Mind the Gap: Multi-hop IPv6 over BLE in the IoT

Hauke Petersen
hauke.petersen@fu-berlin.de
Freie Universität Berlin
Germany

Thomas C. Schmidt
t.schmidt@haw-hamburg.de
HAW Hamburg
Germany

Matthias Wählisch
m.waehlisch@fu-berlin.de
Freie Universität Berlin
Germany

Table 1: Comparison of common IoT radios.

	Radio (●=high …○=low support)				
	BLE (mesh)	BLE (star)	IEEE 802.15.4	LoRa	WLAN
Throughput	◑	◑	◔	○	●
Range	◕	○	◕	●	◔
Node Count	◕	○	◕	●	◑
Energy Efficiency	●	●	◑	◕	○
Available on devices	●	●	◑	◑	●

ABSTRACT

Bluetooth Low Energy (BLE) is today's most popular low-power radio technology with compelling radio performance and battery-friendly characteristics, making it a promising deployment option for the Internet of Things (IoT). Little is known, however, about the performance and pitfalls when utilizing BLE as link layer in multi-hop IP over BLE scenarios, because of the lack of available software platforms and deployment experiences. In this work, we present both a fully open-source, configurable software platform and experiments to analyze multi-hop BLE network behavior. Our experiments, conducted in a larger testbed, reveal unexpected performance drawbacks. Even in scenarios with underutilized links, BLE connections break randomly. This results into large transmission delays on the network layer and thus hinders real-world deployments in the constrained IoT. As key reason for this behavior we identify the BLE connection interval. A deterministic interval leads to unpredictable link behavior and connection losses due to overlapping connection events. We propose randomizing connection intervals as mitigation strategy and demonstrate that this prevents connection losses and sporadic link degradation, improving the overall network behavior.

CCS CONCEPTS

• **Networks → Network protocol design**; *Network experimentation*; Short-range networks; *Network reliability*.

ACM Reference Format:
Hauke Petersen, Thomas C. Schmidt, and Matthias Wählisch. 2021. Mind the Gap: Multi-hop IPv6 over BLE in the IoT. In *The 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '21), December 7–10, 2021, Virtual Event, Germany.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3485983.3494847

1 INTRODUCTION

Bluetooth Low Energy (BLE) is the most deployed low-power wireless technology today, and deployment numbers are projected to grow significantly in the next years [24]. This availability paired with competitive radio performance and low power characteristics make BLE a promising choice for Internet of Things (IoT) applications. The envisioned disruption [34], however, did not happen so far. One of the reasons lies in the silo characteristic of the Bluetooth standard [23], which prevents BLE-based applications to be integrated in heterogeneous application contexts typically demanded in the IoT. Current BLE communication is limited to confined scenarios such as directly connected consumer devices or beacon systems (*e.g.,* [30, 44]).

The foundation for opening up BLE to the wider IoT was laid with the standardization of IPv6 over Bluetooth Low Energy (*6LoBLE*) [33], which defines the usage of BLE as a link layer in IPv6-based 6LoWPAN networks. Together with the advanced standardization of IPv6 mesh over BLE networks [20] this allows for large scale multi-hop deployments, introducing both BLE performance and low energy properties as well as Internet principles such as interoperability and openness.

Table 1 qualitatively compares the most common link layer technologies used in the IoT today. While BLE features very low energy consumption [32, 38], it suffers from limited range and node count when used in its designated star topology. BLE mesh networks mitigate these limitations. There exist multiple approaches to BLE mesh networks [12, 18, 42], including the Bluetooth Mesh standard [9], but none of these support IP connectivity. Since we consider IP end-to-end connectivity as key driver for the future IoT, these concepts are not discussed further in this work. To become a relevant deployment option, two contributions for multi-hop IP over BLE are missing: a deeper understanding of the characteristics and limitations of its network performance, and available software platforms.

In this work, we systematically measure key performance metrics of multi-hop IPv6 over BLE in a medium scale real-world deployment of low-power hardware. Based on 15 low-power nodes in the FIT IoTlab testbed [5] we analyze reliability, latency, throughput, and energy consumption in common IoT scenarios. We show that multi-hop IP over BLE is not only feasible but outperforms IEEE 802.15.4, even though, in contrast to IEEE 802.15.4, BLE was never designed for IP multi-hop operation. Our experiments reveal the impact of *connection shading, i.e.,* time domain interferences of independent BLE connections, on connection stability and network latency. We propose a solution to mitigate the discovered issues.

Over the last decade, no low-power link layer technology has managed to become a clear go-to solution for the IoT. We argue that BLE has the potential to change this for three reasons: its popularity,

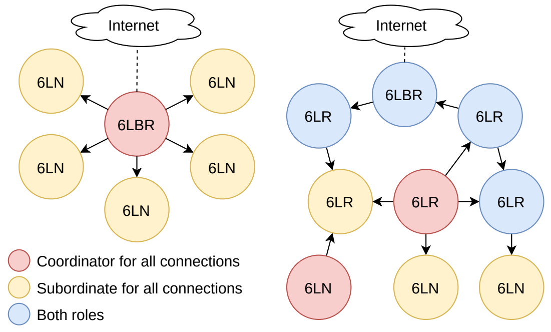


Figure 1: Border Router (6LBR), router (6LR), and non-router (6LN) roles versus BLE roles of nodes in a star topology [33] (left) and a more flexible *6BLEMesh* topology [20] (right).

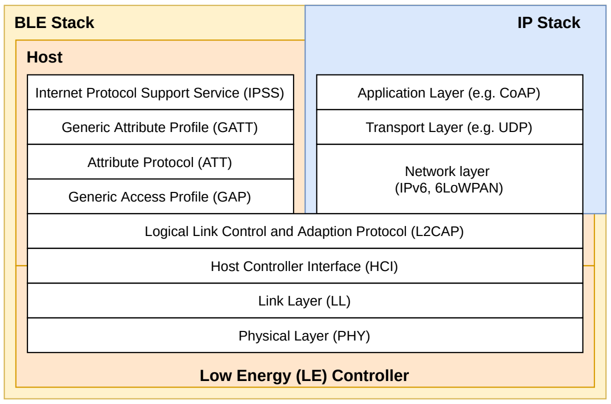

BLE Stack
IP Stack
Host
Internet Protocol Support Service (IPSS)
Generic Attribute Profile (GATT)
Attribute Protocol (ATT)
Generic Access Profile (GAP)
Application Layer (e.g. CoAP)
Transport Layer (e.g. UDP)
Network layer (IPv6, 6LoWPAN)
Logical Link Control and Adaption Protocol (L2CAP)
Host Controller Interface (HCI)
Link Layer (LL)
Physical Layer (PHY)
Low Energy (LE) Controller

Figure 2: IP over BLE architecture.

its best-in-class energy efficiency [32, 38], and its network performance characteristics. Prior work is only based on simulation, carried out on a very small number of nodes, or deployed on hardware that is not low-power. The implementation of software platforms is mainly hindered because most Bluetooth stacks do not provide any or only a largely simplified set of IP functionalities. Furthermore, the majority of Bluetooth stacks are proprietary, challenging developers to integrate third party IP network stacks. As a step towards missing software platforms, we present the first open source platform with full support for multi-hop IP over BLE. The presented platform is based on two well established open source projects, RIOT with its IP stack [8] and the Apache NimBLE BLE stack [1].

In summary, our main contributions are the following:

(1) An open-source, full feature implementation of IP multi-hop over BLE [7] that is on par with common IoT hardware resources. (§ 3)
(2) Reproducible experiments that may guide future research. (§ 4)
(3) A comprehensive performance evaluation of multi-hop IP over BLE in mid-sized networks. We reveal drawbacks in multi-hop BLE that are amplified by local clock drifts, show trade-offs, and compare with IEEE 802.15.4. Even in scenarios with moderate network load, multi-hop BLE may easily fail to provide predictable, reliable transport. (§ 5)
(4) We identified connection shading to be the core reason for unexpected performance penalties in BLE and applications on top. We present a mitigation strategy that provides full reliability for common IoT applications. Our proposal is standard-compliant. (§ 6)

In the remainder, we will introduce background in § 2, present our contributions in § 3–§ 6, discuss related work and overall findings in § 7 and § 8, and conclude in § 9.

2 BACKGROUND

The IPv6 over BLE (*6LoBLE*) standard [33] and the Internet Service Support Profile [22], extended by the IETF IPv6 Mesh over BLE (*6BLEmesh*) draft [20], define that IP data is to be transferred using the connection-based mode of BLE. As a detailed understanding of this mode is crucial to analyze its impact on network performance, this section will explain relevant background.

2.1 IP over BLE

For each BLE connection, one node must take the role of a *connection coordinator*, while the other node takes the role of a *connection subordinate*.[1] In contrast to the star topology that was forced in older Bluetooth versions, Bluetooth 4.2 (and more recent) allow for devices to carry out both roles simultaneously, laying the foundation for actual connection-based mesh networks. Figure 1 illustrates the differences between a star topology compliant to RFC 7668 [33] and a mesh network formed by nodes in various roles.

Figure 2 depicts the generic protocol stack architecture to transfer IP data over BLE links. IP packets are handed from the IP stack to Bluetooth's *Logical Link Control and Adaption Protocol* (L2CAP) layer. The L2CAP layer provides so called *Connection Oriented Channels* with *Credit based Flow Control*, which work similar compared to a pipe and guarantee full duplex, reliable, and in-order transfer of IP data.

2.2 BLE Connections

To improve reliability, BLE connections employ *time-sliced channel hopping* (TSCH) as follows: In the time domain, a connection is split into fixed-size time slices called *Connection Events*, with the interval between two consecutive events denoted as *Connection Interval*. All packets exchanged during a single connection event are transmitted on one of 37 possible frequencies (data channels). For each subsequent connection event the frequency is switched following one out of two defined *channel selection algorithms* (CSA). By enabling these algorithms to further restrict the pool of used channels based on channel maps, BLE allows for *Adaptive Channel Hopping* (ADH). However, the Bluetooth standard only defines means for updating and applying those channel maps. It does not describe how to implement the ADH algorithms. It leaves this completely to implementers of controllers.

Each connection event follows a strict packet flow (*cf.*, Figure 3). Each event is started by the coordinator sending a packet to the subordinate. After receiving the packet, the subordinate will reply with a packet after a fixed amount of time called the *inter frame spacing* (IFS), which is exactly 150μs for the 1Mbps PHY mode. This packet exchange is done at least once in every connection event.

[1]The terms "coordinator" and "subordinate" used in this paper diverge from Bluetooth specifications, to support non-discriminatory language.

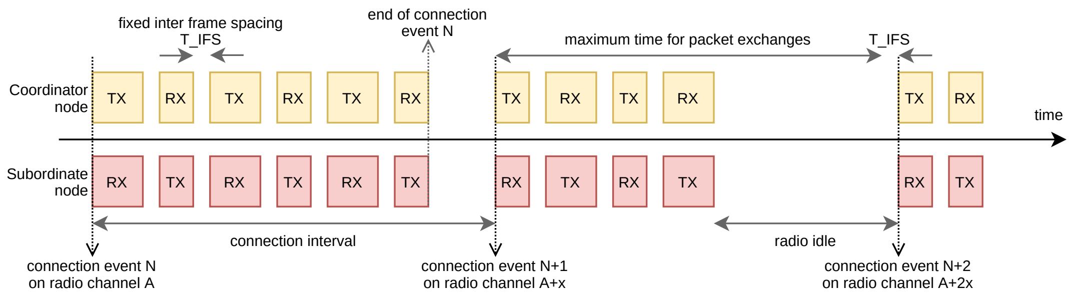

fixed inter frame spacing
T_IFS
end of connection event N
maximum time for packet exchanges
T_IFS
Coordinator node
Subordinate node
TX
RX
time
connection interval
radio idle
connection event N on radio channel A
connection event N+1 on radio channel A+x
connection event N+2 on radio channel A+2x

Figure 3: Packet flow for a single connection. Packets may be exchanged until the next connection event starts minus the inter frame spacing to allow for radio setup. If none of the peers has data to transfer, a connection event includes two empty packets.

In case that none of the peers has any data to transmit, empty payloads are exchanged. Every packet carries a *More Data* (MD) flag, which can be set by each peer to signal to the connection partner that the node has more data to transfer. If more data is ready for transfer, it is up to the coordinator to decide to start one or more additional packet exchanges in the same connection event. Each of the additional packet exchanges inside a connection event has to start exactly one single IFS interval after the last packet was received from the subordinate. Peers may exchange packets until one of two conditions is fulfilled: (*i*) the next connection event of current connection starts, or (*ii*) a BLE activity unrelated to the current BLE connection needs access to the radio, *e.g.,* a connection event of a different connection or an advertising event. In either case, a spacing of at least a single IFS interval between the last received packet by the coordinator and the first packet of the subsequent event must be kept.

Packets are acknowledged between coordinator and subordinate using a 1-bit sequence number that is piggy bagged in the link layer header of each packet. Any lost packet is retransmitted until a valid acknowledgment has been received.

To optimize the energy consumption of a subordinate, the Bluetooth standard further allows connection subordinates to skip a certain number of connection events. This number is defined as *Subordinate Latency*. To prevent dead connections from drowning a controller, BLE defines a *Supervision Timeout*. If the time between the last and next valid packet received exceeds this supervision timeout, the connection is considered to be lost and terminated. In § 5, we will analyse reasons for reaching this timeout.

The timing parameters, the subordinate latency, and the CSA in use are initially defined by the connection coordinator during connection initiation. After a connection is opened, the Bluetooth standard defines a set of link layer control mechanisms that can be used to update these parameters on-the-fly.

2.3 Multi-role Configuration

Since version 4.2, Bluetooth enables a node to act as coordinator and subordinate at the same time for different connections. This challenges the networked system: nodes that attain both roles have to synchronize events and timings with multiple peers because each connection does have its independent set of parameters.

One challenge a multi-role node faces is that a connection coordinator dictates the point in time when an initial connection event

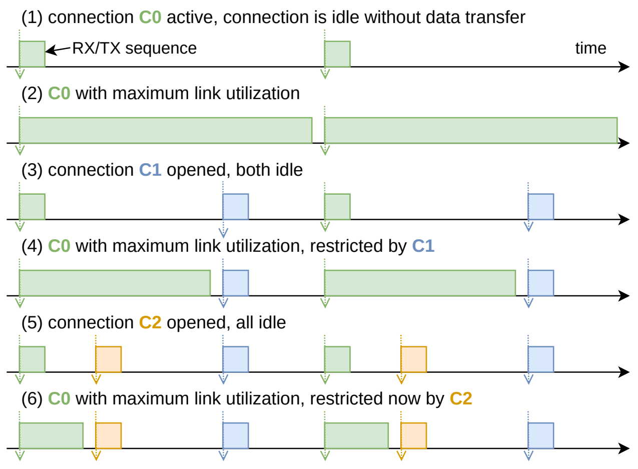

(1) connection C0 active, connection is idle without data transfer
RX/TX sequence
time
(2) C0 with maximum link utilization
(3) connection C1 opened, both idle
(4) C0 with maximum link utilization, restricted by C1
(5) connection C2 opened, all idle
(6) C0 with maximum link utilization, restricted now by C2

Figure 4: Packet flow from a single node with a one, two, and three concurrent connections. In each scenario, the node tries to transfer as much data as possible over connection `C0`, while the capacity of `C0` decreases with each additional connection.

starts, and with this the starting point of each subsequent connection event. If a subordinate maintains two or more connections, this can lead to overlapping connection events and hence prevents this node to properly respond to each event. Unfortunately, the Bluetooth standard does not specify any mitigation strategy for this problem, which we tackle in § 6.

The timing of connection events dictated by the coordinator introduces fluctuating capacities of single connections. To illustrate this, Figure 4 shows the data transfer from the perspective of a single node. Step (1) depicts a single connection `C0` in idle state. Step (2) shows the full utilization of `C0`, packets can be exchanged until the next connection event for `C0` starts. In step (3), an additional connection `C1` was opened, and step (4) illustrates the maximal possible utilization of `C0` in this case. Now, the maximum time available for packet exchanges is limited by the start of the following connection event of `C1`. Steps (5) and (6) show the same situation after a third connection `C2` was opened, decreasing the capacity of `C0` even further.

It is not surprising that each additional connection leads to less radio time for other connections. It is worth noting, though, that

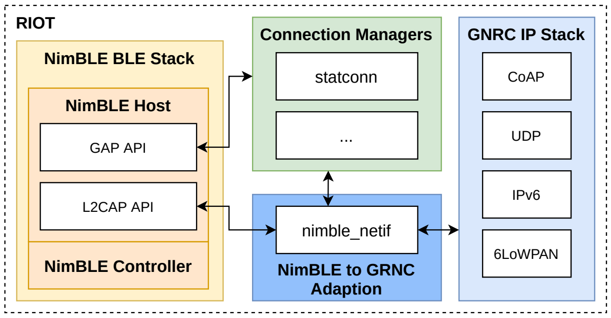


Figure 5: Our software architecture to integrate IP over BLE.

the amount of time each connection may use fully depends on the relative timing of the starting points of the connection events. As the coordinator defines the timing of a new connection, without any knowledge about other potential connections subordinates might maintain, the relative timing of connection events for independent connections on the subordinate node becomes unpredictable. Considering our example above and assuming that the node is subordinate for `C1` and `C2`, the relative position of the connection events of `C1` and `C2` are completely random from the perspective of the node. Consequently, the link capacity for `C0` is randomized as well, as it depends on those relative positions.

3 MULTI-HOP BLE SOFTWARE PLATFORM

For IP over BLE to become a feasible choice for IoT applications there is a need for portable, low-power, secure, and accessible software platforms. Currently, the number of platforms that supports to run IP over BLE applications is limited. One reason for this small number of options is that the complexity of the Bluetooth Standard. This is especially true for the Bluetooth Low Energy Controller part, as the controller has very strict timing requirements. This complexity is a major reason for most available BLE stack implementations to be hardware-vendor specific and closed source. Although these vendor stacks are mostly stable and offer a wide range of BLE features, they have three significant drawbacks: (*i*) they tend to be difficult to integrate with external system software and operating systems, (*ii*) they offer none or only very rudimentary Internet Protocol support, and (*iii*) they raise security concerns due to their closed source nature. To mitigate these effects, a portable, fully open source platform is preferred, allowing the user to have full control over every aspect of the stack and allowing for seamless integration.

We present a new software platform based on the IoT operating system RIOT [8], its GNRC network stack [31], and the Apache NimBLE Bluetooth Low Energy stack [1]. Figure 5 illustrates the basic architecture of our integration of RIOT and NimBLE. NimBLE was ported to RIOT as an external package, allowing it to be managed by the RIOT build system while mapping all of its system layer interfaces onto RIOT APIs. Additionally, we added a wrapper called *nimble_netif* on top of NimBLEs L2CAP and GAP APIs to expose BLE as link layer to the RIOT GNRC network stack. This module forwards IP packets between the IP stack and the L2CAP connection oriented channels opened by the corresponding BLE connections.

Typical machine-to-machine scenarios need automated management of BLE connections, because nodes advertise themselves, scan for neighbors, and decide on when to open or close BLE connections to whom. The *Internet Service Support Profile* specifies how nodes can check for neighbor's IP capabilities but does not define any connection management strategies. Only Lee *et al.* [29] propose connection management in BLE based on RPL meta data. To cater for different connection management strategies in the future, our proposed software architecture features a modular design that allows for implementation and selection of different strategies. For the experiments presented in this work, we implement a connection manger called *statconn*, which enables static connection management. Based on a predefined configuration, a node starts advertising its presence (subordinate role), or starts scanning for advertisements sent by the configured peer and initiates a connection (coordinator role). The *statconn* module monitors the health of each configured connection. If a connection is dropped, the module goes back into advertising/scanning mode to reopen the lost connection.

4 EVALUATION SETUP

In this section, we present the experiment setup to evaluate the performance of IP over BLE networks in typical IoT scenarios, using our software platform in a large-scale testbed.

4.1 Testbed and Hardware Platforms

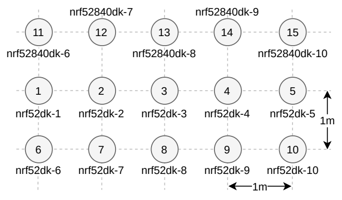
We conduct all experiments in the FIT IoTlab [5], a public testbed for constrained IoT devices. For the BLE-based measurements, we use 15 nodes (ten *nrf52dk* and five *nrf52840dk* nodes) at the Saclay site. All nodes are located in the same room, arranged in a 1m×1m grid (see Figure 6(a)). We had to limit the number of nodes because other BLE capable boards in the IoTlab had not enough memory, were too unstable, or are located in a separate room with unreliable connectivity to the other nodes. The *nrf52dk* nodes are equipped with Nordic nRF52832 SoCs featuring an Arm Cortex-M4F micro controller, 65Kbyte RAM, and 512Kbyte flash. The *nrf52840dk* nodes feature a more recent Nordic nRF52840 SoC, also with an Arm Cortex-M4F core but 256Kbyte RAM and 1Mbyte of flash memory. Both SoCs include a BLE radio that allows for low-level software access.

To compare BLE with IEEE 802.15.4, we use 15 *m3* nodes at the Strasbourg site of the IoTlab. These nodes feature an STM32F103 Arm Cortex-M3 micro controller paired with a IEEE 802.15.4 radio and provide 64Kbyte of RAM and 256Kbyte of ROM. Platforms such as TinySDR [28] are out of scope for this paper, as we do not need the flexibility of software-defined radio but focus on scenarios using an integrated stack of a multi-purpose operating system.

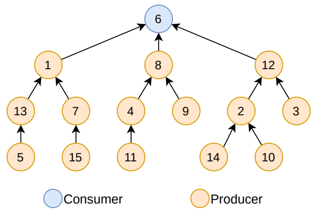
4.2 Software Configuration

For all experiments, we use the software platform introduced in § 3 based on RIOT version 2021.01 and NimBLE version `542806a` (development branch). Please note that we cannot use the latest stable release of NimBLE as we contributed a number of bug fixes, which are not merged yet but essential to run multi-hop BLE without disturbances. If not noted otherwise, we use the default configurations.

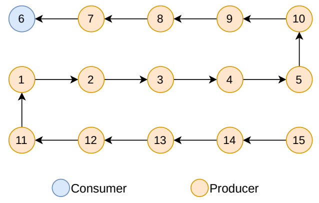
Regarding the GNRC network stack, we enable the 6LoWPAN router on all nodes and enable *gcoap* to include support for the *Constrained Application Protocol* (CoAP) [37]. We disable router



(a) Spatial placement.



(b) Tree topology.



(c) Line Topology.

Figure 6: Physical locations and statically configured BLE connections of nodes used in our experiments.

advertisements, because they are not needed, and increase the maximum number of entries in the neighbor information base to 32, to reach all nodes. The GNRC packet buffer is left to its default value of 6144 bytes.

In all measurements, we deploy the basic BLE PHY mode of 1Mbps because the *nrf52dk* nodes do not support other modes. As in the default configuration of NimBLE in RIOT, the link layer data length extension is enabled and NimBLE's packet buffer is configured to be 6600 bytes. It is worth noting that we found BLE channel 22 permanently jammed by an external signal. To prevent a bias in our results, all nodes are statically configured to not use this channel.

The *statconn* connection manager handles connections on each node, using an advertising interval of 90ms for nodes in a subordinate role and a scan interval and a scan window of 100ms for nodes in a coordinator role. This configuration leads to an average delay between a connection loss and a reconnect event in the range of 10ms – 100ms.

To analyze the state of the network and to track packet flows, we dump events to the STDIO of each node. This is based on a custom event handler, integrated into NimBLE and GNRC modules. This handler is designed carefully to keep the order of events intact and to restrict the number of characters per event to the minimum such that they do not exceed the IoTLab's STDIO capacity.

4.3 Topologies and Traffic Patterns

We deploy two different network topologies, a tree topology with a maximum hop count of 3 and a line topology with a hop count of 14 nodes. While the tree topology resembles a setup that is commonly found in real-world IoT applications, the line topology resembles an extreme case that allows to examine the packet traversal via long routes (see Figure 6). For the tree topology, we select nodes in a randomized fashion to prevent a bias towards an optimized topology. In the line topology, nodes simply connect to their physical neighbors. Both topologies are configured statically as follow: (*i*) the BLE connections are setup using the *statconn* connection manager and (*ii*) the IP routes are manually configured to build routes that forward IP traffic towards the root of the tree or one end of the line. All BLE nodes are in radio range of each other and the node selection has a negligible impact on the experiment results. The use of mobile nodes and dynamic BLE topology building is left for future research.

Although the standards would allow us to deploy IP over BLE in fully meshed topologies, we did not consider full meshes in this work for two reasons: First, a common IPv6 routing protocol for low-power IoT networks, RPL [43], is tree-based. Second, the radio scheduling as well as constrained memory limit the maximum number of simultaneous BLE connections for each node. To fully exploit BLE connections, we decided to focus on link layer topologies that resemble the network layer topologies.

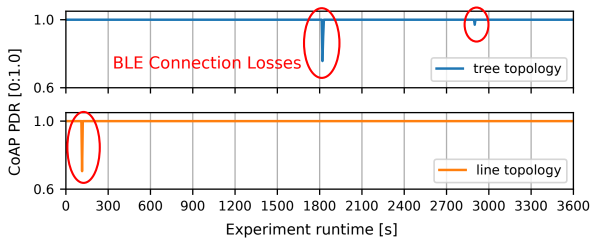
We assume a producer-consumer scenario to generate IP traffic. In every experiment, 14 nodes are configured as producers, while the root node of the tree and one node at the end of the line topology are configured as consumer nodes. Each producer periodically sends a CoAP non-confirmable GET request with a preconfigured payload of 39bytes to the consumer. The consumer node responses with a CoAP acknowledgment for each request it receives. To prevent all producers from sending their requests at the same time, a jitter is added to the producer interval. Overall, IP packet sizes are 100bytes, and final BLE packet sizes are 115bytes. Keeping the packet size below 128bytes allows for direct comparison of IEEE 802.15.4 and BLE, without introducing fragmentation [**?**].

If not stated differently, we use a default producer interval of 1s ±0.5s. We select this interval because it stresses the network sufficiently to reveal typical characteristics, without being effected by overflowing buffers (see § 5.2), and as it reflects typical IoT network load [25]. We further use a default BLE connection interval of 75ms as it provides a suitable trade-off between packet latency and scheduling collisions at the radio of a node.

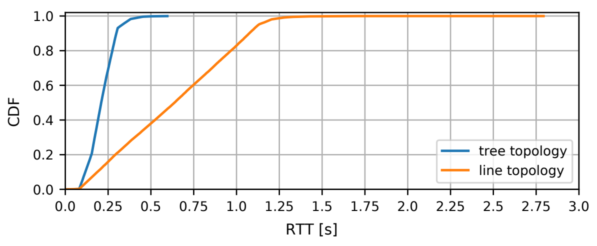
5 BASIC PERFORMANCE PROPERTIES

In this section, we analyze four key metrics, reliability, latency, throughput, and power consumption, to understand the basic performance of multi-hop IP over BLE networks. We mainly focus on (*i*) the packet delivery rate and (*ii*) the round trip time (RTT) of CoAP packets. (*i*) is defined as the ratio of CoAP ACKs received to the number of CoAP requests sent; (*ii*) is the time difference between the moment a CoAP request is handed to the network stack until the moment when the corresponding CoAP ACK is handed back to the application on the same node.

We vary the topologies, sending interval at producers, and the BLE connection intervals at all nodes to analyze different network scenarios. Experiments run for 1 hour (3600s), and we carefully verified that this runtime is sufficient to observe the effects discussed below. To rule out further side effects, every experiment is repeated 5× (see Appendix B). We also verified that the impact of the selected nodes does not bias our results. We measured the same results for each set of experiments. To analyze the results of each parameter set in detail, in this section, we show the results of a single 1 hour run per configuration.



(a) Relative CoAP packet delivery rate.



(b) Distributions of round trip times of CoAP messages.

Figure 7: Overview of typical reliability and latency characteristics for a tree and a line network topologies. Both experiments use a BLE connection interval of 75ms and a producer interval of 1s ±0.5s.

5.1 Moderate Network Load Scenario

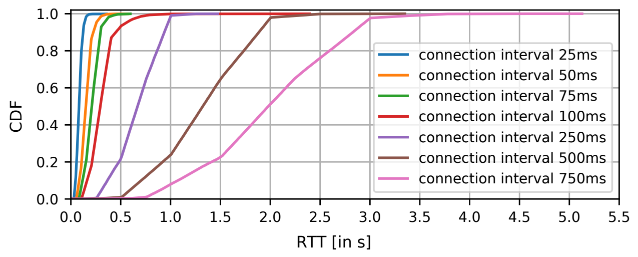
Figure 7(a) summarizes the reliability and latency for the tree and the line topologies, using a producer interval of 1s ±0.5s and a BLE connection interval of 75ms for each BLE connection. During the 1h experiment runtime, the tree and the line topologies experience a packet loss of 26 out of 50,527 and 20 out of 50,412 packets, leading to packet delivery rates of 99.949% and 99.960%, respectively. All packet losses can be attributed to intermediate BLE connection losses. While a link experiences connection loss, all data traffic that tries to traverse that link is dropped. In our setup, the packet loss is relatively small due to the quick reconnect mechanism implemented in the *statconn* connection manager. In real world deployments it is likely that this reconnection mechanism is slower, as one would relax the advertising and scanning parameters to optimize for power efficiency. It is worth noting that broken BLE links may lead to changes of the IP network topology on top, introducing possibly even larger delays due to the need for IP routing to reconfigure.

The latency of IP packets transferred over a single hop BLE link does not only depend on the BLE connection latency but also on the relative timing between the moment the packet is handed to the IP stack and the start of the next connection event [13]. This leads to a packet delivery latency that jitters in the range of a few milliseconds up to the used connection interval. For multi-hop configurations this effect accumulates for each link that a packet traverses. Figure 7(b) shows the RTT CDF of a particular experiment run. We observe the same characteristics for all other network configurations that we analyzed. The results show that most of the packet deliveries directly reflect the network topology (*i.e.,* path length) and the BLE connection interval. The RTT for packets in the line topology is a factor of 3.5 larger than for packets in the tree topology reflecting the average hop count (7.5 vs 2.1). A small number of packets (<3%), however, is prone to increased delays of multiple connection intervals. This effect is caused by link layer packet losses. Each time a link layer packet is lost, the BLE controller resends this packet one connection event later, adding a full connection interval to the packet latency.

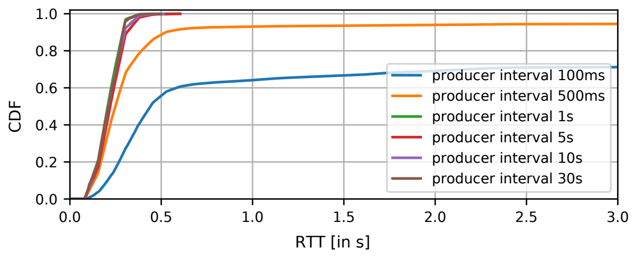
Delay and BLE connection interval:. Figure 8(a) illustrates the impact of the BLE connection interval on the CoAP RTT in a network forming the 3-hop tree topology. It can be observed, that the majority of the packets experiences a round trip time between a single connection interval and 4× the connection interval. As the average hop count in this particular topology is 2.14, these delays are as expected. Notable is once more the possible runaway delays for selected runs. Looking at the experiment with 100ms BLE connection interval, selected packets experience delays that are factor 22 higher then the connection interval.

Especially when looking at at round trip times when using larger BLE connection intervals, it becomes clear that the packet delay can easily grow into seconds, and therefore in the range of typical timeout value for IP protocols like CoAP or TCP. This can lead to unnecessary IP layer packet retransmissions if the original packet is not delivered.

In contrast to the BLE connection interval, the producer interval does not have significant impact on the packet delays, as long as the network is able to handle the applied packet load. Figure 8(b) illustrates that a fixed BLE connection interval of 75ms exhibits similar performance in terms of delay compared to producer intervals of 1s and larger. Slight variations can be explained by the impact of



(a) Varying BLE connection intervals under moderate load.



(b) BLE connection interval of 75ms under varying load.

Figure 8: Round trip times of CoAP messages in a tree topology.

background noise in the testbed. Only for producer intervals that trigger network loads exceeding the networks capacity, increased packet delays can be observed.

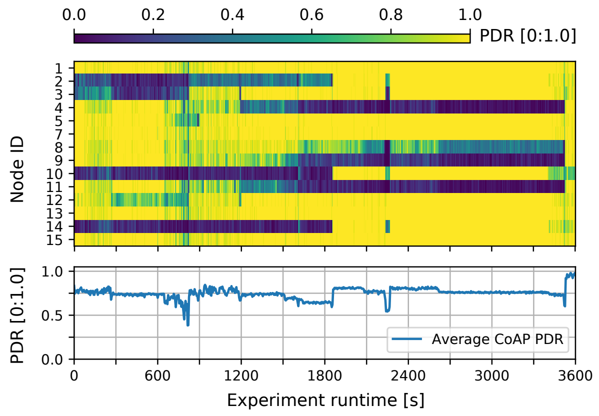
5.2 High Network Load Scenario

In this scenario, we put the network under stress by increasing the traffic until link layer capacities are exceeded and packet losses occur. The experiments are designed to understand where these losses occur, how they caused, and whether further effects such as increased delays become apparent.

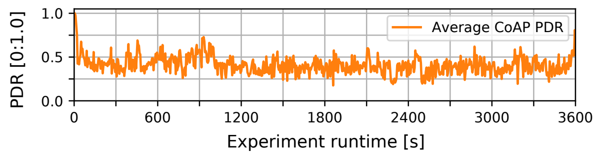
As a baseline using our BLE platform, we were able to achieve a raw L2CAP data throughput of close to 500kbps on a single link between two nrf52dk nodes. When setting the producer interval to 100ms ±50ms, in our tree topology, all 14 producer nodes generate a constant rate of 128.8kbps CoAP request packets towards the consumer node. Ideally, the consumer node would reply acknowledgment packets with a rate of 96.3kbps, in case every request arrives at the consumer in the first place. Although this traffic scenario requires at most 45% of the available capacity of a single link, significant packet losses already occur.

Figure 9 illustrates the relative PDR of one 1h experiment run for two different connection intervals, 75ms (top) and 2s (bottom). For a connection interval of 75ms, the average PDR is ≈ 75%. All packet losses can be attributed to overflowing packet buffers. When the capacity of selected links is saturated, packets that wait to be transmitted on those links are buffered and eventually dropped when buffer space exceeds. The results in Figure 9(a) illustrate two more typical behaviors, which we encountered in all load scenarios we measured. First, the PDR is distributed unevenly among the producers, clearly highlighted by larger fluctuations around the average PDR and clearly visible in the heatmap. This can be explained by the uneven distribution of the radio capacity across connections of a node (see § 2). The network load leads to links exceeding their capacity, which results into packet losses at both the node itself and all child nodes in the subtree. Second, we observe multiple occurrences of sudden PDR increases, most notably after ≈ 52min. The cause for the these jumps are beneficial reconnection events between nodes. By chance the new connection events are moved to time slots exhibiting a better position relatively to the events of the existing connections of a node. This leads to increased capacities of the affected links, and more transported packets increase the PDR of the affected producer and its child nodes.

The PDR decreases further when we increase the BLE connection interval to 2s and configure a producer interval of 1s ±0.5s (Figure 9(b)). The reason for this is the change from constant bit rate towards burst traffic, and burst traffic has even more negative impact on the link capacity and packet buffers. In detail, when IP packets are handed to the BLE stack, they have to wait for the next connection event to occur until they are transferred, and thus are queued during that time. An increase of the BLE connection interval now leads to longer time spans during which packets to be transferred accumulate. When the next connection event starts, all packets from the transmit queue are sent in one single event, hence leading to bursts of packets being transferred at once. In case of packet loss on the link layer, however, the Bluetooth standard defines that a connection event is aborted, even though there are still packets waiting to be transferred. As the probability of link layer packet loss is increasing with the number of packets that are exchanged in a single connection event, the probability of aborting connection events increases as well, preventing links from reaching their capacity.



(a) CoAP PDR for producer interval of 100ms ±50ms and connection interval of 75ms.



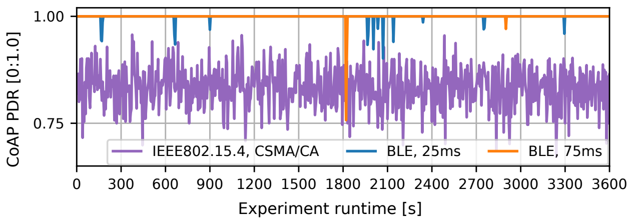
(b) CoAP PDR for BLE connection interval of 2000ms and producer interval of 1s ±0.5s.

Figure 9: Effects of high network load and slow connection intervals on CoAP packet delivery rates in tree topology.

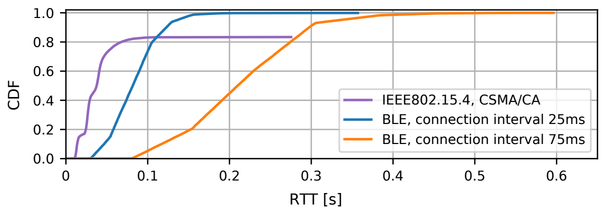
5.3 Comparison with IEEE 802.15.4

We compare multi-hop BLE with IEEE 802.15.4 for two reasons. First, due to historic development and its simplicity to use, IEEE 802.15.4 is the default deployment option in most low-power IoT-centric software platforms and it is the common denominator in existing performance evaluations [17, 25, 26, 35, 36]. Second, IEEE 802.15.4 exhibits different link layer properties compared to multi-hop BLE. It deploys a CSMA/CA media access (instead of time-slotted channel hopping) and a lower data rate 250kbps (instead of 1Mbps). To implement a fair comparison, we benefit from the abstraction layers of our implementation. These enable us run the same benchmark application from our BLE experiments on a set of 15 *m3* nodes, which support IEEE 802.15.4 instead of BLE.

Figure 10 illustrates typical results in terms of reliability and latency of CoAP packets using BLE and IEEE 802.15.4. In the tree topology with moderate traffic, the IEEE 802.15.4-based network is operating on capacity limits, leading to a PDR of 83.3% on average. The BLE network shows a PDR above 99% in the same scenario, with packet losses only appearing due to BLE connection losses. The packet delays of BLE are mainly defined by the connection interval in use, which influences the packet delay of each hop. Each time a packet is retransmitted on the link layer, a full connection interval is added to the latency of the packet delivery. The delay of



(a) Packet delivery rates of CoAP messages.



(b) Round trip times of CoAP messages.

Figure 10: Comparison of BLE and IEEE 802.15.4, using the same tree topology and 1s ±0.5s sending interval.

the IEEE 802.15.4 network, however, is mainly influenced by the backoff timers of the radio, which have significantly smaller values than the connection intervals used for BLE, resulting in shorter transmission delays when packets are delivered. In contrast to BLE, however, packets are dropped after a defined number of retries leading to a higher packet loss in IEEE 802.15.4 (see Figure 10(b)).

5.4 Energy Efficiency

To better understand whether battery-driven BLE multi-hop routers are a feasible deployment option, we measure the average current consumption for a single node in different connection states and scenarios for a *nrf52dk* board using the Nordic Power Profiler Kit.[2]

For each single connection event we measured a used charge of 2.3μC on the coordinator node, and a charge of 2.6μC on the subordinate node. Considering, for example, a connection interval of 75ms, a single idle connection adds 30.7μA or 34,7μA to a node's average current consumption, depending on the node's role.

In our medium traffic load scenario (*i.e.,* producer interval of 1s ±0.5s, connection interval 75ms), a subordinate that acts as forwarder with three active connections shows an additional current consumption of 123μA caused by the BLE connections. Adding this additional current consumption to the board's average idle current consumption of 15μA does allow to run this configuration for 69 days on a 230mAh coin cell battery or little over 2 years on a 2500mAh 18650 cell. This shows that battery-powered IP routers are a feasible option.

To highlight the usability of multi-hop BLE in battery-driven scenarios, we compare the energy consumption of IP over BLE nodes with plain connection-less BLE beacons. For a BLE node configured as beacon sending the maximal usable payload of 31bytes and configuring an advertising interval of 1s, we measure an increased current consumption of 12μA compared to the node in idle mode. When configuring the same node as an IP over BLE coordinator with a single open connection and letting the node send a CoAP packet with the same payload every 1s, the average current consumption increases by 16μA. This shows that—despite a considerable overhead in software complexity and CPU processing time—IP over BLE nodes can compete with plain BLE beacons in terms of energy while at the same time offering more network features and reliability.

[2]https://www.nordicsemi.com/Software-and-tools/Development-Tools/Power-Profiler-Kit

5.5 Implementation vs. Protocol Impact

All experiments are based on the BLE platform introduced in this paper. To the best of our knowledge, there is no other open or closed source implementation that supports multi-hop IP over BLE. This prevents us from running comparative experiments. We want to emphasize, though, that the observed basic performance characteristics are specific to the BLE protocol design and independent of implementation details. Other implementations could use different buffer sizes and thread priorities. In case of high network load, when broken connections are more likely, increased buffer sizes could temporarily mitigate losses and different thread priorities could shift the load between BLE stack and IP stack buffers. Those specifics do not change our observations that connections drop randomly. We will detail the fundamental reasons behind in the next section.

6 HIDDEN IMPACT OF BLE CONNECTION INTERVAL

6.1 Connection Shading

In § 5, we identified two characteristic properties in multi-hop IP over BLE networks that challenge reliability: (*i*) random connection drops and (*ii*) spontaneous bandwidth reduction on links. Both can be explained by a phenomenon we coin *connection shading*. Connection shading occurs on nodes that have multiple active connections, of which they are in the subordinate role for at least one of them. As described in § 2, the link capacity of a single connection depends on the maximum length of its connection events, which is restricted by the positioning of these connection events relative to the connection events of the node's other connections in time. In an ideal world, this effect would statically define the capacity for each connection when a connection is opened. In practice, the connection intervals are subject to clock drifts, which leads to distorted intervals and changing capacities.

The Bluetooth standard mitigates the effects of clock drift with respect to a single connection by defining quality gates for the used clocks and by applying a measure called *window widening* on connection subordinates. This measure tells a connection subordinate to start listening for an incoming packet a defined amount of time before it considers the next connection event to start and keeps listening for the same time span after the connection event should have happened. It ensures that the subordinate is always receiving packets from the coordinator despite the nodes clocks drifting apart over time. On the downside, window widening may lead

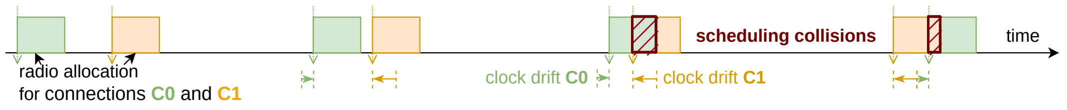


Figure 11: Radio allocation on a subordinate node with two BLE connections. Both connections are subject to clock drift leading to *connection shading* due to radio scheduling collisions.

to negative effects in scenarios in which a subordinate maintains connections to multiple controllers. It is worth recalling that the starting point of each connection event is determined solely by a connection coordinator, and hence it is subject to the coordinator's clock drift only. When a node maintains multiple connections and the node is the coordinator for all of them, the connection intervals for all these connections will deviate by the same amount as they are subject to the same clock drift (Figure 11). If a node maintains multiple connections and at least one of them is coordinated by another node, the connection intervals are subject to different clock drifts. Subsequently, this leads to connection events of the separate connections that move relatively to each other in time.

For example, we consider a node that maintains two connections A and B with a connection interval of 100ms. This node is coordinator for connection A, and the subordinate for connection B. As starting point we consider the ideal situation where the initial connection event for connection B is scheduled to be 50ms after the first connection event of connection A. In this situation, both connections can utilize exactly half of the available link capacity. Unfortunately, as described above, both connections A and B are subject to different clock drifts, leading their connection events shifting relatively to each other in time. If we consider a typical relative clock drift for A and B of 36ms per hour, after 1h the connection events will have shifted in a way that connection events of B are scheduled 86ms after the events of A. Now, the theoretical link capacity is split into 86% and 14% for A and B, respectively. Connection B therefore experiences a continuous degradation of link capacity. After some more time the clock drift leads to the situation where the connection events for both connections start to overlap. The radio can only service one connection event at a time, though, so the controller is left with two choices: (*i*) it can decide to only schedule one of the two connections and skip all connection events for the other connection, or (*ii*) it can schedule both connection events in an alternating fashion. To be reliable choice (*i*) implies that there are no packets to exchange for a certain duration of time via the unscheduled connection. If that amount of time is larger than the connection's supervision timeout, both peers declare the connection as dead and drop it. This is the reason for the random connection losses we observe in § 5. Using choice (*ii*), packets are transferred only at every second connection event, leading to increased transmission delays and a sudden drop in link capacity.

To illustrate the example above, Figure 12 shows link degradation for a single node based on our experiment that configured all connection intervals of 75ms. The node under investigation maintains a connection to the consumer node and attains the role of the coordinator. The consumer node maintains a total of three connections as subordinate. After an experiment runtime of 3100s the link layer packet delivery rate for the observed link starts to

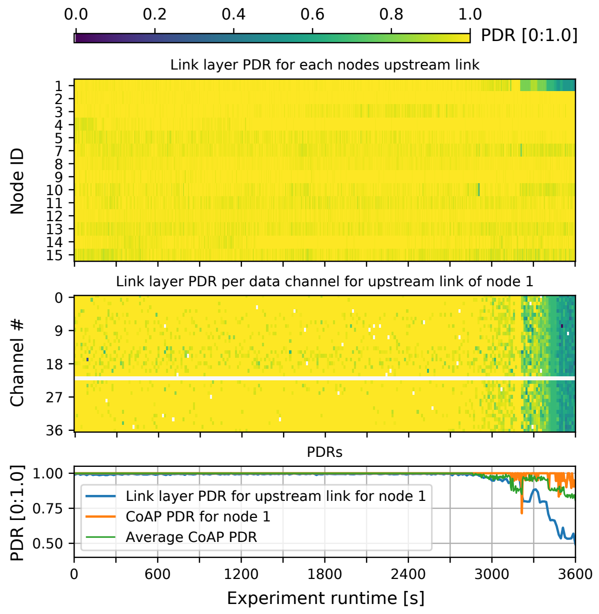


Figure 12: Example for link degradation in a tree topology. During the 1h experiment the upstream link of nrf52dk-1 is subject to connection shading, leading to a drop of link layer and CoAP PDRs.

drop significantly until only ≈ 50% of the link layer packets are transferred. From the perspective of the application layer, the CoAP PDR drops as depicted from 100% to 94% for the producer using this link as upstream route, but as expected also for the producers having this link on their path to the consumer node. The PDR per channel shows clearly that the reliability decreases evenly on all data channels with the appearance of this event, which is as expected when the subordinate of link starts to skip connection events.

6.2 How Likely is Connection Shading?

Two conditions must be fulfilled such that connection shading occurs. (*i*) a node must maintain at least two parallel connections with the same connection interval and (*ii*) the node must be the connection subordinate for at least one of those connections. Then, the probability of two connections shading each other depends on the configured connection interval *and* the relative clock drift of the two clocks controlling the connection events. The clock responsible for scheduling the beginning of a connection event is called the *sleep clock* by the Bluetooth standard. For this clock the standard demands a clock accuracy less than 250ppm. In the worst case, this can lead to a relative clock drift of 500μs per second.

In practice, when measuring the relative clock drift between five different nrf52dk boards, we encountered a maximum relative clock drift 6μs per second.

Assuming that the clock drift is constant over a given period of time, the maximum time it takes until the connection events of two connections will overlap can be computed as $ConnItvl/ClkDrift$ with $ConnItvl$ being the BLE connection interval and $ClkDrift$ the relative clock drift of the two clocks timing each connection. In the worst case, considering the least allowed connection interval of 7.5ms and a relative clock drift of 500μs per second, this leads to two connections shading each other every 15s, and thus leading to 240 shading situations per hour. To put this into perspective: a typical scenario using a connection interval of 75ms and experiencing a relative clock drift of 5μs per second, two connections will shade each other every 4.17h, resulting in 0.24 shading events per hour. Although this value is significantly lower than the worst case, it still means that in a deployment with multiple active BLE connections, there will be many shading events per day, each leading to link degradation and possible connection losses.

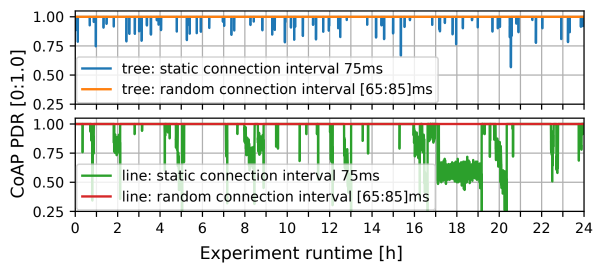
The tree topology used in this work is composed of 14 BLE connections. If we apply the probability of 0.24 shading events per hour to these 14 links, we should see an average of 3.4 shading events per hour or 80.6 events per 24h. The results for the 24h experiment with a static connection interval of 75ms presented in Figure 13(a) show that the nodes experience an overall of 95 connection losses during the experiment runtime. Considering that we do not have exact information about the specific relative clock drift for each pair of nodes, the estimated probabilities seem to be valid.

6.3 Mitigation of Connection Shading: Randomize BLE Connection Intervals

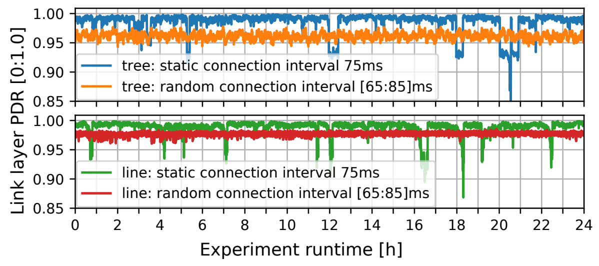
To prevent connection shading, the network needs to ensure that any two connections at any node do not use the same connection interval. In practice, this is challenging to achieve because the coordinator, who is initiating a new connection, dictates the connection interval that is going to be used. The coordinator, however, has neither knowledge about the existence nor the used intervals of any connection the subordinate peer maintains—note, the Bluetooth standard does not offer any means to gather that information from its peer.

Design space. Choosing a non-colliding connection interval is guessing for the coordinator. After the connection was established, the Bluetooth standard allows peers to update the used connection parameters and with those the connection interval for active connections. In cases where connection intervals on a subordinate node would collide, that node could in theory use this update mechanism to set the connection interval to a non-colliding value. As of Bluetooth 4.2, this mechanism has the drawback that it works without negotiation, which gives the peer no option to decline the new set of parameters. This might lead to ongoing link reconfigurations, where nodes constantly change connection intervals to values that collide at their peer nodes. Bluetooth 5.0, on the other hand, introduces a new mechanism to update the connection parameters, which allows for actual parameter negotiation between both peers. In theory that could be used to find a connection interval value that fits both. However, the response packet for such an update request is crafted and filled by the BLE controller and it is up to a controller implementation to prevent connection intervals to be used multiple times. The problem is that the majority of the BLE controllers are black boxes allowing interaction only through the defined HCI interface. This interface does not allow to influence the update mechanism mentioned above, hence, the controller behavior cannot externally be influenced in this regard.

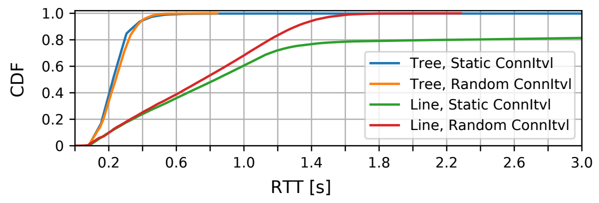
Proposal. We propose to randomize connection intervals in a given window when opening new connections, to prevent the use of the same connection intervals on a node. The coordinator of the new connection randomly selects the connection interval from a predefined window that is chosen around the target connection interval, *e.g.,* for a 75ms target interval the window could be from 65ms to 85ms. The specific window should be chosen small enough, so that a link's performance characteristics (e.g. packet delay) behave close to the target value, and large enough to allow for a sufficient spacing of connection intervals. The minimum window size must be larger than a node's maximum number of connections multiplied by the minimum spacing between connection intervals. There is no upper bound on the window size but choosing the window size



(a) Average CoAP packet delivery rates.



(b) Average link layer packet delivery rates.



(c) Distributions of round trip times of CoAP messages.

Figure 13: Comparing the impact of fixed (standard BLE mesh) and randomized (our proposal) BLE connection intervals in tree and line topologies in 24h experiments.

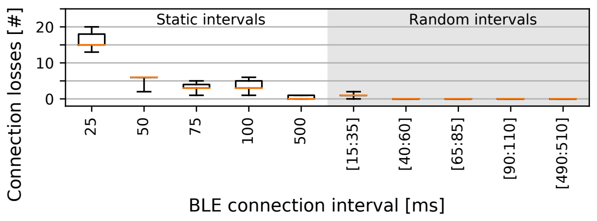


Figure 14: Distribution of BLE connection losses for 1s producer interval using different BLE connection intervals. Each configuration ran for 5×1h.

too large will have a randomizing effect on link performance, as the minimum possible value from the window will behave significantly different than the maximum possible value (see Figure 8(a)). The variation of connection intervals leads to connection events of different connections moving relative to each other in time with an offset much larger then the one introduced by the clock drift. This does not prevent single connection events to collide with each other, but it does prevent connection shading. It is worth noting that our proposal is standard compliant and does not depend on the capabilities of a BLE controller or a BLE stack.

Implementation. To verify our proposal we implemented this randomization in the *statconn* connection manager (see § 3). For comparison reasons with a network that does not introduce colliding connection intervals, we further enhanced the connection interval selection in two aspects. First, each time a coordinator generates a new connection interval to be used for a new connection, that interval is compared to a list of already used intervals and regenerated until it is unique for that node. Second, on each newly opened connection the subordinate compares the connection interval with the intervals of all its other connections. If the interval of the connection collides, the subordinate closes the new connection immediately. This forces the coordinator, triggered by the used *statconn* connection manager, to reopen the connection with a new randomly generated connection interval. This is repeated until the connection interval for each of a node's connections is unique.

Evaluation. The impact of the randomization strategy is notable. In all experiments, we do not observe BLE connection loss related to connection shading. Furthermore, the spacing of connection intervals balances link capacities. Figure 14 illustrates the distribution of connection losses for multiple 1h experiment runs for different configurations of producer and BLE connection intervals. The experiments using randomized connection intervals (highlighted in gray) did largely not suffer from BLE connection losses. When we encounter BLE connection losses in this configuration, this was for a combination of small BLE connection intervals and high network load. The raw data of the experiments in question does suggest that these connection drops are caused by external interferences and not by connection shading.

As a more detailed example for the positive impact of this approach, Figure 13 illustrates the changes in reliability and latency comparing a fixed and random BLE connection interval in a star and a tree topology. Both configurations were deployed for 24h using a producer interval of 1s ±0.5. Figure 13(a) shows the differences of CoAP packet delivery rates. While the network configuration with a static connection interval of 75ms is experiencing a number of connection losses and with these a certain degree of packet loss, the configuration using random connection intervals between 65ms and 85ms does not encounter any connection losses—out of over 1,200,000 CoAP requests sent in both topologies not a single CoAP packet was lost. Using randomized connection intervals decreases the overall link layer packet delivery rates slightly from 98% to 96% in the tree and 99% to 98% in the star topology. This decrease is a trade-off with respect to a stable link behavior that does not suffer from temporal link degradations (see Figure 13(b)). Comparing the round trip times in Figure 13(c), the results show that randomizing the connection interval leads to slightly increased delays in the tree topology due to the increase in link layer retransmissions. At the same time randomizing connection intervals show a lower upper bound on round trip times for the slowest 1% of packets, leading to a more deterministic timing behavior in the network. In the line topology, the delays are slightly lower while we observe the same positive characteristics of the worst case round trip times.

7 RELATED WORK

In parallel to the IP over BLE standardization defined in the IETF, the Bluetooth Special SIG presented Bluetooth Mesh [9] as an alternative approach of implementing multi-hop topologies based on BLE. Darroudi *et al.* [14] show that this proposal applies a very different approach on mesh networking. One major drawback is that Bluetooth Mesh does not support transport of IP data and thus cannot be transparently integrated into the Internet.

The first real-world analysis of *6BLEMesh* running *6BLEMesh* on low-power nodes in a multi-hop environment was published by Darroudi and Gomez [13]. They show the impact of BLE connection parameters on transmission delays and power consumption. The experiments are, however, limited to a setup of 3 nodes and 2 hops. Spörk *et al.* [40] present similar results using comparable hardware but focus on single-hop scenarios only. Recent work focuses on the optimization of BLE links between two nodes [39, 41]. Combining *adaptive frequency hopping* (AFH) with careful exclusion of channels can mitigate the effects of interference on the crowded 2.4GHz frequency spectrum, reducing packet loss and significantly improving link delays. Although the suggested improvements are not applied in this work, these findings suggest that large scale 6BLEMesh applications would also benefit. Further work describing experiments to analyze the performance properties of IP over BLE is not focusing on low-power platforms [11, 27, 29].

Other experiments and simulations measure the raw throughput of BLE links [6, 15, 19, 21]. All studies conclude that the maximum data rate is capped at roughly 220 kbps. Those studies, however, are based on older Bluetooth standards and do not take the *data length extension* and the 2Mbps modes into account. More recent work [10] shows that current versions of BLE can achieve data rates of up to 1300kbps.

Siekkinen *et al.* [38] show that BLE can achieve significant lower energy consumption per bit transmitted compared to IEEE 802.15.4, which is confirmed in [13, 29, 40]. Based on simulations, Feeney *et al.* [16] find that the phase difference between co-located, beacon-enabled IEEE 802.15.4 networks combined with clock drift create

Table 2: Open source IP over BLE (IoB) implementations.

	●=supported ○=not supported			
Implementation	Hardware portability	GATT Service	IoB single-hop	IoB multi-hop
RIOT + NimBLE [3]	●	●	●	●
BLEach [2, 13, 40]	○	○	●	○
Zephyr [4]	●	●	●	○

tempoaral disconnections and packet losses. Although observed in a different context, the effects are similar to connection shading introduced in this work. The impact of a contention-based IEEE 802.15.4 link layer on application protocols such as CoAP was analyzed in detail by Gündogan *et al.* [26].

There are only two other open source implementations available supporting IP over BLE, *BLEach* based on Contiki [2, 13, 40] and Zephyr [4] (see Table 2). BLEach only provides a subset of BLE features needed for IP over BLE. It does not include *GATT* server support and therefore does not comply with the *Internet Service Support Profile*. It is also limited to a small set of hardware platforms. Zephyr features both a full featured BLE and and a 6LoWPAN stack, supporting a wide range of BLE-enabled hardware. Currently both implementations do not support multi-hop IP over BLE, the code used in [13] is not publicly available. An additional reason for providing a new platform is that we also believe that diversity helps to build a reliable IoT ecosystem.

8 DISCUSSION

In this section, we discuss our main findings guided by the question: what is still missing to successfully adapt IP over BLE in the IoT?

Software support. A successful adoption of IP over BLE in the IoT requires wide software support. A major reason for the lack of BLE-6LoWPAN networking is implementation complexity because two complex software modules are needed, a BLE stack and an IPv6 stack. The software platform presented in this work is simple to use and portable, and demonstrates that an established open source Bluetooth stack can be utilized to add BLE capabilities to an existing 6LoWPAN stack if carefully modeled.

Although the NimBLE BLE stack is well established and tested, we found a number of problems in its code base, which caused stability and performance issues. The main reason for these issues to be unnoticed is that the L2CAP connection oriented channels are rarely used in current BLE use cases. The majority of bugs that we encountered were concurrency issues in NimBLE's L2CAP implementation leading to corrupt states and runtime failures. We were able to expose most of the issues due to the multi-threaded architecture of the RIOT network stack, which challenges the thread-safety of software modules. We contributed fixes to NimBLE.

Our BLE implementation shows that multi-hop IP over BLE is a viable solution on top of a multi-purpose operating system for resource constrained IoT devices. Memory requirements of the configuration used in our evaluation (*i.e.,* 140kB of ROM and 58kB of RAM) are on par with modern BLE-capable SoCs. This is an upper bound because our setup contains extensive logging capabilities. To the best of our knowledge, the two dominant platforms for more powerful mobiles, iOS and Android, however, do not support IP over BLE functionality, yet. Their BLE stacks do not expose L2CAP APIs either. We hope that the results of this paper motivate additional activities since broad support will enable more comprehensive IoT scenarios in the future.

Connection intervals should be unique. In multi-hop BLE, a large portion of link layer packet losses can be attributed to local scheduling collisions of the radio on nodes maintaining multiple open connections. A connection interval that is the same for multiple connections combined with common clock drifts on IoT devices result into connection shading, *i.e.,* connection events start to overlap. We showed that randomizing connection intervals on all nodes (and guaranteeing different intervals for all connections per node) solves this issue without introducing additional resource requirements. In general, connection shading is not unique to BLE and can be observed in other time-slotted networks, but due to BLEs strict timing requirements connection shading effects are particularly visible in multi-hop BLE networks.

Carefully choose the connection interval length. From an energy consumption perspective, larger connection intervals are beneficial. The increase of connection intervals, however, leads to an increase of buffer space since outgoing packets need to be queued for transfer until the next connection event starts. Once the buffer space is exceeded, the network reliability declines and packet delays increase significantly. We observed this behavior even in scenarios with a medium network load. Therefore, the length of the connection interval should be configured based on the BLE and IP packet buffer sizes available.

Further care has to be taken when stateful protocols such as CoAP or TCP run on top. Connection intervals in the order of seconds usually conflict with default retransmission timeouts of those protocols. Eventually, this can cause a significant increase in network load due to network layer retransmissions, although the original requests were never lost and are delivered successfully.

9 CONCLUSION

BLE is considered a promising deployment option for low-power IoT scenarios as it offers popular network access available in the mass market. To unfold full potential in future mesh scenarios, multi-hop IP over BLE is needed, though. In this paper, we shed light on the performance of multi-hop IP over BLE from the perspective of constrained devices in realistic IoT scenarios. Based on our software platform, we found that packet delivery and delay can be easily downgraded in mid-sized networks. We proposed a mitigation strategy that is standard compliant and copes with local system peculiarities of IoT devices such as clock-drift and constrained memory. Using a random connection interval of moderate length allowed us to deploy a fully reliable network access without sacrificing energy or round trip times. In the future, we plan to expand the scope to include mobile systems. In particular we identified the management of BLE topologies, the coupling of BLE topologies with IP routing, and the adaptability of IP over BLE networks to dynamic environments and mobile nodes as open research questions.

ACKNOWLEDGMENTS

We would like to thank our shepherd Hamed Haddadi and the anonymous reviewers for their valuable feedback. We are also grateful to Laura Feeney for her very helpful feedback on our artifacts.

Our appreciation goes to Andrzej Kaczmarek and Szymon Janc for their invaluable help with the NimBLE stack. This work was partly supported by the German Federal Ministry of Education and Research (BMBF) within the projects RAPstore and PIVOT.

REFERENCES

[1] [n.d.]. *Apache NimBLE.* https://github.com/apache/mynewt-nimble

[2] [n.d.]. *BLEach IPv6-over-BLE stack.* http://spoerk.github.io/contiki/

[3] [n.d.]. *RIOT OS.* https://github.com/RIOT-OS/RIOT

[4] [n.d.]. *The Zephyr Project: Zephyr OS.* https://zephyrproject.org

[5] Cedric Adjih, Emmanuel Baccelli, Eric Fleury, Gaetan Harter, Nathalie Mitton, Thomas Noel, Roger Pissard-Gibollet, Frederic Saint-Marcel, Guillaume Schreiner, Julien Vandaele, and Thomas Watteyne. 2015. FIT IoT-LAB: A large scale open experimental IoT testbed. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT).* IEEE Press, Piscataway, NJ, USA, 459–464.

[6] A. Ancans, J. Ormanis, R. Cacurs, M. Greitans, E. Saoutieff, A. Faucorr, and S. Boisseau. 2019. Bluetooth Low Energy Throughput in Densely Deployed Radio Environment. In *Proc. of 23rd International Conference Electronics.* IEEE, 1–5. https://doi.org/10.1109/ELECTRONICS.2019.8765577

[7] Anonymous. [n.d.]. Due to double-blind reviewing. Source code will be released on publication.

[8] Emmanuel Baccelli, Cenk Gündogan, Oliver Hahm, Peter Kietzmann, Martine Lenders, Hauke Petersen, Kaspar Schleiser, Thomas C. Schmidt, and Matthias Wählisch. 2018. RIOT: an Open Source Operating System for Low-end Embedded Devices in the IoT. *IEEE Internet of Things Journal* 5, 6 (December 2018), 4428–4440. http://dx.doi.org/10.1109/JIOT.2018.2815038

[9] Bluetooth Special Interest Group. 2019. *Mesh Profile.* Bluetooth Specification 1.0.1. Bluetooth SIG. https://www.bluetooth.com/specifications/mesh-specifications/

[10] Patricio Bulić, Gašper Kojek, and Anton Biasizzo. 2019. Data Transmission Efficiency in Bluetooth Low Energy Versions. *Sensors* 19, 17 (2019). https://doi.org/10.3390/s19173746

[11] J. Campos, S. Colteryahn, and K. Gagneja. 2018. IPv6 transmission over BLE Using Raspberry PI 3. In *2018 International Conference on Computing, Networking and Communications (ICNC).* 200–204. https://doi.org/10.1109/ICCNC.2018.8390350

[12] Seyed Mahdi Darroudi and Carles Gomez. 2017. Bluetooth Low Energy Mesh Networks: A Survey. *Sensors* 17, 7 (2017), 19 pages. https://doi.org/10.3390/s17071467

[13] Seyed Mahdi Darroudi and Carles Gomez. 2020. Experimental Evaluation of 6BLEMesh: IPv6-Based BLE Mesh Networks. *Sensors* 20, 16 (2020). https://doi.org/10.3390/s20164623

[14] S. M. Darroudi, C. Gomez, and J. Crowcroft. 2020. Bluetooth Low Energy Mesh Networks: A Standards Perspective. *IEEE Communications Magazine* 58, 4 (2020), 95–101. https://doi.org/10.1109/MCOM.001.1900523

[15] F. J. Dian, A. Yousefi, and S. Lim. 2018. A practical study on Bluetooth Low Energy (BLE) throughput. In *Proc. of IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON).* IEEE, 768–771. https://doi.org/10.1109/IEMCON.2018.8614763

[16] Laura Marie Feeney and Viktoria Fodor. 2016. Reliability in Co-Located 802.15.4 Personal Area Networks. In *Proc. of the 6th ACM International Workshop on Pervasive Wireless Healthcare (MobiHealth '16).* ACM, New York, NY, USA, 5–10. https://doi.org/10.1145/2944921.2944923

[17] G. Gardasevic, S. Mijovic, A. Stajkic, and C. Buratti. 2015. On the performance of 6LoWPAN through experimentation. In *2015 International Wireless Communications and Mobile Computing Conference (IWCMC).* 696–701. https://doi.org/10.1109/IWCMC.2015.7289168

[18] Muhammad Rizwan Ghori, Tat-Chee Wan, and Gian Chand Sodhy. 2020. Bluetooth Low Energy Mesh Networks: Survey of Communication and Security Protocols. *Sensors* 20, 12 (2020). https://doi.org/10.3390/s20123590

[19] D. Giovanelli, B. Milosevic, and E. Farella. 2015. Bluetooth Low Energy for data streaming: Application-level analysis and recommendation. In *Proc. of 6th International Workshop on Advances in Sensors and Interfaces (IWASI).* IEEE, 216–221. https://doi.org/10.1109/IWASI.2015.7184945

[20] Carles Gomez, Seyed Darroudi, Teemu Savolainen, and M. Spörk. 2020. *IPv6 Mesh over Bluetooth(R) Low Energy using IPSP.* Internet-Draft – work in progress 09. IETF.

[21] C. Gomez, I. Demirkol, and J. Paradells. 2011. Modeling the Maximum Throughput of Bluetooth Low Energy in an Error-Prone Link. *IEEE Communications Letters* 15, 11 (2011), 1187–1189. https://doi.org/10.1109/LCOMM.2011.092011.111314

[22] Bluetooth Special Interest Group. 2014. *Internet Protocol Support Profile.* Bluetooth Specification 1.0.0. Bluetooth SIG. https://www.bluetooth.com/specifications/gatt/

[23] Bluetooth Special Interest Group. 2019. *Bluetooth Core Specification.* Bluetooth Specification 5.2. Bluetooth SIG. https://www.bluetooth.com/specifications/bluetooth-core-specification

[24] Bluetooth Special Interest Group. 2020. *Bluetooth Market Update 2020.* Technical Report. Bluetooth SIG. https://www.bluetooth.com/bluetooth-resources/2020-bmu/

[25] Cenk Gündogan, Peter Kietzmann, Martine Lenders, Hauke Petersen, Thomas C. Schmidt, and Matthias Wählisch. 2018. NDN, CoAP, and MQTT: A Comparative Measurement Study in the IoT. In *Proc. of 5th ACM Conference on Information-Centric Networking (ICN).* ACM, New York, NY, USA, 159–171. https://doi.org/10.1145/3267955.3267967

[26] Cenk Gündogan, Peter Kietzmann, Martine S. Lenders, Hauke Petersen, Michael Frey, Thomas C. Schmidt, Felix Shzu-Juraschek, and Matthias Wählisch. 2021. The Impact of Networking Protocols on Massive M2M Communication in the Industrial IoT. *IEEE Transactions on Network and Service Management (TNSM)* (2021). https://doi.org/10.1109/TNSM.2021.3089549

[27] Haolin Wang, Minjun Xi, Jia Liu, and Canfeng Chen. 2013. Transmitting IPv6 packets over Bluetooth low energy based on BlueZ. In *Proc. of 15th International Conference on Advanced Communications Technology (ICACT).* IEEE, Piscataway, NJ, USA, 72–77.

[28] Mehrdad Hessar, Ali Najafi, Vikram Iyer, and Shyamnath Gollakota. 2020. TinySDR: Low-Power SDR Platform for Over-the-Air Programmable IoT Testbeds. In *Proc. of 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI).* USENIX Association, Santa Clara, CA, 1031–1046.

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

[30] Douglas J. Leith and Stephen Farrell. 2020. Coronavirus Contact Tracing: Evaluating the Potential of Using Bluetooth Received Signal Strength for Proximity Detection. *SIGCOMM Comput. Commun. Rev.* 50, 4 (Oct. 2020), 66–74.

[31] Martine Lenders, Peter Kietzmann, Oliver Hahm, Hauke Petersen, Cenk Gündoğan, Emmanuel Baccelli, Kaspar Schleiser, Thomas C. Schmidt, and Matthias Wählisch. 2018. *Connecting the World of Embedded Mobiles: The RIOT Approach to Ubiquitous Networking for the Internet of Things.* Technical Report arXiv:1801.02833. Open Archive: arXiv.org. https://arxiv.org/abs/1801.02833

[32] Élodie Morin, Mickael Maman, Roberto Guizzetti, and Andrzej Duda. 2017. Comparison of the Device Lifetime in Wireless Networks for the Internet of Things. *IEEE Access* 5 (2017), 7097–7114. https://doi.org/10.1109/ACCESS.2017.2688279

[33] J. Nieminen, T. Savolainen, M. Isomaki, B. Patil, Z. Shelby, and C. Gomez. 2015. *IPv6 over BLUETOOTH(R) Low Energy.* RFC 7668. IETF.

[34] Shahid Raza, Prasant Misra, Zhitao He, and Thiemo Voigt. 2017. Building the Internet of Things with Bluetooth Smart. *Ad Hoc Networks* 57 (2017), 19–31.

[35] Damien Roth, Julien Montavont, and Thomas Noel. 2012. Performance Evaluation of Mobile IPv6 over 6LoWPAN. In *Proceedings of the 9th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks* (Paphos, Cyprus) *(PE-WASUN '12).* Association for Computing Machinery, New York, NY, USA, 77–84. https://doi.org/10.1145/2387027.2387041

[36] Mohamed Seliem, Khaled Elsayed, and Ahmed Khattab. 2017. Optimized Neighbor Discovery for 6LoWPANs: Implementation and Performance Evaluation. *Computer Communications* 112 (08 2017). https://doi.org/10.1016/j.comcom.2017.08.013

[37] Z. Shelby, K. Hartke, and C. Bormann. 2014. *The Constrained Application Protocol (CoAP).* RFC 7252. IETF.

[38] 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 IEEE Wireless Communications and Networking Conference Workshops (WCNCW).* IEEE, Piscataway, NJ, USA, 232–237. https://doi.org/10.1109/WCNCW.2012.6215496

[39] Michael Spörk, Carlo Alberto Boano, and Kay Römer. 2019. Improving the Timeliness of Bluetooth Low Energy in Noisy RF Environments. In *Proc. of International Conference on Embedded Wireless Systems and Networks (EWSN)* (Beijing, China). ACM, Junction Publishing, USA, 23–34.

[40] 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 *Proc. of the 15th ACM Conference on Embedded Network Sensor Systems* (Delft, Netherlands) *(SenSys '17).* ACM, New York, NY, USA, Article 2, 14 pages. https://doi.org/10.1145/3131672.3131687

[41] Michael Spörk, Jiska Classen, Carlo Alberto Boano, Matthias Hollick, and Kay Römer. 2020. Improving the Reliability of Bluetooth Low Energy Connections. In *Proc. of International Conference on Embedded Wireless Systems and Networks on* (Lyon, France) *(EWSN '20).* ACM, Junction Publishing, USA, 144–155.

[42] Nicole Todtenberg and Rolf Kraemer. 2019. A survey on Bluetooth multi-hop networks. *Ad Hoc Networks* 93 (2019), 101922. https://doi.org/10.1016/j.adhoc.2019.101922

[43] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, JP. Vasseur, and R. Alexander. 2012. *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks.* RFC 6550. IETF.

[44] Ding Yi, Ling Liu, Yu Yang, Yunhuai Liu, Tian He, and Desheng Zhang. 2021. From Conception to Retirement: a Lifetime Story of a 3-Year-Old Operational Wireless Beacon System in the Wild. In *Proc. of 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI).* USENIX Association, Berkeley, CA, USA, 859–872.

A ARTIFACTS

This section provides an overview of the artifacts produced for and used in this paper. The artifacts include software that implements our proposed solution, to run the experiments, and to analyze gathered data, as well as the raw data analyzed in this paper. Using the artifacts, you should be able to reproduce the results on real hardware in the FIT IoTlab testbed.

All conducted experiments are based on the open source IP over BLE platform presented in the paper (see § 3 and § A.2). The experiments are automated using a custom experimentation framework based on static configuration files that specify the details for each experiment run (see § 4 and § A.3). In addition to the experiment descriptions we made all raw data used in the paper available (see § 5, § 6, and § A.4).

A.1 Hosting

All artifacts (*i.e.,* software, raw data, and detailed documentation) are available through the following sources:

https://zenodo.org/record/5635607
https://github.com/ilabrg/artifacts-conext21-ble

If you want to reproduce our results, we recommend to follow the `README.md` in this repository.

The archived version on https://zenodo.org/ should provide public access for more than 10 years. Providing both the GitHub repository and the Zenodo archive in the paper allows us to maintain the archive and to have a citable version that reflects the state when the artifacts were granted.

A.2 IP over BLE Platform

The open source IP over BLE platform presented in our paper is published as part of the RIOT [3, 8] and NimBLE [1] open source projects. The full platform is part of the current releases of each project, `RIOT 2021.07` and `NimBLE 1.4.0`, respectively.

Additional bug fixes regarding the IP over BLE platform have been introduced after these releases. Because of this, the current experimentation framework is based on the current development branches, namely `0386aea` for RIOT and `7d3f3cc` for NimBLE.

The GNRC adaption code is placed in the RIOT code tree under `pkg/nimble/netif`. The source code of the *statconn* connection manager implementation is located in the RIOT source tree under `pkg/nimble/statconn`.

A.3 Experimentation Framework

All experiments described and analyzed in our paper were conducted using a custom experimentation framework that consists of a YML-based description format as well as tooling for running and analyzing experiments in an automated fashion.

Each experiment is fully described in form of a static experiment description file. These descriptions contain the hardware nodes used, a mapping of firmware configuration onto these nodes, the runtime node configuration as well as the command sequence to be run during the experiment. This static experiment description ensures repeatability.

For each experiment the framework produces three types of artifacts: (*i*) the static experiment description, (*ii*) a single log file representing the experiment's raw results data, and (*iii*) the intermediate results in form of plots and the preprocessed data those plots are created from.

The `README.md` in the main folder of the repository contains all information and step-by-step instructions to reproduce our experiments.

A.4 Experimentation Results

The raw data for each analysis performed in the paper is available via Zenodo instance at https://zenodo.org/record/5635607. The `logs.tar.gz` archive contains the raw log files of each experiment run. For convenience we also included the intermediate results as created by our analysis tooling in the `plots.tar.gz` archive. These intermediate results can also be created by running the analysis tool included in our experimentation framework repository as described in § A.3.

B SUMMARY OF EXPERIMENT RESULTS

Figure 15 illustrates the aggregated results of running 60 different experiment configurations, each for 5×1 hour.

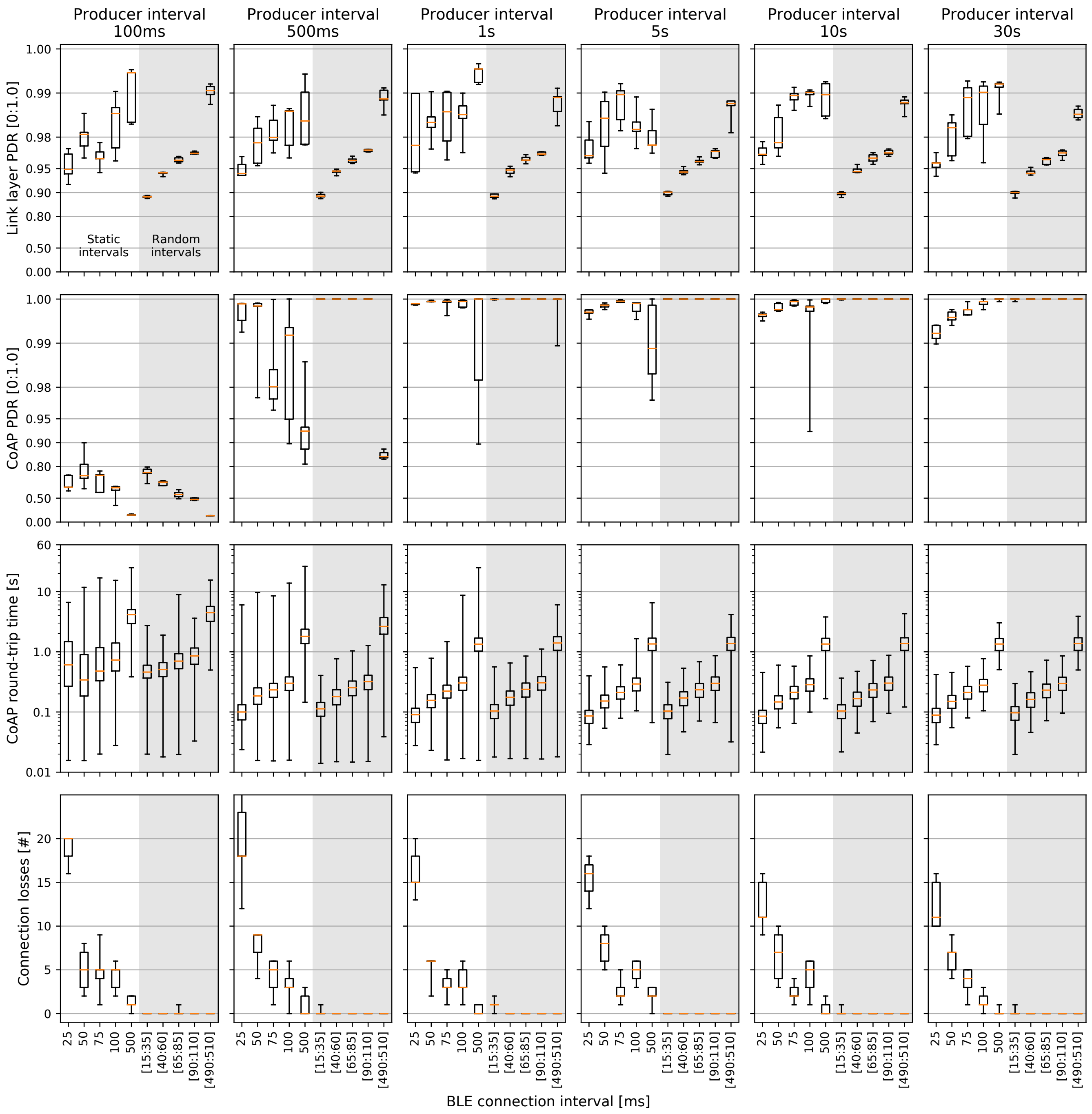


Figure 15: Aggregated results for 60 different experiment configurations. Each configuration ran for 5×1h.