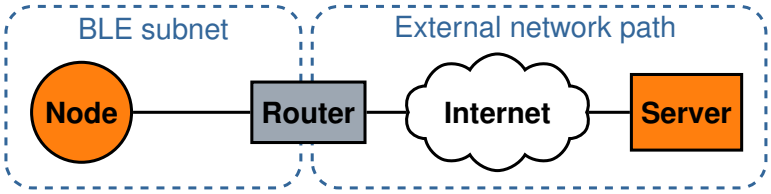


Figure 6.1: Network topology when exchanging data between a BLE node and a cloud server over the Internet. Adapted from Publication E.

nicate with an Amazon Web Service (AWS) cloud server instance located in Frankfurt, Germany. Each node runs an IPv6-over-BLE application built on the Zephyr OS that periodically transmits a UDP packet to the server once every second. Whenever the cloud server receives a UDP packet from one of the nodes, it echoes the received UDP packet back to the node. All packets between nodes and server have an IPv6 packet length of 128 bytes and carry a unique sequence number to match request and response. We use a Raspberry Pi 4 (Pi4) device in combination with a custom BLE HCI-USB dongle as IPv6-over-BLE border router. All nodes use the 2M PHY mode of BLE and all 37 BLE data channels without channel blacklisting to communicate with the router, *i.e.*, the adaptive solutions presented in the previous chapters are disabled in these experiments.

All devices in our experiment (nodes, router, and cloud server) are synchronized to the same NTP server and, therefore, have the same notion of time. With this synchronization, we sustain an average clock offset between nodes and server of -43 $\pm$ 134μs. This allows us to calculate the end-to-end latency (t_{TX}) and the latency within the BLE subnet ($t_{TX_{BLE}}$) for every packet sent from node to server. Similarly, we calculate the end-to-end latency (t_{RX}) and the latency within the BLE subnet ($t_{RX_{BLE}}$) for packets sent from server to node. To test the impact of different technologies providing Internet access to our BLE subnet, our router can make use of a wired Ethernet or a cellular 4G connection for every connected node.

Preliminary results. Table 6.1 and 6.2 show the distribution of the measured latencies between nodes and server measured over 7 days in our testbed.

First, the tables clearly show that the used Internet connection significantly impacts the overall communication latency. In our experiments, the maximum experienced communication latency while using a cellular Internet connection is almost 10 times higher than for a wired connection. Also, the median latency of a cellular connection is significantly higher than for a wired connection. To sustain a given end-to-end latency, a BLE node can hence not simply assume a fixed delay across the external network path but needs to estimate the actual experienced communication delay on the Internet, which we show in Section 6.3.

Second, Table 6.1 and 6.2 show that, as expected, the BLE connection interval CI affects the communication latency for packet transmissions and receptions. However, the BLE slave latency SL does not significantly impact the latency of packets sent by the node ($t_{TX_{BLE}}$), as shown in Table 6.1. Only packets received by the node ($t_{RX_{BLE}}$) are affected by the SL parameter, because the node may skip up to SL connection events when it has no data to transmit, which means that the router may need to wait up to SL connection events for the node to wake up and receive packets (see Section 2.1.4).

Hence, we see that both the BLE connection parameters and the behavior of the external network path affect the end-to-end latency and reliability of packet exchanges. To sustain given end-to-end requirements while limiting unnecessary power consumption, a node needs to dynamically

Table 6.1: Measured latencies of packet *from node to server* (packet transmissions) over 7 days in our testbed for an IPv6 packet length of 128 bytes. *The table shows the median (50%), 95 percentile (95%), and maximum experienced latency (100%) for different configurations.*

Connection	CI **[ms]**	SL	t_{TX} **[ms]**			$t_{TX_{BLE}}$ **[ms]**		
			50%	**95%**	**100%**	**50%**	**95%**	**100%**
wired	50	0	40	88	328	31	79	319
wired	50	4	39	86	732	30	75	258
cellular	50	0	73	1157	3026	28	69	237
cellular	50	4	73	739	4038	27	62	283

Table 6.2: Measured latencies of packet *from server to node* (packet receptions) over 7 days in our testbed for an IPv6 packet length of 128 bytes. *The table shows the median (50%), 95 percentile (95%), and maximum experienced latency (100%) for different configurations.*

Connection	CI **[ms]**	SL	t_{RX} **[ms]**			$t_{RX_{BLE}}$ **[ms]**		
			50%	**95%**	**100%**	**50%**	**95%**	**100%**
wired	50	0	42	92	293	32	82	284
wired	50	4	41	292	1291	32	282	1281
cellular	50	0	67	601	2411	34	64	176
cellular	50	4	268	531	1031	191	257	499

adapt its BLE connection parameters at runtime to changes in the network. Finding suitable BLE connection parameters, however, requires new BLE models, which we show in Section 6.2.

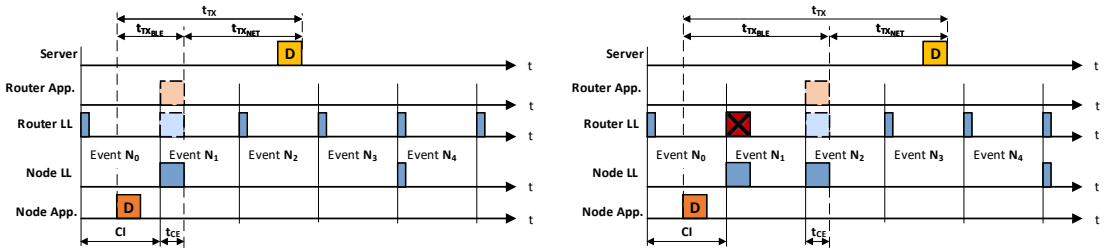
One possible approach to sustain given end-to-end latency bounds would be to use application-level round-trip time measurements on the node to monitor and control the one-way delays t_{TX} and t_{RX}. The main problem with this approach is that a BLE connection usually uses an $SL > 0$, which allows the node to limit its power consumption. An $SL > 0$, however, also leads to packet receptions being unpredictably delayed, which significantly affects the accuracy of individual t_{TX} and t_{RX} estimates. Therefore, this work follows another approach, where we devise a new end-to-end model (Section 6.2) and show how infrequent probing bursts are able to accurately estimate the network latency across the entire network path (Section 6.3). In Section 6.4, we combine our model and network latency estimation to sustain given end-to-end dependability requirements.

6.2 Modeling End-to-End Communication Performance

Based on the local BLE model from Section 4.2 of this dissertation, we devise a new end-to-end model that fits the use case shown in Figure 6.1. In this end-to-end model, we extend the latency model to account for delays on the external network path and investigate how both BLE connection parameters, the BLE connection interval (CI) and the BLE slave latency (SL), affect timeliness when the node transmits (Section 6.2.1) or receives data (Section 6.2.2).

6.2.1 Transmitting Data to the Server

We start by modeling the end-to-end latency (t_{TX}) for a node transmitting data packets to a cloud server using the topology shown in Figure 6.1.



(a) No link-layer errors during BLE data transmission. (b) One link-layer error during BLE data transmission.

Figure 6.2: Timing of a BLE node *transmitting* a data packet (D) to a cloud server on the Internet. Adapted from Publication E.

Figure 6.2 shows two examples of a transmission from a node to the server. Both examples show the node application (Node App.) issuing a data transmission (D) between BLE connection event N_0 and N_1. In Figure 6.2(a), no BLE link-layer errors occur and the BLE link-layer (LL) of the node successfully transmits the data in connection event N_1 to the router. In Figure 6.2(b), however, the BLE connection between node and router experiences link-layer errors (see Section 4.1 for more details on BLE link-layer errors) during connection event N_1. Therefore, the packet transmission during event N_1 is not successful and the node retransmits the packet until successfully received by the router. After the router has successfully received the packet from the node, it forwards the packet via the Internet to the server, which takes $t_{TX_{NET}}$.

In the examples in Figure 6.2, the packet length (D_{TX}) is smaller than the maximum packet lenght of a single connection event (F_{TX}). Therefore, the packet can be transmitted within a single BLE connection event. If $D_{TX} > F_{TX}$, the packet is split into multiple data fragments and each fragment is subsequently sent in its own connection event.

In Figure 6.2 we see that the end-to-end latency (t_{TX}) for transmitting a data packet consists of:

$$t_{TX} = t_{TX_{BLE}} + t_{TX_{NET}}, \tag{6.1}$$

where $t_{TX_{BLE}}$ is the transmission latency of the packet between node and router and $t_{TX_{NET}}$ is the transmission latency of the packet from router to server. As discussed in Section 6.1, $t_{TX_{NET}}$ depends on the technology used to connect to the Internet. The delay $t_{TX_{BLE}}$ can be modeled as:

$$t_{TX_{BLE}} = \left(\sum_{f=1}^{\lceil D_{TX}/F_{TX} \rceil} n_{CE_f} \cdot CI \right) + t_{CE}, \tag{6.2}$$

as we discuss in detail in Section 4.2.2.

The end-to-end transmission latency t_{TX} is calculated by combining Equation 6.1 and 6.2 as:

$$t_{TX} = \left(\sum_{f=1}^{\lceil D_{TX}/F_{TX} \rceil} n_{CE_f} \cdot CI \right) + t_{CE} + t_{TX_{NET}}. \tag{6.3}$$

To calculate the maximum end-to-end transmission latency ($t_{TX_{MAX}}$), we assume that every

packet fragment is transmitted with $n_{CE_f} = n_{CE_{MAX}}$, *i.e.*, that it experiences the same link-layer error probability. We further assume that the data packet experiences the maximum delay across the external network path ($t_{TX_{NET}} = t_{NET_{MAX}}$). Combining these assumptions with Equation 6.3, we can calculate $t_{TX_{MAX}}$ as:

$$t_{TX_{MAX}} \geq n_{CE_{MAX}} \cdot \left\lceil \frac{D_{TX}}{F_{TX}} \right\rceil \cdot CI + t_{CE} + t_{NET_{MAX}}. \tag{6.4}$$

As discussed in detail in Section 2.1.4, the slave latency (SL) does not impact the end-to-end latency of packets transmitted by the node.

6.2.2 Receiving Data from the Server

Next, we model the end-to-end reception latency (t_{RX}) for a node receiving data from the server.

Figure 6.3 shows two examples of a packet sent from the server to the node. In these examples, the server issues a data transmission to the node and, after a delay $t_{RX_{NET}}$, the router application (Router App.) successfully receives the packet between connection events N_1 and N_2. Upon packet reception, the router forwards the packet to the node by issuing a BLE transmission on its link layer (Router LL). Next, the router LL tries to send the packet over the BLE connection to the node in connection event N_2. However, the node makes use of its configured slave latency ($SL = 2$) and does not wake up during event N_2 to receive any packets. Therefore, the router LL retransmits the packet in the subsequent connection events until the packet is successfully received by the node. In Figure 6.3(a), where no link-layer errors occur, the node successfully receives the packet during connection event N_3. In the example shown in Figure 6.3(b), the router and node wake up during connection event N_3, but experience link-layer errors and cannot exchange the packet. Because the node has not received a valid BLE link-layer packet in event N_3, it wakes up during every subsequent connection event until it receives a valid BLE packet from the router, as specified by the BLE specification [26]. In Figure 6.2(b) this happens during connection event N_4, after which the node has successfully received the data from the server.

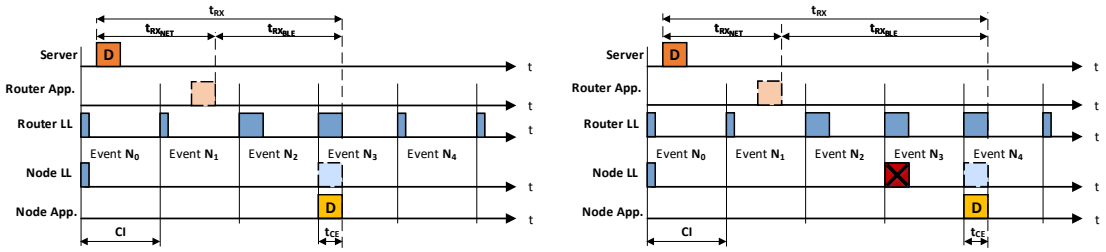
Similar to Section 6.2.1, the packet length (D_{RX}) in these examples is smaller than the maximum packet length supported by a single connection event (F_{RX}). If $D_{RX} > F_{RX}$, the packet is split into multiple fragments that are subsequently sent from router to node in individual connection events. In this case, the MD-field in the BLE link-layer header of packets sent by the router informs the node that more data needs to be exchanged. The node, therefore, does not make use of its slave latency until all data from the router is successfully received.

The end-to-end reception latency (t_{RX}) for a node receiving data from a server consists of:

$$t_{RX} = t_{RX_{NET}} + t_{RX_{BLE}}, \tag{6.5}$$

where $t_{RX_{NET}}$ is the packet latency between server and router and $t_{RX_{BLE}}$ is the packet latency from router to node. Similar to Section 6.2.1, $t_{RX_{NET}}$ depends on the technology used to connect the router to the Internet, and will be studied in Section 6.3. We model $t_{RX_{BLE}}$ as:

$$t_{RX_{BLE}} = \left(\sum_{f=1}^{\lceil D_{RX}/F_{RX} \rceil} n_{CE_f} \cdot CI \right) + t_{CE} + t_{SL}, \tag{6.6}$$



(a) No link-layer errors during BLE data transmission. (b) One link-layer error during BLE data transmission.

Figure 6.3: Timing of a BLE node *receiving* a data packet (D) from a cloud server on the Internet. Adapted from Publication E.

where D_{RX} is the data packet length and F_{RX} is the maximum packet length that can be received by the node within a single BLE connection event. Like in Equation 6.1, CI is the BLE connection interval, t_{CE} is the maximum duration of a single connection event, and n_{CE_f} is the number of connection events necessary to successfully transmit an individual packet or fragment. The delay t_{SL} measures the additional time required by the router, because the node makes use of its SL and skips up to SL connection events. t_{SL} has a maximum value of:

$$t_{SL_{MAX}} = CI \cdot SL. \tag{6.7}$$

We combine Equation 6.5 and 6.6 to calculate the end-to-end reception latency t_{RX} as:

$$t_{RX} = t_{RX_{NET}} + \left(\sum_{f=1}^{\lceil D_{RX}/F_{RX} \rceil} n_{CE_f} \cdot CI \right) + t_{CE} + t_{SL}. \tag{6.8}$$

Furthermore, we use the approach followed in Section 6.2.1 to calculate the maximum end-to-end reception latency ($t_{RX_{MAX}}$) by using $n_{CE_f} = n_{CE_{MAX}}$, $t_{RX_{NET}} = t_{NET_{MAX}}$, and Equation 6.7 as:

$$t_{RX_{MAX}} \geq (n_{CE_{MAX}} \cdot \left\lceil \frac{D_{RX}}{F_{RX}} \right\rceil + SL) \cdot CI + t_{CE} + t_{NET_{MAX}}. \tag{6.9}$$

In Equation 6.9, we can see that the slave latency SL affects the time t_{SL} and therefore also the overall end-to-end reception latency. When a node uses an $SL > 0$, the node minimizes its power consumption by skipping connection events at the price of increased reception delays.

The end-to-end model described in this section relies on accurate estimations of the delay caused by the external network path (t_{NET}), which we investigate next.

6.3 Estimating End-to-End Metrics

One requirement for estimating the delay on the external network path (t_{NET}) is that the estimation is done on the node device in a way that is fully compliant with the end-to-end principle of IP.

Using this approach, a node can estimate t_{NET} without requiring any changes to devices on the network path (*e.g.*, routers). Our approach can, therefore, easily be used by any node connected to an IPv6-over-BLE compliant router. Other estimation approaches that violate the IP end-to-end principle, *i.e.*, requiring changes to routers in the network, are seldomly used in practice due to their huge setup and deployment costs [233]. As our t_{NET} estimation is fully technology-agnostic, a node can use our t_{NET} estimation independently of the applied technology to connect to the Internet. Moreover, our estimation allows us to estimate t_{NET} in any network scope, *e.g.*, applications that span only a local Intranet.

One simple approach to estimate t_{NET} would be to use existing application traffic to calculate the delay introduced by the external network path. Unfortunately, this simple approach does not provide accurate t_{NET} estimations because of the typical asymmetric behavior of BLE connections. During ordinary operation, the BLE connection typically makes use of a long BLE connection interval (CI) and a BLE slave latency (SL) above 0 to sustain a low transmission latency while limiting power consumption on the node. Such a setting, however, causes two problems while estimating t_{NET}. First, an $SL > 0$ leads to packets from router to node being unpredictably delayed (see Section 6.2.2), which significantly affects the accuracy of individual t_{NET} estimates. Second, a large CI value results in a coarse t_{NET} sampling resolution impacting the granularity of our t_{NET} estimates.

6.3.1 Probing Network Latency

We estimate t_{NET} using short, infrequent probing bursts, in which we exchange short probing and corresponding acknowledgment packets between node and cloud server. At the start of every probing burst, the node changes its BLE connection parameters to the smallest possible CI and a $SL = 0$, as discussed in Section 2.1.4. By temporarily changing the used BLE connection parameters for probing, we achieve the lowest possible t_{NET} sampling resolution and eliminate the unpredictable delay of messages from router to node, as discussed in Section 6.3.3.

After setting the new BLE connection parameters, the node sends short probing packets to the cloud server. The server responds to each probing packet with a short acknowledgment packet. The node issues a new probing packet as soon as the previous probing packet has been successfully acknowledged by the server. The node can use ordinary application data packets or distinct IPv6-based packets (*e.g.*, ICMPv6 echo requests/responses) for probing. To achieve the most accurate t_{NET} estimations, however, probe and acknowledgment packets need to fit within a single connection event, *i.e.*, $D_{TX} \leq F_{TX}$ and $D_{RX} \leq F_{RX}$. After node and server have successfully exchanged L_{Probe} probe/acknowledgment packets, the node reverts the BLE connection parameters to the setting used before probing and continues with its normal behavior.

During probing, the node measures the round trip time (t_{RTT}) and the BLE transmission time ($t_{TX_{BLE}}$) of every exchanged probe/acknowledgment pair. t_{RTT} is the time between the node application issuing a probe packet and successfully receiving the corresponding acknowledgment from the cloud server. $t_{TX_{BLE}}$ is measured by monitoring the BLE Host Controller Interface (HCI), as described in detail in Section 4.2.2.2.

The round trip time t_{RTT} measured by the node consists of two parts:

$$t_{RTT} = t_{Probe} + t_{ACK}, \tag{6.10}$$

where t_{Probe} is the end-to-end latency of a probe packet from node to server and t_{ACK} is the end-to-end latency of the corresponding acknowledgment packet from server to node. By using Equation 6.1 and 6.5 for modeling t_{Probe} and t_{ACK}, respectively, we get

$$t_{RTT} = t_{TX_{BLE}} + t_{TX_{NET}} + t_{RX_{NET}} + t_{RX_{BLE}}. \tag{6.11}$$

To simplify our t_{NET} estimation, we assume that the delay across the Internet is symmetric and call this delay t_{NET}. Please note that, although we assume a symmetric delay across the external network path, our t_{NET} estimation approach also accurately captures the one-way latency of asymmetric Internet connections, such as 4G communication, as we show in Publication E. By assuming a symmetric delay, we can set $t_{NET} = t_{TX_{NET}} = t_{RX_{NET}}$, which gives us:

$$t_{RTT} = t_{TX_{BLE}} + 2 \cdot t_{NET} + t_{RX_{BLE}}. \tag{6.12}$$

Using this equation, the delay on the external network path t_{NET} can be calculated as:

$$t_{NET} = \frac{t_{RTT} - t_{TX_{BLE}} - t_{RX_{BLE}}}{2}. \tag{6.13}$$

As mentioned above, the node can measure t_{RTT} and $t_{TX_{BLE}}$ for every probing packet. Unfortunately, $t_{RX_{BLE}}$ cannot directly be measured by the node but needs to be estimated. Because both probing and acknowledgment packets fit within a single connection event and $t_{TX_{BLE}}$ provides us with the most recent link-layer information, we assume $t_{RX_{BLE}} = t_{TX_{BLE}}$. Furthermore, we assume that packets from router to node take at least CI, which is the smallest time unit we can measure with our probing approach, resulting in:

$$t_{RX_{BLE}} = MAX(t_{TX_{BLE}}, CI). \tag{6.14}$$

6.3.2 Estimating the Maximum Network Latency

Using the t_{NET} estimations of individual packet exchanges, we estimate the maximum network latency ($t_{NET_{MAX}}$) of future packet exchanges. With accurate $t_{NET_{MAX}}$ estimations and our proposed end-to-end model from Section 6.2, BLE nodes are able to sustain end-to-end communication requirements, as we discuss in Section 6.4.

Fortunately, estimating upper latency bounds on packet transmissions across the Internet is a well-researched topic [69, 104, 105]. However, most of the existing approaches use sophisticated statistical analysis [69, 180] and are not suitable for constrained devices, such as BLE nodes with limited power supply and processing capabilities. Therefore, we adapt the well-established RTT estimation approach of TCP as specified by Jacobson [103] and standardized in [167], as this approach is suitable for constrained devices.

Instead of using an exponentially weighted moving average as proposed in [167], we record all individual t_{NET} measurements of a probe burst and calculate their average ($t_{NET_{AVG}}$) and variance ($t_{NET_{VAR}}$). Using these values, we can calculate $t_{NET_{MAX}}$ as:

$$t_{NET_{MAX}} = t_{NET_{AVG}} + K \cdot t_{NET_{VAR}}, \tag{6.15}$$

where we use a fixed $K = 4$ as specified in [167].

6.3.3 Choosing a Probe Burst Length

In Publication E, we study the most suitable probe burst length (L_{Probe}) that provides an accurate $t_{NET_{MAX}}$ estimation while limiting the energy consumption caused by probing. We do this empirically using the setup described in Section 6.1.

Our experiments in four different environments show that a probe length $L_{Probe} = 10$ provides the most suitable t_{NET} estimation of future packet exchanges, while also limiting unnecessary energy consumption on the BLE node. Furthermore, our experiments also show that issuing a new probing burst every 1000 seconds performs best in our four experimental environments.

6.4 Meeting End-to-End Requirements

Next, we show how our model (Section 6.2) and our t_{NET} estimations (Section 6.3) can be used by a node to sustain a given end-to-end latency and a given end-to-end reliability, while minimizing its power consumption. We investigate how a node can cope with network loss (Section 6.4.1) and meet end-to-end latency requirements when connected to a router with or without radio duty cycle constraints (Section 6.4.2 and 6.4.3, respectively).

6.4.1 Meeting Reliability Requirements

A node can meet a given minimum end-to-end reliability ($\widehat{r}_{MIN}$) by dynamically adapting the number of necessary application packet transmissions based on the current transmission reliability (r_{NET}) over the network path. As discussed in Section 6.1, no packets get dropped over the BLE connection between node and router, because of the autonomous packet retransmission and flow control of the BLE link layer. However, since we use the lightweight UDP transport layer, packets may be lost over the external network path due to router buffer congestion or topology changes. To cope with this packet loss, a node may send additional data packets to sustain $\widehat{r}_{MIN}$.

Estimating transmission reliability. We estimate the current reliability across the network (r_{NET}) using ordinary application data exchanges and a moving average filter with a window length W_{LOSS}. When we receive an acknowledgment for a data packet within the timeout $t_{Timeout}$, we count the transmission as successful (r_{PKT} = 1). Otherwise, we count it as unsuccessful (r_{PKT} = 0). After every transmission, the node calculates the current r_{NET} as:

$$r_{NET} = \frac{1}{W_{LOSS}} \sum_{j=1}^{W_{LOSS}} r_{PKT}(j). \tag{6.16}$$

Adapting transmission attempts. Based on the current r_{NET} value, the node dynamically adapts the number of necessary application transmission attempts to sustain $\widehat{r}_{MIN}$. Because the packets between BLE node and cloud server are short and infrequent compared to other Internet traffic, we neglect the effect of network congestion caused by our BLE nodes.

Similar to other research [227,229], we assume an independent and identically distributed packet loss with success probability of r_{NET} and model packet loss using a binomial distribution:

$$P(X = k) = \binom{n}{k} r_{TX}^{k} (1 - r_{NET})^{n-k}, \tag{6.17}$$

where $P(X = k)$ is the probability that exactly k data packets out of n transmission attempts are successfully exchanged.

To achieve our end-to-end reliability requirement, at least one transmission attempt needs to be successful. Therefore, $\widehat{r}_{MIN}$ can be calculated as:

$$\widehat{r}_{MIN} = P(X \geq 1) = 1 - P(X = 0). \tag{6.18}$$

Using Equation 6.17 and $\binom{n}{0}$ = 1, we can calculate $\widehat{r}_{MIN}$ as:

$$\widehat{r}_{MIN} = 1 - (1 - r_{NET})^{n}. \tag{6.19}$$

This means that for a given $\widehat{r}_{MIN} < 1$ and an estimated r_{NET}, a node can calculate the number of necessary transmission attempts (N_{TX}) as:

$$N_{TX} = n = \begin{cases} \left\lceil \frac{log(1-\widehat{r}_{MIN})}{log(1-r_{NET})} \right\rceil & \text{if } r_{NET} < 1 \\ 1 & \text{if } r_{NET} = 1 \end{cases} \tag{6.20}$$

By continuously monitoring r_{NET} and adapting the number of packet transmissions N_{TX}, a node can meet the given end-to-end reliability $\widehat{r}_{MIN}$.

6.4.2 Meeting Latency Requirements: Adapting CI & SL

In this section, we show how a node can meet a given upper end-to-end latency bound when connected to a router that has constraints on its BLE radio duty cycle.

One reason for such radio duty cycle constraints on the router may be a limited power supply (*e.g.*, a smartphone as router operating on a battery). Another reason may be that the router needs to sustain a large number of BLE connections simultaneously, which limits the radio duty cycle (RDC) available to any individual BLE connection. Independent of the actual reason of the RDC constraints, the BLE node should require as little BLE radio time on the router as possible in such a scenario. Therefore, the BLE node needs to adapt the BLE connection interval (CI) and the BLE slave latency (SL) to meet its latency bounds while limiting its power consumption and the RDC on the router.

6.4.2.1 Transmitting Data

First, we show how a BLE node can adapt its CI and SL parameters to transmit data within a given maximum end-to-end latency ($\widehat{t}_{TX_{MAX}}$) and a given minimum end-to-end reliability ($\widehat{r}_{MIN}$) to a server. To sustain $\widehat{r}_{MIN}$, we split $\widehat{t}_{TX_{MAX}}$ into N_{TX} equal time slots and let the node initiate a transmission attempt in each of the resulting slots.

Using the model from Equation 6.4, the node calculates a bound for its BLE connection interval (CI) that allows to sustain $\widehat{t}_{TX_{MAX}}$ as:

$$CI \leq \frac{\widehat{t}_{TX_{MAX}}/N_{TX} - t_{NET_{MAX}} - t_{CE}}{n_{CE_{MAX}} \cdot \lceil D_{TX}/F_{TX} \rceil}. \tag{6.21}$$

After that, the node calculates its BLE slave latency (SL) as:

$$SL = n_{CE_{MAX}} \cdot \lceil D_{TX}/F_{TX} \rceil - 1, \tag{6.22}$$

to minimize its power consumption. This SL value allows the node to skip connection events, where only mandatory keep-alive packets would be exchanged (as discussed in Section 2.1.4).

6.4.2.2 Receiving Data

Next, we show how a BLE node can adapt its CI and SL parameters to receive data within a given maximum end-to-end latency ($\widehat{t}_{RX_{MAX}}$) and a given minimum end-to-end reliability ($\widehat{r}_{MIN}$) from a server. In this scenario, the server is responsible for accounting for any loss over the network and adapting its transmission attempts. The node only receives the calculated number of necessary transmission attempts (N_{RX}) from the server.

Using Equation 6.9, the node calculates its CI bound as:

$$CI \leq \frac{\widehat{t}_{RX_{MAX}}/N_{RX} - t_{NET_{MAX}} - t_{CE}}{n_{CE_{MAX}} \cdot \lceil D_{RX}/F_{RX} \rceil + SL}. \tag{6.23}$$

Furthermore, the node chooses $SL = 0$ in this scenario, which limits the RDC of the BLE radio on the router (as discussed in Section 2.1.4).

6.4.3 Meeting Latency Requirements: Adapting SL Only

In contrast to Section 6.4.2, this section shows how a BLE node can meet given end-to-end requirements when connected to a router without constraints on its RDC.

In this scenario, the router configures the BLE connection to use the smallest possible BLE connection interval (CI) it can sustain during BLE connection setup. The node uses the configured CI and only dynamically adapts the BLE slave latency (SL) of the BLE connection.

6.4.3.1 Transmitting Data

Similar to Section 6.4.2.1, the node calculates N_{TX} to meet the given minimum end-to-end reliability ($\widehat{r}_{MIN}$). Next, the node only needs to calculate SL as:

$$SL = \lceil I_{TX}/CI \rceil - 1, \tag{6.24}$$

where I_{TX} is the interval at which the application is issuing packet transmissions and CI is the used BLE connection interval. In this equation, $\lceil I_{TX}/CI \rceil$ measures the maximum number of BLE connection events between two data packet transmissions by the node.

A node using this SL configuration only needs to wake up when it has data to transmit. During the remaining connection events, where only keep-alive packets would be exchanged, the node sleeps to minimize power consumption.

6.4.3.2 Receiving Data

As in the data transmission scenario, the BLE node uses the given CI of the BLE connection and only adapts the SL to meet the end-to-end latency requirements for receiving packets from the server. Using Equation 6.9, the node calculates SL as:

$$SL \leq \frac{\widehat{t}_{RX_{MAX}}/N_{RX} - t_{NET_{MAX}} - t_{CE}}{CI} - n_{CE_{MAX}} \cdot \left\lceil \frac{D_{RX}}{F_{RX}} \right\rceil . \tag{6.25}$$

This SL value allows the node to skip most unnecessary BLE connection events while meeting the desired $\widehat{t}_{RX_{MAX}}$ latency bound.

6.4.4 Discussion

In some scenarios, a node may need to meet end-to-end latency bounds on transmitting and receiving packets simultaneously. To do so, the node independently calculates parameters for transmitting (CI_{TX} and SL_{TX}) and receiving (CI_{RX} and SL_{RX}) using the formulas above. After that, the node uses a $CI = MINIMUM(CI_{TX}, CI_{RX})$ and a $SL = MINIMUM(SL_{TX}, SL_{RX})$ as its BLE connection parameters to meet the given end-to-end requirements.

6.5 Implementation

We implement our proposed adaptation strategies on the Nordic Semiconductor `nRF52840 DK` [159] using the Zephyr RTOS [217]. While we use the `nRF52840` platform for our experiments, our code also runs on all Nordic Semiconductor `nRF52` platform variants. Because we use only standardized BLE functionality, our solutions can easily be ported to other hardware platforms supporting BLE version 4.1 and above.

We extend Zephyr's existing IPv6-over-BLE application on the node by our proposed adaptation mechanisms. After the router has successfully established an IPv6-over-BLE connection with the node, the node transmits a UDP packet with an IPv6 packet length of 128 bytes to the AWS cloud server located in Frankfurt, Germany. As soon as the node receives the first valid server acknowledgment, the node starts to probe $t_{NET_{MAX}}$ using a probe burst length of $L_{Probe} = 10$, a probing interval of $I_{Probe} = 1000\,s$, and an ACK timeout $t_{Timeout} = 2000\,ms$. When a first $t_{NET_{MAX}}$ estimation is available, the node calculates suitable CI and SL parameters using our equations in Section 6.4 and configures these parameters using the standardized BLE connection parameter update process from Section 2.1.4.

We monitor the n_{CE} values of the BLE connection between node and router as shown in Section 4.2. Whenever a new n_{CE_f} value is available, the node is notified via a callback and may update the BLE connection parameters as described in Section 6.4. In our experiments, the BLE connection does not make use of channel blacklisting and uses the 2M PHY mode of BLE.

Table 6.3: Delayed packet (delayed) and maximum number of subsequently delayed packets (max. delays) of the different node configurations under heavy Wi-Fi interference.

Node configuration	delayed [%]	max. delays
CI = 7.5 ms & SL = 0	0.00 ± 0.00	0
Adapt SL only	0.00 ± 0.00	0
Adapt CI & SL	0.61 ± 0.22	2
CI = 1000 ms & SL = 0	50.64 ± 2.64	25

6.6 Evaluation

In this section, we reuse our testbed described in Section 6.1 to experimentally evaluate our two proposed adaptation strategies, which we call `Adapt CI & SL` (proposed in Section 6.4.2) and `Adapt SL only` (proposed in Section 6.4.3). We compare our proposed approaches against a node using the fastest possible static BLE connection parameters ($CI = 7.5\,ms$ & $SL = 0$) and a node using static, power-efficient BLE connection parameters ($CI = 1000\,ms$ & $SL = 0$).

6.6.1 Systematic Evaluation

We start by showing that our proposed solutions can cope with dynamic changes in the BLE subnet and with sudden changes on the external network path. In this systematic evaluation, we focus on a node sending packets to the server using a wired Internet connection and configure the BLE nodes to meet a $\widehat{r}_{MIN} = 99\%$ and a $\widehat{t}_{TX_{MAX}} = 1000\,ms$.

6.6.1.1 Changes in the BLE subnet

First, we experimentally investigate how our proposed adaptation strategies can adapt to dynamic changes in the local BLE subnet, *e.g.*, caused by Wi-Fi interference.

Setup. We reuse our testbed setup described in Section 6.1 and establish a connection between node and router. After an initial phase of 60 s, we use two different Raspberry Pi 3B devices in our testbed to create continuous and heavy Wi-Fi interference on two different Wi-Fi channels.

Results. Table 6.3 shows the delayed packets for each of the four different node configuration over 10 min after the Wi-Fi jamming was started. Whenever a UDP packet exceeds the specified end-to-end latency bound $\widehat{t}_{TX_{MAX}} = 1000\,ms$, we count it as delayed. The table shows the overall number of delayed UDP packets (delayed) during the experiment and the maximum number of subsequently delayed packets (max. delays).

The data in Table 6.3 shows that both of our adaptation approaches (`Adapt SL only` and `Adapt CI & SL`) can successfully cope with sudden link-layer errors in the BLE subnet. While the `Adapt SL only` approach results in no packet delays, also the `Adapt CI & SL` approach results in below 1% of all packet transmissions being delayed, and provides a significant improvement compared to a fixed energy-efficient setting.

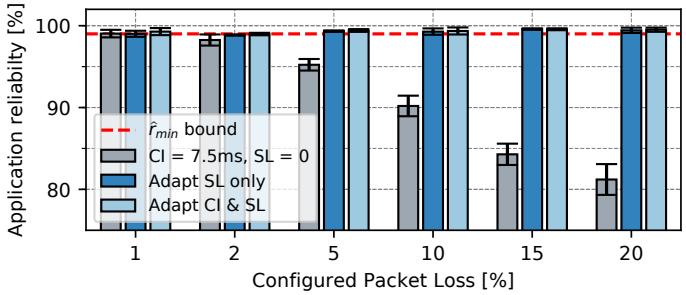


Figure 6.4: Measured end-to-end application reliability for different node configurations and configured packet loss. Adapted from Publication E.

6.6.1.2 Changes in network loss

Next, we measure how our proposed adaptation approaches are able to cope with sudden changes to the reliability of the external network path (r_{NET}), *e.g.*, caused by router buffer congestion.

Setup. Similar to Section 6.6.1.1, we establish a connection between node and server in our testbed. After an initial phase of 60 s, we lower r_{NET} by either 1, 2, 5, 10, 15, or 20% and measure the resulting end-to-end application reliability, *i.e.*, how many node packets are successfully received within $\widehat{t_{TX_{MAX}}}$, for 600 s. In these experiments, we use the standard Traffic Control (`tc`) tool with its Network Emulator (`netem`) of Linux to reproducibly lower r_{NET}. To mimic symmetric network loss, we use `tc` on all outgoing traffic on the router and the cloud server.

Results. Figure 6.4 shows the results of our experiments. The data shows that both of our proposed adaptation approaches (`Adapt SL only` and `Adapt CI & SL`) can successfully cope with loss across the external network path by increasing the transmission attempts sent for every application data packet.

6.6.2 Comparison

In this section, we compare the performance of our proposed adaptation approaches to other node configurations in different environments.

Setup. We reuse our testbed (see Section 6.1) to measure the end-to-end communication latency and reliability, as well as the power consumption of nodes. For every experimental run, we program a node with one of the different node configurations, establish the connection between node and router, and initially wait for 60 s before we start our measurements.

Per node configuration, we measure the number of application packets exceeding the specified end-to-end latency bound (delayed pkts.) and radio duty cycle of the BLE radio on the router (RDC_{Router}) caused by the node communication. Furthermore, we measure the current consumption of the BLE node (I_{Node}) using our testbed to evaluate the power efficiency of the different node configurations. During these measurements, all logging and unused peripherals on the BLE nodes are disabled to minimize power consumption.

In addition to the node configurations from our previous experiments, we also investigate the behavior of the adaptation approach presented in Section 4.2, which we call `NCE only`. The

`NCE only` approach only reacts to changes in the local BLE subnet and does not account for the behavior of the external network path. Because the `NCE only` approach only meets latency bounds on transmissions from node to router, it is not included in our packet reception experiments.

Experimental environments. To evaluate the performance of our adaptation approaches in different conditions, we use four different experimental environments:

Wired & No Interf. This scenario uses a wired Internet connection with low variability and does not introduce any Wi-Fi interference into the BLE subnet.

Wired & Wi-Fi Interf. This scenario uses the same wired Internet connection as above, but introduces continuous and heavy Wi-Fi interference on two different Wi-Fi channels in the BLE subnet.

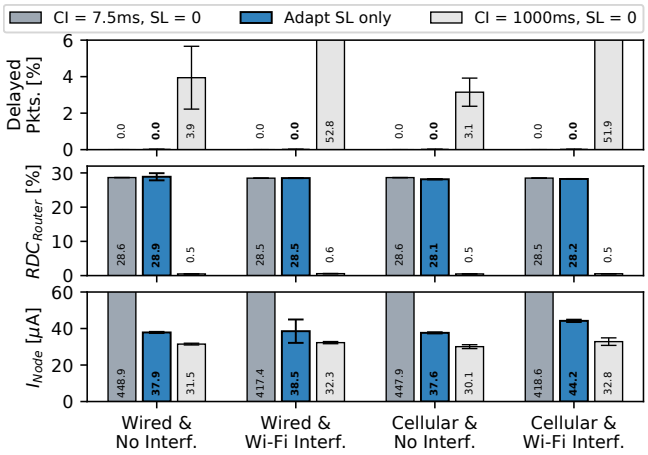
Cellular & No Interf. This scenario uses a 4G connection to connect the router to the Internet and does not introduce any Wi-Fi interference into the BLE subnet. As shown in Section 6.1, the cellular Internet connection experiences longer and more varying delays across the network path.

Cellular & Wi-Fi Interf. This scenario uses the 4G Internet connection and introduces continuous Wi-Fi interference in the BLE subnet. This leads to delays in the BLE subnet, caused by link-layer retransmissions due to Wi-Fi, and on the external network path, caused by the cellular connection.

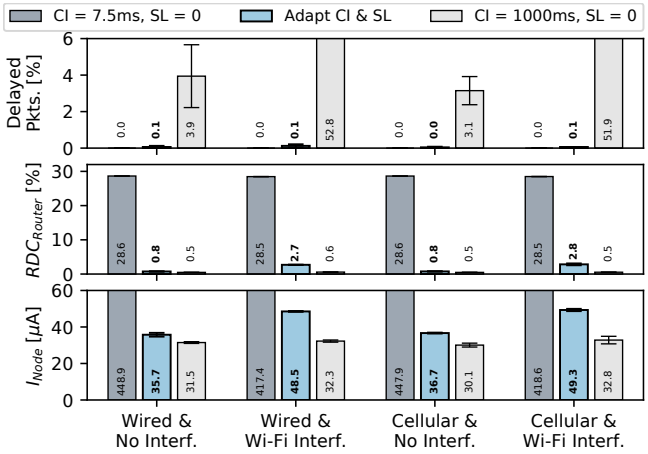
Results. Figure 6.5 shows the performance of our proposed adaptive approaches trying to meet a $\widehat{t}_{TX_{MAX}} = 1000\,ms$ in our four different experimental environments. In every subfigure in Figure 6.5 one adaptive approach is compared to the fastest possible static BLE connection parameters ($CI = 7.5\,ms$ & $SL = 0$) and energy-efficient static BLE connection parameters ($CI = 1000\,ms$ & $SL = 0$). Figure 6.5(a) shows that our proposed `Adapt SL only` approach successfully meets the specified end-to-end latency bound, resulting in no delayed or lost packets, at the cost of a slightly increased node current consumption (I_{Node}) compared to the fixed, energy-efficient setting. As expected and discussed in Section 6.4.3, the `Adapt SL only` approach results in a high RDC at the router (RDC_{Router}), which may not be suitable for some applications. Applications that need to limit the routers RDC (RDC_{Router}) can use our `Adapt CI & SL` approach to successfully meet the specified end-to-end latency bound while also limiting the node's current consumption (I_{Node}), as shown by the data in Figure 6.5(b). Furthermore, we see in Figure 6.5 that our `Adapt SL only` and `Adapt CI & SL` approaches significantly outperform the `NCE only` approach in meeting $\widehat{t}_{TX_{MAX}}$, by at least 100%.

Figure 6.6 shows the performance of our proposed adaptive approaches trying to meet a $\widehat{t}_{RX_{MAX}} = 1000\,ms$ in our four different experimental environments. Like in Figure 6.5, each subfigure in Figure 6.6 shows one adaptive approach compared to the fixed $CI = 7.5\,ms$ & $SL = 0$ and $CI = 1000\,ms$ & $SL = 0$ configurations. The data in Figure 6.6 shows that both of our proposed adaptation approaches successfully meet the specified end-to-end latency bound while limiting the current consumption of the node (I_{Node}). As expected, our `Adapt SL only` approach (shown in Figure 6.6(a)) results in less delayed packets than the `Adapt CI & SL` approach (shown in Figure 6.6(b)) at the cost of a significantly higher RDC_{Router}.

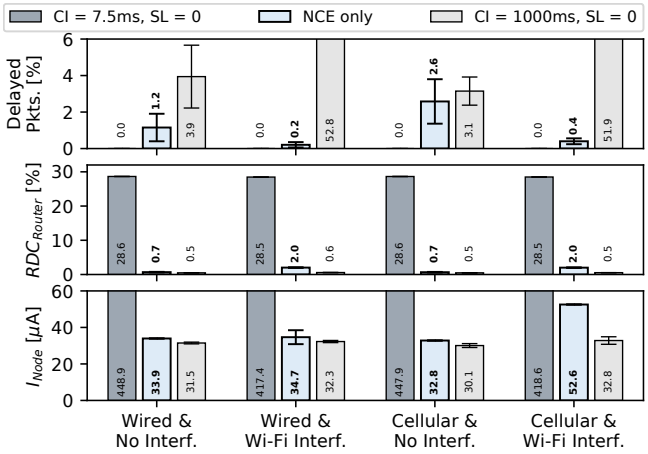
Figure 6.7 and 6.8 show an overview of the performance of all fixed and adaptive node configurations when transmitting or receiving data, respectively. Using the same data as Figure 6.5 and 6.6, these two figures provide a comparison of all node configurations next to each other. Again, we can clearly see that our adaptive approaches drastically improve communication performance and successfully meet the given end-to-end latency requirements.



(a) Performance of our `Adapt SL only` approach.

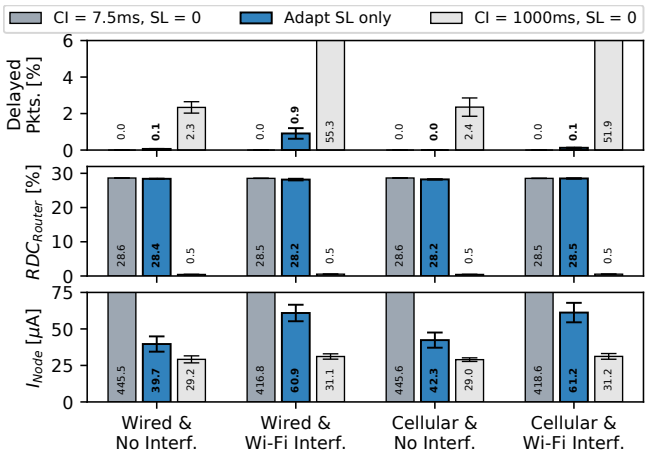


(b) Performance of our `Adapt CI & SL` approach.

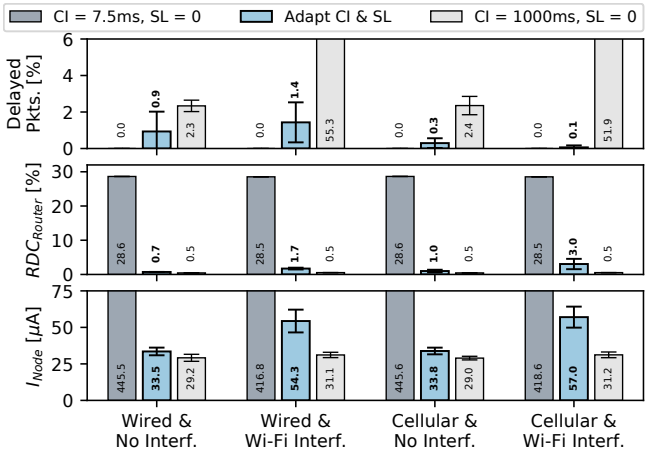


(c) Performance of our `NCE only` approach.

Figure 6.5: Detailed performance of our proposed adaptation approaches for meeting $\widehat{t}_{TX_{MAX}} = 1000\,ms$ on transmitting packets with a length of 128 bytes to a cloud server. Every subfigure shows the performance of a single adaptation approach in comparison to the fastest possible static BLE connection parameter setting ($CI = 7.5\,ms$ & $SL = 0$) and to a static, power-efficient BLE connection parameter setting ($CI = 1000\,ms$ & $SL = 0$).



(a) Performance of our `Adapt SL only` approach.



(b) Performance of our `Adapt CI & SL` approach.

Figure 6.6: Detailed performance of our proposed adaptation approaches for meeting $\widehat{t}_{RX_{MAX}} = 1000\,ms$ on receiving packets with a length of 128 bytes from a cloud server. Every subfigure shows the performance of a single adaptation approach in comparison to the fastest possible static BLE connection parameter setting ($CI = 7.5\,ms$ & $SL = 0$) and to a static, power-efficient BLE connection parameter setting ($CI = 1000\,ms$ & $SL = 0$).

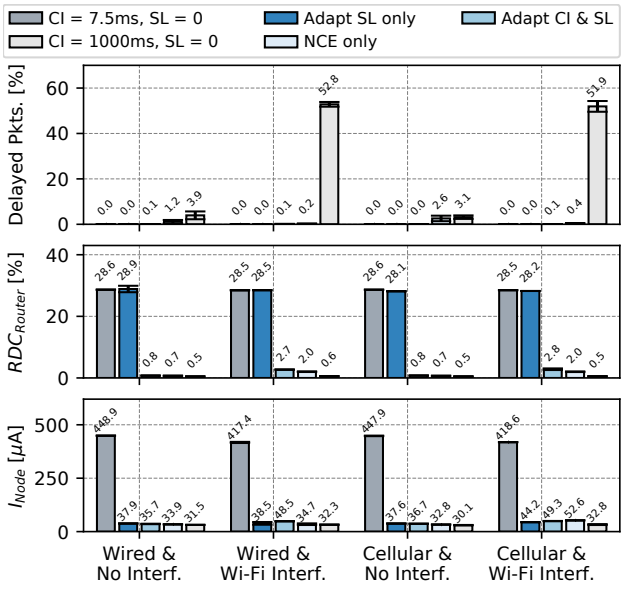


Figure 6.7: Performance overview of fixed and adaptive node configurations for meeting $\widehat{t}_{TX_{MAX}} = 1000\,ms$ on transmitting packets with a length of 128 bytes to a cloud server. Adapted from Publication E.

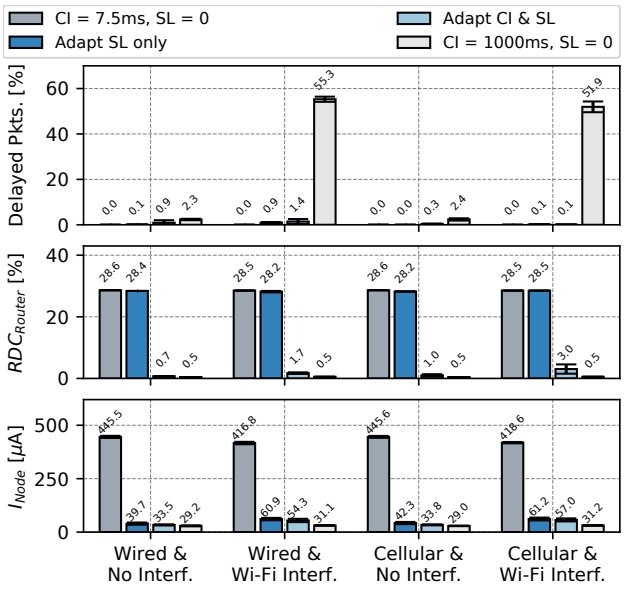


Figure 6.8: Performance overview of fixed and adaptive node configurations for meeting $\widehat{t}_{RX_{MAX}} = 1000\,ms$ on receiving packets with a length of 128 bytes from a cloud server. Adapted from Publication E.

6.7 Summary

In this chapter, we have shown how BLE-based IoT devices can meet given end-to-end communication requirements when exchanging data with a cloud server. Towards this goal, we have derived a new end-to-end BLE model, have designed an end-to-end loss and delay estimation approach, and have shown how BLE nodes can meet a given minimum end-to-end reliability and a given maximum end-to-end latency, even in harsh environments experiencing dynamic changes.

Because our solutions use only standardized BLE functionality and adhere to the end-to-end principle of IP, they can easily be used by existing and future BLE-based IoT applications to improve their performance.

7 Conclusion and Future Work

BLE communication is increasingly used in safety- and time-critical application domains, such as industrial monitoring systems and smart healthcare, due to its ultra-low power consumption and its wide availability in existing consumer devices. While BLE devices operating in such domains need to limit their power consumption to reach a suitable battery lifetime, these devices also need to sustain stringent communication requirements, such as a given minimum reliability or a given maximum latency, to ensure a safe operation.

Sustaining such communication requirements on battery-powered BLE devices is difficult because these devices need to cope with persistent or transient communication problems (*e.g.*, external radio interference and fading effects in the local BLE network, or sudden loss and delays over the Internet) while minimizing the power draw of the wireless communication. This dissertation tackles this problem and shows how to create dependable BLE applications that have a low power consumption and need to meet given end-to-end communication requirements.

7.1 Contributions

Towards this goal, we have developed three main contributions in this thesis.

Supporting time-critical data exchange over BLE connections. First, we show how BLE devices can reliably exchange data within a given latency bound over a BLE connection in real-world application environments, where devices may experience external radio interference, weak signal strength, or fading effects. To achieve time-critical communication, we design three novel BLE adaptation mechanisms that allow off-the-shelf BLE devices to optimize their communication performance at runtime. Our *BLE channel management* significantly increases the link-layer reliability of a BLE connection by passively monitoring the quality of individual BLE data channels and by selecting only channels with high quality for data exchange. Our *BLE PHY mode adaptation* dynamically chooses the most suitable PHY mode of a BLE connection to sustain a given minimum link-layer reliability while limiting the power consumption of a BLE device. Devices using our *BLE connection parameter adaptation* passively monitor the delay of individual BLE transmissions and dynamically adapt the used BLE connection parameters to sustain a given maximum transmission latency while minimizing power consumption.

Connecting BLE devices to the IoT using IPv6. Second, we present BLEach, the first open-source, full-fledged IPv6-over-BLE communication stack for constrained devices. BLE devices using BLEach can directly exchange IPv6 packets with any other IP-compliant device on the Internet, without requiring a gateway to translate standard GATT-based BLE data into IPv6 packets. Furthermore, BLEach exposes the key parameters of IPv6 over BLE, allowing to optimize

IPv6-over-BLE communication performance at runtime. In this dissertation, we use the exposed parameters to support multiple IPv6 traffic flows over IPv6 over BLE, each with its own QoS level.

Meeting end-to-end requirements in BLE-based IoT applications. Finally, we show how BLE devices can communicate with a server on the Internet within given end-to-end reliability and latency bounds. Using our novel end-to-end model, BLE devices can capture and cope with any communication problems across the whole network path, such as external interference in the local BLE subnet or sudden packet loss or delays on the Internet. Our approach is fully compliant with the end-to-end principle of IP and allows BLE nodes to meet given end-to-end communication requirements while minimizing their power consumption, independent of the used Internet connection or the RF environment in the local BLE subnet.

All of our contributions are fully compliant to the BLE specification [24] and can be implemented on off-the-shelf BLE devices to significantly improve their communication performance.

7.2 Future Work

The focus of this dissertation is to enable low-power, reliable, and time-critical IoT applications based on BLE. Although the work in this thesis presents a significant step towards this goal, it has several limitations and possibilities for optimization that may be addressed in future work. Some of the future improvements to the work in this thesis may be:

Providing an all-in-one solution. Although all our improvements to connection-based BLE are complementary with each other, we have experimentally evaluated each of our individual improvement on its own. This has allowed us to accurately measure the communication improvements caused by any proposed contribution. However, a BLE node may use all proposed solutions in combination, *e.g.*, our end-to-end work in combination with our BLE channel management and PHY mode adaptation, to significantly improve its performance even further.

Supporting upcoming BLE features. At the time of writing this thesis, the specifications of BLE version 5.2 [26] and version 5.3 [27] have already been publicly released. These versions provide additional BLE features, such as ISO channels or a faster BLE parameter update, that significantly improve the reliability and real-time capabilities of BLE communication, as discussed in Section 2.1. Unfortunately, the currently available BLE devices have no or only limited support for these new BLE features.

Nevertheless, our proposed solutions are fully compliant to these newer BLE versions and future work may combine the work of this dissertation with this new BLE functionality. For example, BLE nodes may use BLE ISO channels instead of ordinary connection-based BLE to exchange data with a router in real-time. BLE devices may use the faster BLE parameter update in combination with our adaptation approaches to react to changes in the network even faster. Compared to the existing BLE parameter update procedure that uses a fixed adaptation period of 6 connection events, the faster BLE parameter update procedure adapts the BLE connection parameters as fast as possible (*e.g.*, typically within 2 connection events). Nodes may use our link quality estimation approach to dynamically adapt their transmission power - a feature added with BLE version 5.2 - to further improve their energy efficiency and reliability.

Migrating solutions to multi-hop BLE networks. This dissertation focuses on optimizing the

performance of nodes operating in single-hop BLE networks. Nevertheless, some of our contributions may be used by multi-hop BLE networks, such as BLE mesh networks or networks based on concurrent transmissions. For example, multi-hop BLE networks may use our BLE channel management to improve the link-layer reliability of communication. Moreover, nodes may use our BLE PHY mode adaptation to dynamically adapt the used PHY mode of individual links or on a network level. Edge nodes may use our end-to-end delay estimation approach to measure and control their transmission delay across large-scale BLE networks.

Combining our adaptation approaches with application-layer improvements. In this dissertation, we have presented how to optimize connection-based BLE communication via link-layer improvements. Future research may combine our optimizations with application-layer improvements used in state-of-the-art Tactile Internet solutions, such as data reduction, data compression, or AI-based content caching. Combining our improvements with application-layer optimizations will be another significant step towards highly energy-efficient, reliable, and real-time IoT applications based on BLE.

8
Publications

This thesis is based on the following peer-reviewed papers and articles:

A. M. Spörk, C. A. Boano, M. Zimmerling, and K. Römer. BLEach: Exploiting the Full Potential of IPv6 over BLE in Constrained Embedded IoT Devices. In *Proceedings of the* 15^{th} *ACM International Conference on Embedded Networked Sensor Systems (SenSys)*. ACM, Nov. 2017

B. M. Spörk, C. A. Boano, and K. Römer. Improving the Timeliness of Bluetooth Low Energy in Dynamic RF Environments. *ACM Transactions on Internet of Things*, 1(2), Apr. 2020

C. M. Spörk, C. A. Boano, and K. Römer. Performance and Trade-offs of the new PHY Modes of BLE 5. In *Proceedings of the International Workshop on Pervasive Systems in the IoT Era (PERSIST-IoT)*. ACM, July 2019

D. M. Spörk, J. Classen, C. A. Boano, M. Hollick, and K. Römer. Improving the Reliability of Bluetooth Low Energy Connections. In *Proceedings of the* 17^{th} *International Conference on Embedded Wireless Systems and Networks (EWSN)*. Junction Publishing, Feb. 2020

E. M. Spörk, M. Schuß, C. A. Boano, and K. Römer. Ensuring End-to-End Dependability Requirements in Cloud-based Bluetooth Low Energy Applications. In *Proceedings of the* 18^{th} *International Conference on Embedded Wireless Systems and Networks (EWSN)*. Junction Publishing, 2021

Related peer-reviewed papers, posters, and demos presented at International Conferences that are not included in this thesis:

1. M. Spörk, M. Schuß, C. A. Boano, and K. Römer. An Open-Source IPv6 over BLE Stack for Contiki. In *Proceedings of the* 14^{th} *International Conference on Embedded Wireless Systems and Networks (EWSN), poster session*. ACM, Feb. 2017
2. T. Renzler, M. Spörk, C. A. Boano, and K. Römer. Improving the Efficiency and Responsiveness of Smart Objects using Adaptive BLE Device Discovery. In *Proceedings of the* 4^{th} *International Workshop on Experiences with the Design and Implementation of Smart Objects (SMARTOBJECTS)*. ACM, June 2018
3. M. Spörk, C. A. Boano, and K. Römer. Improving the Timeliness of Bluetooth Low Energy in Noisy RF Environments. In *Proceedings of the* 16^{th} *International Conference on Embedded Wireless Systems and Networks (EWSN)*. Junction Publishing, Feb. 2019
4. J. Classen, M. Spörk, C. A. Boano, K. Römer, and M. Hollick. Analyzing Bluetooth Low Energy Connections on Off-the-Shelf Devices. In *Proceedings of the* 17^{th} *International Conference on Embedded Wireless Systems and Networks (EWSN), demo session*. Junction Publishing, Feb. 2020
5. L. Botler, M. Spörk, K. Diwold, and K. Römer. Direction Finding with UWB and BLE: A Comparative Study. In *2020 IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 44–52. IEEE, 2020
6. C. Gomez and S. Darroudi and T. Savolainen and M. Spörk. IETF RFC Draft - IPv6 Mesh over BLUETOOTH(R) Low Energy using IPSP. `https://datatracker.ietf.org/doc/draft-ietf-6lo-blemesh/`, 2021

Publication A

M. Spörk, C. A. Boano, M. Zimmerling, and K. Römer. BLEach: Exploiting the Full Potential of IPv6 over BLE in Constrained Embedded IoT Devices. In *Proceedings of the* 15^{th} *ACM International Conference on Embedded Networked Sensor Systems (SenSys)*. ACM, Nov. 2017

DOI: 10.1145/3131672.3131687
Link: `https://doi.org/10.1145/3131672.3131687`

Abstract. The ability to fine-tune communication performance is key to meeting the requirements of Internet of Things applications. While years of low-power wireless research now allows developers to fully optimize the performance of applications built on top of IEEE 802.15.4, this has not yet happened with Bluetooth Low Energy (BLE), whose networking performance is still *largely unexplored* and whose potential is *not yet fully exploited*. Indeed, BLE radios are often treated as a *black box*, because they are meant to only execute data transfer commands and manufacturers build BLE soft devices with closed-source network stacks. As a result, developers working with BLE cannot modify the radio driver or the link-layer, and hence have no direct control over radio duty cycling and packet re-transmissions.

To tackle these challenges, we analyze and model how specific BLE features can be used to fine-tune communication performance at run-time. We further present the design and implementation of BLEach, an IPv6-over-BLE stack that exposes *tuning knobs* for controlling the energy usage and timeliness of BLE transmissions and that allows to enforce a variety of quality-of-service (QoS) metrics. We design three exemplary modules for BLEach providing novel BLE functionality: adaptive radio duty cycling, IPv6-over-BLE traffic prioritization and multiplexing, as well as indirect link-quality monitoring. We integrate BLEach into Contiki and release its code, thus addressing the lack of a full-fledged open-source IPv6-over-BLE stack. Experiments demonstrate that BLEach is lightweight, interoperable with other standard-compliant devices, and reduces energy costs by up to 50 % while giving QoS guarantees by quickly adapting to changes in interference, traffic priority, and traffic load.

My contribution. I am the main author of this publication and was responsible for the design and implementation of BLEach, as well as, executing all experiments presented in this paper. Fortunately, my colleague Rainer Hofmann helped me in performing the communication range measurements presented in Section 6.4 of the paper. Carlo Alberto Boano significantly helped me in defining the scientific questions discussed in this paper, coming up with the corresponding technical solutions, and planning the experimental evaluation. Furthermore, my co-authors significantly helped me in writing and structuring the paper. I presented the paper at SenSys'17. This work was executed in collaboration with TU Dresden.

BLEach: Exploiting the Full Potential of IPv6 over BLE in Constrained Embedded IoT Devices

Michael Spörk
Graz University of Technology
michael.spoerk@tugraz.at

Carlo Alberto Boano
Graz University of Technology
cboano@tugraz.at

Marco Zimmerling
Dresden University of Technology
marco.zimmerling@tu-dresden.de

Kay Römer
Graz University of Technology
roemer@tugraz.at

ABSTRACT

The ability to fine-tune communication performance is key to meeting the requirements of Internet of Things applications. While years of low-power wireless research now allows developers to fully optimize the performance of applications built on top of IEEE 802.15.4, this has not yet happened with Bluetooth Low Energy (BLE), whose networking performance is still *largely unexplored* and whose potential is *not yet fully exploited*. Indeed, BLE radios are often treated as a *black box*, because they are meant to only execute data transfer commands and manufacturers build BLE soft devices with closed-source network stacks. As a result, developers working with BLE cannot modify the radio driver or the link-layer, and hence have no direct control over radio duty cycling and packet re-transmissions.

To tackle these challenges, we analyze and model how specific BLE features can be used to fine-tune communication performance at run-time. We further present the design and implementation of BLEach, an IPv6-over-BLE stack that exposes *tuning knobs* for controlling the energy usage and timeliness of BLE transmissions and that allows to enforce a variety of quality-of-service (QoS) metrics. We design three exemplary modules for BLEach providing novel BLE functionality: adaptive radio duty cycling, IPv6-over-BLE traffic prioritization and multiplexing, as well as indirect link-quality monitoring. We integrate BLEach into Contiki and release its code, thus addressing the lack of a full-fledged open-source IPv6-over-BLE stack. Experiments demonstrate that BLEach is lightweight, interoperable with other standard-compliant devices, and reduces energy costs by up to 50 % while giving QoS guarantees by quickly adapting to changes in interference, traffic priority, and traffic load.

CCS CONCEPTS

• **Computer systems organization** → **Dependable and fault-tolerant systems and networks**; • **Networks** → *Network protocols*; *Network performance evaluation*;

KEYWORDS

Bluetooth Low Energy, Contiki, Internet of Things, IPv6 over BLE.

SenSys '17, Delft, Netherlands

DOI: 10.1145/3131672.3131687

ACM Reference format:
Michael Spörk, Carlo Alberto Boano, Marco Zimmerling, and Kay Römer. 2017. BLEach: Exploiting the Full Potential of IPv6 over BLE in Constrained Embedded IoT Devices. In *Proceedings of SenSys '17, Delft, Netherlands, November 6–8, 2017*, 14 pages.
DOI: 10.1145/3131672.3131687

1 INTRODUCTION

Bluetooth Low Energy (BLE) is a low-power wireless technology that has gained popularity in recent years due to its wide adoption in consumer devices, such as smartphones, wearables, and laptops. These devices can act as gateways, seamlessly integrating "smart objects" into the Internet of Things (IoT). This way, BLE supports a range of powerful applications, from home entertainment and automation to health-care monitoring and fitness tracking.

Challenges. Approved in October 2015, RFC 7668 describes how IPv6 packets can be exchanged using BLE connections (IPv6 over BLE) [24], allowing any "smart object" supporting BLE to communicate with any other IPv6-enabled device using headless routers. This has paved the way for an even richer set of applications using BLE technology. It also represents a significant leap in IoT research, where for almost a decade IPv6 support for low-power wireless networks was limited to IPv6 over IEEE 802.15.4 (6LoWPAN).

Unfortunately, even two years later, little is known about how to optimize BLE's performance and how to unveil its full potential, especially in combination with IPv6. Most BLE works found in the literature are based on the connection-less mode, which does not support IPv6. Moreover, BLE radios are often treated as a *black box* for two reasons. First, they are typically built as drop-in communication peripherals attached to a host processor that simply executes data transfer commands [37]. Second, manufacturers often build BLE soft devices with closed-source network stacks provided as libraries in binary format [35]. As a result, developers cannot modify the BLE radio driver and link-layer implementations, and hence have *no explicit control* over link-layer (re-)transmissions and radio duty cycling, which largely determine application performance.

This state of affairs represents a significant problem, as the lower layers in the protocol stack must be *tuned at runtime* to meet the different requirements of IoT applications operating in dynamic environments [4, 14, 36, 38]. For instance, some applications may need to minimize energy consumption for economic viability, while still ensuring timely delivery of alarm messages in response to, for example, deteriorating vital signs of a patient [4]. The problem is further exacerbated by the lack of open-source stacks supporting IPv6 and BLE connection-based communication [41].

Thus, there is a need to gain a deeper understanding of how BLE features can be used to fine-tune communication performance. Designing an IPv6-over-BLE stack that exposes *tuning knobs* to control the energy expenditure and timeliness of BLE communications and that provides different quality-of-service (QoS) levels would greatly improve the performance of IoT applications using BLE.

Contributions. In this paper, we analyze BLE connection-based transmissions and model their energy consumption and latency as functions of key BLE parameters. We further present BLEach, the first full-fledged IPv6-over-BLE stack that exposes these parameters as tuning knobs to optimize the performance of single-hop BLE.

BLEach retains the intended simplicity of the BLE standard including its focus on single-hop networking, while supporting IPv6 functionality with minimal processing and memory overhead. Fully interoperable with other IPv6-compliant devices, BLEach is agnostic to the hardware platform and the application; that is, it supports single- and dual-core platforms through minimal changes to the lower layers, while completely hiding the platform specifics from the application. BLEach's modular design enables alternative implementations of BLE features or adding completely new functionality to improve performance. To show its capabilities, we enrich BLEach with three exemplary modules: the first one implementing a duty-cycling strategy that dynamically adapts BLE parameters to traffic load, the second one performing IPv6-over-BLE traffic prioritization and multiplexing, and the third one indirectly monitoring the link quality at run-time for black- and whitelisting of radio channels.

We integrate BLEach into Contiki and release its source code (http://www.iti.tugraz.at/BLEach), which makes it the first open-source IPv6-over-BLE stack supporting both resource-constrained master and slave devices. We thus fill an important gap identified by the research community [41], as further discussed in Sec. 7.

Experiments with BLEach on the popular TI CC2650 platform reveal that (*i*) BLEach is interoperable with RFC-compliant border routers, (*ii*) its minimal processing overhead and low memory footprint make it suitable for constrained embedded IoT devices, and (*iii*) the three exemplary modules we have developed significantly increase BLE's resilience and efficiency. Moreover, we compare the performance of BLEach and Contiki's default IPv6-over-IEEE 802.15.4 stack when running the same exemplary application on top, showing that BLEach is more energy efficient. We also compare the communication range achieved by BLE and IEEE 802.15.4 on the same platform outdoors, and are the first experimental study highlighting that BLE achieves a significantly lower range than IEEE 802.15.4 regardless of the employed transmission power.

In summary, this paper makes the following contributions:

- We analyze BLE's connection-based communication and model how specific BLE features and parameters can be used to fine-tune communication performance.
- We present *BLEach*, the first full-fledged IPv6-over-BLE stack that exposes these parameters to control the energy expenditure and timeliness of BLE communications.
- We show how BLEach empowers IoT research by enriching its architecture with novel features: a duty-cycling strategy that adapts BLE parameters to traffic load, QoS-aware BLE traffic prioritization and multiplexing, and adaptive channel blacklisting using indirect link-quality monitoring.

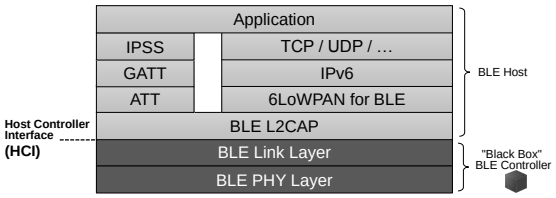


Figure 1: BLE architecture as specified by RFC 7668 [24].

- We compare BLEach to Contiki's default IPv6-over-IEEE 802.15.4 stack on the same platform and show that BLEach is more energy efficient, although BLE achieves a lower communication range than IEEE 802.15.4.

2 CHALLENGES OF IPV6 OVER BLE

RFC 7668 [24] specifies how BLE devices can communicate with any other IPv6-enabled device using headless routers. To exchange IPv6 packets, BLE devices form star networks with a central node, called *master*, acting as gateway to the Internet [24]. After establishing a connection with the master, BLE devices perform IPv6 neighbor discovery according to RFC 6775 [45], carry out IPv6 address autoconfiguration according to RFC 7136 [3], and exchange compressed IPv6 packets with the IPv6 prefix advertised by the master. Both the above standards and current BLE platforms pose several challenges that must be addressed in the design of an IPv6-over-BLE stack.

Support for connection-based mode. BLE supports two modes of communication: *connection-less* and *connection-based*. Most BLE-based IoT applications today use the *connection-less mode* [18, 25]: devices are either advertisers sending unidirectional broadcast messages, or scanners reading and processing the messages sent by advertisers. Connection-less communication is simple and supported by most existing IoT systems. IPv6 over BLE, instead, requires devices to communicate bidirectionally using the *connection-based mode*: devices first establish a connection to a master using the advertisement channels, and then exchange data during periodic connection events. However, BLE's connection-based mode is underexplored and not well supported by existing IoT systems.

Nature of BLE controllers. The BLE architecture consists of two main components, *controller* and *host*, which exchange commands via the Host Controller Interface (HCI) [24], as shown in Fig. 1. To simplify application development, the controller implementing the lowest layers of the stack acts as a *black box* to the upper layers: it autonomously handles link-layer (re-)transmissions and acknowledgments, and manages connection events according to a set of parameters passed through the HCI interface. Most BLE controllers are also closed source and not programmable, so developers cannot modify their operation or implement functionality that goes beyond the BLE standard. For example, the nRF52 only provides a proprietary HCI library to support the BLE controller running on the main processor, without the possibility to access its internals. Other platforms, like the TI CC2650 with its separate BLE core, provide a vendor-specific radio API that developers use to implement correct communication management. Although implementing such *open* BLE controllers is more complex, it also allows to accurately fine-tune and extend the functionality of the BLE radio.

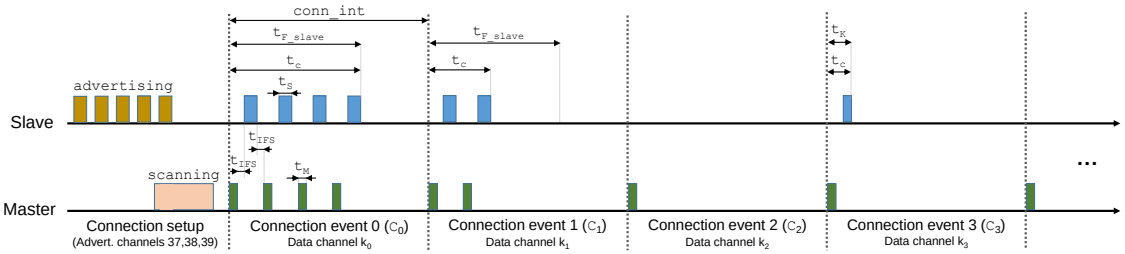


Figure 2: Example of a BLE connection-based data exchange between a slave and a master using a slave latency of 1.

Temporal decoupling from upper layers. Using the connectionless mode, a higher-layer protocol or the application can turn on the radio and transmit packets at any time. Using the connection-based mode, however, it is only possible to fill the radio buffer and wait for the BLE controller to transmit the packet(s) during the next connection event. Hence, in connection-based mode, packet transmissions are temporally decoupled from the upper layers. Moreover, there is no immediate feedback on ongoing transmissions.

Parametrization of connections. The only interface that developers have to control connection-based communication is a limited set of connection parameters passed through the HCI interface to the BLE controller. The set of connection parameters is traditionally chosen by the master, which informs its slaves about the parameters to be used for communication. How to select these connection parameters to meet certain performance goals is an open problem.

Runtime adaptation. Most BLE devices are slaves, communicating with a more powerful master acting as a gateway to the Internet, such as an embedded PC or smartphone. Such gateways use default parameters optimized for peak traffic load; constrained slaves using these parameters quickly drain their batteries. To avoid this problem, the BLE standard foresees the possibility to *negotiate* connection parameters: slave devices can ask the master to select other parameters that better fit their constraints and requirements. If the master provides this feature, slaves should not only compute *optimized* parameters and communicate them to the master, but also *adapt* the parameters at runtime in response to changes in traffic load. Although similar ideas have been explored in the context of IEEE 802.15.4 [30, 46], it is unclear which parameters are most relevant for BLE performance and how to determine optimized values for a set of parameters.

QoS support. IPv6-enabled BLE devices can exchange data with any other IPv6-enabled device, and hence likely experience different kinds of traffic (*e.g.*, periodic traffic due to sensing and sporadic ICMP traffic from the border router or other peer devices on the Internet). It is thus important to give these devices the ability to, for example, prioritize traffic and support different QoS levels via traffic prioritization and multiplexing. Such features are often required in practice but not prescribed by the standard. Conversely, IPv6 over BLE prescribes the use of *LE Credit-Based Flow Control*, whereby a device grants credits to its peer to prevent buffer overflows during a connection [6]. A practical IPv6-over-BLE stack must adhere to the standard while allowing for additional features to be added.

3 UNDERSTANDING CONNECTION-BASED BLE COMMUNICATION

To address the first five challenges mentioned above, we analyze BLE's connection-based mode in detail. We describe the sequence of operations during connection-based communication, discuss the impact of the most important parameters on BLE's performance, and derive analytical expressions that formalize these relationships.

3.1 Anatomy of Connection Events

BLE's connection-based mode provides bidirectional data transfer between a slave and a master. As shown in Fig. 2, after a setup phase, communication occurs in non-overlapping slots called *connection events*. The time between the start of two consecutive connection events, the *connection interval conn_int*, is fixed. At the beginning of a connection event, a data channel is selected according to the adaptive frequency hopping (AFH) algorithm.[1] Then master and slave alternately exchange link-layer packets, which are separated by the mandatory Inter Frame Spacing (IFS) of length t_{IFS}.

The duration of a connection event t_c depends on the number and the size of the exchanged link-layer packets. Master and slave may transmit several packets or simply keep the connection alive by exchanging only one link-layer packet each that indicates that no more transmissions take place in the current connection event. When the data exchange is completed, both devices turn off their radios until the next connection event starts. Master and slave can send at most F bytes each during a connection event. If they reach this *connection capacity*, they turn off the radio and resume communication at the beginning of the next connection event.

Fig. 2 shows an example where a slave transmits six packets to the master. Four of them let the slave reach its connection capacity F in connection event C_0, so the remaining two are sent during connection event C_1. In every connection event, the master transmits the first packet. In the example of Fig. 2, the master sends packets with an empty payload as it has no data to transmit, while the slave sends packets with maximum payload. As a result, the length of a transmission by the master t_M is shorter than that of the slave t_S.

In connection event C_2 the slave has no more data to send, yet a connection event should contain at least one packet exchange. In our example, the master transmits in C_2 but the slave keeps silent. This situation is foreseen by the BLE standard: a slave *may* skip S_L consecutive connection events, where S_L is called *slave latency*. In

[1] BLE uses 40 channels in the unlicensed 2.4 GHz band. Three advertisement channels are reserved for unidirectional broadcast (connection-less mode); the remaining 37 data channels are only used for bidirectional unicast (connection-based mode).

our example, we have $S_L = 1$, so the slave may decide to interact with the master only on every second connection event. During connection event C_3 both nodes again exchange one packet with an empty payload and turn off their radios after t_K seconds.

BLE's connection-based mode automatically handles packet acknowledgments (ACKs) and link-layer flow control using a 1-bit sequence number and 1-bit ACK field in the frame header. In case of unreliable links, the *supervision timeout* S_T is used to detect the loss of BLE links, specifying the maximum time between two received packets before the connection is marked as lost. If S_T fires, the connection is canceled, and slave and master return to advertising and scanning mode, respectively. To ensure reliable communication, BLE uses the AFH algorithm, which selects only one of the enabled data channels in the *channel map* C_{map} provided as input. By changing C_{map}, a master can adaptively blacklist or whitelist data channels (*e.g.*, to evade interference from co-located networks).

3.2 Relevant Connection Parameters

Table 1 lists the most important BLE connection parameters. BLEach exposes all of them as tuning knobs (see Sec. 4), yet three deeply impact BLE's energy consumption and communication latency:

- A short connection interval *conn_int* increases throughput and shortens latency at the cost of higher energy consumption. A long *conn_int* has the opposite effect.
- A high slave latency S_L reduces energy consumption, but also reduces throughput and increases latency of master-to-slave data exchange as the slave is free to choose during which connection event to interact with the master.
- The higher the connection capacity F, the lower the energy consumption when transmitting large data packets as less time is spent for pre- and post-processing (t_{pre} and t_{post}).

Next, we formalize these observations by deriving an analytical model that expresses BLE's energy consumption and communication latency as functions of the three parameters. This model can be used to find parameter values that meet given performance goals.

3.3 Modeling the Impact of BLE Connection Parameters on Network Performance

Starting from an entire data transfer, we move on to individual connection events, and finally look at single link-layer transmissions.

Data transfer. We are interested in the time and energy needed to send D bytes from slave to master. We focus on the slave as the master is frequently recharged or wall-powered. We neglect packet loss, which keeps our expressions simple without sacrificing accuracy, as long as AFH finds a high-quality channel (see Sec. 6). Extending our models to account for packet loss is beyond the scope of this paper, but existing approaches can be used [46].

We start with latency. As data may arrive at any time with respect to scheduled connection events, the entire data transfer takes

$$t_{avg} = (n_F - 1/2) \cdot conn\_int + t_c \tag{1}$$

on average, where n_F is the number of connection events, each with connection capacity F, needed to send D bytes and given by

$$n_F = \lceil D/F \rceil \tag{2}$$

Table 1: BLE parameters exposed in BLEach as tuning knobs.

Parameter name	Possible values
Connection interval *conn_int*	[7.5–4000] ms in 1.25 ms steps
Slave latency S_L	[0–500] connection intervals
Connection capacity F	hardware-specific no. of bytes
Channel map C_{map}	bitmask with 37 entries

and t_c is the time needed to exchange the remaining data during the last connection event of the data transfer. In the worst case, the data arrives just after the start of a connection event, which leads to the following upper bound on the time needed to send D bytes

$$t_{max} = n_F \cdot conn\_int + t_F \tag{3}$$

Here we assume that the slave transmits the maximum number of bytes F also in the last connection event, which takes t_F seconds.

We now turn to the energy consumed over a time interval t_D while sending D bytes of data; t_D may be the sampling interval in an IoT application. As t_D can be longer than the time needed for the data transfer, we not only need to account for energy E_D spent on exchanging data, but also for energy E_M spent on keep-alive messages, for energy E_C spent during skipped connection events if the slave latency $S_L > 0$, and for energy E_I spent in idle mode. Thus, the total energy consumed is the sum of these components

$$E = E_D + E_M + E_C + E_I \tag{4}$$

The energy spent on exchanging actual data can be expressed as

$$E_D = \begin{cases} \frac{D}{F} \cdot E_F, & \text{if } D \bmod F = 0 \\ \lfloor \frac{D}{F} \rfloor \cdot E_F + (D \bmod F) \cdot \frac{E_F - E_K}{F} + E_K, & \text{otherwise} \end{cases} \tag{5}$$

where E_F is the energy for exchanging the maximum of F bytes during a connection event and E_K is the energy for exchanging a packet with zero payload, such as a keep-alive message. The energy for exchanging keep-alive messages is given by

$$E_M = n_K \cdot E_K = \left\lfloor \left(\left\lfloor \frac{t_D}{conn\_int} \right\rfloor - n_F \right) \frac{1}{1 + S_L} \right\rfloor \cdot E_K \tag{6}$$

where n_K is the number of connection events in which a keep-alive message is exchanged, which can only happen when no actual data transfer takes place. If $S_L > 0$, a slave may also skip connection events; however, it still wakes up to check if there are data to be transmitted. Knowing that one such check consumes E_S, we can express the energy for n_S skipped connection events as

$$E_C = n_S \cdot E_S = \left(\left\lfloor \frac{t_D}{conn\_int} \right\rfloor - n_F - n_K \right) \cdot E_S \tag{7}$$

During the remaining time, a slave is in idle mode consuming

$$E_I = [t_D - (n_F \cdot t_F + n_K \cdot t_K)] \cdot P_I \tag{8}$$

of energy, where t_F is the duration of a connection event wherein the slave sends the maximum of F bytes, t_K is the duration of a connection event wherein the slave sends a keep-alive message, and P_I is the power draw when the slave resides in idle mode.

Connection events. Fig. 3 shows power draw and packet transmissions of a slave during a BLE connection event, recorded using a mixed-signal oscilloscope on the TI CC2650. The slave sends multiple link-layer packets to the master for a total of 256 bytes.

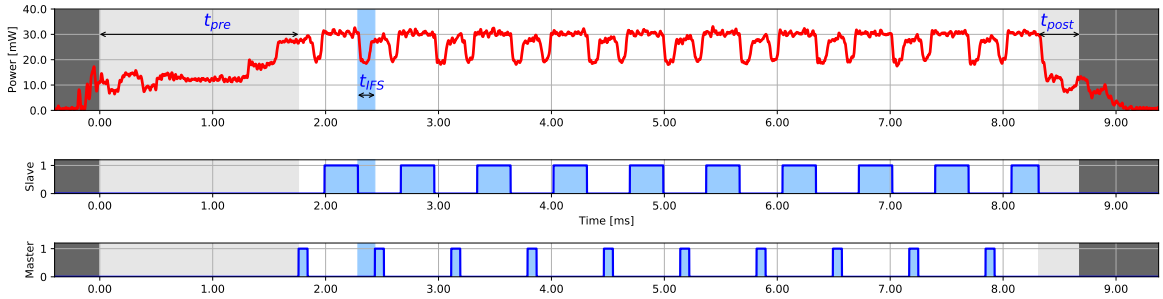


Figure 3: Power draw, packet transmissions, and gaps between packets during a BLE connection event, recorded using a mixed-signal oscilloscope on a TI CC2650 platform that acts as the slave and communicates with a master in connection-based mode.

Table 2: Power, time, and energy measured on the TI CC2650.

P_{pre}	16.024 ± 0.28 mW	t_{pre}	1.780 ± 0.001 ms
P_M	28.090 ± 0.27 mW	t_M	0.080 ± 0.000 ms
P_S	30.484 ± 0.44 mW	t_S	0.296 ± 0.000 ms
P_{IFS}	23.091 ± 0.30 mW	t_{IFS}	0.150 ± 0.000 ms
P_{post}	20.245 ± 0.48 mW	t_{post}	0.363 ± 0.001 ms
P_I	0.939 ± 0.02 mW	E_S	4.751 ± 0.126 μJ

After a pre-processing phase, whose duration t_{pre} is hardware specific, the master sends its first packet M_1. The slave replies with its first packet S_1 after the mandatory IFS of fixed duration t_{IFS}. The packet exchange continues until master and slave have no more data to send or they have reached their connection capacity F. A post-processing phase with hardware-specific duration t_{post} completes the connection event. Thus, the time spent by master and slave in a generic connection event with n packet exchanges is given by

$$t_c = t_{pre} + t_{post} + (2n-1)\cdot t_{IFS} + \sum_{i=1}^{n}\left(t_{M_i} + t_{S_i}\right) \tag{9}$$

where t_{M_i} and t_{S_i} represent the times needed by master and slave to transmit their packets $M_1, \ldots, M_n$ and $S_1, \ldots, S_n$, respectively. Using (9), we can obtain t_F by letting the number and size of these packets correspond to the connection capacity F, and t_K by letting M_1 and S_1 be (keep-alive) packets with zero payload.

Similarly, we obtain the energy consumed by a slave in a generic connection event by multiplying each individual time in (9) with the respective power draw as follows

$$E_c = E_{pre} + E_{post} + (2n-1)\cdot E_{IFS} + \sum_{i=1}^{n}\left(E_{M_i} + E_{S_i}\right) \tag{10}$$

Table 2 lists the individual times, power draws, and energy consumption to instantiate (9) and (10) for the TI CC2650. Using (10), we can also obtain E_F and E_K as described above for t_F and t_K. To calibrate our model for a different platform, we only need to measure the parameters listed in Table 2.

Link-layer transmissions. BLE's physical- and link-layer specification prescribes a common structure for all packets [5]. Each packet consists of a header and a payload. The header has a 1-byte preamble, a 4-byte access address, a 2-byte link-layer header, and a 3-byte CRC, which amounts to a link-layer header overhead of $O_{LL} = 10$ bytes. The payload size of a link-layer packet P_{LL} ranges between 0 and 27 bytes (or higher depending on the BLE version[2]). The transmit bitrate in BLE is $R_{LL} = 1\,\text{Mbit/s} = 125\,\text{kB/s}$[3]. Thus, the time needed to transmit a link-layer packet is

$$t_{LL} = (O_{LL} + P_{LL})/R_{LL} \tag{11}$$

The energy needed to transmit a link-layer packet is simply

$$E_{LL} = P_S \cdot t_{LL} \tag{12}$$

where P_S is the power draw in transmit mode and, for example, given in Table 2 for the TI CC2650 at 0 dBm transmit power.

4 BLEACH: DESIGN AND IMPLEMENTATION

We present BLEach, a modular open-source IPv6-over-BLE stack that exposes key features and parameters allowing to control the energy consumption and timeliness of communication in single-hop BLE networks in accordance with RFC 7668 [24]. BLEach's design is compatible with the architecture of Contiki [17], a widely used operating system for embedded IoT devices with IPv6 connectivity.

Next, we describe BLEach's modular design and highlight the main differences to Contiki's IPv6-over-IEEE 802.15.4 stack, discuss how we integrate BLEach into the Contiki architecture, and describe an implementation of BLEach for the popular TI CC2650 platform.

4.1 Design

Fig. 4 shows the architecture of BLEach. Since RFC 7668 does not foresee any changes to the network (*i.e.*, IPv6) and transport layers, only the lowest four layers are specifically designed to support BLE.

A key challenge in designing BLEach is the nature of BLE controllers. As described in Sec. 2, they temporally decouple radio processing from higher-layer protocols and applications by automatically handling packet (re-)transmissions and radio duty cycling. This leads to a number of fundamental differences compared with existing network stack architectures for IoT devices.

BLE link and PHY layer. The lowest layer in the BLEach stack is the BLE link and PHY layer, which implements all the services provided by a BLE controller and exposes to the upper layers an interface to create BLE connections, to append packets to the queue of outgoing packets, and to get notified about any incoming packets. It

[2]Up to 27 and 251 bytes of payload are supported by BLE v4.1 and v4.2, respectively.
[3]BLE v5.0, introduced in December 2016, offers an additional PHY mode that supports higher transmit bitrate, longer range, higher output power, and periodic advertising.

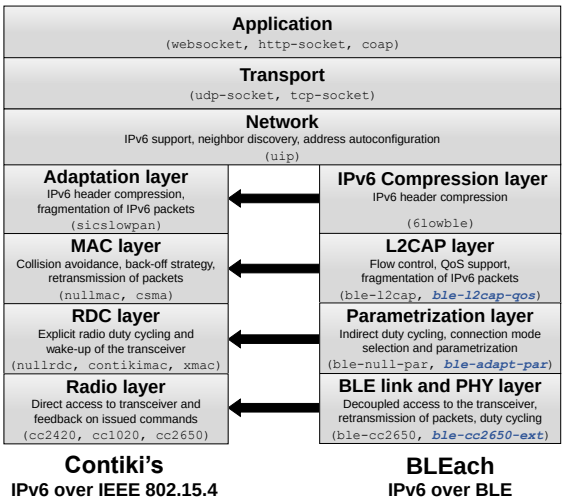


Figure 4: Architecture of BLEach and corresponding layers in Contiki's IPv6-over-IEEE 802.15.4 stack.

is the only hardware-specific layer within BLEach, and hence needs to accommodate both closed-source and open BLE controllers.

Both types of controllers implement services such as scheduling link-layer transmissions, managing data buffers, and notifying the upper layers upon packet reception. While closed-source controllers hide the implementation of these services and provide the standardized HCI to issue corresponding BLE commands, open controllers require the developer to implement the services, including the (re-)transmission of link-layer packets and the AFH algorithm, using the vendor-specific radio API. Thus, the implementation of the BLE link and PHY layer is more complex for open BLE controllers, but it also allows to accurately fine-tune and modify the functionality of the BLE radio. As an example, we describe in Sec. 5.3 the implementation of an indirect link-quality monitoring approach for the TI CC2650 BLE controller that blacklists interfered channels and increases the system's timeliness and energy efficiency.

Parametrization layer. On top of the BLE link and PHY layer sits the parametrization layer. It *selects* and *adapts* the connection mode used for communication and the parameters to be used by the BLE controller to schedule transmissions at run-time. In particular, by changing the connection parameters, this layer is also able to *indirectly* duty cycle the radio, which is a major difference to existing radio-duty-cycling techniques used in IEEE 802.15.4 stacks [16, 34].

The selection of connection parameters strongly affects system performance. The parametrization layer allows developers to use these parameters to influence the timeliness and energy efficiency of BLE. For example, rather than using the default parameters set by the master, a slave can compute an optimized parameter set according to device constraints and application needs using the model presented in Sec. 3.3 and negotiate it with the master. We present in Sec. 5.1 an implementation of an adaptive parametrization layer that dynamically changes the connection interval at run-time, showing that it can decrease the energy cost of a device by a factor of two.

Furthermore, this layer could be used to implement aggressive duty-cycling strategies that temporarily terminate a BLE connection and resume it at a negotiated point in time so as to significantly improve the performance of mostly-off sensing devices [9, 11]. The parametrization layer can also act as a building block to create IPv6-over-BLE mesh networks using connection-less BLE by scheduling the transmission of advertisements embedding IPv6 traffic and scanning for advertisements embedding a response.

L2CAP layer. According to RFC 7668 [24], the L2CAP layer has two main functions: fragmentation of IPv6 packets and prevention of buffer overflows by means of LE credit-based flow control. The fragmentation and reassembly mechanism of L2CAP makes it possible to exchange large IPv6 packets over constrained BLE links by fragmenting them into smaller chunks called L2CAP fragments whose size is upper-bounded by the connection capacity F.

L2CAP also creates logical channels between two peer devices and makes use of credits to control the flow of fragments and avoid buffer overflows. Specifically, both devices grant their peer a given amount of credits for communication. Each time a fragment is sent, the sender decreases its credit count by one. As soon as a device has no more credits left, it is no longer allowed to send fragments on that specific L2CAP channel. A device may grant its peer additional credits anytime using a separate L2CAP signaling channel.

The LE credit-based flow control mode can be used to provide QoS mechanisms that enhance the simple buffer overflow prevention defined by the standard. We present in Sec. 5.2 an implementation of an L2CAP layer allowing to *prioritize at run-time* either specific slaves or specific IPv6 traffic from a given node.

IPv6 compression layer. To improve the efficiency of IPv6 communications, RFC 7668 foresees the use of IPv6 header compression as specified by RFC 6282 [23]. This mechanism allows to compress the header of IPv6 packets sent within the same subnet from 40 down to 2 bytes. Since the L2CAP layer handles fragmentation, this layer is a lightweight version of the existing adaptation layer in Contiki, without its 802.15.4-specific fragmentation mechanism.

Upper layers. RFC 7668 does not foresee any specific change in IPv6 addressing, neighbor discovery, and packet format. Moreover, it supports any transport layer on top of IPv6. Hence, BLEach can reuse any IPv6 implementation for constrained devices such as Contiki's uIP, Sensinode's NSv6, and Arch Rock's ARv6, and supports TCP, UDP, or any other upper layer running on top.

4.2 Integration into Contiki

We integrate BLEach into Contiki by reusing its IPv6 and UDP support and by mapping each of the four lowest layers to an existing layer in Contiki's IPv6-over-IEEE 802.15.4 stack as shown in Fig. 4.

Same number of layers, different functionality. BLEach's lowest layer, the BLE link and PHY layer, directly maps into Contiki's radio layer, but with completely different functionality. Whilst Contiki's radio layer (and traditional layers tailored to IEEE 802.15.4) offers *direct radio access* and immediate feedback on issued radio commands, this is not the case in the BLE link and PHY layer, as the access to the radio is *decoupled* from the upper layers.

BLEach's parametrization layer maps into Contiki's radio duty cycling (RDC) layer. The key difference is that whilst existing RDC

layers in Contiki (*e.g.*, ContikiMAC [16] and X-MAC [10]) directly issue primitives switching on and off the radio module to control their duty cycle and provide more energy-efficient communication, BLEach's parametrization layer can only carry out an *indirect* form of duty cycling by means of connection parameter adaptation.

The L2CAP layer of BLEach directly maps into Contiki's medium access control (MAC) layer. Its responsibilities, however, are not taking care of collision avoidance, back-off strategies and retransmission of packets as in Contiki's IPv6-over-IEEE 802.15.4 stack (as these are already accomplished by the BLE link and PHY layer), but rather to provide fragmentation and flow control, as well as QoS support by means of traffic prioritization and multiplexing.

Finally, the IPv6 compression layer of BLEach is a subset of Contiki's adaptation layer. The latter also needs to fragment IPv6 packets, which is already accomplished by BLEach's L2CAP.

Easily portable. Because we keep the architecture of BLEach generic, porting it to an arbitrary BLE platform only consists of adapting the BLE link and PHY layer implementation—all other layers remain unchanged. Furthermore, as the size of the buffers employed for packet fragmentation and frame transmission or reception as well as the maximum number of simultaneous connections are configurable, developers can optimize the stack for the hardware platform at hand. This makes it possible to support a large range of BLE devices, from very constrained platforms such as the TI CC2650 with only 20 kB of memory to more powerful platforms like the Nordic Semiconductor nRF52 with 128 kB of memory. BLEach is also not limited to a specific BLE version and can be easily configured to use BLE v4.1, v4.2, and v5.0, depending on the version supported by the target hardware platform.

Application agnostic. Because it strictly adheres to Contiki's system architecture, BLEach is agnostic to the application running on top; developers may run the same application using IPv6 over IEEE 802.15.4 or IPv6 over BLE by simply changing the project's configuration file at compile time. In Sec. 6.4 we exploit the two radios on the TI CC2650 to run the same exemplary application on top of IPv6 over IEEE 802.15.4 and IPv6 over BLE.

Support for multi-radio operation. An interesting avenue for future work is adding support for concurrently using the multiple radios available on state-of-the-art platforms. Based on similar prior efforts [26], doing so requires changes that cross-cut the entire system stack in Contiki. We believe that combining BLEach and the existing IEEE 802.15.4 would be a good place to start since they follow the same architecture and provide compatible interfaces.

4.3 Implementation

We implement BLEach on the TI CC2650 platform, which features an ARM Cortex-M3 application core with 20 kB of memory and an ARM Cortex-M0 radio core providing either IEEE 802.15.4 or BLE communication. We describe next BLEach's basic modules for each layer shown in Fig. 4, starting from the BLE link and PHY layer.

`ble-cc2650`. Our implementation of the BLE link and PHY layer supports both slave and master mode according to the BLE specification v4.1 [5]. In slave mode, the device may only be connected to a single master at a time. In master mode, the device is able to maintain connections to multiple slaves, whose number depends on the selected connection capacity F. In our default implementation, we limit the connection capacity to 256 bytes in order to support at least 4 slaves, and allow to select a connection interval in the range from 20 ms to 4000 ms and a slave latency between 0 and 500.

The TI CC2650 features an open BLE controller; that is, its radio core uses shared memory and dedicated handshake hardware to interact with the application core [42]. The radio core expects commands that specify the beginning and the end of a BLE event, as well as which radio channel to use, and does not provide any autonomous scheduling of BLE advertising or connection events, BLE buffer management, or AFH mechanism. To support BLE connections, we implement the connection schedule at the application core using Contiki's `rtimer` in order to wake-up the radio core and issue the BLE command with the right parameters in time for the connection event to be properly scheduled. The application core wakes up 1.5 ms before the start of a connection event and performs the following tasks: (*i*) it uses BLE's AFH algorithm to select the data channel to be used during the connection, (*ii*) adds the data to be transmitted over the connection to the transmission queue of the radio core, (*iii*) enables and initializes the radio core, and finally (*iv*) issues the BLE command. Once the radio core completes the connection event, the application core disables the radio.

`ble-null-par`. Using the minimal configuration of BLEach, the parametrization layer makes use of the default parameters set by the master device and provides an interface for further extensions.

`ble-l2cap`. In the minimal configuration, the L2CAP layer supports IPv6-over-BLE transmissions with a maximum length of 1280 bytes split into 256 byte fragments (*i.e.*, a buffer size of $\theta = 5$ fragments is allocated for each connected device). After a link-layer connection between two devices is established, the master creates a single L2CAP channel to the slave. The created LE credit-based flow control channel is then used for any IPv6 packet sent, and master and slave grant credits to each other to prevent buffer overflows. The flow control mechanism we provide checks if the credits of a peer device fall below a threshold $\tau = 2$. If this is the case, $\gamma = 4$ additional credits are granted. As $\gamma \leq \theta$ this simple mechanism guarantees that two devices can communicate at least one fragment at any given time and that no buffer overflow occurs.

`6lowble` and `uip`. We use Contiki's `sicslowpan` to build BLEach's `6lowble` module by stripping away the IPv6 fragmentation functionality so as to provide only IPv6 header compression according to RFC 6282 [23]. We further use Contiki's `uip` [15] suite to provide BLEach with IPv6 communication, neighbor discovery, address autoconfiguration, and support for UDP and TCP traffic.

5 EXTENDING BLEACH

We present three modules that extend BLEach with novel BLE functionality: an adaptive duty cycling parametrization layer (Sec. 5.1), an L2CAP module supporting QoS by means of traffic prioritization and multiplexing (Sec. 5.2), and a BLE link and PHY layer module that additionally provides indirect link-quality monitoring (Sec. 5.3). These extensions are highlighted in Fig. 4 and evaluated in Sec. 6.3.

5.1 Adaptive Radio Duty Cycling

A slave with limited battery capacity may not be able to afford a BLE connection with a border router using a default set of connection

parameters chosen to sustain a high traffic load. We thus design a parametrization layer implementing an adaptive duty-cycling mechanism that adapts the connection interval at run-time to the current traffic load. Using this mechanism, a slave can negotiate with the border router a new set of connection parameters that better fits its application needs.

Furthermore, this adaptive parametrization layer also allows the slave to quickly adapt to sudden changes in the traffic load by temporarily increasing the connection interval if the traffic load decreases over time or by decreasing the connection interval in case more bandwidth is needed. Unlike existing adaptive duty-cycling approaches for IEEE 802.15.4 that tune the radio duty cycle on a per-node [30] or per-network [46] basis, our approach adapts the radio duty cycle for each individual BLE connection.

`ble-adapt-par`. The new parametrization layer extends the basic `ble-null-par` module to periodically monitor the BLE connection and calculates the weighted moving average of the BLE connection *utilization*, that is, the percentage of available connection events actually used to transmit data. If the average utilization drops below a threshold $U_{low} = 25\,\%$, the adaptation mechanism negotiates a longer connection interval to increase the energy efficiency. Instead, if the average utilization increases above a threshold $U_{up} = 75\%$, the adaptation mechanism negotiates a shorter connection interval so that the slave is able to sustain a higher data rate.

To negotiate new connection parameters, the slave sends a BLE *connection parameter request* to the master. The master responds with a BLE *connection update request* that either confirms or declines the set of proposed connection parameters. BLE specifies that the new connection parameters take effect six connection events after the connection update request was sent. This inevitable delay increases the reaction time of our adaptation mechanism.

5.2 Traffic Prioritization and Multiplexing

L2CAP's LE credit-based flow control mode can be used to provide novel QoS mechanisms that enhance the simple buffer overflow prevention defined by the standard. We establish multiple L2CAP LE credit-based flow control channels between peer devices, each one with its own fragmentation buffer and credit count, and transport a different type of IPv6 traffic on each channel.

By granting a different amount of credits to each channel, a device can prioritize a *specific type of traffic* over another one. Furthermore, a master can *prioritize different nodes* in the network by granting fewer credits to slaves with low priority and more credits to slaves with high priority.

`ble-l2cap-qos`. We implement this simple principle by extending the basic `ble-l2cap` module as follows. First, we use one L2CAP channel for every IPv6 traffic class supported, multiplexing IPv6 traffic over a single BLE connection. Second, we adapt the fragmentation of the L2CAP layer such that it prioritizes the transmissions of the channel with the highest credit count. We further allow IPv6-over-BLE devices to change the priority of incoming traffic classes dynamically using the standardized L2CAP LE flow control credit message. Although multiple L2CAP channels for IPv6 are not foreseen by RFC 7668, we do not violate any specification and only use standardized primitives to implement our approach.

5.3 Indirect Link-quality Monitoring

Most BLE radios implement an AFH algorithm that blacklists data channels with insufficient link quality. Unfortunately, several radios are black boxes and a developer neither knows which metric is used to estimate the link quality of a data channel nor under which conditions a channel is blacklisted. We extend BLEach with a BLE link and PHY layer that implements indirect link-quality monitoring and adapts the list of blacklisted channels at run-time.

`ble-cc2650-ext`. We extend the basic `ble-cc2650` module by measuring the link-quality of data channels without the need to actively sense surrounding interference by means of RSSI scanning. In particular, if the status of a completed connection event is `BLE_DONE_NO_SYNC`, we increase the number of connection errors n_{err} of the current data channel. This status is returned if a BLE handshake between master and slave could not be performed. If n_{err} exceeds a *blacklisting threshold* β for a data channel, the master blacklists this channel and updates the channel map by sending a BLE *channel map update*. The master may whitelist data channels after a predefined time to not run out of active data channels.

6 EVALUATION

Our evaluation quantitatively answers the following questions:

- Are our analytical models accurate so they can be used to find optimized BLE connection parameters? (Sec. 6.1)
- Is BLEach interoperable and efficient with regard to memory, processing, and energy constraints? (Sec. 6.2)
- How effective are the three exemplary modules we design in making BLEach adaptive to changes in traffic load, traffic priorities, and wireless interference? (Sec. 6.3)
- Does BLEach achieve performance gains compared with an IPv6-over-IEEE 802.15.4 stack? (Sec. 6.4)

6.1 Model Validation

We begin by validating our analytical models from Sec. 3.3.

Setup. We run BLEach on two TI CC2650, one acting as master and the other as slave. We let the slave exchange $D = 512$ bytes with the master for varying data generation intervals t_D. To measure latency and energy consumption for different connection parameters, we connect both devices to a Keysight MSO-S 254A mixed-signal oscilloscope. We calibrate our analytical models with the hardware-dependent parameters listed in Table 2, and compare their output against our measurements for the same connection parameters.

Results. Figs. 5 and 6 plot energy and latency against connection interval `conn_int` for two different slave latencies S_L and varying t_D; the connection capacity F is fixed at 256 bytes. We see that our models are highly accurate. The predicted energy matches the measured energy and the theoretical upper bound on latency is always slightly above the measured latency across all settings.

We can make further observations important for parameter tuning. For example, looking at Fig. 5, we see that changing the connection interval `conn_int` from 20 to 100 ms or the slave latency S_L from 0 to 10 decreases the energy consumption by a factor of 2, regardless of the application interval t_D. Connection intervals longer than 100 ms result in only marginal energy reductions but in a linear increase in latency. This is because in this operating regime

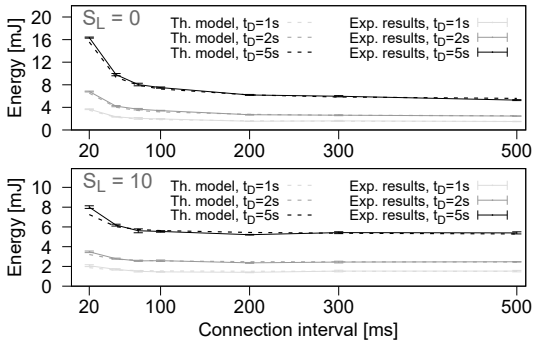


Figure 5: Impact of connection interval and slave latency on energy consumption for 256 bytes connection capacity.

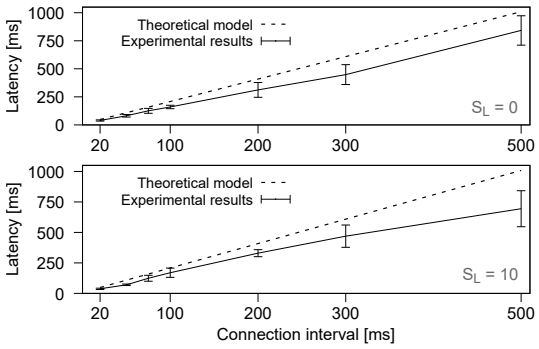


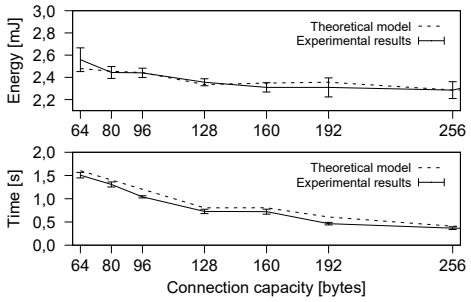
Figure 6: Impact of connection interval and slave latency on communication latency for 256 bytes connection capacity.

the idle energy E_I dominates. Thus, a connection interval around 100 ms gives a good trade-off between energy and latency in this setting. It is also worth noting that the slave latency S_L, which has a significant impact on energy, has no impact on latency: a slave only skips connection events in the absence of data to be transmitted.

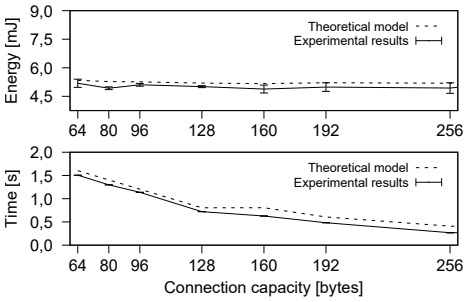
Fig. 7 plots energy and latency against connection capacity F for two different t_D; connection interval and slave latency are fixed to 200 ms and 10, respectively. Again, we see that our models accurately predict performance. We also see that the connection capacity F affects energy only minimally, but has a strong impact on latency. Indeed, we learn from Fig. 7 that by increasing F from 64 to 256, latency drops by a factor of 3. The impact of F on energy becomes more visible at frequent traffic. For $t_D = 1$ s, energy drops by 8% when increasing F from 64 to 256. These findings are important insights when tuning BLE connection parameters to optimize performance and to meet given application requirements.

6.2 Evaluating Minimal BLEach

Next, we run experiments using BLEach's minimal configuration (`ble-cc2650` + `ble-null-par` + `ble-l2cap`), evaluating interoperability, memory footprint, processing overhead, and energy cost.



(a) t_D = 2000 ms



(b) t_D = 5000 ms

Figure 7: Impact of connection capacity on energy consumption and latency for a connection interval of 200 ms and a slave latency of 10 for two different application intervals t_D.

Table 3: Interoperability of BLEach.

Master device	Interoperable?	Energy cost slave
TI CC2650	✓	103.796 ± 0.659 mJ
Raspberry Pi 3	✓	103.821 ± 0.895 mJ
LogiLink BZ0015	✓	104.597 ± 0.791 mJ

6.2.1 Interoperability. Nodes running BLEach are interoperable with any border router compliant with RFC 7668. To demonstrate this, we deploy BLEach on a TI CC2650 device acting as slave and let it interact with three different devices acting as master: (*i*) a LogiLink BZ0015 BLE-USB dongle attached to a Raspberry Pi 1 Model B, (*ii*) a Raspberry Pi 3 with an embedded Cypress Semiconductor BCM43438 BLE radio, and (iii) a TI CC2650. Both Pis run the Raspbian OS with the BlueZ stack [7] and the 6LoWPAN driver; the latter supports a maximum fragmentation size of 128 bytes. The TI CC2650 acting as a master runs BLEach in the border router configuration, using a fragmentation size of 128 bytes for a fair comparison with the two Pis. We instruct the slave to transmit a 256-byte IPv6 packet to the master every second. The master is supposed to always reply with an IPv6 packet of the same length, and to instruct the slave to use `conn_int` = 125 ms and S_L = 0. We measure the energy consumption of the slave using the oscilloscope. We verify in different runs that the slave successfully exchanges IPv6 packets with all three masters. Table 3 shows that the slave consumes the same energy regardless of the master it talks to.

Table 4: Memory footprint of BLEach when supporting a maximum IPv6 packet length of D = 512 bytes.

Device	RAM usage [kB]	ROM usage [kB]
Slave	3.318	10.941
Master (1 slave)	3.318	12.938
Master (2 slaves)	5.771	13.023
Master (4 slaves)	10.678	13.023

6.2.2 Memory footprint. We quantify the memory footprint of BLEach on slave and master devices in terms of RAM and ROM usage. Table 4 shows BLEach's footprint when supporting a maximum IPv6 length of D = 512 bytes. In this configuration, the master can support up to 4 slaves simultaneously; 8 slaves would be possible with D = 200 bytes. The RAM usage of a master with 1 supported slave is the same as the RAM usage of a slave; the ROM usage is slightly higher. A footprint of 3.3 kB in RAM and 10.9 kB in ROM is a good compromise for IoT devices, yet the former can be further reduced by configuring BLEach with a smaller D. For instance, with D = 64 bytes, the slave requires only 1.53 kB of RAM. Due to the memory efficiency of BLE, BLEach is very lightweight and well-suited for resource-constrained embedded IoT devices.

6.2.3 Processing overhead. To evaluate the processing overhead of BLEach, we measure the duration of UDP transmissions with different payload sizes from slave to master using the oscilloscope and break down the time spent in each layer of the stack (`conn_int` = 50 ms and S_L = 0). The x-axis in Fig. 8 shows the IPv6 packet length, including IPv6 header, UDP header, and UDP payload.

Fig. 8 shows that the largest fraction of time is spent in the BLE controller performing the actual data transmission. Indeed, we notice that the higher the connection capacity, the lower the absolute time spent by the BLE controller to complete the transfer. Compared to this fraction of time, the processing time of the remaining layers accounts for only 1.5–4.7 %. It is important to highlight that the operation of the BLE controller is in most cases hidden from the developer and that this overhead is not introduced by BLEach.

Fig. 8 also shows that the L2CAP layer performing fragmentation accounts for the largest processing overhead among the upper layers in BLEach, especially when the connection capacity F is small. When F is much smaller than the UDP payload, the L2CAP layer needs to process many small fragments and its efficiency decreases. This means that employing larger fragments is a better choice. Even more so, as the processing time of the BLE controller is two orders of magnitude larger than that of the L2CAP layer, it is more efficient to accumulate data at a BLE node and send it in bigger chunks. The overhead of the network and transport layers, instead, varies only minimally as a function of the payload length.

6.2.4 Energy consumption. We also evaluate the energy consumption of a TI CC2650 master running BLEach as a function of the number of connected slaves, and compare it with the energy expenditure of a TI CC2650 slave device using BLEach. We employ the aforementioned setup and let each slave in the network periodically send IPv6 packets with a length of 256 bytes to the master using a connection interval `conn_int` = 125 ms, a slave latency S_L = 0, and F = 256 bytes. We then use the oscilloscope to measure the energy consumption of each individual device.

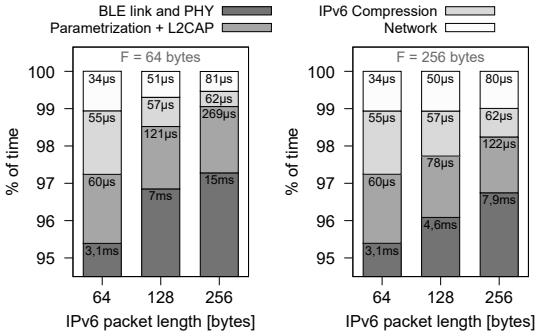


Figure 8: Breakdown of BLEach's processing time per layer when serving IPv6 transmissions of varying packet length.

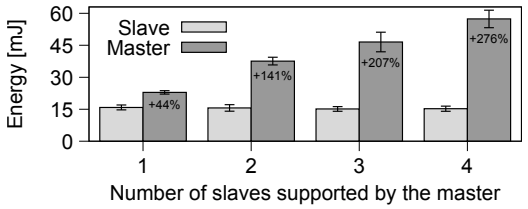


Figure 9: Energy consumption of slave and master running BLEach against the number of supported slaves in a subnet.

Fig. 9 shows that the energy consumption of the master is slightly higher than the one of the slave when having only one slave connected, and that it increases proportionally to the number of connected slaves. Instead, a slave does not exhibit any increase in energy consumption as more slaves connect to the master.

6.3 Evaluating Extended BLEach

We evaluate next the three extension modules we designed for BLEach—adaptive duty cycling, IPv6-over-BLE traffic prioritization and multiplexing, and indirect link-quality monitoring—and show that they help in increasing the network performance and resilience.

6.3.1 Adaptive duty cycling. We first evaluate the ability of the `ble-adapt-par` module described in Sec. 5.1 to adapt the connection interval at run-time to unforeseen changes in traffic load. To this end, we let two slave devices communicate to a master. The first slave runs BLEach's `ble-rdc` RDC layer and uses fixed connection parameters (`conn_int` = 62.5 ms and S_L = 0), which correspond to the default settings in Linux' BlueZ stack. The second slave runs `ble-adapt-par`, adapting the connection interval at runtime. We vary the traffic load of the slaves over time. Each slave initially schedules the transmission of a 256-byte packet every second. The number of scheduled transmissions is then halved, doubled, quadrupled, and finally reduced to a quarter in consecutive phases. We measure power draw and packet delivery rate of the two slaves.

Fig. 10 (top) plots the scheduled transmissions over time, which result in different packet rates. Below we plot the connection intervals selected by the two slaves. The two charts at the bottom of Fig. 10 plot packet delivery rate and power draw of both slaves.

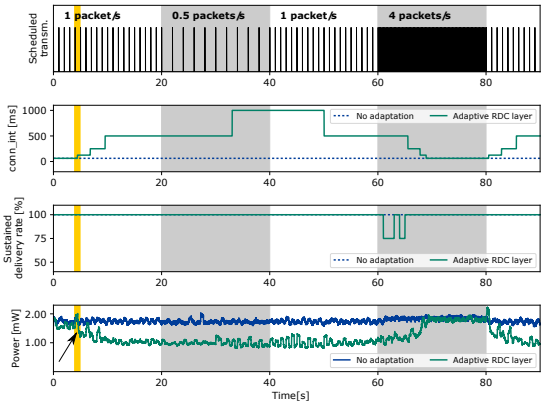


Figure 10: Performance of BLEach with and without adaptive duty cycling as the traffic load changes over time.

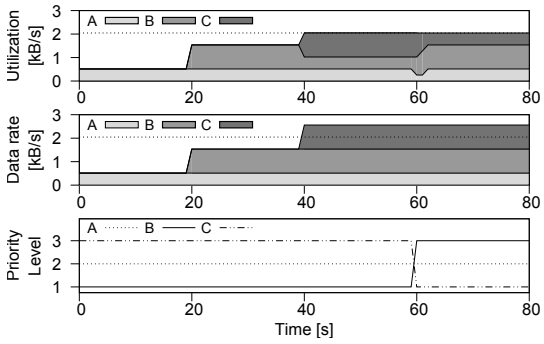


Figure 11: IPv6-over-BLE traffic multiplexing and prioritization in action, as provided by `ble-l2cap-qos` in BLEach.

First, we observe that our adaptive duty cycling module achieves a significant reduction in power draw of up to 50 % compared with the static setting. Indeed, as visible in Fig. 10, the latter is optimized for a very high traffic load, which causes excess energy consumption during all other phases with lighter traffic load.

Second, we see a slight drop in packet delivery rate at the beginning of the phase with the highest traffic load. This is because our current implementation of `ble-adapt-par` is compliant with the BLE standard, which specifies that a parameter change takes effect exactly after *six* connection events [5]. Thus, if the current connection interval is 500 ms, which is the case right before the high-load phase starts, it takes at least 3 seconds until the first parameter update occurs. Nevertheless, using BLEach, we would be able to easily overrule these specifications on open BLE controllers, such as the TI CC2650, which is not foreseen by existing BLE stacks.

Third, looking at the arrow in Fig. 10, we can see that the few additional packets sent by the slave to the master to inform about its preferred parameters incur a negligible cost. This cost is in fact a very good investment given the energy savings it enables.

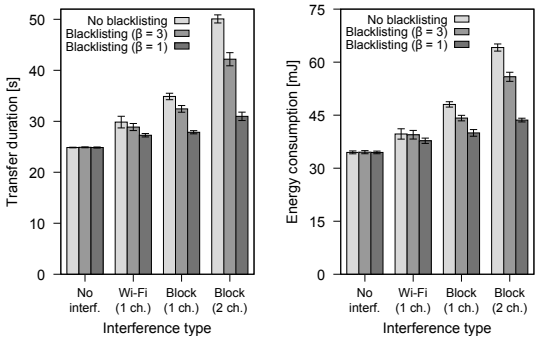


Figure 12: Benefit of indirect link-quality monitoring and channel blacklisting in the presence of radio interference.

6.3.2 QoS support. BLEach's `ble-l2cap-qos` module described in Sec. 5.2 allows to prioritize and multiplex IPv6-over-BLE traffic. We now evaluate its performance by running an exemplary BLE system consisting of a master M and a slave S employing three types of IPv6-over-BLE traffic A, B, and C. Traffic A transports sensor data collected by S, traffic C embeds actuation commands issued by S to other nodes in the network in response to specific sensed events, whilst traffic B is maintenance ICMPv6 traffic. Each traffic class is assigned its own L2CAP channel and a priority adjustable at run-time by the master using L2CAP's *LE credit-based flow control* mechanism. We study how the operations of BLEach's `ble-l2cap-qos` module affect the performance of the overall system. In our experiments, we select `conn_int` = 125ms and F = 256 bytes, *i.e.*, the BLE link layer can send at a maximum data rate of 2 kB/s using eight connections per second carrying 256 bytes each.

Fig. 11 (middle) shows the data rate of the three traffic sources. Starting from time 0, the slave S transmits periodic ICMPv6 traffic A at 512 bytes/s. At time 20, S generates 1024 bytes/s traffic of type B, signaling to the master M the occurrence of a specific event. At time 40, S reacts on the detected event by issuing actuation commands for other nodes in the network and hence traffic of type C at a data rate of 1024 bytes/s. As the sum of the three traffic flows exceeds the maximum utilization of the BLE link between S and M, the higher-priority traffic C gets scheduled first with the result that only a portion of the lower-priority traffic B is served.

At time 60, the master M is interested in verifying that the actuation commands previously sent through the network have achieved the desired effect, and dynamically switches the priority of traffic B and traffic C, thus prioritizing sensed data. Fig. 11 shows that, as a result, as soon as the number of credits left for traffic C are used up, traffic A and B gets scheduled first, thereby prioritizing sensed data as instructed by the master. Our exemplary implementation of the `ble-l2cap-qos` module only *assigns* credits; dynamically *reducing* credits at run-time is an interesting direction for future work.

6.3.3 Indirect link-quality monitoring. We now look at BLEach's `ble-cc2650-ext` module described in Sec. 5.3, assessing the benefits provided by indirect link-quality monitoring in the face of wireless interference. To this end, we generate controlled interference using JamLab [8]. The interference resembles either Wi-Fi

Table 5: Processing time per layer (in milliseconds) for the IPv6-over-IEEE 802.15.4 and BLEach stacks for varying payload size.

	IPv6 over IEEE 802.15.4				IPv6 over BLE (BLEach)			
Payload	**Radio**	**RDC**	**MAC**	**Upper Layers**	**BLE L. & P.**	**Parametr.**	**L2CAP**	**Upper Layers**
128 bytes	9.10 ± 2.23	17.35 ± 0.21	0.025 ± 0.001	0.120 ± 0.001	4.53 ± 0.02	0.022 ± 0.001	0.057 ± 0.001	0.107 ± 0.001
256 bytes	16.48 ± 2.45	23.32 ± 0.27	0.054 ± 0.001	0.254 ± 0.001	7.84 ± 0.02	0.041 ± 0.001	0.081 ± 0.001	0.142 ± 0.001
512 bytes	27.33 ± 2.49	29.22 ± 0.35	0.090 ± 0.002	0.406 ± 0.001	16.47 ± 0.04	0.084 ± 0.004	0.201 ± 0.003	0.213 ± 0.001

data exchange on a single Wi-Fi channel (*Wi-Fi 1 ch.*), or one or two extremely congested Wi-Fi channels due to improper or malicious activity in the surrounding (*Block (1 ch.)* and *Block (2 ch.)*).

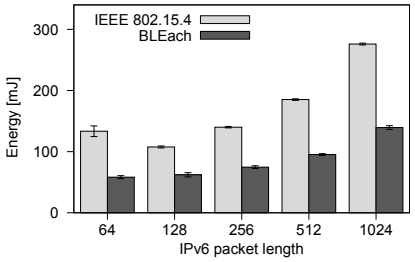
We use an application that exchanges 50 consecutive IPv6 packets with a length of 256 bytes between a master and a slave using `conn_int` = 250 ms and $S_L = 0$. Each packet needs to be acknowledged before the next one can be transmitted. We run BLEach using `ble-cc2650-ext` and show that the ability to monitor the connection event status allows to quickly influence the `channel_map` used by the BLE controller to select the channel for sending packets.

Fig. 12 shows that this significantly reduces the duration of the data exchange between master and slave as well as the energy consumption. The plot also highlights how aggressively blacklisting a channel after only one failed connection event (β=1) can actually minimize both latency and energy costs. Finally, Fig. 12 also shows that, in absence of interference, no additional energy is consumed by `ble-cc2650-ext`. Indeed, the implemented strategy does not make use of passive RSSI scanning, which would cause the radio to remain active for an additional period of time as in [33].

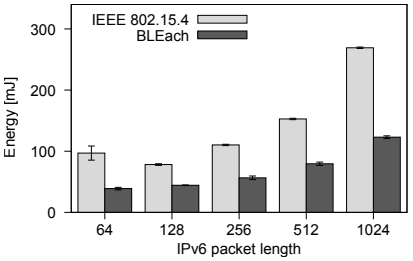
6.4 Comparison with IPv6 over IEEE 802.15.4

The TI CC2650 comes with a BLE and an IEEE 802.15.4 radio, allowing us to compare the performance of BLEach against Contiki's IPv6-over-IEEE 802.15.4 stack on the same platform. Because BLEach fully adheres to Contiki's system and stack architecture, we can run the same application without any changes on top of both stacks. We enable ContikiMAC's *phase lock* optimization and configure a wake-up interval *wakeup_int* of 62.5 and 125 ms (a channel check rate of 16 and 8 respectively). We compare its performance with our BLEach stack configured with a connection interval *conn_int* of 62.5 and 125 ms, respectively, to ensure a fair comparison. Both radios transmit packets with a transmission power of 0 dBm. ContikiMAC is configured with a guard time of 22.9 ms and a maximum phase strobe time of 30.52 ms (1500 and 2000 `rtimer` ticks, respectively) and the maximum number of frame retries of CSMA is set to 1. We employ an application that triggers a series of 60 request-response interactions between a client and a server. Every second, the client (BLE slave) sends one UDP packet of variable size to the server (BLE master), who replies with an UDP packet of the same length.

Energy consumption. Fig. 13 plots the energy consumption of the client (BLE slave) measured with the oscilloscope for different IPv6 packet sizes. The x-axis in Fig. 13 shows the overall length of the exchanged IPv6 packet, including IPv6 header, UDP header, and UDP payload. BLEach consumes on average approximately 50% less energy than the IPv6-over-IEEE 802.15.4 stack, regardless of the selected wake-up or connection interval, as well as packet length. This energy saving can be explained by analyzing the different processing times of the two stacks.



(a) Wake-up/connection interval of 62.5 ms



(b) Wake-up/connection interval of 125 ms

Figure 13: Average power draw of BLEach and the IPv6-over-IEEE 802.15.4. stack of Contiki on the TI CC2650.

Processing time. Table 5 shows the processing time of each layer in the two stacks for different payload length and a `wakeup_int` or `conn_int` of 125 ms. The IPv6-over-IEEE 802.15.4 stack exhibits a higher overhead compared to BLEach due to the large CPU time spent in the RDC layer scheduling ContikiMAC's strobe transmissions and clear channel assessments, as well as due to the higher processing time in the radio layer. There are three reasons for the higher radio time when using IEEE 802.15.4. First, IEEE 802.15.4 transmits at 250 kbit/s, which is four times lower than the data rate of BLE (1 Mbit/s). Hence, when transmitting the same number of bytes, IEEE 802.15.4 has a longer transmission time compared to BLE. Second, the maximum payload length of IEEE 802.15.4 is limited to 125 bytes. Packets exceeding this maximum payload length are fragmented into smaller chunks and transmitted sequentially. Compared to IEEE 802.15.4, the connection capacity F of BLE is not limited by the BLE specification and is configured to be $F = 256$ bytes in BLEach. Therefore, IPv6 over IEEE 802.15.4 needs to transmit more fragments when sending long IPv6 packets than BLEach, introducing link-layer overhead with every additional fragment. Third, ContikiMAC with phase lock enabled repeatedly sends

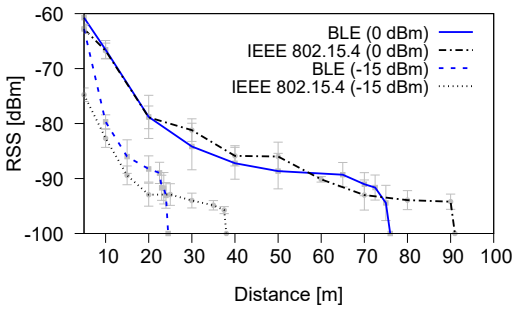


Figure 14: Received signal strength measured with the TI CC2650 using the BLE and IEEE 802.15.4 radio modes.

the first fragment of every packet exchange until it is acknowledged, contributing to the higher radio time shown in Table 5.

Communication range. Although the previous experiments show that BLE is more energy efficient than IEEE 802.15.4 when using the same platform (confirming the results of [13, 40]), it is important to set this into perspective with the achievable communication range of both wireless technologies. We experimentally observe that IEEE 802.15.4 supports a larger range by comparing the communication range of two TI CC2650 devices deployed outdoor with direct line-of-sight that exchange IPv6-over-IEEE 802.15.4 and IPv6-over-BLE packets. Fig. 14 shows the evolution of the received signal strength (RSS) as a function of the distance between the two nodes for two different transmission power levels. The figure also shows the distance at which no more packets were received, highlighted by the dots at -100 dBm, where both BLE and IEEE 802.15.4 fail to exchange messages. When sending packets at a transmission power of 0 dBm, the maximum achievable communication range is 75 and 90 meters for BLE and IEEE 802.15.4, respectively, whilst it is 24 and 38 meters when using a transmission power of -15 dBm. This allows us to conclude that, when using the TI CC2650 in BLE mode, the communication range is indeed shorter than the one that can be reached using IEEE 802.15.4.

7 RELATED WORK

The lack of open-source BLE stacks has significantly hindered BLE networking research [41]. Instead, the community has focused on designing battery-driven [41] or energy-harvesting [12] BLE platforms and on exploiting BLE's connection-less mode for other services, such as neighbor discovery [25], indoor localization [18], group management [20], and locality-based authorization [19]. Below we review related work on BLE stacks for embedded IoT devices, run-time adaptability, and relevant BLE networking projects.

BLE stacks. There exists a number of proprietary BLE stacks *lacking* IPv6 support, for example, from TI [43] and Mindtree [32]. Open-source BLE support in TinyOS and Contiki is completely missing or limited. Contiki only features transmissions of advertisement packets without IPv6 for the TI CC2650 radio, and a closed-source BLE radio and L2CAP slave implementation for the nRF52 that does not support fragmentation of IPv6 packets. Android uses the BlueDroid stack, which supports both classic Bluetooth and BLE but not IPv6 over BLE [1]. Apache MyNewt [2] and Zephyr [44] come with stacks implementing full-fledged BLE connections. However, Apache's NimBLE stack does not support IPv6 and is memory-hungry (4.5 kB of RAM, 69 kB of flash), and Zephyr's stack supports IPv6 over BLE only on slave devices, and cannot fragment large IPv6 packets, which makes it unsuitable for constrained IoT devices.

In contrast to these stacks, BLEach is open-source, streamlined for easy integration into Contiki, supports IPv6 on master and slave devices, and lightweight to be used on constrained IoT devices. In addition, it provides an API to tune key BLE parameters at run-time.

BLE measurements. Other works have studied the energy efficiency and timeliness of BLE. Gomez et al. [21] have reported the energy consumption and packet latency of Attribute Protocol communications with a maximum link-layer packet length of 37 bytes. Dementyev et al. [13] have measured the energy consumption of BLE slaves that periodically send 8-byte data packets over a BLE connection. Likewise, Siekkinen et al. [40] have studied the energy consumption of connection-less and connection-based BLE for a maximum link-layer payload size of 27 bytes. Both studies conclude that the BLE link-layer has a higher energy efficiency than IEEE 802.15.4 in their experimental setup.

Our experiments confirm their results and further provide a detailed comparison of IPv6 over BLE and IPv6 over IEEE 802.15.4 on the same hardware platform for a wide range of IPv6 packet lengths, as well as an analysis of the processing time introduced by each layer of the communication stack.

BLE runtime adaptability. Gomez et al. show that connection interval and slave latency impact BLE performance, suggesting that these parameters could be tuned to meet given application requirements [21]. Similarly, Lee et al. report on experiments showing that the connection interval affects the packet delivery rate [29]. Kindt et al. adapt the connection interval to traffic load for energy efficiency [27]. However, their approach lacks practicality, since they assume that the adaptation logic runs on the master, whose firmware is often not easily accessible (*e.g.*, an IPv6 gateway or smart-phone). Mikhaylov adjusts the connection interval to reduce the time and energy needed for BLE connection establishment [31].

Unlike these works, we accurately model the impact of all key parameters affecting connection-based communication performance (*i.e.*, including slave latency and connection capacity), and expose them to slaves through a standard-compliant negotiation-based interface. Moreover, to the best of our knowledge, we are the first to consider adaptive L2CAP functionality, providing QoS guarantees by means of dynamic traffic multiplexing and traffic prioritization.

Other relevant BLE networking research. A few works aim to unleash BLE from rigid single-hop networking. For instance, Roest presents a BLE port of the all-to-all Chaos primitive [28], demonstrating performance gains when nodes use all 40 BLE channels in a randomized fashion [39]. Lee et al. exploit a powerful embedded Linux PC to run RPL over a tree-based multi-hop topology [29], showing performance benefits of RPL over BLE compared to RPL over IEEE 802.15.4. Hussain et al. enable mobility through seamless BLE connection migration between gateways [22].

We believe that the work presented in this paper and its open-source availability can empower more of such novel BLE networking research in the years to come.

8 CONCLUSIONS

BLEach is the first open-source stack with full-fledged support for IPv6 over BLE. It is modular, interoperable, efficient, and can be ported to a variety of BLE platforms with minimal effort. BLEach exposes several key parameters to fine-tune communication performance at runtime, using our accurate latency and energy models. Three novel modules we design make BLEach adaptive to traffic fluctuations and wireless interference, while providing QoS guarantees through on-demand traffic prioritization and multiplexing.

ACKNOWLEDGMENTS

We thank our Shepherd, Brano Kusy, and all the anonymous reviewers for their constructive comments. 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 DFG within cfaed and the SCOTT project. SCOTT (http://www.scott-project.eu) has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 737422. This joint undertaking receives support from the European Unions Horizon 2020 research and innovation programme and Austria, Spain, Finland, Ireland, Sweden, Germany, Poland, Portugal, Netherlands, Belgium, Norway. SCOTT is also funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between May 2017 and April 2020. More information at https://iktderzukunft.at/en/.

REFERENCES

[1] Android. 2017. Bluetooth. https://source.android.com/devices/bluetooth. (2017).
[2] Apache MyNewt. 2017. NimBLE Introduction. http://mynewt.apache.org/network/ble/ble_intro/. (2017).
[3] B. Carpenter et al. 2014. RFC 6775 - Significance of IPv6 Interface Identifiers. https://tools.ietf.org/html/rfc7136. (2014).
[4] BLE Home. 2017. iAlert Sensing Motion: Quick Start Guide. http://www.blehome.com/ialert.htm. (2017).
[5] Bluetooth SIG. 2013. Specification of the Bluetooth System – Covered Core Package version: 4.1. https://www.bluetooth.org/en-us/specification/adopted-specifications. (2013).
[6] Bluetooth SIG. 2014. Internet Protocol Support Profile - Bluetooth Specification version: 1.0.0. https://www.bluetooth.org/en-us/specification/adopted-specifications. (2014).
[7] BlueZ Project. 2016. BlueZ - Official Linux Bluetooth protocol stack. http://www.bluez.org/. (2016).
[8] C.A. Boano, T. Voigt, C. Noda, K. Römer, and M.A. Zúñiga. 2011. JamLab: Augmenting Sensornet Testbeds with Realistic and Controlled Interference Generation. In *Proc. of the* 10^{th} *ACM/IEEE IPSN Conference.*
[9] W. Bober and C.J. Bleakley. 2014. BailighPulse: A Low Duty Cycle Data Gathering Protocol for Mostly-off Wireless Sensor Networks. *Computer Networks* 69 (2014).
[10] M. Buettner, G.V. Yee, E. Anderson, and R. Han. 2006. X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In *Proc. of the* 4^{th} *ACM SenSys Conference.*
[11] N. Burri, P. von Rickenbach, and R. Wattenhofer. 2007. Dozer: Ultra-low Power Data Gathering in Sensor Networks. In *Proceedings of the* 6^{th} *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN).*
[12] B. Campbell, J. Adkins, and P. Dutta. 2016. Cinamin: A Perpetual and Nearly Invisible BLE Beacon. In *Proc. of the* 1^{st} *NextMote Workshop.*
[13] A. Dementyev, S. Hodges, S. Taylor, and J. Smith. 2013. Power Consumption Analysis of Bluetooth Low Energy, ZigBee and ANT Sensor Nodes in a Cyclic Sleep Scenario. In *Proc. of the* 1^{st} *IEEE IWS Symposium.*
[14] K.M. Diaz, D.J. Krupka, M.J. Chang, J. Peacock, Y. Ma, J. Goldsmith, J.E. Schwartz, and K.W. Davidson. 2015. Fitbit: An Accurate and Reliable Device for Wireless Physical Activity Tracking. *International Journal of Cardiology* 185 (2015).
[15] A. Dunkels. 2002. *uIP-A free small TCP/IP stack.* Technical Report.
[16] A. Dunkels. 2011. *The ContikiMAC Radio Duty Cycling Protocol.* Technical Report T2011:13. Swedish Institute of Computer Science.
[17] A. Dunkels, B. Grönvall, and T. Voigt. 2004. Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proc. of the* 1^{st} *EmNetS Workshop.*
[18] R. Faragher and R. Harle. 2015. Location Fingerprinting With Bluetooth Low Energy Beacons. *IEEE Journal on Selected Areas in Communications* 33, 11 (2015), 2418–2428.
[19] J. Fürst, K. Chen, M. Aljarrah, and P. Bonnet. 2016. Leveraging Physical Locality to Integrate Smart Appliances in Non-Residential Buildings with Ultrasound and Bluetooth Low Energy. In *Proc. of the* 1^{st} *IEEE IoTDI Conference.*
[20] D. Giovanelli, B. Milosevic, C. Kiraly, A.L. Murphy, and E. Farella. 2016. Dynamic group management with Bluetooth Low Energy. In *Proc. of the* 2^{nd} *IEEE ISC2 Conference.*
[21] C. Gomez, J. Oller, and J. Paradells. 2012. Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology. *Sensors* 12, 9 (2012).
[22] S.R. Hussain, S. Mehnaz, S. Nirjon, and E. Bertino. 2017. SeamBlue: Seamless Bluetooth Low Energy Connection Migration for Unmodified IoT Devices. In *Proc. of the* 14^{th} *EWSN Conference.*
[23] Ed. J. Hui, Arch Rock Corporation, P. Thubert, and Cisco. 2011. RFC 6282 - Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. https://tools.ietf.org/html/rfc6282. (2011).
[24] J. Nieminen et al. 2015. RFC 7668 - IPv6 over Bluetooth Low Energy. https://tools.ietf.org/html/rfc7668. (2015).
[25] C. Julien, C. Liu, A.L. Murphy, and G.P. Picco. 2017. BLEnd: Practical Continuous Neighbor Discovery for Bluetooth Low Energy. In *Proc. of the* 16^{th} *ACM/IEEE IPSN Conference.*
[26] R. Jurdak, K. Klues, B. Kusy, C. Richter, K. Langendoen, and M. Brunig. 2011. Opal: A Multiradio Platform for High Throughput Wireless Sensor Networks. *IEEE Embedded Systems Letters* 3, 4 (2011).
[27] P. Kindt, D. Yunge, M. Gopp, and S. Chakraborty. 2015. Adaptive Online Power-Management for Bluetooth Low Energy. In *Proc. of the IEEE INFOCOM Conference.*
[28] O. Landsiedel, F. Ferrari, and M. Zimmerling. 2013. Chaos: Versatile and Efficient All-to-all Data Sharing and In-network Processing at Scale. In *Proc. of the* 11^{th} *ACM SenSys Conference.*
[29] T. Lee, M. S. Lee, H. S. Kim, and S. Bahk. 2016. A Synergistic Architecture for RPL over BLE. In *Proc. of the* 13^{th} *IEEE SECON Conference.*
[30] Andreas Meier, Matthias Woehrle, Marco Zimmerling, and Lothar Thiele. 2010. ZeroCal: Automatic MAC Protocol Calibration. In *Proceedings of the 6th IEEE Int. Conf. on Distributed Computing in Sensor Systems (DCOSS).*
[31] K. Mikhaylov. 2014. Accelerated Connection Establishment (ACE) Mechanism for Bluetooth Low Energy. In *Proc. of the IEEE PIMRC Conference.*
[32] Mindtree. 2017. EtherMind Bluetooth 5 and 4.2 Stack & Profile for BR/EDR and Bluetooth low energy. http://www.mindtree.com/solutions/bluetooth-technology/ethermind. (2017).
[33] R. Musaloiu-E. and A. Terzis. 2007. Minimising the Effect of WiFi Interference in 802.15.4 Wireless Sensor Networks. *International Journal of Sensor Networks (IJSNet)* 3, 1 (2007).
[34] B. Al Nahas, S. Duquennoy, V. Iyer, and T. Voigt. 2014. Low-Power Listening Goes Multi-Channel. In *Proceedings of the* 10^{th} *IEEE DCOSS Conference.*
[35] P. Narendra, S. Duquennoy, and T. Voigt. 2015. BLE and IEEE 802.15.4 in the IoT: Evaluation and Interoperability Considerations. In *Proc. of the* 13^{th} *INDIN Conference.* 919–922.
[36] Nuki. 2017. The Bluetooth Door Lock for Smart Access via Smartphone. https://nuki.io/en/. (2017).
[37] R. Quinnell. 2017. BLE Module Guide for Quick and Easy Product Selection. *Electronic Products Magazine* (2017).
[38] Roche Media Release. 2016. Roche launches innovative Accu-Chek Guide blood glucose monitoring system. (Aug. 2016).
[39] C. Roest. 2015. *Enabling the Chaos Networking Primitive on Bluetooth LE.* Master's thesis. Delft University of Technology, Delft, The Netherlands.
[40] M. Siekkinen, M. Hiienkari, J.K. Nurminen, and J. Nieminen. 2012. How Low Energy is Bluetooth Low Energy? Comparative Measurements with ZigBee/802.15.4. In *Proc. of the IEEE WCNCW Workshop.*
[41] S.Raza, P. Misra, Z. He, and T. Voigt. 2017. Building the Internet of Things with Bluetooth Smart. *Ad Hoc Networks* 57 (2017).
[42] Texas Instruments. 2016. CC13xx, CC26xx SimpleLink Wireless MCU Technical Reference Manual. http://www.ti.com/lit/ug/swcu117f/swcu117f.pdf. (2016).
[43] Texas Instruments. 2017. Bluetooth Low Energy software stack. http://www.ti.com/tool/ble-stack. (2017).
[44] The Zephyr Project. 2017. An RTOS for IoT. https://www.zephyrproject.org/. (2017).
[45] Z. Shelby et al. 2012. RFC 6775 - Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs). https://tools.ietf.org/html/rfc6775. (2012).
[46] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele. 2012. pTunes: Runtime Parameter Adaptation for Low-power MAC Protocols. In *Proc. of the* 11^{th} *ACM/IEEE IPSN Conference.*

Publication B

M. Spörk, C. A. Boano, and K. Römer. Improving the Timeliness of Bluetooth Low Energy in Dynamic RF Environments. *ACM Transactions on Internet of Things*, 1(2), Apr. 2020

DOI: 10.1145/3375836
Link: `https://doi.org/10.1145/3375836`

Abstract. The ability to communicate within given delay bounds in noisy RF environments is crucial for Bluetooth Low Energy (BLE) applications used in safety-critical application domains, 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.

My contribution. I am the main author of this article and conceived the ideas to investigate the latency of BLE connections under harsh environmental conditions and to use existing HCI communication to monitor and control the BLE transmission delay. All experiments presented in this article were executed by me, although my colleague Markus Schuß significantly helped me in using the D-Cube testbed facility. I wrote the majority of the paper in close collaboration and discussion with my co-authors. This article is an extension of our conference paper published at EWSN'19 [201].

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

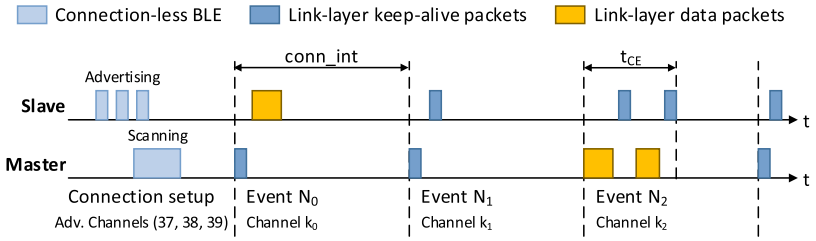


Fig. 1. BLE connection between slave and master.

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

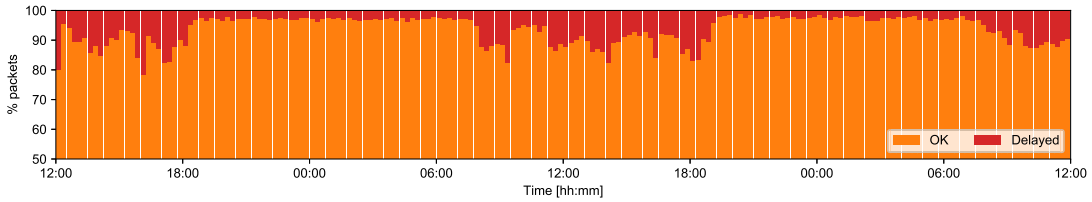


Fig. 2. Percentage of data packets exceeding t_{max} across 48 hours in a common office environment when using a Raspberry Pi 3 with its on-board Broadcom BCM43439 radio as BLE master. During daytime, up to 21.8% of the transmitted packets are delayed due to surrounding interference.

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

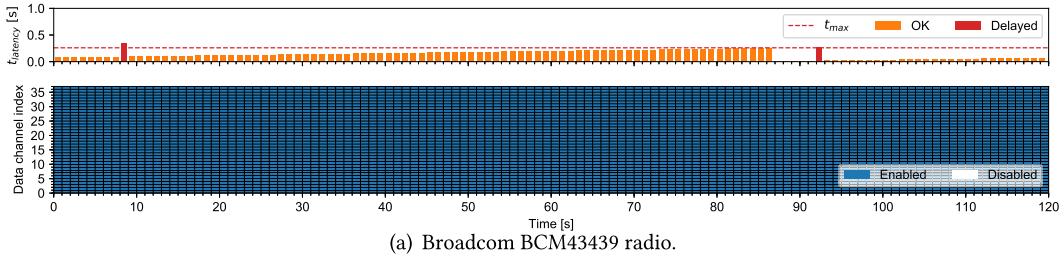
We perform our experiments on our testbed facility, which allows us to have fine-grained control over the RF noise experienced by each BLE node in our testbed.

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.

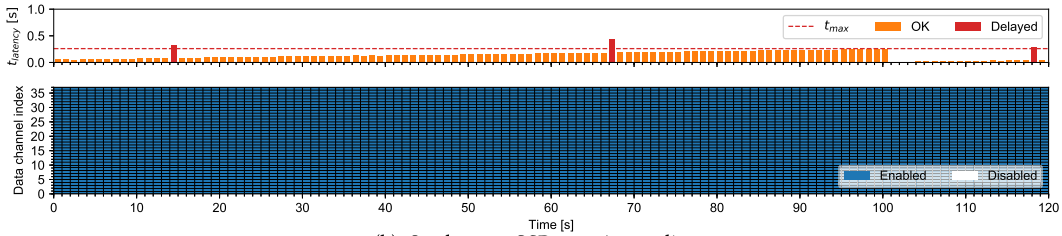
Generating interference. All RPi3 in the testbed are used to re-program the BLE nodes and to monitor the status of each experiment by logging data in persistent memory. We further use the RPi3s in the testbed to generate Bluetooth and Wi-Fi interference using their on-board Broadcom BCM43439 radio chip [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.

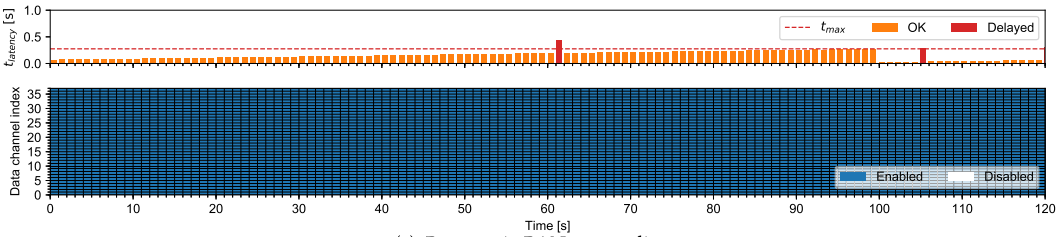
BLE slave. We use each of the `nRF52840` devices (except the one connected to the RPi3 acting as a master) as BLE slave. Each slave waits for the BLE master to initiate an IPv6-over-BLE connection and sends a UDP message to the master every second once the connection is established. Each UDP message 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}.

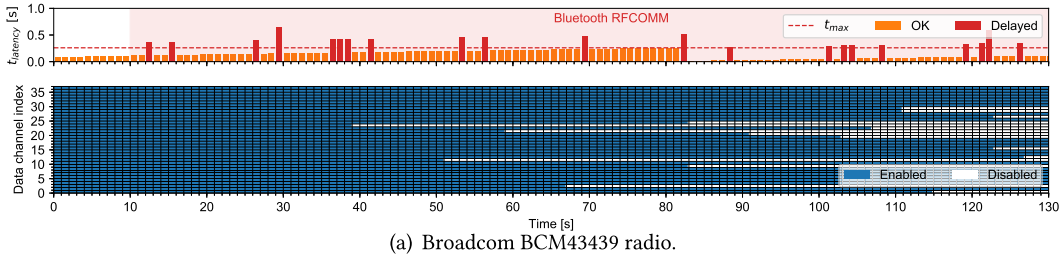
Furthermore, we see that non-delayed data transmissions (marked as OK in Figure 3) experience a $t_{latency}$ between t_{CE} and t_{max}. This is caused by the unsynchronized schedules of BLE application and BLE connection. As discussed in Section 2, an application can issue a data transmission at any 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.

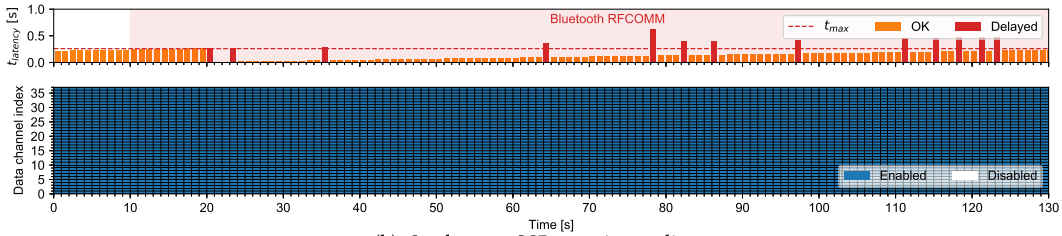
Each plot in Figure 4 shows $t_{latency}$ (top) and the data channel map (bottom) of a BLE connection between a master and a slave communicating at a distance of 10 meters with direct line-of-sight. Again, packets exceeding $t_{max} = 260\,ms$ (shown as horizontal dashed line) are marked as *delayed*. 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.

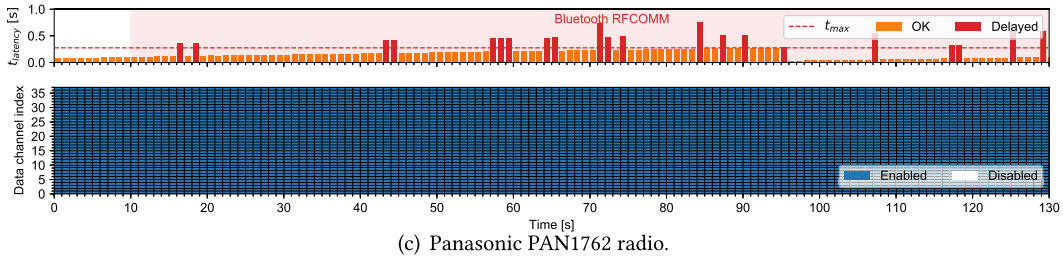
Furthermore, Figure 4(a) shows that the AFH implementation of the Broadcom BCM43439 is trying to update the data channel map to mitigate the effect of the Bluetooth interference on the BLE connection. However, the master is not able to accurately predict the frequencies used by Bluetooth and its blacklisting 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

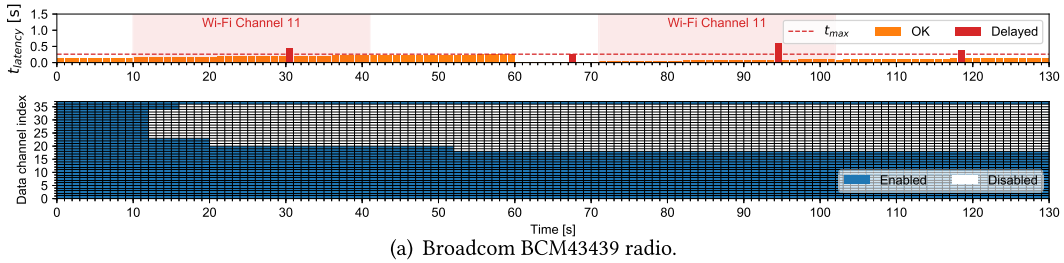
The table shows the percentage of delayed packets (DELAYED), the average (AVG), median (MED), 90 and 99 percentile (90% and 99%), and maximum experienced transmission delay (MAX) over 10 test runs per radio.

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

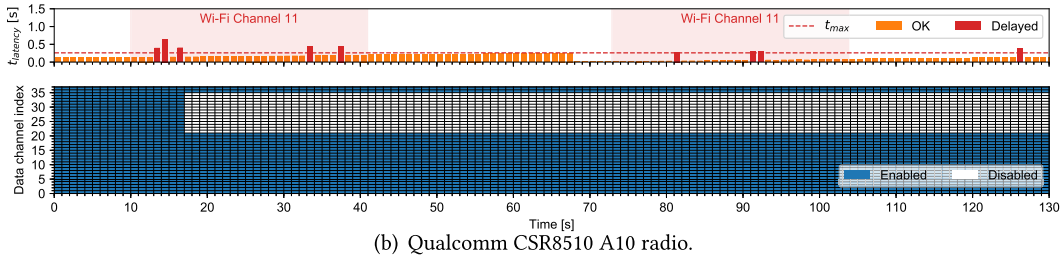
Note that the same effect shown in Figure 4 is experienced by every BLE slave in our testbed, even those that are only three meters away from the master and have direct line-of-sight.

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.

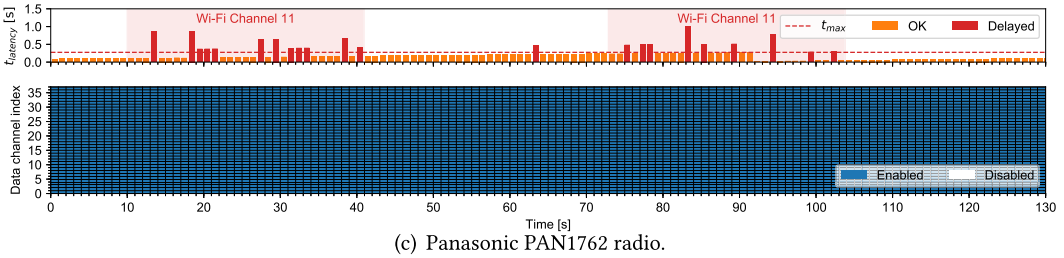
Wi-Fi interference near the master. The plots in Figure 5 show the measured $t_{latency}$ (top) and used data channel map (bottom) of a BLE connection in the presence of Wi-Fi interference located near the master. Also in this case, master and slave are at a distance of 10 meters with direct line-of-sight.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

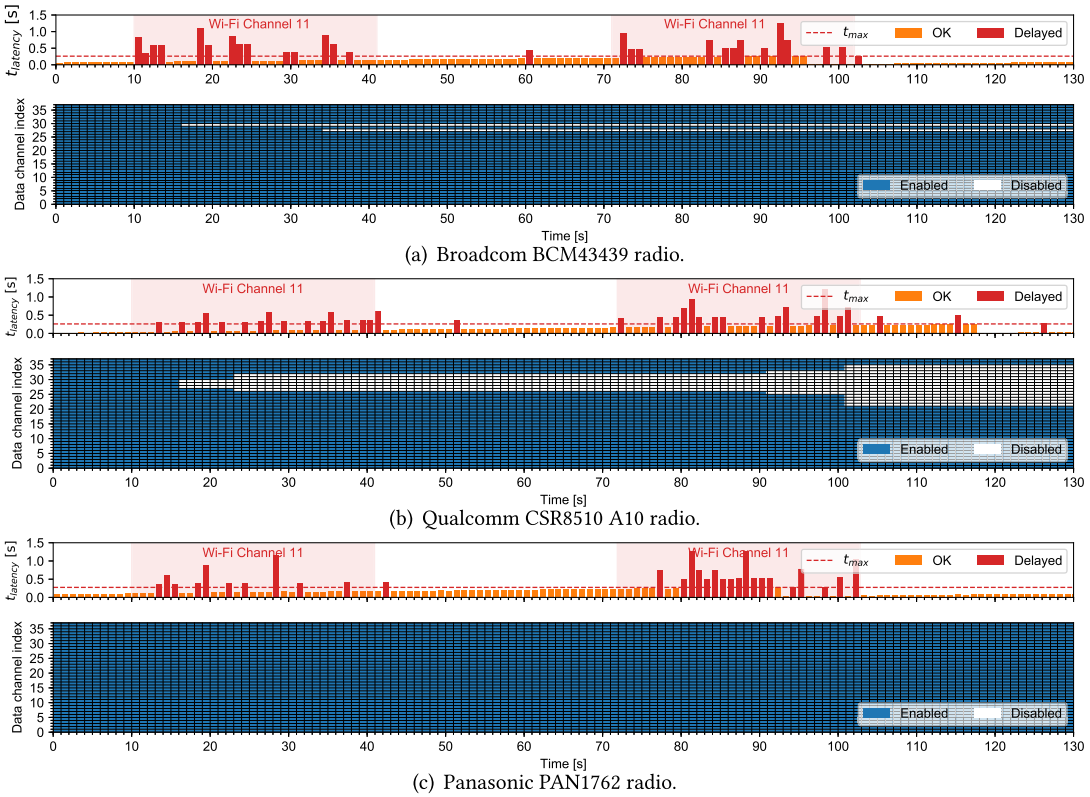
Table 3. Performance of the Three BLE Radios Acting as a BLE Master under *Wi-Fi Interference Near the Master*

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

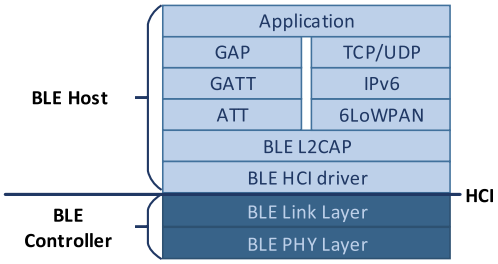


Fig. 7. Standard BLE and IPv6-over-BLE stack.

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.

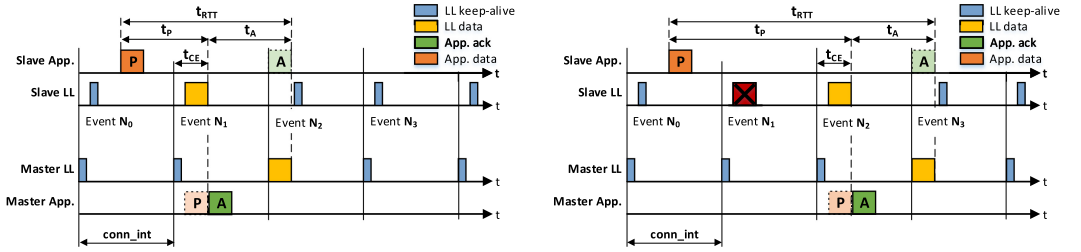
By knowing the n_{CE} of *each* fragment, Equation (2) now captures the impact of RF noise on the quality of a BLE connection and is hence able to provide an upper bound on transmission delay. This, however, requires a precise n_{CE} measurement.

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:

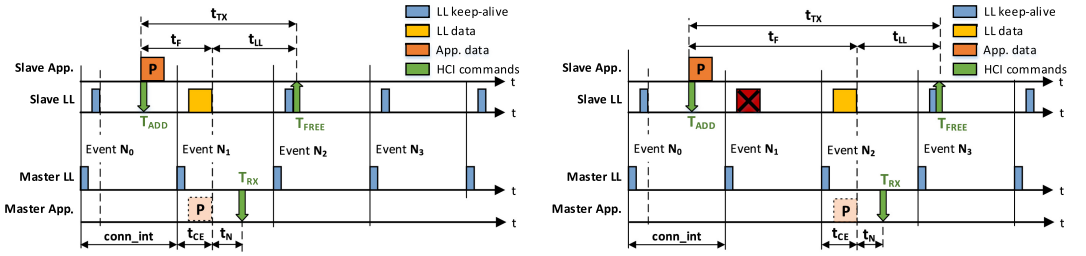
$$n_{CE_f} = \left\lceil \frac{t_{RTT} - 2 \cdot t_{CE}}{2 \cdot \lceil D/F \rceil \cdot conn\_int} \right\rceil. \tag{3}$$

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:

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

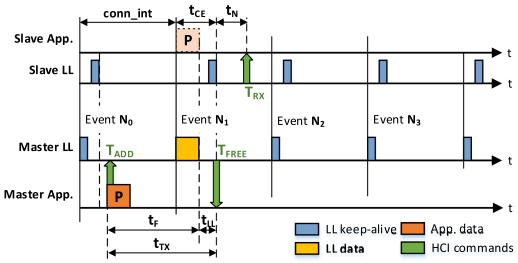


Fig. 10. HCI-based n_{CE} estimation for a BLE master transmitting a single data fragment ($n_{CE_f} = 1$).

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).

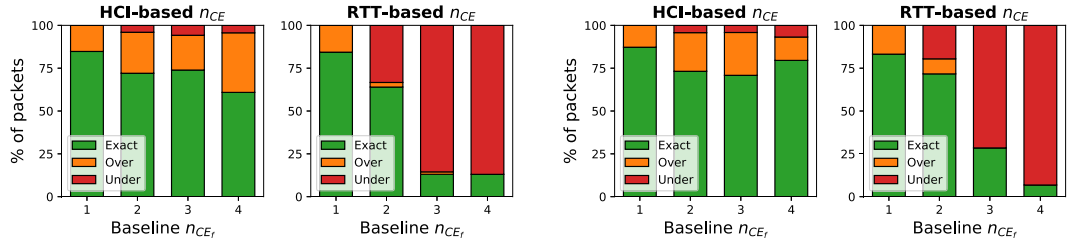
5.2 HCI-based n_{CE} Estimation

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:

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

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 (≤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

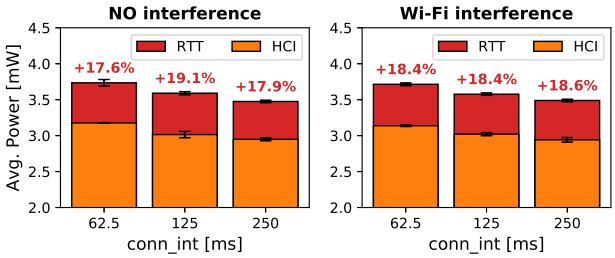


Fig. 12. Average power consumption of n_{CE} estimators for different connection intervals and interference.

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.

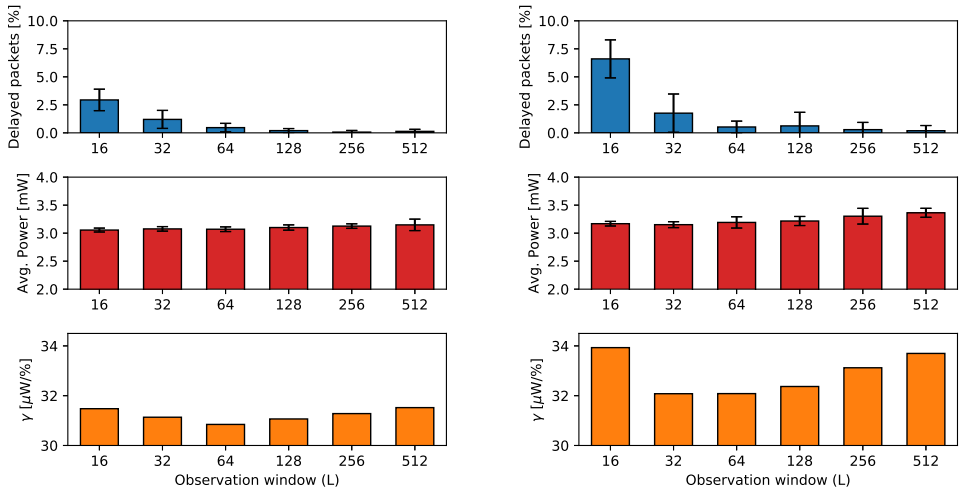
Finding an optimal *L*. We next experimentally investigate a suitable observation window length L. We consider six different lengths (16, 32, 64, 128, 256, and 512) and compute $n_{CE_f\star}$ according to Equation (12). We then instruct a BLE slave to adapt its connection interval according to Equation (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.

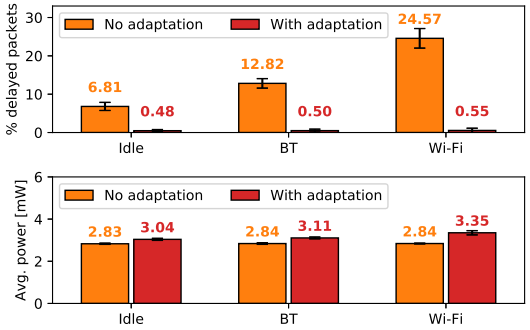


Fig. 14. Delayed packets and power consumption of a slave with and without connection interval adaptation.

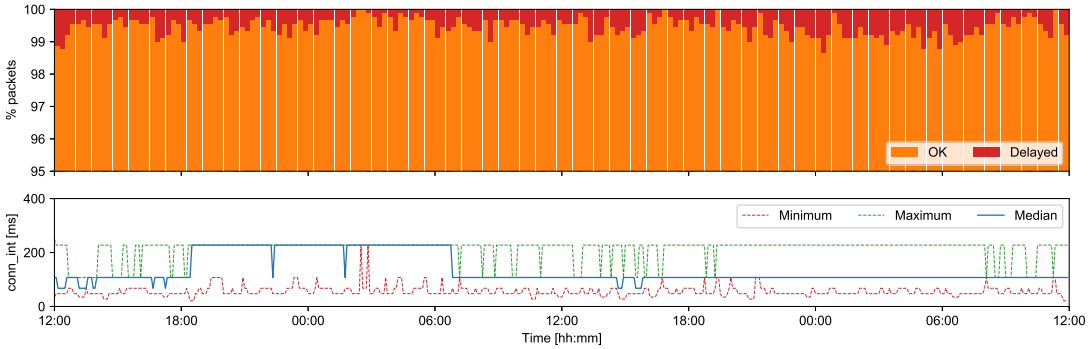


Fig. 15. When adapting its connection interval at runtime using an observation window of 64 fragments, a slave connected to a Broadcom BCM43439 master is able to significantly increase the timeliness of its BLE communications. Note the more granular y-axis of the top plot in this figure compared to Figure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

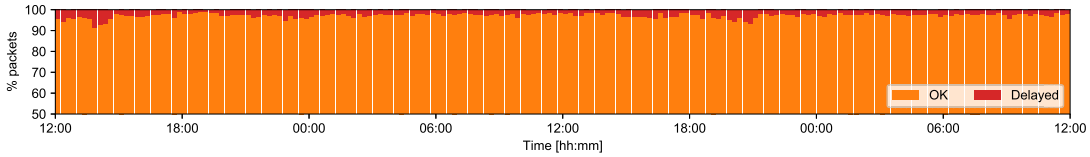


Fig. 16. Percentage of packets exceeding t_{max} across 48 hours in a student laboratory when *using a fixed connection interval* and a *Broadcom BCM43439 radio* as BLE master.

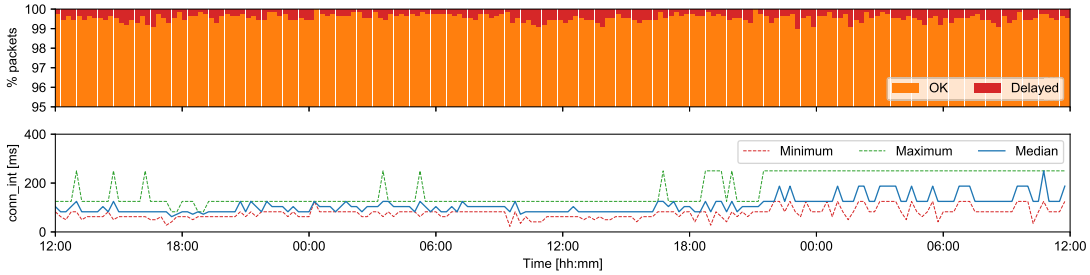


Fig. 17. Percentage of delayed packets (top) and adapted connection interval *conn_int* (bottom) over 48 hours in a student lab when using our *adaptive approach* and a *Broadcom BCM43439* as BLE master. Please note the more granular y-axis of the top plot in this figure compared to Figure 16.

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.

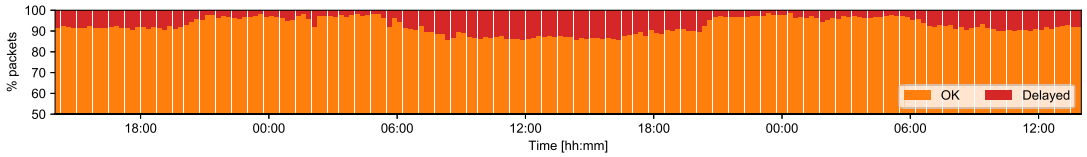


Fig. 18. Percentage of packets exceeding t_{max} across 48 hours in a student laboratory when *using a fixed connection interval* and a *Qualcomm CSR8510 A10 radio* as BLE master.

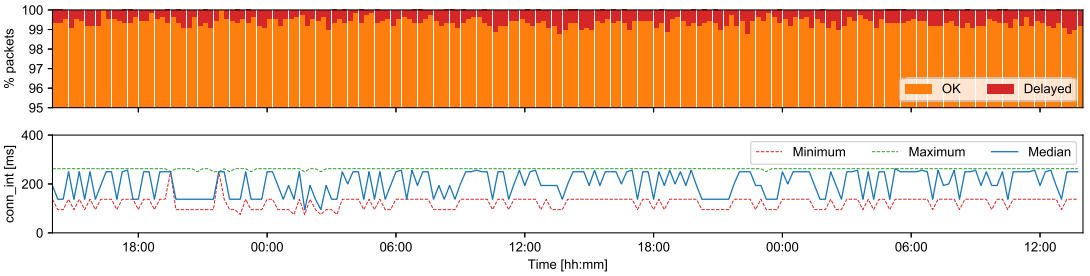


Fig. 19. Percentage of delayed packets (top) and adapted connection interval *conn_int* (bottom) over 48 hours in a student lab when using our *adaptive approach* and a *Qualcomm CSR8510 A10* as BLE master. Please note the more granular y-axis of the top plot in this figure compared to Figure 18.

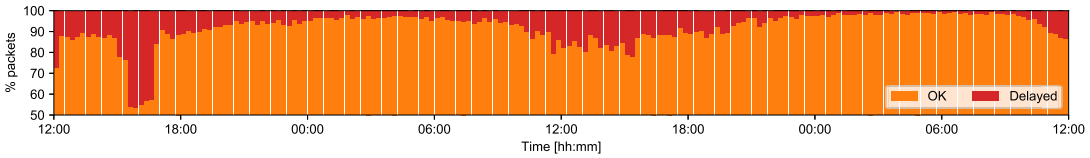


Fig. 20. Percentage of packets exceeding t_{max} across 48 hours in a student laboratory when *using a fixed connection interval* and a *Panasonic PAN1762 radio* as BLE master.

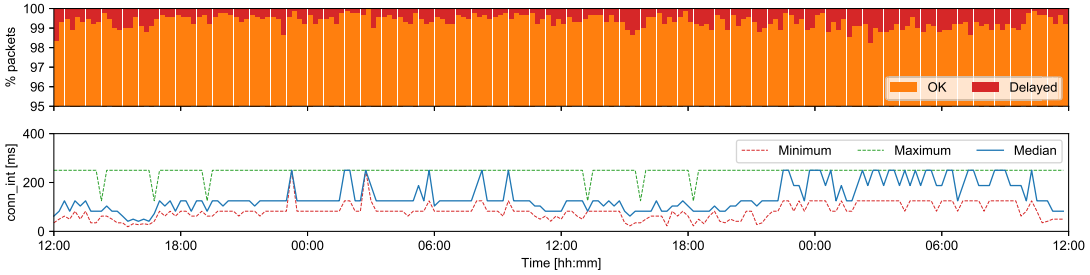
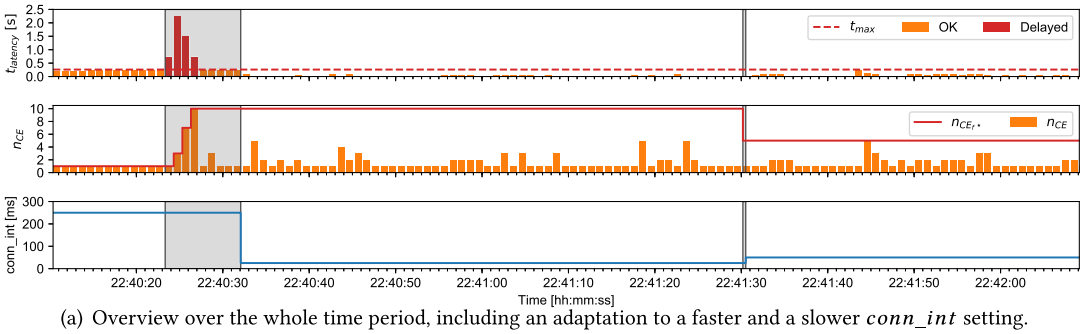
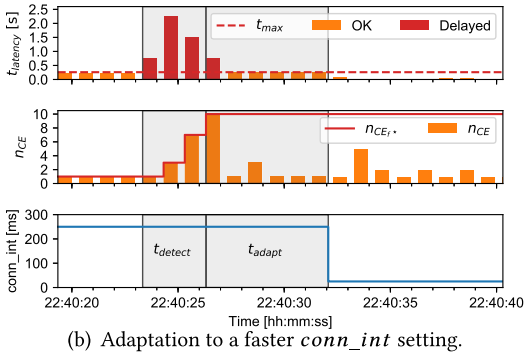


Fig. 21. Percentage of delayed packets (top) and adapted connection interval *conn_int* (bottom) over 48 hours in a student lab when using our *adaptive approach* and a *Panasonic PAN1762* as BLE master. Please note the more granular y-axis of the top plot in this figure compared to Figure 20.

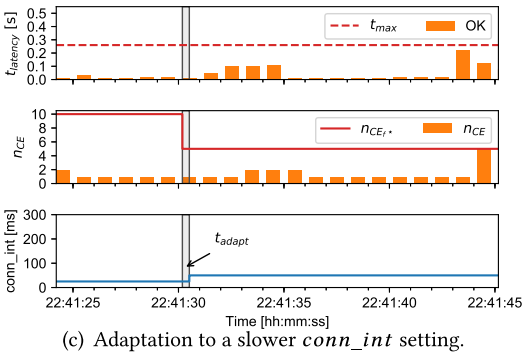
Panasonic PAN1762 radio. We repeat the experiment using two Panasonic PAN1762 radios as BLE masters. Figure 20 shows the percentage of delayed packets of S_{fixed} over two days, where the student lab was extensively used during daytime. Overall, 8.09% of all packets exceed t_{max} with a maximum of 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

Publication C

M. Spörk, C. A. Boano, and K. Römer. Performance and Trade-offs of the new PHY Modes of BLE 5. In *Proceedings of the International Workshop on Pervasive Systems in the IoT Era (PERSIST-IoT)*. ACM, July 2019

DOI: 10.1145/3331052.3332471
Link: `https://doi.org/10.1145/3331052.3332471`

Abstract. With the release of Bluetooth Low Energy (BLE) version 5, the Bluetooth Special Interest Group introduced three additional physical (PHY) modes for BLE communication. These PHY modes enable an application to either double its throughput, or significantly improve its reliability, making BLE applicable to an even wider range of application domains. Unfortunately, no experimental study has yet investigated the actual performance of BLE 5's PHY modes in BLE connections or shown their trade-offs between energy efficiency, reliability, and throughput. Thus, how to use BLE 5's PHY modes to achieve specific application requirements is still an open question.

To fill this gap, we experimentally study the performance of all four BLE 5 PHY modes in BLE connections and observe that it is, indeed, possible to double BLE's throughput or to increase BLE's reliability by using the new PHY modes. Furthermore, we provide guidelines using our measurements on how to select the most suitable PHY mode based on specific application requirements.

My contribution. I am the main author of this publication and have planned and executed all presented experiments. Furthermore, I wrote the vast majority of the paper in collaboration and discussion with my co-authors. I presented the paper at the 2019 PERSIST-IoT Workshop.

Performance and Trade-offs of the new PHY Modes of BLE 5

Michael Spörk
michael.spoerk@tugraz.at
Graz University of Technology
Graz, Austria

Carlo Alberto Boano
cboano@tugraz.at
Graz University of Technology
Graz, Austria

Kay Römer
roemer@tugraz.at
Graz University of Technology
Graz, Austria

ABSTRACT

With the release of Bluetooth Low Energy (BLE) version 5, the Bluetooth Special Interest Group introduced three additional physical (PHY) modes for BLE communication. These PHY modes enable an application to either double its throughput, or significantly improve its reliability, making BLE applicable to an even wider range of application domains. Unfortunately, no experimental study has yet investigated the actual performance of BLE 5's PHY modes in BLE connections or shown their trade-offs between energy efficiency, reliability, and throughput. Thus, how to use BLE 5's PHY modes to achieve specific application requirements is still an open question.

To fill this gap, we experimentally study the performance of all four BLE 5 PHY modes in BLE connections and observe that it is, indeed, possible to double BLE's throughput or to increase BLE's reliability by using the new PHY modes. Furthermore, we provide guidelines using our measurements on how to select the most suitable PHY mode based on specific application requirements.

CCS CONCEPTS

• **Networks** → **Network performance analysis**; • **Computer systems organization** → *Embedded systems.*

KEYWORDS

Bluetooth Low Energy, BLE 5, Throughput, Reliability

ACM Reference Format:
Michael Spörk, Carlo Alberto Boano, and Kay Römer . 2019. Performance and Trade-offs of the new PHY Modes of BLE 5. In *ACM MobiHoc Workshop on Pervasive Systems in the IoT Era (PERSIST-IoT'19), July 2, 2019, Catania, Italy.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3331052.3332471

1 INTRODUCTION

Bluetooth Low Energy (BLE) has become a pervasive wireless technology to connect constrained and low-power devices to the Internet of Things (IoT). BLE does not only provide an energy-efficent and reliable way of communication; its wide adoption in almost all consumer electronic devices, like smartphones and tablets, makes it the technology of choice for a wide range of application domains, such as smart health [6], smart cities [2], and smart homes [14].

In order to support an even wider range of applications, the Bluetooth Special Interest Group released version 5 of the Bluetooth specification, so called BLE 5, in June 2016 [4]. Besides a longer advertising packet length and increased coexistence capabilities, BLE 5 introduces three new physical (PHY) modes. One of these, the 2M PHY, promises to double BLE's throughput. The other two modes, the Coded S2 PHY and the Codes S8 PHY, promise to increase BLE's communication reliability [3].

Although BLE 5 devices have been available since 2017 [9] and even BLE version 5.1 (enabling a more advanced localization using multiple antennas) has been released in 2019, no experimental study has, to the best of our knowledge, investigated if the new PHY modes of BLE 5 actually perform as advertised when used in BLE connections. Furthermore, how to use the different PHY modes to sustain specific application requirements, such as a certain power consumption, communication reliability, or throughput, has not been studied in detail and still remains an open question.

Contributions. In this paper, we fill this gap and experimentally study the performance of BLE 5 and its new PHY modes. Our investigation allows to understand: (i) whether the BLE 5 PHYs deliver on their promises, and (ii) how to select the best PHY for a given application. To this end, we perform the first comprehensive experimental study of all four BLE 5 PHY modes used in a BLE connection and answer the following questions:

- Does the 2M PHY really allow to double the throughput?
- Do the Coded S2 PHY and the Coded S8 PHY really increase the relibility of a BLE connection?
- How does the chosen PHY mode affect the overall power consumption of a BLE device?

Based on these measurements, we show the trade-off between energy efficiency, reliability, and throughput for each PHY mode and provide guidelines on how to select the most suitable PHY for a given application. For this purpose, we derive the effective throughput and effective power consumption of all four PHY modes for BLE connections with different link quality and investigate:

- Which PHY provides the maximum effective throughput?
- Which PHY minimizes the effective power consumption?

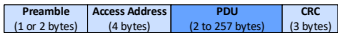
The remainder of this paper is structured as follows: Sect. 2 provides the necessary technical background on the four PHY modes of BLE 5 and connection-based BLE. Sect. 3 describes our experimental setup and evaluates the (i) power consumption, (ii) data throughput, (iii) packet reception rate, and (iv) robustness to Wi-Fi interference of a BLE 5 connection using the four PHY modes. Sect. 4 provides guidelines on choosing the best PHY mode for specific application requirements. Sect. 5 lists related work and Sect. 6 summarizes our findings and discusses future work.

2 BLE 5 PRIMER

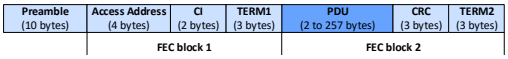
In this section, we provide the necessary technical details on the different PHY modes of BLE 5 (Sect. 2.1) and discuss how they are used in BLE connections for bi-directional data exchange (Sect. 2.2).

PERSIST-IoT'19, July 2, 2019, Catania, Italy

ACM ISBN 978-1-4503-6805-6/19/07…$15.00
https://doi.org/10.1145/3331052.3332471



(a) BLE link layer packet format for the 1M and 2M PHY mode.



(b) BLE link layer packet format for the Coded S2 and S8 PHY mode.

Figure 1: Overview of the BLE link layer packet structure.

2.1 BLE 5 PHY Modes

With the publication of the BLE 5 specification [4] in June 2016, three additional physical (PHY) modes, the 2M, the Coded S2, and the Coded S8 PHY mode, were introduced. While the 2M PHY, where the M stands for Megasymbols/s (Msym/s), promises twice the data rate compared to the existing 1M PHY mode, the two Coded PHY modes are meant to increase the communication reliability of BLE devices [3]. Hence, the four PHY modes provide application developers with additional possibilities to fine-tune BLE's performance to individual application requirements, such as energy-efficiency, throughput, and reliability.

To achieve these different characteristics, the PHY modes use different error correction/detection, modulation, and coding schemes. Besides, all PHYs use Gaussian Frequency Shift Keying on the 40 BLE radio channels located in the 2.4 GHz ISM band.

2.1.1 1M PHY. This is the original mode of BLE and was the sole mode for all BLE communication with a version number below 5. Therefore, this is the only backwards-compatible mode that can be used with BLE devices not supporting BLE 5. In this mode, the modulation scheme supports a physical modulation of 1 Msym/s, meaning that transmitting a single bit of payload takes 1 μs. All packet data is not coded and, therefore, has no error correction.

Fig. 1a shows the link-layer packet format of the 1M PHY mode. The preamble is 1-byte long and is followed by 4 bytes containing the access address. After the Protocol Data Unit (PDU), which has a variable length between 2 and 257 bytes, the packet ends with a 3-byte CRC checksum, which is used to check for packet corruption.

2.1.2 2M PHY. In case an application needs to sustain a high throughput, it may use the 2M PHY mode of BLE. In contrast to the other three PHYs, this mode uses a physical modulation of 2 Msym/s, resulting in 0.5 μs of air time for a single payload bit. Similar to the 1M PHY, data sent with the 2M PHY is not coded and has no error correction. The 2M PHY link-layer packet format is similar to the format of the 1M PHY shown in Fig. 1a, however a 2-byte preamble is used. Hence, the new 2M PHY promises twice the throughput at the cost of a lower reliability for poor link qualities.

2.1.3 Coded S2 PHY. When a more reliable communication is needed (*e.g.*, due to a long communication distance or the presence of co-located radio interference), the Coded S2 PHY mode may be used. This mode uses a physical modulation of 1 Msym/s, but makes use of forward error correction (FEC) with a symbol coding of 2 (S2), leading to an increased robustness. A single data bit encoded with S2 coding takes 2 μs on the air, resulting in a data rate of 500 kb/s.

The link layer packet format of this mode is shown in Fig. 1b. As this figure shows, the packet is split into three parts: a preamble, a FEC block 1, and a FEC block 2. The preamble is 80 μs long and is sent without coding. The FEC block 1 contains the access address as well as the coding indicator (CI) and ends with the first termination field (TERM1). The FEC block 2 contains the PDU and a 3-byte CRC checksum and is terminated by a second termination field (TERM2). According to the BLE specification [4], even when a packet is sent with the Coded S2 PHY, the FEC block 1 always uses S8 Coding. Compared to the 1M PHY, this PHY improves BLE's reliability, at the cost of less throughput and an increased power consumption.

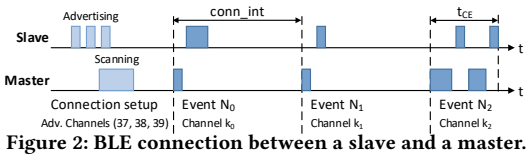


Figure 2: BLE connection between a slave and a master.

2.1.4 Coded S8 PHY. The Coded S8 PHY uses an even more robust coding and error correction scheme than the Coded S2 PHY. This PHY also transmits with a physical modulation of 1 Msym/s, but uses an FEC with a symbol coding of 8 (S8) for the whole packet. Using the Coded S8 PHY, a single data bit takes 8 μs on the air. Fig. 1b shows the link-layer packet format of the Coded S8 PHY mode. Using the Coded S8 PHY, all packet fields are sent with a coding of 8, resulting in a data rate of 125 kb/s for the whole packet. The Coded S8 PHY promises to improve BLE's reliability for poor link qualities even further, at the cost of a lower throughput and increased power consumption compared to the other PHYs.

2.2 BLE Connections

BLE supports two modes of communication: a *connection-less* and a *connection-based* mode. In the connection-less mode, a device is either broadcasting short data packets on the three BLE advertisement channels (37, 38, and 39) or scanning for such broadcast messages. However, if two devices need to bidirectionally exchange data packets, they need to use connection-less primitives to establish a BLE connection. In the connection-based mode, one device acts as a master and the other as a slave; communication takes place during *connection events* (N_0 ... N_i), as shown in Fig. 2.

The time between the start of two consecutive connection events is defined by the *connection interval* (*conn_int*). Every connection event starts with a link-layer packet from the master, to which the slave responds. In case master and slave have no additional data to send, the connection event ends after this mandatory exchange of keep-alive messages. If, however, more data needs to be transmitted, master and slave keep exchanging link-layer packets until all data is successfully sent or the *maximum connection event length* (t_{CE}) is reached. The last link-layer packet during a connection event is always sent by the slave, after which both devices disable their radio and resume communication at the next connection event.

Fig. 2 shows an example where, after the connection setup using connection-less primitives, the master starts connection event N_0 by sending a short keep-alive packet to the slave. The slave has data to send and therefore responds with a link-layer packet (which is longer than the keep-alive packet from the master) carrying the data. During connection event N_1, both devices have no data to transmit and therefore only exchange the mandatory keep-alive packets. In connection event N_2, however, the master has data to send and therefore starts the connection event by sending a link-layer packet carrying data. Because the master has additional data to send, it waits for the slave's response before sending a second

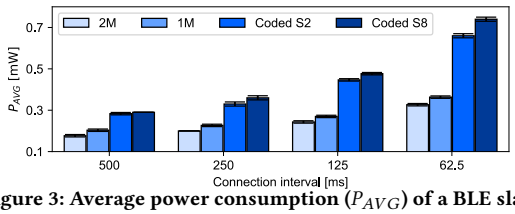


Figure 3: Average power consumption (P_{AVG}) of a BLE slave for different connection intervals and PHY modes when using a fixed PDU length of 253 bytes.

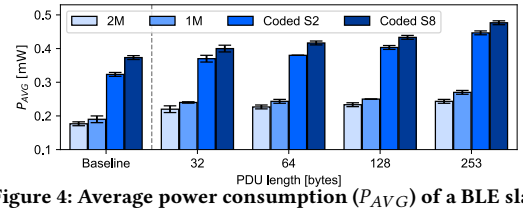


Figure 4: Average power consumption (P_{AVG}) of a BLE slave for different PDU lengths and PHY modes when using a fixed connection interval of 125 ms.

link-layer packet containing the rest of the data. The slave responds with a second link-layer packet, ending the connection event.

In the connection-based BLE mode, the link layer autonomously handles packet acknowledgment (ACK) and flow control using a 1-bit ACK field and a 1-bit sequence number in every link-layer packet header. If a link-layer packet was not successfully sent, it is automatically retransmitted. To further ensure reliable communication, BLE connections use adaptive frequency hopping (AFH). Using AFH, one of the enabled data channels in the *data channel map* is selected at the start of every connection event and is used by master and slave to exchange all packets during the event.

3 MEASURING THE PERFORMANCE OF BLE 5

To evaluate the performance of the four different PHY modes of BLE 5, we perform an experimental study (Sect. 3.1) and measure the power consumption (Sect. 3.2), the maximum achievable throughput (Sect. 3.3), the link-layer packet reception rate (Sect. 3.4), and the robustness under Wi-Fi interference (Sect. 3.5) of a BLE connection.

3.1 Experimental Setup

We perform our experiments on a testbed located in a vacant University lab (6x10 meters). The measurements were taken on a BLE connection between a BLE master and slave for all four PHY modes.

BLE master and slave have direct line of sight and use a transmission power of 0 dBm in all of our experiments.

BLE master. We use an nRF52840 DK device from Nordic Semiconductor [10] as a BLE master for all of our measurements. The master scans for a BLE slave and initiates a BLE connection. After the connection has been established, the master subscribes to a custom Generic Attribute Profile (GATT) attribute on the slave.

BLE slave. We use another nRF52840 DK device as a BLE slave that advertises its presence and waits for a BLE connection to be initiated by the BLE master. Once the master has successfully subscribed to the custom GATT attribute, the slave periodically notifies the master with a GATT notification. In our experiments, we are able to vary the length of the GATT notification in bytes and the time between two consecutive notifications.

Controlling the BLE PHY mode. Both master and slave run the Zephyr operating system [19]. Zephyr provides a standard-compliant BLE stack allowing access to the inner workings of the BLE link layer. This way, we have fine-grained control over the connection settings and the PHY mode of the BLE connection.

3.2 Power Consumption

To evaluate the impact of the different PHY modes on the power consumption of a system, we use the experimental setup described in Sect. 3.1 and measure the power consumption of a BLE slave in different configurations. We focus on the power consumption of the slave, as it usually operates on a constrained energy budget, such as a coin cell battery, while the BLE master usually has a continuous power supply. The power consumption of a master sustaining a single BLE connection is comparable to the slave's consumption, but increases with every connection that needs to be maintained.

Once the master has established a BLE connection and has subscribed to the custom GATT attribute, the slave periodically sends a notification of configurable length to the master every 1000 ms. In this set of experiments, master and slave have a distance of approximately 1 meter and direct line of sight. We disable all debugging and application logging features on the slave and use the Monsoon Power Monitor [8] to measure the system's power consumption. We vary the used PHY mode, connection interval, and PDU length and measure the average power consumption (P_{AVG}) of the slave for every settings over 120 seconds. We repeat the measurements for each configuration five times to ensure statistical significance.

Fig. 3 shows P_{AVG} for different PHY modes and connection intervals when using a fixed PDU length of 253 bytes. First, we can clearly observe that P_{AVG} increases when using a lower connection interval, as the BLE radio is more active (see Sect. 2.2). This matches our expectations as well as the models and measurements shown in [17]. Second, we can see a significant difference in power consumption when different PHY modes are used. As expected, the 2M PHY mode results in the lowest P_{AVG}, as it has the lowest radio duty cycle due to its fast data rate. The Coded S8 PHY, however, leads to the highest power consumption, because of its higher radio duty cycle caused by the overhead of the employed coding scheme. Compared to the legacy 1M PHY, the 2M PHY consumes approximately 8% less power in our experiments. The Coded S2 and S8 PHY consume approximately 61% and 70% more power compared to the 1M PHY for all four connection intervals, respectively.

Fig. 4 shows P_{AVG} of the slave for different PHY modes and PDU lengths when using a fixed connection interval of 125 ms. The values labeled `Baseline` shows P_{AVG} when the BLE connection is alive, no data is transmitted by the application, therefore showing the power consumed for maintaining the BLE connection (*i.e.*, to exchange only keep-alive packets). Similar to the data shown in Fig. 3, the 2M PHY mode is the most energy efficient, while the Coded S8 PHY mode results in the highest power consumption.

Comparing Fig. 3 and Fig. 4 we see that the used connection interval has a high impact on P_{AVG}. For example, a BLE slave using the 2M PHY mode consumes approximately 85% more power when using a connection interval of 62.5 ms instead of 500 ms. Using the Coded S8 PHY, a slave consumes even 155% more power when using

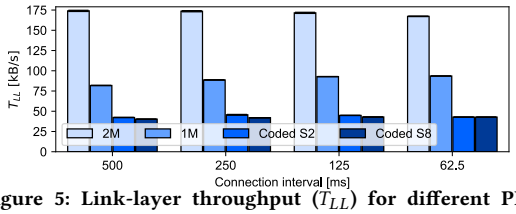


Figure 5: Link-layer throughput (T_{LL}) for different PHY modes and connection intervals measured at the master. *Note that the standard deviation is at most ±0.4% in our tests and, therefore, not clearly visible in the figure.*

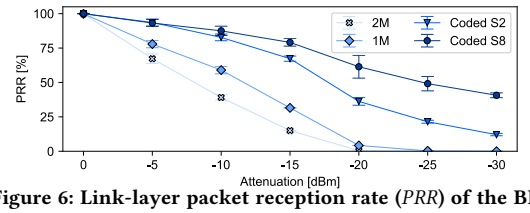


Figure 6: Link-layer packet reception rate (*PRR*) of the BLE connection for different PHY modes and attenuation values.

a connection interval of 62.5 ms instead of 500 ms. The used PDU length, however, only slightly increases the power consumption, leading to 20% more consumption when sending a PDU length of 253 bytes instead of 32 bytes for the Coded S8 PHY mode. We also see that the power used for maintaining the BLE connection (shown as `Baseline` in Fig. 4) accounts for a significant portion, between 72% and 91%, of the overall power consumption of the system.

3.3 Throughput

Next, we measure the maximum achievable throughtput of the different PHY modes of BLE 5. We use the experimental setup discussed in Sect. 3.1 and keep BLE master and slave at an approximate distance of 1 meter with direct line of sight. Similar to Sect. 3.2, the master initiates a BLE connection and subscribes to the custom GATT attribute on the slave. However, for these measurements the slave sends a new notification every 5 ms with a PDU length of 253 bytes, filling the outgoing transmission buffer of the slave.

The master application measures the time it takes to receive 10000 subsequent GATT notifications from the slave and uses the measured time to calculate the link-layer throughput (T_{LL}) of the BLE connection. Every throughput measurement consists of 10 test runs, where each run consists of 10 throughput measurements (over 10000 GATT notifications) per PHY mode and connection interval.

To achieve the maximum possible throughput, we configure the BLE devices to use the maximum number of link-layer transmission and reception buffers (18 and 19, respectively) and increase the L2CAP buffer and fragment count to 50. This maximizes the connection event length t_{CE} of the BLE connection (see Sect. 2.2) and hence the number of packets sent during a connection event.

Fig. 5 shows the T_{LL} for different PHY modes and connection intervals measured at the master. As expected, the 2M PHY, having a physical modulation of 2 Msym/s, provides the highest throughput of all PHYs, while the Coded S8 PHY has the lowest T_{LL} in our experiments. According to our measurements, the 2M PHY provides between 178% and 212% of the 1M PHY mode's throughput, therefore keeping its promise of doubling its throughput [3]. Contrary to our expectations in Sect. 2.1, the Coded S8 PHY mode provides almost 50% of the throughput of the 1M PHY mode. This, however, can be explained by the behavior of the BLE stack of Zephyr [19] on the nRF52840 platform. Whenever the transmission buffers of the BLE link layer get filled, the logic in the link layer implementation always uses the Coded S2 PHY, even if the Coded S8 PHY was chosen by the developer. Therefore, although we configure master and slave to use the Coded S8 PHY, they autonomously switch to the Coded S2 PHY mode when the BLE buffers are filled.

Another observation from Fig. 5 is that the used connection interval has no significant impact on T_{LL} in our experiment, because we configure the BLE buffers on master and slave so that multiple notification packets can be transmitted in a single connection event.

3.4 Packet Reception Rate

To estimate the differences in communication reliability of the PHY modes of BLE 5, we measure the link-layer packet reception rate (PRR) for different link qualities. We use the experimental setup described in Sect. 3.1, but change the distance between the master and slave to approximately 10 meters with direct line of sight. In order to accurately and repeatably measure the PRR for different link qualities, we connect a programmable attenuation device [7] and an external 2.4 GHz antenna to the antenna connector of the slave. This programmable attenuator allows us to have fine-grained control over the antenna attenuation on the slave, which we use to lower the link quality of the BLE connection in 5 dBm steps.

The master initiates a BLE connection with a connection interval of 25 ms and subscribes to the custom GATT attribute of the slave, similar to the experiments described above. The BLE slave tries to send a notification with a PDU length of 253 bytes every 25 ms. If a notification is pending (*i.e.*, it has not been successfully sent yet), no new notification is added by the slave, leading to a maximum of one notification sent per connection event.

We insert log ouputs into the link layer implementation of the BLE master to accurately measure the link-layer PRR of the BLE connection. The master's link layer logs the number of link-layer transmissions and retransmissions, which we use to calculate the PRR. We also limit the data channel map of the BLE connection to the BLE channels 12 to 19, as these channels are not interfered by any co-located radio technology; this minimizes the effects of RF noise on our measurements. We measure the PRR for every PHY and attenuation setting over more than 3000 connection events per setting and repeat each test 10 times.

Fig. 6 shows the PRR of the four PHY modes of BLE 5 for different attenuation values. The x-axis of Fig. 6 shows the effective attenuation of the antenna of the slave (the +3 dBm of the external 2.4 GHz antenna minus the configured attenuation on the programmable attenuator). As expected, the 2M PHY mode has the lowest PRR out of the four available PHY modes. For example, while the Coded S2 and S8 PHYs provide a PRR of 67% and 80% for an attenuation of -15 dBm, the 2M PHY is only able to sustain a 15% PRR. The 1M PHY is able to sustain a PRR of 32% for an attenuation of -15 dBm. Hence, the Coded S2 PHY and the Coded S8 PHY increase the link-layer PRR of BLE connections for poor link qualities, and therefore BLE's reliability, due to their employed coding schemes.

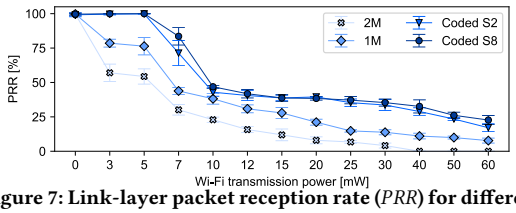


Figure 7: Link-layer packet reception rate (*PRR*) for different PHY modes and interfering Wi-Fi transmission power. *Note the non-linear x-axis scale showing the Wi-Fi transmission power.*

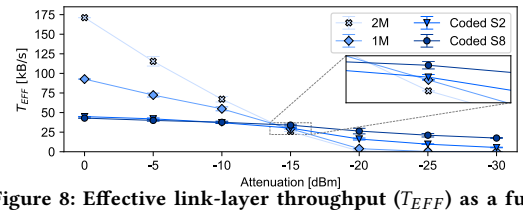


Figure 8: Effective link-layer throughput (T_{EFF}) as a function of the PHY mode and the attenuation of the BLE slave antenna when using a connection interval of 125 ms.

3.5 Robustness to Interference

We finally experimentally evaluate the robustness under Wi-Fi interference of the four PHY modes of BLE 5. To this end, we measure the link-layer packet reception rate (PRR) of a BLE connection in the presence of co-located Wi-Fi interference. We make use of the experimental setup from Sect. 3.1 and introduce a Raspberry Pi 3 (RPi3) [12], which we use to generate repeatable Wi-Fi interference. For this experiment, we place the BLE master and slave at a distance of approximately 3 meters with direct line of sight. The RPi3 used for Wi-Fi jamming is placed at a distance of 1 meter to the slave and 4 meters to the master. To create Wi-Fi interference, we use `JamLab-NG` [15] and let the RPi3 generate IEEE 802.11b packets on Wi-Fi channel 6 using its on-board Broadcom `bcm43438` radio. We let the RPi3 send a 1500-byte long packet every 10 milliseconds.

Similar to the previous experiments, the master initiates a BLE connection with the slave and subscribes to the slave's custom GATT attribute. The BLE connection is configured to use a connection interval of 125 ms and uses only the BLE data channels 12 to 19 that all overlap with the Wi-Fi channel 6 where interference in generated. The slave sends a GATT notification with a PDU length of 253 bytes in the same way as described in Sect. 3.4. We use the log output of the master's link layer and count the link-layer transmissions and retransmissions, which we use to calculate the PRR of the BLE connection. We measure the PRR for 5 minutes (resulting in over 2400 values per test) for every PHY and Wi-Fi transmission power and repeat the experiment 10 times for each setting.

Fig. 7 shows the average PRR of the BLE connection for different PHY modes and Wi-Fi transmission power settings. Our measurements show that, as expected, the Coded S8 PHY mode provides the highest PRR and thus the highest reliability under interference. The data also show that the Coded S2 and S8 PHY increase the link budget by 5dBm under Wi-Fi interference. While the Coded S2 and S8 PHYs are able to sustain almost 100% PRR, the 2M PHY only provides a PRR of 54% for a Wi-Fi transmission power of 5mW.

4 CHOOSING THE MOST SUITABLE PHY

The measurements in Sect. 3 show that the used BLE 5 PHY mode significantly influences the energy efficiency, throughput, and reliability of a BLE connection. Based on our measurements, we investigate how to maximize the effective throughput (Sect. 4.1) and minimize the effective power consumption (Sect. 4.2) by selecting the most suitable PHY mode (Sect. 4.3).

4.1 Maximizing Throughput

In this section, we answer the question: *Which PHY mode of BLE 5 provides the maximum effective data throughput?* To this end, we calculate the effective link-layer throughput of the four PHY modes of BLE 5 for different link qualities of the used BLE connection.

We define the effective link-layer throughput (T_{EFF}) as the number of link-layer bytes per second that are successfully sent over a BLE connection and calculate T_{EFF} as

$$T_{EFF} = PRR \cdot T_{LL}, \tag{1}$$

where *PRR* is the measured link-layer packet reception rate for a given PHY mode and antenna attenuation (shown in Sect. 3.4), whereas T_{LL} is the maximum achievable link-layer throughput of a given PHY mode and connection interval (shown in Sect. 3.3).

Fig. 8 shows T_{EFF} for different BLE antenna attenuations when using a connection interval of 125 ms. The data suggest that the best PHY mode to sustain a maximum effective throughput depends on the link quality of the BLE connection (indicated by the BLE antenna attenuation). In case only a few link-layer data packets are corrupted and thus need to be retransmitted, the 2M PHY mode provides the highest T_{EFF}. If packets are frequently corrupted, because of a poor link quality of the underlying BLE connection, the Coded S8 PHY is able to recover most corrupted packets and hence achieves the highest effective throughput. Using the 1M or the Coded S2 PHY mode always leads to a suboptimal T_{EFF}.

Results for connection intervals of 62.5 ms, 250 ms, and 500 ms show similar behavior, but are omitted due to space constraints.

4.2 Minimizing Power Consumption

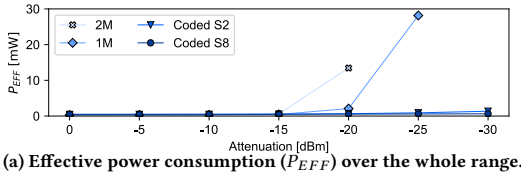
In this section, we answer the question: *Which PHY mode of BLE 5 minimizes the effective power consumption?* To this end, we calculate the effective power consumption of a slave using the four PHY modes of BLE 5 for different link qualities of the BLE connection.

We define the effective power consumption (P_{EFF}) as the overall power consumption of a slave periodically transmitting application data, accounting for the additional power consumption introduced by packet retransmissions. P_{EFF} is calculated as

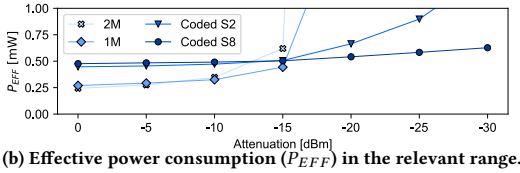
$$P_{EFF} = P_M + \frac{P_{DATA}}{PRR}, \tag{2}$$

where P_M is the overall power consumption of a slave for maintaining the BLE connection, *i.e.*, to only exchange the mandatory keep-alive packets. The P_M value for a given connection inverval and PHY mode is shown in Fig. 4 labeled as `Baseline`. *PRR* is the link-layer packet reception rate for a given PHY mode and antenna attenuation (shown in Sect. 3.4). Finally, P_{DATA} is the power consumed for transmitting the actual application data over the BLE connection. By measuring the average power consumption while transmitting data (P_{AVG}) and the power consumed for maintaining the BLE connection (P_M), P_{DATA} can be calculated as

$$P_{DATA} = P_{AVG} - P_M. \tag{3}$$



(a) Effective power consumption (P_{EFF}) over the whole range.



(b) Effective power consumption (P_{EFF}) in the relevant range.

Figure 9: Effective power consumption (P_{EFF}) as a function of the used PHY and antenna attenuation of a BLE slave using a connection interval of 125 ms. *Note that (a) shows P_{EFF} over its whole range, while (b) shows the range relevant for analysis.*

Fig. 9 shows the effective power consumption (P_{EFF}) of a BLE slave when using a connection interval of 125 ms and a PDU length of 253 bytes. Similar to the analysis in Sect. 3.3, the data in Fig. 9 suggest that the most energy efficient PHY mode mainly depends on the link quality of the BLE connection. In case only a few link-layer packets are corrupted and therefore retransmitted, due to a good link quality, the 2M PHY mode provides the most energy-efficient communication. When packets are frequently corrupted, the Coded S8 PHY leads to a lower power consumption, as most corrupted packets can be recovered and do not need to be retransmitted. In contrast to Sect. 3.3, we can see that during a small transition area (an attenuation between -10 dBm and -15 dBm) the 1M PHY mode slightly outperforms the other PHYs in our experiments.

Similar results for P_{EFF} can be observed for a connection interval of 250 ms and 500 ms, but are omitted due to space constraints.

4.3 PHY Mode Selection

The previous sections suggest that choosing the most suitable PHY mode to achieve a maximum throughput or a minimum power consumption requires information about the link quality or the number of link-layer retransmissions of the underlying BLE connection. Unfortunately, off-the-shelf BLE devices do not provide developers with this information per default. However, a developer may use the approach presented by Spörk et al. [18] to statically measure the link quality of a BLE connection during device deployment and manually select the most suitable PHY mode for an application.

Furthermore, an application may use the approach in [18] to dynamically measure the link quality and adapt the used PHY mode at runtime. To this end, a device can use the standardized *L2CAP PHY Update Procedure* of BLE 5 to change the used PHY mode [4].

5 RELATED WORK

Several studies have experimentally investigated the key performance metrics of BLE. While most of these studies have focused on connection-less BLE [14], a few evaluated the connection-based mode [16, 17]. The latter compare the performance of BLE to IEEE 802.15.4 and conclude that BLE is, indeed, more energy efficient at the cost of a shorter communication range. However, these works used BLE v4.0 [16] and v4.1 [17] for their measurements.

A few works [1, 5, 11, 13] investigate BLE 5 and its different PHY modes. Ray and Agarwal [13] describe the capabilities of BLE 5, including its PHY modes, but only theoretically discuss BLE 5's potential in the IoT. Others experimentally investigate BLE 5's different PHY modes for connection-less BLE communication [1, 5, 11]. These works conclude that the used PHY has an effect on the power consumption and throughput when used in BLE advertising.

In this paper, to the best of our knowledge, we provide the first experimental study that investigates the performance of all four PHY modes of BLE 5 when using connection-based BLE. Furthermore, we are the first to provide guidelines for selecting the best PHY mode to achieve specific application requirements.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we provide the first experimental performance analysis of BLE 5's new PHY modes in BLE connections. We highlight the trade-offs of each PHY mode and show how the used PHY mode affects the energy efficiency, communication reliability, and throughput of a connection-based BLE application. We further provide guidelines showing how to select the most suitable PHY mode to sustain specific application requirements, such as a minimum effective power draw or a maximum effective throughput.

Our results can be used to improve the performance of existing BLE applications. Furthermore, BLE applications may use our results and guidelines to dynamically adapt the used PHY at runtime.

ACKNOWLEDGMENTS

This work has been performed within the LEAD project "Dependable Internet of Things in Adverse Environments" funded by Graz University of Technology.

REFERENCES

[1] B. Al Nahas and S. Duquennoy and O. Landsiedel. 2019. Concurrent Transmissions for Multi-Hop Bluetooth 5. In *Proc. of the 16th EWSN Conf.*
[2] A. K. Bhattacharjee et al. 2017. Extending Bluetooth Low Energy PANs to Smart City Scenarios. In *Proc. of the SMARTCOMP Conf.*
[3] Bluetooth SIG. 2016. Bluetooth 5 Brochure: 5 is the Future . https://www.bluetooth.com/~/media/files/marketing/bluetooth5-brochure-web-144ppi.ashx.
[4] Bluetooth SIG. 2018. Bluetooth Core Specification v5.0.
[5] M. Collotta et al. 2018. Bluetooth 5: A concrete step forward toward the IoT. *IEEE Communications Magazine* 7 (2018).
[6] M. Ghamari et al. 2016. A Survey on Wireless Body Area Networks for eHealthcare Systems in Residential Environments. *Sensors* 16. Issue 6.
[7] Mini-Circuits. 2017. RCDAT-8000-90 - Programmable Attenuator.
[8] Monsoon Solutions Inc. 2015. Monsoon Power Monitor.
[9] Nordic Semiconductor. 2017. Bluetooth 5 in Samsung Smartphone. https://blog.nordicsemi.com/getconnected/bluetooth-5-in-samsung-smartphone.
[10] Nordic Semiconductors. 2018. nRF52840 Specifications.
[11] G. Pau et al. 2017. Bluetooth 5 Energy Management through a Fuzzy-PSO Solution for Mobile Devices of Internet of Things. *Energies* 7 (2017).
[12] Raspberry Pi Foundation. 2018. RASPBERRY PI 3 MODEL B - Raspberry Pi.
[13] P. P. Ray and S. Agarwal. 2016. Bluetooth 5 and Internet of Things: Potential and Architecture. In *Proc. of the 2016 Int. IEEE SCOPES Conf.*
[14] T. Renzler, M. Spörk, C.A. Boano, and K. Römer. 2018. Improving the Efficiency and Responsiveness of Smart Objects Using Adaptive BLE Device Discovery. In *Proc. of the 4th ACM MobiHoc SMARTOBJECTS Workshop.*
[15] M. Schuß et al. 2019. JamLab-NG: Benchmarking Low-Power Wireless Protocols under Controllable and Repeatable Wi-Fi Interference. In *Proc. of the EWSN Conf.*
[16] M. Siekkinen et al. 2012. How Low Energy is Bluetooth Low Energy? Comparative Measurements with ZigBee/802.15.4. In *Proc. of the IEEE WCNCW Workshop.*
[17] M. Spörk et al. 2017. BLEach: Exploiting the Full Potential of IPv6 over BLE in Constrained Embedded IoT Devices. In *Proc. of the 15th ACM SenSys Conf.*
[18] M. Spörk, C.A. Boano, and K. Römer. 2019. Increasing the Timeliness of Bluetooth Low Energy in Noisy RF Environments. In *Proc. of the 16th EWSN Conf.*
[19] The Zephyr Project. 2018. Zephyr OS: An RTOS for IoT.

Publication D

M. Spörk, J. Classen, C. A. Boano, M. Hollick, and K. Römer. Improving the Reliability of Bluetooth Low Energy Connections. In *Proceedings of the* 17^{th} *International Conference on Embedded Wireless Systems and Networks (EWSN)*. Junction Publishing, Feb. 2020

DOI: 10.5555/3400306.3400324
Link: `https://dl.acm.org/doi/abs/10.5555/3400306.3400324`

Abstract. To sustain a reliable data exchange, applications based on BLE need to effectively blacklist channels and adapt the physical mode of an active connection at runtime. Although the BLE specification foresees the use of these two mechanisms, their implementation is left up to the radio vendors and has not been studied in detail yet.

This paper fills this gap: we first investigate experimentally how to assess the quality of a BLE connection at runtime using information gathered from the radio. We then show how this information can be used to promptly blacklist poor channels and select a physical mode that sustains a high link-layer reliability while minimizing power consumption. We implement both mechanisms on two popular platforms and show experimentally that they allow to significantly improve the reliability of BLE connections, with a reduction in packet loss by up to $22\,\%$ compared to existing solutions.

My contribution. I am the main author of this paper and developed the idea to improve the link-layer reliability of BLE connections via BLE channel management and BLE PHY mode adaptation. I have designed and implemented the software on the Nordic Semiconductor hardware, as well as, executed all experiments presented in the paper. Jiska Classen was responsible for implementing our improvement on the Raspberry Pi platform and helped me in deriving the most suitable BLE channel management approach. I wrote the vast majority of the paper in collaboration and discussion with my co-authors and presented the paper at EWSN'20. This work was executed in collaboration with TU Darmstadt.

Improving the Reliability of Bluetooth Low Energy Connections

Michael Spörk[†], Jiska Classen[§], Carlo Alberto Boano[†], Matthias Hollick[§], and Kay Römer[†]
[†]Institute of Technical Informatics, Graz University of Technology, Austria
[§]Secure Mobile Networking Lab, Darmstadt University of Technology, Germany
{michael.spoerk, cboano, roemer}@tugraz.at, {jclassen, mhollick}@seemoo.de

Abstract

To sustain a reliable data exchange, applications based on Bluetooth Low Energy (BLE) need to effectively blacklist channels and adapt the physical mode of an active connection at runtime. Although the BLE specification foresees the use of these two mechanisms, their implementation is left up to the radio vendors and has not been studied in detail yet.

This paper fills this gap: we first investigate experimentally how to assess the quality of a BLE connection at runtime using information gathered from the radio. We then show how this information can be used to promptly blacklist poor channels and select a physical mode that sustains a high link-layer reliability while minimizing power consumption. We implement both mechanisms on two popular platforms and show experimentally that they allow to significantly improve the reliability of BLE connections, with a reduction in packet loss by up to 22 % compared to existing solutions.

Categories and Subject Descriptors

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

General Terms

Design, Measurement, Performance, Reliability.

Keywords

BLE, PHY Mode, Adaptive Frequency Hopping.

1 Introduction

BLE is a low-power wireless technology that is increasingly used to create pervasive Internet of Things (IoT) applications, *e.g.*, in the smart health [13], smart city [12], and smart grid [8] domains. Many of these applications are safety critical and impose strict requirements on communication performance, especially with respect to the *reliability* of the data exchange; that is, BLE systems are expected to sustain a minimal packet loss and to ensure short transmission delays.

To increase the reliability of the data exchange, one can make use of information available on the BLE host to adapt BLE's connection parameters at runtime [17, 32]. This helps developers to *cope with packet loss* at the link layer by minimizing its impact on transmission delays. However, it *does not allow to prevent packet loss* at the link layer, which is necessary to maximize the reliability of a BLE connection. The BLE specification [4] foresees two mechanisms to improve the link-layer reliability of BLE connections: adaptive frequency hopping with *channel blacklisting* and *physical (PHY) mode adaptation*. Channel blacklisting allows to exclude poor-performing channels from being used for data exchange. PHY mode adaptation allows to trade receiver sensitivity and error correction capabilities (improving communication range and robustness) for a higher data rate.

The problem. While the primitives for channel blacklisting and PHY mode adaptation are fully embedded in the BLE specification, how these mechanisms should actually be used to improve link-layer reliability is not defined and left to the radio vendors. This results in some BLE platforms not implementing blacklisting at all (*e.g.*, the Nordic Semiconductor nRF52), and other platforms employing blacklisting strategies that were shown to be ineffective in real-world settings (*e.g.*, the Raspberry Pi 3) [32]. Similarly, the lack of guidance on how to use the various PHY modes has triggered several studies investigating their performance [3, 9, 33], but no concrete solution employing them to improve link-layer reliability at runtime has been proposed yet. How to effectively blacklist channels and adapt the PHY mode to minimize link-layer packet loss remains an open question.

The challenges. Link-layer transmissions may fail due to several reasons, such as weak signal strength, multipath fading, or external radio interference [5, 36]. All these factors decrease link-layer reliability, which leads to higher power consumption and transmission delays. To avoid this, one needs to detect these factors and react accordingly, which requires insights about the quality of the used BLE channels.

How to measure link quality? Understanding the quality of the overall BLE connection and individual channels requires to investigate in depth what kind of information can be gathered by the BLE radio at runtime and how this information may be used to blacklist individual channels or adapt the connection's PHY mode.

How to effectively blacklist channels? Based on the information available in the BLE radio, we need to promptly

International Conference on Embedded Wireless Systems and Networks (EWSN) 2020
17–19 February, Lyon, France
ISBN: 978-0-9949886-4-5

detect and blacklist channels with poor quality, while leaving enough channels available for a reliable communication.

How to determine the most suitable PHY mode? Based on the information available in the BLE radio, we need to select the PHY mode that allows to sustain a high link-layer reliability while minimizing the power consumption.

How to design a general solution? In order to be usable by a large fraction of BLE platforms on the market, we need to design and implement effective blacklisting and PHY mode adaptation mechanisms such that both techniques can be used cooperatively on any standard-compliant BLE device that allows link-layer access.

Contributions. We tackle each of these challenges and ultimately improve the link-layer reliability of BLE connections by designing an effective channel blacklisting and PHY mode adaptation mechanism for standard-compliant devices.

To this end, we first perform an extensive experimental campaign to gain a detailed understanding of the information provided by different link quality metrics available in the BLE radio. Such an experimental study fills the gap of existing research and serves as a reference to guide researchers and practitioners working on BLE's link-layer.

Based on this experimental study, we design and evaluate different channel blacklisting mechanisms and observe that passively monitoring the Packet Delivery Ratio (PDR) of individual channels is the most effective way to promptly detect poor channels in order to blacklist them. Our proposed channel blacklisting mechanism is hardware-independent and can be used in any standard-compliant BLE device that allows access to the BLE link layer.

We also design a PHY mode adaptation mechanism that monitors recent Signal-to-Noise Ratio (SNR) measurements to dynamically change the used PHY mode when necessary. The proposed mechanism allows to sustain a specified minimum link-layer reliability while limiting power consumption; all of this using standardized BLE primitives, such that it can be used by any device supporting multiple PHYs.

We implement both mechanisms on two popular hardware platforms: the Nordic Semiconductor nRF52 and the Raspberry Pi 3 (Pi3). As we only make use of standardized BLE primitives, our approach can be easily ported to other standard-compliant BLE platforms. Finally, we experimentally show that our mechanisms on the nRF52 cooperatively improve link-layer reliability by up to 22 %. Using our improvements, the nRF52 sustains a link-layer reliability of over 99 % in all experiments. Our improvements on the Pi3[1] increase the link-layer reliability of BLE connections by up to 10 % without incurring additional power consumption.

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

- We experimentally study the link quality metrics that can be gathered by the BLE radio at runtime and discuss the insights that each metric provides (Sect. 3).
- We design an effective channel blacklisting mechanism that uses recent PDR measurements to detect and blacklist BLE channels with poor link quality (Sect. 4).

[1] Our improvements on the Pi3 are openly available at `https://github.com/seemoo-lab/internalblue/tree/master/examples`

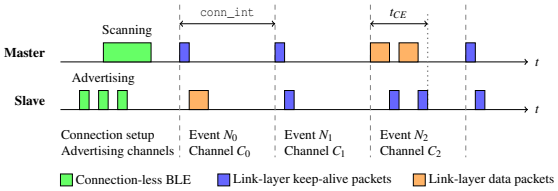


Figure 1. BLE connection between a master and a slave.

- We design an effective PHY mode adaptation mechanism using SNR readings that can sustain a given reliability while minimizing power consumption (Sect. 5).
- We implement the proposed mechanisms on two popular hardware platforms (Sect. 6) and evaluate their performance in different real-world scenarios (Sect. 7).

After summarizing related work in Sect. 8, we conclude this paper in Sect. 9 along with a discussion of future work.

2 Background on BLE Connections

BLE provides two communication modes: a *connectionless* and a *connection-based* mode. While the connectionless mode makes use of three advertisement channels to broadcast short data packets, the connection-based mode supports bidirectional data transfer. To enter the connection-based mode two devices use connection-less primitives to establish a BLE connection. In this connection, one device acts as master and the other as slave. Data exchange happens only during *connection events* (N_0 ... N_i), as shown in Fig. 1.

The *connection interval* (`conn_int`) specifies the time between the start of two consecutive connection events. During a connection event, master and slave exchange link-layer packets until both devices have no more data to send or the *maximum connection event length* (t_{CE}) is reached. These link-layer packets either carry application data (orange) or an empty payload to keep the connection alive (blue). Every connection event starts with a transmission by the master, to which the slave responds. If no data needs to be sent, only mandatory keep-alive packets are exchanged. The last link-layer packet in a connection event is always sent from slave to master, after which the connection event is closed and communication is resumed at the next connection event.

In the example shown in Fig. 1, the master starts connection event N_0 by sending a keep-alive packet to the slave, and the slave responds with a link-layer data packet carrying application data. In connection event N_1, master and slave have no data to send and therefore only exchange the mandatory keep-alive packets. During connection event N_2, the master transmits data to the slave. Because this data exceeds the maximum length of link-layer data packets, the master splits the data into two link-layer packets that are both acknowledged by the slave with a link-layer keep-alive packet.

AFH algorithm. At the beginning of every connection event, one out of 37 data channels is selected by BLE's adaptive frequency hopping (AFH) algorithm.[2] A new channel is chosen for every connection event and is used for all transmissions taking place during this event.

[2] With the release of BLE 5, BLE devices can use one of two possible AFH algorithms. As we show in Sect. 7, the employed AFH algorithm does not have significant impact on the link-layer reliability of a BLE connection.

Data channel selection. All 37 data channels are located in the license-free 2.4 GHz Industrial, Scientific, and Medical (ISM) band, which makes them likely to experience interference by other radio technologies using the same frequencies, *e.g.*, Wi-Fi, classic Bluetooth, or IEEE 802.15.4. Such interference, as well as multipath fading, may cause packet loss that decreases the link-layer reliability of BLE.

To mitigate the effects of link-layer packet loss, BLE radios may *blacklist* channels with poor quality by updating the *channel map* (C_{map}) of the connection at runtime. A connection's C_{map} specifies which data channels may be selected by the AFH algorithm; a channel disabled in the C_{map} will not be used for communication until being *whitelisted* (re-enabled) again. The BLE specification defines standardized commands that a master can use to update the C_{map} of an active connection; a slave is not allowed to change the C_{map}. However, when and how a master updates the C_{map} is not defined by the BLE specification. This leaves it up to developers to implement an effective blacklisting strategy, often leading to ineffective solutions on existing systems [32].

Link-layer ACK and flow control. To provide a reliable data exchange, the BLE link layer automatically handles packet *acknowledgment* (ACK) and flow control. This is achieved using a 1-bit *Sequence Number (SN)* and a 1-bit *Next Expected Sequence Number (NESN)* in each link-layer header. A link-layer packet is only successfully acknowledged when a BLE radio receives a NESN that is not equal to the SN of the transmitted packet. The link-layer packet is automatically retransmitted until a valid ACK is received.

PHY modes. Devices supporting BLE version 5 and above are able to choose one out of four physical (PHY) modes: the 1M, the 2M, the Coded S2, and the Coded S8 PHY [4].

The *1M PHY*, where the M stands for Megasymbols/s (Msym/s), is the original mode of BLE and is the only available PHY on BLE devices with a version below 5. The 1M PHY uses a physical modulation of 1 Msym/s, no symbol coding, and no Forward Error Correction (FEC). Similarly, the *2M PHY* mode uses no symbol coding and no FEC; however, it uses a physical modulation of 2 Msym/s leading to twice the data throughput compared to the 1M PHY [33].

The *Coded S2* or the *Coded S8 PHY* modes use symbol coding and FEC to reconstruct flipped bits in received packets, leading to a more robust communication [33]. Like the 1M PHY, both Coded PHYs use a physical modulation of 1 Msym/s. The Coded S2 PHY uses a symbol coding of 2, resulting in a maximum physical data rate of 500 kb/s. Likewise, the Coded S8 PHY uses a symbol coding of 8, resulting in a maximum physical data rate of 125 kb/s.

3 Experimental Study of BLE Reliability

We investigate next how to estimate the link quality of BLE links based on information gathered by the link layer on standard BLE radios. We first list available link-layer metrics in Sect. 3.1 and show their behavior in Sect. 3.2. We further evaluate the impact of different PHY modes on BLE reliability in Sect. 3.3 and discuss our findings in Sect. 3.4.

3.1 BLE Link Quality Metrics

We estimate the link quality of BLE data channels on master devices, which has several benefits that originate in the BLE specification [4]. First, the BLE master is usually less energy constrained (it is constantly powered or frequently charged) and can afford to probe the RF environment (*e.g.*, to measure the noise floor). Second, the master receives feedback on link-layer transmission within the same connection event, while on the slave this feedback is delayed (see Sect. 2). Third, only the master is allowed to black- and whitelist data channels used by the BLE connection.

To be compliant with the BLE specification [4], we *passively* estimate the link quality based on metrics available in the BLE link layer. Using an active approach that exchanges link-layer probe packets as done in the context of IEEE 802.15.4 [14] is not suitable for our approach, because sending additional probe packets is not compliant with the BLE specification. Furthermore, probe packets would introduce additional radio time and increase power consumption.

We focus on link-layer metrics that are hardware-agnostic and available on standard BLE radios allowing access to the link layer. We hence consider the following metrics:

Noise floor. We measure the noise floor of an individual data channel by probing the channel when the BLE radio does not exchange packets. This provides us with information on any nearby technology using the 2.4 GHz ISM band.

Signal-to-Noise Ratio (SNR). We calculate the SNR of every successfully received link-layer packet, by measuring a packet's Received Signal Strength Indicator (RSSI) and subtracting the current noise floor on the used data channel.

Packet Delivery Ratio (PDR). To measure the PDR, we adapt the work in [11] (based on IEEE 802.15.4) to be used in BLE connections. Because we are unable to arbitrarily probe individual channels, we use existing link-layer header fields to calculate the PDR of link-layer exchanges. The PDR measures the round-trip reliability of individual link-layer transmissions issued by a master and is computed as:

$$PDR = \frac{\#ACK(S \rightarrow M)}{\#TX(M \rightarrow S)}, \quad (1)$$

where $\#TX(M \rightarrow S)$ is the number of issued link-layer transmissions from master to slave and $\#ACK(S \rightarrow M)$ is the number of received valid link-layer acknowledgments. A link-layer acknowledgment from the slave is considered valid if it carries a valid CRC checksum and an updated NESN. As a result, every *individual* link-layer transmission can be either successful ($PDR = 100\%$) or unsuccessful ($PDR = 0\%$).

3.2 Measuring BLE Link Quality

We study the metrics from Sect. 3.1 in multiple scenarios to gain an understanding about the insights they provide.

Experimental setup. To have full control over the RF environment, we perform all experiments in a wireless testbed located in a laboratory. During all our tests, the lab is vacant and all Wi-Fi access points in proximity are deactivated, to limit the impact of Wi-Fi activity on our results.

We use two *Nordic Semiconductor nRF52840 DK (nRF52)* [24] exchanging data over a BLE connection: one device acts as master and the other one as slave. While the master uses its on-board PCB antenna for communication, we mechanically remove the PCB antenna of the slave to eliminate undesirable overlapping antenna effects and connect a programmable attenuator [21] as well as a 2.4 GHz

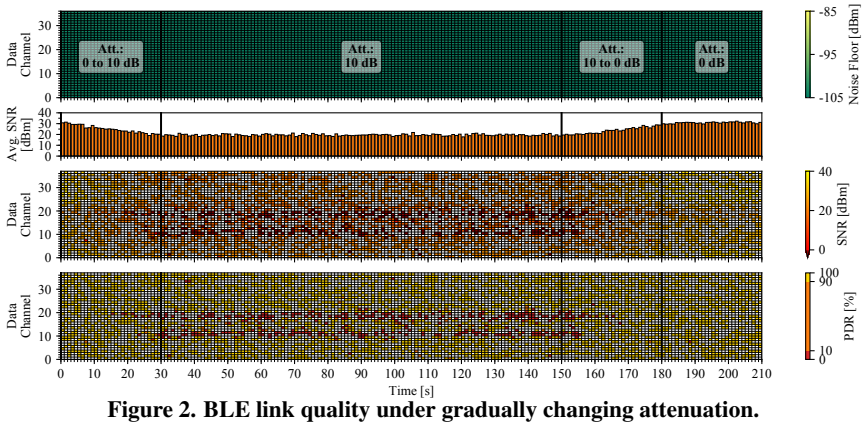


Figure 2. BLE link quality under gradually changing attenuation.

antenna (with a gain of 3 dB) to the external antenna connector of the nRF52. This setup allows us to manually reduce the power of the signal that is sent and received by the slave.

For this experimental campaign, master and slave have free line of sight and a distance of approx. 10 m. The master initiates a BLE connection with a connection interval of 50 ms and a slave latency of 0 and subscribes to a BLE GATT attribute on the slave. The slave issues a GATT indication on this attribute (with a link-layer length of 27 B) every 500 ms.

Both master and slave run the Zephyr OS [35], which comes with a standard-compliant BLE stack supporting BLE version 5. We adapt the link-layer implementation of the master to log the PDR and SNR of every link-layer transmission. The master also measures and logs the noise floor of each data channel after every connection event.

For these experiments, the connection uses the Channel Selection Algorithm (CSA) #2 of BLE 5 and the 1M PHY mode. The master does not use blacklisting, which is the default behavior of an nRF52 device running the Zephyr OS.

Methodology. We measure the behavior of the link quality metrics under changing antenna attenuation (Sect. 3.2.1), Bluetooth (Sect. 3.2.2), and Wi-Fi interference (Sect. 3.2.3).

The noise floor shown in Figs. 2 to 4 is the maximum noise floor recorded on each channel within every second. Likewise, the SNR plots in Figs. 2 to 4 show the average SNR within a second on every channel. When the master does not receive a link-layer ACK from the slave (and hence no RSSI value), the SNR of this unsuccessful packet exchange is discarded and these exchanges are marked in brown. In addition to the SNR per channel, we calculate the average SNR (*Avg. SNR*) across all used BLE data channels.

The PDR of individual link-layer transmissions is calculated as explained in Sect. 3.1. Figs. 2 to 4 show the average PDR within a second per channel. We classify each channels into *good*, *intermediate*, and *poor* based on its PDR, adapting the approach proposed by Srinivasan et al. [34]. Channels with a PDR $< 10\%$ are classified as poor; channels with a PDR between 10% and 90% are classified as intermediate; channels sustaining a PDR $\geq 90\%$ are classified as good.

Please note that, in Figs. 2 to 4, when a data channel is not used within a second, its PDR and SNR are marked in white.

3.2.1 Changing Antenna Attenuation

First, we investigate how the link quality of the BLE data channels is affected when the wireless signal is attenuated, *e.g.*, due to an increasing communication distance or obstacles blocking the line of sight. Using the programmable attenuator, we change the slave's antenna attenuation over time, mimicking a slave that moves away from the master.

Fig. 2 shows the measured noise floor, the average SNR of all used data channels, the SNR per data channel, and the PDR per data channel under changing attenuation over time. The effective attenuation of the BLE master starts at 0 dB (3 dB from the antenna gain minus a programmed attenuation of 3 dB) and, from time 0 to 30 s, the effective attenuation increases linearly to 10 dB. The attenuation stays at 10 dB for 120 s, before gradually going back to an effective attenuation of 0 dB, where it stays until the end of the experiment. We can see that the Avg. SNR reflects this change in attenuation and that the PDR of some data channels decreases significantly when a high attenuation is used.

The data in Fig. 2 shows that the noise floor measurements are not able to detect any of these link quality problems. As expected, the Avg. SNR captures the change in signal strength and the SNR measurement per channel detects when the master is not able to successfully receive a link-layer packet. SNR measurements, however, do not detect when a valid link-layer packet was received, but its CRC or NESN indicate a bad packet transmission. Only the PDR measurements are successfully able to capture all link-layer packet loss caused by a high antenna attenuation.

3.2.2 Classic Bluetooth Interference

Next, we measure BLE's link quality under external interference caused by co-located classic Bluetooth devices.

Like BLE, classic Bluetooth uses the 2.4 GHz ISM band and employs frequency hopping. However, classic Bluetooth uses a different modulation scheme with 1 MHz wide channels and is optimized for data throughput. In our experiments, we keep the effective attenuation at 0 dB and use two pairs of Pi3 sending Bluetooth RFCOMM packets of 1000 B length every 11.034 ms. This results in two classic Bluetooth connections, each exchanging packets at 725 kbit/s.

Fig. 3 shows the various link quality metrics in the presence of Bluetooth interference, generated for roughly 120 s,

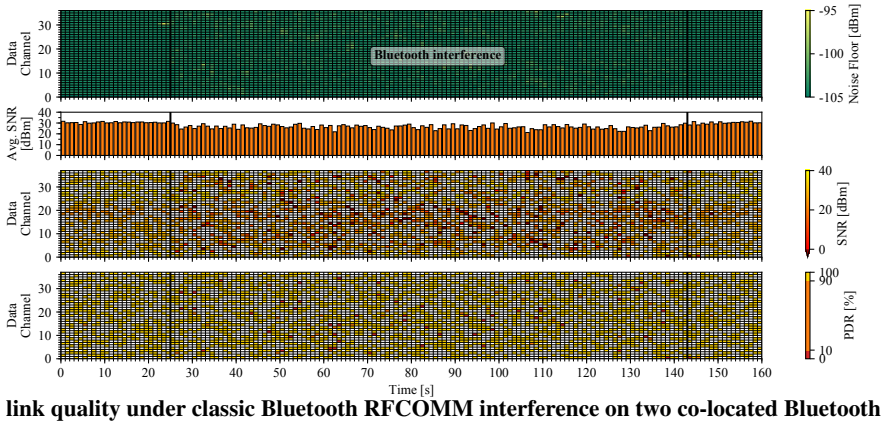


Figure 3. BLE link quality under classic Bluetooth RFCOMM interference on two co-located Bluetooth connections.

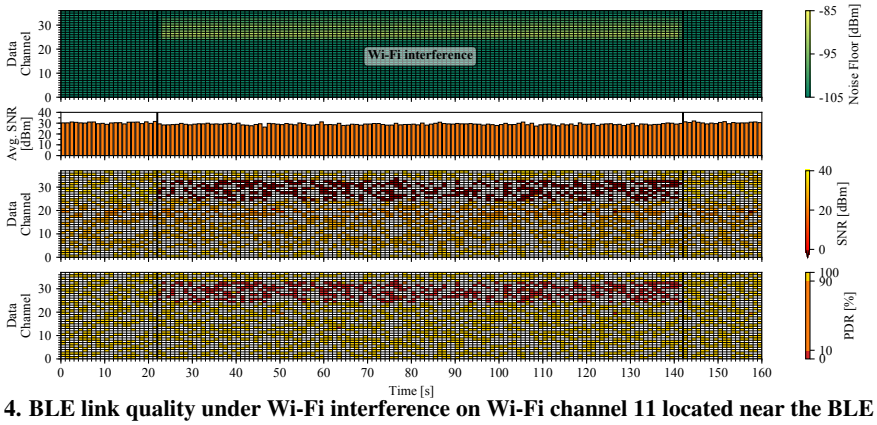


Figure 4. BLE link quality under Wi-Fi interference on Wi-Fi channel 11 located near the BLE slave.

starting 30 s from the beginning of the experiment. The data shows that classic Bluetooth interference decreases the reliability of the BLE connection on all data channels.

In this scenario, the noise floor measurements are barely able to detect the co-located radio communication. The Avg. SNR and the SNR per channel are able to detect the Bluetooth interference, but SNR readings may actually underestimate the link quality in this case, as several SNR values are very low ($SNR < 5$ dBm), even though a successful packet exchange is possible. Similar to the previous scenario, only the PDR is able to accurately capture link-layer packet loss caused by co-located Bluetooth RFCOMM interference.

3.2.3 Wi-Fi Interference

We measure next the link quality of the data channels in the presence of radio interference generated by co-located Wi-Fi devices. To generate the Wi-Fi interference, we use a Pi3 in our testbed, located near the BLE slave and run JamLab-NG [28] to generate Wi-Fi packets on Wi-Fi channel 11 with a length of 1500 B every 10 ms and a transmission power of 30 mW. In this experiment, we keep the effective antenna attenuation constant at 0 dB.

Fig. 4 shows the link quality metrics in the presence of Wi-Fi interference, generated for roughly 120 s, starting 30 s from the beginning of the experiment: overall, the Wi-Fi interference significantly decreases the link-layer reliability.

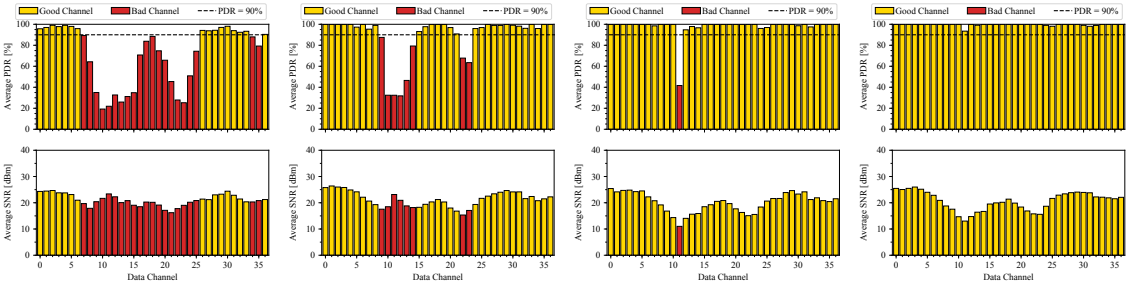
In this scenario, the noise floor measurements detect the Wi-Fi interference. Similar to Sect. 3.2.1, the SNR measurements detect when the master is not able to successfully receive link-layer packets, but do not detect failures due to invalid CRC or missing NESN updates. PDR measurements accurately detect link-layer errors due to Wi-Fi interference.

3.3 Using Different PHY Modes

We investigate how the used PHY mode of the BLE connection affects reliability. In contrast to existing work [33], we evaluate how individual data channels behave under various link conditions when using the different PHY modes.

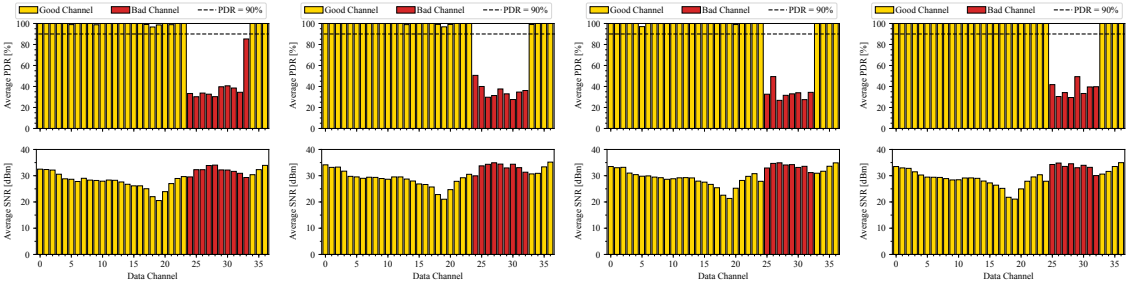
Antenna attenuation. We repeat the experiment from Sect. 3.2.1 with all four PHY modes of BLE 5. Fig. 5 shows the average PDR and average SNR per channel for the different PHYs measured while applying an effective attenuation of 10 dB. Data channels with an average PDR $< 90\,\%$ are classified as *bad* (marked in red), while channels with an average PDR $\geq 90\,\%$ are classified as *good* (marked in yellow).

In this experimental scenario, the used PHY has a significant effect on the overall reliability of the BLE connection and the number of data channels classified as good. When using the fastest 2M PHY, 22 of the available data channels are bad, compared to the 8 bad channels when using the 1M PHY. Due to their use of FEC and symbol coding, the Coded



(a) 2M PHY Mode. (b) 1M PHY Mode. (c) Coded PHY (S2) Mode. (d) Coded PHY (S8) Mode.

Figure 5. Average PDR and SNR of all data channels using different PHYs and an antenna attenuation of 10 dB. *We see that the used PHY mode of the BLE connection significantly affects the overall link-layer reliability in this scenario.*



(a) 2M PHY Mode. (b) 1M PHY Mode. (c) Coded PHY (S2) Mode. (d) Coded PHY (S8) Mode.

Figure 6. Average PDR and SNR of all data channels using different PHYs under Wi-Fi interference on channel 11. *We see that the used PHY mode of the BLE connection does not significantly affect the overall link-layer reliability in this scenario.*

S2 and S8 PHYs are able to sustain a high PDR even for channels that have a low average SNR. Therefore, switching to a more robust PHY mode when experiencing a low signal strength significantly increases link-layer reliability.

Radio interference. Next, we repeat the experiment from Sect. 3.2.3 with all PHY modes of BLE 5. Fig. 6 shows the average PDR and average SNR per channel measured when Wi-Fi interference was generated near the BLE slave.

Unlike the previous experiment, the used PHY mode does not significantly improve the reliability of the BLE connection and the number of good data channels under external interference. All four PHYs experience similar link quality problems on the data channels affected by the co-located Wi-Fi interference on Wi-Fi channel 11. The 2M PHY provides an overall PDR of 83 %, while the Coded S8 PHY leads to an overall PDR of 86 %. Switching the PHY from the 2M PHY to the most robust Coded S8 PHY would only increase the PDR of the BLE connection by 3 %. If we would blacklist all poor channels in this scenario, instead, the BLE connection would have an overall PDR $> 99\,\%$ for all PHYs. Note that, in contrast to the data shown in Fig. 5, link-layer packet loss caused by radio interference does not cause the average SNR on the affected data channels to drop.

3.4 Lessons Learned

Based on our results, we draw the following conclusions.

Noise floor. Noise floor measurements of data channels successfully detect link-layer loss caused by external radio interference (Sect. 3.2.2 and 3.2.3). However, the noise floor fails to capture link-layer loss caused by a weak signal strength, *e.g.*, due to a large communication distance (Sect. 3.2.1).

SNR per data channel. SNR measurements detect link-layer loss on individual channels, when link-layer packets are missing and therefore no SNR can be calculated. SNR readings, however, miss link problems indicated by an invalid CRC or missing update of NESN. Furthermore, classifying channels based on recent SNR readings may result in an inaccurate classification, as some successful link-layer exchanges may have an SNR below 5 dBm that would indicate a problem based on the measurements shown in Fig. 5.

Average SNR. The Avg. SNR across all used channels accurately captures the signal strength of the BLE connection. The Avg. SNR can be used to detect when a BLE connection is experiencing link-layer errors due to weak signal, *e.g.*, caused by a long communication distance. Furthermore, the Avg. SNR is not significantly affected by external radio interference, as shown in Fig. 3 and Fig. 4. Therefore, the SNR can be used to detect when the connection's signal strength is the problem for link-layer loss, as we show in Sect. 5.

Packet Delivery Ratio. As expected, PDR readings accurately detect any link-layer packet loss across all of our experiments. This makes PDR the most suitable metrics to detect and blacklist poor data channels, as we show next.

4 BLE Channel Blacklisting

Using our findings from Sect. 3, we design an effective channel blacklisting mechanism that *passively* monitors the individual data channels and classifies them into *good* and *bad*. Good channels experience zero or few link-layer er-

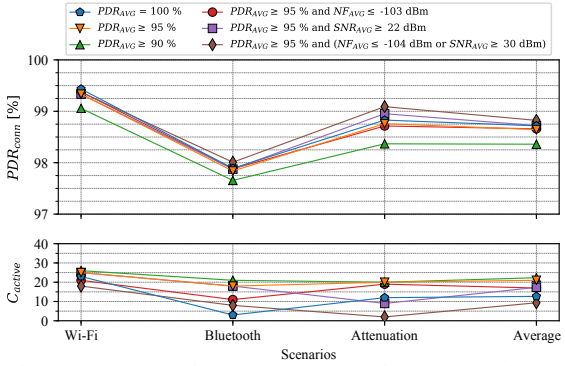


Figure 7. PDR of the BLE connection (PDR_{conn}) and number of active channels (C_{active}) at the end of each experiment for different filtering mechanisms. *Overall, the $PDR_{AVG} \geq 95\%$ approach provides the best trade-off between reliability (PDR_{conn}) and data channel usage (C_{active}).*

rors, while bad channels have an insufficient link quality and should not be used for communication. Similar to the work in [11], our goal is not to exactly estimate the link quality of a channel, but to detect when a channel is experiencing problems (*e.g.*, due to radio interference or weak signal strength) in order to blacklist it. To this end, we study the best way to detect bad channels at runtime (Sect. 4.1), when and how to blacklist channels (Sect. 4.2), as well as when to whitelist the channels of an active BLE connection (Sect. 4.3).

4.1 Detecting Bad Channels at Runtime

To find the most suitable way to detect bad data channels, we use the experimental traces from Sect. 3.2 and investigate the performance of different channel classification approaches. For every scenario, we simulate the behavior of one channel classification approach and calculate the overall PDR of the BLE connection (PDR_{conn}) and the number of active channels (C_{active}) at the end of each experiment.

Therefore, we step through each experimental trace and increase $\#TX(M \rightarrow S)$ by one for every issued link-layer packet. If the transmission was successful ($PDR = 100\%$), we also increase $\#ACK(S \rightarrow M)$ by one. With $\#TX(M \rightarrow S)$ and $\#ACK(S \rightarrow M)$, we calculate PDR_{conn} using Eq. 1. While stepping through an individual trace, we simulate the behavior of different channel classification approaches. Whenever the used classification approach detects a bad data channel, we mark it as blacklisted and do not count any subsequent link-layer exchanges on this channel. This simulates the actual channel blacklisting behavior that we implement on standard BLE devices, which is described in Sect. 6.

Fig. 7 shows PDR_{conn} and C_{active} of the six investigated channel classification approaches in three different experimental scenarios. All investigated approaches calculate the moving average (PDR_{AVG}) of recent PDR values using a window length W_{PDR}, as only the PDR detects all link-layer failures independent of their cause (see Sect. 3).

The approach $PDR_{AVG} = 100\%$ aggressively blacklists individual data channels on their first link-layer packet loss. The $PDR_{AVG} \geq 95\%$ approach uses a $W_{PDR} = 20$ and blacklists a channel when its PDR_{AVG} drops below 95 %. Similarly, $PDR_{AVG} \geq 90\%$ uses a $W_{PDR} = 10$ and a threshold of 90 %. The chosen W_{PDR} for every approach is the minimum window that still allows to measure the PDR_{AVG} in the necessary resolution. To measure the PDR of a channel with a resolution of 5%, which is necessary for our $PDR_{AVG} \geq 95\%$ approach, we need at least a $W_{PDR} = 20$.

Out of these classification approaches, $PDR_{AVG} \geq 95\%$ is able to sustain an average $PDR_{conn} = 98.6\,\%$ and a minimum $PDR_{conn} = 97.8\,\%$ in all three scenarios. Although the $PDR_{AVG} = 100\%$ provides a slightly higher average $PDR_{conn} = 98.7\,\%$, its aggressive behavior leads to a significantly lower C_{active}, which may result in a terminated BLE connection when sudden radio interference appears.

In addition to the first three approaches using only PDR measurements, we investigate if additional information about the average noise floor (NF_{AVG}) or SNR (SNR_{AVG}) improves channel blacklisting. We use the $PDR_{AVG} \geq 95\%$ approach and combine it with NF_{AVG} and SNR_{AVG}, leading to three additional classification approaches; all of them use a $W_{PDR} = 20$ for filtering PDR_{AVG}, NF_{AVG}, and SNR_{AVG}. The individual configurations for these approaches were chosen by running every scenario with every threshold combination and selecting the best combination for the average case.

Overall, the $PDR_{AVG} \geq 95\%$ approach is the most suitable channel classification approach for blacklisting. With its ability to sustain a PDR_{AVG} above 97.8 % while providing an average C_{active} of 21 channels, $PDR_{AVG} \geq 95\%$ provides the most suitable trade-off between link-layer reliability and number of active channels. Based on our experiments in Sect. 3, we see that sudden Wi-Fi interference may affect up to 10 subsequent BLE data channels. Sustaining a $C_{active} > 10$, as the $PDR_{AVG} \geq 95\%$ is able to do, mitigates BLE connection loss due to sudden co-located Wi-Fi traffic.

4.2 Blacklisting BLE Data Channels

Next, we discuss the necessary steps between detecting a bad channel and excluding it from further communication.

Issuing a channel map update. As detailed in Sect. 2, only the BLE master can update the used channel map (C_{map}) of a BLE connection by sending an `LL_CHANNEL_MAP_IND` request to the slave. This request carries a 37-bit data C_{map} that indicates if a channel is used in the connection or not. If the corresponding bit of a data channel is set, the channel is actively used for communication, otherwise it is not and is hence *blacklisted*. A blacklisted channel stays inactive until another `LL_CHANNEL_MAP_IND` request *whitelists* (re-enables) the channel. A slave receiving this request cannot negotiate and needs to adhere to the received information.

In our approach, we update C_{map} as soon as we detect that a data channel is bad. If the master has recently issued a C_{map} update that has not yet been acknowledged by the slave, the master waits for this ACK and then immediately issues a new C_{map} update. This approach may create a separate `LL_CHANNEL_MAP_IND` request for every blacklisted data channel, resulting in additional radio time. However, sending multiple `LL_CHANNEL_MAP_IND` requests does not significantly increase the power consumption, as shown in Sect. 7.

Mandatory update delay. According to the BLE specification [4], a new C_{map} only takes effect after a mandatory

delay of at least 6 connection events. This means when a BLE master detects a bad channel and therefore issues an `LL_CHANNEL_MAP_IND` request at connection event N, the new C_{map} is used starting from connection event $N+6$. Hence, a channel already marked as blacklisted may be used in another connection event, before being actually disabled. To maintain interoperability with standard-compliant BLE devices, however, we adhere to this mandatory delay.

Clearing information about blacklisted channels. As soon as a channel is blacklisted, we are not able to estimate its link quality using our approach. Therefore, any link-layer information of blacklisted channels needs to be cleared, as it may have expired. Only when a data channel is whitelisted, we collect fresh information to accurately estimate its quality.

4.3 Whitelisting BLE Data Channels

As mentioned above, when a channel is blacklisted all of its link quality information may be outdated. Hence, we cannot observe when a blacklisted channel turns good in order to whitelist it. One possible solution for this is to exchange additional probes on blacklisted channels to measure their PDR. This, however, is not compliant to the BLE specification and would introduce an unnecessary overhead. Another approach is to whitelist data channels on a time basis, *i.e.*, re-enable blacklisted channels several seconds after being blacklisted. Finding a suitable timeout, however, largely depends on the actual cause of current link-layer packet loss.

In this work, we trigger a whitelisting whenever the number of active channels in the current data channel map drops below the minimum number of data channels (C_{min}).[3] In particular, we re-enable all 37 data channels and probe them using regular connection events to measure their PDR_{AVG}: this allows us to get the most accurate link quality estimation of the usable channels. Using this approach, whenever the master blacklists a bad channel, it first checks the number of used data channels: if the number of used channels drops below C_{min}, the master performs the following procedure.

Updating the connection interval. During whitelisting, we re-enable data channels with unknown link quality, which can cause link-layer errors leading to high latencies [32]. To ensure a reliable data exchange during whitelisting, we adapt the used BLE connection interval to a faster setting before re-enabling any data channel. During whilelisting, we temporarily overprovision (O_{WL}) the BLE connection by:

$$O_{WL} = \lceil C_{max}/C_{min} \rceil, \tag{2}$$

where C_{min} is the number of channels used before initiating a whitelisting and C_{max} is the number of channels that are probed during whitelisting. Since we enable all 37 data channels during whitelisting, $C_{max} = 37$ for our approach.

The faster connection interval ($conn\_int_{WL}$) temporarily used during whitelisting is calculated as:

$$conn\_int_{WL} = conn\_int / O_{WL}, \tag{3}$$

where $conn\_int$ is the connection interval used before whitelisting and O_{WL} is the necessary overprovisioning calculated by Eq. 2. For example, if we use a $C_{min} = 10$ and a $C_{max} = 37$, we need to change the connection interval to be $O_{WL} = 4$ times faster to mitigate the effects of potential link-layer errors on BLE transmission delays during whitelisting.

Similar to the channel map update, updating the connection interval requires a delay of at least 6 connection events between issuing and using the new connection interval [4].

Probing data channels. After the temporary $conn\_int_{WL}$ has been set, we issue a data channel map update re-enabling all 37 BLE data channels to probe their link quality. During this probing phase, we use ordinary BLE connection events to measure the PDR_{AVG} and link quality of the individual data channels, as discussed in Sect. 4.1. In this phase, however, we temporarily disable channel blacklisting to get the most accurate estimation of each channel's link quality.

This probing phase lasts for t_{probe}, in which we probe every data channel $S_{channel}$ times. t_{probe} is calculated as:

$$t_{probe} = S_{channel} \cdot C_{max} \cdot conn\_int_{WL}, \tag{4}$$

where $S_{channel}$ is the number of samples per channel, C_{max} is the number of BLE data channels, and $conn\_int_{WL}$ is the connection interval used during probing, calculated with Eq. 3.

After the probing phase has ended, we blacklist any poor channels and revert back to the original $conn\_int$, resuming communication with the original BLE connection parameters and a new channel map with only reliable data channels.

5 BLE PHY Mode Adaptation

We design a PHY mode adaptation mechanism allowing BLE devices to sustain a specified link-layer reliability while limiting unnecessary power consumption. Specifically, the proposed adaptation mechanism *passively* monitors recent SNR measurements to detect when a change is necessary.

As discussed in Sect. 3, the average SNR over used data channels accurately captures the BLE signal strength and is not significantly affected by external radio interference. As a result, our proposed PHY mode adaptation mechanism is independent from the blacklisting mechanism presented in Sect. 4, making both mechanisms easily portable to other hardware platforms. In case a device does not support different BLE PHY modes, such as the Raspberry Pi 3, we can use our channel blacklisting mechanism to improve reliability. On devices supporting different PHY modes, both mechanisms work together in parallel to improve reliability while minimizing power consumption, as we show in Sect. 7.

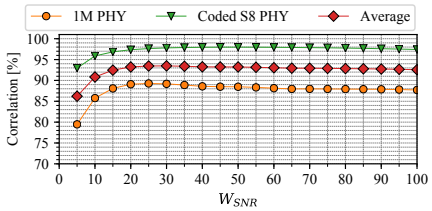
As the BLE specification allows slaves to change the used PHY mode, any BLE device may make use of our proposed PHY mode adaptation to increase link-layer reliability.

Next, we discuss how to filter recent SNR readings (Sect. 5.1), choose a suitable PHY mode (Sect. 5.2), and adapt the used PHY mode of a BLE connection (Sect. 5.3).

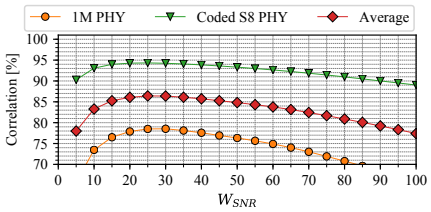
5.1 Filtering SNR Readings

We use a moving average filter on recent SNR values on all used data channels to predict the SNR of future link-layer exchanges. To find a suitable window length W_{SNR} for our SNR filter, we reuse the experimental setup employed to derive Fig. 2 for different antenna attenuation settings and PHY modes. We step through these traces and investigate which W_{SNR} leads to the best predictions of the average SNR of subsequent link-layer exchanges. For this evaluation, we choose a prediction window of 100 future link-layer exchanges.

[3] Since BLE 5, a slave can change the minimum number of data channels by sending an `LL_MIN_USED_CHANNELS_IND` request to the master. Older BLE devices do not have this possibility and only mandate that $C_{min} \geq 2$.

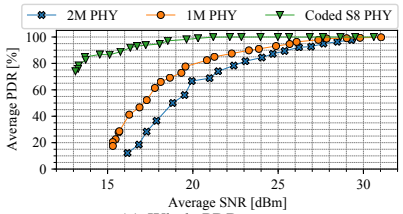


(a) Attenuation = 10 dB.

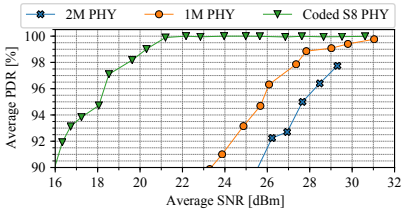


(b) Attenuation = 20 dB.

Figure 8. Correlation of the predicted and actual future SNR for different observation window lengths (W_{SNR}). *The data suggest that using a W_{SNR} of 25 provides the best prediction of future SNR values across both scenarios.*



(a) Whole PDR range.



(b) PDR range between 90 % and 100 %.

Figure 9. Relationship between average PDR and the average SNR of a BLE connection for different PHY modes.

Fig. 8 shows the correlation between predicted average SNR, filtered with different W_{SNR}, and the actual future average SNR. The data shows that using a $W_{SNR} = 25$ to filter recent link-layer exchanges provides the most accurate estimation of future SNR across the investigated traces.

5.2 Choosing a Suitable PHY Mode

Using the filtered SNR readings, a device can choose the most suitable PHY mode that sustains a specified minimum link-layer reliability (PDR_{min}) while limiting unnecessary power consumption. To find the relationship between average SNR and PDR, we re-use the data captured in Sect. 5.1 for different attenuation settings. We process each trace and calculate the average PDR and the average SNR during the time of constant attenuation (time 30 to 150 s in Fig. 2).

Fig. 9 shows the relationship between average SNR and PDR for three different PHY modes. We only investigate the Coded S8 PHY because the symbol coding used by the Coded PHY (either S2 or S8) is decided by every individual BLE device and cannot be negotiated at runtime. Therefore, we fix symbol coding for the Coded PHY mode to S8 on our devices, as this provides the highest reliability (see Fig. 5).

As expected, the Coded S8 PHY mode sustains the highest average PDR for a given SNR. For example, while the Coded S8 PHY provides an average PDR above 98.5 % for an SNR of 20 dBm, 1M and the 2M PHY only sustain an average PDR of approx. 82 % and 66 %, respectively. This increased reliability, however, comes with the cost of additional power consumption caused by the longer radio times of the Coded S8 PHY. Using the setup from Sect. 3.2, a slave using the Coded S8 PHY has an average power consumption of 581.79 μA, while the same slave using the 1M or the 2M PHY consumes 407.93 μA and 397.07 μA, respectively.

Our measurements show that a slave using the 1M PHY consumes only slightly more power (approx. +2.73 %) compared to using the 2M PHY, but the 1M PHY provides a significantly higher link-layer reliability across all our tests. We therefore argue that using the 2M PHY does not pay off in application scenarios where devices need to sustain a given PDR_{min} on a constrained energy budget and data throughput is not an issue. In such applications, one should make use of the data shown in Fig. 9 to decide between the Coded S8 and the 1M PHY mode based on the average experienced SNR.

5.3 Adapting the Used PHY Mode

Using a specified PDR_{min}, our PHY mode adaptation selects a SNR threshold (SNR_{PHY}) based on the data in Fig. 9. When the average SNR is $\geq SNR_{PHY}$, our mechanism uses the 1M PHY to conserve energy. When the average SNR is below SNR_{PHY}, our mechanism chooses the Coded S8 PHY to sustain PDR_{min}. If a device operates at approximately SNR_{PHY}, it may continuously switch between the two PHY modes, which may lead to additional power consumption or a PDR below PDR_{min}. To mitigate such behavior, we switch to the Coded S8 PHY when the average SNR drops below SNR_{PHY}, but only switch to the 1M PHY, when the average SNR $\geq SNR_{PHY} + SNR_{offset}$. To find a suitable SNR_{offset} for our PHY adaptation mechanism, we use the traces collected in Sect. 5.2 and investigate the number of PHY mode adaptations (N_{adapt}) for different SNR_{offset} and attenuations when using the filtering mechanism from Sect. 5.1. Fig. 10 shows that even a SNR_{offset} = 1 dBm mitigates unnecessary PHY mode adaptations across all our tests.

To adapt the used PHY mode, we use the standardized *PHY update procedure* defined by the BLE specification [4]. Similar to the channel map update, adapting the PHY mode requires a mandatory delay of 6 connection events.

6 Implementation

To show the portability of our mechanism, we implement our approaches on two popular platforms: the Nordic Semiconductor nRF52 (Sect. 6.1) and Raspberry Pi 3 (Sect. 6.2).

6.1 Nordic Semiconductor nRF52

We run the Zephyr OS [35] on the `nrf52840` chip, which embeds an ARM Cortex-M4F processor providing 1024 kB

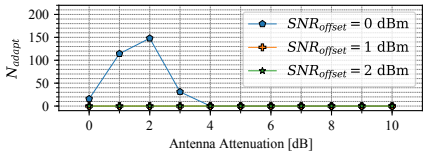


Figure 10. Number of PHY mode adaptations (N_{adapt}) for different thresholds and antenna attenuation values. *Even an $SNR_{offset} = 1\,dBm$ eliminates unnecessary PHY mode adaptations (N_{adapt}) on stable links.*

of flash and 256 kB of memory, as well as a standard-compliant BLE 5 radio supporting all four PHY modes. Although we use the `nrf52840` chip in our experiments, our code runs on all chips that are part of the nRF52 series.

Zephyr already provides a fully standard-compliant BLE communication stack that allows link-layer access. We modify the existing BLE stack by extending the `lll_conn_isr_rx` function of the link-layer implementation. This function is called after every link-layer packet reception and provides information about the packet's data channel, PDR, and RSSI. After every successfully received packet, we probe the noise floor of the used channel to calculate the packet's SNR.

Our channel blacklisting mechanism is notified about the used channel and PDR of every link-layer packet exchange by a callback in `lll_conn_isr_rx`. Using this information, our blacklisting mechanism detects bad channels and blacklists them, using the $PDR_{AVG} \geq 95\%$ channel classification, a $C_{min} = 10$, and a $S_{channel} = 10$, as described in Sect. 4.

Similarly, our PHY mode adaptation mechanism receives a new SNR readings after every successful packet reception via a callback in `lll_conn_isr_rx`. Using recent SNR values, our PHY mode adaptation chooses the best PHY that sustains a $PDR_{min} > 99\%$ while minimizing power consumption, following the approach discussed in Sect. 5.

6.2 Raspberry Pi 3

The Pi3 uses Broadcom's `BCM43430A1` chip for BLE communication [26]. This proprietary radio chip is closed source and autonomously handles all BLE's link-layer functionality, as well as classic Bluetooth and Wi-Fi communication. With *InternalBlue* [19], the firmware of most Cypress and Broadcom chips, including the `BCM43430A1`, can be analyzed and even patched with custom Assembly code.

To use our channel blacklisting mechanism on the Broadcom BLE radio, we use InternalBlue to analyze the handling of BLE link-layer packets in this radio chip. We detect that the function `_connTaskRxDone` is called upon reception of any BLE link-layer packet. In this function, we can retrieve the used data channel and PDR of the most recent link-layer exchange, which we use for blacklisting. We extend the `_connTaskRxDone` function by patching the radio's firmware to send a custom Host Controller Interface (HCI) event, containing the most recent data channel and PDR, over the standardized HCI to the BLE host after every link-layer transmission. We parse these HCI packets in InternalBlue and perform channel blacklisting as described in Sect. 4. Whenever we need to update the data channel map of the connection, we issue a standardized `Host_Channel_Classification` command via the HCI to the BLE radio. Using the standardized HCI, our approach is still fully compliant to the BLE specification.

The Broadcom radio already implements basic BLE channel blacklisting and radio co-existence mechanism that run autonomously in the background. Our channel blacklisting extends the channel blacklisting on the Pi3, but does not disable these existing link-quality improvements.

As the `BCM43430A1` radio only supports the 1M PHY, we do not implement PHY mode adaptation on the Pi3.

7 Evaluation

In this section, we experimentally study the performance of the proposed blacklisting mechanism alone (Sect. 7.1) and in parallel to the PHY mode adaptation scheme (Sect. 7.2).

Experimental setup. For this evaluation, we use an experimental setting similar to the one described in Sect. 3. To evaluate the power consumption of the BLE slave, we measure the slave's average current draw (I_{Slave}) using D-Cube [27]. We focus on the consumption of the slave, as the latter usually operates on a tight energy budget (see Sect. 2).

To measure the overall link-layer reliability (*PDR*), we parse the link-layer logs of the used master devices and calculate the PDR across the whole BLE connection.

Experimental scenarios. Following the methodology used in Sect. 3.2, our experiments start with an effective antenna attenuation of 0 dB and without any external radio interference. After the BLE connection is established, we wait for 60 s before either changing the attenuation or introducing radio interference. We gradually vary the effective attenuation of the slave antenna over 30 s from 0 dB to either 10 dB or 20 dB. We keep the attenuation at this setting for 600 s before gradually reverting back to 0 dB over 30 s. For scenarios investigating the reliability under interference, we start interference for a duration of 600 s, as described in Sect. 3.2.

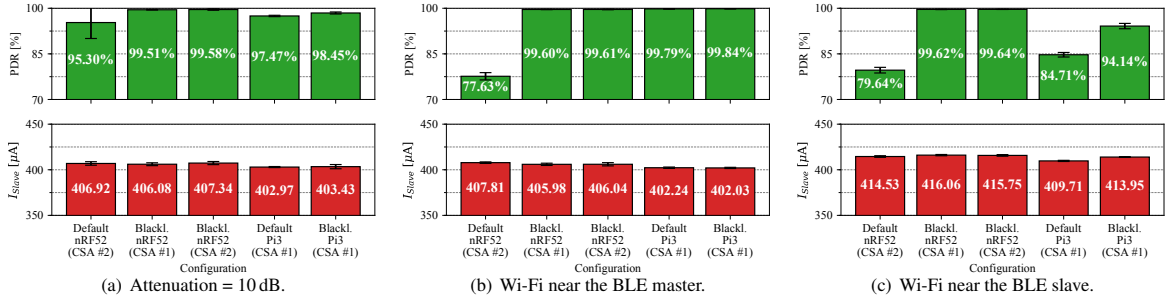
7.1 Evaluating BLE Channel Blacklisting

We evaluate the performance of our proposed channel blacklisting mechanism by measuring *PDR* and I_{Slave} in three different experimental scenarios and on two different hardware platforms, as described in Sect. 6. During this experiments, we disable our PHY mode adaptation mechanism.

Fig. 11 shows the average *PDR* and I_{Slave} for five different blacklisting mechanisms in three scenarios, where every experiment was repeated 5 times. Two bars show the default behavior of the nRF52 (*Default nRF52*) and the Raspberry Pi 3 (*Default Pi3*). The two bars named *Blackl. nRF52* show the performance of our channel blacklisting mechanism implemented on the nRF52 master for Channel Selection Algorithm (CSA) #1 and CSA #2 . The bar named *Blackl. Pi3* shows the Pi3 using our blacklisting mechanism.

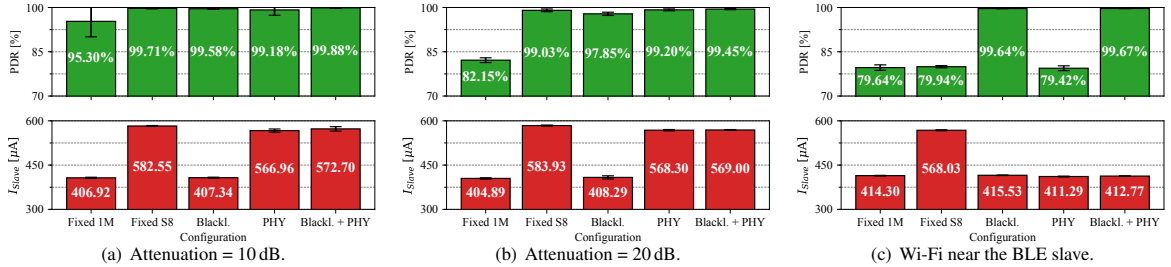
Overall, we see that our proposed channel blacklisting mechanism significantly increases the reliability (*PDR*) without introducing additional power consumption (I_{Slave}). Compared to the default behavior of the nRF52 and Pi3, our mechanism improves the *PDR* by up to +22 % and +10 %, respectively. The used CSA does not significantly impact the link-layer reliability, as shown by the similar performance of *Blackl. nRF52(CSA #1)* and *Blackl. nRF52(CSA #1)*.

In case of an antenna attenuation of 10 dB (Fig. 11(a)), our mechanism on the nRF52 sustains a PDR over 99 %.



(a) Attenuation = 10 dB. (b) Wi-Fi near the BLE master. (c) Wi-Fi near the BLE slave.

Figure 11. Link-layer reliability (*PDR*) and the slave's average current consumption (I_{Slave}) of different blacklisting mechanisms in three scenarios. *The connection was either using Channel Selection Algorithm #1 (CSA #1) or CSA #2.*



(a) Attenuation = 10 dB. (b) Attenuation = 20 dB. (c) Wi-Fi near the BLE slave.

Figure 12. Link-layer reliability (*PDR*) and the slave's average current consumption (I_{Slave}) for five different configurations. *Running both mechanisms in parallel (Blackl. + PHY) provides a PDR > 99% while minimizing power consumption.*

On the Pi3, our mechanism improves the PDR by over 1 %, reaching an overall PDR of 98.45 %. The reason for the slightly lower PDR on the Pi3 compared to the nRF52 is the proprietary radio coexistence mechanism constantly running on the Pi3, which autonomously re-enables data channels.

Under Wi-Fi interference near the BLE master (Fig. 11(b)), the default behavior of the Pi3 sustains a *PDR* above 99 %. This matches findings by Spörk et al. [32] showing that the Pi3 likely uses noise floor measurements to proactively blacklist interfered channels. Nevertheless, our passive blacklisting mechanism sustains a comparable *PDR* in this setting. Under Wi-Fi interference near the BLE slave (Fig.11(c)), the default blacklisting mechanism of the Pi3 does not detect link-layer errors and is only able to sustain a PDR of 84.71 %. Our proposed blacklisting mechanism, instead, increases the *PDR* on the Pi3 by almost 10 %. Overall, in all three experimental scenarios, our blacklisting mechanism on the nRF52 is able to sustain a PDR above 99 %.

7.2 Evaluating PHY Mode Adaptation

To evaluate the performance of our proposed PHY mode adaptation mechanism on the nRF52, we re-run the experiments from Sect. 7.1 with five different configurations: no blacklisting and a fixed 1M PHY mode (*Fixed 1M*); no blacklisting and a fixed Coded S8 PHY mode (*Fixed S8*); and our blacklisting mechanism and a fixed 1M PHY mode (*Blackl.*); no blacklisting and only our PHY mode adaptation (*PHY*); running both proposed mechanisms in parallel (*Blackl. + PHY*). We configure the PHY mode adaptation to sustain a minimum PDR of 99 %, as described in Sect. 6.

Adapting the PHY mode improves the PDR when the signal strength drops (shown in Figs. 12(a) and 12(b)). Changing to a more reliable PHY mode, however, introduces an additional current consumption (I_{Slave}) of approx. +38.9 % on the BLE slave. We can also see in Fig. 12(c) that our PHY mode adaptation does not adapt the PHY mode when co-located Wi-Fi interference is introducing link-layer errors. This shows that our blacklisting mechanism and PHY mode adaptation mechanisms work in parallel and do not conflict.

Our PHY adaptation mechanism alone is able to sustain a PDR of 99% while minimizing power consumption when possible. This is shown by the similar PDR and I_{Slave} sustained by our PHY mode adaptation compared to the use of a fixed Coded S8 PHY (Fixed S8).

Overall, we see that it is best to use both of our improvements in parallel, as they do not conflict and as they can sustain, together, a PDR $\geq$ 99% across all of our experiments.

8 Related Work

BLE reliability. Several works focus on measuring [6, 20, 30, 37] or improving [7, 23] the coexistence of BLE with other radio technologies in the 2.4 GHz ISM band, but none of them detects and blacklists poor channels at runtime. Other studies have investigated the AFH algorithms of BLE using mathematical models and simulations, but did not make use of channel blacklisting [1, 2] or employed hardware-specific features to detect and blacklist poor channels [31]. A few works use standard information available in the BLE host to estimate link quality [17] and improve timeliness by adapting connection parameters [32]. However, these approaches only *counteract* the effects of link-layer loss and are not able to improve reliability. In this work, we use link-layer metrics available in standard BLE radios to reduce packet loss at the link layer and increase reliability.

BLE 5 PHY modes. A few works study the impact of different PHY modes in connection-less [3, 9, 25] and connection-based [33] BLE systems. While the focus of these works is on comparing the achievable performance only, our current work investigates how to dynamically adapt the PHY mode used by BLE connections at runtime to sustain a high reliability while minimizing the power consumption.

Other low-power radio technologies. Besides works simulating Classic Bluetooth's frequency hopping scheme [22], most of the research on channel diversity in low-power radios has focused on IEEE 802.15.4. This body of literature, however, does not use channel blacklisting [11, 15], statically selects a channel map [36], or periodically probes the quality of channels [10, 16]. Other studies change the data channel used for infrequent transmissions (1 packet/5 minutes) when the channel quality drops over long periods [29] or use machine learning to predict the link quality of IEEE 802.15.4 channels [18]. Unlike these works, we use link-layer information available in standard BLE radios to promptly blacklist bad channels and adapt the PHY mode.

9 Conclusion and Future Work

The BLE specification foresees mechanisms to improve link-layer reliability, but does not provide guidelines on how to efficiently design and implement them. As a result, BLE devices may use proprietary solutions that provide poor link-layer reliability in real-world settings. We propose two standard-compliant mechanisms that use existing BLE primitives to increase link-layer reliability while minimizing power consumption, significantly outperforming the default solutions of popular BLE platforms. Our next steps include (i) porting the proposed improvements to other BLE platforms, such as Samsung Galaxy S10 and iPhone 11 smartphones, as well as (ii) the dynamic adaptation of BLE's transmission power to further minimize power consumption.

Acknowledgments

This work has been performed within the LEAD project "Dependable Internet of Things in Adverse Environments" funded by Graz University of Technology and in the context of the LOEWE centre emergenCITY. This work has been funded by the DFG as part of SFB 1053 MAKI, and the BMBF and the State of Hesse within ATHENE. This work was also partially funded by DFG within cfaed and the SCOTT project. SCOTT (`http://www.scott-project.eu`) has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 737422. This joint undertaking receives support from the European Unions Horizon 2020 research and innovation programme and Austria, Spain, Finland, Ireland, Sweden, Germany, Poland, Portugal, Netherlands, Belgium, Norway. SCOTT is also funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program "ICT of the Future" between May 2017 and April 2020. More info at `https://iktderzukunft.at/en`.

10 References

[1] M. Al Kalaa et al. Selection probability of data channels in Bluetooth Low Energy. In *Proc. of the* 11th *IWCMC Conf.* IEEE, 2015.

[2] M. O. Al Kalaa et al. Evaluating bluetooth low energy in realistic wireless environments. In *Proc. of the* 12th *IWCMC Conf.* IEEE, 2016.

[3] B. Al Nahas et al. Concurrent Transmissions for Multi-Hop Bluetooth 5. In *Proc. of the* 16th *EWSN Conf.*, 2019.

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

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

[6] W. Bronzi et al. Bluetooth low energy performance and robustness analysis for inter-vehicular communications. *Ad Hoc Networks*, 2016.

[7] O. Carhacioglu et al. Time-domain cooperative coexistence of BLE and IEEE 802.15.4 networks. In *Proc. of the* 28th *PIMRC Symp.*, 2017.

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

[9] M. Collotta et al. Bluetooth 5: A concrete step forward toward the IoT. *IEEE Communications Magazine*, (7), 2018.

[10] P. Du et al. Adaptive time slotted channel hopping for wireless sensor networks. In *Proc. of the* 4th *CEEC Conf.*, 2012.

[11] R. Hermeto et al. Passive Link Quality Estimation for Accurate and Stable Parent Selection in Dense 6TiSCH Networks. In *Proc. of the* 15th *EWSN Conf.*, 2018.

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

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

[14] J. Ko and M. Chang. Momoro: Providing mobility support for low-power wireless applications. *IEEE Systems Journal*, 2014.

[15] V. Kotsiou et al. Is local blacklisting relevant in slow channel hopping low-power wireless networks? In *Proc. of the ICC Conf.* IEEE, 2017.

[16] V. Kotsiou et al. LABeL: Link-based adaptive blacklisting technique for 6TiSCH wireless industrial networks. In *Proc. of MSWiM*, 2017.

[17] T. Lee, M.-S. Lee, H.-S.Kim, and S. S. Bahk. A synergistic architecture for RPL over BLE. In *Proc. of the* 13th *SECON Conf.*, 2016.

[18] T. Liu and A. E. Cerpa. Foresee (4C): Wireless link prediction using link features. In *Proc. of the* 10th *IPSN Conf.*, 2011.

[19] D. Mantz et al. InternalBlue - Bluetooth Binary Patching and Experimentation Framework. In *Proc. of the* 17th *MobiSys Conf.*, 2019.

[20] A. Marinčić et al. Interoperability of iot wireless technologies in ambient assisted living environments. In *Proc. of the WTS Symp.*, 2016.

[21] Mini-Circuits. RCDAT-8000-90 - Programmable Attenuator, 2017.

[22] K. Morsi et al. Performance estimation and evaluation of Bluetooth frequency hopping selection kernel. In *Proc. of the JCPC Conf.*, 2009.

[23] R. Natarajan et al. Analysis of Coexistence between IEEE 802.15. 4, BLE and IEEE 802.11 in the 2.4 GHz ISM Band. In *Proc. of the* 42th *IECON Conf.* IEEE, 2016.

[24] Nordic Semiconductors. nRF52840 Specifications, 2018.

[25] G. Pau et al. Bluetooth 5 Energy Management through a Fuzzy-PSO Solution for Mobile Devices of Internet of Things. *Energies*, 2017.

[26] Raspberry Pi Foundation. Raspberry Pi 3 Model B, 2018.

[27] M. Schuß et al. Moving Beyond Competitions: Extending D-Cube to Seamlessly Benchmark Low-Power Wireless Systems. In *Proc. of the* 1st *CPSBench Worksh.*, 2018.

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

[29] M. Sha et al. ARCH: Practical channel hopping for reliable home-area sensor networks. In *Proc. of the* 17th *RTAS Symp.*, 2011.

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

[31] M. Spörk et al. BLEach: Exploiting the Full Potential of IPv6 over BLE in Constrained Embedded IoT Devices. In *Proc. of the* 15th *ACM SenSys Conf.*, 2017.

[32] M. Spörk et al. Improving the Timeliness of Bluetooth Low Energy in Noisy RF Environments. In *Proc. of the* 16th *EWSN Conf.*, 2019.

[33] M. Spörk et al. Performance and Trade-offs of the new PHY modes of BLE 5. In *Proc. of the 2019 PERSIST-IoT Workshop.* ACM, 2019.

[34] K. Srinivasan et al. An empirical study of low-power wireless. *ACM Transactions on Sensor Networks*, 6(2), 2010.

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

[36] T. Watteyne et al. Reliability through frequency diversity: why channel hopping makes sense. In *Proc. of the* 6th *PE-WASUN Symp.*, 2009.

[37] T. Winkel. Robustness of Bluetooth Low Energy in In-Vehicle Networks-An Experimental Study. Master's thesis, 2016.

Publication E

M. Spörk, M. Schuß, C. A. Boano, and K. Römer. Ensuring End-to-End Dependability Requirements in Cloud-based Bluetooth Low Energy Applications. In *Proceedings of the* 18^{th} *International Conference on Embedded Wireless Systems and Networks (EWSN)*. Junction Publishing, 2021

DOI: 10.5555/3451271.3451277
Link: `https://dl.acm.org/doi/10.5555/3451271.3451277`

Abstract. Bluetooth Low Energy (BLE) is increasingly used for time-critical IoT applications, where BLE-based smart objects need to exchange data with a remote server within stringent end-to-end latency and reliability bounds. While existing research has investigated how to timely send packets between pairs of BLE devices, it is still unclear how a BLE device can sustain time-critical end-to-end communication with a remote server, for example, hosted in the cloud.

In this paper, we tackle this problem and show how BLE devices can autonomously measure and cope with end-to-end network delays and loss along the path to the remote server. To this end, we first devise an analytical model of the communication between a BLE end-node and the cloud. We then leverage this model to dynamically adapt the communication parameters of the BLE device and sustain the desired end-to-end dependability requirements while minimizing the energy expenditure. Specifically, we design and implement two adaptation strategies on the popular `nRF52` platform, and experimentally show that they both allow to sustain a given end-to-end reliability and a given end-to-end latency for data transmissions from/to the BLE node, while limiting the node's power consumption.

My contribution. I am the main author of this paper and was responsible for designing and implementing the work to sustain end-to-end communication requirements. I wrote the vast majority of the paper in collaboration and discussion with the co-authors and carried out all the experiments presented in this work. However, Markus Schuß significantly helped me in instrumenting the D-Cude testbed to run our experiments. I presented the paper at EWSN'21.

Ensuring End-to-End Dependability Requirements in Cloud-based Bluetooth Low Energy Applications

Michael Spörk, Markus Schuß, Carlo Alberto Boano, and Kay Römer
Institute of Technical Informatics,
Graz University of Technology, Austria
{michael.spoerk, markus.schuss, cboano, roemer}@tugraz.at

Abstract

Bluetooth Low Energy (BLE) is increasingly used for time-critical IoT applications, where BLE-based smart objects need to exchange data with a remote server within stringent end-to-end latency and reliability bounds. While existing research has investigated how to timely send packets between pairs of BLE devices, it is still unclear how a BLE device can sustain time-critical end-to-end communication with a remote server, for example, hosted in the cloud.

In this paper, we tackle this problem and show how BLE devices can autonomously measure and cope with end-to-end network delays and loss along the path to the remote server. To this end, we first devise an analytical model of the communication between a BLE end-node and the cloud. We then leverage this model to dynamically adapt the communication parameters of the BLE device and sustain the desired end-to-end dependability requirements while minimizing the energy expenditure. Specifically, we design and implement two adaptation strategies on the popular `nRF52` platform, and experimentally show that they both allow to sustain a given end-to-end reliability and a given end-to-end latency for data transmissions from/to the BLE node, while limiting the node's power consumption.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*; C.4.6 [**Performance of Systems**]: Reliability, availability, and serviceability.

General Terms

Design, Measurement, Performance, Protocols, Reliability.

Keywords

Adaptive Protocols, Bluetooth Low Energy, End-to-end Latency, End-to-end Reliability, Internet of Things.

1 Introduction

Bluetooth Low Energy (BLE) is one of the most popular low-power wireless technologies in the IoT landscape, due to its wide adoption in consumer electronics devices connected to the Internet such as smart-phones, tablets, and laptops [5]. The resulting ease with which resource-constrained BLE-based smart objects can interface to such devices and connect to the Internet is a key enabler for the development of attractive IoT systems based on BLE technology. Many of these IoT systems operate in time-critical domains: examples are smart grids [8], smart cities [15], and smart healthcare [16, 22] applications, where resource-constrained BLE nodes often need to exchange data with a cloud server within strict end-to-end transmission reliability and latency bounds; all of this while operating on batteries for months or years.

For example, in remote ECG monitoring systems [22] a BLE node measures cardiac signals of a patient and sends these measurements to a router in the same BLE subnet, as shown in Fig. 1. The router forwards these data packets containing ECG measurements via the external network path, *i.e.*, the Internet, to a cloud server to be processed or stored.

The data exchange between a BLE node and a remote server is often subject to dynamic changes in the transmission delay and loss across the entire network path. In the BLE subnet, packets may be significantly delayed due to persistent or transient link-layer problems [35, 36]: this is often due to multipath fading effects and due to the presence of RF interference from surrounding devices (*e.g.*, co-located Wi-Fi access points). On the external network path, packets may be lost due to buffer congestion and CPU-intensive tasks (*e.g.*, routing table updates) on backbone routers. Data transmissions can also be significantly delayed due to routing changes in the Internet backbone or due to link quality fluctuations in the case of cellular connections [11, 13, 40].

To sustain end-to-end dependability requirements on communication, BLE nodes need to capture and adapt to all these changes at runtime. This requires a proper knowledge about the delay and loss across the entire network path and appropriate models: only this way, a BLE node can adapt its parameters at runtime and select the right trade-off between communication timeliness, reliability, and power efficiency.

Capturing network delay and loss. Although several works have investigated how to capture and adapt to changes in delay and loss across the Internet [1, 2, 11, 13, 30], the focus has always been on devices that have a high-bandwidth

International Conference on Embedded Wireless Systems and Networks (EWSN) 2021
17–19 February, Delft, The Netherlands
ISBN: 978-0-9949886-5-2

Internet connectivity and that are not constrained in their processing capabilities or power supply.

In contrast to the studies above, a large body of research has investigated how to sustain latency and reliability bounds in constrained and low-power wireless networks, based for example on IEEE 802.15.4 [14, 17, 42] or BLE [35, 36]. Building upon these works, nodes can cope with link-layer problems in the local subnet and sustain a timely and reliable communication while minimizing their power consumption. The problem, however, is that none of these studies investigates how to sustain *end-to-end* requirements for communications that go beyond the local subnet, *e.g.*, when a node exchanges data with a cloud server over the Internet as illustrated in Fig. 1. Therefore, how BLE nodes can effectively capture delays and loss across the *entire* network path remains an open question. Answering the latter is fundamental to allow the selection of suitable BLE connection parameters at runtime in order to sustain end-to-end dependability bounds while limiting a node's energy expenditure.

Sustaining end-to-end requirements on a budget. Another open issue is that existing analytical models of BLE's communication performance as a function of the available connection parameters are unsuitable to design adaptive strategies at runtime. Several models, indeed, require low-level channel information that is not available on off-the-shelf BLE devices [23, 29, 34]. Only the model presented by Spörk et al. [36] is able to use standard BLE information to monitor the timeliness of BLE communications. Unfortunately, however, this model is limited to transmissions within the local BLE subnet. Furthermore, all the aforementioned models suffer from two additional limitations: (i) they focus only on transmitting nodes and neglect how a node can sustain dependability bounds when acting as a receiver; (ii) they solely adapt the BLE connection interval, *i.e.*, they neglect the *BLE slave latency*, a connection parameter that can greatly influence the behaviour of a BLE device [5].

There is hence a need for new end-to-end models that keep the entire network path in the picture and also include parameters such as the BLE slave latency. This way, one has the means to properly adapt BLE communication parameters at runtime to sustain given end-to-end latency and reliability bounds while preserving power efficiency.

Contributions. In this work, we tackle all these challenges and show how BLE nodes can sustain time-critical communication with a remote server on the cloud in both directions.

First, we devise a new end-to-end BLE model, incorporating a local model by Spörk et al. [36], that captures the additional latency introduced by the network path outside of the BLE subnet. In doing so, we also let the model capture the impact of the BLE slave latency on communication latency as well as the interplay between the connection interval and slave latency on the timeliness of packet receptions.

Next, we show how a BLE node can accurately and efficiently estimate the communication latency across the entire network path by using short and infrequent probing bursts. Our estimation approach fully adheres to the end-to-end principle of the Internet, *i.e.*, it requires no changes to any of the devices routing data on the network path.

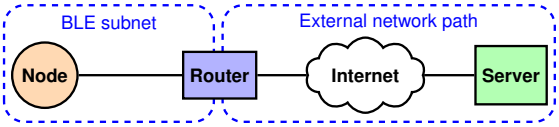


Figure 1. Network topology when exchanging data between a BLE node and a cloud server over the Internet.

We further leverage the proposed model and latency estimation scheme to let a node adapt its BLE communication parameters at runtime and sustain a given end-to-end transmission reliability as well as a given end-to-end latency when communicating with a cloud server. Specifically, we propose adaptation strategies for two different use cases: (i) the router being constrained in its BLE radio duty cycle and (ii) the router not having radio duty cycle constraints.

We implement our adaptation approaches on the popular Nordic Semiconductor `nRF52` platform [26] using Zephyr OS [37] and experimentally show that both approaches effectively find suitable BLE parameters at runtime that minimize delayed packets and power consumption, outperforming other static or adaptive node configurations. In our implementation we make use of IPv6-over-BLE communication as specified by the RFC 7668 [25] to exchange data between a BLE node and cloud as illustrated in Fig. 1. Nevertheless, our model, estimation, and adaptation approaches are independent of the used network layer on top of the BLE connection and can directly be used for IoT applications using BLE communication based on GATT.

After introducing the necessary background information and providing a real-world measurement of the delays and packet loss in typical cloud-based BLE applications in Sect. 2, this paper makes the following contributions:

- We devise a new end-to-end BLE model that captures the *end-to-end* latency across the whole network path and embeds the role of the BLE slave latency (Sect. 3).
- We show how a BLE node can autonomously estimate the latency across the entire network path while complying to the end-to-end principle of IP (Sect. 4).
- We propose two different adaptation strategies that a node can use at runtime to sustain end-to-end requirements under different constraints (Sect. 5).
- We implement both approaches on the `nRF52` platform using Zephyr (Sect. 6), and we experimentally evaluate their performance in detail (Sect. 7).

After describing related research in Sect. 8, we conclude our paper and list future work in Sect. 9.

2 Investigating cloud-based BLE applications

We start our work by experimentally investigating the end-to-end latency and end-to-end reliability of a BLE node exchanging data with a cloud server on the Internet.

Fig. 1 shows the used network topology, where an IPv6-over-BLE node device is connected to an IPv6-over-BLE router providing Internet access. To exchange IPv6 packets, node and router establish an IPv6-over-BLE connection according to the RFC 7668 [25]. Once this connection is established, the router forwards the packets to the nodes within the IPv6-over-BLE subnet or to IP devices on the Internet, such as our server. This common network topology is used

Table 1. Measured latencies between node and server over 7 days in our testbed for an IPv6 packet length of 128 bytes. *The table shows the median (50%), 95 percentile (95%), and maximum experienced latency (100%) for different configurations.*

conn.	*CI* [ms]	*SL*	t_{TX} [ms]			$t_{TX_{BLE}}$ [ms]			t_{RX} [ms]			$t_{RX_{BLE}}$ [ms]		
			50%	**95%**	**100%**	**50%**	**95%**	**100%**	**50%**	**95%**	**100%**	**50%**	**95%**	**100%**
wired	50	0	40	88	328	31	79	319	42	92	293	32	82	284
wired	50	4	39	86	732	30	75	258	41	292	1291	32	282	1281
cellular	50	0	73	1157	3026	28	69	237	67	601	2411	34	64	176
cellular	50	4	73	739	4038	27	62	283	268	531	1031	191	257	499

by popular IoT protocols, such as CoAP, MQTT, or MQTT-SN. One major constraint for low-power IoT applications, however, is that they usually do not use the heavyweight TCP transport layer to reliably send data. Instead, these applications use a UDP transport layer that allows low-power consumption, at the cost of packet loss on the network path [4].

To exchange data, node and router set up a BLE connection, where communication happens during connection events. During these connection events, node and router bidirectionally exchange data until both devices have no more data to send or until the maximum connection event length (t_{CE}) has been reached. The *connection interval* (*CI*) defines the time between two consecutive connection events. Even if no data needs to be transmitted, node and router exchange short mandatory keep-alive link-layer packets in every connection event, to keep the BLE connection alive. As these keep-alive messages cause unnecessary power consumption on the BLE devices, the BLE specification foresees the BLE *slave latency* (*SL*) parameter, which allows the node to skip up to *SL* connection events.

BLE connections make use of adaptive frequency hopping and autonomous packet retransmissions to ensure that every packet that is scheduled to be transmitted over BLE will be successfully received. These mechanisms, indeed, lead to a reliability of 100% within the BLE subnet, as shown in [36], although some packets may be delayed due to link-layer effects, such as external radio interference.

The timeliness and reliability of IPv6 packets across the external network path depend on the employed technology (*e.g.*, Ethernet or 4G) and cannot be controlled by the node. To sustain an upper bound on the end-to-end latency between the node and server, the node can only dynamically adapt its BLE connection interval and slave latency

To investigate the real-world behavior of BLE-based IoT applications, we perform a first experimental study.

Experimental setup. To measure latency of IPv6 packets across the whole network topology (Fig. 1), we perform our measurements in a wireless testbed powered by D-Cube nodes [31] located in a vacant laboratory.

We use four Nordic Semiconductor `nRF52840 DK` devices, each running an IPv6-over-BLE node application built on the Zephyr OS that transmits a UDP packet to the server once every second. The server is an Amazon Web Service (AWS) instance located in the AWS center in Frankfurt, Germany, and runs a Python server echoing every UDP packet received by a node. Both, packet to and from the server, have an IPv6 packet length of 128 bytes and carry a unique sequence number to match request to response. We use a Raspberry Pi 4 (Pi4) as IPv6-over-BLE router running Raspbian OS, which provides Internet access to the nodes. To make use of the features of BLE version 5, we use another `nRF52840 DK` and program it with Zephyr's BLE HCI-USB firmware, which allows us to use this nRF device as BLE radio on our router. For all of our experiments, the BLE connection uses the 2M PHY mode of BLE, all available 37 BLE data channels and does not adaptively blacklist channels, which is the default behavior of the HCI-USB firmware.

Every time a BLE node issues a UDP packet, it triggers a GPIO event that is captured by a D-Cube device (one D-Cube device per nRF52 node). We further log all IPv6 traffic on the router to measure the communication latency of IPv6 packets in the BLE subnet. Finally, we also log the timestamp and source address of UDP packets received by the server. All devices in our experiment are synchronized to the same NTP server and, therefore, share the same notion of time (the average clock offset between nodes and server is -43 $\pm$ 134μs). This allows us to calculate the end-to-end latency (t_{TX}) and the latency within the BLE subnet ($t_{TX_{BLE}}$) for every packet sent from node to server. Similarly, we calculate the end-to-end latency (t_{RX}) and the latency within the BLE subnet ($t_{RX_{BLE}}$) for packets sent from server to node.

To test the impact of different technologies providing Internet access to our BLE subnet, our router can make use of a wired Ethernet or a cellular 4G connection.

Preliminary results. Tab. 1 shows the distribution of the measured latencies for different BLE connection parameters and two different Internet connections measured over 7 days.

When comparing a wired Internet connection to a cellular connection, we can clearly see that the used Internet connectivity significantly affects the overall communication latency in both directions. In our experiments, the maximum latency for transmitting and receiving IPv6 packets with a cellular connection is almost 10 times higher than a wired connection. Also the median latency in both directions is significantly higher for a cellular than for a wired connection. This shows that a node cannot simply assume a fixed delay across the external network path to sustain latency bounds.

When investigating different BLE connection parameters, the data in Tab. 1 shows that the *SL* parameter does not significantly affect the latency in the BLE subnet of packets sent by the node ($t_{TX_{BLE}}$). As expected, the *SL*, however, significantly affects the latency of packets received by the node ($t_{RX_{BLE}}$). The reason for that is the node skipping up to *SL* connection events when it has no data to transmit, which means that the router may need to wait up to *SL* connection events for the node to wake up and receive packets[1].

During our experiment, the wired Internet connection experienced an average end-to-end transmission reliability of

[1]The actual slave latency behavior depends on the node's BLE link-layer implementation and may differ between different BLE chip vendors.

99.35%, with at least 97% of any 100 subsequent transmissions successfully sent. The cellular connection sustained an average end-to-end transmission reliability of 97.65% and a minimum reliability of 92% for any 100 transmissions. Similar to [36], no packet was lost within the BLE subnet.

Based on the experimental data, we can see that the end-to-end latency and reliability of packets sent by a BLE node depend on (i) the used BLE connection parameters and (ii) the behavior of the external network path. To sustain a given end-to-end reliability and a given end-to-end latency while limiting unnecessary power consumption, a node needs to dynamically adapt its BLE connection parameters to changes in the overall network path. To find the most suitable BLE connection parameters, however, new BLE models are required that, in addition to capturing the effects of both the connection interval and slave latency, need to cope with the behavior of the entire network path.

One approach to sustain given end-to-end latency bounds would be to use application-level round-trip time (RTT) measurements on the node to adapt the BLE connection parameters. The main problem, however, is that existing application traffic cannot be used to accurately estimate the one-way communication latencies t_{TX} or t_{RX}. During normal operation, the BLE node uses a BLE slave latency greater than 0, which allows the node to limit power consumption while sustaining given latency bounds. A value $SL > 0$ leads to packet receptions being unpredictably delayed, which significantly affects the accuracy of individual t_{TX} and t_{RX} estimates.

In this work, we follow another approach, where we first devise new end-to-end models that captures the delay across the entire network path (Sect. 3) and show how a BLE node can use infrequent probing bursts to accurately estimate the network latency across the entire network path in Sect. 4. In Sect. 5, we combine our model and network latency estimation to sustain given end-to-end dependability requirements.

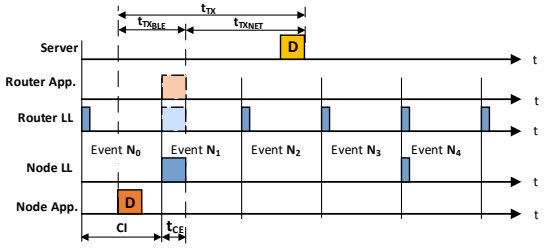
3 Modeling BLE communication

In this section, we devise a new end-to-end model, which incorporates the local BLE model from [36], to fit the use case shown in Fig. 1. Towards this goal, we do not only extend the timeliness model to account for delays across the Internet, but also investigate how the BLE connection interval and slave latency affect timeliness when the node transmits (Sect. 3.1) and receives data (Sect. 3.2).

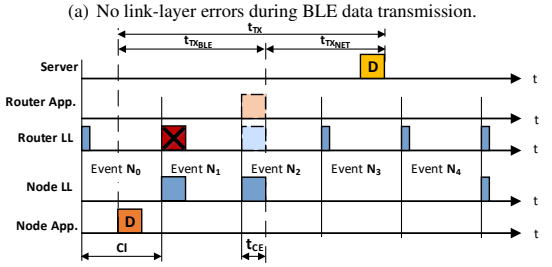
3.1 Transmitting IPv6-over-BLE data

First, we model the end-to-end latency (t_{TX}) for a node transmitting a data packet to a cloud server.

Fig. 2 shows two exemplary transmissions from the BLE node via a router to the server. In both examples, the application on the BLE node (Node App.) issues a data transmission (D) between connection event N_0 and N_1. When no link-layer errors occur (shown by Fig. 2(a)), the BLE link-layer (LL) of the node and router successfully exchange the data in connection event N_1. In case there are link-layer errors, *i.e.*, due to external Wi-Fi interference (shown in Fig. 2(b) in connection event N_1), the link-layer of the BLE node retransmits the packet until successfully received by the router. After successfully receiving the packet, the router forwards the packet via the Internet to the server, which takes $t_{TX_{NET}}$.



(a) No link-layer errors during BLE data transmission.



(b) One link-layer error during BLE data transmission.

Figure 2. Timing for a BLE node transmitting a data packet (D) to a cloud server over the Internet.

In these examples, the packet length (D_{TX}) is smaller than the maximum packet length that can be sent during a single connection event (F_{TX}) and, therefore, only one successful connection event is used to transmit the packet. If $D_{TX} > F_{TX}$, the packet is split into multiple data fragments, where every fragment requires its own successful connection event.

As Fig. 2 shows, the end-to-end latency (t_{TX}) for sending a data packet from node to server consists of:

$$t_{TX} = t_{TX_{BLE}} + t_{TX_{NET}}, \tag{1}$$

where $t_{TX_{BLE}}$ is the transmission latency of the packet between node and router and $t_{TX_{NET}}$ is the transmission latency of the packet from router to server. The $t_{TX_{NET}}$ delay depends on the used Internet connection and will be estimated in Sect. 4. We model the $t_{TX_{BLE}}$ delay as shown in [36] as:

$$t_{TX_{BLE}} = \left(\sum_{f=1}^{\lceil D_{TX}/F_{TX} \rceil} n_{CE_f} \cdot CI \right) + t_{CE}, \tag{2}$$

where D_{TX} is the length of the sent packet and F_{TX} is the maximum packet length that can be sent from node to router within a single BLE connection event. n_{CE_f} is the number of connection events necessary to successfully transmit an individual packet or fragment, capturing any link-layer retransmissions. CI represents the BLE connection interval, and t_{CE} is the maximum duration of a single connection event.

When combining Eq. 1 with Eq. 2, we can calculate the end-to-end transmission latency t_{TX} as:

$$t_{TX} = \left(\sum_{f=1}^{\lceil D_{TX}/F_{TX} \rceil} n_{CE_f} \cdot CI \right) + t_{CE} + t_{TX_{NET}}. \tag{3}$$

To calculate the upper bound on the end-to-end transmission latency ($t_{TX_{MAX}}$), we assume that every individual packet fragment is sent with $n_{CE_f} = n_{CE_{MAX}}$, *i.e.*, every link-layer transmission experiences the same link-layer error proba-

bility. Furthermore, we assume that the packet experiences the maximum current delay across the external network path ($t_{TX_{NET}} = t_{NET_{MAX}}$), which we discuss in Sect. 4. Applying these assumptions to Eq. 3, we derive that:

$$t_{TX_{MAX}} \geq n_{CE_{MAX}} \cdot \left\lceil \frac{D_{TX}}{F_{TX}} \right\rceil \cdot CI + t_{CE} + t_{NET_{MAX}}. \quad (4)$$

Note that the slave latency (SL) does not impact t_{TX} and $t_{TX_{MAX}}$. Indeed, the slave latency allows a BLE node to skip connection events if no data has to be transmitted. If the node, however, has data to transmit (as it is the case here), the node simply does not make use of the slave latency.

3.2 Receiving IPv6-over-BLE data

Next, we model the end-to-end latency (t_{RX}) for a node receiving a data packet from the server.

Fig. 3 shows two scenarios involving a packet sent from the server to the BLE node. In both examples, the server issues a data transmission between connection event N_0 and N_1 and the router application (Router App.) successfully receives the packet between event N_1 and N_2, which takes $t_{RX_{NET}}$. Next, the router forwards the packet to the node and therefore issues a transmission on its link layer (Router LL). In connection event N_2, the router LL tries to send the packet over BLE to the node, but the node makes use of its configured slave latency ($SL = 2$) and does not wake up during event N_2 to receive any data. The router LL retransmits the packet to the node until it is successfully received. In the example in Fig. 3(a), this happens during connection event N_3. In Fig. 3(b), router and node wake up during connection event N_3, but due to a link-layer problem (*e.g.*, external radio interference), the packet is not successfully received by the node. As the node has not received a valid BLE link-layer packet from the router, it follows the behavior required by the BLE specification [5] and wakes up during every subsequent connection event until a valid BLE packet from the router is received. In our example (Fig. 3(b)) this happens during event N_4, after which the packet from the server is successfully received by the application on the BLE node.

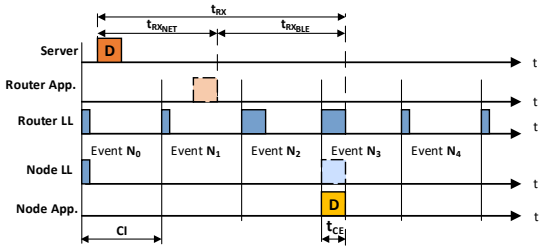
In both examples, the packet length (D_{RX}) is smaller than the maximum packet length that can be received during a single connection event (F_{RX}). If $D_{RX} > F_{RX}$, multiple successful connection events are necessary to send the packet from router to node. In such a case, the node is informed about more data on the router via the MD-field in the BLE link-layer header and does not make use of its slave latency until all data from the router is successfully received.

As shown in Fig. 3, the end-to-end latency (t_{RX}) for a node receiving a packet from a remote server consists of:

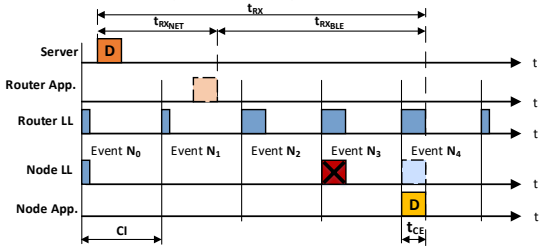
$$t_{RX} = t_{RX_{NET}} + t_{RX_{BLE}}, \quad (5)$$

where $t_{RX_{NET}}$ is the latency of the packet between server and router and $t_{RX_{BLE}}$ is the latency of the packet from router to node. Similar to Sect. 3.1, $t_{RX_{NET}}$ is dependent on the quality of the Internet connection and will be studied in Sect. 4. $t_{RX_{BLE}}$ can be modeled as:

$$t_{RX_{BLE}} = \left(\sum_{f=1}^{\lceil D_{RX}/F_{RX} \rceil} n_{CE_f} \cdot CI \right) + t_{CE} + t_{SL}, \quad (6)$$



(a) No link-layer errors during BLE data transmission.



(b) One link-layer error during BLE data transmission.

Figure 3. Timing for a BLE node receiving a data packet (D) from a cloud server over the Internet.

where D_{RX} is the length of the data packet and F_{RX} is the maximum packet length that can be received by the node within a single BLE connection event. n_{CE_f} is the number of connection events necessary to successfully transmit an individual packet or fragment (capturing any necessary link-layer retransmissions), CI is the BLE connection interval, and t_{CE} is the maximum duration of a single connection event. t_{SL} measures the additional time the router needs to wait until the node is waking up after skipping up to SL connection events, which has a maximum value of:

$$t_{SL_{MAX}} = CI \cdot SL. \quad (7)$$

When combining Eq. 5 with Eq. 6, we can calculate the overall end-to-end reception latency t_{RX} as:

$$t_{RX} = t_{RX_{NET}} + \left(\sum_{f=1}^{\lceil D_{RX}/F_{RX} \rceil} n_{CE_f} \cdot CI \right) + t_{CE} + t_{SL}. \quad (8)$$

Similar to Sect. 3.1, we adapt Eq. 8 to calculate the upper bound on the end-to-end reception latency ($t_{RX_{MAX}}$) by using $n_{CE_f} = n_{CE_{MAX}}$, $t_{RX_{NET}} = t_{NET_{MAX}}$, and Eq. 7 to obtain:

$$t_{RX_{MAX}} \geq (n_{CE_{MAX}} \cdot \left\lceil \frac{D_{RX}}{F_{RX}} \right\rceil + SL) \cdot CI + t_{CE} + t_{NET_{MAX}}. \quad (9)$$

Compared to Sect. 3.1, we can see that the slave latency (SL) impacts t_{SL} and hence the time it takes for a node to receive a packet from the server. Using a value of $SL > 0$, the BLE node skips connection events to minimize power consumption when it has no data to transmit. This, however, means that the router may need to wait up to SL connection events until the node wakes up to receive data.

The models presented in Sect. 3.1 and 3.2 allow us to calculate the latency for transmitting and receiving data packets on the BLE node. One critical aspect missing, however, is the delay that is caused by the network path outside the BLE subnet (t_{NET}), which we investigate next.

4 Estimating Network Latency

In this section, we discuss how a BLE node can estimate t_{NET} in a way that is fully compliant to the end-to-end principle of IP. Following this principle, our t_{NET} estimation approach allows nodes to estimate t_{NET} without requiring any changes to devices on the network path, such as the router. Therefore, our estimation approach can easily be used on any node and works with any IPv6-over-BLE router that adheres to the specification in [25]. Approaches violating the IP end-to-end principle, *i.e.*, requiring changes to the routers in the network, lead to huge setup and deployment costs and are therefore seldomly used in practice [41].

Our t_{NET} estimation is fully technology-agnostic and estimates t_{NET} independently of the applied technology to connect to the Internet. Furthermore, our estimation approach can also be used to estimate t_{NET} in applications of any network scope, *e.g.*, applications spanning only a local Intranet.

While we use our estimation approach to estimate the maximum latency across the entire network path, our approach may also be used to calculate the average latency in order to synchronize a node's clock, *e.g.*, via NTP.

The main problem in estimating network latency in our use case, however, is that existing application traffic cannot be used to accurately estimate t_{NET}. During normal operation, the BLE node uses a long BLE connection interval (*CI*) and a BLE slave latency (*SL*) greater than 0 that allow the node to sustain end-to-end latency bounds while limiting power consumption, but cause two problems while estimating t_{NET}. As described in Sect. 3.2, $SL > 0$ leads to packet receptions being unpredictably delayed, affecting the accuracy of individual t_{NET} estimates. Furthermore, a large *CI* value leads to a coarse sampling resolution of the application-level round trip time and therefore leads to coarse t_{NET} estimates.

4.1 Probing network latency

To estimate t_{NET} on a node, we periodically perform short probing bursts, where we exchange short probe and corresponding acknowledgment packets between node and server.

At the start of every probing burst, the node updates its BLE connection parameter to the smallest possible *CI* and $SL = 0$[2]. With the smallest possible *CI*, the node is able to sample t_{NET} with the lowest possible sampling resolution. By using $SL = 0$ during probing, we eliminate any unpredictable delay caused by *SL* during reception (as captured by t_{SL} in Eq. 8) that would impact our estimation accuracy.

After the BLE connection parameters for probing are successfully set, the node transmits a short probing packet to the server, to which the server responds with a short acknowledgment packet. The node issues a new probing packet as soon as the previous probing packet has been acknowledged by the server. These probe packets can be ordinary application data packets or also distinct IPv6-based packets solely used for probing, such as ICMPv6 echo requests and responses. To get the most accurate t_{NET} estimations, however, probe and acknowledgment packets need to fit within a single connection event, *i.e.*, $D_{TX} \leq F_{TX}$ and $D_{RX} \leq F_{RX}$. When node and server have exchanged L_{Probe} probe/acknowledgment packets, the node reverts the BLE connection parameters to the settings used before probing and continues with its normal behavior.

For every exchanged probe/acknowledgment pair, the BLE node measures the round trip time (t_{RTT}) and the BLE transmission time ($t_{TX_{BLE}}$). t_{RTT} is measured as the time between the node application issuing the probe transmission and the node successfully receiving the corresponding acknowledgment. To measure $t_{TX_{BLE}}$, the BLE node monitors the communication on the standardized BLE Host Controller Interface (HCI) of the BLE controller, as shown by [36]. By measuring the time between the node application issuing the BLE data transmission and the BLE controller actually freeing the corresponding data buffer, the node application is able to measure $t_{TX_{BLE}}$ in a standard-compliant way.

The measured time t_{RTT} can be modeled as:

$$t_{RTT} = t_{Probe} + t_{ACK}, \tag{10}$$

where t_{Probe} is the end-to-end transmission latency of a probe packet from node to server and t_{ACK} is the end-to-end transmission latency of the acknowledgment from server to node. By using Eq. 1 and Eq. 5 for modeling t_{Probe} and t_{ACK}, respectively, we get:

$$t_{RTT} = t_{TX_{BLE}} + t_{TX_{NET}} + t_{RX_{NET}} + t_{RX_{BLE}}. \tag{11}$$

One simplification for our estimation approach is that we assume the delay across the Internet is symmetric and call this delay t_{NET}, *i.e.*, $t_{NET} = t_{TX_{NET}} = t_{RX_{NET}}$, which gives us:

$$t_{RTT} = t_{TX_{BLE}} + 2 \cdot t_{NET} + t_{RX_{BLE}}. \tag{12}$$

The network delay t_{NET} can thereby be calculated as:

$$t_{NET} = \frac{t_{RTT} - t_{TX_{BLE}} - t_{RX_{BLE}}}{2}. \tag{13}$$

Although we assume a symmetric Internet delay, our t_{NET} estimation approach can also accurately capture the actual one-way network latency of asymmetric Internet connections, such as 4G communication, as we show in Sect. 4.3.

While the node can measure t_{RTT} and $t_{TX_{BLE}}$ for every probe, $t_{RX_{BLE}}$ cannot be measured by monitoring HCI communication, but needs to be estimated by using the information available on the BLE node. For our estimation approach, we assume that $t_{RX_{BLE}} = t_{TX_{BLE}}$, as both probing and acknowledgment packets fit within one single connection event and $t_{TX_{BLE}}$ provides us with the most recent link-layer information. Furthermore, we assume that receiving a packet from the router takes at least *CI*, which is the smallest time unit we can measure with our probing approach, resulting in:

$$t_{RX_{BLE}} = MAX(t_{TX_{BLE}}, CI). \tag{14}$$

4.2 Estimating maximum network latency

After measuring the network latency (t_{NET}) of individual packet exchanges during probing, we use these measurements to estimate the maximum network latency ($t_{NET_{MAX}}$) of future packet exchanges. With $t_{NET_{MAX}}$ estimations and our proposed model in Sect. 3, we are able to sustain end-to-end communication requirements, as we show in Sect. 5.

Fortunately, estimating upper bounds on transmissions on the Internet is a well-researched topic [11, 19, 20]. While

[2] This is done by using the standardized BLE connection parameter negotiation process defined by the BLE specification [5].

most recent research uses sophisticated statistical analysis [11, 30], only a subset of these studies propose solutions that are suitable for devices that are constrained in their processing capabilities and power supply, such as BLE nodes. For our estimation, we adapt the efficient and well-established round-trip time estimation approach of TCP as specified by Jacobson [18] and standardized in [28].

Instead of using an exponentially weighted moving average as proposed in [28], we store all individual t_{NET} measurements during a probe burst and calculate their average ($t_{NET_{AVG}}$) and variance ($t_{NET_{VAR}}$) at the end of every burst. Using the average and variance, we can calculate $t_{NET_{MAX}}$ as:

$$t_{NET_{MAX}} = t_{NET_{AVG}} + K \cdot t_{NET_{VAR}}, \quad (15)$$

where we use a fixed $K = 4$ as specified in [28].

4.3 Choosing a probe burst length

We study next the most suitable probe burst length (L_{Probe}) that provides an accurate $t_{NET_{MAX}}$ estimation while limiting unnecessary energy consumption on the BLE node, caused by probing. We do this empirically by letting an `nrf52`-based BLE node exchange probe/acknowledgment packets once every second with a server in four different environments for eight hours per environments.

Experimental environments. For our measurements, we use the following four different experimental environments:

Wired & No Interf. In this scenario, we use a wired Internet connection on the router with very low variability in combination with no Wi-Fi interference in the BLE subnet.

Wired & Wi-Fi Interf. This scenario uses the same wired Internet connection as above. To generate link-layer problems and, therefore, delays in the BLE subnet, we introduce continuous Wi-Fi interference on two different Wi-Fi channels (sending 1500 bytes of Wi-Fi data every 10 ms with a TX power of 50 mW) using two different co-located Rasperry Pi 3 devices in our testbed running Jamlab-NG [32].

Cellular & No Interf. In this scenario, we use a 4G connection to connect our router to the Internet. Compared to the wired Internet connection, the 4G connection experiences longer and more variable t_{NET} values, as we show in Sect. 2. This scenario does not introduce any Wi-Fi interference.

Cellular & Wi-Fi Interf. This scenario uses the 4G Internet connection and introduces continuous Wi-Fi interference in the BLE subnet. This leads to delays in the BLE subnet, caused by link-layer retransmissions due to Wi-Fi, and on the external network path, caused by the cellular connection.

To evaluate different probing settings, we measure the one-way t_{NET} delay of every sent probe using our NTP-synchronized router and cloud server, as described in Sect. 2. Additionally, the node estimates a t_{NET} value, as described in Sect. 4.1, for every exchanged probe/acknowledgment pair.

We step through the recorded t_{NET} estimates of the node and use Eq. 15 to calculate a new $t_{NET_{MAX}}$ after every t_{NET} estimate. For every new $t_{NET_{MAX}}$ value, we check how many of the future t_{NET} measurements, *i.e.*, the actual one-way t_{NET} measurements, exceed the $t_{NET_{MAX}}$ estimate and are therefore underestimated. For our evaluation we use a very conservative prediction window, *i.e.*, the probing interval (I_{Probe}), of

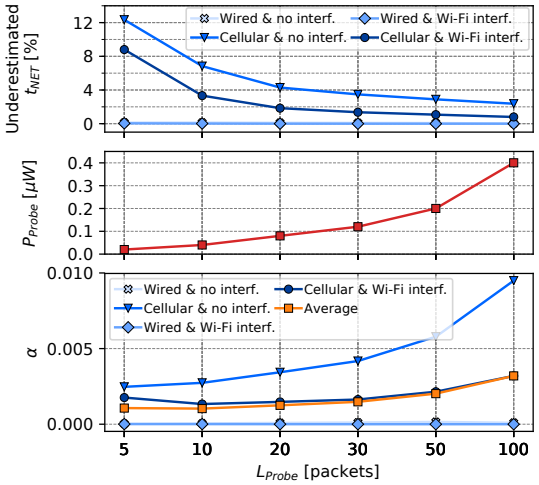


Figure 4. Percentage of underestimated t_{NET} values (top), power consumption P_{Probe} (middle), and cost α (bottom) for different L_{Probe} values in four different environments.

1000 seconds. With this configuration, a node initiates a new probing burst every 1000 seconds to estimate t_{NET}.

Fig. 4 shows the performance of different L_{Probe} values in four different environments. The top of Fig. 4 shows the average number of underestimated future t_{NET} values. For example, when using a cellular Internet connection and no interference, $L_{Probe} = 10$ results in approximately 7% of all future t_{NET} values being underestimated. P_{Probe} shows the calculated average power consumption of a node for probing with different L_{Probe} and a $I_{Probe} = 1000\,s$. The bottom of Fig. 4 shows the cost α for every L_{Probe} as the number of underestimated t_{NET} predictions multiplied by P_{Probe}.

Fig. 4 shows that a short $L_{Probe} = 10$ has the lowest cost α in all four environments, *i.e.*, it provides the most suitable t_{NET} estimation of future packet exchanges while limiting unnecessary energy consumption on the BLE node. For wired Internet connectivity, even $L_{Probe} = 5$ would provide an accurate $t_{NET_{MAX}}$. For cellular Internet connectivity, a larger L_{Probe} would slightly improve the $t_{NET_{MAX}}$ estimation, at the cost of a higher average power draw on the BLE node.

Next, we investigate the performance of our t_{NET} estimation for different I_{Probe} values and $L_{Probe} = 10$. We reuse the recorded data to calculate the percentage of underestimated t_{NET} values, the average power consumption for probing (P_{Probe}), and the cost α of different probing intervals.

Fig. 5 shows that $I_{Probe} = 1000\,s$ performs best in all four experimental environments, as it has the lowest cost α. In our setting, a smaller I_{Probe} does not significantly reduce the percentage of underestimated t_{NET} values. Using a smaller I_{Probe}, however, significantly increases the power consumed by the node for probing t_{NET}.

IoT applications that experience vast and sudden changes in delay across the Internet, *e.g.*, a 4G-based mobile router experiences link quality degradation and needs to change to a 3G backhaul, may use a shorter probe interval (I_{Probe}) at the cost of a higher power consumption of the BLE node.

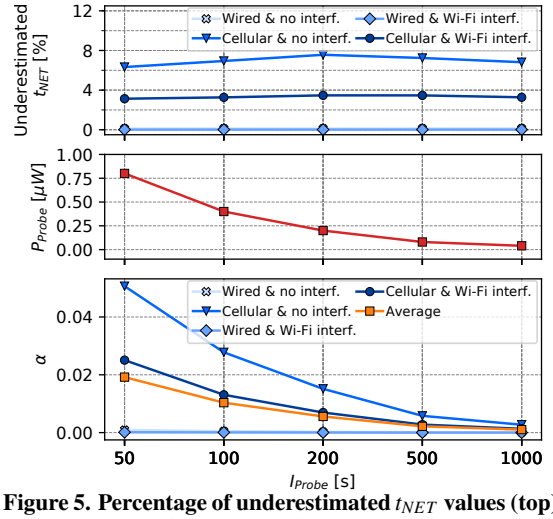


Figure 5. Percentage of underestimated t_{NET} values (top), power consumption P_{Probe} (middle), and cost α (bottom) for different I_{Probe} values in four different environments.

5 Sustaining End-to-End Requirements

In this section, we discuss how a BLE node can use our model (Sect. 3) and t_{NET} estimations (Sect. 4) to sustain a given end-to-end latency and a given end-to-end reliability, while minimizing its power consumption. Therefore, we investigate how a node can cope with network loss (Sect. 5.1) and sustain end-to-end requirements when connected to a router with (Sect. 5.2) or without radio duty cycle constraints (Sect. 5.3). We also discuss how a node can sustain bounds on packet transmissions and receptions (Sect. 5.4).

5.1 Sustaining Reliability

First, we investigate how a BLE node can sustain a given transmission reliability ($\widehat{r}_{MIN}$) by dynamically adapting the number of necessary transmissions of application packets based on the current transmission reliability (r_{NET}) over the network path. As shown in Sect. 2, no packets are dropped within the BLE subnet, due to the autonomous packet retransmission and flow control of the BLE link layer. Since we use the light-weight UDP transport layer, as most constrained IoT applications do [4], packets may be dropped on the Internet. To cope with this loss, we may need to send additional data packets to sustain our given reliability.

Estimating transmission reliability. To estimate the current reliability across the network (r_{NET}), we use a moving average filter with a window length W_{LOSS} on ordinary data transmissions. When we receive an acknowledgment for an application data packet within a timeout $t_{Timeout}$, we count this transmission as successful (r_{PKT} = 1). Otherwise, we assume the transmission has failed (r_{PKT} = 0). After every transmission, the current r_{NET} is calculated as:

$$r_{NET} = \frac{1}{W_{LOSS}} \sum_{j=1}^{W_{LOSS}} r_{PKT}(j). \tag{16}$$

Adapting transmission attempts. By knowing the current r_{NET} value of the network path, we can dynamically adapt the number of application transmission attempts necessary to achieve the given minimum transmission reliability ($\widehat{r}_{MIN}$). Compared to other Internet traffic, our data and acknowledgment packets are short and rather infrequent, which means that the BLE nodes will hardly cause congestion in the Internet path just because of few additional packets.

Similar to other research [39, 40], we assume that packet loss over the network path is independent and identically distributed with a success rate of r_{NET}, which means we can model it with the binomial distribution as:

$$P(X = k) = \binom{n}{k} r_{TX}^{k} (1 - r_{NET})^{n-k}, \tag{17}$$

where $P(X = k)$ is the probability that exactly k out of n transmission attempts are successful.

In our case, we need at least one transmission attempt to have a successful packet exchange between node and server. The end-to-end reliability $\widehat{r}_{MIN}$ can be calculated as:

$$\widehat{r}_{MIN} = P(X \geq 1) = 1 - P(X = 0). \tag{18}$$

By using Eq. 17 and $\binom{n}{0} = 1$, $\widehat{r}_{MIN}$ can be calculated as:

$$\widehat{r}_{MIN} = 1 - (1 - r_{NET})^{n}. \tag{19}$$

For a given $\widehat{r}_{MIN} < 1$ and an estimated r_{NET}, we calculate the number of necessary transmission attempts (N_{TX}) as:

$$N_{TX} = n = \begin{cases} \left\lceil \frac{log(1-\widehat{r}_{MIN})}{log(1-r_{NET})} \right\rceil & \text{if } r_{NET} < 1 \\ 1 & \text{if } r_{NET} = 1 \end{cases} \tag{20}$$

By sending N_{TX} data packet attempts, we can sustain the given application reliability ($\widehat{r}_{MIN}$). Next, we investigate the necessary timing of the N_{TX} transmissions so that the application also sustains a given maximum end-to-end latency.

5.2 Sustaining Latency: Adapting CI & SL

Next, we investigate how a BLE node can sustain a given upper end-to-end latency bound when connected to a router that needs to limit its BLE radio duty cycle.

Some router devices do not have a continuous power supply (*e.g.*, smartphones) and/or need to sustain a BLE connection with a large number of BLE nodes simultaneously. In such cases, it it necessary that a BLE node requires as little BLE radio time on the router as possible. If BLE nodes would require huge portions of the BLE radio duty cycle of the router, *e.g.*, because the router needs to check every 10 ms if the node has data to send, the node would drain the router's battery or would limit the number of simultaneous BLE connections that can be offered on the router.

In these scenarios, a BLE node needs to adapt the BLE connection interval (*CI*) and the BLE slave latency (*SL*) to sustain its latency bounds while limiting its power draw and the radio duty cycle on the router.

5.2.1 *Transmitting Data*

We show how a BLE node can choose suitable *CI* and *SL* values to transmit data with a given end-to-end latency ($\widehat{t}_{TX_{MAX}}$) and a given end-to-end reliability ($\widehat{r}_{MIN}$) to a server.

From Sect. 5.1, we know that we need to transmit N_{TX} application packets within $\widehat{t}_{TX_{MAX}}$ to sustain $\widehat{r}_{MIN}$. Therefore, we split $\widehat{t}_{TX_{MAX}}$ into N_{TX} equal time slots. In each of these time slots one transmission attempt is initiated.

We can now use Eq. 4 to calculate the bound on the connection interval (CI) that allows us to sustain the latency requirements for transmitting packets from node to server as:

$$CI \leq \frac{\widehat{t}_{TX_{MAX}}/N_{TX} - t_{NET_{MAX}} - t_{CE}}{n_{CE_{MAX}} \cdot \lceil D_{TX}/F_{TX} \rceil}. \quad (21)$$

To minimize the power consumption on the BLE node, we can also choose its BLE slave latency (SL) as:

$$SL = n_{CE_{MAX}} \cdot \lceil D_{TX}/F_{TX} \rceil - 1. \quad (22)$$

This allows the node to skip unnecessary connection events, where only mandatory keep-alives would be exchanged.

5.2.2 Receiving Data

In this section, we show how a BLE node can choose suitable CI and SL values to receive data within a given end-to-end latency ($\widehat{t}_{RX_{MAX}}$) and a given end-to-end reliability ($\widehat{r}_{MIN}$) from a server. In this case, the server needs to account for any loss over the Internet and adapt its transmission attempts, as discussed in Sect. 5.1. The node gets N_{RX} from the server, after it has calculated the necessary transmission attempts.

Using Eq. 9, we can calculate the bound on CI that allows to sustain the given end-to-end latency as:

$$CI \leq \frac{\widehat{t}_{RX_{MAX}}/N_{RX} - t_{NET_{MAX}} - t_{CE}}{n_{CE_{MAX}} \cdot \lceil D_{RX}/F_{RX} \rceil + SL}. \quad (23)$$

Furthermore, we choose $SL = 0$ for this scenario. This allows the node to sustain a given upper bound on the reception latency while limiting unnecessary BLE radio time on the router due to skipped connection event.

5.3 Sustaining Latency: Adapting SL Only

In contrast to Sect. 5.2, we next investigate how a BLE node can sustain a given upper end-to-end latency bound on transmitting or receiving messages when the router has no constraints on its BLE radio duty cycle (*i.e.*, the router is not constrained in its power consumption) and scalability (*i.e.*, the number of connected BLE devices) is not an issue.

In this scenario, the router provides the BLE node with the smallest possible BLE connection interval (CI) during BLE connection setup that the router can sustain. The BLE node uses the provided CI and only adapts the BLE slave latency (SL) according to its application latency requirements.

5.3.1 Transmitting Data

Similar to Sect. 5.2.1, the BLE node first calculates the number of necessary data transmission attempts (N_{TX}) to sustain the given application reliability ($\widehat{r}_{MIN}$).

In contrast to Sect. 5.2, where the node needs to calculate a suitable CI, the BLE node already uses the fastest CI possible and only needs to adapt SL to conserve power by limiting unnecessary empty connection events. To limit these unnecessary connection events, we set SL as:

$$SL = \lceil I_{TX}/CI \rceil - 1, \quad (24)$$

where I_{TX} is the interval at which the application is issuing packet transmissions and CI is the used BLE connection interval. $\lceil I_{TX}/CI \rceil$ measures the maximum number of connection events between two data packet transmissions.

Using this SL, the node only needs to wake up when it has data to transmit. During the remaining connection events, the node sleeps for SL events to minimize power draw.

5.3.2 Receiving Data

Similar to the data transmission case, the BLE node uses the CI value provided by the router and only adapts the SL according to the end-to-end timing requirements for receiving packets from the server. To sustain the given end-to-end reception latency, we use Eq. 9 and solve for SL as:

$$SL \leq \frac{\widehat{t}_{RX_{MAX}}/N_{RX} - t_{NET_{MAX}} - t_{CE}}{CI} - n_{CE_{MAX}} \cdot \left\lceil \frac{D_{RX}}{F_{RX}} \right\rceil. \quad (25)$$

Using such a SL allows the node to skip most BLE connection events while sustaining the desired latency bound.

5.4 Discussion

To sustain end-to-end latency bounds on transmitting and receiving packets simultaneously, a node independently calculates suitable parameters for transmitting (CI_{TX} and SL_{TX}) and receiving (CI_{RX} and SL_{RX}) using the formulas above.

To ensure that both end-to-end latency bounds are sustained, the node uses a value of $CI = MIN(CI_{TX}, CI_{RX})$ and $SL = MIN(SL_{TX}, SL_{RX})$ as its BLE connection parameters.

6 Implementation

In this section, we present the implementation of our proposed adaptation strategies on the Nordic Semiconductor `nRF52840 DK` [26]. This platform uses an ARM Cortex-M4F CPU, comes with 1024 kB of flash and 256kB of RAM, and embeds a radio supporting BLE communication up to version 5. While we use the `nRF52840` platform, our implementation also runs on all `nRF52` platform variants.

For our adaptation mechanisms, we use only fully standardized BLE functionality, which means that our implementation can be easily ported to other hardware platforms that support BLE version 4.1 and above, such as the Texas Instruments CC26xx platform.

We use the Zephyr OS [37] for our implementation, because this OS already includes a BLE and IPv6-over-BLE communication stack fully compliant to the BLE specification and uses the standardized HCI to exchange data between the BLE controller and host. We extend Zephyr's IPv6-over-BLE application on the node, which is located on the BLE host, by our proposed adaptation mechanisms. Our node application waits for the router to initiate an IPv6-over-BLE connection. After the connection is initiated, the node transmits a UDP packet with an IPv6 packet length of 128 bytes to the server, which runs a Python UDP server application in an Amazon Web Service (AWS) instance in Frankfurt. As soon as the node receives the first valid server acknowledgment, it starts to probe $t_{NET_{MAX}}$ using a probe burst length of $L_{Probe} = 10$. After a first $t_{NET_{MAX}}$ estimation is available, the node calculates the most suitable CI and SL configuration, as described in Sect. 5, and configures these values by sending a BLE `LL CONNECTION UPDATE REQUEST` to the router.

For both adaptation approaches, we use an ACK timeout $t_{Timeout} = 2000\,ms$ and a probing interval $I_{Probe} = 1000\,s$. We monitor the n_{CE} values of the underlying BLE connection by following the approach presented in [36]. Therefore, we add n_{CE} estimation on the BLE host in the HCI driver layer (`hci_core`) and monitor the `HCI ACL Data Packet` command and `HCI Number of Completed Packets` event to get $t_{TX_{BLE}}$ and n_{CE_f} for every transmitted application packet.

Table 2. Delayed packet and maximum number of subsequently delayed packets (max. delays) of the different node configurations under heavy Wi-Fi interference.

Node config.	delayed [%]	max. delays
CI = 7.5 ms & SL = 0	0.00 ± 0.00	0
Adapt SL only	0.00 ± 0.00	0
Adapt CI & SL	0.61 ± 0.22	2
CI = 1000 ms & SL = 0	50.64 ± 2.64	25

Whenever a new n_{CE_f} value is available, the node application is notified via a callback and can update the BLE connection parameters according to Sect. 5.

The current $n_{CE_{MAX}}$ value of the BLE connection is calculated via a moving maximum filter with a window length of 100 most recent n_{CE_f} measurements, as described in [36]. We further use a lower bound of $n_{CE_{MAX}} = 2$ in our implementation, which means that we slightly overestimate loss over the BLE connection, even when no link-layer errors happen during recent packet transmissions.

Our BLE connections do not make use of BLE data channel blacklisting and use the 2M PHY Mode of BLE.

7 Evaluation

We evaluate our proposed adaptation strategies experimentally. We start by evaluating the detailed behavior of both proposed approaches, which we call `Adapt CI & SL` (Sect. 5.2) and `Adapt SL only` (Sect. 5.3). We do so in the presence of dynamic changes on the network path (Sect. 7.1) and further provide a comparison with other BLE node configurations (Sect. 7.2). Specifically, we compare our approaches against a node using the fastest possible static BLE connection parameters ($CI = 7.5\,ms$ & $SL = 0$) and a node using static and power efficient BLE connection parameters ($CI = 1000\,ms$ & $SL = 0$).

7.1 Systematic Evaluation

To show how our proposed solutions can cope with dynamic changes in the BLE subnet and on the external network path, we focus on a node transmitting packets to the server using a wired Internet connection. All BLE nodes are configured to sustain a $\widehat{r}_{MIN} = 99\%$ and a $\widehat{t}_{TX_{MAX}} = 1000\,ms$.

7.1.1 Changes in the BLE subnet

First, we investigate how a node can adapt to sudden changes (*e.g.*, Wi-Fi interference) in the local BLE subnet.

Setup. We use the experimental setup described in Sect. 2 and start the experiment by setting up the connection between node and server and wait for 60 s until all initial adaptations are done. After this initial phase, we introduce heavy and continuous Wi-Fi interference on two different Wi-Fi channels (sending 1500 bytes of Wi-Fi data every 10 ms with a TX power of 50 mW) using two different Rasperry Pi 3 devices in our testbed running Jamlab-NG [32].

Results. Tab. 2 shows the number of delayed UDP packets (delayed) for each of the four different node configurations over the 10 min after the Wi-Fi jamming was started, across 5 runs for every node configuration. Furthermore, the table shows the maximum number of subsequently delayed packets (max. delays) across all test runs for every configuration.

We can see that both of our proposed adaptation approaches (`Adapt SL only` and `Adapt CI & SL`) are able to effectively cope with link-layer errors in the BLE subnet. While the `Adapt SL only` approach results in no end-to-end packet delays, also the `Adapt CI & SL` approach results in below 1% of all packet transmissions being delayed.

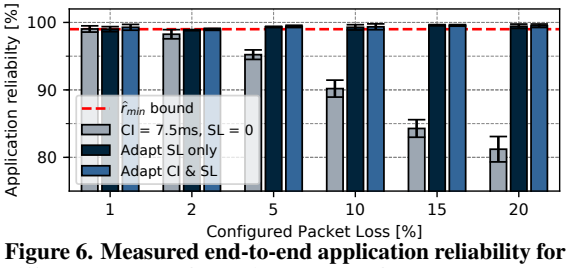


Figure 6. Measured end-to-end application reliability for different node configurations and configured packet loss.

7.1.2 Changes in network loss

Next, we evaluate the performance of our adaptation approaches when the transmission reliability of the external network path (r_{NET}) changes.

Setup. We use the setup from Sect. 2 to establish a connection between node and server and wait for 60 s until all initial adaptation is done. Next, we lower r_{NET} by either 1, 2, 5, 10, 15, or 20% and measure the resulting end-to-end application reliability, *i.e.*, how many node packets are successfully received within $\widehat{t}_{TX_{MAX}}$, for 600 s. To reproducibly lower r_{NET}, we use the standard Traffic Control (`tc`) tool with its Network Emulator (`netem`), which are both part of Linux distributions. We use `tc` on the all outgoing IPv6 packets on the router and the AWS server to mimic symmetric network loss.

Results. Fig. 6 shows the average transmission reliability of our application packets for three different node configurations across 5 experimental runs per configuration. We can see that both of our adaptive approaches successfully sustain the configured $\widehat{r}_{MIN} = 99\%$, by increasing the transmission attempts sent for every application data packet.

7.2 Comparison

We next investigate how our approaches perform in comparison to other node configurations.

Setup. Again, we use our testbed setup (see Sect. 2) to measure end-to-end latency and power consumption of nodes. We program a node with one of the different node configurations, setup the communication between node and server, and initially wait 60 s before we start our data collection.

To measure the performance of the different node configurations, we measure the number of application packets that exceed the specified latency bound (delayed pkts.). We further measure how the different node configurations affect the BLE radio duty cycle of the router (RDC_{Router}) by monitoring GPIO events of our router's USB-BLE radio, which triggers a GPIO event whenever the BLE radio is enabled. To investigate the power efficiency of the different node configurations, we measure the current consumption of the BLE node (I_{Node}) using D-Cube [31]. For this experiments, all log messages and unused peripheral devices on the BLE node are disabled to minimize the overall power consumption.

In addition to the two static and two adaptive node configurations, we also measure the performance of the adaptation approach presented in [36] in this experiment. This

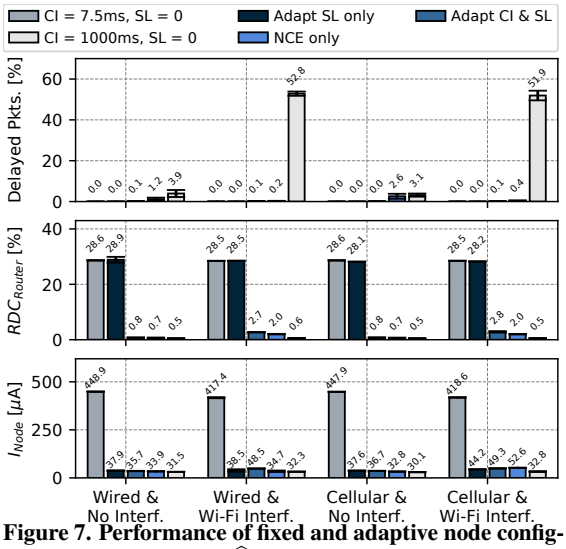


Figure 7. Performance of fixed and adaptive node configurations for sustaining $\widehat{t}_{TX_{MAX}} = 1000ms$ on transmitting packets with a length of 128 bytes to a cloud server.

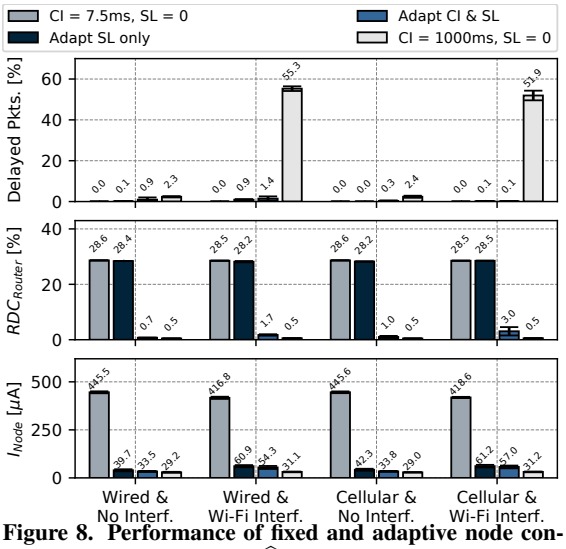


Figure 8. Performance of fixed and adaptive node configurations for sustaining $\widehat{t}_{RX_{MAX}} = 1000ms$ on receiving packets with a length of 128 bytes from a cloud server.

approach, which we call `NCE only`, only reacts to changes in the BLE subnet and does not account for any network delay. As the `NCE only` approach only sustains transmission bounds, it is not included in our packet reception experiment.

Results. Fig. 7 shows the performance of four different node configurations when the node tries to sustain a $\widehat{t}_{TX_{MAX}} = 1000ms$ in the four different experimental environments from Sect. 4.3. Every experimental run was repeated 5 times. We can clearly see that our `Adapt CI & SL` approach results in at most 0.1% of transmission being delayed, while also limiting the router's radio duty cycle and the slaves current consumption. If we follow the `Adapt SL only` approach, no transmissions are delayed. This, however, comes with the cost of at least a factor 10 increase of the RDC on the router. Furthermore, we can see that both of our approaches also significantly outperform the `NCE only` approach in sustaining $\widehat{t}_{TX_{MAX}}$, by at least 100%.

Similarly, Fig. 8 shows the performance of the different node configurations when the node tries to sustain a $\widehat{t}_{RX_{MAX}} = 1000ms$. Also in this setting, we can clearly see that both of our adaptation approaches result in at most 1.4% of packets receptions being delayed, while limiting the power consumption of the BLE node. Similar to Fig. 7, the `Adapt SL only` approach results in less delayed packets than the `Adapt CI & SL` approach, at the cost of a higher RDC_{Router}.

8 Related Work

Cloud-based BLE applications. There exists a vast number of BLE-based applications that communicate with a cloud server in time-critical domains, such as smart-grid [8], smart city [15], or smart health applications [3, 16, 22, 38]. None of these works, however, adapt their communication to delay or loss in the BLE connection or the remaining network path.

Contrary to these works, our paper shows how BLE-based IoT applications can sustain end-to-end reliability and latency bounds for communication between nodes and a server, while limiting the power draw of the BLE devices.

BLE Modeling. Existing BLE timeliness models cannot be used on off-the-shelf BLE devices, as they require BLE link-layer information, such as CRC error count or bit error rate, which are not available to BLE applications [23, 29, 34]. Only a few works [27, 36] make use of information that is available on every standard BLE device to estimate and control the transmission delay on a BLE connection. The major problem with these models, however, is that they only model the transmission delay of packets in a BLE subnet, and hence cannot be used for communication spanning across multiple networks. Furthermore, these models either only investigate the effects of a single BLE connection parameter on communication latency [27, 36, 34] or do not apply to packets that need to be received within given latency bounds [23, 29].

This paper, to the best of our knowledge, presents the first model capturing the effect of all BLE communication parameters on the timeliness of traffic from and to a BLE node.

Dependable low-power wireless communication. A plethora of low-power wireless research studies have investigated how to sustain time-critical communication while minimizing power consumption in IEEE 802.15.4 [14, 17, 42] and WirelessHART applications [7, 10, 24]. These works, however, *only* focus on sustaining latency and reliability within their low-power networks. Some low-power wireless research have investigated dependable communication over multiple networks, but neglect end-to-end timeliness [4, 6] or explicitly exclude communication on the Internet [9].

In this work, instead, we investigate how low-power and constraint nodes can sustain end-to-end dependability requirements on traffic from and to a cloud server.

Estimating Internet characteristics. Capturing and mitigating loss and delays across the Internet has been extensively studied over the last decades. The proposed mech-

anisms to estimate end-to-end delay or loss, however, rely on complex primitives, such as multiple Kalman filters [20], machine learning [30] or two-level markov models [11].

In this paper, we present a simple and efficient approach that accurately estimates the maximum network delay across the Internet. We use the well-established RTT estimation of TCP [18] as a base and revise it to be use on low-power, asymmetric links, such as BLE connections.

Tactile Internet. Research works on the Tactile Internet have investigated how to sustain minimal end-to-end latency and maximum reliability on the Internet [12]. These works, however, focus on optimizing 5G communication to reach high data rates and round trip latencies below 1 ms [21, 33].

Contrary to these studies, our work focus on sustaining given end-to-end dependability requirements while minimizing the power consumption of BLE devices.

9 Conclusion and Future Work

State-of-the-art BLE-based IoT applications are not able to cope with delays or loss along the whole network path between nodes and cloud. In this work, we show how BLE nodes can estimate and mitigate network loss and delay by dynamically adapting their BLE parameters. Using our approaches, nodes are able to sustain end-to-end latency and reliability requirements while minimizing their power draw.

Our next steps include combining our adaptation approaches with sophisticated BLE channel management, such as the work presented in [35], to improve the performance of nodes sustaining end-to-end dependability requirements even further. Furthermore, our adaptation approaches can be used in combination with application-level improvements, such as data reduction, compression, and prediction mechanisms used in Tactile Internet applications [21].

10 Acknowledgments

This work has been performed within the LEAD project "Dependable Internet of Things in Adverse Environments" funded by Graz University of Technology.

11 References

[1] A. Abdel-Hadi and C. Clancy. A Robust Optimal Rate Allocation Algorithm and Pricing Policy for Hybrid Traffic in 4G-LTE. In *Proc. of the* 24^{th} *Int. IEEE PIMRC Symposium*, 2013.

[2] A. Abdel-Hadi et al. A Utility Proportional Fairness Approach for Resource Allocation in 4G-LTE. In *Proc. of the ICNC Conf.*, 2014.

[3] G. Alfian et al. A Personalized Healthcare Monitoring System for Diabetic Patients by Utilizing BLE-Based Sensors and Real-Time Data Processing. *Sensors*, 18(7), 2018.

[4] A. Betzler et al. CoAP Congestion Control for the Internet of Things. *IEEE Communications Magazine*, 54(7), 2016.

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

[6] R. Brummet et al. A Flexible Retransmission Policy for Industrial Wireless Sensor Actuator Networks. In *Proc. of the IEEE ICII Conf*, 2018.

[7] Y. Chen et al. Probabilistic Per-Packet Real-Time Guarantees for Wireless Networked Sensing and Control. *IEEE Transactions on Industrial Informatics*, 14(5), 2018.

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

[9] DetNet Working Group. Derministic Networking (detnet), 2020.

[10] B. Dezfouli et al. Real-time Communication in Low-Power Mobile Wireless Networks. In *Proc. of the 13th IEEE CCNC Conf.*, 2016.

[11] M. Ellis et al. A two-level Markov model for packet loss in UDP/IP-based real-time video applications targeting residential users. *Computer Networks*, 70, 2014.

[12] G. Fettweis. The Tactile Internet: Applications and Challenges. *IEEE Vehicular Technology Magazine*, 9(1), 2014.

[13] S. Forconi and A. Vizzarri. Review of Studies on End-to-End QoS in LTE Networks. In *AEIT Annual Conference 2013*. IEEE, 2013.

[14] G. Franchino and G. Buttazzo. A power-aware MAC layer protocol for real-time communication in wireless embedded systems. *Journal of Network and Computer Applications*, 82, 2017.

[15] S. Geetha and S. Gouthami. Internet of things enabled real time water quality monitoring system. *Smart Water*, 2(1), 2016.

[16] M. Hasan et al. Real-time healthcare data transmission for remote patient monitoring in patch-based hybrid OCC/BLE networks. *Sensors*, 19(5), 2019.

[17] R. Jacob et al. End-to-end Real-time Guarantees in Wireless Cyber-physical Systems. In *Proc. of the 2016 IEEE RTSS Symposium*, 2016.

[18] V. Jacobson. Congestion Avoidance and Control. In *Proc. of the 1988 SIGCOMM Symposium*, 1988.

[19] K. Jacobsson et al. Estimation of RTT and Bandwidth for Congestion Control Applications in Communication Networks. In *IEEE CDC*, 2004.

[20] K. Jacobsson et al. Some Modeling and Estimation Issues in Control of Heterogeneous Networks. In *MTNS*, 2004.

[21] K. K. Antonakoglou et al. Toward Haptic Communications Over the 5G Tactile Internet. *IEEE Communications Surveys & Tutorials*, 2018.

[22] P. Kakria, N. Tripathi, and P. Kitipawang. A real-time health monitoring system for remote cardiac patients using smartphone and wearable sensors. *International Journal of telemedicine and applications*, 2015.

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

[24] Y. Ma et al. Optimal Dynamic Scheduling of Wireless Networked Control Systems. In *Proc. of the* 10^{th} *Int. ACM/IEEE CPS Conf.*, 2019.

[25] J. Nieminen et al. RFC 7668 - IPv6 over Bluetooth Low Energy, 2015.

[26] Nordic Semiconductors. nRF52840 Specifications, 2020.

[27] E. Park et al. AdaptaBLE: Adaptive Control of Data Rate, Transmission Power, and Connection Interval in Bluetooth Low Energy. *Computer Networks*, 2020.

[28] V. Paxson et al. RFC6298: Computing TCP's Retransmission Timer, 2011.

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

[30] P. S. Rossi et al. Joint End-to-End Loss-Delay Hidden Markov Model for Periodic UDP Traffic Over the Internet. *IEEE Transactions on Signal Processing*, 54(2), 2006.

[31] M. Schuß et al. Moving Beyond Competitions: Extending D-Cube to Seamlessly Benchmark Low-Power Wireless Systems. In *Proc. of the* 1^{st} *CPSBench Worksh.*, 2018.

[32] M. Schuß et al. JamLab-NG: Benchmarking Low-Power Wireless Protocols under Controllable and Repeatable Wi-Fi Interference. In *Proc. of the* 16^{th} *EWSN Conf.*, 2019.

[33] M. Simsek et al. 5G-enabled Tactile Internet. *IEEE Journal on Selected Areas in Communications*, 34(3), 2016.

[34] M. Spörk et al. BLEach: Exploiting the Full Potential of IPv6 over BLE in Constrained Embedded IoT Devices. In *Proc. of the* 15^{th} *ACM SenSys Conf.*, 2017.

[35] M. Spörk et al. Improving the Reliability of Bluetooth Low Energy Connections. In *Proc. of the* 17^{th} *Int. EWSN Conf.*, 2020.

[36] M. Spörk et al. Improving the Timeliness of Bluetooth Low Energy in Dynamic RF Environments. *ACM Transactions on IoT*, 2020.

[37] The Zephyr Project. Zephyr OS: An RTOS for IoT, 2020.

[38] F. Touati, R. Tabish, and A. B. Mnaouer. A Real-Time BLE enabled ECG System for Remote Monitoring. *APCBEE procedia*, 7, 2013.

[39] K. Winstein et al. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *Proc. of the* 10^{th} *USENIX NSDI Symposium*, 2013.

[40] F. Yan et al. Pantheon: the training ground for Internet congestion-control research. In *Proc. of the 2018 USENIX ATC Conf.*, 2018.

[41] Y. Zaki et al. Adaptive Congestion Control for Unpredictable Cellular Networks. In *Proc. of the 2015 ACM SIGDC Conf.*, 2015.

[42] M. Zimmerling et al. Adaptive Real-Time Communication for Wireless Cyber-Physical Systems. *ACM Transactions on CPS*, 1(2), 2017.

Bibliography

[1] A. Abdel-Hadi and C. Clancy. A Robust Optimal Rate Allocation Algorithm and Pricing Policy for Hybrid Traffic in 4G-LTE. In *2013 IEEE 24th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2013.

[2] A. Abdel-Hadi and C. Clancy. A Utility Proportional Fairness Approach for Resource Allocation in 4G-LTE. In *2014 International Conference on Computing, Networking and Communications (ICNC)*, 2014.

[3] H. Aghajan, J. C. Augusto, C. Wu, P. McCullagh, and J.-A. Walkden. Distributed vision-based accident management for assisted living. In *International Conference on Smart Homes and Health Telematics*, pages 196–205. Springer, 2007.

[4] D. Agnoletto, M. Jonsson, and E. P. de Freitas. Time slot transmission scheme with packet prioritization for bluetooth low energy devices used in real-time applications. *International Journal of Wireless Information Networks*, 27(4):518–534, 2020.

[5] A. Ahad, M. Tahir, M. Aman Sheikh, K. I. Ahmed, A. Mughees, and A. Numani. Technologies trend towards 5G network for smart health-care using IoT: A review. *Sensors*, 20(14):4047, 2020.

[6] M. Al Kalaa et al. Bluetooth standard v4. 1: Simulating the Bluetooth low energy data channel selection algorithm. In *Globecom Workshops*. IEEE, 2014.

[7] M. Al Kalaa et al. Selection probability of data channels in Bluetooth Low Energy. In *Proc. of the* 11^{th} *IWCMC Conf.* IEEE, 2015.

[8] M. O. Al Kalaa et al. Evaluating bluetooth low energy in realistic wireless environments. In *Proc. of the* 12^{th} *IWCMC Conf.* IEEE, 2016.

[9] B. Al Nahas, S. Duquennoy, and O. Landsiedel. Concurrent Transmissions for Multi-Hop Bluetooth 5. In *Proc. of the* 16^{th} *EWSN Conf.*, 2019.

[10] G. Alfian, M. Syafrudin, M. F. Ijaz, M. A. Syaekhoni, N. L. Fitriyani, and J. Rhee. A Personalized Healthcare Monitoring System for Diabetic Patients by Utilizing BLE-Based Sensors and Real-Time Data Processing. *Sensors*, 18(7), 2018.

[11] Android Authority. A quick history of Bluetooth. `https://www.androidauthority.com/history-bluetooth-explained-846345/`, 2018. Accessed on 09/03/2021.

[12] Apache Software Foundation. Apache MyNewt: BLE User Guide, 2021.

[13] L. Arden Media Company. Bluetooth basics and how it's used in smart buildings. `https://inbuildingtech.com/buildings/bluetooth-smart-buildings/`, 2018. Accessed on 01/03/2021.

[14] A. A. Ateya, A. Muthanna, I. Gudkova, A. Abuarqoub, A. Vybornova, and A. Koucheryavy. Development of intelligent core network for tactile internet and future smart systems. *Journal of Sensor and Actuator Networks*, 7(1):1, 2018.

[15] N. Baccour, A. Koubâa, L. Mottola, M. A. Zúñiga, H. Youssef, C. A. Boano, and M. Alves. Radio Link Quality Estimation in Wireless Sensor Networks: a Survey. *ACM Transactions*

on Sensor Networks, 8(4), 2012.

[16] M. Baddeley, A. Aijaz, U. Raza, A. Stanoev, Y. Jin, M. Schuß, C. A. Boano, and G. Oikonomou. 6TiSCH++ With Bluetooth 5 and Concurrent transmissions. In *Proceedings of the* 18^{th} *International Conference on Embedded Wireless Systems and Networks (EWSN)*. Junction Publishing, Feb. 2021.

[17] A. Baid and D. Raychaudhuri. Understanding channel selection dynamics in dense Wi-Fi networks. *IEEE Communications Magazine*, 53(1):110–117, 2015.

[18] Y. Baozhou and Z. Qi. A QoS-based channel allocation and power control algorithm for device-to-device communication underlaying cellular networks. *Journal of Communications*, 11(7):624–631, 2016.

[19] P. Baracca, L. G. Giordano, A. Garcia-Rodriguez, G. Geraci, and D. López-Pérez. Downlink performance of uplink fractional power control in 5G massive MIMO systems. In *2018 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2018.

[20] B. Bellalta. IEEE 802.11 ax: High-efficiency WLANs. *IEEE Wireless Communications*, 23(1):38–46, 2016.

[21] A. Betzler et al. CoAP Congestion Control for the Internet of Things. *IEEE Communications Magazine*, 54(7), 2016.

[22] Bluetooth SIG. Specification of the Bluetooth System – Covered Core Package version: 4.0. `https://www.bluetooth.com/specifications/specs/core-specification-4-0/`, 2010.

[23] Bluetooth SIG. Specification of the Bluetooth System – Covered Core Package version: 4.1. `https://www.bluetooth.com/specifications/specs/core-specification-4-1/`, 2013.

[24] Bluetooth SIG. Bluetooth Core Specification v5.0. `https://www.bluetooth.com/specifications/specs/core-specification-5/`, 2016.

[25] Bluetooth SIG. Bluetooth Core Specification v5.1. `https://www.bluetooth.com/specifications/specs/core-specification-5-1/`, 2019.

[26] Bluetooth SIG. Bluetooth Core Specification v5.2. `https://www.bluetooth.com/specifications/specs/core-specification-5-2/`, 2019.

[27] Bluetooth SIG. Bluetooth Core Specification v5.3. `https://www.bluetooth.com/specifications/specs/core-specification/`, 2019.

[28] Bluetooth SIG. Bluetooth Mesh Networking - An Introduction for Developers. `https://www.bluetooth.com/wp-content/uploads/2019/03/Mesh-Technology-Overview.pdf`, 2020.

[29] Bluetooth Studio. History of Bluetooth - the Evolution of Bluetooth Technology. `https://bluetoothstudio.com/history-of-bluetooth/`, 2016. Accessed on 09/03/2021.

[30] C. A. Boano and K. Römer. External Radio Interference. In *Radio Link Quality Estimation in Low-Power Wireless Networks*, SpringerBriefs in Electrical and Computer Engineering - Cooperating Objects. 2013.

[31] C. A. Boano, T. Voigt, C. Noda, K. Römer, and M. A. Zúñiga. JamLab: Augmenting Sen-

sornet Testbeds with Realistic and Controlled Interference Generation. In *Proceedings of the* 10^{th} *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, Apr. 2011.

[32] W. Bober and C. J. Bleakley. BailighPulse: A Low Duty Cycle Data Gathering Protocol for Mostly-off Wireless Sensor Networks. *Computer Networks*, 69, 2014.

[33] S. Böcker, C. Arendt, and C. Wietfeld. On the suitability of Bluetooth 5 for the Internet of Things: Performance and scalability analysis. In *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–7. IEEE, 2017.

[34] P. Bonnet, A. Beaufour, M. B. Dydensborg, and M. Leopold. Bluetooth-based sensor networks. *ACM SIGMOD Record*, 32(4):35–40, 2003.

[35] M. Bor and U. Roedig. LoRa transmission parameter selection. In *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 27–34. IEEE, 2017.

[36] L. Botler, M. Spörk, K. Diwold, and K. Römer. Direction Finding with UWB and BLE: A Comparative Study. In *2020 IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 44–52. IEEE, 2020.

[37] Brian T. Horowitz. IoT Makes Fire Detection Systems Smarter - IEEE Spectrum. `https://spectrum.ieee.org/tech-talk/sensors/remote-sensing/how-iot-makes-fire-detection-systems-smarter`, 2020. Accessed on 24/02/2021.

[38] W. Bronzi et al. Bluetooth low energy performance and robustness analysis for inter-vehicular communications. *Ad Hoc Networks*, 2016.

[39] R. Brummet et al. A Flexible Retransmission Policy for Industrial Wireless Sensor Actuator Networks. In *Proc. of the IEEE ICII Conf*, 2018.

[40] N. Burri, P. von Rickenbach, and R. Wattenhofer. Dozer: Ultra-low Power Data Gathering in Sensor Networks. In *Proceedings of the* 6^{th} *ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN)*, 2007.

[41] C. Gomez and S. Darroudi and T. Savolainen and M. Spörk. IETF RFC Draft - IPv6 Mesh over BLUETOOTH(R) Low Energy using IPSP. `https://datatracker.ietf.org/doc/draft-ietf-6lo-blemesh/`, 2021.

[42] O. Carhacioglu et al. Time-domain cooperative coexistence of BLE and IEEE 802.15.4 networks. In *Proc. of the* 28^{th} *PIMRC Symp.*, 2017.

[43] K.-H. Chang. Bluetooth: a viable solution for IoT?[Industry Perspectives]. *IEEE Wireless Communications*, 21(6):6–7, 2014.

[44] L.-J. Chen, R. Kapoor, K. Lee, M. Sanadidi, and M. Gerla. Audio streaming over Bluetooth: an adaptive ARQ timeout approach. In *24th International Conference on Distributed Computing Systems Workshops, 2004. Proceedings.*, pages 196–201, 2004.

[45] Y. Chen et al. Probabilistic Per-Packet Real-Time Guarantees for Wireless Networked Sensing and Control. *IEEE Transactions on Industrial Informatics*, 14(5), 2018.

[46] A. Chiumento, B. Reynders, Y. Murillo, and S. Pollin. Building a connected BLE mesh: A

network inference study. In *2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 296–301. IEEE, 2018.

[47] K. Cho, W. Park, M. Hong, G. Park, W. Cho, J. Seo, and K. Han. Analysis of Latency Performance of Bluetooth Low Energy (BLE) Networks. *Sensors*, 15(1), 2014.

[48] J. Classen, M. Spörk, C. A. Boano, K. Römer, and M. Hollick. Analyzing Bluetooth Low Energy Connections on Off-the-Shelf Devices. In *Proceedings of the* 17^{th} *International Conference on Embedded Wireless Systems and Networks (EWSN), demo session*. Junction Publishing, Feb. 2020.

[49] M. Collotta et al. A Solution Based on Bluetooth Low Energy for Smart Home Energy Management. *Energies*, 8, 2015.

[50] M. Collotta et al. Bluetooth 5: A concrete step forward toward the IoT. *IEEE Communications Magazine*, (7), 2018.

[51] Contiki-NG. Contiki-NG - Texas Instruments CC2650. `https://github.com/contiki-ng/contiki-ng/tree/master/arch/cpu/cc26x0-cc13x0`, 2021. Accessed on 10/06/2021.

[52] Contiki-NG. Contiki-NG for nRF52 Development Kit. `https://github.com/contiki-ng/contiki-ng/wiki/Platform-nrf52dk`, 2021. Accessed on 10/06/2021.

[53] S. M. Darroudi and C. Gomez. Bluetooth Low Energy Mesh networks: A survey. *Sensors*, 17(7):1467, 2017.

[54] S. M. Darroudi, C. Gomez, and J. Crowcroft. Bluetooth Low Energy Mesh Networks: A Standards Perspective. *IEEE Communications Magazine*, 58(4):95–101, 2020.

[55] A. Dementyev, S. Hodges, S. Taylor, and J. Smith. Power Consumption Analysis of Bluetooth Low Energy, ZigBee and ANT Sensor Nodes in a Cyclic Sleep Scenario. In *Proceedings of the* 1^{st} *IEEE International Wireless Symposium (IWS)*, 2013.

[56] D.-J. Deng, S.-Y. Lien, J. Lee, and K.-C. Chen. On quality-of-service provisioning in IEEE 802.11 ax WLANs. *IEEE Access*, 4:6086–6104, 2016.

[57] DetNet Working Group. Derministic Networking (detnet), 2020.

[58] B. Dezfouli et al. Real-time Communication in Low-Power Mobile Wireless Networks. In *Proc. of the 13th IEEE CCNC Conf.*, 2016.

[59] K. Diaz, D. Krupka, M. Chang, J. Peacock, Y. Ma, J. Goldsmith, J. Schwartz, and K. Davidson. Fitbit: An Accurate and Reliable Device for Wireless Physical Activity Tracking. *International Journal of Cardiology*, 185, 2015.

[60] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira. PCC: Re-architecting congestion control for consistent high performance. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 395–408, 2015.

[61] W. Dong, C. Chen, X. Liu, Y. He, Y. Liu, J. Bu, and X. Xu. Dynamic packet length control in wireless sensor networks. *IEEE Transactions on wireless communications*, 13(3):1172–1181, 2014.

[62] W. Dong, J. Yu, and P. Zhang. Exploiting error estimating codes for packet length adaptation in low-power wireless networks. *IEEE Transactions on Mobile Computing*, 14(8):1601–

1614, 2014.

[63] P. Du et al. Adaptive time slotted channel hopping for wireless sensor networks. In *Proc. of the* 4^{th} *CEEC Conf.*, 2012.

[64] A. Dunkels. uIP-A free small TCP/IP stack. Technical report, 2002.

[65] A. Dunkels. The ContikiMAC Radio Duty Cycling Protocol. Technical Report T2011:13, Swedish Institute of Computer Science, 2011.

[66] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proc. of the* 1^{st} *EmNetS Workshop*, 2004.

[67] S. Duquennoy, B. A. Nahas, O. Landsiedel, and T. Watteyne. Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH. In *Proc. of the* 13^{th} *ACM SenSys Conference*, 2015.

[68] M. H. Dwijaksara, W. S. Jeon, and D. G. Jeong. A Channel Access Scheme for Bluetooth Low Energy to Support Delay-Sensitive Applications. In *Proc. of the* 27^{th} *Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2016.

[69] M. Ellis et al. A two-level Markov model for packet loss in UDP/IP-based real-time video applications targeting residential users. *Computer Networks*, 70, 2014.

[70] A. Elsts, X. Fafoutis, R. Piechocki, and I. Craddock. Adaptive channel selection in IEEE 802.15. 4 TSCH networks. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6. IEEE, 2017.

[71] Enterprise IoT Insights. Football trials real-time BLE-based tracking and analytics ahead of new season. `https://enterpriseiotinsights.com/20200630/channels/news/football-trials-real-time-ble-based-tracking-and-analytics`, 2020. Accessed on 24/02/2021.

[72] R. Faragher and R. Harle. An analysis of the accuracy of bluetooth low energy for indoor positioning applications. In *Proceedings of the 27th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2014)*, pages 201–210, 2014.

[73] R. Faragher and R. Harle. Location Fingerprinting With Bluetooth Low Energy Beacons. *IEEE Journal on Selected Areas in Communications*, 33(11):2418–2428, 2015.

[74] E. Fernández de Gorostiza, J. Berzosa, J. Mabe, and R. Cortiñas. A method for dynamically selecting the best frequency hopping technique in industrial wireless sensor network applications. *Sensors*, 18(2):657, 2018.

[75] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Low-power wireless bus. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, pages 1–14, 2012.

[76] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with Glossy. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 73–84. IEEE, 2011.

[77] G. Fettweis. The Tactile Internet: Applications and Challenges. *IEEE Vehicular Technology Magazine*, 9(1), 2014.

[78] E.-B. Fgee, J. D. Kenney, W. J. Phillips, W. Robertson, and S. Sivakumar. Compari-

son of QoS performance between IPv6 QoS management model and IntServ and DiffServ QoS models. In *3rd Annual Communication Networks and Services Research Conference (CNSR'05)*, pages 287–292. IEEE, 2005.

[79] S. Forconi and A. Vizzarri. Review of Studies on End-to-End QoS in LTE Networks. In *AEIT Annual Conference 2013*. IEEE, 2013.

[80] G. Franchino and G. Buttazzo. A power-aware MAC layer protocol for real-time communication in wireless embedded systems. *Journal of Network and Computer Applications*, 82, 2017.

[81] R. Friedman, A. Kogan, and Y. Krivolapov. On power and throughput tradeoffs of wifi and bluetooth in smartphones. *IEEE Transactions on Mobile Computing*, 12(7):1363–1376, 2012.

[82] J. Fürst, K. Chen, M. Aljarrah, and P. Bonnet. Leveraging Physical Locality to Integrate Smart Appliances in Non-Residential Buildings with Ultrasound and Bluetooth Low Energy. In *Proc. of the* 1^{st} *IEEE IoTDI Conference*, 2016.

[83] S. Geetha and S. Gouthami. Internet of things enabled real time water quality monitoring system. *Smart Water*, 2(1), 2016.

[84] M. Ghamari et al. A Survey on Wireless Body Area Networks for eHealthcare Systems in Residential Environments. volume 16, 2016.

[85] D. Giovanelli, B. Milosevic, C. Kiraly, A. Murphy, and E. Farella. Dynamic group management with Bluetooth Low Energy. In *Proc. of the* 2^{nd} *IEEE ISC2 Conference*, 2016.

[86] R. D. Gomes, D. V. Queiroz, A. C. Lima Filho, I. E. Fonseca, and M. S. Alencar. Real-time link quality estimation for industrial wireless sensor networks using dedicated nodes. *Ad Hoc Networks*, 59:116–133, 2017.

[87] C. Gomez, J. Oller, and J. Paradells. Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology. *Sensors*, 12(9), 2012.

[88] B. Großwindhager, C. A. Boano, M. Rath, and K. Römer. Enabling runtime adaptation of physical layer settings for dependable UWB communications. In *2018 IEEE 19th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pages 01–11. IEEE, 2018.

[89] J. W. Guck and W. Kellerer. Achieving end-to-end real-time quality of service with software defined networking. In *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, pages 70–76. IEEE, 2014.

[90] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating latency between arbitrary internet end hosts. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 5–18, 2002.

[91] N. K. Gupta. *Inside Bluetooth low energy*. Artech House, 2016.

[92] Haolin Wang, Minjun Xi, Jia Liu, and Canfeng Chen. Transmitting IPv6 packets over Bluetooth low energy based on BlueZ. In *2013 15th International Conference on Advanced Communications Technology (ICACT)*, pages 72–77, 2013.

[93] M. Hasan et al. Real-time healthcare data transmission for remote patient monitoring in patch-based hybrid OCC/BLE networks. *Sensors*, 19(5), 2019.

[94] R. Hermeto et al. Passive Link Quality Estimation for Accurate and Stable Parent Selection in Dense 6TiSCH Networks. In *Proc. of the* 15^{th} *EWSN Conf.*, 2018.

[95] Á. Hernández-Solana, D. Perez-Diaz-de Cerio, A. Valdovinos, and J. L. Valenzuela. Proposal and evaluation of BLE discovery process based on new features of Bluetooth 5.0. *Sensors*, 17(9):1988, 2017.

[96] R. Hofmann. X-Burst: Cross-Technology Communication for Off-the-Shelf IoT Devices. Master's thesis, Graz University of Technology, 2018.

[97] Honeywell. White Paper - Choosing the Right Industrial Wireless Network. `https://www.honeywellprocess.com/library/support/Public/Documents/WirelessWhitePaper_Nov2006.pdf`, 2006. Accessed on 24/02/2021.

[98] D. Hortelano, T. Olivares, M. C. Ruiz, C. Garrido-Hidalgo, and V. López. From sensor networks to internet of things. Bluetooth low energy, a standard for this evolution. *Sensors*, 17(2):372, 2017.

[99] S. Hussain, S. Mehnaz, S. Nirjon, and E. Bertino. SeamBlue: Seamless Bluetooth Low Energy Connection Migration for Unmodified IoT Devices. In *Proc. of the* 14^{th} *EWSN Conference*, 2017.

[100] B. Islam, M. Uddin, S. Mukherjee, and S. Nirjon. Rethinking Ranging of Unmodified BLE Peripherals in Smart City Infrastructure. In *Proc. of the* 9^{th} *ACM Multimedia Systems Conference (MMSys)*, 2018.

[101] E. J. Hui, A. R. Corporation, P. Thubert, and Cisco. RFC 6282 - Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. `https://tools.ietf.org/html/rfc6282`, 2011.

[102] R. Jacob et al. End-to-end Real-time Guarantees in Wireless Cyber-physical Systems. In *Proc. of the 2016 IEEE RTSS Symposium*, 2016.

[103] V. Jacobson. Congestion Avoidance and Control. In *Proc. of the 1988 SIGCOMM Symposium*, 1988.

[104] K. Jacobsson et al. Estimation of RTT and Bandwidth for Congestion Control Applications in Communication Networks. In *IEEE CDC*, 2004.

[105] K. Jacobsson et al. Some Modeling and Estimation Issues in Control of Heterogeneous Networks. In *MTNS*, 2004.

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

[107] C. Julien, C. Liu, A. Murphy, and G. Picco. BLEnd: Practical Continuous Neighbor Discovery for Bluetooth Low Energy. In *Proc. of the* 16^{th} *ACM/IEEE IPSN Conference*, 2017.

[108] K. K. Antonakoglou et al. Toward Haptic Communications Over the 5G Tactile Internet. *IEEE Communications Surveys & Tutorials*, 2018.

[109] P. Kakria, N. Tripathi, and P. Kitipawang. A real-time health monitoring system for remote cardiac patients using smartphone and wearable sensors. *International Journal of telemedicine and applications*, 2015.

[110] G. Kalantar, A. Mohammadi, and S. N. Sadrieh. Analyzing the effect of bluetooth low energy (BLE) with randomized MAC addresses in IoT applications. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 27–34. IEEE, 2018.

[111] M. Karakus and A. Durresi. Quality of service (QoS) in software defined networking (SDN): A survey. *Journal of Network and Computer Applications*, 80:200–218, 2017.

[112] H. Karvonen, K. Mikhaylov, M. Hämäläinen, J. Iinatti, and C. Pomalaza-Ráez. Interference of Wireless Technologies on BLE Based WBANs in Hospitals. In *Proc. of the Symposium on Personal, Indoor, and Mobile Radio Communications*, 2017.

[113] H. Karvonen, C. Pomalaza-Ráez, K. Mikhaylov, M. Hämäläinen, and J. Iinatti. Experimental performance evaluation of ble 4 versus ble 5 in indoors and outdoors scenarios. In *Advances in Body Area Networks I*, pages 235–251. Springer, 2019.

[114] E. Khorov, A. Kiryanov, A. Lyakhov, and G. Bianchi. A tutorial on IEEE 802.11 ax high efficiency WLANs. *IEEE Communications Surveys & Tutorials*, 21(1):197–216, 2018.

[115] P. Kindt, M. Saur, and S. Chakraborty. Neighbor Discovery Latency in BLE-Like Duty-Cycled Protocols. *arXiv preprint arXiv:1509.04366*, 2015.

[116] P. Kindt, D. Yunge, M. Gopp, and S. Chakraborty. Adaptive Online Power-Management for Bluetooth Low Energy. In *Proc. of the IEEE INFOCOM Conference*, 2015.

[117] P. H. Kindt, M. Saur, M. Balszun, and S. Chakraborty. Neighbor discovery latency in BLE-like protocols. *IEEE Transactions on Mobile Computing*, 17(3):617–631, 2017.

[118] P. H. Kindt, D. Yunge, R. Diemer, and S. Chakraborty. Energy Modeling for the Bluetooth Low Energy Protocol. *To Appear in the ACM Transactions on Embedded Computing Systems (TECS)*, 2020.

[119] Knapp AG. Autonomous mobile robots give logistics at a new edge. `https://www.knapp.com/en/blogposts/open-shuttles-autonomous-mobile-robots-for-flexible-processes/`, 2020. Accessed on 24/02/2021.

[120] V. Kotsiou et al. Is local blacklisting relevant in slow channel hopping low-power wireless networks? In *Proc. of the ICC Conf.* IEEE, 2017.

[121] V. Kotsiou et al. LABeL: Link-based adaptive blacklisting technique for 6TiSCH wireless industrial networks. In *Proc. of MSWiM*, 2017.

[122] V. Kotsiou, G. Z. Papadopoulos, D. Zorbas, P. Chatzimisios, and F. Theoleyre. Blacklisting-based channel hopping approaches in low-power and lossy networks. *IEEE Communications Magazine*, 57(2):48–53, 2019.

[123] G. Ku and J. M. Walsh. Resource allocation and link adaptation in LTE and LTE advanced: A tutorial. *IEEE communications surveys & tutorials*, 17(3):1605–1633, 2014.

[124] S. Kumar, M. P. Andersen, H.-S. Kim, and D. E. Culler. Performant TCP for Low-Power Wireless Networks . In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 911–932, Santa Clara, CA, Feb. 2020. USENIX Association.

[125] M. Lacage, M. H. Manshaei, and T. Turletti. IEEE 802.11 rate adaptation: a practical

approach. In *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 126–134, 2004.

[126] E. A. Lee and D. G. Messerschmitt. Digital communication. 1994.

[127] T. Lee, J. Han, M.-S. Lee, H.-S. Kim, and S. Bahk. CABLE: connection interval adaptation for BLE in dynamic wireless environments. In *Proc. of the* 14^{th} *IEEE Conference on Sensing, Communication, and Networking (SECON)*, 2017.

[128] T. Lee, M. S. Lee, H. S. Kim, and S. Bahk. A Synergistic Architecture for RPL over BLE. In *Proc. of the* 13^{th} *IEEE SECON Conference*, 2016.

[129] L. Leonardi, G. Patti, and L. L. Bello. Multi-hop real-time communications over bluetooth low energy industrial wireless mesh networks. *IEEE Access*, 6:26505–26519, 2018.

[130] M. Leopold, M. B. Dydensborg, and P. Bonnet. Bluetooth and sensor networks: A reality check. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 103–113, 2003.

[131] P. Li, T. Vermeulen, H. Liy, and S. Pollin. An adaptive channel selection scheme for reliable TSCH-based communication. In *2015 International Symposium on Wireless Communication Systems (ISWCS)*, pages 511–515. IEEE, 2015.

[132] Z. Li, M. A. Uusitalo, H. Shariatmadari, and B. Singh. 5G URLLC: Design challenges and system concepts. In *2018 15th International Symposium on Wireless Communication Systems (ISWCS)*, pages 1–6. IEEE, 2018.

[133] C.-J. M. Liang, N. B. Priyantha, J. Liu, and A. Terzis. Surviving wi-fi interference in low power zigbee networks. In *Proceedings of the 8th ACM conference on embedded networked sensor systems*, pages 309–322, 2010.

[134] S. Lin, F. Miao, J. Zhang, G. Zhou, L. Gu, T. He, J. A. Stankovic, S. Son, and G. J. Pappas. ATPC: adaptive transmission power control for wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 12(1):1–31, 2016.

[135] J. Liu, C. Chen, and Y. Ma. Modeling and Performance Analysis of Device Discovery in Bluetooth Low Energy Networks. In *Global Communications Conference (GLOBECOM), 2012 IEEE*. IEEE, 2012.

[136] J. Liu, C. Chen, Y. Ma, and Y. Xu. Energy Analysis of Device Discovery for Bluetooth Low Energy. In *Proc. of the* 78^{th} *IEEE VTC Fall Conference*. IEEE, 2013.

[137] T. Liu and A. E. Cerpa. Foresee (4C): Wireless link prediction using link features. In *Proc. of the* 10^{th} *IPSN Conf.*, 2011.

[138] Y. Ma et al. Optimal Dynamic Scheduling of Wireless Networked Control Systems. In *Proc. of the* 10^{th} *Int. ACM/IEEE CPS Conf.*, 2019.

[139] J. Mack, S. Gazor, A. Ghasemi, and J. Sydor. Dynamic channel selection in cognitive radio WiFi networks: An experimental evaluation. In *2014 IEEE International Conference on Communications Workshops (ICC)*, pages 261–267. IEEE, 2014.

[140] I. Mahadevan and K. M. Sivalingam. Quality of service architectures for wireless networks: IntServ and DiffServ models. In *Proceedings Fourth International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'99)*, pages 420–425. IEEE, 1999.

[141] H. Malik, H. Pervaiz, M. M. Alam, Y. Le Moullec, A. Kuusik, and M. A. Imran. Radio

resource management scheme in NB-IoT systems. *IEEE Access*, 6:15051–15064, 2018.

[142] D. Mantz et al. InternalBlue - Bluetooth Binary Patching and Experimentation Framework. In *Proc. of the* 17^{th} *MobiSys Conf.*, 2019.

[143] A. Marinčić et al. Interoperability of IoT wireless technologies in ambient assisted living environments. In *Proc. of the WTS Symp.*, 2016.

[144] M. Marinoni, A. Biondi, P. Buonocunto, G. Franchino, D. Cesarini, and G. Buttazzo. Real-time analysis and design of a dual protocol support for Bluetooth LE devices. *IEEE Transactions on Industrial Informatics*, 13(1):80–91, 2016.

[145] B. Martinez, F. Adelantado, A. Bartoli, and X. Vilajosana. Exploring the performance boundaries of NB-IoT. *IEEE Internet of Things Journal*, 6(3):5702–5712, 2019.

[146] K. Mikhaylov. Accelerated Connection Establishment (ACE) Mechanism for Bluetooth Low Energy. In *Proc. of the IEEE PIMRC Conference*, 2014.

[147] Mini-Circuits. RCDAT-8000-90 - Programmable Attenuator, 2017.

[148] Y. Murillo, A. Chiumento, B. Reynders, and S. Pollin. SDN on BLE: Controlling resource constrained mesh networks. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2019.

[149] Y. Murillo, B. Reynders, A. Chiumento, S. Malik, P. Crombez, and S. Pollin. Bluetooth Now or Low Energy: Should BLE Mesh become a flooding or connection oriented network? In *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–6. IEEE, 2017.

[150] B. A. Nahas, S. Duquennoy, V. Iyer, and T. Voigt. Low-Power Listening Goes Multi-Channel. In *Proceedings of the* 10^{th} *IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, May 2014.

[151] W. Nam, D. Bai, J. Lee, and I. Kang. Advanced interference management for 5G cellular networks. *IEEE Communications Magazine*, 52(5):52–60, 2014.

[152] P. Narendra, S. Duquennoy, and T. Voigt. BLE and IEEE 802.15.4 in the IoT: Evaluation and Interoperability Considerations. In *Proc. of the* 1^{st} *EAI InterIoT Conference*, 2015.

[153] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. El-Bakoury. Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research. *IEEE Communications Surveys & Tutorials*, 21(1):88–145, 2018.

[154] R. Natarajan, P. Zand, and M. Nabi. Analysis of Coexistence between IEEE 802.15. 4, BLE and IEEE 802.11 in the 2.4 GHz ISM band. In *Proc. of the* 42^{nd} *IEEE Conference of the Industrial Electronics Society (IECON)*, 2016.

[155] J. Nieminen, C. Gomez, M. Isomaki, T. Savolainen, B. Patil, Z. Shelby, M. Xi, and J. Oller. Networking solutions for connecting Bluetooth Low Energy enabled machines to the internet of things. *IEEE network*, 28(6):83–90, 2014.

[156] J. Nieminen, T. Savolainen, M. Isomaki, Nokia, B. Patil, AT&T, Z. Shelby, Arm, and C. Gomez. RFC 7668 - IPv6 over Bluetooth Low Energy. `https://tools.ietf.org/html/rfc7668`, 2015.

[157] Z. Ning, X. Wang, J. J. Rodrigues, and F. Xia. Joint computation offloading, power alloca-

tion, and channel assignment for 5G-enabled traffic management systems. *IEEE Transactions on Industrial Informatics*, 15(5):3058–3067, 2019.

[158] Nordic Semiconductor. Nordic Semiconductor: nRF5 SDK, 2021.

[159] Nordic Semiconductors. nRF52840 Specifications. `https://www.nordicsemi.com/eng/Products/nRF52840`, 2018.

[160] Nuki. The Bluetooth Door Lock for Smart Access via Smartphone. `https://nuki.io/en/`, 2017.

[161] M. R. Palattella, M. Dohler, L. A. Grieco, G. Rizzo, J. Torsen, T. Engel, and L. Ladid. Internet of Things in the 5G Era: Enablers, Architecture and Business Models. *IEEE Journal on Selected Areas In Communications*, 2016.

[162] E. Park et al. AdaptaBLE: Adaptive Control of Data Rate, Transmission Power, and Connection Interval in Bluetooth Low Energy. *Computer Networks*, 2020.

[163] E. Park, H.-S. Kim, and S. Bahk. BLEX: Flexible Multi-Connection Scheduling for Bluetooth Low Energy. 2021.

[164] M. Park. IEEE 802.11 ac: Dynamic bandwidth channel access. In *2011 IEEE international conference on communications (ICC)*, pages 1–5. IEEE, 2011.

[165] G. Patti, L. Leonardi, and L. L. Bello. A Bluetooth low energy real-time protocol for industrial wireless mesh networks. In *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*, pages 4627–4632. IEEE, 2016.

[166] G. Pau et al. Bluetooth 5 Energy Management through a Fuzzy-PSO Solution for Mobile Devices of Internet of Things. *Energies*, 2017.

[167] V. Paxson et al. RFC6298: Computing TCP's Retransmission Timer, 2011.

[168] G. Pocovi, K. I. Pedersen, and P. Mogensen. Joint link adaptation and scheduling for 5G ultra-reliable low-latency communications. *IEEE Access*, 6:28912–28922, 2018.

[169] P. Popovski. Ultra-reliable communication in 5G wireless systems. In *1st International Conference on 5G for Ubiquitous Connectivity*, pages 146–151. IEEE, 2014.

[170] P. Popovski, K. F. Trillingsgaard, O. Simeone, and G. Durisi. 5G wireless network slicing for eMBB, URLLC, and mMTC: A communication-theoretic view. *IEEE Access*, 6:55765–55779, 2018.

[171] P. Rashidi and A. Mihailidis. A survey on ambient-assisted living tools for older adults. *IEEE journal of biomedical and health informatics*, 17(3):579–590, 2012.

[172] R. Ratasuk, B. Vejlgaard, N. Mangalvedhe, and A. Ghosh. NB-IoT system for M2M communication. In *2016 IEEE wireless communications and networking conference*, pages 1–5. IEEE, 2016.

[173] R. Razavi, M. Fleury, and M. Ghanbari. Fuzzy Control of Adaptive Timeout for Video Streaming over a Bluetooth Interconnect. In *2007 12th IEEE Symposium on Computers and Communications*, pages MW – 27–MW – 32, 2007.

[174] O. Reich, E. Hübner, B. Ghita, M. Wagner, and J. Schäfer. Bluetooth Performance Evaluation based on Notify for Real-time Body-Area Sensor Networks. In *2020 IEEE International Workshop on Metrology for Industry 4.0 & IoT*, pages 516–520. IEEE, 2020.

[175] T. Renzler, M. Spörk, C. A. Boano, and K. Römer. Improving the Efficiency and Responsiveness of Smart Objects using Adaptive BLE Device Discovery. In *Proceedings of the 4^{th} International Workshop on Experiences with the Design and Implementation of Smart Objects (SMARTOBJECTS)*. ACM, June 2018.

[176] M. Rezaee and M. H. Y. Moghaddam. SDN-based quality of service networking for wide area measurement system. *IEEE Transactions on Industrial Informatics*, 16(5):3018–3028, 2019.

[177] C. Roest. Enabling the Chaos Networking Primitive on Bluetooth LE. Master's thesis, Delft University of Technology, Delft, The Netherlands, Oct. 2015.

[178] R. Rondón, M. Gidlund, and K. Landernäs. Evaluating Bluetooth Low Energy Suitability for Time-Critical Industrial IoT Applications. *Journal of Wireless Information Networks*, 24(3), 2017.

[179] R. Rondón, K. Landernäs, and M. Gidlund. An Analytical Model of the Effective Delay Performance for BLE. In *Proc. of the 27^{th} Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2016.

[180] P. S. Rossi et al. Joint End-to-End Loss-Delay Hidden Markov Model for Periodic UDP Traffic Over the Internet. *IEEE Transactions on Signal Processing*, 54(2), 2006.

[181] R. B. Sattar, N. Ahmed, and M. Rahman. An Adaptive Approach for Video Streaming and Evaluation over Bluetooth Network. In *2012 8th International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1–4, 2012.

[182] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RFC3550: RTP: A transport protocol for real-time applications, 2003.

[183] M. Schuß, C. Boano, and K. Römer. Moving Beyond Competitions: Extending D-Cube to Seamlessly Benchmark Low-Power Wireless Systems. In *Proc. of the Workshop on Benchmarking Cyber-Physical Networks and Systems*, 2018.

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

[185] M. Schuß, C. A. Boano, M. Weber, M. Schulz, M. Hollick, and K. Römer. JamLab-NG: Benchmarking Low-Power Wireless Protocols under Controlable and Repeatable Wi-Fi Interference. In *Proc.of the 16^{th} Conference on Embedded Wireless Systems and Networks (EWSN)*, 2019.

[186] N. Semiconductors. Bluetooth Low Energy connected ski boot integrates multiple force and motion sensors to help skiers improve balance, control, and technique. `https://www.nordicsemi.com/News/2019/03/Atomic-ski-boot-integrates-multiple-force-and-motion-sensors`, 2019. Accessed on 01/03/2021.

[187] M. Serror, S. Vaaßen, K. Wehrle, and J. Gross. Practical evaluation of cooperative communication for ultra-reliability and low-latency. In *2018 IEEE 19th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pages 14–15. IEEE, 2018.

[188] M. Sha et al. ARCH: Practical channel hopping for reliable home-area sensor networks. In *Proc. of the 17^{th} RTAS Symp.*, 2011.

[189] M. U. Sheikh, B. Badihi, K. Ruttik, and R. Jäntti. Adaptive Physical Layer Selection for Bluetooth 5: Measurements and Simulations. *Wireless Communications & Mobile Computing*.

[190] M. Siekkinen, M. Hiienkari, J. K. Nurminen, and J. Nieminen. How Low Energy is Bluetooth Low Energy? Comparative Measurements with ZigBee/802.15.4. In *Proceedings of the IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, 2012.

[191] B. SIG. 3 Roles Bluetooth Plays in Enabling the Smart Building. https://www.bluetooth.com/blog/3-roles-of-bluetooth-in-the-smart-building/, 2019. Accessed on 01/03/2021.

[192] B. SIG. Bluetooth - Market Update 2020. https://www.bluetooth.com/wp-content/uploads/2020/03/2020_Market_Update-EN.pdf, 2020. Accessed on 24/02/2020.

[193] S. Silva, S. Soares, T. Fernandes, A. Valente, and A. Moreira. Coexistence and interference tests on a Bluetooth Low Energy front-end. In *Proc. of the Science and Information Conference (SAI)*, 2014.

[194] Simbex. Medical Device and Consumer Health Products - Simbex. https://simbex.com/our-work/, 2021. Accessed on 24/02/2021.

[195] A. Simonsson and A. Furuskar. Uplink power control in LTE-overview and performance, subtitle: principles and benefits of utilizing rather than compensating for SINR variations. In *2008 IEEE 68th Vehicular Technology Conference*, pages 1–5. IEEE, 2008.

[196] M. Simsek et al. 5G-enabled Tactile Internet. *IEEE Journal on Selected Areas in Communications*, 34(3), 2016.

[197] M. Slabicki, G. Premsankar, and M. Di Francesco. Adaptive configuration of LoRa networks for dense IoT deployments. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2018.

[198] Smaxtec. Understand your cow. Even better. From within. https://smaxtec.com/en/, 2021. Accessed on 23/02/2021.

[199] U. L. Solutions. Why Bluetooth controlled lighting is better than WiFi. https://unlumited.com/news/why-bluetooth-controlled-lighting-is-better-than-wifi/, 2021. Accessed on 01/03/2021.

[200] T. Soultanopoulos, S. Sotiriadis, E. Petrakis, and C. Amza. Internet of Things data management in the cloud for Bluetooth Low Energy (BLE) devices. In *Proceedings of the Third International Workshop on Adaptive Resource Management and Scheduling for Cloud Computing*, pages 35–39, 2016.

[201] M. Spörk, C. A. Boano, and K. Römer. Improving the Timeliness of Bluetooth Low Energy in Noisy RF Environments. In *Proceedings of the* 16^{th} *International Conference on Embedded Wireless Systems and Networks (EWSN)*. Junction Publishing, Feb. 2019.

[202] M. Spörk, C. A. Boano, and K. Römer. Performance and Trade-offs of the new PHY Modes of BLE 5. In *Proceedings of the International Workshop on Pervasive Systems in the IoT*

Era (PERSIST-IoT). ACM, July 2019.

[203] M. Spörk, C. A. Boano, and K. Römer. Improving the Timeliness of Bluetooth Low Energy in Dynamic RF Environments. *ACM Transactions on Internet of Things*, 1(2), Apr. 2020.

[204] M. Spörk, C. A. Boano, M. Zimmerling, and K. Römer. BLEach: Exploiting the Full Potential of IPv6 over BLE in Constrained Embedded IoT Devices. In *Proceedings of the* 15^{th} *ACM International Conference on Embedded Networked Sensor Systems (SenSys)*. ACM, Nov. 2017.

[205] M. Spörk, J. Classen, C. A. Boano, M. Hollick, and K. Römer. Improving the Reliability of Bluetooth Low Energy Connections. In *Proceedings of the* 17^{th} *International Conference on Embedded Wireless Systems and Networks (EWSN)*. Junction Publishing, Feb. 2020.

[206] M. Spörk, M. Schuß, C. A. Boano, and K. Römer. An Open-Source IPv6 over BLE Stack for Contiki. In *Proceedings of the* 14^{th} *International Conference on Embedded Wireless Systems and Networks (EWSN), poster session*. ACM, Feb. 2017.

[207] M. Spörk, M. Schuß, C. A. Boano, and K. Römer. Ensuring End-to-End Dependability Requirements in Cloud-based Bluetooth Low Energy Applications. In *Proceedings of the* 18^{th} *International Conference on Embedded Wireless Systems and Networks (EWSN)*. Junction Publishing, 2021.

[208] Sportcor. Sportcor - IoT Sports Development. `https://www.sportcor.com/`, 2021. Accessed on 24/02/2021.

[209] S.Raza, P. Misra, Z. He, and T. Voigt. Building the Internet of Things with Bluetooth Smart. *Ad Hoc Networks*, 57, 2017.

[210] K. Srinivasan et al. An empirical study of low-power wireless. *ACM Transactions on Sensor Networks*, 6(2), 2010.

[211] Statistica. Bluetooth low energy (BLE) enabled devices market volume worldwide, from 2013 to 2020. `https://www.statista.com/statistics/750569/worldwide-bluetooth-low-energy-device-market-volume/`, 2021. Accessed on 24/02/2021.

[212] Statistica. Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025. `https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide`, 2021. Accessed on 23/02/2021.

[213] S. Sun, S. Moon, and J.-K. Fwu. Practical link adaptation algorithm with power density offsets for 5G uplink channels. *IEEE Wireless Communications Letters*, 9(6):851–855, 2020.

[214] T. Szigeti, C. Hattingh, R. Barton, and K. Briley Jr. *End-to-End QoS Network Design: Quality of Service for Rich-Media & Cloud Networks*. Cisco press, 2013.

[215] R. Tavakoli, M. Nabi, T. Basten, and K. Goossens. Dependable interference-aware time-slotted channel hopping for wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 14(1):1–35, 2018.

[216] Texas Instruments. Texas Instruments - CC2650 SensorTag. `http://www.ti.com/ww/en/wireless_connectivity/sensortag/`, 2016.

[217] The Zephyr Project. Zephyr OS: An RTOS for IoT, 2020.

[218] A. Thierer and A. Castillo. Projecting the Growth and Economic Impact of the Internet of Things. *George Mason University, Mercatus Center, June*, 15, 2015.

[219] F. Touati, R. Tabish, and A. B. Mnaouer. A Real-Time BLE enabled ECG System for Remote Monitoring. *APCBEE procedia*, 7, 2013.

[220] J. J. Treurniet, C. Sarkar, R. V. Prasad, and W. d. Boer. Energy Consumption and Latency in BLE Devices under Mutual Interference: An Experimental Study. In *Proc. of the* 3^{rd} *Conference on Future Internet of Things and Cloud*, 2015.

[221] U. Varshney, A. Snow, M. McGivern, and C. Howard. Voice over IP. *Communications of the ACM*, 45(1):89–96, 2002.

[222] J.-P. Vasseur and A. Dunkels. *Interconnecting Smart Objects with IP: The Next Internet*. Morgan Kaufmann, 2010.

[223] P. Vishnupriya. Mitigation of co-channel interference in long term evolution. *The International Journal of Engineering and Science (IJES)*, 2(4):2319–1805, 2013.

[224] Y. A. Wang, C. Huang, J. Li, and K. W. Ross. Queen: Estimating packet loss rate between arbitrary internet hosts. In *International Conference on Passive and Active Network Measurement*, pages 57–66. Springer, 2009.

[225] T. Watteyne et al. Reliability through frequency diversity: why channel hopping makes sense. In *Proc. of the* 6^{th} *PE-WASUN Symp.*, 2009.

[226] T. Watteyne, S. Lanzisera, A. Mehta, and K. S. Pister. Mitigating multipath fading through channel hopping in wireless sensor networks. In *2010 IEEE International Conference on Communications*, pages 1–5. IEEE, 2010.

[227] K. Winstein et al. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *Proc. of the* 10^{th} *USENIX NSDI Symposium*, 2013.

[228] M. Xiao, S. Mumtaz, Y. Huang, L. Dai, Y. Li, M. Matthaiou, G. K. Karagiannidis, E. Björnson, K. Yang, I. Chih-Lin, et al. Millimeter wave communications for future mobile networks. *IEEE Journal on Selected Areas in Communications*, 35(9):1909–1935, 2017.

[229] F. Yan et al. Pantheon: the training ground for Internet congestion-control research. In *Proc. of the 2018 USENIX ATC Conf.*, 2018.

[230] X. Yang, D. Scholz, and M. Helm. Deterministic Networking (DetNet) vs Time Sensitive Networking (TSN). *Network*, 79, 2019.

[231] Y. Yao, X. Chen, L. Rao, X. Liu, and X. Zhou. LORA: Loss differentiation rate adaptation scheme for vehicle-to-vehicle safety communications. *IEEE Transactions on Vehicular Technology*, 66(3):2499–2512, 2016.

[232] T. Zachariah, N. Klugman, B. Campbell, J. Adkins, N. Jackson, and P. Dutta. The internet of things has a gateway problem. In *Proceedings of the 16th international workshop on mobile computing systems and applications*, pages 27–32, 2015.

[233] Y. Zaki et al. Adaptive Congestion Control for Unpredictable Cellular Networks. In *Proc. of the 2015 ACM SIGDC Conf.*, 2015.

[234] P. Zenker, S. Krug, M. Binhack, and J. Seitz. Evaluation of BLE Mesh capabilities: A case study based on CSRMesh. In *2016 Eighth International Conference on Ubiquitous and*

Future Networks (ICUFN), pages 790–795. IEEE, 2016.

[235] Zigbee Alliance. Project Connected Home over IP. `https://www.connectedhomeip.com/`, 2019. Accessed on 24/02/2021.

[236] M. Zimmerling et al. Adaptive Real-Time Communication for Wireless Cyber-Physical Systems. *ACM Transactions on CPS*, 1(2), 2017.

[237] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele. pTunes: Runtime parameter adaptation for low-power MAC protocols. In *Proceedings of the 11th international conference on Information Processing in Sensor Networks*, pages 173–184, 2012.