Ensuring End-to-End Dependability Requirements in Cloud-based Bluetooth Low Energy Applications

Michael Spörk, Markus Schuß, Carlo Alberto Boano, and Kay Römer
Institute of Technical Informatics,
Graz University of Technology, Austria
{michael.spoerk, markus.schuss, cboano, roemer}@tugraz.at

Abstract

Bluetooth Low Energy (BLE) is increasingly used for time-critical IoT applications, where BLE-based smart objects need to exchange data with a remote server within stringent end-to-end latency and reliability bounds. While existing research has investigated how to timely send packets between pairs of BLE devices, it is still unclear how a BLE device can sustain time-critical end-to-end communication with a remote server, for example, hosted in the cloud.

In this paper, we tackle this problem and show how BLE devices can autonomously measure and cope with end-to-end network delays and loss along the path to the remote server. To this end, we first devise an analytical model of the communication between a BLE end-node and the cloud. We then leverage this model to dynamically adapt the communication parameters of the BLE device and sustain the desired end-to-end dependability requirements while minimizing the energy expenditure. Specifically, we design and implement two adaptation strategies on the popular `nRF52` platform, and experimentally show that they both allow to sustain a given end-to-end reliability and a given end-to-end latency for data transmissions from/to the BLE node, while limiting the node's power consumption.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*; C.4.6 [**Performance of Systems**]: Reliability, availability, and serviceability.

General Terms

Design, Measurement, Performance, Protocols, Reliability.

Keywords

Adaptive Protocols, Bluetooth Low Energy, End-to-end Latency, End-to-end Reliability, Internet of Things.

1 Introduction

Bluetooth Low Energy (BLE) is one of the most popular low-power wireless technologies in the IoT landscape, due to its wide adoption in consumer electronics devices connected to the Internet such as smart-phones, tablets, and laptops [5]. The resulting ease with which resource-constrained BLE-based smart objects can interface to such devices and connect to the Internet is a key enabler for the development of attractive IoT systems based on BLE technology. Many of these IoT systems operate in time-critical domains: examples are smart grids [8], smart cities [15], and smart healthcare [16, 22] applications, where resource-constrained BLE nodes often need to exchange data with a cloud server within strict end-to-end transmission reliability and latency bounds; all of this while operating on batteries for months or years.

For example, in remote ECG monitoring systems [22] a BLE node measures cardiac signals of a patient and sends these measurements to a router in the same BLE subnet, as shown in Fig. 1. The router forwards these data packets containing ECG measurements via the external network path, *i.e.*, the Internet, to a cloud server to be processed or stored.

The data exchange between a BLE node and a remote server is often subject to dynamic changes in the transmission delay and loss across the entire network path. In the BLE subnet, packets may be significantly delayed due to persistent or transient link-layer problems [35, 36]: this is often due to multipath fading effects and due to the presence of RF interference from surrounding devices (*e.g.*, co-located Wi-Fi access points). On the external network path, packets may be lost due to buffer congestion and CPU-intensive tasks (*e.g.*, routing table updates) on backbone routers. Data transmissions can also be significantly delayed due to routing changes in the Internet backbone or due to link quality fluctuations in the case of cellular connections [11, 13, 40].

To sustain end-to-end dependability requirements on communication, BLE nodes need to capture and adapt to all these changes at runtime. This requires a proper knowledge about the delay and loss across the entire network path and appropriate models: only this way, a BLE node can adapt its parameters at runtime and select the right trade-off between communication timeliness, reliability, and power efficiency.

Capturing network delay and loss. Although several works have investigated how to capture and adapt to changes in delay and loss across the Internet [1, 2, 11, 13, 30], the focus has always been on devices that have a high-bandwidth

Internet connectivity and that are not constrained in their processing capabilities or power supply.

In contrast to the studies above, a large body of research has investigated how to sustain latency and reliability bounds in constrained and low-power wireless networks, based for example on IEEE 802.15.4 [14, 17, 42] or BLE [35, 36]. Building upon these works, nodes can cope with link-layer problems in the local subnet and sustain a timely and reliable communication while minimizing their power consumption. The problem, however, is that none of these studies investigates how to sustain *end-to-end* requirements for communications that go beyond the local subnet, *e.g.*, when a node exchanges data with a cloud server over the Internet as illustrated in Fig. 1. Therefore, how BLE nodes can effectively capture delays and loss across the *entire* network path remains an open question. Answering the latter is fundamental to allow the selection of suitable BLE connection parameters at runtime in order to sustain end-to-end dependability bounds while limiting a node's energy expenditure.

Sustaining end-to-end requirements on a budget. Another open issue is that existing analytical models of BLE's communication performance as a function of the available connection parameters are unsuitable to design adaptive strategies at runtime. Several models, indeed, require low-level channel information that is not available on off-the-shelf BLE devices [23, 29, 34]. Only the model presented by Spörk et al. [36] is able to use standard BLE information to monitor the timeliness of BLE communications. Unfortunately, however, this model is limited to transmissions within the local BLE subnet. Furthermore, all the aforementioned models suffer from two additional limitations: (i) they focus only on transmitting nodes and neglect how a node can sustain dependability bounds when acting as a receiver; (ii) they solely adapt the BLE connection interval, *i.e.*, they neglect the *BLE slave latency*, a connection parameter that can greatly influence the behaviour of a BLE device [5].

There is hence a need for new end-to-end models that keep the entire network path in the picture and also include parameters such as the BLE slave latency. This way, one has the means to properly adapt BLE communication parameters at runtime to sustain given end-to-end latency and reliability bounds while preserving power efficiency.

Contributions. In this work, we tackle all these challenges and show how BLE nodes can sustain time-critical communication with a remote server on the cloud in both directions.

First, we devise a new end-to-end BLE model, incorporating a local model by Spörk et al. [36], that captures the additional latency introduced by the network path outside of the BLE subnet. In doing so, we also let the model capture the impact of the BLE slave latency on communication latency as well as the interplay between the connection interval and slave latency on the timeliness of packet receptions.

Next, we show how a BLE node can accurately and efficiently estimate the communication latency across the entire network path by using short and infrequent probing bursts. Our estimation approach fully adheres to the end-to-end principle of the Internet, *i.e.*, it requires no changes to any of the devices routing data on the network path.

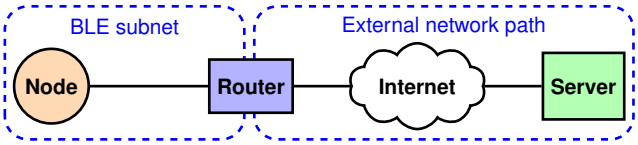


Figure 1. Network topology when exchanging data between a BLE node and a cloud server over the Internet.

We further leverage the proposed model and latency estimation scheme to let a node adapt its BLE communication parameters at runtime and sustain a given end-to-end transmission reliability as well as a given end-to-end latency when communicating with a cloud server. Specifically, we propose adaptation strategies for two different use cases: (i) the router being constrained in its BLE radio duty cycle and (ii) the router not having radio duty cycle constraints.

We implement our adaptation approaches on the popular Nordic Semiconductor `nRF52` platform [26] using Zephyr OS [37] and experimentally show that both approaches effectively find suitable BLE parameters at runtime that minimize delayed packets and power consumption, outperforming other static or adaptive node configurations. In our implementation we make use of IPv6-over-BLE communication as specified by the RFC 7668 [25] to exchange data between a BLE node and cloud as illustrated in Fig. 1. Nevertheless, our model, estimation, and adaptation approaches are independent of the used network layer on top of the BLE connection and can directly be used for IoT applications using BLE communication based on GATT.

After introducing the necessary background information and providing a real-world measurement of the delays and packet loss in typical cloud-based BLE applications in Sect. 2, this paper makes the following contributions:

- We devise a new end-to-end BLE model that captures the *end-to-end* latency across the whole network path and embeds the role of the BLE slave latency (Sect. 3).
- We show how a BLE node can autonomously estimate the latency across the entire network path while complying to the end-to-end principle of IP (Sect. 4).
- We propose two different adaptation strategies that a node can use at runtime to sustain end-to-end requirements under different constraints (Sect. 5).
- We implement both approaches on the `nRF52` platform using Zephyr (Sect. 6), and we experimentally evaluate their performance in detail (Sect. 7).

After describing related research in Sect. 8, we conclude our paper and list future work in Sect. 9.

2 Investigating cloud-based BLE applications

We start our work by experimentally investigating the end-to-end latency and end-to-end reliability of a BLE node exchanging data with a cloud server on the Internet.

Fig. 1 shows the used network topology, where an IPv6-over-BLE node device is connected to an IPv6-over-BLE router providing Internet access. To exchange IPv6 packets, node and router establish an IPv6-over-BLE connection according to the RFC 7668 [25]. Once this connection is established, the router forwards the packets to the nodes within the IPv6-over-BLE subnet or to IP devices on the Internet, such as our server. This common network topology is used

Table 1. Measured latencies between node and server over 7 days in our testbed for an IPv6 packet length of 128 bytes.
The table shows the median (50%), 95 percentile (95%), and maximum experienced latency (100%) for different configurations.

conn.	CI [ms]	SL	t_{TX} [ms]			$t_{TX_{BLE}}$ [ms]			t_{RX} [ms]			$t_{RX_{BLE}}$ [ms]		
			50%	95%	100%	50%	95%	100%	50%	95%	100%	50%	95%	100%
wired	50	0	40	88	328	31	79	319	42	92	293	32	82	284
wired	50	4	39	86	732	30	75	258	41	292	1291	32	282	1281
cellular	50	0	73	1157	3026	28	69	237	67	601	2411	34	64	176
cellular	50	4	73	739	4038	27	62	283	268	531	1031	191	257	499

by popular IoT protocols, such as CoAP, MQTT, or MQTT-SN. One major constraint for low-power IoT applications, however, is that they usually do not use the heavyweight TCP transport layer to reliably send data. Instead, these applications use a UDP transport layer that allows low-power consumption, at the cost of packet loss on the network path [4].

To exchange data, node and router set up a BLE connection, where communication happens during connection events. During these connection events, node and router bidirectionally exchange data until both devices have no more data to send or until the maximum connection event length (t_{CE}) has been reached. The *connection interval* (CI) defines the time between two consecutive connection events. Even if no data needs to be transmitted, node and router exchange short mandatory keep-alive link-layer packets in every connection event, to keep the BLE connection alive. As these keep-alive messages cause unnecessary power consumption on the BLE devices, the BLE specification foresees the BLE *slave latency* (SL) parameter, which allows the node to skip up to SL connection events.

BLE connections make use of adaptive frequency hopping and autonomous packet retransmissions to ensure that every packet that is scheduled to be transmitted over BLE will be successfully received. These mechanisms, indeed, lead to a reliability of 100% within the BLE subnet, as shown in [36], although some packets may be delayed due to link-layer effects, such as external radio interference.

The timeliness and reliability of IPv6 packets across the external network path depend on the employed technology (*e.g.*, Ethernet or 4G) and cannot be controlled by the node. To sustain an upper bound on the end-to-end latency between the node and server, the node can only dynamically adapt its BLE connection interval and slave latency

To investigate the real-world behavior of BLE-based IoT applications, we perform a first experimental study.

Experimental setup. To measure latency of IPv6 packets across the whole network topology (Fig. 1), we perform our measurements in a wireless testbed powered by D-Cube nodes [31] located in a vacant laboratory.

We use four Nordic Semiconductor `nRF52840 DK` devices, each running an IPv6-over-BLE node application built on the Zephyr OS that transmits a UDP packet to the server once every second. The server is an Amazon Web Service (AWS) instance located in the AWS center in Frankfurt, Germany, and runs a Python server echoing every UDP packet received by a node. Both, packet to and from the server, have an IPv6 packet length of 128 bytes and carry a unique sequence number to match request to response. We use a Raspberry Pi 4 (Pi4) as IPv6-over-BLE router running Raspbian OS, which provides Internet access to the nodes. To make use of the features of BLE version 5, we use another `nRF52840 DK` and program it with Zephyr's BLE HCI-USB firmware, which allows us to use this nRF device as BLE radio on our router. For all of our experiments, the BLE connection uses the 2M PHY mode of BLE, all available 37 BLE data channels and does not adaptively blacklist channels, which is the default behavior of the HCI-USB firmware.

Every time a BLE node issues a UDP packet, it triggers a GPIO event that is captured by a D-Cube device (one D-Cube device per nRF52 node). We further log all IPv6 traffic on the router to measure the communication latency of IPv6 packets in the BLE subnet. Finally, we also log the timestamp and source address of UDP packets received by the server. All devices in our experiment are synchronized to the same NTP server and, therefore, share the same notion of time (the average clock offset between nodes and server is -43 $\pm$ 134μs). This allows us to calculate the end-to-end latency (t_{TX}) and the latency within the BLE subnet ($t_{TX_{BLE}}$) for every packet sent from node to server. Similarly, we calculate the end-to-end latency (t_{RX}) and the latency within the BLE subnet ($t_{RX_{BLE}}$) for packets sent from server to node.

To test the impact of different technologies providing Internet access to our BLE subnet, our router can make use of a wired Ethernet or a cellular 4G connection.

Preliminary results. Tab. 1 shows the distribution of the measured latencies for different BLE connection parameters and two different Internet connections measured over 7 days.

When comparing a wired Internet connection to a cellular connection, we can clearly see that the used Internet connectivity significantly affects the overall communication latency in both directions. In our experiments, the maximum latency for transmitting and receiving IPv6 packets with a cellular connection is almost 10 times higher than a wired connection. Also the median latency in both directions is significantly higher for a cellular than for a wired connection. This shows that a node cannot simply assume a fixed delay across the external network path to sustain latency bounds.

When investigating different BLE connection parameters, the data in Tab. 1 shows that the SL parameter does not significantly affect the latency in the BLE subnet of packets sent by the node ($t_{TX_{BLE}}$). As expected, the SL, however, significantly affects the latency of packets received by the node ($t_{RX_{BLE}}$). The reason for that is the node skipping up to SL connection events when it has no data to transmit, which means that the router may need to wait up to SL connection events for the node to wake up and receive packets[1].

During our experiment, the wired Internet connection experienced an average end-to-end transmission reliability of

[1]The actual slave latency behavior depends on the node's BLE link-layer implementation and may differ between different BLE chip vendors.

99.35%, with at least 97% of any 100 subsequent transmissions successfully sent. The cellular connection sustained an average end-to-end transmission reliability of 97.65% and a minimum reliability of 92% for any 100 transmissions. Similar to [36], no packet was lost within the BLE subnet.

Based on the experimental data, we can see that the end-to-end latency and reliability of packets sent by a BLE node depend on (i) the used BLE connection parameters and (ii) the behavior of the external network path. To sustain a given end-to-end reliability and a given end-to-end latency while limiting unnecessary power consumption, a node needs to dynamically adapt its BLE connection parameters to changes in the overall network path. To find the most suitable BLE connection parameters, however, new BLE models are required that, in addition to capturing the effects of both the connection interval and slave latency, need to cope with the behavior of the entire network path.

One approach to sustain given end-to-end latency bounds would be to use application-level round-trip time (RTT) measurements on the node to adapt the BLE connection parameters. The main problem, however, is that existing application traffic cannot be used to accurately estimate the one-way communication latencies t_{TX} or t_{RX}. During normal operation, the BLE node uses a BLE slave latency greater than 0, which allows the node to limit power consumption while sustaining given latency bounds. A value $SL > 0$ leads to packet receptions being unpredictably delayed, which significantly affects the accuracy of individual t_{TX} and t_{RX} estimates.

In this work, we follow another approach, where we first devise new end-to-end models that captures the delay across the entire network path (Sect. 3) and show how a BLE node can use infrequent probing bursts to accurately estimate the network latency across the entire network path in Sect. 4. In Sect. 5, we combine our model and network latency estimation to sustain given end-to-end dependability requirements.

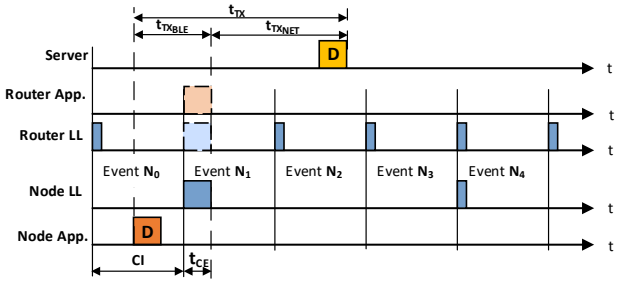
3 Modeling BLE communication

In this section, we devise a new end-to-end model, which incorporates the local BLE model from [36], to fit the use case shown in Fig. 1. Towards this goal, we do not only extend the timeliness model to account for delays across the Internet, but also investigate how the BLE connection interval and slave latency affect timeliness when the node transmits (Sect. 3.1) and receives data (Sect. 3.2).

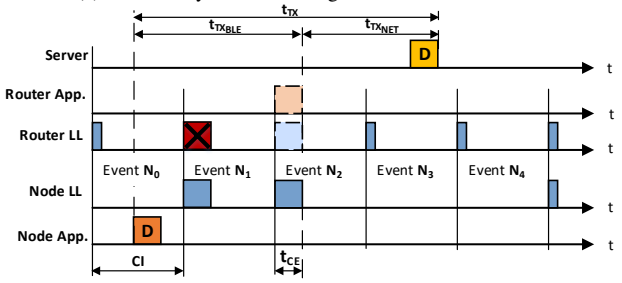
3.1 Transmitting IPv6-over-BLE data

First, we model the end-to-end latency (t_{TX}) for a node transmitting a data packet to a cloud server.

Fig. 2 shows two exemplary transmissions from the BLE node via a router to the server. In both examples, the application on the BLE node (Node App.) issues a data transmission (D) between connection event N_0 and N_1. When no link-layer errors occur (shown by Fig. 2(a)), the BLE link-layer (LL) of the node and router successfully exchange the data in connection event N_1. In case there are link-layer errors, *i.e.*, due to external Wi-Fi interference (shown in Fig. 2(b) in connection event N_1), the link-layer of the BLE node retransmits the packet until successfully received by the router. After successfully receiving the packet, the router forwards the packet via the Internet to the server, which takes $t_{TX_{NET}}$.



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



(b) One link-layer error during BLE data transmission.

Figure 2. Timing for a BLE node transmitting a data packet (D) to a cloud server over the Internet.

In these examples, the packet length (D_{TX}) is smaller than the maximum packet length that can be sent during a single connection event (F_{TX}) and, therefore, only one successful connection event is used to transmit the packet. If $D_{TX} > F_{TX}$, the packet is split into multiple data fragments, where every fragment requires its own successful connection event.

As Fig. 2 shows, the end-to-end latency (t_{TX}) for sending a data packet from node to server consists of:

$$t_{TX} = t_{TX_{BLE}} + t_{TX_{NET}}, \tag{1}$$

where $t_{TX_{BLE}}$ is the transmission latency of the packet between node and router and $t_{TX_{NET}}$ is the transmission latency of the packet from router to server. The $t_{TX_{NET}}$ delay depends on the used Internet connection and will be estimated in Sect. 4. We model the $t_{TX_{BLE}}$ delay as shown in [36] as:

$$t_{TX_{BLE}} = \left(\sum_{f=1}^{\lceil D_{TX}/F_{TX} \rceil} n_{CE_f} \cdot CI \right) + t_{CE}, \tag{2}$$

where D_{TX} is the length of the sent packet and F_{TX} is the maximum packet length that can be sent from node to router within a single BLE connection event. n_{CE_f} is the number of connection events necessary to successfully transmit an individual packet or fragment, capturing any link-layer retransmissions. CI represents the BLE connection interval, and t_{CE} is the maximum duration of a single connection event.

When combining Eq. 1 with Eq. 2, we can calculate the end-to-end transmission latency t_{TX} as:

$$t_{TX} = \left(\sum_{f=1}^{\lceil D_{TX}/F_{TX} \rceil} n_{CE_f} \cdot CI \right) + t_{CE} + t_{TX_{NET}}. \tag{3}$$

To calculate the upper bound on the end-to-end transmission latency ($t_{TX_{MAX}}$), we assume that every individual packet fragment is sent with $n_{CE_f} = n_{CE_{MAX}}$, *i.e.*, every link-layer transmission experiences the same link-layer error proba-

bility. Furthermore, we assume that the packet experiences the maximum current delay across the external network path ($t_{TX_{NET}} = t_{NET_{MAX}}$), which we discuss in Sect. 4. Applying these assumptions to Eq. 3, we derive that:

$$t_{TX_{MAX}} \geq n_{CE_{MAX}} \cdot \left\lceil \frac{D_{TX}}{F_{TX}} \right\rceil \cdot CI + t_{CE} + t_{NET_{MAX}}. \quad (4)$$

Note that the slave latency (SL) does not impact t_{TX} and $t_{TX_{MAX}}$. Indeed, the slave latency allows a BLE node to skip connection events if no data has to be transmitted. If the node, however, has data to transmit (as it is the case here), the node simply does not make use of the slave latency.

3.2 Receiving IPv6-over-BLE data

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

Fig. 3 shows two scenarios involving a packet sent from the server to the BLE node. In both examples, the server issues a data transmission between connection event N_0 and N_1 and the router application (Router App.) successfully receives the packet between event N_1 and N_2, which takes $t_{RX_{NET}}$. Next, the router forwards the packet to the node and therefore issues a transmission on its link layer (Router LL). In connection event N_2, the router LL tries to send the packet over BLE to the node, but the node makes use of its configured slave latency ($SL = 2$) and does not wake up during event N_2 to receive any data. The router LL retransmits the packet to the node until it is successfully received. In the example in Fig. 3(a), this happens during connection event N_3. In Fig. 3(b), router and node wake up during connection event N_3, but due to a link-layer problem (*e.g.*, external radio interference), the packet is not successfully received by the node. As the node has not received a valid BLE link-layer packet from the router, it follows the behavior required by the BLE specification [5] and wakes up during every subsequent connection event until a valid BLE packet from the router is received. In our example (Fig. 3(b)) this happens during event N_4, after which the packet from the server is successfully received by the application on the BLE node.

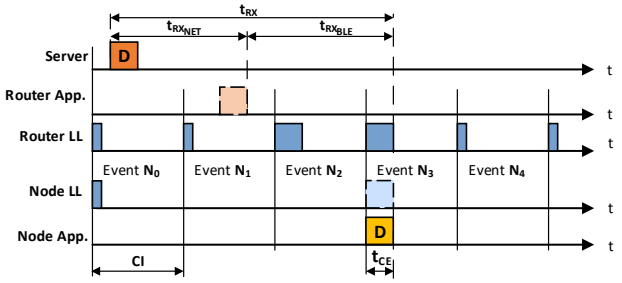
In both examples, the packet length (D_{RX}) is smaller than the maximum packet length that can be received during a single connection event (F_{RX}). If $D_{RX} > F_{RX}$, multiple successful connection events are necessary to send the packet from router to node. In such a case, the node is informed about more data on the router via the MD-field in the BLE link-layer header and does not make use of its slave latency until all data from the router is successfully received.

As shown in Fig. 3, the end-to-end latency (t_{RX}) for a node receiving a packet from a remote server consists of:

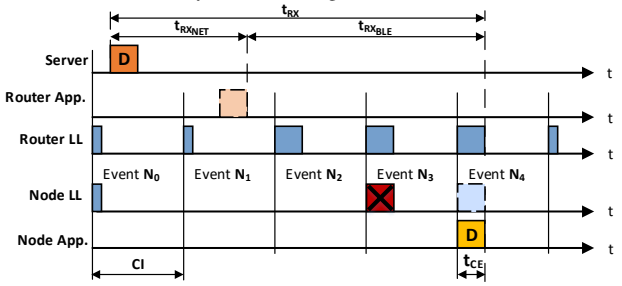
$$t_{RX} = t_{RX_{NET}} + t_{RX_{BLE}}, \quad (5)$$

where $t_{RX_{NET}}$ is the latency of the packet between server and router and $t_{RX_{BLE}}$ is the latency of the packet from router to node. Similar to Sect. 3.1, $t_{RX_{NET}}$ is dependent on the quality of the Internet connection and will be studied in Sect. 4. $t_{RX_{BLE}}$ can be modeled as:

$$t_{RX_{BLE}} = \left(\sum_{f=1}^{\lceil D_{RX}/F_{RX} \rceil} n_{CE_f} \cdot CI \right) + t_{CE} + t_{SL}, \quad (6)$$



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



(b) One link-layer error during BLE data transmission.

Figure 3. Timing for a BLE node receiving a data packet (D) from a cloud server over the Internet.

where D_{RX} is the length of the data packet and F_{RX} is the maximum packet length that can be received by the node within a single BLE connection event. n_{CE_f} is the number of connection events necessary to successfully transmit an individual packet or fragment (capturing any necessary link-layer retransmissions), CI is the BLE connection interval, and t_{CE} is the maximum duration of a single connection event. t_{SL} measures the additional time the router needs to wait until the node is waking up after skipping up to SL connection events, which has a maximum value of:

$$t_{SL_{MAX}} = CI \cdot SL. \quad (7)$$

When combining Eq. 5 with Eq. 6, we can calculate the overall end-to-end reception latency t_{RX} as:

$$t_{RX} = t_{RX_{NET}} + \left(\sum_{f=1}^{\lceil D_{RX}/F_{RX} \rceil} n_{CE_f} \cdot CI \right) + t_{CE} + t_{SL}. \quad (8)$$

Similar to Sect. 3.1, we adapt Eq. 8 to calculate the upper bound on the end-to-end reception latency ($t_{RX_{MAX}}$) by using $n_{CE_f} = n_{CE_{MAX}}$, $t_{RX_{NET}} = t_{NET_{MAX}}$, and Eq. 7 to obtain:

$$t_{RX_{MAX}} \geq (n_{CE_{MAX}} \cdot \left\lceil \frac{D_{RX}}{F_{RX}} \right\rceil + SL) \cdot CI + t_{CE} + t_{NET_{MAX}}. \quad (9)$$

Compared to Sect. 3.1, we can see that the slave latency (SL) impacts t_{SL} and hence the time it takes for a node to receive a packet from the server. Using a value of $SL > 0$, the BLE node skips connection events to minimize power consumption when it has no data to transmit. This, however, means that the router may need to wait up to SL connection events until the node wakes up to receive data.

The models presented in Sect. 3.1 and 3.2 allow us to calculate the latency for transmitting and receiving data packets on the BLE node. One critical aspect missing, however, is the delay that is caused by the network path outside the BLE subnet (t_{NET}), which we investigate next.

4 Estimating Network Latency

In this section, we discuss how a BLE node can estimate t_{NET} in a way that is fully compliant to the end-to-end principle of IP. Following this principle, our t_{NET} estimation approach allows nodes to estimate t_{NET} without requiring any changes to devices on the network path, such as the router. Therefore, our estimation approach can easily be used on any node and works with any IPv6-over-BLE router that adheres to the specification in [25]. Approaches violating the IP end-to-end principle, *i.e.*, requiring changes to the routers in the network, lead to huge setup and deployment costs and are therefore seldomly used in practice [41].

Our t_{NET} estimation is fully technology-agnostic and estimates t_{NET} independently of the applied technology to connect to the Internet. Furthermore, our estimation approach can also be used to estimate t_{NET} in applications of any network scope, *e.g.*, applications spanning only a local Intranet.

While we use our estimation approach to estimate the maximum latency across the entire network path, our approach may also be used to calculate the average latency in order to synchronize a node's clock, *e.g.*, via NTP.

The main problem in estimating network latency in our use case, however, is that existing application traffic cannot be used to accurately estimate t_{NET}. During normal operation, the BLE node uses a long BLE connection interval (CI) and a BLE slave latency (SL) greater than 0 that allow the node to sustain end-to-end latency bounds while limiting power consumption, but cause two problems while estimating t_{NET}. As described in Sect. 3.2, $SL > 0$ leads to packet receptions being unpredictably delayed, affecting the accuracy of individual t_{NET} estimates. Furthermore, a large CI value leads to a coarse sampling resolution of the application-level round trip time and therefore leads to coarse t_{NET} estimates.

4.1 Probing network latency

To estimate t_{NET} on a node, we periodically perform short probing bursts, where we exchange short probe and corresponding acknowledgment packets between node and server.

At the start of every probing burst, the node updates its BLE connection parameter to the smallest possible CI and $SL = 0$[2]. With the smallest possible CI, the node is able to sample t_{NET} with the lowest possible sampling resolution. By using $SL = 0$ during probing, we eliminate any unpredictable delay caused by SL during reception (as captured by t_{SL} in Eq. 8) that would impact our estimation accuracy.

After the BLE connection parameters for probing are successfully set, the node transmits a short probing packet to the server, to which the server responds with a short acknowledgment packet. The node issues a new probing packet as soon as the previous probing packet has been acknowledged by the server. These probe packets can be ordinary application data packets or also distinct IPv6-based packets solely used for probing, such as ICMPv6 echo requests and responses. To get the most accurate t_{NET} estimations, however, probe and acknowledgment packets need to fit within a single connection event, *i.e.*, $D_{TX} \leq F_{TX}$ and $D_{RX} \leq F_{RX}$. When node and server have exchanged L_{Probe} probe/acknowledgment packets, the node reverts the BLE connection parameters to the settings used before probing and continues with its normal behavior.

For every exchanged probe/acknowledgment pair, the BLE node measures the round trip time (t_{RTT}) and the BLE transmission time ($t_{TX_{BLE}}$). t_{RTT} is measured as the time between the node application issuing the probe transmission and the node successfully receiving the corresponding acknowledgment. To measure $t_{TX_{BLE}}$, the BLE node monitors the communication on the standardized BLE Host Controller Interface (HCI) of the BLE controller, as shown by [36]. By measuring the time between the node application issuing the BLE data transmission and the BLE controller actually freeing the corresponding data buffer, the node application is able to measure $t_{TX_{BLE}}$ in a standard-compliant way.

The measured time t_{RTT} can be modeled as:

$$t_{RTT} = t_{Probe} + t_{ACK}, \tag{10}$$

where t_{Probe} is the end-to-end transmission latency of a probe packet from node to server and t_{ACK} is the end-to-end transmission latency of the acknowledgment from server to node. By using Eq. 1 and Eq. 5 for modeling t_{Probe} and t_{ACK}, respectively, we get:

$$t_{RTT} = t_{TX_{BLE}} + t_{TX_{NET}} + t_{RX_{NET}} + t_{RX_{BLE}}. \tag{11}$$

One simplification for our estimation approach is that we assume the delay across the Internet is symmetric and call this delay t_{NET}, *i.e.*, $t_{NET} = t_{TX_{NET}} = t_{RX_{NET}}$, which gives us:

$$t_{RTT} = t_{TX_{BLE}} + 2 \cdot t_{NET} + t_{RX_{BLE}}. \tag{12}$$

The network delay t_{NET} can thereby be calculated as:

$$t_{NET} = \frac{t_{RTT} - t_{TX_{BLE}} - t_{RX_{BLE}}}{2}. \tag{13}$$

Although we assume a symmetric Internet delay, our t_{NET} estimation approach can also accurately capture the actual one-way network latency of asymmetric Internet connections, such as 4G communication, as we show in Sect. 4.3.

While the node can measure t_{RTT} and $t_{TX_{BLE}}$ for every probe, $t_{RX_{BLE}}$ cannot be measured by monitoring HCI communication, but needs to be estimated by using the information available on the BLE node. For our estimation approach, we assume that $t_{RX_{BLE}} = t_{TX_{BLE}}$, as both probing and acknowledgment packets fit within one single connection event and $t_{TX_{BLE}}$ provides us with the most recent link-layer information. Furthermore, we assume that receiving a packet from the router takes at least CI, which is the smallest time unit we can measure with our probing approach, resulting in:

$$t_{RX_{BLE}} = MAX(t_{TX_{BLE}}, CI). \tag{14}$$

4.2 Estimating maximum network latency

After measuring the network latency (t_{NET}) of individual packet exchanges during probing, we use these measurements to estimate the maximum network latency ($t_{NET_{MAX}}$) of future packet exchanges. With $t_{NET_{MAX}}$ estimations and our proposed model in Sect. 3, we are able to sustain end-to-end communication requirements, as we show in Sect. 5.

Fortunately, estimating upper bounds on transmissions on the Internet is a well-researched topic [11, 19, 20]. While

[2] This is done by using the standardized BLE connection parameter negotiation process defined by the BLE specification [5].

most recent research uses sophisticated statistical analysis [11, 30], only a subset of these studies propose solutions that are suitable for devices that are constrained in their processing capabilities and power supply, such as BLE nodes. For our estimation, we adapt the efficient and well-established round-trip time estimation approach of TCP as specified by Jacobson [18] and standardized in [28].

Instead of using an exponentially weighted moving average as proposed in [28], we store all individual t_{NET} measurements during a probe burst and calculate their average ($t_{NET_{AVG}}$) and variance ($t_{NET_{VAR}}$) at the end of every burst. Using the average and variance, we can calculate $t_{NET_{MAX}}$ as:

$$t_{NET_{MAX}} = t_{NET_{AVG}} + K \cdot t_{NET_{VAR}}, \quad (15)$$

where we use a fixed $K = 4$ as specified in [28].

4.3 Choosing a probe burst length

We study next the most suitable probe burst length (L_{Probe}) that provides an accurate $t_{NET_{MAX}}$ estimation while limiting unnecessary energy consumption on the BLE node, caused by probing. We do this empirically by letting an `nrf52`-based BLE node exchange probe/acknowledgment packets once every second with a server in four different environments for eight hours per environments.

Experimental environments. For our measurements, we use the following four different experimental environments:

Wired & No Interf. In this scenario, we use a wired Internet connection on the router with very low variability in combination with no Wi-Fi interference in the BLE subnet.

Wired & Wi-Fi Interf. This scenario uses the same wired Internet connection as above. To generate link-layer problems and, therefore, delays in the BLE subnet, we introduce continuous Wi-Fi interference on two different Wi-Fi channels (sending 1500 bytes of Wi-Fi data every 10 ms with a TX power of 50 mW) using two different co-located Rasperry Pi 3 devices in our testbed running Jamlab-NG [32].

Cellular & No Interf. In this scenario, we use a 4G connection to connect our router to the Internet. Compared to the wired Internet connection, the 4G connection experiences longer and more variable t_{NET} values, as we show in Sect. 2. This scenario does not introduce any Wi-Fi interference.

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

To evaluate different probing settings, we measure the one-way t_{NET} delay of every sent probe using our NTP-synchronized router and cloud server, as described in Sect. 2. Additionally, the node estimates a t_{NET} value, as described in Sect. 4.1, for every exchanged probe/acknowledgment pair.

We step through the recorded t_{NET} estimates of the node and use Eq. 15 to calculate a new $t_{NET_{MAX}}$ after every t_{NET} estimate. For every new $t_{NET_{MAX}}$ value, we check how many of the future t_{NET} measurements, *i.e.*, the actual one-way t_{NET} measurements, exceed the $t_{NET_{MAX}}$ estimate and are therefore underestimated. For our evaluation we use a very conservative prediction window, *i.e.*, the probing interval (I_{Probe}), of

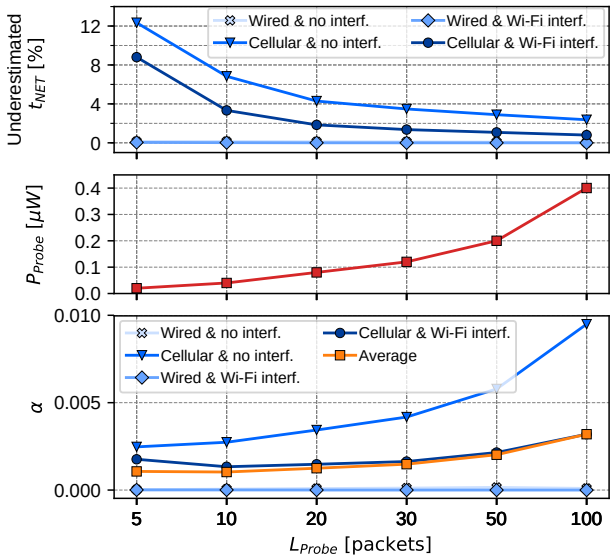


Figure 4. Percentage of underestimated t_{NET} values (top), power consumption P_{Probe} (middle), and cost α (bottom) for different L_{Probe} values in four different environments.

1000 seconds. With this configuration, a node initiates a new probing burst every 1000 seconds to estimate t_{NET}.

Fig. 4 shows the performance of different L_{Probe} values in four different environments. The top of Fig. 4 shows the average number of underestimated future t_{NET} values. For example, when using a cellular Internet connection and no interference, $L_{Probe} = 10$ results in approximately 7% of all future t_{NET} values being underestimated. P_{Probe} shows the calculated average power consumption of a node for probing with different L_{Probe} and a $I_{Probe} = 1000\,s$. The bottom of Fig. 4 shows the cost α for every L_{Probe} as the number of underestimated t_{NET} predictions multiplied by P_{Probe}.

Fig. 4 shows that a short $L_{Probe} = 10$ has the lowest cost α in all four environments, *i.e.*, it provides the most suitable t_{NET} estimation of future packet exchanges while limiting unnecessary energy consumption on the BLE node. For wired Internet connectivity, even $L_{Probe} = 5$ would provide an accurate $t_{NET_{MAX}}$. For cellular Internet connectivity, a larger L_{Probe} would slightly improve the $t_{NET_{MAX}}$ estimation, at the cost of a higher average power draw on the BLE node.

Next, we investigate the performance of our t_{NET} estimation for different I_{Probe} values and $L_{Probe} = 10$. We reuse the recorded data to calculate the percentage of underestimated t_{NET} values, the average power consumption for probing (P_{Probe}), and the cost α of different probing intervals.

Fig. 5 shows that $I_{Probe} = 1000\,s$ performs best in all four experimental environments, as it has the lowest cost α. In our setting, a smaller I_{Probe} does not significantly reduce the percentage of underestimated t_{NET} values. Using a smaller I_{Probe}, however, significantly increases the power consumed by the node for probing t_{NET}.

IoT applications that experience vast and sudden changes in delay across the Internet, *e.g.*, a 4G-based mobile router experiences link quality degradation and needs to change to a 3G backhaul, may use a shorter probe interval (I_{Probe}) at the cost of a higher power consumption of the BLE node.

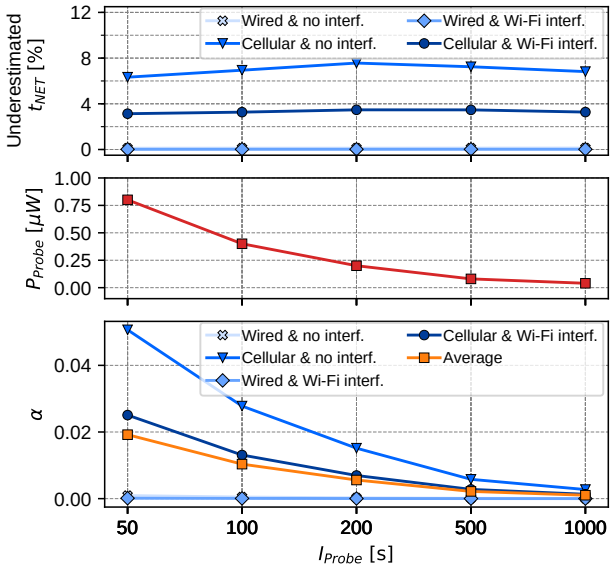


Figure 5. Percentage of underestimated t_{NET} values (top), power consumption P_{Probe} (middle), and cost α (bottom) for different I_{Probe} values in four different environments.

5 Sustaining End-to-End Requirements

In this section, we discuss how a BLE node can use our model (Sect. 3) and t_{NET} estimations (Sect. 4) to sustain a given end-to-end latency and a given end-to-end reliability, while minimizing its power consumption. Therefore, we investigate how a node can cope with network loss (Sect. 5.1) and sustain end-to-end requirements when connected to a router with (Sect. 5.2) or without radio duty cycle constraints (Sect. 5.3). We also discuss how a node can sustain bounds on packet transmissions and receptions (Sect. 5.4).

5.1 Sustaining Reliability

First, we investigate how a BLE node can sustain a given transmission reliability ($\widehat{r}_{MIN}$) by dynamically adapting the number of necessary transmissions of application packets based on the current transmission reliability (r_{NET}) over the network path. As shown in Sect. 2, no packets are dropped within the BLE subnet, due to the autonomous packet retransmission and flow control of the BLE link layer. Since we use the light-weight UDP transport layer, as most constrained IoT applications do [4], packets may be dropped on the Internet. To cope with this loss, we may need to send additional data packets to sustain our given reliability.

Estimating transmission reliability. To estimate the current reliability across the network (r_{NET}), we use a moving average filter with a window length W_{LOSS} on ordinary data transmissions. When we receive an acknowledgment for an application data packet within a timeout $t_{Timeout}$, we count this transmission as successful (r_{PKT} = 1). Otherwise, we assume the transmission has failed (r_{PKT} = 0). After every transmission, the current r_{NET} is calculated as:

$$r_{NET} = \frac{1}{W_{LOSS}} \sum_{j=1}^{W_{LOSS}} r_{PKT}(j). \tag{16}$$

Adapting transmission attempts. By knowing the current r_{NET} value of the network path, we can dynamically adapt the number of application transmission attempts necessary to achieve the given minimum transmission reliability ($\widehat{r}_{MIN}$). Compared to other Internet traffic, our data and acknowledgment packets are short and rather infrequent, which means that the BLE nodes will hardly cause congestion in the Internet path just because of few additional packets.

Similar to other research [39, 40], we assume that packet loss over the network path is independent and identically distributed with a success rate of r_{NET}, which means we can model it with the binomial distribution as:

$$P(X = k) = \binom{n}{k} r_{TX}^{k} (1 - r_{NET})^{n-k}, \tag{17}$$

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

In our case, we need at least one transmission attempt to have a successful packet exchange between node and server. The end-to-end reliability $\widehat{r}_{MIN}$ can be calculated as:

$$\widehat{r}_{MIN} = P(X \geq 1) = 1 - P(X = 0). \tag{18}$$

By using Eq. 17 and $\binom{n}{0} = 1$, $\widehat{r}_{MIN}$ can be calculated as:

$$\widehat{r}_{MIN} = 1 - (1 - r_{NET})^{n}. \tag{19}$$

For a given $\widehat{r}_{MIN} < 1$ and an estimated r_{NET}, we calculate the number of necessary transmission attempts (N_{TX}) as:

$$N_{TX} = n = \begin{cases} \left\lceil \frac{log(1-\widehat{r}_{MIN})}{log(1-r_{NET})} \right\rceil & \text{if } r_{NET} < 1 \\ 1 & \text{if } r_{NET} = 1 \end{cases} \tag{20}$$

By sending N_{TX} data packet attempts, we can sustain the given application reliability ($\widehat{r}_{MIN}$). Next, we investigate the necessary timing of the N_{TX} transmissions so that the application also sustains a given maximum end-to-end latency.

5.2 Sustaining Latency: Adapting CI & SL

Next, we investigate how a BLE node can sustain a given upper end-to-end latency bound when connected to a router that needs to limit its BLE radio duty cycle.

Some router devices do not have a continuous power supply (*e.g.*, smartphones) and/or need to sustain a BLE connection with a large number of BLE nodes simultaneously. In such cases, it it necessary that a BLE node requires as little BLE radio time on the router as possible. If BLE nodes would require huge portions of the BLE radio duty cycle of the router, *e.g.*, because the router needs to check every 10 ms if the node has data to send, the node would drain the router's battery or would limit the number of simultaneous BLE connections that can be offered on the router.

In these scenarios, a BLE node needs to adapt the BLE connection interval (*CI*) and the BLE slave latency (*SL*) to sustain its latency bounds while limiting its power draw and the radio duty cycle on the router.

5.2.1 *Transmitting Data*

We show how a BLE node can choose suitable *CI* and *SL* values to transmit data with a given end-to-end latency ($\widehat{t}_{TX_{MAX}}$) and a given end-to-end reliability ($\widehat{r}_{MIN}$) to a server.

From Sect. 5.1, we know that we need to transmit N_{TX} application packets within $\widehat{t}_{TX_{MAX}}$ to sustain $\widehat{r}_{MIN}$. Therefore, we split $\widehat{t}_{TX_{MAX}}$ into N_{TX} equal time slots. In each of these time slots one transmission attempt is initiated.

We can now use Eq. 4 to calculate the bound on the connection interval (CI) that allows us to sustain the latency requirements for transmitting packets from node to server as:

$$CI \leq \frac{\widehat{t}_{TX_{MAX}}/N_{TX} - t_{NET_{MAX}} - t_{CE}}{n_{CE_{MAX}} \cdot \lceil D_{TX}/F_{TX} \rceil}. \quad (21)$$

To minimize the power consumption on the BLE node, we can also choose its BLE slave latency (SL) as:

$$SL = n_{CE_{MAX}} \cdot \lceil D_{TX}/F_{TX} \rceil - 1. \quad (22)$$

This allows the node to skip unnecessary connection events, where only mandatory keep-alives would be exchanged.

5.2.2 Receiving Data

In this section, we show how a BLE node can choose suitable CI and SL values to receive data within a given end-to-end latency ($\widehat{t}_{RX_{MAX}}$) and a given end-to-end reliability ($\widehat{r}_{MIN}$) from a server. In this case, the server needs to account for any loss over the Internet and adapt its transmission attempts, as discussed in Sect. 5.1. The node gets N_{RX} from the server, after it has calculated the necessary transmission attempts.

Using Eq. 9, we can calculate the bound on CI that allows to sustain the given end-to-end latency as:

$$CI \leq \frac{\widehat{t}_{RX_{MAX}}/N_{RX} - t_{NET_{MAX}} - t_{CE}}{n_{CE_{MAX}} \cdot \lceil D_{RX}/F_{RX} \rceil + SL}. \quad (23)$$

Furthermore, we choose $SL = 0$ for this scenario. This allows the node to sustain a given upper bound on the reception latency while limiting unnecessary BLE radio time on the router due to skipped connection event.

5.3 Sustaining Latency: Adapting SL Only

In contrast to Sect. 5.2, we next investigate how a BLE node can sustain a given upper end-to-end latency bound on transmitting or receiving messages when the router has no constraints on its BLE radio duty cycle (*i.e.*, the router is not constrained in its power consumption) and scalability (*i.e.*, the number of connected BLE devices) is not an issue.

In this scenario, the router provides the BLE node with the smallest possible BLE connection interval (CI) during BLE connection setup that the router can sustain. The BLE node uses the provided CI and only adapts the BLE slave latency (SL) according to its application latency requirements.

5.3.1 Transmitting Data

Similar to Sect. 5.2.1, the BLE node first calculates the number of necessary data transmission attempts (N_{TX}) to sustain the given application reliability ($\widehat{r}_{MIN}$).

In contrast to Sect. 5.2, where the node needs to calculate a suitable CI, the BLE node already uses the fastest CI possible and only needs to adapt SL to conserve power by limiting unnecessary empty connection events. To limit these unnecessary connection events, we set SL as:

$$SL = \lceil I_{TX}/CI \rceil - 1, \quad (24)$$

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

Using this SL, the node only needs to wake up when it has data to transmit. During the remaining connection events, the node sleeps for SL events to minimize power draw.

5.3.2 Receiving Data

Similar to the data transmission case, the BLE node uses the CI value provided by the router and only adapts the SL according to the end-to-end timing requirements for receiving packets from the server. To sustain the given end-to-end reception latency, we use Eq. 9 and solve for SL as:

$$SL \leq \frac{\widehat{t}_{RX_{MAX}}/N_{RX} - t_{NET_{MAX}} - t_{CE}}{CI} - n_{CE_{MAX}} \cdot \left\lceil \frac{D_{RX}}{F_{RX}} \right\rceil. \quad (25)$$

Using such a SL allows the node to skip most BLE connection events while sustaining the desired latency bound.

5.4 Discussion

To sustain end-to-end latency bounds on transmitting and receiving packets simultaneously, a node independently calculates suitable parameters for transmitting (CI_{TX} and SL_{TX}) and receiving (CI_{RX} and SL_{RX}) using the formulas above.

To ensure that both end-to-end latency bounds are sustained, the node uses a value of $CI = MIN(CI_{TX}, CI_{RX})$ and $SL = MIN(SL_{TX}, SL_{RX})$ as its BLE connection parameters.

6 Implementation

In this section, we present the implementation of our proposed adaptation strategies on the Nordic Semiconductor `nRF52840 DK` [26]. This platform uses an ARM Cortex-M4F CPU, comes with 1024 kB of flash and 256kB of RAM, and embeds a radio supporting BLE communication up to version 5. While we use the `nRF52840` platform, our implementation also runs on all `nRF52` platform variants.

For our adaptation mechanisms, we use only fully standardized BLE functionality, which means that our implementation can be easily ported to other hardware platforms that support BLE version 4.1 and above, such as the Texas Instruments CC26xx platform.

We use the Zephyr OS [37] for our implementation, because this OS already includes a BLE and IPv6-over-BLE communication stack fully compliant to the BLE specification and uses the standardized HCI to exchange data between the BLE controller and host. We extend Zephyr's IPv6-over-BLE application on the node, which is located on the BLE host, by our proposed adaptation mechanisms. Our node application waits for the router to initiate an IPv6-over-BLE connection. After the connection is initiated, the node transmits a UDP packet with an IPv6 packet length of 128 bytes to the server, which runs a Python UDP server application in an Amazon Web Service (AWS) instance in Frankfurt. As soon as the node receives the first valid server acknowledgment, it starts to probe $t_{NET_{MAX}}$ using a probe burst length of $L_{Probe} = 10$. After a first $t_{NET_{MAX}}$ estimation is available, the node calculates the most suitable CI and SL configuration, as described in Sect. 5, and configures these values by sending a BLE `LL CONNECTION UPDATE REQUEST` to the router.

For both adaptation approaches, we use an ACK timeout $t_{Timeout} = 2000\,ms$ and a probing interval $I_{Probe} = 1000\,s$. We monitor the n_{CE} values of the underlying BLE connection by following the approach presented in [36]. Therefore, we add n_{CE} estimation on the BLE host in the HCI driver layer (`hci_core`) and monitor the `HCI ACL Data Packet` command and `HCI Number of Completed Packets` event to get $t_{TX_{BLE}}$ and n_{CE_f} for every transmitted application packet.

Table 2. Delayed packet and maximum number of subsequently delayed packets (max. delays) of the different node configurations under heavy Wi-Fi interference.

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

Whenever a new n_{CE_f} value is available, the node application is notified via a callback and can update the BLE connection parameters according to Sect. 5.

The current $n_{CE_{MAX}}$ value of the BLE connection is calculated via a moving maximum filter with a window length of 100 most recent n_{CE_f} measurements, as described in [36]. We further use a lower bound of $n_{CE_{MAX}} = 2$ in our implementation, which means that we slightly overestimate loss over the BLE connection, even when no link-layer errors happen during recent packet transmissions.

Our BLE connections do not make use of BLE data channel blacklisting and use the 2M PHY Mode of BLE.

7 Evaluation

We evaluate our proposed adaptation strategies experimentally. We start by evaluating the detailed behavior of both proposed approaches, which we call `Adapt CI & SL` (Sect. 5.2) and `Adapt SL only` (Sect. 5.3). We do so in the presence of dynamic changes on the network path (Sect. 7.1) and further provide a comparison with other BLE node configurations (Sect. 7.2). Specifically, we compare our approaches against a node using the fastest possible static BLE connection parameters ($CI = 7.5ms$ & $SL = 0$) and a node using static and power efficient BLE connection parameters ($CI = 1000ms$ & $SL = 0$).

7.1 Systematic Evaluation

To show how our proposed solutions can cope with dynamic changes in the BLE subnet and on the external network path, we focus on a node transmitting packets to the server using a wired Internet connection. All BLE nodes are configured to sustain a $\widehat{r}_{MIN} = 99\%$ and a $\widehat{t}_{TX_{MAX}} = 1000ms$.

7.1.1 Changes in the BLE subnet

First, we investigate how a node can adapt to sudden changes (*e.g.*, Wi-Fi interference) in the local BLE subnet.

Setup. We use the experimental setup described in Sect. 2 and start the experiment by setting up the connection between node and server and wait for 60 s until all initial adaptations are done. After this initial phase, we introduce heavy and continuous Wi-Fi interference on two different Wi-Fi channels (sending 1500 bytes of Wi-Fi data every 10 ms with a TX power of 50 mW) using two different Rasperry Pi 3 devices in our testbed running Jamlab-NG [32].

Results. Tab. 2 shows the number of delayed UDP packets (delayed) for each of the four different node configurations over the 10 min after the Wi-Fi jamming was started, across 5 runs for every node configuration. Furthermore, the table shows the maximum number of subsequently delayed packets (max. delays) across all test runs for every configuration.

We can see that both of our proposed adaptation approaches (`Adapt SL only` and `Adapt CI & SL`) are able to effectively cope with link-layer errors in the BLE subnet. While the `Adapt SL only` approach results in no end-to-end packet delays, also the `Adapt CI & SL` approach results in below 1% of all packet transmissions being delayed.

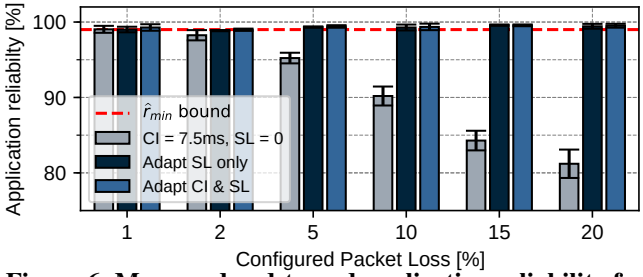


Figure 6. Measured end-to-end application reliability for different node configurations and configured packet loss.

7.1.2 Changes in network loss

Next, we evaluate the performance of our adaptation approaches when the transmission reliability of the external network path (r_{NET}) changes.

Setup. We use the setup from Sect. 2 to establish a connection between node and server and wait for 60 s until all initial adaptation is done. Next, we lower r_{NET} by either 1, 2, 5, 10, 15, or 20% and measure the resulting end-to-end application reliability, *i.e.*, how many node packets are successfully received within $\widehat{t}_{TX_{MAX}}$, for 600 s. To reproducibly lower r_{NET}, we use the standard Traffic Control (`tc`) tool with its Network Emulator (`netem`), which are both part of Linux distributions. We use `tc` on the all outgoing IPv6 packets on the router and the AWS server to mimic symmetric network loss.

Results. Fig. 6 shows the average transmission reliability of our application packets for three different node configurations across 5 experimental runs per configuration. We can see that both of our adaptive approaches successfully sustain the configured $\widehat{r}_{MIN} = 99\%$, by increasing the transmission attempts sent for every application data packet.

7.2 Comparison

We next investigate how our approaches perform in comparison to other node configurations.

Setup. Again, we use our testbed setup (see Sect. 2) to measure end-to-end latency and power consumption of nodes. We program a node with one of the different node configurations, setup the communication between node and server, and initially wait 60 s before we start our data collection.

To measure the performance of the different node configurations, we measure the number of application packets that exceed the specified latency bound (delayed pkts.). We further measure how the different node configurations affect the BLE radio duty cycle of the router (RDC_{Router}) by monitoring GPIO events of our router's USB-BLE radio, which triggers a GPIO event whenever the BLE radio is enabled. To investigate the power efficiency of the different node configurations, we measure the current consumption of the BLE node (I_{Node}) using D-Cube [31]. For this experiments, all log messages and unused peripheral devices on the BLE node are disabled to minimize the overall power consumption.

In addition to the two static and two adaptive node configurations, we also measure the performance of the adaptation approach presented in [36] in this experiment. This

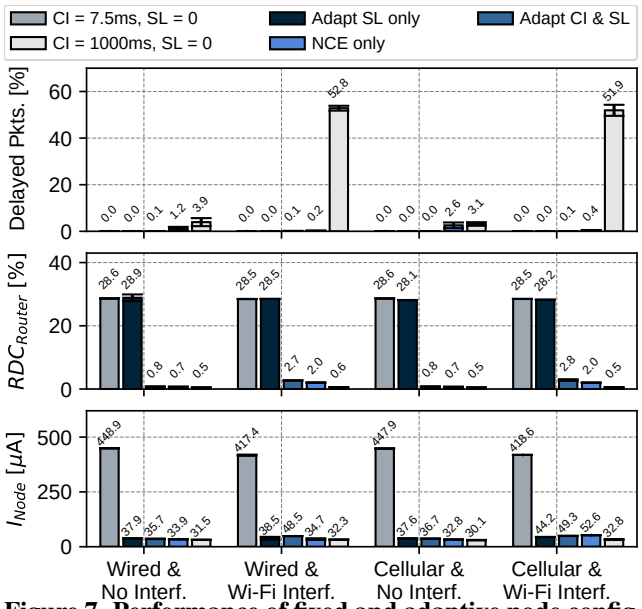


Figure 7. Performance of fixed and adaptive node configurations for sustaining $\widehat{t}_{TX_{MAX}} = 1000\,ms$ on transmitting packets with a length of 128 bytes to a cloud server.

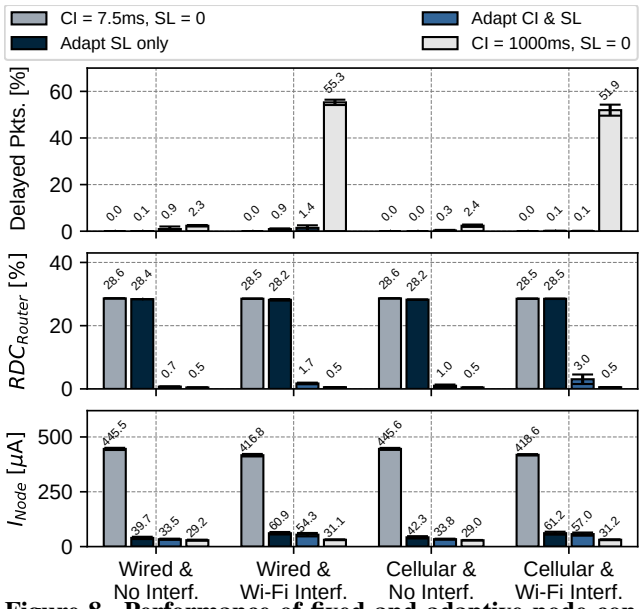


Figure 8. Performance of fixed and adaptive node configurations for sustaining $\widehat{t}_{RX_{MAX}} = 1000\,ms$ on receiving packets with a length of 128 bytes from a cloud server.

approach, which we call `NCE only`, only reacts to changes in the BLE subnet and does not account for any network delay. As the `NCE only` approach only sustains transmission bounds, it is not included in our packet reception experiment.

Results. Fig. 7 shows the performance of four different node configurations when the node tries to sustain a $\widehat{t}_{TX_{MAX}} = 1000\,ms$ in the four different experimental environments from Sect. 4.3. Every experimental run was repeated 5 times. We can clearly see that our `Adapt CI & SL` approach results in at most 0.1% of transmission being delayed, while also limiting the router's radio duty cycle and the slaves current consumption. If we follow the `Adapt SL only` approach, no transmissions are delayed. This, however, comes with the cost of at least a factor 10 increase of the RDC on the router. Furthermore, we can see that both of our approaches also significantly outperform the `NCE only` approach in sustaining $\widehat{t}_{TX_{MAX}}$, by at least 100%.

Similarly, Fig. 8 shows the performance of the different node configurations when the node tries to sustain a $\widehat{t}_{RX_{MAX}} = 1000\,ms$. Also in this setting, we can clearly see that both of our adaptation approaches result in at most 1.4% of packets receptions being delayed, while limiting the power consumption of the BLE node. Similar to Fig. 7, the `Adapt SL only` approach results in less delayed packets than the `Adapt CI & SL` approach, at the cost of a higher RDC_{Router}.

8 Related Work

Cloud-based BLE applications. There exists a vast number of BLE-based applications that communicate with a cloud server in time-critical domains, such as smart-grid [8], smart city [15], or smart health applications [3, 16, 22, 38]. None of these works, however, adapt their communication to delay or loss in the BLE connection or the remaining network path.

Contrary to these works, our paper shows how BLE-based IoT applications can sustain end-to-end reliability and latency bounds for communication between nodes and a server, while limiting the power draw of the BLE devices.

BLE Modeling. Existing BLE timeliness models cannot be used on off-the-shelf BLE devices, as they require BLE link-layer information, such as CRC error count or bit error rate, which are not available to BLE applications [23, 29, 34]. Only a few works [27, 36] make use of information that is available on every standard BLE device to estimate and control the transmission delay on a BLE connection. The major problem with these models, however, is that they only model the transmission delay of packets in a BLE subnet, and hence cannot be used for communication spanning across multiple networks. Furthermore, these models either only investigate the effects of a single BLE connection parameter on communication latency [27, 36, 34] or do not apply to packets that need to be received within given latency bounds [23, 29].

This paper, to the best of our knowledge, presents the first model capturing the effect of all BLE communication parameters on the timeliness of traffic from and to a BLE node.

Dependable low-power wireless communication. A plethora of low-power wireless research studies have investigated how to sustain time-critical communication while minimizing power consumption in IEEE 802.15.4 [14, 17, 42] and WirelessHART applications [7, 10, 24]. These works, however, *only* focus on sustaining latency and reliability within their low-power networks. Some low-power wireless research have investigated dependable communication over multiple networks, but neglect end-to-end timeliness [4, 6] or explicitly exclude communication on the Internet [9].

In this work, instead, we investigate how low-power and constraint nodes can sustain end-to-end dependability requirements on traffic from and to a cloud server.

Estimating Internet characteristics. Capturing and mitigating loss and delays across the Internet has been extensively studied over the last decades. The proposed mech-

anisms to estimate end-to-end delay or loss, however, rely on complex primitives, such as multiple Kalman filters [20], machine learning [30] or two-level markov models [11].

In this paper, we present a simple and efficient approach that accurately estimates the maximum network delay across the Internet. We use the well-established RTT estimation of TCP [18] as a base and revise it to be use on low-power, asymmetric links, such as BLE connections.

Tactile Internet. Research works on the Tactile Internet have investigated how to sustain minimal end-to-end latency and maximum reliability on the Internet [12]. These works, however, focus on optimizing 5G communication to reach high data rates and round trip latencies below 1 ms [21, 33].

Contrary to these studies, our work focus on sustaining given end-to-end dependability requirements while minimizing the power consumption of BLE devices.

9 Conclusion and Future Work

State-of-the-art BLE-based IoT applications are not able to cope with delays or loss along the whole network path between nodes and cloud. In this work, we show how BLE nodes can estimate and mitigate network loss and delay by dynamically adapting their BLE parameters. Using our approaches, nodes are able to sustain end-to-end latency and reliability requirements while minimizing their power draw.

Our next steps include combining our adaptation approaches with sophisticated BLE channel management, such as the work presented in [35], to improve the performance of nodes sustaining end-to-end dependability requirements even further. Furthermore, our adaptation approaches can be used in combination with application-level improvements, such as data reduction, compression, and prediction mechanisms used in Tactile Internet applications [21].

10 Acknowledgments

This work has been performed within the LEAD project "Dependable Internet of Things in Adverse Environments" funded by Graz University of Technology.

11 References

[1] A. Abdel-Hadi and C. Clancy. A Robust Optimal Rate Allocation Algorithm and Pricing Policy for Hybrid Traffic in 4G-LTE. In *Proc. of the* 24^{th} *Int. IEEE PIMRC Symposium*, 2013.

[2] A. Abdel-Hadi et al. A Utility Proportional Fairness Approach for Resource Allocation in 4G-LTE. In *Proc. of the ICNC Conf.*, 2014.

[3] G. Alfian et al. A Personalized Healthcare Monitoring System for Diabetic Patients by Utilizing BLE-Based Sensors and Real-Time Data Processing. *Sensors*, 18(7), 2018.

[4] A. Betzler et al. CoAP Congestion Control for the Internet of Things. *IEEE Communications Magazine*, 54(7), 2016.

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

[6] R. Brummet et al. A Flexible Retransmission Policy for Industrial Wireless Sensor Actuator Networks. In *Proc. of the IEEE ICII Conf*, 2018.

[7] Y. Chen et al. Probabilistic Per-Packet Real-Time Guarantees for Wireless Networked Sensing and Control. *IEEE Transactions on Industrial Informatics*, 14(5), 2018.

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

[9] DetNet Working Group. Deterministic Networking (detnet), 2020.

[10] B. Dezfouli et al. Real-time Communication in Low-Power Mobile Wireless Networks. In *Proc. of the 13th IEEE CCNC Conf.*, 2016.

[11] M. Ellis et al. A two-level Markov model for packet loss in UDP/IP-based real-time video applications targeting residential users. *Computer Networks*, 70, 2014.

[12] G. Fettweis. The Tactile Internet: Applications and Challenges. *IEEE Vehicular Technology Magazine*, 9(1), 2014.

[13] S. Forconi and A. Vizzarri. Review of Studies on End-to-End QoS in LTE Networks. In *AEIT Annual Conference 2013*. IEEE, 2013.

[14] G. Franchino and G. Buttazzo. A power-aware MAC layer protocol for real-time communication in wireless embedded systems. *Journal of Network and Computer Applications*, 82, 2017.

[15] S. Geetha and S. Gouthami. Internet of things enabled real time water quality monitoring system. *Smart Water*, 2(1), 2016.

[16] M. Hasan et al. Real-time healthcare data transmission for remote patient monitoring in patch-based hybrid OCC/BLE networks. *Sensors*, 19(5), 2019.

[17] R. Jacob et al. End-to-end Real-time Guarantees in Wireless Cyber-physical Systems. In *Proc. of the 2016 IEEE RTSS Symposium*, 2016.

[18] V. Jacobson. Congestion Avoidance and Control. In *Proc. of the 1988 SIGCOMM Symposium*, 1988.

[19] K. Jacobsson et al. Estimation of RTT and Bandwidth for Congestion Control Applications in Communication Networks. In *IEEE CDC*, 2004.

[20] K. Jacobsson et al. Some Modeling and Estimation Issues in Control of Heterogeneous Networks. In *MTNS*, 2004.

[21] K. K. Antonakoglou et al. Toward Haptic Communications Over the 5G Tactile Internet. *IEEE Communications Surveys & Tutorials*, 2018.

[22] P. Kakria, N. Tripathi, and P. Kitipawang. A real-time health monitoring system for remote cardiac patients using smartphone and wearable sensors. *International Journal of telemedicine and applications*, 2015.

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

[24] Y. Ma et al. Optimal Dynamic Scheduling of Wireless Networked Control Systems. In *Proc. of the* 10^{th} *Int. ACM/IEEE CPS Conf.*, 2019.

[25] J. Nieminen et al. RFC 7668 - IPv6 over Bluetooth Low Energy, 2015.

[26] Nordic Semiconductors. nRF52840 Specifications, 2020.

[27] E. Park et al. AdaptaBLE: Adaptive Control of Data Rate, Transmission Power, and Connection Interval in Bluetooth Low Energy. *Computer Networks*, 2020.

[28] V. Paxson et al. RFC6298: Computing TCP's Retransmission Timer, 2011.

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

[30] P. S. Rossi et al. Joint End-to-End Loss-Delay Hidden Markov Model for Periodic UDP Traffic Over the Internet. *IEEE Transactions on Signal Processing*, 54(2), 2006.

[31] M. Schuß et al. Moving Beyond Competitions: Extending D-Cube to Seamlessly Benchmark Low-Power Wireless Systems. In *Proc. of the* 1^{st} *CPSBench Worksh.*, 2018.

[32] M. Schuß et al. JamLab-NG: Benchmarking Low-Power Wireless Protocols under Controllable and Repeatable Wi-Fi Interference. In *Proc. of the* 16^{th} *EWSN Conf.*, 2019.

[33] M. Simsek et al. 5G-enabled Tactile Internet. *IEEE Journal on Selected Areas in Communications*, 34(3), 2016.

[34] M. Spörk et al. BLEach: Exploiting the Full Potential of IPv6 over BLE in Constrained Embedded IoT Devices. In *Proc. of the* 15^{th} *ACM SenSys Conf.*, 2017.

[35] M. Spörk et al. Improving the Reliability of Bluetooth Low Energy Connections. In *Proc. of the* 17^{th} *Int. EWSN Conf.*, 2020.

[36] M. Spörk et al. Improving the Timeliness of Bluetooth Low Energy in Dynamic RF Environments. *ACM Transactions on IoT*, 2020.

[37] The Zephyr Project. Zephyr OS: An RTOS for IoT, 2020.

[38] F. Touati, R. Tabish, and A. B. Mnaouer. A Real-Time BLE enabled ECG System for Remote Monitoring. *APCBEE procedia*, 7, 2013.

[39] K. Winstein et al. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *Proc. of the* 10^{th} *USENIX NSDI Symposium*, 2013.

[40] F. Yan et al. Pantheon: the training ground for Internet congestion-control research. In *Proc. of the 2018 USENIX ATC Conf.*, 2018.

[41] Y. Zaki et al. Adaptive Congestion Control for Unpredictable Cellular Networks. In *Proc. of the 2015 ACM SIGDC Conf.*, 2015.

[42] M. Zimmerling et al. Adaptive Real-Time Communication for Wireless Cyber-Physical Systems. *ACM Transactions on CPS*, 1(2), 2017.