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*;

KEYWORDS

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

SenSys '17, November 6–8, 2017, Delft, Netherlands

ACM ISBN 978-1-4503-5459-2/17/11…$15.00
https://doi.org/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*. ACM, New York, NY, USA, 14 pages. https://doi.org/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.

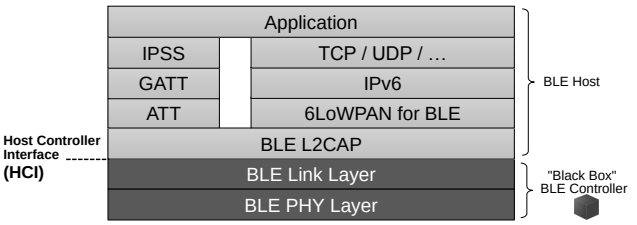


Figure 1: BLE architecture as specified by RFC 7668 [24].

- 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.

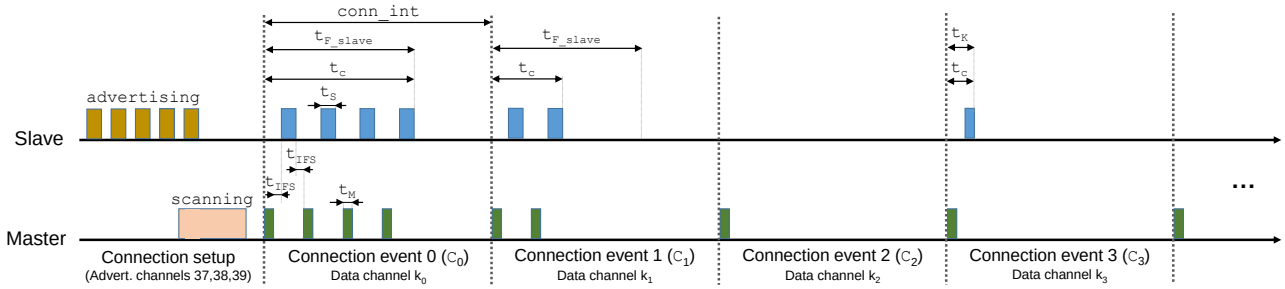


Figure 2: Example of a BLE connection-based data exchange between a slave and a master using a slave latency of 1.

Temporal decoupling from upper layers. Using the connection-less 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

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

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.

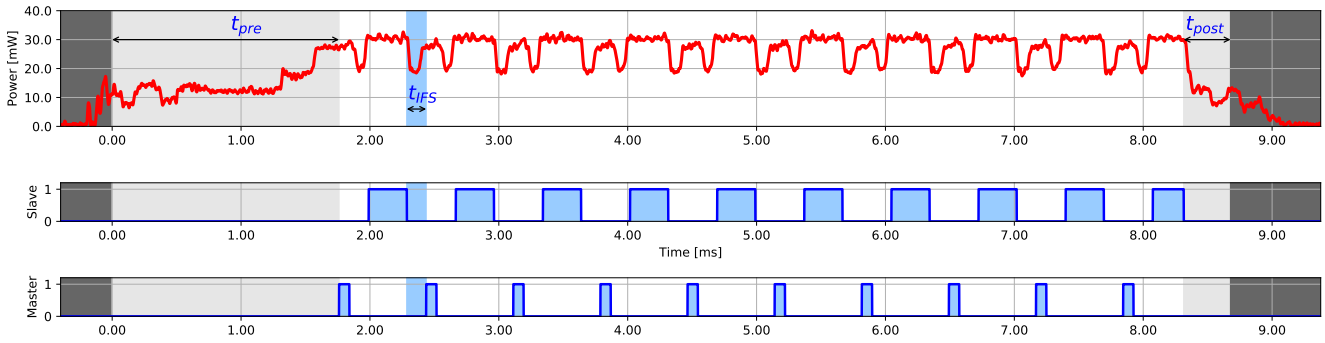


Figure 3: Power draw, packet transmissions, and gaps between packets during a BLE connection event, recorded using a mixed-signal oscilloscope on a TI CC2650 platform that acts as the slave and communicates with a master in connection-based mode.

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.

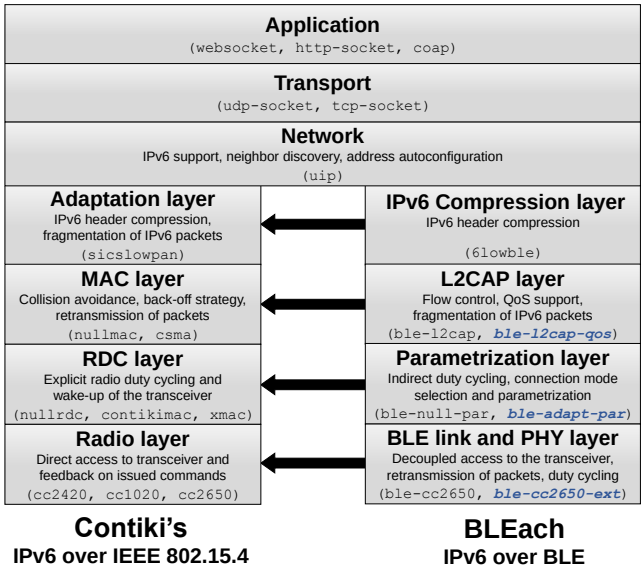


Figure 4: Architecture of BLEach and corresponding layers in Contiki's IPv6-over-IEEE 802.15.4 stack.

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

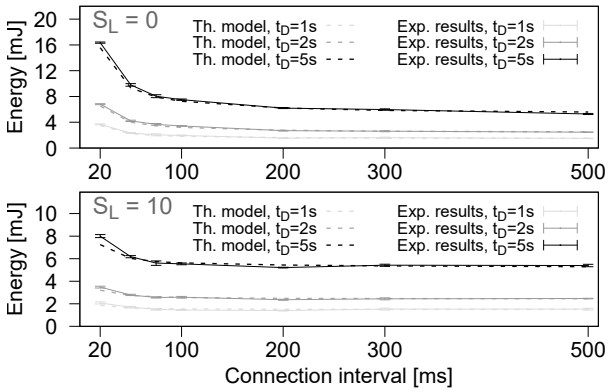


Figure 5: Impact of connection interval and slave latency on energy consumption for 256 bytes connection capacity.

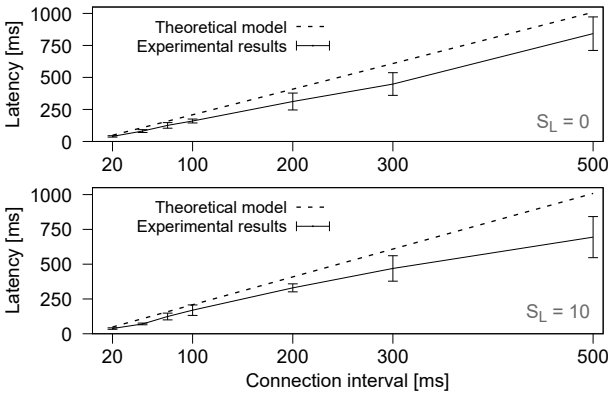


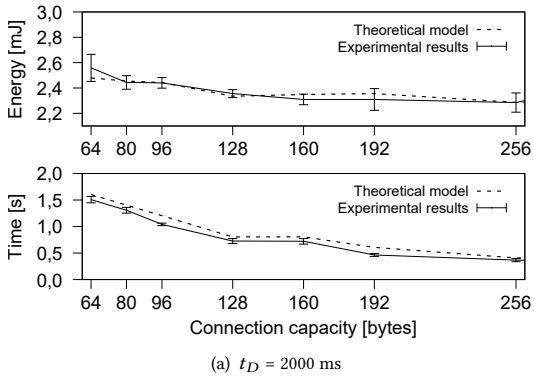
Figure 6: Impact of connection interval and slave latency on communication latency for 256 bytes connection capacity.

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.

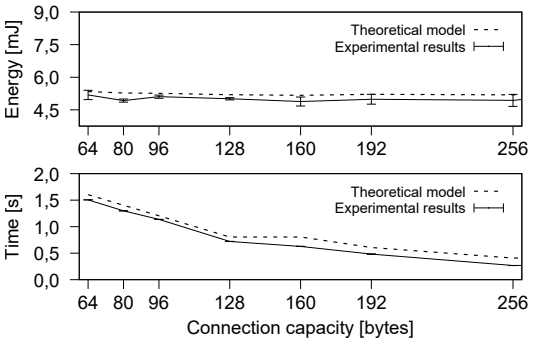
Fig. 7 plots energy and latency against connection capacity F for two different t_D; connection interval and slave latency are fixed to 200 ms and 10, respectively. Again, we see that our models accurately predict performance. We also see that the connection capacity F affects energy only minimally, 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.

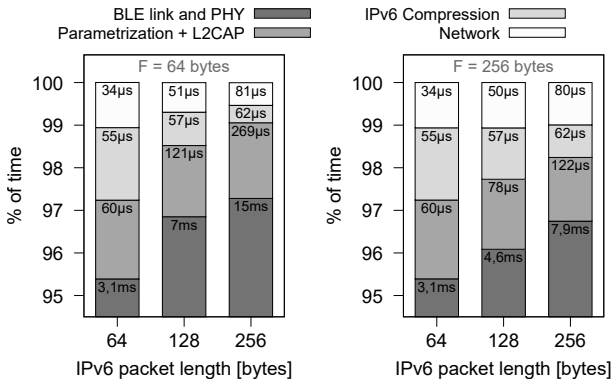


Figure 8: Breakdown of BLEach's processing time per layer when serving IPv6 transmissions of varying packet length.

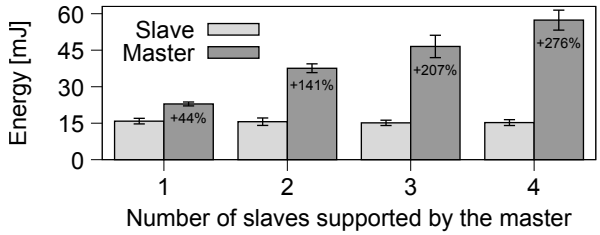


Figure 9: Energy consumption of slave and master running BLEach against the number of supported slaves in a subnet.

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.

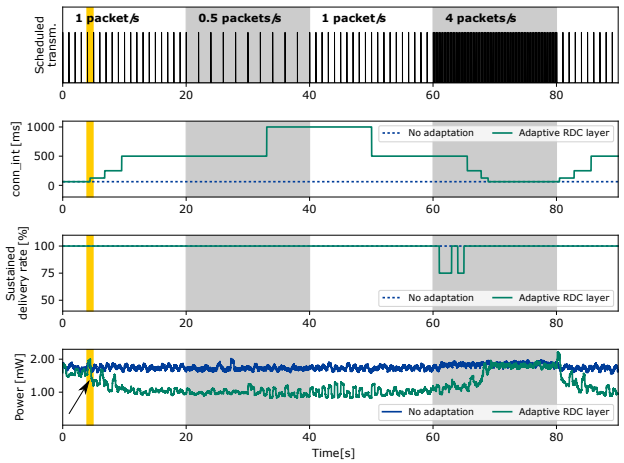


Figure 10: Performance of BLEach with and without adaptive duty cycling as the traffic load changes over time.

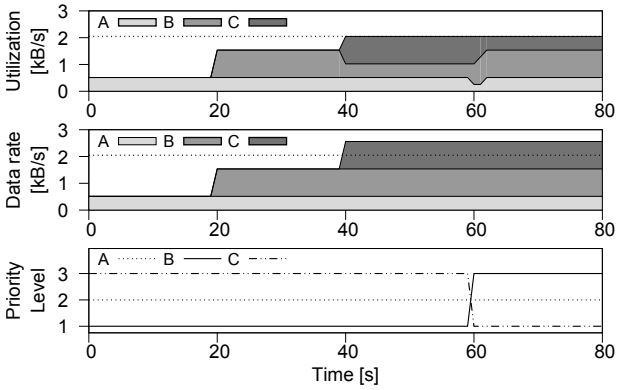


Figure 11: IPv6-over-BLE traffic multiplexing and prioritization in action, as provided by `ble-l2cap-qos` in BLEach.

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.

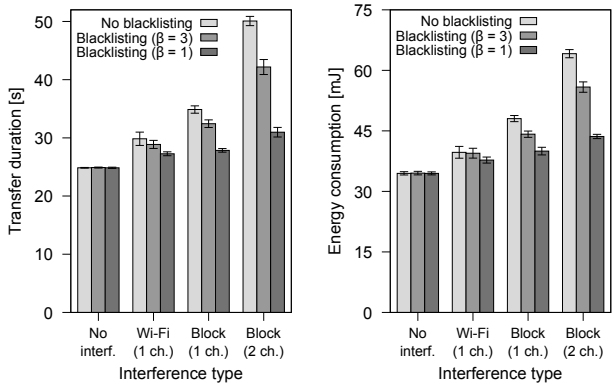


Figure 12: Benefit of indirect link-quality monitoring and channel blacklisting in the presence of radio interference.

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.)*).

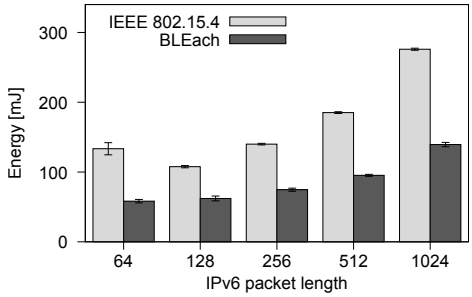
We use an application that exchanges 50 consecutive IPv6 packets with a length of 256 bytes between a master and a slave using `conn_int` = 250 ms and $S_L = 0$. Each packet needs to be acknowledged before the next one can be transmitted. We run BLEach using `ble-cc2650-ext` and show that the ability 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].

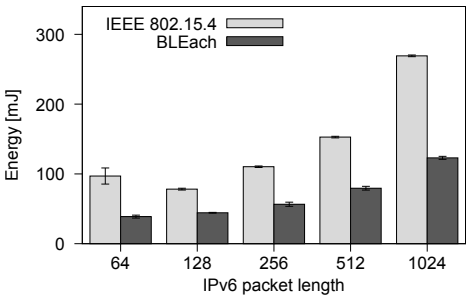
6.4 Comparison with IPv6 over IEEE 802.15.4

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

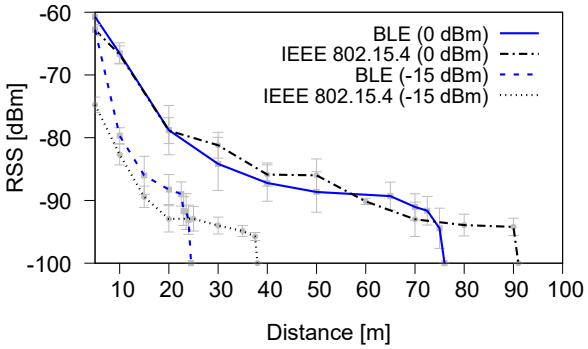


Figure 14: Received signal strength measured with the TI CC2650 using the BLE and IEEE 802.15.4 radio modes.

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.*