Thesis for The Degree of Doctor of Philosophy

Adaptive and Resource-Efficient Systems for the Internet of Things

Protocols, Systems, and Evaluation Infrastructures

Laura Harms

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2024

Adaptive and Resource-Efficient Systems for the Internet of Things

Protocols, Systems, and Evaluation Infrastructures

Laura Harms

ISBN 978-91-8103-027-3
Doktorsavhandlingar vid Chalmers Tekniska Högskola
Ny serie nr 5485
ISSN 0346-718X

Department of Computer Science & Engineering
Division of Computer and Network Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Phone: +46 (0)31 772 1000

This thesis has been prepared using LaTeX.
Printed by Chalmers Digitaltryck,
Gothenburg, Sweden 2024.

In loving memory of my sister Julia.
Thank you for all the time we had together.

Adaptive and Resource-Efficient Systems for the Internet of Things

Protocols, Systems, and Evaluation Infrastructures

Laura Harms

Department of Computer Science & Engineering
Chalmers University of Technology

Abstract

With the growing number of Internet of Things (IoT) devices and the emergence of the Industrial Internet of Things (IIoT), there is a growing demand for adaptive and resource-efficient wireless communication protocols and systems. Industrial networks play a crucial role in monitoring pipelines and facilitating communication among collaborating devices, such as robots in a smart factory. These applications are safety-critical and necessitate long-term reliable and low-latency communication. However, the rising number of IoT communicating devices and deployments increasingly congests the wireless medium, which leads to interference and makes the latency and reliability requirements more challenging to accomplish. Current solutions and protocols are incapable of addressing these evolving demands. Therefore, there is a need for novel communication protocols and systems capable of dynamically adapting to unforeseen interference and changes in the wireless medium.

In this thesis, we design, implement, and evaluate protocols, systems, and evaluation infrastructures tailored for modern IoT solutions. To facilitate long-term stable communication within centrally scheduled IEEE 802.15.4 Time-Slotted Channel Hopping (TSCH) networks, we propose a centralized scheduler and a flow-based retransmission strategy. This strategy allocates retransmissions to be utilized at any node within a communication flow, thereby enhancing resilience against unforeseen interference. We then introduce Autobahn, a communication protocol that integrates opportunistic routing and synchronous transmissions with TSCH to mitigate local wideband interference while keeping latency to a minimum. With TBLE, we bring TSCH to Bluetooth Low Energy (BLE), further reducing latency without compromising reliability. To provide comprehensive insights into distributed wireless communication protocols on testbeds, we propose Grace, a low-cost time-synchronized General-Purpose Input/Output (GPIO) tracing system for existing testbeds. Finally, we demonstrate with BlueSeer that a device can recognize its environment—such as home, office, restaurant, or street—solely from received ambient BLE signals using an embedded machine learning model. BlueSeer enables small IoT devices like wireless headphones to adapt their behaviors to the surrounding environment.

Keywords

Internet of Things, IoT, Industrial Internet of Things, IIoT, Time-Slotted Channel Hopping, TSCH, Centralized Scheduling, Routing, Opportunistic Routing, Synchronous Transmissions, Time-Synchronization, Bluetooth Low Energy, BLE, IEEE 802.15.4, TinyML

Acknowledgment

After 6.5 years, my journey towards obtaining a PhD is drawing to a close. Throughout this time, I have had the privilege of meeting many remarkable people, and I wish to express my heartfelt gratitude to each of you for transforming this journey into an enriching experience.

Foremost, I extend my deepest thanks to my supervisor, Olaf Landsiedel, for embarking on this journey with me and providing unwavering support throughout these years. Your mentoring has been invaluable in helping me grow as a researcher. I really appreciate how you have always been there to answer my questions and give me feedback, even when you had plenty of other things going on. I'm also grateful to my co-supervisor, Magnus Almgren, for keeping the link to Chalmers alive, even after my relocation to Kiel.

I am grateful to my colleagues and fellow PhD students at Chalmers for providing a pleasant work environment and for the wonderful fika breaks we had. To Dimitris, Christos, Georgia, Beshr, Babis, Bastian, Thomas, Karl, Hannah, Carlo, Francisco, Aljoscha, Fazeleh, and Romaric, I extend my sincere appreciation. Additionally, I express gratitude to Tomas, Monica, Rebecca, Clara, and Marianne, without whom working and doing a PhD at Chalmers would not have been possible. Furthermore, I extend thanks to my examiner, Gerardo Schneider, for his support in facilitating my PhD continuation at Chalmers while residing in Germany.

In Kiel, I extend my gratitude to Valentin, my companion from the outset of my PhD journey at Chalmers, who, alongside me, followed Olaf to Kiel. Thank you for the countless discussions we had when sharing an office, for the collaboration, as well as our joint travels around the globe. I extend my thanks to Patrick and Janek for being excellent office mates, Christian for our collaboration and technical discussions, and Steffi, Gerd, and Brigitte, along with my fellow PhD students Birkan, Naina, Tayyaba, Marc, Momin, Kainat, Julia, and Ali, for their contributions to a great and friendly work environment.

I am grateful to Ambuj for igniting my passion for research during my master's thesis, as without your inspiration, the prospect of pursuing a PhD may never have crossed my mind.

Lastly, I offer a heartfelt thank you to my parents, Agnes and Rainer, my sisters Melanie and Julia, and my best friend Marcel, for always being there for me, no matter how far apart we are. *Danke!*

Laura Harms
Kiel, March 2024

List of Publications

Appended publications

This thesis is based on the following publications:

[A] **L. Harms**, O. Landsiedel
"MASTER: Long-Term Stable Routing and Scheduling in Low-Power Wireless Networks"
Proceedings of the 16th International Conference on Distributed Computing in Sensor Systems (DCOSS), pp. 86-94, 2020.

[B] **L. Harms**, O. Landsiedel
"Opportunistic Routing and Synchronous Transmissions Meet TSCH"
Proceedings of the 46th IEEE Conference on Local Computer Networks (LCN), pp. 107–114, 2021.

[C] V. Poirot, **L. Harms**, H. Martens, O. Landsiedel
"BlueSeer: AI-Driven Environment Detection via BLE Scans"
Proceedings of the ACM/IEEE Design Automation Conference (DAC), pp. 871–876, 2022.
This paper was nominated as a candidate for the best paper award.

[D] **L. Harms**, C. Richter, O. Landsiedel
"Grace: Low-Cost Time-Synchronized GPIO Tracing for IoT Testbeds"
Elsevier Computer Networks, vol. 228, p. 109746, 2023.
The above is an extended version of the work that previously appeared in:
L. Harms, C. Richter, O. Landsiedel
"Grace: Low-Cost Time-Synchronized GPIO Tracing for IoT Testbeds"
Proceedings of the 18th International Conference on Distributed Computing in Sensor Systems (DCOSS), pp. 9–16, 2022.
This paper won the best paper award.

[E] **L. Harms**, O. Landsiedel
"TSCH meets BLE: Routed Mesh Communication over BLE"
Proceedings of the 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT), pp. 187–195, 2023.

Other publications

The following publications were published during my PhD studies, or are currently in submission/under revision. However, they are not appended to this thesis, due to contents overlapping that of appended publications or contents not related to the thesis.

[a] A. Varshney, **L. Harms**, C. Pérez-Penichet, C. Rohner, F. Hermans, T. Voigt
"LoRea: A Backscatter Architecture That Achieves a Long Communication Range"
Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems (SenSys), pp. 1–14, 2017.

[b] **L. Harms**
"C-TSCH: A Centralized Scheduler for TSCH"
Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN), pp. 314–315, 2019.
Poster Abstract

[c] **L. Harms**, O. Landsiedel
"Competition: Centrally Scheduled Low-Power Wireless Networking for Dependable Data Collection"
Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN), pp. 300–301, 2019.
Poster Abstract

[d] **L. Harms**, O. Landsiedel
"(POSTER) Overtake: Opportunistic Routing and Concurrent Transmissions for TSCH"
Proceedings of the 16th International Conference on Distributed Computing in Sensor Systems (DCOSS), pp. 141–143, 2020.

[e] **L. Harms**
"BLE-Based and AI-Driven Environment Detection"
KuVS - AI in Networking Summer School, 2 pages, 2022.

[f] **L. Harms**, O. Landsiedel
"TBLE: Time-Synchronized Routed Mesh Communication for BLE"
Proceedings of the 20th GI/ITG KuVS Fachgespräch Sensornetze (FGSN 2023), pp. 5–6, 2024.

Contents

List of Figures

List of Tables

Thesis Overview

---

1 Introduction

Over the past century, our world transformed from a world without computers to a connected one. Nowadays, we are surrounded by computers and everyday devices with built-in computing capabilities. They aid our lives and always keep us connected with others through the Internet, a situation that was a mere utopia decades or even years ago. Nowadays, we are so used to always being online that we feel nervous when we are not [1]. Moreover, we are living in smart homes that react to our presence and make our lives more convenient. Many of us live in smart cities that are increasingly connected, providing us with live information and adapt to our demands, e.g., helping us to get around more easily.

The *Internet of Things (IoT)* is a main driver for this development of a connected world. The Internet of Things forms a network of physical everyday objects and enables them to communicate with each other. These objects, also referred to as *smart* objects, contain sensors or actuators that interface with the physical world, a communication interface, and embedded computing capabilities for data processing. IoT devices come in various forms, spanning from smart home devices like thermostats, spill detection sensors, and light bulbs to security cameras, smart electricity meters, fitness trackers, and many others. With the IoT, we connect smart devices, usually using wireless communication, to form local networks or even become part of the global internet. Many IoT devices are mobile or cannot be connected to mains power and thus have to run for years on batteries or be solar-powered instead. Therefore, they have tight energy budgets, supporting only limited computing and communication capabilities.

While we experience a drastic change in our daily lives with IoT devices all around us, this trend of smart and connected devices and automation is also present in industry. Over the past decades, process automation as part of Industry 3.0 [2,3] was a leading motive. However, for the past decade, Industry 4.0 [2–4] and the Industrial Internet of Things (IIoT) enhances automation by interconnecting industrial systems and creating collaborative ones, e.g., smart factories [4,5]. Connected industrial systems form sensor-actuator networks, which allow sensors and smart machines to autonomously exchange information to trigger actions or provide insights for machines or humans alike. Similar to the IoT, industrial IoT devices also use wireless communication as moving parts

or existing infrastructure are a hindrance for wired communication. Moreover, to retrofit IIoT solutions to existing infrastructure, battery operation is also common in industrial IoT.

The increasing number of connected and connectable devices both in the realm of consumer devices (IoT), and in infrastructure and industrial settings (IIoT) leads to new applications. These applications include public infrastructure in smart cities that can steer intelligent electricity consumption in households, observe and react to their inhabitants' needs, or smartly coordinate traffic flow. Moreover, highly or fully automated factories are an application unthinkable without networked communication between manufacturing steps. This increasing presence of IoT devices all around us not only leads to new communication-based applications. It also enables applications to exploit surrounding signals for localizing a device or intelligently reacting to the environment.

As the wireless spectrum is shared, deploying additional devices increases the complexity of communication for both existing and future deployments. With the wireless medium being inherently dynamic, the introduction of each new device amplifies the likelihood of interfering communications and exacerbates the unreliability of wireless transmissions. Consequently, future-proof communication protocols must possess the capability to dynamically adapt to changes in the wireless environment, ensuring stable communication and reliable operation. Furthermore, individual IoT devices must efficiently manage their energy and computational resources to facilitate adaptation within both the network stack and the application layer.

2 Motivation and Goals

Low-power wireless devices and the Internet of Things (IoT) have enabled novel applications, such as personal devices like wireless headphones or fitness trackers, connected smart cities, and sensor networks overseeing safety-critical manufacturing processes. Despite being battery-powered, these devices must uphold reliable communication and adhere to strict deadlines while operating within tight energy constraints. Moreover, they must contend with an inherently unreliable wireless medium, susceptible to environmental factors and interference from potentially stronger communication signals.

Over recent decades, numerous communication technologies (such as IEEE 802.15.4 [6] and Bluetooth Low Energy [7]) and protocols (e.g., [8–14]) have been introduced by both academia and industry to ensure reliable communication in wireless single-hop and multi-hop networks. IEEE 802.15.4 [6] finds primary application in industrial and smart home scenarios, while Bluetooth Low Energy (BLE) [7] stands as the current standard for consumer electronics. Some communication protocols, like Thread [13] and Matter [14], are now widely employed within the context of smart homes. Others, such as WirelessHART [12] and Orchestra [8], focus more on applications for the Industrial Internet of Things (IIoT) and achieve over 99.99% end-to-end delivery rates in multi-hop networks. However, many of these protocols are either best effort, emphasizing high reliability with less consideration for latency [8], or focus on maximum throughput while meeting deadlines but are

unable to maintain reliable communication in the face of unforeseen external interference [10, 11, 15, 16].

Challenges

To use wireless communication as an alternative to wired communication in an industrial setting, wireless communication has to be reliable and guarantee to meet deadlines and an application's latency requirements. Today's protocols achieve this by making strong assumptions regarding the level of interference, and thus only meet these requirements for known levels of interference. In the real-world, these assumptions do not hold. The wireless medium is inherently unreliable, and we can no longer guarantee an application to have sole access to a part of the wireless spectrum. Instead, protocols have to account for interference from other communication infrastructure and especially unforeseeable interference levels and changes in the wireless medium. Achieving reliable communication under these circumstances while giving latency guarantees and accounting for interference is challenging. Systems giving deadline and latency guarantees commonly use static communication schedules with static routes and static numbers of retransmissions that cannot react to local interference variations alongside a communication path. Therefore, the first challenge we address in this thesis is to devise protocols for the low-power IIoT that enable end-to-end communication flows to dynamically adapt to variations in the wireless environment to ensure stable and reliable communication with low latency. We envision a novel retransmission scheme for centrally scheduled networks that dynamically uses retransmissions wherever needed along a communication path. Moreover, we design a routing scheme that does not rely on a single path but can concurrently use multiple paths to circumvent interference sources without impacting latency.

For low-power communication in the license-free 2.4 GHz band, we see two protocols in use today: IEEE 802.15.4 and Bluetooth Low Energy (BLE). We see IEEE 802.15.4 in IIoT and smart home applications, while BLE is prevalent in smartphone-centric consumer applications but also in cheaper smart home devices. The latest iPhone 15 Pro supports Thread [17] and thus IEEE 802.15.4 in addition to BLE, but BLE radios are still cheaper and more widely available. Therefore, our next challenge is bringing reliable low-latency mesh networking to BLE, making it ready for applications in the Industrial Internet of Things. We solve this challenge by combining the IEEE 802.15.4 TSCH network stack with BLE replacing the IEEE 802.15.4 physical layer.

Next to challenges requiring new and better adjusted communication protocols, there is also the challenge of testing and evaluating these protocols. It is especially challenging to test and evaluate the inner workings of communication protocols. To gain fine-grained insight into concurrent executions on different devices in a distributed network, like an IoT network, it is essential that the evaluation infrastructure offers precise timestamping capabilities. While state-of-the art solutions require specific hardware, we address this challenge by building a low-cost time-synchronization system that is retrofittable to existing infrastructure, extending it by GPIO timestamping capabilities.

The last challenge we address in this thesis concerns IoT devices that move around, like smartphones, smartwatches, wireless headphones, and other

wearables. The environment these devices are in changes constantly, and the devices should adjust their behavior to the environment they are in. For example, headphones should lower their noise-cancellation levels near roads to ensure the safety of their users, and smartphones and smartwatches should keep silent while in a theater. To tackle this challenge, we envision and develop an adaptive IoT system capable of recognizing the current environment and communicating relevant information to the device's application. This enables the device to dynamically adjust to changes in its surroundings. We solve the challenge for devices that cannot rely on external sensors and thus have to recognize the environment solely with the help of a wireless radio, or more specifically a BLE radio.

Goals

From these challenges, we can derive the following goals:

1. We aim to create adaptive and resource-efficient low-power IoT systems. We tackle this problem in two ways, (1) at the level of communication protocols and (2) at the application level. Firstly, we must add dynamic retransmission and routing approaches to IIoT communication protocols, allowing them to react to changes in the wireless environment and thus enabling long-term stable reliable communication. Secondly, we must enable IoT devices to detect their environment to make adjustments to the device's functionality.

2. We want to reduce communication latency in low-power wireless mesh networks. Therefore, we must borrow aspects from different low-power wireless communication approaches and combine them into novel communication protocols. We address this by (1) combining communication approaches to dynamically route traffic around interference sources, and (2) replacing the IEEE 802.15.4 physical layer with the BLE physical layer.

3. We aim to cost-efficiently debug and evaluate timing-critical low-power communication protocols. Therefore, we must provide a retrofittable solution for existing testbeds that can timestamp concurrent events on multiple IoT devices.

Contributions

We address these goals with the following contributions:

A. We build MASTER[1], a centralized scheduler for TSCH networks. MASTER includes a novel and flexible flow-based retransmission strategy that dynamically uses retransmissions wherever necessary across links. MASTER addresses the first part of our first goal.

[1]We are aware of the discussion and stigma around the word master and discuss the name choice for the protocol in Section 6.1.

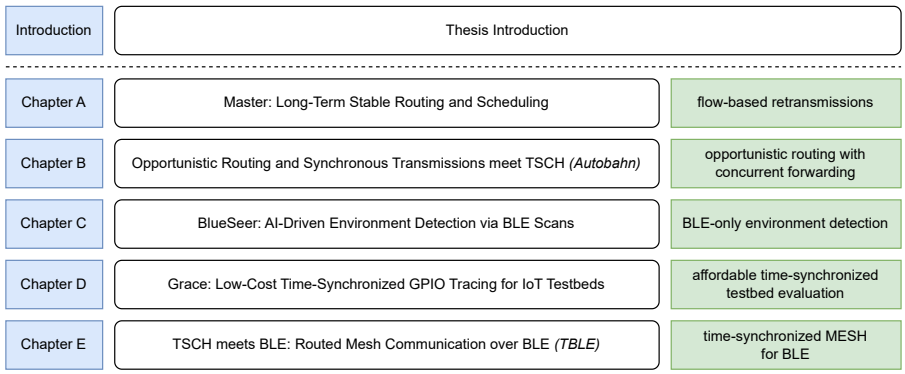


Figure 1: Structure of the thesis. It includes 6 chapters, one for the introduction and five for the appended publications in chronological order of publishing. For each chapter, we highlight the main contribution in the green box.

B. We design AUTOBAHN, a robust scheduling and routing policy that withstands link and node failures in the presence of interference. AUTOBAHN combines opportunistic routing, synchronous transmissions, and Time-Slotted Channel Hopping (TSCH) into a single protocol achieving low-latency and long-term stable communication addressing parts of the first two goals.

C. We build BlueSeer, an Environment detection system able to classify environments solely using received BLE packets. BlueSeer addresses the second part of our first goal and enables devices to adapt their functionality to their surrounding environment.

D. We introduce Grace, a low-cost time-synchronized GPIO tracing system for IoT testbeds. Grace accomplishes the third goal by using low-cost off-the-shelf components and is able to synchronize both building-wide and campus-wide testbeds.

E. We present TBLE, a protocol achieving routed mesh-communication in BLE. TBLE combines TSCH with BLE which reduces communication latency and addresses the second part of our second goal.

Approach

In this thesis, we employ experimental computer science methods. We design, implement, and experimentally evaluate IoT protocols and systems which enable stable and reliable low-latency communication and adaptive IoT systems. We make our designs, protocols, and implementations available to enable their use within the scientific community and beyond.

Outline

This thesis contains an introduction to the scope of this thesis and five articles published over the recent years. We organize the thesis into six chapters,

consisting of a thesis overview and the five articles. The thesis overview starts with an introduction to the topic of this thesis, and a motivation of the targeted problems. Afterward, we introduce the required background this thesis builds on and survey the current state-of-the-art of related research. We conclude this chapter with the research statement, an overview and discussion of the following articles, and an outlook towards future directions. The following chapters each include one of the five articles in chronological order of publishing. Figure 1 provides a visual overview of the structure of this thesis and each article's main contribution.

3 Background

In this section, we introduce the core concepts this thesis builds on and the necessary technical background we use in this thesis. We provide an overview of low-power wireless communication (Section 3.1), with a focus on short-range 2.4 GHz communication protocols, mainly IEEE 802.15.4 (Section 3.2) and Bluetooth Low Energy (Section 3.3). We overview medium access methods in IEEE 802.15.4, focusing on TSCH (Section 3.4) and synchronous transmissions (Section 3.5). Moreover, we present routing approaches in short-range communication systems. Afterward, we discuss time-synchronization (Section 3.6) and evaluation methods (Section 3.7) for wireless networks. Lastly, we provide an introduction into embedded intelligence and tiny machine learning (Section 3.8).

3.1 Low-Power Wireless Communication

Wireless communication in the field of the (Industrial) Internet of Things has to be highly reliable, meet deadlines, and adhere to tight energy budgets. It has to be energy efficient and low-power to allow its use on battery for long periods of time—sometimes decades. To achieve this, multiple wireless communication technologies have been proposed over the years that have to make a tradeoff between range and data rate. Wireless personal area networks (WPAN) offer data rates of hundreds of kbit/s up to a few Mbit/s achieving communication ranges of a few tens of meters [18]. This category of networks is most suitable for smart homes, smart factories, and devices we carry with us like wearables. For covering longer communication ranges and to boost communication reliability, they use multi-hop communication. The most prominent standards for short-range low-energy communication are IEEE 802.15.4 [6] and Bluetooth Low Energy (BLE) [7]. Both are narrowband communication standards operating in the license-free 2.4 GHz ISM band. IEEE 802.15.4 also supports subGHz frequencies [19], as well as higher frequencies above 3 GHz for ultra-wideband communication [19]. In contrast, low-power wide-area networks (LPWAN) like LoRa [20] and Sigfox [21] achieve high communication ranges of above 10 km with data rates of 100 bit/s up to tens of kbit/s [22], while cellular LPWANs like NB-IoT, and LTE-M [23] achieve higher data rates of hundreds of kbit/s with slightly shorter communication ranges. These communication ranges, make LPWANs perfectly suitable for citywide IoT networks or larger low-power networks outside of cities without requiring multi-hop communication. Thus, they are a perfect fit for applications such as smart cities and smart agriculture.

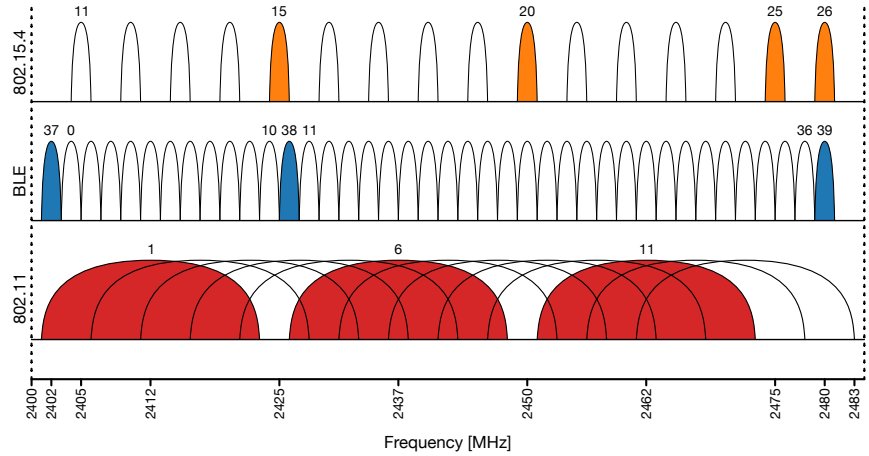


Figure 2: Channel mapping of BLE, IEEE 802.15.4, and IEEE 802.11 (WiFi) sharing the 2.4 GHz frequency band. BLE and IEEE 802.15.4 channels are 2 MHz wide with 2 MHz and 5 MHz spacing between center frequencies of adjacent channels, respectively. WiFi channels are 22 MHz wide. Blue: primary BLE advertising channels. Red: common, non-overlapping WiFi channels. Orange: commonly used 4-channel hopping sequence in IEEE 802.15.4 TSCH.

However, all of these LPWAN technologies require infrastructure like cell towers or base stations for their operation.

Within this thesis, we mainly focus on the former category of low-power wireless communication, more precisely, communication with IEEE 802.15.4 and Bluetooth Low Energy (BLE), operating in the 2.4 GHz ISM band.

3.1.1 Communication Challenges in the 2.4 GHz ISM Band

The 2.4 GHz ISM band is a shared wireless medium with several technologies including IEEE 802.15.4, BLE, and WiFi that have to coexist. Especially for low-power communication standards like IEEE 802.15.4 and BLE communication is more challenging in comparison to WiFi which transmits several orders of magnitude more powerful signals ($\leq$1 mW vs. 100 mW). However, not only the presence of WiFi is a challenge, but also the presence of multiple BLE or IEEE 802.15.4 networks and the coexistence of BLE and IEEE 802.15.4 is challenging as their signals are likely to collide and thus interfere with each other. Networks employing IEEE 802.15.4 or BLE do not uniformly adopt identical channel selection mechanisms or communication protocols, thereby increasing the probability of collisions. Additionally, as the number of devices increases, these challenges intensify further. To ensure reliable communication amidst interference, WPANs employ a range of medium access and networking strategies. These encompass multi-hop communication, retransmissions, channel hopping, and multi-path communication.

3.1.2 Medium Access Methods

To avoid interference within a single network and thus achieve reliable communication, multiple strategies to access the wireless medium exist. These strategies either set on avoiding collisions by probing the wireless medium to detect whether it is clear to send (e.g., Carrier Sense Multiple Access with Collision Avoidance: CSMA/CA), or split the medium to give exclusive access to a portion for a specific communication. We can split the wireless medium along the time-axis for exclusive access of the medium in specific timeslots, or along the frequency-axis, allowing concurrent communication on different frequency channels. We can increase the distance between concurrent transmissions, or we can use orthogonal codings to restore the signal from overlapping concurrent signals [24]. We can use each of these approaches individually or in combination.

3.1.3 Long-Distance Communication

While both IEEE 802.15.4 and BLE only have limited communication range, they often have to communicate over longer distances nonetheless to, e.g., transmit sensor data to an actuator that is not in the sensor's immediate communication range. While base stations like in LPWAN or WiFi networks would be a possibility, they require fixed infrastructure. Another approach is multi-hop communication, which is the fundamental method for mesh networks. In mesh networks, nodes between the data source and the destination act as a relay and receive data and forward it towards the destination. Mesh networks do not require fixed infrastructure. Instead, they only require a sufficiently dense mesh to transmit data from any potential sender to any potential receiver, and thus can form ad-hoc networks. Both IEEE 802.15.4 and BLE offer mesh networking solutions [6, 9, 25, 26].

3.2 IEEE 802.15.4

IEEE 802.15.4 is a widespread standard for low-rate wireless networks introduced in 2003 [27]. It builds the foundation for communication protocols such as Zigbee [28], WirelessHART [12] and Thread [13]. While it defines multiple frequency ranges and modulation schemes, the most prominent one is its 2.4 GHz physical layer using the O-QPSK modulation scheme [29] with direct-sequence spread spectrum (DSSS) [30]. This physical layer (from now on denoted as IEEE 802.15.4) with its robust modulation scheme offers a data rate of 250 kbit/s. Its direct-sequence spread spectrum modulation scheme uses a wider spectrum than necessary for the data rate, and thus is less affected by interference than other modulation schemes [30]. IEEE 802.15.4 splits the 2.4 GHz ISM band into 16 channels (11–26) that are 2 MHz wide and 5 MHz apart (cf. Figure 2). The IEEE 802.15.4 standard defines both the physical layer and several medium access control (MAC) layers.

3.2.1 IEEE 802.15.4 PHY

A physical layer IEEE 802.15.4 packet consists of a 5-byte synchronization header, a 1 byte packet length, and up to 127 bytes of payload (cf. Figure 3).

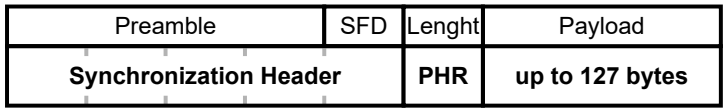


Figure 3: Packet format of an IEEE 802.15.4 packet. At a data rate of 250 kbit/s the transmission of each byte takes 32 µs.

The synchronization header contains 4 preamble bytes, all set to `0x00`, and a 1 byte start of frame delimiter (SFD) set to `0xA7`.

3.2.2 IEEE 802.15.4 MAC

The initial IEEE 802.15.4 standard [27] defines the IEEE 802.15.4 MAC, a single-channel MAC layer. This MAC layer defines two modes: beacon-enabled and non-beacon enabled [31]. The non-beacon enabled mode forms a non-time-synchronized, asynchronous network, and communication between nodes uses the unslotted CSMA/CA algorithm to determine whether the channel is free to send. In the beacon-enabled network, a *Personal Area Network coordinator (PAN coordinator)* employs a superframe structure with periodic beacons for time synchronization. Between two beacons, there is an active period for communication and an optional inactive (sleep) period. Within the active period, communication can take place in one of 16 equal length time slots, contention-based using CSMA/CA or contention-free in one of up to 7 guaranteed timeslots (GTS). Due to several limitations like the use of a single channel, inflexible numbers of guaranteed timeslots, unbounded latency, and low reliability [32, 33], this protocol is unsuitable for applications with strict quality of service (QoS) requirements. However, if the whole network uses a single channel, it allows the coexistence of several IEEE 802.15.4 networks in the same space.

The 2015 amendment of the IEEE 802.15.4 standard (IEEE 802.15.4e) [6] defines five new MAC modes, namely, *Blink Radio Frequency Identification (Blink RFID)*, *Asynchronous Multi-Channel Adaptation (AMCA)*, *Low-Latency Deterministic Networks (LLDN)*, *Deterministic and Synchronous Multi-Channel Extension (DSME)*, and *Time-Slotted Channel Hopping (TSCH)*. Both Blink RFID and AMCA are non-real-time MAC modes. The former is intended for applications identifying, locating, and tracking objects or personnel. It is widely used in contactless credit card transactions. The latter targets large deployments such as smart utility networks, infrastructure monitoring networks, and process control networks with a variance of channel quality and link asymmetry. It uses multiple channels and performs active scans, testing the link quality on all available channels to select the channel with the highest link quality for communication. AMCA is used in non-beacon-enabled, asynchronous networks [33]. The other three MAC modes are able to provide deterministic latency guarantees for time-critical applications. LLDN targets dense single-channel networks for factory automation requiring very low latency. In this mode, all nodes in the network must be directly associated with the PAN coordinator, forming a star topology. DSME and TSCH both target time-synchronized, beacon-enabled multi-hop multichannel mesh networks and

are able to form medium to large networks. DSME's main focuses are scalable, time-critical applications that require high reliability and low deterministic latencies, such as health monitoring, factory automation, smart metering, home automation, and smart buildings. DSME extends the initial beacon-enabled MAC mode. It defines a multi-superframe structure consisting of one or more IEEE 802.15.4 superframes. DSME accommodates a higher number of GTS slots and allows the use of multiple channels in the contention-free period. Moreover, the coordinator can limit the number of contention-based slots to the first superframe. When using multiple channels, DSME can either use the link quality indicator (LQI) to switch channels, or a network-wide hopping sequence [34]. TSCH targets application domains such as industrial automation and process control with time-critical applications that require high reliability. These applications include oil/refinery industries, water treatment, and process monitoring of food/chemical/pharmaceutical product, many applications which concern human and environmental safety. Moreover, TSCH is suitable for networks prone to interference from other wireless networks [34]. TSCH performs all communication in slots but does not follow a strict superframe structure as other modes. Within each slot, communication can either be contention-based using CSMA/CA, or contention-free using dedicated timeslots. As this thesis builds heavily on TSCH, we discuss TSCH in detail in Section 3.4.

3.3 Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE) [7] is a short-range and low-power communication protocol in the 2.4 GHz ISM band, mainly targeting single-hop communication between two devices. BLE was introduced as part of the Bluetooth 4.0 standard in 2010. While it shares a name with Bluetooth, it is a separate low-power communication protocol. BLE uses 40 2-MHz wide frequency channels (cf. Figure 2), and the Gaussian Frequency Shift Keying (GFSK) [35] modulation scheme. This modulation scheme is much simpler than the O-QPSK [29] modulation scheme IEEE 802.15.4 uses [27]. GFSK encodes only a single bit per symbol, compared to O-QPSK encoding two bits per symbol. Moreover, GFSK does not need to detect phase changes but rather only needs to differentiate two frequencies, one for encoding a one, and the other one for encoding a zero. Three of the 40 BLE channels are reserved for (primary) advertisements and broadcasts (see channels marked in blue in Figure 2), while the other 37 are reserved for connected communication and secondary advertisements. The initial BLE standard defines a physical layer (PHY) with a data rate of 1 Mbps (standard data rate) and a maximum transmittable advertisement payload of 37 bytes [36]. Bluetooth 5.0, introduces a 2 Mbps PHY, and two long-range coded PHYs with data rates of 125 kbps and 500 kbps. Further, Bluetooth 5.0 increases the maximum payload of BLE advertisement to 255 bytes [37].

3.3.1 PHY Packet Format

The physical layer packet format of BLE differs between the uncoded PHY *(1 Mbps and 2 Mbps)* and the coded PHY *(125 kbps and 500 kbps)* (cf. Figure 4). An uncoded PHY packet starts with a 1 or 2 byte preamble of alternating ones and zeros, for a data rate of 1 Mbps and 2 Mbps, respectively. It is followed by the 4-byte access address, identifying packets belonging to a connection.

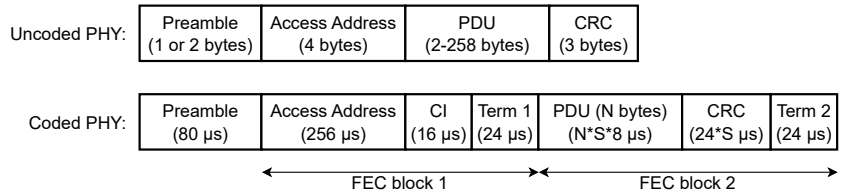


Figure 4: Packet format of a BLE packet for both the uncoded PHYs *(1 Mbps and 2 Mbps)* and the coded PHYs *(125 kbps and 500 kbps)*.

For packets not belonging to a connection and instead advertising services, so-called advertisement packets, the access address is fixed to `0x8E89BED6` (cf. Section 3.3.2). Afterward, the packet contains between 2 and 258 bytes payload (PDU) and a 3-byte cyclic redundancy check (CRC) code for error detection. A coded PHY packet generally contains the same components, however, with additional fields for forward error correction (see Figure 4). The preamble is uncoded, consisting of 10 repetitions of `0x3C` transmitted with a data rate of 1 Mbps. The two following forward error correction (FEC) blocks also use a symbol rate of 1 Mbps but with a coding rate of 1/8 for the first block and 1/2 or 1/8 for the second block. The first forward error correction (FEC) block thus is always transmitted with an effective data rate of 125 kbps containing the access address and the coding indicator (CI). The CI indicates the coding of the second FEC block, denoting whether it uses an effective data rate of 125 kbps or 500 kbps. The PDU and the CRC are then transmitted with the indicated coding afterward.

3.3.2 BLE Medium Access

BLE has two modes of operation: connected and non-connected. Both modes have independent roles and independent ways of coordinating the access to the wireless medium. In non-connected mode, each device accesses the medium without coordinating with any other device but tries to limit collisions by adding a random delay between transmissions. In connected mode, a central device determines a communication interval and channel hopping sequence which the peripheral devices follow for their communication. A device can communicate using both the connected and non-connected mode, which usually do not interfere due to using different channels (cf. blue and white channels in the BLE spectrum in Figure 2).

Advertising. In the non-connected mode, BLE devices disclose their presence and advertise their services to nearby devices. These services include, i.e., media control services, weather information (e.g., temperature data), or connectability (e.g., wireless headphones announcing their presence and that they are ready to connect).

The advertiser, often a low-power device like a wearable or an IoT sensing device (e.g., a smart thermometer), broadcasts an advertisement subsequently on all advertisement channels (37, 38, and 39) and repeats it pseudo-periodically at a fixed interval plus a random delay to avoid collisions. The receiver, such as a smartphone, scans for potential advertisements by listening to a specific channel during a scan window and changes the channels it listens to after the

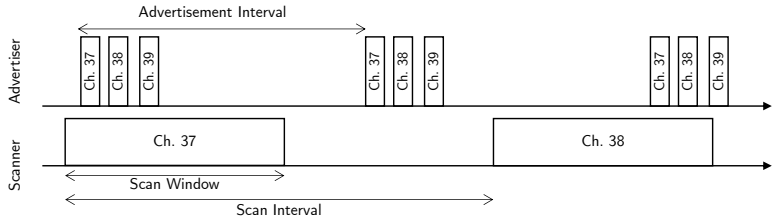


Figure 5: BLE Advertisements. The advertiser pseudo-periodically sends advertisement packets on all three advertising channels. The scanner periodically listens for advertisements.

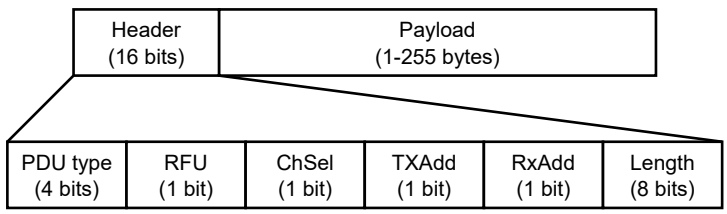


Figure 6: PDU structure of an advertising packet. The first byte of the header contains a 4-bit PDU type, 1 bit reserved for future use, a 1-bit flag whether the advertiser supports the BLE channel selection algorithm, two 1-bit flags whether the advertiser's and the target device's addresses are random or public, respectively. The second header byte contains the length of the subsequent advertising payload.

end of a scan interval (cf. Figure 5). By responding to an advertisement, a BLE device can initiate a connection.

Advertisement data. In non-connected mode, an advertiser sends data in the form of a so-called Protocol Data Unit (PDU). The PDU contains a list of Advertising Data (AD) structures. The advertising data can define, i.e., the device's name, a list of offered services (e.g., heart rate sensor or computer mouse), the device's address, the radio's transmit power, and manufacturer-specific data. Each AD has a Universally Unique Identifier (UUID), defined in the BLE standard [7]. For example, manufacturer-specific data has the UUID `0xFF` and the COVID-19 Exposure Notification AD has the UUID `0xFD6F` [38].

On the packet level, the BLE advertisement PDU consists of a 2-byte header, followed by one or multiple advertising data (AD) structures (cf. Figure 6). An AD structure consists of a 1-byte length identifier, the AD type (e.g., an identifier that a list of service UUIDs follows) and the AD data (e.g., the list of service UUIDs).

Connected mode. In connected mode, BLE devices exchange data related to the services announced in advertisements. For example, a keyboard can send key presses to a computer or a smartphone can turn on a BLE-enabled light bulb. The data exchange follows the schedule given by the central device [7].

3.3.3 BLE Mesh Networking

For communicating over more than a single hop, BLE offers a mesh networking extension called Bluetooth Mesh [25, 26]. Bluetooth Mesh uses modified advertisements and shorter intervals between advertisements compared to the non-connected mode to communicate data. When transmitting data, a sending node waits for a random backoff time before advertising the data on one of the three advertisement channels. Relay nodes continuously scan for data and switch between channels, keeping the radio on at all times. Upon receiving a packet, a relay node processes it if it's the packet's destination or advertises the data itself after a random backoff interval [39]. For packets destined for a low-power node not participating in relaying, a designated neighboring relay node can forward the packet to the intended recipient. Bluetooth Mesh employs managed flooding, a routing-free approach involving all relay nodes in the network. It encompasses various roles for different nodes in a network and an entire network stack [40].

Several studies have examined the performance of Bluetooth Mesh, highlighting its strengths and limitations [41, 42]. With a payload limit of 16 bytes, Bluetooth Mesh can achieve 99% reliability within a round-trip time of 200 ms.

3.4 Time-Slotted Channel Hopping (TSCH)

Time-Slotted Channel Hopping (TSCH) [6] is a MAC layer protocol for IEEE 802.15.4 targeting reliable communication in industrial low-power wireless networks. Its design builds on the Time-Synchronized Mesh Protocol (TSMP), which is part of the industry standards ISA100.11a [43] and WirelessHART [12]. It combines the medium access methods of Time-Division Multiple Access (TDMA) and Frequency-Division Multiple Access (FDMA) and adds a pseudo-random channel hopping mechanism to overcome narrowband interference. Communication in TSCH uses distinct time slots in one of the up to 16 frequency channels (see Figure 2), allowing up to 16 parallel communications in close vicinity to each other.

TSCH forms a time-synchronized mesh network with all participants in the network. For network formation, nodes listen for beacons (cf. Section 3.4.1) from nodes that are already part of the network. A PAN coordinator starts the process of sending these beacons. When receiving a beacon, a node joins the network and takes part in broadcasting beacons to further reach nodes that might want to join the network. For keeping the network synchronized, TSCH builds a time-source tree from the PAN coordinator to each node in the network. For communication in a TSCH network, a scheduler assigns slots and specific roles to each node. We discuss this in Section 3.4.3.

3.4.1 TSCH Slots

TSCH uses time slots that have a standardized length of 10 ms. Each time slot has a specific role. It is either dedicated, shared, or empty. In certain implementations, like the one in the Contiki-NG [44] operating system, a fourth role, beaconing (only), is available. In the case of a beaconing slot, nodes can send Enhanced Beacons (EB) containing control information for network- and time-synchronization. These beacons are essential for joining the network

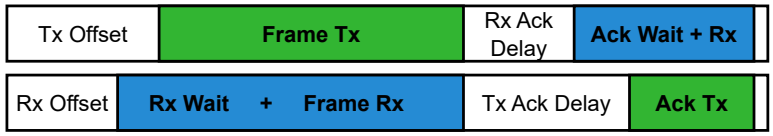
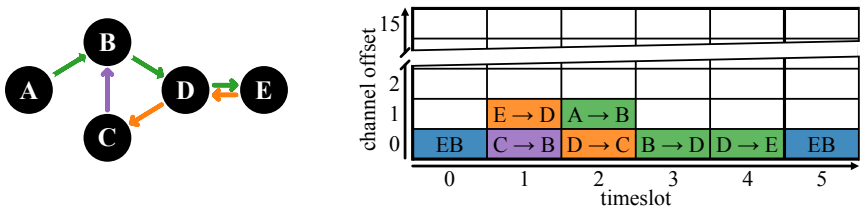


Figure 7: Simplified TSCH timeslot timing. We omit the optional `CCAOffset` and the `CCA`, which happen during `TxOffset`, if enabled. Please note: the illustrated timing is not to scale.



(a) Sample network. **(b) Sample TSCH schedule.**

Figure 8: Example of a TSCH schedule. The schedule contains three flows (A→E, E→C, and C→B) that use dedicated slots for communication and one shared beacon slot. The schedule has a slotframe length of 5 slots, therefore, slot 5 is a repetition of slot 0.

and maintaining the network's connections and time synchronization. In a dedicated slot, a node either receives or transmits data. The transmitting node can either communicate with a specific node (unicast) or with all nodes in range (broadcast). In the case of unicast transmissions, the receiver acknowledges the reception within the same slot (cf. Figure 7). A shared slot allows any communication of the above, beaconing, unicast, and broadcast messages. As all possible senders share the same slot, these slots use the CSMA/CA back-off algorithm to limit collisions. Nodes that do not have anything to transmit in this shared slot listen for incoming packets. If a slot has none of the above roles, it is empty. In empty slots, the node's radio remains off to save energy.

TSCH groups slots in slotframes. A slotframe has a defined length and contains at least one active slot. After passing all slots in the slotframe, it repeats. Contrary to the legacy IEEE 802.15.4 MAC and DSME, TSCH supports multiple slotframes with different lengths simultaneously. Each of these slotframes has a priority. If two or more slotframes have an active slot scheduled for the same time, TSCH executes the slot belonging to the slotframe with the highest priority. For example, slotframes with beaconing slots are usually given the highest priority, as they are necessary for maintaining the network. All other communication is distributed across the other slotframes. As slotframes can have different sizes, not always the same slots overlap. All slotframes together form a TSCH schedule. Figure 8 shows an example of a TSCH schedule with a single slotframe.

3.4.2 Channel Hopping

Next to the availability of up to 16 frequency slots (FDMA) that allow communication in up to 16 TSCH slots in parallel, TSCH uses the concept of channel hopping to counteract narrowband interference. In channel hopping, all active channels cycle through a pseudo-random hopping sequence. Through this hopping sequence, every channel uses a different physical frequency at subsequent time slots. The hopping sequence has to include at least as many frequencies as parallelly active channels, with a maximum of 16 frequencies. Thus, for every nth slot (n = number of frequencies), a channel uses the same physical frequency for transmission. For example, in a common hopping sequence of four channels, TSCH cycles through the channels 15, 25, 26, and 20. These channels use frequencies that do not overlap with frequencies of the three most common non-overlapping WiFi channels (1, 6, and 11) (see Figure 2).

3.4.3 TSCH Scheduling

The IEEE 802.15.4 standard defines the TSCH MAC layer and the structure of a schedule. However, it does not define how such a schedule is to be created. Therefore, many bodies of work (cf. [45–47]) study this problem, creating different ways of scheduling TSCH networks. Scheduling communication efficiently is essential to ensure reliable communication that is capable of meeting deadlines and saving energy. The scheduling problem of allocating time slots and channels for communication partners meeting an application's requirements is an NP-hard problem, as Saifullah et al. [10] show. Below, we discuss the core concepts of scheduling traffic in TSCH. In Section 4.1 we take a more in-depth look at scheduling solutions related to our research.

In TSCH, schedulers are commonly grouped into three classes: centralized, decentralized (collaborative), and autonomous. Centralized schedulers use global knowledge to schedule communication, and commonly combine scheduling and routing. They usually target static topologies and often perform the scheduling separated from the wireless network at the edge, or in the cloud. Decentralized and autonomous schedulers have no global knowledge of the network topology to perform their slot allocation. Decentralized schedulers collaborate with their neighbors to agree on a schedule and autonomous schedulers schedule communication independent of any knowledge, even independent of the nodes' neighborhood. We briefly discuss each class of schedulers in the following paragraphs.

Centralized Scheduling. The first and oldest category of TSCH schedulers are centralized ones, targeting static deployments, often in industrial settings. Centralized schedulers combine the routing of traffic in a mesh network with the actual scheduling of communication slots. We define a route as the order of transmissions (hops) that are necessary for end-to-end communication between a specific sensor-actuator pair. We discuss routing in mesh networks in more detail in Section 3.4.4.

Centralized schedulers operate on global knowledge. Therefore, they use information about the network, especially about the quality of the wireless links. We commonly measure the quality of a link using the link's packet reception rate, or its inverse, the expected transmission count (ETX) [48]. The latter is the average number of transmissions necessary to successfully transmit

a packet over the specific link. With this global knowledge of the network topology, the centralized scheduler can combine routing and scheduling to form a schedule accommodating the communication requirements of each node. Modern centralized schedulers commonly add retransmission slots for lossy links with an expected number of transmissions larger than one ($ETX > 1$). After computing the schedule, a central node disseminates it through the network to all nodes.

Distributed Scheduling. Distributed or collaborative schedulers operate on local knowledge. These schedulers negotiate communication slots between neighboring nodes during runtime. Moreover, they perform only the scheduling between neighbors and usually separate scheduling the medium access from routing.

Autonomous Scheduling. The last class of TSCH scheduling algorithms are autonomous schedulers. While the other two scheduler classes either use global knowledge or local knowledge to make a scheduling decision, autonomous scheduling algorithms commonly operate without knowing the network topology at all. Autonomous schedulers neither centrally plan communications or allocate resources, nor do they negotiate resources between neighbors in a distributed fashion. Instead, they use a higher layer metric like the routing distance to the network root or a hash function to schedule slots. Thus, they do not specifically schedule links in the network, but provide a framework for nodes to select the intended slots for the situation the nodes are in during operation. Similar to distributed schedulers, autonomous ones usually leave routing to a higher layer in the network stack.

3.4.4 TSCH Network Stack

While TSCH commonly concerns the medium access in IEEE 802.15.4 networks, multiple network stacks on top of TSCH exist. These network stacks define the routing in TSCH networks, as well as higher layer protocols.

Routing in TSCH. Routing is the selection of a path that data takes from a sender to a receiver. Routing determines the order of links over which data travels through the network, whereas scheduling concerns the allocation of the individual links that are available for routing. The two major approaches to routing in TSCH are tree-based routing and shortest-path routing. In tree-based routing, packets travel a tree upwards to a common ancestor of sender and receiver and then downwards the tree to the receiver. Distributed and autonomous TSCH schedulers commonly use this routing approach. The other approach is shortest-path routing, which requires global knowledge of the network topology and uses algorithms like Dijkstra's shortest path algorithm [49] or the A* algorithm [50]. Shortest-path algorithms are a good choice for centralized routing, and thus commonly used in combination with centralized schedulers. However, shortest-path routing is not the only approach used in centrally scheduled TSCH networks. Other approaches perform, i.a., an asymmetric routing approach [51], applying specific routing strategies for different communications in a network, or a conflict-aware real-time routing approach [52] that takes path conflicts originating from scheduling decisions into account when making routing decisions.

In the case of tree-routing, the *Routing Protocol for Low-Power and Lossy Networks (RPL)* [53] is widely used in low-power wireless networks and in TSCH. RPL is a best-effort routing protocol for low-power wireless networks susceptible to packet loss. For generating the routing path, RPL uses a directed acyclic graph (DAG). Packets are forwarded (upwards routed) hop-by-hop from parent to parent until reaching the root of the graph. From there, the packets are routed downwards hop-wise until they reach their destination. That means all routing in RPL is a combination of upwards-routing to the tree's root and downwards routing to the packet's destination. A variation to this approach is the storing mode in which each node saves information about its children. This variation routes traffic only upwards until meeting a common ancestor of the packet's source and destination. The path RPL uses for routing is based on the shortest distance to the tree's root using a metric like ETX.

6TiSCH stack. A common network stack for TSCH networks, especially those using RPL for routing, is 6TiSCH, a standardized set of protocols to enable IP-addressable low-power wireless devices [54, 55]. The Internet Engineering Task Force's (IETF) 6TiSCH bridges TSCH networks with 6LoWPAN networks. The network stack consists of IETF standards including UDP, IPv6, RPL, and 6LowPAN and a 6TiSCH operation sublayer (6top) [55] connecting the network protocols with TSCH. The 6top protocol defines how to exchange scheduling requests and statistics between the layers. 6TiSCH is implemented in several embedded operating systems such as Contiki-NG [44] and OpenWSN [56], and is the standard network layer for most distributed and autonomous TSCH schedulers.

3.5 Concurrent Transmissions

Two or more devices transmitting signals at the same time on the same frequency lead to interference. This interference commonly destructs the signals and makes it either impossible to receive any data or makes the received data unintelligible. However, there are circumstances in which interference is not a hindrance or potentially even a benefit for successful communication.

In Concurrent Transmissions (CT), or Synchronous Transmissions, multiple nodes transmit data simultaneously and synchronously. In two cases, a receiver can receive data transmitted synchronously. The receiver can either, with a high probability, extract the strongest of the received signals due to the Capture Effect [57], or receive the data due to non-destructive interference [9]. In both cases, the transmissions have to be tightly synchronized to allow the decoding of concurrent transmissions. Within the past decade, multiple solutions have shown the feasibility of network-wide flooding with Concurrent Transmissions for both IEEE 802.15.4 O-QPSK with DSSS [9, 58, 59] and BLE [60]. Many CT protocols build upon Glossy [9], which created the foundation for synchronous transmissions in low-power wireless IEEE 802.15.4 networks and offers network-wide flooding and high-accuracy synchronization.

3.5.1 Capture Effect

The Capture Effect, which was initially observed for FM receivers [57] states that a receiver can extract one signal from many colliding ones if its signal strength is significantly higher (power capture) than the combined signal strength of all other signals. For example, to utilize the Capture Effect in IEEE 802.15.4 O-QPSK with DSSS, the signal has to have at least twice the power (+3 decibel (dB)) as the combined other signals [58]. Moreover, the stronger signal must not arrive later than 160 µs after the first signal [59], the duration of the synchronization header in IEEE 802.15.4 (cf. Figure 3).

Two well-known examples of protocols using the Capture Effect are Chaos [59] and Crystal [61]. Chaos is a protocol for all-to-all data sharing in low-power wireless networks. Nodes transmit (potentially different) data concurrently and thus speed up network-wide data sharing. Crystal builds a network stack for sporadic data collection, with many nodes generating data at the same time.

3.5.2 Non-Destructive Concurrent Transmissions

In the other case of non-destructive Concurrent Transmissions, multiple nodes transmit the same data. For this case, nodes have to be even more tightly synchronized than for the Capture Effect. If the time offset between all transmissions stays below 0.5 µs, no signal strength delta is necessary to successfully receive the data. Ferrari et al. [9] were the first to study the use of these non-destructive concurrent transmissions in IEEE 802.15.4. Later works [62–64] study it further to better understand why the overlapping signals are indeed non-destructive.

Within the past 13 years, many bodies of work designed protocols utilizing tightly synchronized non-destructive Concurrent Transmissions in IEEE 802.15.4 networks to efficiently and reliably share data. In 2011, Glossy [9] created the foundation for synchronous transmissions in low-power wireless IEEE 802.15.4 networks and uses synchronized flooding to disseminate data from one node to all others in a network. LWB [65] builds on Glossy floods by adding the capability of scheduling individual network floods for data collection. While LWB is not a real-time protocol, Blink [66] performs deadline-based real-time communication, achieving high reliability on top of it. The protocols above perform network-wide flooding and thus involve the whole network for communicating data. Works like WSNShape/Sparkle [67] and CXFS [68] deviate from that approach and limit the number of forwarders performing directional flooding or essentially routing. CXFS, for example, introduces a forwarder selection strategy in networks using concurrent transmissions. LaneFlood [69] builds upon this forwarder selection and extends it to even allow the flooding of IPv6 traffic along a routed lane.

3.6 Time Synchronization

Up to this section, we discussed networking in low-power wireless networks. As this thesis also targets orthogonal applications like time-synchronized evaluation of low-power wireless networks and new directions like wireless environment detection, we discuss the relevant background for these in the following sections.

For a precise evaluation of distributed protocols, we require timestamping across multiple devices. To facilitate the evaluation of locally timestamped events, and set them into perspective to events on other devices, the timestamping system should have a common timescale. Moreover, when devices communicate with each other, they often require a timed operation. For example, in low-power wireless communication, devices need to be synchronized when turning on their radios to be awake when data is scheduled to be sent [6]. Other wireless applications like ranging and positioning also require a high degree of time synchronization [70]. If, for example, a device needs to compute its distance from another device using a signal's time-of-flight, the clocks on both devices need to timestamp signals and thus need to be synchronized [70]. Therefore, to ensure efficient (wireless) communication and evaluation systems, we require time synchronization.

For keeping time, computers commonly use crystal oscillators to drive their local clocks [71]. However, due to material imperfections and environmental circumstances, such as temperature, humidity, and air pressure, oscillators are not perfect and their frequency varies within a single oscillator and between different oscillators and devices [71, 72]. Thus, in a distributed setting with more than one device, systems to time-synchronize the local clocks of devices are necessary. Depending on the use case, it might be sufficient to synchronize all clocks of a distributed system to each other (*internal synchronization*), e.g., a local (wireless) network. In other cases, it is necessary to synchronize all clocks with a global timescale (Universal Coordinated Time (UTC)) (*external synchronization*), e.g., networks distributed over multiple locations or positioning systems. An internal synchronization system has the goal to keep clocks *precise*, which means that it is sufficient to keep the deviation between the clocks of any two devices within a specified bound [71]. In contrast, an external synchronization system aims to have *accurate* clocks that do not deviate more than a specific amount from UTC time [71].

3.6.1 Accuracy-Based Time-Synchronization

Accuracy-based time-synchronization uses an external time source to synchronize to a global timescale. A common approach is using the time signal of a highly accurate atomic clock. To use the signal directly, one can use Global Navigation Satellite Systems (GNSS), like the Global Positioning System (GPS) [73]. GNSS satellites carry each an atomic clock, and thus a GNSS receiver can generate an accurate time signal (error $<$50 ns [71]) for a connected system. While systems outdoor or close to a window can be equipped with GNSS receivers, GNSS is an impractical solution for most devices, as the receiver requires a direct line of sight to the GNSS satellites. For these scenarios, a common approach is the use of timeservers. The most notable example is the Network Time Protocol (NTP) [74]. One or more servers are equipped with a reference clock, such as a GNSS receiver or an atomic clock. Other servers can contact these servers to update their clock and be themselves timeservers for other servers, building a hierarchical structure. This system is the one synchronizing the internet with an accuracy of $\leq$50 ms [71, 75].

3.6.2 Precision-Based Time-Synchronization

In networks, where contacting a timeserver is not practicable or networks that do not require global time-synchronization, precision-based clock synchronization systems are available. Especially wireless networks, like IoT networks, with resource-constrained devices require more precise clock synchronization systems than NTP. A notable approach is the Reference Broadcasting System (RBS) [71, 76], a system for time-synchronizing nodes in a single-hop wireless network. In RBS, one device (node) broadcasts a time signal to all nodes in its vicinity. Due to the signal propagation speed, all nodes receive the signal at approximately the same time, or with a negligible offset, and can synchronize their clocks. As all devices listen to the same time source, the processing time before sending the time stamp is not critical as the clocks are not synchronized to a global timescale, thus the critical path in RBS solely consists of the time on air and the precise timestamping of the reception at the receivers. Neighboring nodes can exchange their recorded delivery times and use linear regression to convert each other's local offsets, making it unnecessary to update their local clocks.

For multi-hop wireless networks, RBS is not a sufficient solution, as no single node exists that can reach all other nodes of the network. In time-slotted networks, like TSCH networks, with predefined communication times within a single slot, nodes can synchronize their clocks by observing the time offset between the expected reception of a signal and the actual reception of the signal. Networks can build a time-synchronization tree to keep all nodes synchronized and connected (cf. Section 3.4). As clock drift is to be expected, the synchronization in TSCH is possible on any message a node receives from its time source [6].

3.7 Evaluation of Low-Power Wireless Networks

Research on low-power wireless networks is not limited to the definition of protocols, but a crucial aspect is also to test, evaluate, and demonstrate the protocols' performance. For testing and evaluating protocols, target deployments, e.g., deployed IIoT devices in a factory, are rarely accessible. Therefore, simulators or dedicated testbeds are the common solutions for evaluating low-power wireless protocols and networks.

3.7.1 Simulators

Simulators simulate or even emulate different aspects of low-power devices or low-power communication. The network simulator ns-3 [77] and the INET Framework [78] for OMNeT++ [79] simulate the network stack to test and evaluate various network protocols. They support several physical and medium access layers, including low-power wireless technologies such as IEEE 802.15.4. OpenSim [56], Cooja [80], and Renode [81] take a different approach. Instead of focusing solely on the network, they emulate specific IoT devices and offer propagation models for simulating wireless communication. Renode simulates both IEEE 802.15.4 and BLE PHYs, whereas Cooja only supports the IEEE 802.15.4 PHY but offers more sophisticated wireless propagation models. Instead of emulating entire devices or simulating the full network stack, the 6TiSCH simulator [82] and TSCH-Sim [83] are discrete event simulators implementing

TSCH and 6TiSCH. These simulators work on a higher abstraction layer and allow the prototyping of TSCH schedulers. TSCH-Sim also has the goal to simulate large TSCH networks.

3.7.2 Testbeds

While simulation allows certain insights into the execution of protocols, they cannot accurately replicate all aspects of real hardware and real wireless environments [84,85]. Therefore, the research community proposed and operates several testbeds [84–98] to evaluate and benchmark low-power wireless protocols and IoT systems. Testbeds colocate IoT devices with observer infrastructure to program IoT devices and gain insights into the execution locally on a device, and into the interaction with other devices in the case of testing or evaluating a network protocol. Most testbeds offer insights through recording serial messages, while others can additionally record traces of GPIO pins [84–86, 93] or through the JTAG interface [85, 86, 94–96]. Within the past two decades, the research community proposed and built several testbeds, starting with MoteLab [99]. The community currently operates several testbeds for testing, debugging, evaluating, and benchmarking low-power IoT protocols, with some of them being openly accessible [87, 98]. A long-time used testbed in offices of a university building was Flocklab [84], and its update Flocklab 2 [85, 86]. Publicly accessible testbeds include FIT IoTLab [98] and D-Cube [87]. FIT IoTLab offers several testbeds with a wide range of IoT platforms, with both deployments at fixed locations and moving robots. D-Cube is a testbed with a different focus. Instead of offering a testbed for testing and evaluation, D-Cube is intended for benchmarking IoT protocols in a wireless environment with controllable degrees of interference.

3.8 Embedded Intelligence and Tiny Machine Learning

So far, we introduced protocols and approaches concerning networking and closely related aspects. One chapter of this thesis combines networking aspects with Artificial Intelligence (AI). Therefore, this section introduces background information on Machine Learning and Embedded AI.

Within the past decades, Artificial Intelligence (AI) and its subfield Machine Learning (ML) experienced major development, making intelligent components an essential part of many modern software systems. By the end of the last century, AI systems like the chess computer Deep Blue [100] have shown that intelligent machines can outperform humans in specific domains. However, the major breakthrough of AI, or more specifically ML, followed in the past decade with major advances in deep learning. Beginning with image recognition in Computer Vision [101, 102], followed by beating humans at the board game Go [103] and major advances in Natural Language Processing (NLP) [104–106], deep learning showed its potential and capability of outperforming traditional methods. For training ML models, various approaches exist, including supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning.

Deep Learning, a subfield of Machine Learning, is a method that uses neural networks with multiple consecutive layers. Modern neural networks can have over a hundred layers with hundreds of billions of parameters (e.g.,

weights), with the currently largest model (GPT-4) estimated to have 1.8 trillion parameters across 120 layers [107]. While GPT-4 is an extreme case, other widely used NLP models like BERT [105] and ChatGPT (gpt-3.5-turbo) [108] have up to 330 million and 20 billion parameters, respectively. In computer vision, the numbers of parameters are lower, with, for example, ResNet101 containing 44.7 million parameters [102, 109].

Deep Learning models require all their parameters during inference, and thus, only powerful servers are capable of executing large models. These servers have to load hundreds of megabytes to terabytes of weights, commonly stored as 32-bit floating-point numbers, and perform up to trillions of operations for a single classification or prediction of the network.

3.8.1 Tiny Machine Learning

This development of increasingly larger models stands in stark contrast to the growing amount and need for intelligent, resource-constrained devices. For example, detecting sleep patterns on a smartwatch, tracking activities with a fitness tracker, performing face identification on smartphones, or waking up speech assistants with a wake word require machine learning capabilities on end devices with computing resources several magnitudes smaller than servers or data centers. While it is thinkable to upload data and perform the machine learning tasks in the cloud, this is only possible for devices with a steady internet connection. Moreover, it adds latency, requires significant amounts of energy for communication, and cannot guarantee to preserve privacy. For example, sensitive data like one's conversations or health data should remain local on the end device. While standard deep learning models are too large and too compute-heavy to fit and be executed on end devices, several approaches reduce their footprint and enable their execution on resource-constrained devices. For example, MobileNets [110] and FOMO [111] enable fast and reliable image classification and object detection on smartphones or even embedded devices like certain Arduinos [112]. One method for shrinking model sizes and increasing execution speed is quantization [113]. Quantization transforms the network's parameters from a floating-point representation to an integer representation, often using 8 bits per parameter. Such a model takes less space and does not require floating-point operations during inference. Instead, it can stick to much faster integer operations, saving time and energy [113]. Other approaches use binary neural networks [114], sparse-matrix multiplications [115], or splitting the network to offload heavy computation to edge devices or the cloud [116,117].

4 Related Work

In this section, we present a selection of work representing the current state of research in the field of this thesis. We consider literature in the areas of (a) TSCH scheduling and routing, (b) mesh networking in BLE, (c) infrastructure for in-depth testing and evaluating low-power wireless networks, and (d) methods for wireless fingerprinting and environment detection.

4.1 Scheduling TSCH

Scheduling in TSCH is a vast field with over a hundred different scheduling solutions. With this vast number of papers, we limit our discussion of TSCH schedulers to a selection of notable papers. For a more complete picture, [46] and [47] review TSCH schedulers with a focus on IIoT and traffic awareness, respectively. As in the background on TSCH (cf. Section 3.4), we split our discussion of schedulers along the categories of centralized, distributed, and autonomous schedulers, followed by the related topic of network softwarization. Contrary to the background in which we introduce the core concepts, here we discuss related ideas and similar directions to our work.

4.1.1 Centralized Scheduling

Many bodies of work propose centralized schedulers for TSCH [46, 47] and WirelessHART [10, 16]. Scheduling algorithms in this category target different aspects, such as energy efficiency [118], throughput maximization [15, 16], schedulability [10], and lossy links [119–122].

One of the first schedulers for TSCH is TASA [11], a traffic-aware protocol. It creates a schedule based on the network topology and the traffic load; however, it makes the same assumption as many TSCH and WirelessHART [12] schedulers to operate on interference-free channels and thus does not include retransmissions in its schedule. In a subsequent work to TASA, Palattella et al. [123] propose a mathematical analysis and method of computing the minimum number of active slots within a network. Ojo et al. [15] propose a graph theoretical approach to maximize throughput in a centrally scheduled TSCH network. These works on TSCH and WirelessHART, including the C-LLF [10] scheduling algorithm, focus on the highest possible schedulability for large amounts of deadline-aware communications. Gunatilaka et al. [16] study the use of a channel for multiple simultaneous communications in the same timeslot if the communication partners are physically far enough apart to increase schedulability even further. Similarly, Ojo et al. [118] propose EES, an energy-efficient scheduling model avoiding interference between concurrent communications.

Darbandi et al. [124] propose a path collision-aware least-laxity first (PC-LLF) scheduling algorithm, which Rugamba et al. [125] implement as part of a centralized scheduler. Moreover, they describe a method of distributing such a schedule to nodes in a centrally scheduled network. Instead of optimizing throughput and focusing on networks with joined routing and scheduling, CLS [126] and QSS [127] are centralized schedulers for networks using RPL for routing and focus on reducing control message overhead when scheduling and rescheduling links. T^2AS [128] is a traffic-aware and topology-aware scheduler that sequentially creates a schedule prioritizing longer flows in a data collection application to reduce latency. Portaluri et al. [129] build a scheduler that adds fairness in resource allocation, rebalancing the requested communication resources of end devices. Gaitán et al. [130] propose a scheduler for networks with multiple gateways that uses unsupervised learning methods to reduce the number of overlaps of flows.

The above schedulers are capable of creating effective schedules with high communication throughput. However, many of them can only do so because

they assume interference-free channels and perfect links, and thus do not include retransmissions. A protocol deviating from this assumption is SchedEx [121], a scheduler extension that calculates and adds a necessary number of retransmissions for each link of the network. AMUS [119] also adds retransmission slots to allow the use of lossy links. AMUS adds backup retransmission slots for vulnerable links in otherwise empty slots of the schedule. Khorov et al. [122] create a slotframe structure including retry segments and shared segments and study how to identify the optimal number of shared cells. Gaillard et al. [120] propose another method of retransmissions—slot-based retransmissions—as an extension to TASA. Slot-based retransmissions repeat the same slot multiple times to increase reliability. Such a schedule is also applicable for end-to-end communication flows in non-collection-based networks. Schedules with retransmissions shift the focus from pure schedulability towards reliability, making them capable of withstanding local narrowband interference and usable in networks susceptible to interference.

While slot-based retransmissions add resilience towards interference and allow the use of lossy links, they increase end-to-end latency and cannot react to strongly fluctuating interference levels. To overcome these limitations, we can deviate from focusing on individual links and focus on end-to-end communication flows instead, and allocate resources for the entire flow. The first two protocols taking this approach are the flow-centric policy (FCP) [131] for WirelessHART and our first paper's flow-based retransmission scheme (cf. Section 6.1) for TSCH. Both activate nodes and allocate retransmission slots in a way that retransmissions are available wherever needed along the path. This deviation from link-based to flow-based communication allocates resources for flows and not for individual links. Thus, we can achieve stable and reliable communication even in the presence of unforeseen interference.

4.1.2 Distributed Scheduling

Tinka et al. [132] present two distributed scheduling algorithms for networks constantly changing due to mobility. They present an algorithm for continuously announcing a node's presence and another one distributing scheduling information for quickly forming a network-wide schedule. Since the introduction of 6TiSCH [54], the design of distributed schedulers has been heavily influenced by it and its minimal scheduling algorithm [133]. This scheduler uses a single cell shared over all nodes. More sophisticated scheduling functions include the Minimum Scheduling Function (MSF) [134], the Low-Latency Scheduling Function (LLSF) [135], and the Low-latency Distributed Scheduling Function (LDSF) [136]. MSF has shared cells, autonomous cells and negotiated cells to schedule different kinds of traffic. LLSF schedules cells for multi-hop communication closer together to allow forwarding traffic immediately after reception. LDSF achieves lower latency than other scheduling functions by splitting slotframes into shorter blocks and introducing retransmission options in consecutive blocks, focusing on improving latency in distributed TSCH. The field of distributed TSCH scheduling also covers a range of traffic-aware scheduling solutions. DeTAS [137], a distributed version of TASA [11] adds traffic-aware scheduling to build collision-free schedules along a routing tree. Domingo-Prieto et al. [138] propose a PID-based scheduling solution that adds

or removes cells from a schedule dependent on traffic demand and network state to counteract network changes and allow non-periodic and bursty traffic. Palattella et al. [139] propose an algorithm matching the number of cells between nodes to the actual demand. Similarly, OST [140], allocates slots for each directional link and adapts its period according to the amount of traffic. Many of these schedulers focus on low-latency and reliable communication instead of schedulability as many centralized TSCH schedulers do. Jung et al. [141] take a different approach and propose a solution balancing latency, degree of activity of each node, and collision avoidance to achieve a long network lifetime and high quality of service.

Lately, distributed TSCH schedulers started using machine learning, especially reinforcement learning. RL-TSCH [142] and RL-SF [143] use reinforcement learning for building optimized traffic-aware schedules with a focus on reliability and energy efficiency or bandwidth optimization, respectively. Another recent approach is the use of game theory [144] to build optimal schedules utilizing selfish node behavior.

4.1.3 Autonomous Scheduling

A prominent example of an autonomous scheduler is Orchestra [8]. Orchestra is a best-effort autonomous scheduler that uses sender-based or receiver-based communication slots reserved for certain groups of nodes. Orchestra requires a hash function to determine which nodes can send in which slot. To enable the efficient use of TSCH in scenarios with high-rate unpredictable traffic, Elsts et al. [145] propose a hybrid approach with shared and dedicated slots and reappropriates slots to different nodes depending on the traffic demands. DiGS [146] adds autonomous scheduling to otherwise central WirelessHART networks, adding robustness through path diversity introduced by devices selecting their own routing path. Oh et al. [147] propose Escalator, focusing on minimizing transmission delays in convergecast scenarios by allocating consecutive time slots along a packet's path. Moreover, contrary to Orchestra, Escalator uses multiple channels. Alice [148] deviates from the node-based slot allocation of Orchestra and uses link-based slots instead. Moreover, Alice uses multiple channels with link-based channel offsets instead of a single channel, and changes cell allocation of all unicast slots at every slotframe. The initial autonomous schedulers can achieve high reliability but are unaware of traffic flows and do not achieve the low latency of other scheduling approaches. However, TESLA [149] proposes a traffic-aware cell scheduling method to add adaptability to different traffic loads. TESLA adds and removes slots dependent on the traffic load of neighboring nodes. Jung et al. [150] propose a parameterized slot scheduler that adapts to the traffic load of nodes. The scheduler tries to find a trade-off between energy efficiency, reliability, and latency by using shared slots for nodes transmitting to a joint receiver if collisions are unlikely. Rekik et al. [151] present e-TSCH-Orch, an enhancement to Orchestra avoiding congestion by adaptively adding transmission slots for a node depending on the number of packets in the node's queue. ATRIA [152,153], another traffic-aware scheduling method, allocates slots according to the traffic demand of each link. To improve network performance, ATRIA includes a method for selecting the optimal slotframe length and uses subslotframes to

avoid slot conflicts. OSCAR [154] optimizes Orchestra for convergecast traffic by assigning different amounts of cells to nodes depending on the distance of the node to the network root. Kim et al. [155] propose A^3, a traffic load based and adaptive slot allocation algorithm for autonomous schedulers targeting scenarios with dynamic traffic or heavy traffic load. A^3 dynamically adjusts the number of slots per slotframe.

Layered [156] introduces flow-based scheduling to autonomous TSCH schedulers. It allocates end-to-end slots to a flow to reduce latency and apply spatial channel reuse for a more efficient use of the wireless spectrum. In a subsequent work, Urke et al. [157] extend Layered to support end-to-end communication flows for sensor-actuator traffic, deviating from the convergecast scenario of the initial version and many other autonomous schedulers.

4.1.4 Network Softwarization

Many centralized schedulers for WirelessHART and TSCH discussed above perform centralized routing and scheduling at the edge, separating the control plane from the data plane, and thus achieve one part of network softwarization. However, to upload a schedule or monitor the network and perform rescheduling, additional solutions bridging these two are necessary. In this context of centrally scheduled 6TiSCH networks, Thubert et al. [158] discuss the challenges of Software Defined Networking (SDN). Exarchakos et al. propose plexi [159], a framework and a web interface for the task of reconfiguring and rescheduling TSCH communication. In the general field of software defined Wireless Sensor Networks, Galluccio et al. [160] and Baddeley et al. [161] present Software Defined Networking solutions for network monitoring and reconfiguration. Galluccio et al. [160] propose SDN-WISE, a stateful solution aiming to reduce communication overhead with the SDN network controller and make sensor nodes programmable as finite state machines. Baddeley et al. [161] introduce µSDN, a lightweight solution to gain global knowledge, network (re)configurability, and virtualization in IEEE 802.15.4 networks.

4.2 Overcoming Interference in TSCH

TSCH uses channel hopping to overcome narrowband interference. However, in the presence of local wideband interference or local interference on multiple channels, TSCH reaches its limits. To nonetheless overcome these kinds of interference, channel blacklisting, multi-path routing, and opportunistic routing are possible ways. Channel blacklisting has the goal to eliminate channels with high amounts of interference to increase the packet reception rate and require fewer transmissions. Adaptive TSCH (A-TSCH) [162], and Enhanced TSCH (ETSCH) [163] propose modifications of TSCH to estimate the channel quality and distribute blacklist information as part of Enhanced Beacons (EB) to build a global blacklist. MABO-TSCH [164] performs local blacklist negotiation for the communication of individual links. Elsts et al. [165] study adaptive channel selection methods for TSCH to reduce the number of retransmissions. Instead of limiting the number of channels, multi-path routing and opportunistic routing aim to avoid interference through routing decisions. In the literature, multi-path routing mainly appears for RPL-based distributed TSCH schedulers.

Instead of using the standard single-path tree-routing of RPL, multi-path routing uses triangular-based redundancy patterns [166], overhearing in RPL networks [167], scheduling a second path at each node [168], or packet replication algorithms [169]. Gaillard et al. [170] propose Kausa, which does not use redundancy patterns, but instead builds a schedule with different paths to balance the traffic load in the entire network.

4.2.1 Opportunistic Routing

Contrary to multi-path routing, opportunistic routing does not use or maintain multiple independent paths for routing. Instead, it addresses more than one potential forwarder for a packet [171–173] to use the best suitable path for the current state of the network. Opportunistic routing is not TSCH-specific, and generally has the goal of improving the throughput and reliability of multi-hop wireless networks. For example, in ExOR [171] each packet is addressed to a set of potential forwarding nodes, prioritized by routing progress. Based on their priority, each node in the forwarder set is assigned a time slot for acknowledging the packet. It only utilizes this time slot if it did not overhear the acknowledgement of the packet in a previous time slot. Thus, only one node acknowledges the reception of the packet and forwards it. Later works such as ORW [174] select the forwarder set based on the expected wait time. The first receiver of the forwarding set that successfully receives the packet and provides routing progress acknowledges and forwards the packet. ORW introduces opportunistic routing to duty-cycled, low-power wireless networking. ORPL [175] combines the ideas of ORW and ExOR. ORR [176] follows the same approach as ORW, but chooses the set of forwarders based on their residual energy, wait times and the amount of redundant packets. Hawbani et al. [177] introduce a candidate zone to limit the number of potential forwarders and selects the forwarder based on local distance distributions to reduce waiting times when forwarding. FLORA [178] takes both location characteristics regarding the destination and energy levels of nodes into account to select the best forwarder. SDORP [179] proposes an SDN-based opportunistic routing approach improving on ORW and ORR by selecting the forwarder set based on residual energy, transmission distance, and number of hops to the network's sink.

The protocols above build their protocols for asynchronous MAC layers, however, TSCH is a synchronous MAC layer protocol and thus the protocols are not one-to-one applicable for TSCH networks. Huynh et al. [180], Hermeto et al. [181], and Hosni et al. [182] combine opportunistic routing with TSCH. They study the use of opportunistic routing or anycasts in TSCH and propose changes to TSCH to allow non-colliding acknowledgments from multiple receivers. BOOST [183] assigns different sending delays to the potential forwarders and lets the forwarders use carrier sense to ensure a single forwarder. Our second paper (cf. Section 6.2) takes a different approach by combining opportunistic routing with concurrent transmissions (cf. Section 3.5) in TSCH to eliminate the need of choosing a single forwarder.

4.3 TSCH and BLE

TSCH was designed as a MAC layer protocol for the 2.4 GHz mode of IEEE 802.15.4. However, to allow successful communication, it has to be able to coexist with other 2.4 GHz low-power technologies like BLE. Moreover, the performance of TSCH makes it interesting to use it with other physical layers bringing efficient mesh communication to them.

4.3.1 TSCH on Other PHYs

While TSCH is a MAC layer protocol for 2.4 GHz IEEE 802.15.4, multiple research efforts were taken to bring it to other PHYs. Brachmann et al. [184] study the use of TSCH for long-range low data rate communication using the subGHz PHYs of IEEE 802.15.4 [19]. The authors demonstrate the feasibility of using TSCH when adapting the TSCH timeslot timings, and they show the possibility of combining multiple PHYs in a common TSCH schedule. Van Leemput et al. [185] extend the multi-PHY approach by adaptively selecting a PHY based on the current link. They choose a common timeslot length sufficiently large for the lowest supported data rate. Rady et al. [186] build g6TiSCH, another work combining multiple PHYs in a single TSCH network. They perform modifications along the 6TiSCH stack to generalize 6TiSCH, allowing it to intelligently choose which PHY to use in a TSCH slot. Haubro et al. [187] explore the combination of TSCH with LoRa, a different subGHz PHY. With TSCH, they bring multi-hop communication and higher layer protocols like Orchestra [8] to long-range communication with LoRa which could be an infrastructure-free alternative to LoRaWAN [20]. King et al. [188] and Charlier et al. [189] explore the feasibility of using TSCH for Ultra-Wideband (UWB) communication.

4.3.2 Coexistence of TSCH and BLE

As both IEEE 802.15.4 and BLE use the same frequency range with channels that highly overlap (cf. Figure 2) and have similar signal strength, the coexistence between them is essential. Especially for TSCH networks tuned for reliable communication, coordinating the coexistence could be beneficial. Carhacioglu et al. [190] study this coexistence and propose a system with a common TSCH and BLE orchestrator to overcome cross-technology interference. While Carhacioglu et al. try to avoid interference by coordinating the communication of the two technologies, Hajizadeh et al. [191] build a simulation framework analyzing the coexistence and the number of expectable collisions for coexisting TSCH and BLE networks.

4.3.3 Combination of TSCH and BLE

Instead of improving the coexistence of IEEE 802.15.4 TSCH and BLE there exists the possibility to combine IEEE 802.15.4 and BLE within a TSCH schedule. Many new radios for the 2.4 GHz band, like the Nordic nRF52840 [192], support multiple PHYs such as both BLE and IEEE 802.15.4. Therefore, the combination of IEEE 802.15.4, TSCH, and BLE in a single network is thinkable. While the solution by Carhacioglu et al. [190] using a common orchestrator

for TSCH and BLE is one approach to create a common schedule, Baddeley et al. [193] take an entirely different approach. They propose 6TiSCH++ which uses the standard TSCH slots over IEEE 802.15.4 for data communication, but replaces the beaconing slots with concurrent transmissions over BLE. In 6TiSCH++, multiple subsequent concurrent BLE transmissions fit into one TSCH slot and allow for a faster transmission of control information in a TSCH network.

4.3.4 Mesh Networking in BLE

In BLE, the standard solution for mesh networking is Bluetooth Mesh (see Section 3.3.3). However, alternative BLE-based mesh protocols have been explored. Patti et al. [194] devise a connection-oriented protocol for real-time mesh communication atop BLE that uses subnetworks. These subnetworks, forming a star topology and operating in connected mode, interlink nodes across multiple subnetworks to form a mesh network. Subsequently, Leonardi et al. [195] extend and implement this approach. While the previous two form mesh networks using the connected mode of BLE, RESEMBLE [196] is a protocol designed for Bluetooth Mesh that enables TDMA-based communication with time slots and clock synchronization, thus enabling real-time communication within Bluetooth Mesh networks. Petersen et al. [197] extend BLE to support efficient multi-hop IPv6 over BLE using subnetworks, while Lee et al. [198] introduce the RPL routing protocol to BLE.

4.3.5 TSCH on BLE

In contrast to the approaches of merging IEEE 802.15.4 with BLE or constructing a BLE mesh solution utilizing the standard BLE network stack, our fifth paper TBLE (cf. Section 6.5) and BlueTiSCH [199] examine the integration of TSCH and BLE, eliminating the necessity for IEEE 802.15.4 in TSCH networks. While BlueTiSCH utilizes a simulation-based approach employing TSCH-Sim [83], we implement our solution in Contiki-NG [44] and conduct a testbed-driven evaluation in a real-world setting. This approach introduces new use-cases for BLE devices, including static networks currently relying on IEEE 802.15.4.

4.4 Time-Synchronized Testing and Evaluation

To gain meaningful insights when testing and evaluating low-power wireless protocols on testbeds, we need to be able to low-intrusively trace executions simultaneously on multiple nodes that are physically apart. Moreover, we need time-synchronization capabilities to combine and evaluate the individual traces. Several testbeds offer tracing and time-synchronization capabilities. Flocklab [84] can trace up to 5 GPIO pins at a sampling rate of 10 kHz and uses NTP for time synchronization with a time stamping error below 40 µs. Tracelab [93] extends Flocklab with a more capable GPIO acquisition system based on an FPGA, achieving a short-term sampling frequency of up to 100 MHz and a continuous sampling frequency of 285 kHz. It uses Glossy [9] on 868 MHz with an FPGA-based clock correction control loop, achieving a time-synchronization error as low as 1.5 µs. Flocklab 2 [85, 86] uses the

programmable real-time unit (PRU) of a Beaglebone Green [200] for GPIO tracing. For time-synchronization, it uses GNSS with an accuracy of approx. 50 ns where available, and the Precision Time Protocol (PTP) with an accuracy of approx. 1 µs at all other locations. It also supports Serial Wire Debug (SWD) tracing through a J-Link debug probe.

Aveksha [94], Minerva [95], and HATBED [96] use different J-Link tracing methods instead of GPIO tracing, and can trace the program counter, or perform watchpoint tracing non-intrusively. Aveksha allows a polling period of 30 µs but does not perform synchronized timestamping. Minerva adds time-synchronized tracing, and synchronous stopping of the execution. It uses NTP for time synchronization and timestamping, reaching a millisecond accuracy. HATBED [96] uses on-chip debugging capabilities of ARM Cortex-M3/M4 processors. It supports watchpoint-logging and logging of print statements and uses a logic analyzer to trace the output.

Our fourth paper, Grace (cf. Section 6.4) also performs time-synchronized GPIO tracing. However, instead of using specialized hardware like an FPGA or a specific observer board, we use low-cost off-the-shelf logic analyzers and an RBS-like (cf. Section 3.6.2) time-synchronization system with 433 MHz radios. We perform the time-synchronization in software to achieve a low-cost retrofittable time-synchronized testing and evaluation solution for existing testbeds.

4.5 Wireless Localization & Sensing

The common focus of wireless signals is the exchange of information. However, we can also use these signals for other use-cases, like positioning and localizing a device, detecting and locating people in a room or tracking peoples movement. Each of these solutions exploits different characteristics of wireless signals and thus uses application specific wireless fingerprints.

4.5.1 Wireless Positioning

Wireless signals travel with the speed of light and thus, in a line-of-sight setting, devices can use the time-of-flight (ToF) to determine their distance to the signal's source. In combination with tracking the time-of-arrival (ToA), devices can use trilateration/true-range multilateration [70] to compute their position without a synchronized clock on the measuring device. GNSS systems like GPS use this approach for positioning [70, 73]. Another system using a similar approach combining ToF and Time Difference of Arrival (TDoA) is Ultra-wideband (UWB), a precise indoor positioning system (IPS) [70]. Other characteristics that are available for some radios are the angle of arrival (AoA) as well as the received signal strength (RSS) [70].

While a system like GPS works well outdoors and UWB works well within a single room, both require a (direct) line-of-sight connection for the wireless signals [70, 201]. However, many environments are more complex with walls or objects causing reflections of wireless signals, multi-path fading, shadowing, and signal attenuation [201]. Therefore, it is not easy to model indoor signal propagation and even additional parameters like the received signal strength (RSS) are unstable in changing environments, including changes as small as

moving around or opening a door, as well as environmental changes like a change in humidity [201].

Yet, while these parameters are unstable, they are still used for localization. For a successful use of the received signal strength, data is collected at several reference points. RADAR [202] uses WiFi signals to locate and track users inside buildings. With a k-nearest neighbor pattern matching system, RADAR can use the RSS of three IEEE 802.11 access points to determine a user's location with 2–3 meter resolution. Several subsequent works use Bayesian-based filtering methods, e.g., Kalman filters or particle filters, to increase the robustness of the positioning system [203–206]. Other methods use Support Vector Machines [207] for positioning or transfer learning [208] to share localization models over time, across space and to other devices. Chow et al. [209] perform a more coarse-grained locality classification, deciding whether a device is within a specific area. Zhang et al. [210] showcase the usability of deep learning for RF localization. They propose an indoor positioning system using a deep neural network (DNN) in combination with a Hidden Markov Model (HMM). The four-layer DNN outputs a coarse position estimate based on pre-processed RSS from WiFi access points. The HMM then further refines the coarse estimates, resulting in a positioning accuracy of ~0.4 meters. Other works investigate the use of BLE for localization [211–215] with recent solutions [216] using deep learning approaches. Faragher et al. [213] and Zhuang et al. [214] both achieve <3 m localization accuracy using BLE advertisements. Koutris et al. [216] use a CNN to process received signals of multi-antenna anchors to locate a BLE sender with an accuracy of 70 cm.

4.5.2 Device Detection

Next to localization, we can also use wireless signals to find hidden electronic devices [217], or use RF fingerprinting to identify specific devices based on their radio signatures [218–220]. For the latter, one can identify the inaccuracies in the signal, a specific device transmits and use this for classifying the respective device. PARADIS [218] distinguishes 130 identical devices with an accuracy of 99% by analyzing the devices' differences in the modulation domain using support vector machines and k-nearest neighbors. ORACLE [219] uses a CNN to distinguish devices based on I/Q sample variations between the devices, with an accuracy of 99%. However, Al Shawabka et al. [221] show that the wireless channel significantly impacts the performance of CNN solutions.

4.5.3 Detecting and Locating Humans

Instead of locating electronic devices, wireless signals can also be used to locate non-electronic devices or humans within an environment [222–227]. This method is referred to as device-free localization. Zhang et al. [222] show how one can estimate the position of a human within a grid deployment using the received signal strength (RSS). SCPL [223] uses an algorithmic approach using RSS values to count and locate multiple moving humans down to 1.3 meters. Wi-CaL [226] uses machine learning and deep learning methods, which utilize channel state information (CSI) for counting and localizing humans with above 90% accuracy. Rapid [224] combines the CSI information with acoustic measurements to identify an individual person within a group of people. Lastly,

Wang et al. [227] use machine learning approaches to detect occupancy of rooms with both stationary and moving humans. Instead of using WiFi, BLECS [225] uses BLE and reinforcement learning to detect whether there are humans in a room.

4.5.4 Activity Detection

We can not only use wireless signal to detect the occupation of rooms or the movement of humans. Several works show that we can even use RF signals for activity detection – the estimation of the current activity of a human. Sigg et al. [228] can detect the presence of humans and that they are active, as well as human gestures from changes in the received signal strength of WiFi signals at a phone. Wang et al. [229] are able to classify in-place activities using WiFi channel state information. Chen et al. [230] use a ceiling mounted UWB radio to recognize a set of human activities within a room. Other works [231–233] use RF signals to detect multiple people behind walls and visualize and detect their activities. Zhao et al. [231] use RF signals to estimate 2D poses of humans. Li et al. [232] continue on this work and build a spatio-temporal attention feature learning model which allows skeleton-based activity recognition of humans. Geng et al. [233] use WiFi for 3D pose estimation using a transfer learning approach from image-based systems to WiFi signal-based ones.

Instead of using wireless signals, other signal-free human activity recognition methods use mobile sensors like accelerometers, gyroscopes, magnetometers, or barometers to track a user's movement or their activities – sensors that are commonly available on smartphones or wearables [234].

4.5.5 Environment detection

While the localization problem focuses on detecting the specific location of a device and the detection methods discussed above focus on detecting a specific device or person, environment detection aims at classifying the general surrounding environment, such as home, office, street, or shop. Several works use audio-based systems to accurately detect the environment [235–237]. Ma et al. [235] use a Hidden Markov Model classifier to distinguish between 12 environments—such as bus, office, street, or supermarket—with an average accuracy of 92% using 3-second long recordings. Qamhan et al. [236] combine Convolutional Neural Networks (CNN) with Long Short-Term Memory (LSTM) to classify the environment and microphone features from voiced and unvoiced segments of a recording. They distinguish between three environments (office, cafeteria, and sound-proof room) with an accuracy of 98%. Heittola et al. [237] create audio fingerprints for different environments as histograms and compare recordings using a k-nearest neighbors algorithm to match these histograms. They can distinguish 12 environments with a maximum accuracy of 92.4% using >120 second recordings. All three do not address privacy concerns in using audio recordings for environment classification. Choi et al. [238] combine an acoustic and a visual system to detect the context—such as car, bag, office, or subway—a user's device is located in. Liang and Wang [239] use a convolutional neural network to detect a user's mode of transport—such as car, bicycle, or train—from a smartphone's accelerometer data. The system achieves an accuracy of 94.48% for distinguishing 8 modes of transportation.

In our third paper of this thesis (cf. Section 6.3) we argue that received Bluetooth Low Energy advertisements are sufficient to classify the surrounding environment and are less privacy-invasive than audio-based systems.

5 Research Questions

Protocols for the (Industrial) Internet of Things stand out for reliable communication using an unreliable wireless medium. With an increasingly congested wireless medium and the growing number of deployments, modern IoT solutions have to achieve reliable communication or operation in even harsher environments. We expect IIoT networks to remain fully functional, reliable and still meet communication deadlines even in the presence of unforeseen interference and degrading communication links. We expect IoT devices accompanying us, like wireless headphones, to smartly adjust their settings to the surrounding environment.

As we discuss in Sections 2–4, state-of-the-art protocols are not ready for these challenges. Therefore, we design and evaluate protocols, that combine strategies from different low-power wireless communication approaches to automatically and resource-efficiently adapt to the changes in the wireless environment without neglecting application-specific requirements. Moreover, we realize new applications for wireless IoT solutions to enable devices to adapt their functionality to the surrounding environment. This thesis specifically identifies and answers the following five Research Questions (RQs).

RQ1: How can we ensure highly stable communication in centrally scheduled low-power wireless networks which are susceptible to dynamic changes in the wireless environment?

RQ2: How can we reduce latency in low-power wireless mesh networks without negatively impacting reliability?

RQ3: Can resource-constrained low-power wireless communication devices accurately recognize their environment without additional sensors?

RQ4: How can we build a cost-efficient debugging and evaluation system for low-power wireless protocols?

RQ5: How can we use low-power communication protocols to create adaptive and efficient systems for modern IoT solutions?

RQ1 stems from the observation, that communication for the Industrial Internet of Things usually uses centrally scheduled networks. However, these networks follow strict schedules that do not allow for flexibility within end-to-end communication flows and are not able to adapt to interference changes in the wireless medium. In Chapters A and B, we answer this question. Chapter A introduces flow-based retransmissions, adaptively utilizing retransmissions wherever interference makes them necessary along an end-to-end communication flow. Chapter B introduces opportunistic routing with concurrent forwarding in TSCH to dynamically route traffic around interference sources along the main communication path.

RQ2 approaches the challenge of communication systems with the lowest feasible latency without compromising reliability. We answer this question in Chapters A, B, and E. Both solutions in Chapters A and B achieve a best-case latency equivalent to a system without retransmissions, using a single path for routing. These solutions circumvent the necessity to await unneeded communication slots or forwarder coordination. Chapter E takes a different approach to tackle this question by replacing the IEEE 802.15.4 physical layer with BLE physical layers while retaining the TSCH network stack.

RQ3 looks at novel use-cases of wireless communication signals. Instead of using wireless signals for communication, we ask whether we can use wireless signals to detect the environment and adapt the functionality of smart devices like wireless headphones. Chapter C answers this question by building a solution that can infer a device's environment solely from the fingerprint of ambient Bluetooth Low Energy advertisement signals.

RQ4 approaches a problem orthogonal to the previous research questions, yet important for fully exploring and understanding low-power wireless mesh protocols. For evaluating these protocols, the research community commonly uses testbeds, however, evaluating timing-specific aspects in a distributed setting requires the ability to timestamp instructions or events in these distributed testbeds. In Chapter D, we present an affordable time-synchronized testbed evaluation system that provides us with in-depth insights into the protocols we develop to answer RQ1 and RQ2, and that is retrofittable to existing testbeds.

RQ5 reflects the general idea of this thesis as it relates to the general problem of modern IoT solutions that must be able to resource-efficiently adapt to changes in the environment when using wireless communication. All five papers we present in Chapters A – E include aspects answering this question. Chapters A and B add dynamicity to centrally, and often statically scheduled IEEE 802.15.4 networks. Chapter C builds a system capable of adapting applications based on the environment. Chapter D uses subGHz low-power wireless communication to efficiently time-synchronize evaluation infrastructure, and Chapter E demonstrates the use of the 6TiSCH stack on top of BLE opening the field for future IoT and IIoT solutions.

6 Thesis Contributions

This section summarizes the papers that constitute the main contribution of this thesis. Table 1 outlines the chapters of this thesis, along with the name of the protocol or system we discuss in each chapter, and the research questions it addresses.

6.1 Chapter A – Master: Long-Term Stable Routing and Scheduling in Low-Power Wireless Networks

Context and Challenge. The Industrial Internet of Things (IIoT) requires efficient ways of scheduling communication in low-power wireless networks. Such schedules have to ensure energy-efficient and reliable communication while keeping latency low to enable systems to quickly react to changes in industrial processes.

Table 1: Research questions and the corresponding thesis chapters addressing them, together with the name of the protocol or system discussed in each chapter.

	Chapter A *Master*	**Chapter B** *Autobahn*	**Chapter C** *BlueSeer*	**Chapter D** *Grace*	**Chapter E** *TBLE*
RQ1	●	●	○	○	○
RQ2	●	●	○	○	●
RQ3	○	○	●	○	○
RQ4	○	○	○	●	○
RQ5	●	●	●	●	●

As we discuss in Section 4.1.1, centralized TSCH schedulers until this work were unable to react to unforeseen interference. Many solutions focus on schedulability of communication assuming interference-free channels with perfect links [10, 11], or schedule backup slots for retransmission along specific links, but usually only one backup slot per link [16, 119]. However, in networks without exclusive access to certain channels and other wireless traffic such as WiFi, a scheduler has to account for dynamic wireless environments with changing levels of interference on each link.

Approach. MASTER is a centralized scheduler for TSCH networks, that addresses the challenge of handling unforeseen changes in the wireless medium. MASTER separates the control plane from the data plane, and thus consists of two parts: a central server component and a networking component (cf. Figure 9). Its central server component runs on the edge of a sensor network or even in the cloud and performs the routing and scheduling. Its networking component runs on each sensor node on top of TSCH and performs the schedule installation and the forwarding according to the central component's routing. This twofold design splits the compute-heavy scheduling task from the network operation of the usually resource-constraint devices in an IoT or sensor network.

MASTER's design, is flexible to dynamically handle unexpected interference levels at any link along a multi-hop communication flow and does not limit retransmissions to specific links. We can dynamically react to local interference by using a novel flow-based retransmission scheme, in the paper denoted as *Sliding Windows*. This scheme allocates retransmissions for a flow instead of a specific link, allowing any link of the flow to use between no and all retransmission slots (cf. Figure 10). If no link needs retransmissions, we do not have to wait for retransmission slots to pass. Instead, we can immediately continue with the next hop in a subsequent slot to keep latency at a minimum. This dynamic approach allows us to use retransmissions for different links at different iterations of the schedule, wherever we need them due to the currently prevalent interference pattern. Thus, MASTER achieves robust and long-term stable schedules while keeping latency at a minimum.

Results. We evaluate MASTER and its flow-based retransmission strategy in experimental testbed evaluations and show that it achieves a delivery ratio of above 99%. We experimentally compare its performance to the well-established

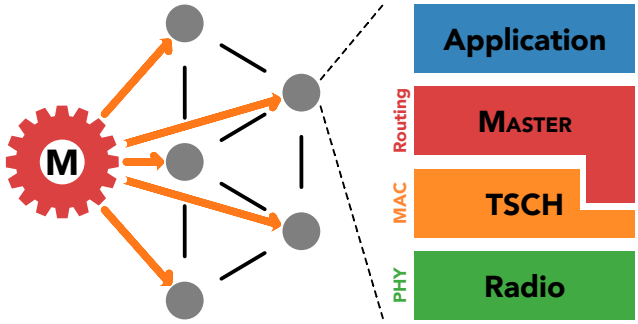


Figure 9: Master consists of an external centralized scheduler (M) and a routing layer. The external scheduler performs the global routing and scheduling and pushes the computed schedule onto the network. In each node, Master's routing layer implements the schedule in TSCH and performs the routing during runtime.

autonomous TSCH scheduler Orchestra [8] showing that MASTER with its flow-based retransmission scheme has 85–90% lower latency than Orchestra while maintaining the high reliability Orchestra is known for. This clearly shows MASTER's advantage when communication partners are known upfront. Lastly, we perform a long-term evaluation over 24 hours showing its long-term stability for highly reliable (>99.6%), low-latency (<4.5 slots) communications.

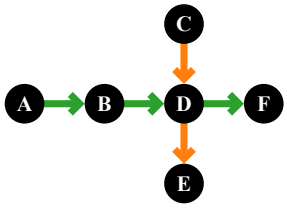
Contributions. In summary, this chapter makes the following contributions:

- We present MASTER, an open-source, centralized router and scheduler for TSCH-based networks designed with easy extendability in mind.
- We design Sliding Windows, a transmission strategy for MASTER to increase the flexibility, stability, and reliability of centrally scheduled communications.
- We propose flow-based queues as an extension to TSCH to enable the use of central scheduling algorithms.
- We implement MASTER as part of Contiki-NG and evaluate it in environments susceptible to interference.
- We show the long-term stability of schedules computed by MASTER in experiments of 24 hours.

Statement of Personal Contribution. I am the lead designer and implementer of MASTER and its flow-based retransmission scheme. Additionally, I designed and conducted the experimental evaluation. I am also the lead author of the paper.

The chapter was published as a paper in the Proceedings of the 16th International Conference on Distributed Computing in Sensor Systems (DCOSS), 2020 [240], and its source code is available on GitHub[2].

[2]Available at `https://github.com/ds-kiel/master-scheduler`



(a) Sample Topology with two intersecting flows.

	1	2	3	4	5	6	7	8	9
A				TX	TX	TX	TX		
B				RX	RXTX	RXTX	RXTX	TX	
C	TX	TX	TX						
D	RX	RXTX	RXTX	TX	RX	RXTX	RXTX	RXTX	TX
E		RX	RX	RX					
F						RX	RX	RX	RX

(b) Flow-based retransmission schedule, with one transmission slot per hop and two (orange flow) or three (green flow) retransmission slots to be shared among the nodes of a flow.

Figure 10: Sample schedules for Master's flow-based retransmission strategy. One flow originates at node A to end at node F while the second one originates at node C and ends at node E.

Naming choice. When selecting a name for our centralized scheduler, we sought a term that accurately portrays its function within a network. The scheduler both masters the skill of computing an efficient schedule and is the expert in a system of wireless sensor nodes. Consequently, we chose the name Master.

We are cognizant of the connotations associated with the words "master" and "slave" in the United States due to the country's historical context. However, the term "master" itself holds a broader meaning and application beyond its combination with "slave," and particularly outside the US, it may not carry the same negative connotation. Although our protocol does not involve any components referred to as "slaves," in hindsight, we would likely opt for a different name to avoid unintentional associations in English-speaking regions. Potentially, we might choose a word from a different language to sidestep any inadvertent connotations.

Discussion and possible extensions. With Master we create a centralized scheduler that can adapt to changes in the wireless environment by using flow-based retransmissions and eliminating the need for regular rescheduling. Master adds dynamic aspects to otherwise static schedules and reduces latency to a minimum without compromising reliability. For Master we use a Reverse Longest Path First (R-LPF) scheduling algorithm that reduces waiting times and thus reduces latency within end-to-end flows. However, many applications in the IIoT are deadline-driven, and therefore, Master would be more applicable for real-world applications if it supported deadline-based scheduling algorithms. A bachelor's thesis [241] supervised by me implements three deadline-driven scheduling algorithms to close this gap. Moreover, Master has no built-in method for neighbor data collection and schedule dissemination connecting the control and data planes. A master's thesis [242] supervised by me builds a system for an initial collection of neighbor data and disseminating a first schedule, to increase the usability of Master in unknown deployments.

In this paper, we present a method for computing the required number of retransmission slots for both link-based and flow-based retransmissions. Our approach utilizes the ETX metric [48]. While effective for routing decisions,

this metric may not be optimal for determining the number of retransmission slots. Instead, employing a Markov chain, such as the approach introduced in the paper proposing a flow-centric policy in WirelessHART [131], offers a more sensible solution for selecting the number of retransmissions. Incorporating this method into MASTER would be beneficial.

is a more sensible solution for selecting the number of retransmissions, and it is worth adding it to MASTER.

Especially with the introduction of TSCH on BLE (cf. Section 6.5) MASTER would also be a viable candidate to build centrally scheduled networks featuring heterogeneous physical layers. With a global view of the network, it becomes feasible to schedule more efficient communication by employing a specific physical layer for each flow or link within the network.

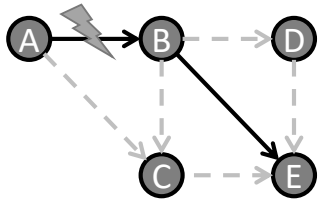
6.2 Chapter B – Opportunistic Routing and Synchronous Transmissions Meet TSCH

Context and Challenge. As the number of mobile and IoT devices communicating increases, the wireless medium becomes increasingly congested, posing a greater challenge for successful communication. This results in a significantly higher spectrum occupation and heightened complexity for successful communication. Scheduled communication in networks utilizing Time-Slotted Channel Hopping brings measures to reliably communicate in such a harsh environment, including retransmissions and channel hopping. Retransmissions across different physical frequencies increase the likelihood of successful communication in the presence of narrowband interference [8, 131, 240].

However, with an increasingly occupied spectrum, wideband interference emerges as a new challenge. Schedules designed for long-term stability must be immune to wideband interference, as interference can lead to link failures or render nodes unreachable, significantly impacting routed communication. While our solution of flexible retransmissions (discussed in Section 6.1) effectively mitigates narrowband interference, it shares a common limitation with other TSCH schedulers when facing wideband interference. Communication in TSCH networks typically follows a single path, and if there's a wideband interference source along that path, it cannot be handled, resulting in communication failure (cf. Figure 11a).

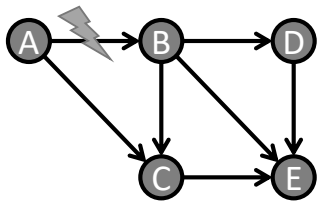
Approach. In this chapter, we show that we can tackle the problem of local wideband interference by combining three established technologies in low-power wireless networks: Time-Slotted Channel Hopping (TSCH) [6, 8, 131, 240], Opportunistic Routing [171, 174, 175], and Synchronous Transmissions [9, 59, 65]. We design and evaluate AUTOBAHN, a hybrid routing scheme that combines these three technologies. AUTOBAHN establishes long-term stable schedules in the presence of local wideband interference by routing traffic from a sender to a receiver along a broader path and enabling neighboring nodes to concurrently transmit the same data in the same timeslot using the same frequency channel. The routing approach follows the concept of opportunistic routing (cf. Section 4.2.1), where the specific path a packet takes to reach its destination is inconsequential.

In AUTOBAHN, a node forwards a packet opportunistically to multiple neighboring nodes, which subsequently, in their next scheduled slot, simultaneously



	1	2	3	4
A	TX	TX		
B	RX	TXRX	TX	
C				
D				
E		RX	RX	

(a) Established central scheduling approaches employ a single routing path. Their schedule will fail if one of the links fails, such as the link between nodes A and B in this example.



	1	2	3	4
A	TX	TX		
B	RX	TXRX	TX	
C	RX	TXRX	TX	
D		RX	RXTX	TX
E		RX	RX	RX

(b) Autobahn utilizes opportunistic routing and thereby provides redundant options in case routes fail. In this example, packets can travel via node C to destination E.

Figure 11: Autobahn compared to established centralized TSCH scheduling approaches. In this example, we assume a topology of five nodes, with node A as source and node E as destination. We show both the scheduled paths and the TSCH schedule, using RX, RXTX, and TX slots as typical for flow-based retransmission schemes (with a retransmission window of two). Grayed-out slots present slots where reception and transmission are not possible due to previously failed interfered receptions.

forward it opportunistically to their neighbors. In scenarios where a node cannot participate in forwarding the packet due to interference, this poses no issue as other nodes can step in to forward the packet instead. Figure 11 illustrates an example scenario and schedule for AUTOBAHN in comparison with a flow-based single-path scenario created by schedulers like MASTER (cf. Section 6.1).

The combination of TSCH, opportunistic routing, and synchronous transmissions addresses the main challenge of each of these three individual concepts. TSCH provides the time-synchronization required for synchronous transmissions. Synchronous transmissions eliminate the challenge of selecting a single forwarder of a packet in opportunistic routing by enabling multiple concurrent forwarders. And opportunistic routing resolves the issue of stability loss in the presence of a wideband interference source along a single path.

Results. With AUTOBAHN, we demonstrate the feasibility of synchronous transmissions in TSCH networks. TSCH networks are well-enough synchronized to allow the reception of synchronously transmitted packets due to the capture effect (cf. Section 3.5.1). Through the combination of opportunistic routing, synchronous transmissions, and TSCH, we keep complexity low and do not

increase the minimum latency compared to a single path schedule. Moreover, this combination even allows a lower average latency in the case of local narrowband interference. Our evaluation shows that AUTOBAHN is capable of outperforming single-path retransmission strategies including MASTER's flow-based retransmission strategy and other single-path retransmission strategies both in the presence of and without interference. Moreover, AUTOBAHN offers long-term stability with over 95% reliability over several days without the need for rescheduling.

Contributions. In summary, this chapter makes the following contributions:

- We are the first to combine the concepts of opportunistic routing, synchronous transmissions, and Time-Slotted Channel Hopping (TSCH) into a single protocol, achieving long-term stable routed communication.
- We design AUTOBAHN, a robust scheduling and routing policy that withstands link and node failures in the presence of interference.
- We implement AUTOBAHN for Contiki-NG [44] and evaluate it in environments susceptible to interference. We show the long-term stability of schedules using AUTOBAHN over 12 days and under various interference levels for 25 hours. These experiments achieve reliability under interference of 96.8% and latency of 4.2 slots, outperforming both the central scheduler MASTER [240] and the autonomous TSCH scheduler Orchestra [8].

Statement of Personal Contribution. I am the lead designer and implementer of AUTOBAHN. Additionally, I am the main designer of the evaluation and the main author of the paper. The chapter was published as a paper in the Proceedings of the 46th IEEE Conference on Local Computer Networks (LCN), 2021 [243], and its source code is available on GitHub[3].

Discussion and possible extensions. In this work, we demonstrate the feasibility of synchronous transmissions within TSCH, showcasing their potential to reduce the coordination overhead associated with forwarder selection in opportunistic routing. While time synchronization in TSCH enables successful utilization of the Capture Effect, it remains necessary for a single node's radio signal to be significantly stronger than the combined strength of all other signals. AUTOBAHN shows that successful forwarder selection is achievable; however, there exists room for enhancement, particularly by considering the expected received signal strength. If signals from multiple senders exhibit comparable signal strengths, AUTOBAHN might inadvertently decrease communication reliability. Consequently, by incorporating received signal strength into the forwarder selection process, a more robust version of AUTOBAHN could be developed.

So far, AUTOBAHN demonstrates the possibility of synchronous transmissions in TSCH using the IEEE 802.15.4 physical layer. However, with TSCH now available on additional physical layers (cf. [184, 187–189] and Section 6.5), the question arises whether a solution like AUTOBAHN could be applicable to these alternative physical layers. While the general feasibility of synchronous

[3]Available at `https://github.com/ds-kiel/autobahn`

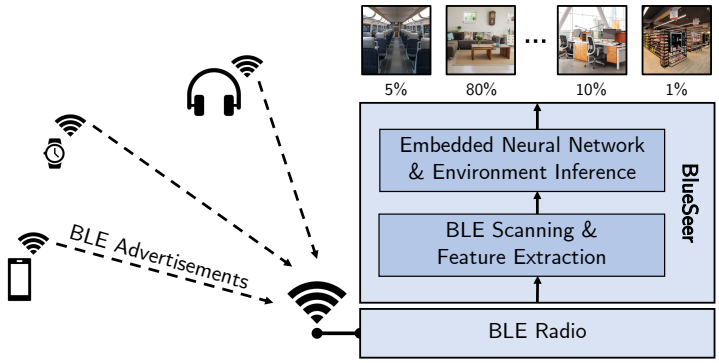


Figure 12: BlueSeer: System architecture. BlueSeer scans the wireless medium for BLE packets and extract features from them. An embedded neural network classifies the environment between 7 categories.

transmissions on BLE has been demonstrated in [60] and [64], it necessitates a greater power difference to use the capture effect. Nonetheless, when combined with meticulous forwarder selection during scheduling, this approach may offer a compelling strategy, particularly for ensuring reliable communication in networks employing multiple physical layers.

6.3 Chapter C – BlueSeer: AI-Driven Environment Detection via BLE Scans

Context and Challenge. Bluetooth Low Energy (BLE) devices are all around us, such as wireless keyboards, wireless headphones, smartphones, smart speakers, smart sensors, and many others. These devices constantly announce their presence and their provided service for other devices, like a smartphone or a computer, to be able to interact with them. During the COVID-19 pandemic, we saw another strength of BLE: due to its low signal range, it is possible to trace devices and thus people who have been in a user's vicinity to thus trace the spread of infections [38].

This behavior of BLE announcing its services and sending advertisements all the time makes its use also thinkable for other applications. One such application could be detecting the environment using ambient BLE signals instead of sensors like microphones [235–237], cameras [238], or accelerometers [239]. Detecting the environment from ambient BLE signals would enable low-power wireless devices without additional sensors to adapt their behavior to the environment. For example, wireless headphones could lower their noise-cancellation levels in traffic to ensure the safety of their users, and wearables could keep silent while in a theater.

Approach. In this work, we present BlueSeer, a smart environment-detection system capable of inferring environments when running on resource-constraint IoT devices, like wearables or microcontrollers. BlueSeer only requires a BLE radio scanning for BLE advertisements of nearby devices. We perform feature extraction from the collected data and use an embedded neural

network to accurately predict the current environment (cf. Figure 12). BlueSeer distinguishes between 7 environments (home, office, shopping, transport, nature, street, and restaurant) based on 23 different features extracted from the BLE data. A feed-forward neural network with a single 500-neuron hidden layer and a 7-neuron output layer performs the classification of the current environment. This network is sufficient for successfully classifying the environment and easily fits the memory footprint of modern microcontrollers.

Results. With BlueSeer, we demonstrate that it is feasible to infer an environment solely from advertisement packets received with a BLE radio on resource-constrained hardware. A small neural network that fits the memory footprint of a modern microcontroller is sufficient to achieve an overall classification accuracy of 84%. Different classes of environments pose different challenges, with environments with low dynamics such as home, restaurant, or office being better detectable with an accuracy of around 98%. BlueSeer is able to perform the inference within 13 ms on a Cortex-M4 microcontroller clocked at 64 MHz [192].

Contributions. In summary, this chapter makes the following contributions:

- We show that it is possible to categorize environments exclusively from received BLE advertisements.
- We present BlueSeer, an Environment Detection system able to classify environments solely using received BLE packets. BlueSeer distinguishes between 7 categories: home, office, shopping, transport, nature, street, and restaurant.
- We carry out extensive feature engineering and identify 7 features from BLE advertisements, ranging from number of devices in proximity and RSS measurements, to the diversity of offered services.
- We devise a neural network and show that its quantized, embedded version classifies its environment with up to 84% accuracy on a low-power platform with a 64 MHz MCU, and uses 65 KB of memory.

Statement of Personal Contribution. I am the second author and a co-designer of BlueSeer. Additionally, I designed and conducted the data collection and on-device evaluation. We attribute 60% of the workload to the first author and 40% to me, the second author.

The chapter was published as a paper in the Proceedings of the ACM/IEEE Design Automation Conference (DAC), 2022 [244], and was nominated as *candidate for the best paper award* at the conference. Its source code is available on GitHub[4].

Discussion and possible extensions. With BlueSeer, we show that BLE is the sole requirement to identify an environment. We show that a system like this is feasible on a microcontroller, however, the limited computing capabilities and the current state of supported AI/ML features in TensorFlow Lite for Microcontrollers [245] limit the exploration space significantly. To further explore the field and the capabilities of using ambient BLE signals, a

[4]Available at `https://github.com/ds-kiel/blueseer`

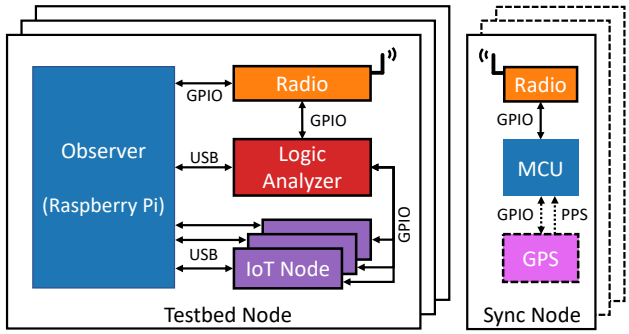


Figure 13: Design Overview of Grace. The testbed node consists of an observer platform and one or more IoT nodes. We add a logic analyzer and a radio for time-synchronized GPIO tracing. The synchronization node consists of a microcontroller and a radio generating the time signal. In case of larger deployments, we can use multiple synchronization nodes and add an otherwise optional GPS receiver.

smartphone would be a better fit. A smartphone is still a movable device carried around by its user and allows us to try larger neural networks and different approaches like unsupervised learning for environment detection on a mobile end-device. Two master's theses [246,247] supervised by me demonstrate that a BLE-based classification system cannot only distinguish between environments, but can also more granularly distinguish between multiple instances of the same environment, i.e., we can identify specific homes or offices. More elaborate models could be able to identify not only known locations, but also the places a user regularly visits. Such a system could identify changes in a user's behavior when they visit unknown places or remain longer at certain known ones.

6.4 Chapter D – Grace: Low-Cost Time-Synchronized GPIO Tracing for IoT Testbeds

Context and Challenge. Designing wireless communication protocols requires infrastructure to test and evaluate the protocols' performance. While simulations [80,81] allow high-level insights into protocols and algorithms, they cannot accurately replicate all details of real hardware like CPU-specific timing and environmental factors, nor are they capable of simulating the specifics of the wireless environment and random interference patterns the protocol has to handle [84,85]. Therefore, research on wireless communication protocols and systems commonly uses testbeds—deployments of IoT devices co-located with observer infrastructure for instrumentation, logging, and deployment control.

We can use testbeds to evaluate protocols and their performance under real-world conditions with interference sources not under our control or with artificially generated interference [248, 249]. However, for testing the inner workings of protocols, or evaluating the concurrent execution of instructions, like the participation of nodes in concurrently forwarding data (cf. Section 6.2), we need distributed logging or tracing capabilities. Most testbeds provide

serial logging capabilities, however, printing messages takes several hundreds of microseconds leading to side effects on program execution and limits accurate timestamping, especially if we need to trace microsecond offsets between IoT nodes. A less intrusive and more time accurate method is tracing state changes of General-Purpose Input/Output (GPIO) pins of a processor or microcontroller, which are usually accessible on any IoT development hardware. However, tracing these signals requires specialized hardware or specific observer platforms. Moreover, in a distributed system, we require precise time-synchronization to trace microsecond offsets between IoT nodes.

Approach. In this chapter, we present Grace, a time-synchronized GPIO tracing solution for IoT testbeds (cf. Figure 13). While Grace is not the first solution of its kind [84, 85, 93], it does, in contrast to the other approaches, not require any specific observer platform. Instead, it is retrofittable to existing testbeds and uses low-cost off-the-shelf hardware, making it a cost-efficient and affordable solution. Grace uses a unidirectional RBS-like time-synchronization system (cf. Section 3.6.2) to synchronize the testbed observers. A synchronization node broadcasts a time signal once a second and the testbed nodes receive the time signal at roughly the same time with a negligible time difference. We use wireless 433 MHz transceivers to send and receive the timestamps, and logic analyzers for GPIO tracing and logging events. The logic analyzers also trace the reception of the time signal for precisely synchronizing the GPIO tracing system. Grace can use either a single synchronization node for building-wide testbeds, or multiple synchronization nodes for campus-wide testbeds. In the design with multiple synchronization nodes, we use the 1-PPS signal of GPS receivers to simultaneously send a time signal from all synchronization nodes. To not interfere with the signals from other nodes, physically close ones use different frequency channels for their transmission.

Results. We experimentally evaluate Grace in our testbed showing that it is capable of continuously logging sparse amounts of data as commonly produced when debugging IoT systems, such as wireless protocols, at a rate of 8 MHz. Grace is able to achieve a time-synchronization offset between multiple testbed nodes of on average 1.53 µs using a single synchronization node, which is sufficient for most applications. When using multiple synchronization nodes, Grace achieves a time-synchronization offset of on average 15.3 µs between nodes using a different time source. Nodes using the same time source remain at the lower offset as in the system with a single synchronization node.

Contributions. In summary, this chapter makes the following contributions:

- We present Grace, a low-cost time-synchronized GPIO tracing system for IoT testbeds.
- We implement Grace using off-the-shelf hardware to enable easy adoption in other testbeds and make both the software and the hardware setup openly available.
- We introduce multiple types of synchronization nodes, enabling time-synchronization for both building-wide and campus-wide testbeds.
- We discuss and evaluate the intrusiveness of GPIO tracing on IoT platforms.

- We show Grace's low cost of less than €20 per node.

- We evaluate Grace, showing its degree of time-synchronization between nodes of a single synchronization node of on average 1.53 µs, while not exceeding a worst-case synchronization offset of 3.75 µs.

- We evaluate the time-synchronization performance of Grace when using multiple time sources and show that its degree of time-synchronization is on average around 15.3 µs between nodes using different time sources.

Statement of Personal Contribution. I am the main designer of Grace. Additionally, I designed and conducted the experimental evaluation. I am also the lead author of the paper. Grace was co-designed and implemented by Christian Richter as part of his bachelor's thesis [250] and as a student assistant under my supervision.

The initial version of Grace was published as a paper in the Proceedings of the 18th International Conference on Distributed Computing in Sensor Systems (DCOSS), 2022 [251]. This paper received the *best paper award* at the conference. The version presented in this chapter is an extension of the initial version and was published as a paper in the Elsevier Computer Networks journal, volume 228, 2023 [252]. Grace's source code is available on GitHub[5].

Discussion and possible extensions. With Grace, we bring distributed, time-synchronized GPIO tracing to testbeds without the need for specialized observer platforms. Grace is retrofittable to existing testbeds and requires only readily available off-the-shelf components. However, the current system also has its limitations. While transmitting the time signal with standard 433 MHz radios works and leads to a sufficient synchronization, there is still room for improvement, especially regarding the long-term stability of receiving the signal. The 433 MHz receivers are very susceptible to interference. While skipping the reception of an occasional time signal is not that problematic, sometimes the communication fails entirely. An idea to continue using the 433 MHz band while improving the stability of the communication would be to use 433 MHz LoRa [253] to achieve stable communication through the use of a less interference-prone modulation scheme.

Another weakness lies in the logic analyzer and the USB protocol. Some low-cost logic analyzers utilized in Grace encounter occasional failures in communication with the host platform via USB, occurring randomly. In instances of failure, the host may intermittently fail to recognize the logic analyzer until a reboot is performed. While similar issues may arise when using the logic analyzer locally on a single computer, the impact is exacerbated in a distributed tracing system. If tracing fails on a critical node, the entire experiment may necessitate rerunning. Therefore, to increase the stability of low-cost GPIO tracing, debugging the USB communication or exploring alternative tracing solutions could prove valuable.

[5]Available at `https://github.com/ds-kiel/grace`

6.5 Chapter E – TSCH meets BLE: Routed Mesh Communication over BLE

Context and Challenge. While IEEE 802.15.4 and Bluetooth Low Energy (BLE) operate within the same wireless spectrum, BLE has emerged as the dominant standard for communication in low-power wireless networks. Smartphones utilize BLE to interact with wearables, wireless speakers, household items, and various low-cost smart-home devices. However, in applications requiring coordinated mesh networking, such as the Industrial Internet of Things (IIoT) and advanced smart home setups, IEEE 802.15.4 is the preferred choice [12–14]. IEEE 802.15.4, when coupled with the Time-Slotted Channel Hopping (TSCH) MAC layer and the 6TiSCH stack, serves as the foundation for numerous industrial applications with stringent latency and reliability requirements.

Although BLE offers a mesh networking solution known as Bluetooth Mesh (cf. Section 3.3.3), its widespread adoption has been limited. One plausible explanation is that BLE devices, unlike other IoT devices, are frequently mobile rather than stationary for extended durations or do not belong to the same network. Additionally, current use cases for BLE devices typically do not necessitate mesh communication. Moreover, Bluetooth Mesh employs flooding, which involves significant portions of the network in transmitting a single message. Furthermore, it lacks time synchronization, thereby inheriting the same limitations as BLE advertisements for packet transfer from hop to hop. While Bluetooth Mesh is suitable for handling low data volumes [41, 42], its lack of a coordinated network structure restricts its usability in (static) industrial networks with strict requirements.

Approach. In this chapter, we introduce TBLE, a novel solution that brings Time-Slotted Channel Hopping (TSCH) to Bluetooth Low Energy (BLE) for efficient time-synchronized mesh networking in BLE. TBLE seamlessly integrates the BLE physical layer with the TSCH MAC layer, enabling the utilization of well-established protocols, including real-time communication protocols, atop BLE. With TBLE, we establish a time-synchronized BLE network capable of sending and receiving BLE advertisements in a timely manner. To ensure compatibility with other BLE applications, we embed standard TSCH packets within valid BLE advertisements. Consequently, devices not running TBLE can easily discard these advertisements, while devices running TBLE can join the network as in TSCH on IEEE 802.15.4. We develop versions of TBLE for each of the four BLE data rates and evaluate their performance in comparison with TSCH on IEEE 802.15.4. Our evaluation includes assessing whether BLE can effectively replace IEEE 802.15.4 in the context of reliable low-latency mesh communication.

Results. We evaluate TBLE in experiments on our testbed showing its performance in direct comparison with IEEE 802.15.4. We evaluate both the general feasibility of TBLE and its performance using the autonomous TSCH scheduler Orchestra [8]. While BLE is capable of data rates as low as 125 kbps and high data rates of up to 2 Mbps, the medium data rates of 500 kbps and 1 Mbps are those achieving the best performance. TBLE is capable of forming a network using any of these data rates. Further, at the two medium data rates, TBLE matches the reliability performance of TSCH on IEEE 802.15.4

while reducing its latency by up to 20%.

With a higher spectral efficiency than in IEEE 802.15.4 (40 vs. 16 channels), on-par reliability and lower latency, TBLE enables routed mesh communication over BLE, effectively replacing the need for IEEE 802.15.4.

Contributions. In summary, this chapter makes the following contributions:

- We present TBLE, a protocol closing the gap of routed mesh-communication in BLE. TBLE extends the established TSCH standard.
- We design and implement a BLE driver for the Nordic nRF52840 DK for Contiki-NG and adjust it to be compatible with the Contiki-NG IEEE 802.15.4 TSCH and 6TiSCH stack.
- We are the first to run TSCH over BLE, demonstrating TBLE as a practical routed mesh-protocol for BLE.
- We experimentally evaluate TBLE and compare its performance to IEEE 802.15.4 TSCH, showing its feasibility and a performance increase over TSCH without modifying any upper-layer protocols.

Statement of Personal Contribution. I am the sole designer and implementer of TBLE. Additionally, I designed and conducted the experimental evaluation. I am also the lead author of the paper.

The chapter was published as a paper in the Proceedings of the 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT), 2023 [254], and its source code is available on GitHub[6].

Discussion and possible extensions. With TBLE, we demonstrate that coordinated mesh communication in the 2.4 GHz band can be achieved using a simpler modulation scheme, and thus cheaper radios. This chapter serves as an initial step and proof of concept, paving the way for further exploration of TSCH on BLE. Thus far, our evaluation has focused on using a single PHY at a time and payloads not exceeding the 127 bytes limit of IEEE 802.15.4. However, BLE supports larger payloads, and with the availability of multiple PHYs, a combination of them might be feasible. Novel scheduling approaches could allocate certain communications using one of the BLE data rates and concurrent communications using another data rate to maximize spectrum utilization and minimize latency while maintaining high reliability. Furthermore, the combination of IEEE 802.15.4 TSCH and BLE is possible as many modern radios support both protocols. Consequently, a scheduler could construct a network comprising devices supporting either or both of these PHY technologies, effectively combining IEEE 802.15.4 and BLE within a single network.

Around the same time we developed TBLE, another paper (BlueTiSCH [199]) explored the combination of TSCH and BLE. They arrive at slightly different conclusions regarding the performance of the different PHYs. Both works concur that the coded PHY with a data rate of 500 kbps is one of the two best-performing options. However, while our experiments suggest that the

[6]Available at `https://github.com/ds-kiel/TBLE`

uncoded 1 Mbps PHY has similar performance to IEEE 802.15.4, their results differ and indicate that the coded 125 kbps PHY is the second viable option. The most notable distinction between the two papers is our use of a real-world testbed for evaluation, whereas their work relies on simulation. By employing a testbed, we assess performance under realistic conditions, accounting for the physical characteristics of BLE and its performance in environments with interfering communication beyond our control. These divergent results leave room to further investigate the differences in performance between these two bodies of work.

7 Conclusion and Emerging Directions

In this thesis, we argue that future low-power IoT devices will face increasing challenges due to unforeseen interference from other devices and networks. Therefore, future communication protocols must dynamically adjust retransmissions and routing decisions to withstand interference and ensure long-term stable and reliable communication. Additionally, to reduce latency in multi-hop low-power wireless communication systems, we must integrate aspects from different communication approaches into novel protocols. Furthermore, future IoT devices will be able to adapt their functionality based on their surrounding environment. Hence, we require methods to recognize and adapt to changes in the device's environment.

This thesis introduces five protocols, systems, and evaluation infrastructures for modern IoT solutions. We demonstrate that flow-based retransmissions (*Master*) and opportunistic routing with concurrent forwarding (*Autobahn*) add dynamics to overcome unforeseen interference in otherwise static and centrally scheduled TSCH networks. These solutions enable long-term stable communication without frequent adjustments to routing and scheduling. Both effectively reduce latency to a minimum. To further reduce latency, we introduce *TBLE*, which combines TSCH with Bluetooth Low Energy (BLE) PHYs, replacing the IEEE 802.15.4 PHY. To ensure the intended functionality of these protocols and gain insights during evaluation, we develop a retrofittable time-synchronized GPIO tracing solution for testbeds (*Grace*). Lastly, we introduce *BlueSeer*, which uses embedded machine learning to demonstrate that ambient BLE signals are sufficient to recognize the current environment.

Based on our work, we identify several directions for future research. Firstly, with TBLE, we enable TSCH to operate successfully on five 2.4 GHz PHYs—IEEE 802.15.4 and the four BLE PHYs. As modern radios support all of these, scheduling multi-PHY TSCH networks in the 2.4 GHz band suggests promising prospects to route and schedule communication even more efficiently in a congested wireless medium. Secondly, efficient routing and retransmission schemes for reliable multi-hop communication as demonstrated in Master and Autobahn could potentially be extended to other wireless communication technologies. While LPWAN technologies like LoRa and NB-IoT already cover larger areas and longer distances, time-slotted multi-hop extensions could enhance coverage in currently underserved areas, albeit with significant time-synchronization overhead. Additionally, there is growing potential in further exploring the combination of IoT and AI. With BlueSeer, we take a first step

into the direction of using machine learning in combination with IoT devices. With the general trend in machine learning and continuously advancing machine learning approaches, several directions open up for research projects or potential future theses. We see the potential to revisit multi-hop wireless communication and use machine learning to develop better scheduling and routing algorithms, also considering multi-PHY approaches with more degrees of freedom than in single-PHY networks. Yet, we see the bigger research potential in improving on-device and resource-constrained machine learning frameworks. These could enable devices to evolve with application or environment changes without external training or major firmware updates, while developing and tuning new neural networks for resource-constrained devices could enhance smart features in IoT devices.

A

Master: Long-Term Stable Routing and Scheduling in Low-Power Wireless Networks

---

Laura Harms, Olaf Landsiedel

Proceedings of the 16th International Conference on Distributed Computing in Sensor Systems (DCOSS), 2020, pp. 86–94.

Abstract

Wireless Sensor-Actuator Networks (WSANs) are an important driver for the Industrial Internet of Things (IIoT) as they easily retrofit existing industrial infrastructure. Industrial applications require these networks to provide stable communication with high reliability and guaranteed low latency. A common way is using a central scheduler to plan transmissions and routes so that all packets are delivered before a deadline. However, existing centralized schedulers are only able to achieve high reliability in the absence of interference. This limitation lowers the feasibility of using centralized schedulers in most environments susceptible to interference.

This paper addresses the challenge of stable, centrally scheduled communication in low-power wireless networks susceptible to interference. We introduce Master, a centralized scheduler and router, for IEEE 802.15.4 TSCH (Time-Slotted Channel Hopping). Master uses Sliding Windows, a novel transmission strategy, which builds on flow-based retransmissions instead of link-based ones. We show in our experimental evaluation that Master with Sliding Windows achieves routing and scheduling stability for over 24 hours with end-to-end reliability of over 99.6%. Moreover, we show that Master outperforms Orchestra, a state-of-the-art autonomous scheduler, in terms of latency by a factor of 8 while achieving similar reliability under a slight duty-cycle increase.

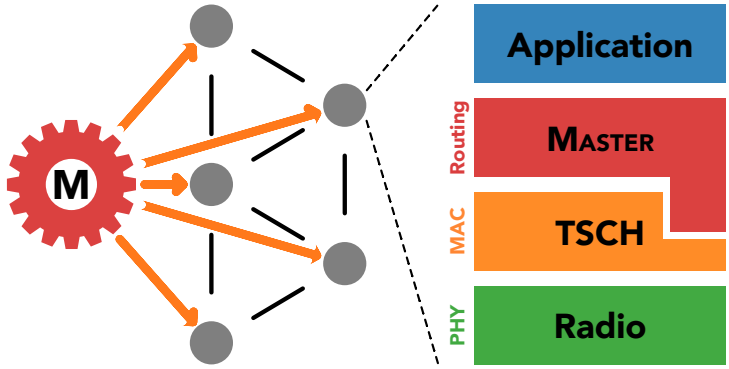


Figure A.1: Master consists of an external centralized scheduler (M) and a routing layer. The external scheduler performs the global routing and scheduling and pushes the computed schedule onto the network. In each node, Master's routing layer implements the schedule in TSCH and performs the routing during runtime.

1 Introduction

For many applications in the Industrial Internet of Things (IIoT), it is essential that network traffic meets deadlines. To achieve this goal, commonly, a centralized scheduler collects information about the network topology and the wireless links. With this global knowledge, representing a major advantage over distributed solutions, the scheduler is able to compute optimal routes and transmission schedules of end-to-end communication (traffic flows). In IEEE 802.15.4, the scheduler assigns communication slots in the time and frequency domain to nodes, i.e., it employs Time-Slotted Channel Hopping (TSCH) [6]. However, due to wireless link dynamics, centralized schedulers have to account for the risk of packet losses and, therefore, usually include multiple retransmission slots for each link. These retransmission slots increase latency and reduce the available bandwidth, thus, causing an increased radio on-time.

Many recent centralized scheduling algorithms assume the availability of interference-free channels or at least a static amount of interference [10, 16]. These assumptions do not hold in many of today's environments where IEEE 802.15.4 IIoT networks co-exist with an increasingly large number of WiFi and Bluetooth networks. This coexistence results in large amounts of interference and thereby limits the stability and reliability of those centralized solutions.

In this paper, we introduce MASTER, a centralized scheduler designed for TSCH. It combines the traditional steps of central scheduling and routing with a novel transmission strategy which we call Sliding Windows. Our Sliding Windows algorithm introduces the flexibility needed to accomplish long-term schedule stability and communication reliability while meeting the latency requirements of industrial applications. As a result, MASTER enables long-term stable schedules and thereby eliminates the need for frequent rescheduling, a key drawback of today's central schedulers. Furthermore, we design MASTER

as an open[1] and easily extendable platform to foster rapid experimentation with central scheduling policies.

Our evaluation shows that MASTER with Sliding Windows outperforms slot-based retransmission strategies of centralized schedulers. Moreover, it outperforms the low-power autonomous scheduler Orchestra [8] in terms of latency while achieving similar reliability and consuming not significantly more energy, making it particularly suitable for low-power systems. Overall, this paper makes the following contributions:

- We present MASTER, an open-source, centralized router and scheduler for TSCH-based networks designed with easy extendability in mind.
- We design Sliding Windows, a transmission strategy for MASTER to increase the flexibility, stability, and reliability of centrally scheduled communications.
- We propose flow-based queues as an extension to TSCH to enable the use of central scheduling algorithms.
- We implement MASTER as part of Contiki-NG and evaluate it in environments susceptible to interference. We show the long-term stability of schedules computed by MASTER in experiments of 24 hours. These experiments result in highly reliable (>99.6%), low-latency (<4.5 slots) communications.

The remainder of this paper is organized as follows. Section 2 gives the necessary background information on TSCH as well as TSCH schedulers. Section 3 introduces the design of MASTER, and Section 4 presents our testbed evaluation. Section 5 reviews related work, followed by the conclusion in Section 6.

2 Background

This section gives an overview of relevant concepts on (A) Time-Slotted Channel Hopping (TSCH), (B) the ETX metric, (C) scheduling, and (D) retransmissions.

2.1 Time-Slotted Channel Hopping

Time-Slotted Channel Hopping (TSCH) is one of the MAC-layer protocols defined in the IEEE 802.15.4e standard [6]. TSCH uses dedicated time- and frequency-slots (TDMA and FDMA) for accessing the wireless medium. These slots are standardized to a length of 10 ms, and each slot uses one out of maximally 16 channels. TSCH continuously cycles through a hopping sequence of all active channels. Thus, it is changing the channel every slot. Assigning different frequencies to slots allows TSCH to increase the network's resilience to interference. Slots dedicated to control-information, so-called Enhanced Beacon (EB) slots, provide broadcasts which support both network formation and time

[1] Available as open-source at: `https://github.com/ds-kiel/master-scheduler`

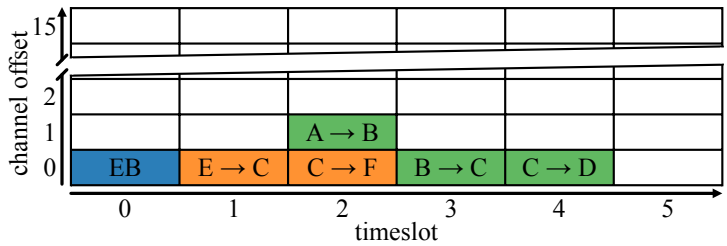


Figure A.2: Sample TSCH schedule. Slot 0 is a shared slot for sending and receiving Enhanced Beacons (EB) while slots 1-4 are unicast slots with one transmission per channel at a time. This simple schedule contains two multi-hop communication flows, highlighted in green and orange. The channel offset is added on top of the usual hopping sequence.

synchronization, both essential for maintaining a schedule of synchronized transmissions as in TSCH.

Multiple TSCH slots are grouped into slotframes, and multiple slotframes form a TSCH schedule, see Figure A.2. Each node has a custom TSCH schedule determining its behavior in each slot. Slots are either dedicated, shared, or empty: In a dedicated slot, a node either transmits or receives. In shared slots, nodes may broadcast or receive control information, such as Enhanced Beacons. Such slots are not assigned to individual nodes and have multiple nodes contending for transmissions. To limit collisions, these slots employ the CSMA-CA back-off algorithm. If a slot is neither dedicated nor shared, it is empty, and the radio remains off to save energy.

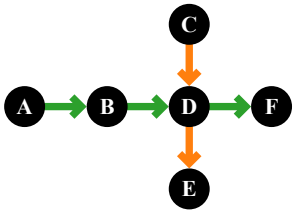
2.2 Link quality metric

Link quality metrics, such as the expected transmission count, ETX [48], represent the quality of a wireless link. ETX specifies the number of transmissions expected to transmit a packet successfully over a wireless link. The ETX value is the inverse of the packet reception rate (PRR) of a link ($ETX = 1/PRR$).

2.3 Scheduling

In the context of wireless communications, scheduling is the process of allocating resources for communications to meet all requirements such as release-time and deadline. Scheduling is an NP-hard problem, meaning, it is not optimally solvable in polynomial time (cf. [10]). Therefore, different heuristics and algorithms were developed to solve scheduling problems sufficiently well for specific scenarios.

TSCH scheduling: TSCH does not specify how communications are scheduled. Therefore, scheduling TSCH communications can be performed in a centralized, distributed, or autonomous manner. In the distributed case, subsets of the network perform cooperative scheduling (cf. 6TiSCH MSF [134]). The autonomous case, used by the well-known TSCH scheduler Orchestra [8], performs an autonomous mapping of links to resources. The centralized schedul-



(a) Sample Topology with two intersecting flows.

	1	2	3	4
A		TX		
B		RX	TX	
C	TX			
D	RX	TX	RX	TX
E		RX		
F				RX

(b) Baseline schedule without retransmissions.

	1	2	3	4	5	6	7	8
A			TX	TX				
B			RX	RX	TX	TX		
C	TX	TX						
D	RX	RX	TX	TX	RX	RX	TX	TX
E			RX	RX				
F							RX	RX

(c) A slot-based schedule with one transmission and one retransmission slot per hop.

	1	2	3	4	5	6	7	8	9
A				TX	TX	TX	TX		
B				RX	RXTX	RXTX	RXTX	TX	
C	TX	TX	TX						
D	RX	RXTX	RXTX	TX	RX	RXTX	RXTX	RXTX	TX
E		RX	RX	RX					
F						RX	RX	RX	RX

(d) Sliding Windows schedule with one transmission slot per hop and two (orange flow) or three (green flow) retransmission slots to be shared among the nodes of a flow.

Figure A.3: Example: One flow originates at node A to end at node F while the second one originates at node C and ends at node E.

ing approach provides us with global topology knowledge, and we can allocate resources using established algorithms such as Dijkstra's Shortest Path First algorithm [49].

2.4 Retransmissions

As wireless communication links are unreliable, transmissions are never guaranteed to be received. To increase the reliability, schedulers commonly include retransmission slots to retry a failed transmission. A common way of adding retransmissions is the duplication of single slots. This slot-based approach increases the reliability, by including multiple tries per hop. In this paper, we introduce a new, flow-based transmission strategy to increase both performance and flexibility, see Section 3.2.

3 Design

In this section, we present the design of Master, our transmission strategy Sliding Windows, and the system architecture of Master.

3.1 Centralized Routing and Scheduling with Master

A fundamental building block of Master is its centralized scheduler. Its design is a three-step process to build a long-term stable, low-latency, reliable

communication schedule. This process is a sequential top-down approach of (1) centralized routing, (2) applying a transmission strategy, and (3) scheduling. The input to the process is (a) a set of traffic flows specified by source, destination, periodicity, and deadline, as well as (b) the network topology with long-term link reliability statistics. The application commonly provides the set of flows, and we derive the network topology from long-term link measurements, see Section 3.4.5.

3.1.1 Centralized Routing

Routing is the first step in MASTER and uses the previously specified flows and network link-reliability as input. To perform the routing, MASTER constructs a directed weighted graph using an ETX-based metric (ETX^n, $n \in \mathbb{N}$, usually $n = 2$), corresponding to the link reliability statistics. A higher ETX-power favors a higher number of highly reliable links over a lower number of links with lower reliability. Using this graph, we compute the shortest end-to-end routes. As shortest path routing finds the optimal path for each flow, the flow latency selected by the routing process stays minimal. The result of our routing is an extended set of flows that consists of a source, a destination, and the intermediate hops. In MASTER, we use Dijkstra's algorithm for shortest-path routing, but our modular design allows us to plug-in any routing algorithm and metric.

3.1.2 Transmission Strategies

After computing the route for each flow, we employ a transmission strategy to ensure reliable communication over unreliable wireless links. Thus, the transmission strategy adds retransmission slots to each flow to handle failed transmissions due to link dynamics and interference. The transmission strategy extends each flow by a specific number of slots. In the case of highly reliable links in an interference-free environment, we can employ a simple transmission strategy of assigning only one slot per hop. In practice, however, we add retransmission slots according to the expected link reliability of each hop. We employ either a slot-based transmission strategy (see Section 2.4) or our new approach of a flow-based transmission strategy (see Section 3.2 below).

3.1.3 Scheduling

After applying one of the transmission strategies, we pass the modified flows to the scheduler. The scheduler builds a communication schedule for all flows considering their periodicity.

For our application scenarios and to be comparable to Orchestra, we employ a non-deadline-based scheduling algorithm. It is especially suitable for best-effort, periodic, deadline-free systems. The algorithm is *Reverse Longest Path First (R-LPF)*, our own flavor of the *Shortest Path First (SPF)* scheduling algorithm. SPF is based on the process scheduling algorithm Shortest Job First (SJF) [255]. Contrary to starting with the shortest flow, our scheduler performs backward scheduling, starting with the end of the longest flow. This modification of the scheduling algorithm results, in our experience, in a lower number

of unused slots within a flow. A lower number of unused slots corresponds with lower latency.

Figure A.3 shows a schedule for two flows generated using no retransmissions, a slot-based retransmission strategy, as well as the transmission strategy of Sliding Windows with a transmission number based on Equation (A.3) and a scaling factor of 1. To generate the schedule of Figure A.3d, we assume the ETX-value of each link to be between 1 and 2 ($ETX_{link} \in]1, 2[$).

Any scheduling algorithm, including deadline-based ones, can easily be implemented in MASTER. For the remainder of this paper, we use R-LPF.

3.2 Master's Flow-based transmission strategy

Our flow-based transmission strategy assigns a specific number of retransmissions to a flow instead of using a per-hop basis, as done traditionally, see Section 2.4. The flow-based retransmission slots allow the nodes of a flow to share these slots and use them as needed along the path, see Figure A.3d. As a result, we can increase the communication reliability while potentially using minimally more slots in the final schedule (see Node D in Figure A.3c and Figure A.3d).

With this, we divert from the traditional scheme of two active nodes to one with multiple active nodes: Traditionally, at a single time-slot, frequency, and within a localized area, only one node transmits and another one receives. Instead, we now have more than two nodes awake that either transmit or receive. Our transmission strategy has the advantage of being adaptable to network changes, e.g., due to interference. Thus, during the journey of a packet, we can use the shared transmission slots in whichever part of the flow interference impacts communication. This adaptability is traditionally possible within distributed schedulers that can locally adapt to link changes. With Sliding Windows, we now enable such flexibility in centralized ones.

3.2.1 Window Size

The maximal number of transmission slots (TX_{max}, later denoted as $\#transmissions$) in a flow and the hop-count of the flow determine the window size which is calculated by

$$window\_size = 2 + TX_{max} - hops \tag{A.1}$$

This window size is the number of nodes maximally active in a slot of a flow. Moreover, it matches the maximum number of active slots of a node for a given flow. According to this relation, the window size is equal to the shared number of slots of a node for transmission or reception ($TX_{max} - hops$) plus its first and last slot allocated for reception and transmission, respectively.

In MASTER, we have two flow-based transmission policies: (1) fixed window size and (2) metric-based window size. For the first policy, we use the same window size for all flows independent of their length or link quality. For the second one, our scheduler determines the window size and number of transmissions depending on the flow's or link's ETX-values. The metric-based window size allows us to account for both the number of hops and the reliability of the individual links.

Using the link's ETX values, we can calculate the total number of transmissions of the flow with either

$$\#transmissions = n * \lceil \sum ETX_{link} \rceil, \quad n \in \mathbb{N} \tag{A.2}$$

or

$$\#transmissions = n * \sum \lceil ETX_{link} \rceil, \quad n \in \mathbb{N}, \tag{A.3}$$

including a scaling factor n. This scaling factor regulates the conservativeness of the scheduler. If we choose a scaling factor of 1 for Equation (A.3), the number of transmissions is equal to the one using an ETX-based, slot-based retransmission strategy (cf. Section 2.4). Equation (A.2) uses the end-to-end ETX-value of the flow, while Equation (A.3) uses the ETX-values of the individual links.

Throughout the remainder of this paper, we use the following naming scheme to refer back to these equations:

$$SW - < Equation\ number > [- < scaling\ factor\ n >]$$

SW denotes it as a Sliding Windows transmission strategy. The naming scheme includes the scaling factor only if referring to a specific representation of the strategy. When referring to the general strategy, it is not included.

Please note that for long flows, i.e., with many hops such a strategy could lead to a large window, and thereby too many nodes being awake at the same point in time. Too many active nodes lead to inefficiencies, and we counterbalance it by splitting a flow into sub-flows once it exceeds a limit N. The flow-based strategy is then applied to each sub-flow individually. In MASTER, we use a threshold of $N = 10$. Thus, for example, a flow of length 11 is split into two overlapping sub-flows of length 6.

3.2.2 Algorithm

In Algorithm A.1, we present the algorithm for applying a flow-based transmission strategy. The algorithm takes as input a flow consisting of multiple nodes, the network's ETX graph, the strategy (SW-2 or SW-3), and the scaling factor. The algorithm starts calculating the flow's total ETX cost, as well as the flow's number of transmissions according to the given strategy (SW-2 or SW-3) and the window size according to Equation (A.1). From line 17 onward, the algorithm computes the active slots for each node of the flow and inserts the nodes into the respective slots of the new flow. For example, slot 6 of Figure A.3d would be represented in the new flow as a list containing the elements A, B, D, F in this order.

3.2.3 Flow-based transmissions vs. Flow Centric Policy (FCP)

Recently, a paper by Brummet et al. [131] introduced a similar idea of moving from link-based to flow-based transmissions.

The main difference between Brummet's proposed Flow Centric Policy (FCP) and our Sliding Windows strategy are the rules for determining the optimal number of flow transmissions. FCP only defines fixed numbers of retransmissions with a maximum of up to 4 retransmissions for a flow. Sliding

Algorithm A.1 Sliding Windows transmission strategy

Input: $flow$, $graph_{ETX}$, strategy, scaling factor n
Output: $flow_{new}$ (modified version of $flow$)
1: $cost_{total} = 0$
2: **for** $i = 0$ to $length_{flow} - 1$ **do**
3: $sender_{hop} \leftarrow flow[i]$
4: $receiver_{hop} \leftarrow flow[i + 1]$
5: **if** $strategy =$ "$SW - 2$" **then**
6: $cost_{total} = cost_{total} + graph_{ETX}[sender_{hop}][receiver_{hop}]$
7: **else if** $strategy =$ "$SW - 3$" **then**
8: $cost_{total} = cost_{total} + \lceil graph_{ETX}[sender_{hop}][receiver_{hop}] \rceil$
9: **end if**
10: **end for**
11: **if** $strategy =$ "$SW - 2$" **then**
12: $cost_{total} = \lceil cost_{total} \rceil$
13: **end if**
14: $\#transmissions \leftarrow n * cost_{total}$
15: $window\_size \leftarrow 2 + \#transmissions - length_{flow}$
16: $flow_{new} \leftarrow$ list of $\#transmissions$ lists
17: **for** $i = 0$ to $length_{flow} - 1$ **do**
18: **if** $i = 0$ **then**
19: $slots \leftarrow$ list $[0 \,..\, window\_size - 1]$
20: **else if** $i = (length_{flow} - 1)$ **then**
21: $slots \leftarrow$ list $[i - 1 \,..\, i + window\_size - 2]$
22: **else**
23: $slots \leftarrow$ list $[i - 1 \,..\, window\_size - 1]$
24: **end if**
25: **for** $slot$ in $slots$ **do**
26: extend $flow_{new}[slot]$ by $flow[i]$
27: **end for**
28: **end for**
29: **return** $flow_{new}$

Windows, on the other hand, allows choosing the number of transmissions based on a metric, in our case, the ETX metric. Moreover, Sliding Windows allows a different number of transmissions for each flow in the same network due to its use of the ETX metric. Because Sliding Windows is based on link qualities, we argue that it offers better adaptability to a network's link characteristics during the scheduling process.

3.3 Time Synchronization

Stable time synchronization is essential for TSCH networks. It ensures that clocks do not drift apart, and nodes wake-up for transmissions and reception within the guard times specified by TSCH. Master achieves this by building a clock synchronization tree from the root as part of the scheduling process. Similar to the routing of the flows, a minimal spanning tree with ETX as metric and with the coordinator of the TSCH network as root is computed using Dijkstra's algorithm. This tree assigns each node a parent node for clock synchronization.

3.4 System Design

Next, we detail on the system architecture of Master. It consists of both the external scheduler and the routing layer on each node (see Figure A.1). Here we put a particular focus on the integration with TSCH and Contiki-NG [256].

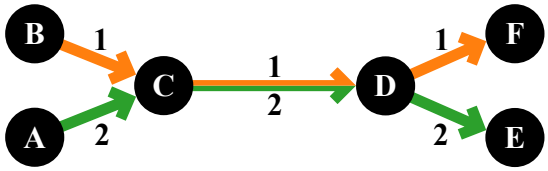


Figure A.4: Example of 2 flows sharing a common link between nodes C and D.

3.4.1 Central Logic of Master

The central logic of MASTER consists of a centralized router and scheduler with all the functionality described above. We implement MASTER in Python to enable easy extendability and rapid experimentation of new routing, transmission, and scheduling strategies.

3.4.2 Schedule Distribution

For schedule distribution, MASTER can work together with most schedule distributors (e.g., plexi [159]), as scheduling and distributing the schedule are orthogonal. Moreover, it can also directly upload schedules via the serial port for rapid experimentation.

3.4.3 Per node routing layer

The routing layer of MASTER has multiple functions: it performs neighbor discovery (Section 3.4.5), implements the schedule, and adds a routing header to the communication payload to be compliant with the lower layers as well as relaying the packet to the next hop (Section 3.4.6). We place it in the Contiki-NG network stack above TSCH, see Figure A.1, and implement it in C.

3.4.4 Contiki-NG/TSCH Extensions

To match the requirements of MASTER and its scheduling algorithm, we extend the elements of TSCH and its implementation in Contiki-NG: (1) the packet buffer implementation and (2) the TSCH queues.

In the packet buffer, we add fields to store the flow identifier and the time to live of a transmission. With these two fields, the TSCH stack and MASTER can map incoming packets to flows and thereby follow the global schedule on each node. We extend the TSCH queue to enable a transmission order differing from the reception order at a node, e.g., the forwarding of a packet to a specific neighbor before forwarding an earlier received packet to the same neighbor. To allow this behavior, we add flow-based queues, in addition to the neighbor-based queues of TSCH. We realize the flow-based queues through the use of virtual neighbors.

Figure A.4 illustrates why neighbor-based queues as used by Contiki-NG cannot be practicably used by MASTER. If packet 2 is received by node C first, but packet 1 has an earlier deadline, packet 1 will be stuck behind packet 2 until the first is transmitted to node D. With flow-based queues, packets 1 and

2 will be added to different queues at C. Therefore, they are independent of each other and packet 1 can be forwarded first.

This new queue design increases the schedulability of the presented scheduler, which is crucial for deadline-dependent systems. It also decreases the latency in networks that are not deadline-critical by reducing congestion at bottlenecks of the network. Moreover, it allows us to use scheduling algorithms initially developed for process scheduling, a domain without these congestion problems.

3.4.5 Neighbor Discovery and Bootstrapping

Before MASTER can build any schedule, it requires information about all links between the nodes in the network. Thus, to bootstrap and collect topology information with MASTER, we deploy a custom, topology agnostic schedule only designed for neighbor discovery. In this schedule, we use one independent transmission slot per node present in the network. This neighbor discovery schedule is similar to the sender-based operation mode of the autonomous scheduler Orchestra [8]. Each node sends a numbered broadcast in its active slot and listens in all other slots for broadcasts of other nodes in its surroundings.

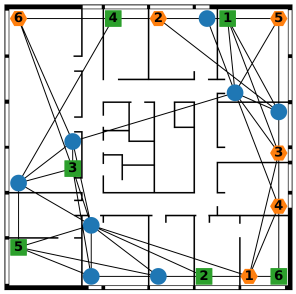
Please note that this schedule only serves for bootstrapping. After deployment of the actual transmission schedule, the task of probing neighbors becomes part of the normal TSCH beaconing process. Nodes collect this information for any potential later update of the schedule.

3.4.6 Header format

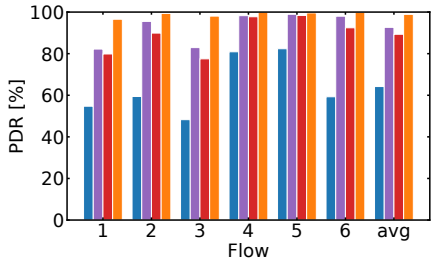
MASTER routes packets based on flows, and as a result, we add a custom routing header. The routing layer of MASTER adds a 7-byte routing header to each packet. This header contains a flow identifier (1 byte), a sequence number (2 bytes), the time-to-live (TTL) (2 bytes), and the earliest TSCH transmission slot (2 bytes). The header is necessary for nodes to know whether they are the receiver of the packet or a forwarder. Moreover, the header specifies, where to forward the packet to, and whether there is still time left for forwarding. In practice, our header replaces the IPv6 header which we could use instead in a system using the full IPv6 stack.

4 Evaluation

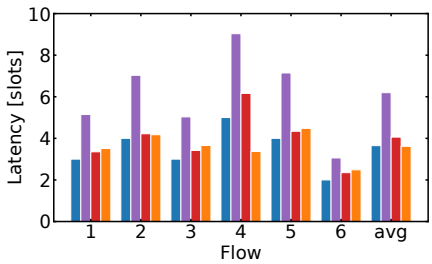
In this section, we evaluate the performance of MASTER and compare it to the state-of-the-art. We begin by evaluating our newly proposed flow-based scheduling policy and compare it to state-of-the-art scheduling policies, including a baseline strategy without retransmissions (cf. TASA [11]) and a slot-based transmission strategy (cf. AMUS [119]). Next, we compare MASTER to Orchestra, the default autonomous scheduler in Contiki-NG, which also builds on TSCH. Finally, we evaluate MASTER's ability to compose long-term stable schedules.



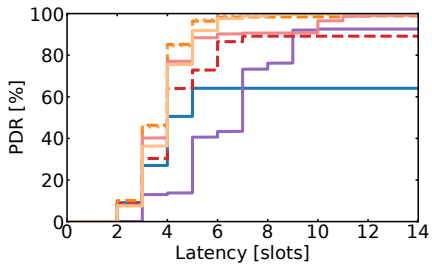
(a) 500 m^2 testbed of 20 nodes at Kiel University. Source nodes: orange hexagons; Sink nodes: green squares; Relay-only nodes: blue circles; Numbers: corresponding flow



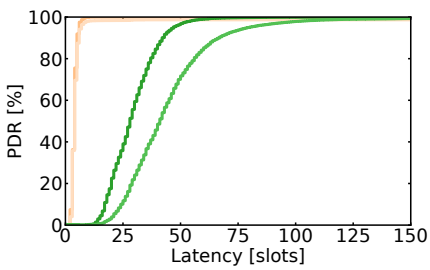
(b) Reliability of Master's transmission strategies: baseline, slot-based, SW-2-1 and SW-3-1.



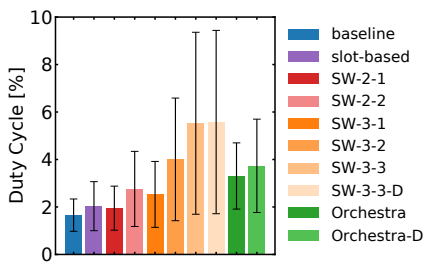
(c) Latency of Master's transmission strategies: baseline, slot-based, SW-2-1 and SW-3-1.



(d) Combined latency and reliability CDF of Master's transmission strategies.



(e) Combined latency and reliability CDF of Master's transmission strategy SW-3-3 and Orchestra at nighttime and daytime.



(f) Duty cycle of Master and Orchestra.

Figure A.5: Evaluation of Master's transmission strategies and comparison to Orchestra. SW-3 outperforms all other strategies reliability-wise and outperforms Orchestra latency-wise. We display the legend of figures A.5b - A.5f in Figure A.5f.

4.1 Evaluation Setup

4.1.1 Testbed

We run on a 20 node testbed deployed in offices and student lab rooms, see Figure A.5a. It is located on the top most floor of a university building with spanning an area of 500 m^2. The testbed shares the wireless spectrum with WiFi and Bluetooth communications outside of our control. Due to this, the testbed is exposed to high levels of interference, especially during work hours.

4.1.2 Metrics, Comparison, and Duration

We evaluate our scheduler in terms of end-to-end reliability, end-to-end latency, as well as network energy consumption. We measure these metrics for different centralized scheduling approaches with and without retransmissions. Moreover, we compare our scheduler with the autonomous scheduler Orchestra [8]. These comparisons are based on 2-hour experiments for each strategy, except for the long-term stability evaluation in Section 4.5, which has a duration of 24-hours per experiment.

4.1.3 Implementation

We implement MASTER for Contiki-NG [256]. We target the Zoul Firefly platform, featuring a 32 MHz 32-bit CC2538 Cortex-M3 CPU, 32 KB of RAM, 512 KB of flash, with an IEEE 802.15.4 compatible radio.

4.1.4 Channels

Due to the high levels of interference, we use only the four channels (15, 20, 25, and 26), defined in the standard four-channel TSCH hopping sequence. Furthermore, Orchestra uses by default only these four channels as well.

4.1.5 Application Payload and Overhead

For all experiments, we include a 64-byte randomly generated data payload, a medium packet size supported by TSCH. In addition to this data payload, MASTER adds its 7-byte routing header independent of the specific scheduling policy. Orchestra, on the other hand, uses the IPv6 headers and requires additional network layer control traffic.

4.1.6 Notations

Throughout the evaluation, we use the following naming scheme: The baseline strategy without retransmissions we call *baseline*, and the slot-based retransmission strategy (as used by many state-of-art schedulers) with $\lceil ETX_{link} \rceil$ transmissions per link we label *slot-based*. The Sliding Windows strategies use the naming scheme we present in Section 3.2.1. Experiments performed during daytime are extended by the marker *-D*.

4.2 Baselines

We compare Master's Sliding Windows policies to three other scheduling policies. These are Master's baseline strategy without retransmissions, Master's slot-based retransmission strategy, and the autonomous scheduler Orchestra [8]. The design of the baseline strategy is based on the transmission policy used in, e.g., TASA [11], and uses one distinct slot per hop. The slot-based strategy is inspired by policies presented in several recent publications, including AMUS [119]. Contrary to most of these, our design performs all possible retransmissions of a hop before proceeding to the next hop, which favors high reliability over low latency contrary to AMUS's approach. Moreover, to be in line with our Sliding Windows strategies, Master's slot-based strategy uses an ETX-based number of retransmissions per link ($\lceil ETX_{link} \rceil$). Lastly, we use Orchestra to compare our centralized routing and scheduling solution to distributedly routed and autonomously scheduled solutions to verify the adaptability of Master to dynamic environments predestined for distributed policies.

4.3 Performance of Master's transmission strategies

We first evaluate the performance of different transmission strategies supported by our scheduler. We compare the Sliding Windows transmission strategy with a baseline strategy without retransmissions and with the traditional slot-based retransmission strategy mentioned above. We run experiments with six scheduled flows, a number of flows used at a recent EWSN dependability competition [257]. The flows have a length of 2 to 4 hops each. Each flow has a sole source and destination node. Each source node generates a packet roughly every second with a configured time to live of one second. The length of the communication slotframes of 1 second corresponds roughly with 101 slots.

Figure A.5b shows the reliability of transmission approaches scheduled with Master. The transmission approaches include the baseline and slot-based strategy, as well as Sliding Windows transmission strategies SW-2-1 and SW-3-1; see Section 3.2.1 for notations. The latter of the two Sliding Windows strategies has the same number of transmissions per flow as the slot-based strategy.

All strategies with retransmissions clearly outperform the baseline without retransmissions, which shows the presence of interference in the used channels. The slot-based strategy reaches an average reliability of 92.7% whereas the Sliding Windows strategies reach average reliabilities of 89.3% and 98.9%, respectively. The SW-2-1 strategy has for all flows lower reliability than the slot-based strategy, but the number of scheduled slots per flow is only by one larger than the baseline number of slots, see Table A.1. The SW-3-1 strategy outperforms all other strategies while using no more slots per flow than the slot-based strategy. Its least reliable flow achieves a packet delivery rate (PDR) of 98.1% while the slot-based strategy drops as low as 82.2%.

We can model this superiority of SW-3-1 over SW-2-1 and over the other strategies mathematically using the probability mass function of the binomial distribution [258]:

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k} \tag{A.4}$$

Table A.1: Summary of the results plotted in Figure A.5c: Maximum latency (slots) for each flow and for flow 4 maximum number slots active in parentheses.

Flow	Baseline	Slot-Based	SW-2-1	SW-3-1
1	2	4	3	4
2	3	6	4	6
3	3	6	4	6
4	5 (3)	10 (6)	7 (4)	12 (6)
5	4	8	5	8
6	4	8	5	8

This probabilistic model also explains the lower reliability of SW-2-1 compared to the slot-based strategy.

As an example, we consider a flow of three hops ($n = 3$), e.g., the green flow in Figure A.3a, with the same ETX value for each link of 1.2 ($p = \frac{5}{6}$). Thus, the number of transmissions for SW-2-1 and SW-3-1 are 4 and 6 slots, respectively. The expected PDRs for SW-2-1 and SW-3-1 are $P(X = 3) + P(X = 4) \approx 0.868$ and $P(X = 3) + P(X = 4) + P(X = 5) + P(X = 6) \approx 0.991$, respectively. Likewise, the expected PDR for the baseline, is $P(X = 3) \approx 0.579$. The slot-based strategy can be seen as 3 independent, subsequent chains of two binomial trials each ($n = 2, k \geq 1$). This results in an expected PDR of $(P(X = 1) + P(X = 2))^3 = 0.919$. These mathematical results confirm the trend we see in Figure A.5b.

Latency-wise, both Sliding Windows strategies perform much better than the slot-based strategy. Moreover, their latency is minimally higher than the latency of a strategy without retransmissions (see Figure A.5c), which, in turn, has a high packet loss rate. It appears that SW-3-1 has a lower latency for flow 4 than the baseline. Contrary to all other flows, flow 4's schedule contains more slots than active slots throughout all strategies. Due to the flow-based approach of SW-3-1 and a large enough number of continuous active slots at the beginning of the schedule, most packets were received within a few slots, leading to a latency lower than the baseline's one. Table A.1 shows that the maximal number of active slots is still smaller for the baseline strategy.

Figure A.5d visualizes the latency and reliability of a wider range of transmission strategies. Solid lines represent the baseline, the slot-based, the SW-2-2, and the SW-3-3 strategies. For the Sliding Windows strategies SW-2-1 and SW-3-1, the figure uses dashed lines, and for the SW-3-2 strategy, it uses a dotted line. The figure shows that the slot-based strategy is the worst latency-wise. The SW-3 Sliding Windows strategies are superior to the other Sliding Windows strategies (SW-2). The superior strategies with a scaling factor of 2 and 3, both perform well. The strategy with the higher scaling factor reaches the maximal possible reliability. Therefore, we use the Sliding Windows strategy SW-3-3 for the following comparison to Orchestra.

The duty-cycle evaluation in Figure A.5f shows a higher radio on-time for a higher number of scheduled slots. SW-3-3 has a radio on-time of up to 11.95% for a node with a lot of traffic.

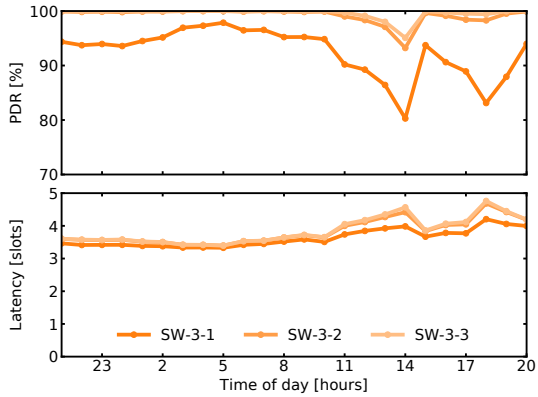


Figure A.6: Reliability and latency evaluation of Sliding Windows according to Equation (A.3) for all 3 scaling factors. Each value corresponds with the hour, that started at the given time. Note, that the y-axis of the PDR plot does not begin at zero.

4.4 Master vs. Orchestra

We now evaluate the performance of Master in comparison to Orchestra, the default, autonomous scheduler of TSCH in Contiki-NG. We use Orchestra as is, with a receiver-based schedule of length 7 in non-storing mode. We schedule the same six flows used before. As transmission strategy for Master, we use the one with the highest reliability of those presented above (SW-3-3). To provide detailed information on the performance, we present runs of both Master and Orchestra during nighttime as well as during office hours in the daytime. Figure A.5e shows the latency and reliability of the four experiments. Master's latency is drastically shorter than the latency of Orchestra with a mean latency of 3.9 and 4.2 slots compared to 25.9 and 40.9 slots during nighttime and daytime, respectively, while reaching similar reliability. The four rightmost columns in Figure A.5f show the duty cycle for the experiments included in this section of the evaluation. Orchestra has on average a two percentage points lower duty cycle than Master (3.52% vs. 5.55%) and the maximum duty cycle of a node of four percentage points lower (7.73% vs. 11.95%). As each node in Orchestra is only able to use every seventh slot, the possible duty cycle is automatically lower than the one for Master. However, this lower duty cycle results in much higher latency, as presented above.

4.5 Long-term stability of Master

In the last part of our evaluation, we investigate Master's long-term stability. In Figure A.6, we present the reliability and latency of the SW-3 Sliding Windows strategies for 24 hours (Day 1, 21:00 - Day 2, 21:00) during workdays. During the night and the early morning, both SW-3-2 and SW-3-3 reach a PDR of above 99.99% and an average latency of around 3.5 slots. Between 14:00 and 15:00, the reliability drops for all strategies to 95%, 93.2%, and 80.3%, respectively, under a slight latency increase. During this time, a group of students entered the lab, leading to a drastic increase in WiFi and BLE

traffic and thereby an interference level increase. Another reliability drop, mainly for SW-3-1, is visible at the end of the working day. Over the whole period of 24 hours, the average reliability of SW-3-1, SW-3-2, and SW-3-3 is 99.6%, 99.2%, and 92.5%, respectively. The high average reliability, as well as the reliability recovery after times of high interference, validates MASTER's long-term stability.

5 Related Work

We first discuss centralized schedulers and algorithms, followed by a discussion of autonomous scheduling solutions.

After the introduction of TSCH, TASA [11] was one of the first central scheduling algorithms proposed. It is traffic aware, yet like other papers focusing on scheduling algorithms like C-LLF [10], it assumes the availability of interference-free channels and, therefore, does not include retransmissions. Saifullah et al. [10] and Gunatilaka et al. [16] focus in their work on the highest possible schedulability for a large amount of communications meeting deadlines but not much on the network reliability. AMUS [119] is one of the protocols for TSCH that includes slot-based retransmissions. It schedules additional resources for vulnerable links and allocates backup slots in empty cells of the scheduler. Rugamba et al. [125] build another centralized scheduler based on a path collision-aware least-laxity first scheduling algorithm by Darbandi et al. [124]. Moreover, Rugamba et al. describe a method of distributing a centrally computed schedule. The first approach of moving from slot-based retransmissions to flow-based ones is the flow-centric policy (FCP) [131]. The authors present a dynamic approach of retransmissions not fixed to specific links. This approach is similar to the transmission strategy of Sliding Windows presented in this paper. We discuss the differences between the two in Section 3.2.3.

Besides the advances regarding scheduling, Wu et al. [52] present advances in the field of centralized routing in combination with central scheduling. The authors present a conflict aware real-time routing approach, that is aware of scheduling decisions and the possible conflicts of routed paths. Li et al. [51] take a different, asymmetric approach in routing by applying different routing strategies for different communications in one network.

Related to these central scheduling and routing approaches, are systems focusing on network softwarization. plexi [159] is a framework exposing TSCH network resources through a web interface and allowing the rescheduling of communications. Similarly, Baddeley et al. [161] and Galluccio et al. [160] present SDN solutions for Wireless Sensor Networks for network monitoring and reconfiguration. These SDN solutions are conceptually in line with central schedulers calculating schedules externally. Moreover, a combination of our work with SDN solutions is imaginable.

Next to the centralized approaches, a significant focus of recent work is on autonomous scheduling, a concept introduced by Orchestra [8]. Orchestra, as well as Alice [148] and DiGS [259] are autonomous solutions for TSCH, as they do not require neither any central infrastructure nor the exchange of data to build a schedule and achieve high reliabilities of 99.999%. However,

autonomous schedulers are not able to achieve this reliability with latency guarantees necessary for many industrial applications as they have no knowledge on the underlying topology.

6 Conclusion

This paper introduces MASTER, a central scheduling solution for TSCH networks. MASTER introduces a novel Sliding Windows transmission strategy and achieves high reliability independent of knowing the optimal amount of retransmissions per link. Instead, it schedules a number of retransmissions for a flow that can be used at all links of a flow where necessary. The key idea is enabling centralized schedulers to adapt to interference changes without the need for rescheduling while keeping the lowest possible latency. Thus, eliminating a significant overhead of traditional central schedulers.

We implement MASTER in Contiki-NG and evaluate it extensively on a testbed in an environment susceptible to interference. We demonstrate MASTER's practicality and ability to keep stability for over 24 hours and achieve latencies much smaller than Orchestra while achieving similar reliability.

As part of future work, we plan to investigate the challenges of neighbor data collection and schedule distribution to provide a comprehensive central scheduling solution. Moreover, we are planning to evaluate the use of centralized schedulers in harsh wireless environments, such as the ones used in the EWSN dependability competitions [257].

B

Opportunistic Routing and Synchronous Transmissions Meet TSCH

Laura Harms, Olaf Landsiedel

Proceedings of the 46th IEEE Conference on Local Computer Networks (LCN), 2021, pp. 107–114.

Abstract

Low-power wireless networking commonly uses either Time-Slotted Channel Hopping (TSCH), synchronous transmissions, or opportunistic routing. All three of these different, orthogonal approaches strive for efficient and reliable communication but follow different trajectories. With this paper, we combine these concepts into one protocol: Autobahn.

Autobahn merges TSCH scheduling with opportunistically routed, synchronous transmissions. This opens the possibility to create long-term stable schedules overcoming local interference. We prove the stability of schedules over several days in our experimental evaluation. Moreover, Autobahn outperforms the autonomous scheduler Orchestra under interference in terms of reliability by 13.9 percentage points and in terms of latency by a factor of 9 under a minor duty cycle increase of 2.1 percentage points.

1 Introduction

Within the past 20 years, research on low-power wireless networking resulted in a multitude of different protocols. They fall into three prominent fields: Time-Slotted Channel Hopping (TSCH), opportunistic routing, and synchronous transmissions. So far, all three of these fields have little to no overlap, while all strive for a common goal of stable, reliable communication in low-power wireless networks.

In the first field of protocols, the IEEE 802.15.4 Time-Slotted Channel Hopping (TSCH) [6] MAC layer protocol forms the basis for many routed communication protocols. This protocol is standardized and dominates the industry. One category of TSCH protocols uses centralized schedulers, separating the network communication from the routing and scheduling. In recent works [131, 240], centralized schedulers show high reliability and stability. Another category are autonomous schedulers with Orchestra [8] as a prominent example.

TSCH protocols offer stability regarding narrow-band interference. However, long-term stable schedules that are immune to wide-band interference are an open challenge. Wide-band interference likely leads to link failures or even node failures heavily affecting routed communication.

The other two fields can overcome these challenges. Opportunistic routing [171, 174, 175] utilizes anycasts instead of unicasts to add forwarding flexibility by addressing a packet to multiple potential forwarders. It increases the possibility of successful reception in the presence of wireless link dynamics. Protocols building upon synchronous transmissions [9, 59, 65] allow multiple nodes to transmit packets concurrently, commonly by network-wide flooding.

Synchronous transmissions achieve high reliability even in the presence of wide-band interference. However, they have an impact on all nodes in a network. If, for example, in a 1000 node network, two nodes two-hops apart want to communicate, the whole network is involved. In a routed network, only a fraction of these nodes needs to communicate.

In this paper, we ask the following question: Can we combine the benefits of opportunistic routing, synchronous transmissions and centralized TSCH scheduling? For this, we introduce Autobahn: a hybrid routing scheme that combines the best of these worlds: centrally scheduled flows and one-to-one routing of packets as in traditional networking combined with the reliability and robustness of opportunistic routing and synchronous transmissions.

The basic concept of Autobahn is as follows: Its central scheduler schedules a flow along a wider path and allows neighboring nodes to transmit concurrently the same data at the same timeslot and frequency. Thus, a node forwards a packet opportunistically to multiple neighboring nodes, which in turn, in the next slot, concurrently forward opportunistically to their neighbors. In our evaluation, we show that by combining these three approaches, Autobahn efficiently provides reliable, low-latency packet delivery even when links fail, and its schedules are stable for days even in the presence of dynamic interference.

Overall, this paper makes the following contributions:

- We are the first to combine the concepts of opportunistic routing, synchronous transmissions, Time-Slotted Channel Hopping (TSCH) into a

single protocol to achieve long-term stable routed communication.

- We design AUTOBAHN, a robust scheduling and routing policy that withstands link and node failures in the presence of interference.
- We implement AUTOBAHN for Contiki-NG [256] and evaluate it in environments susceptible to interference. We show the long-term stability of schedules using AUTOBAHN over 12 days and under various interference levels for 25 hours. These experiments achieve reliability under interference of 96.8% and latency of 4.2 slots outperforming both the central scheduler MASTER [240] and the autonomous TSCH scheduler Orchestra [8].

The remainder of this paper is organized as follows. Section 2 gives the necessary background information and reviews related work on TSCH as well as the concepts combined in AUTOBAHN. In Section 3, we introduce the design of AUTOBAHN. In Section 4 we evaluate AUTOBAHN's performance experimentally, followed by the conclusion in Section 5.

2 Background & Related Work

In this section, we introduce the necessary background on TSCH, opportunistic routing, and concurrent transmissions and discuss the relevant related work.

2.1 Time-Slotted Channel Hopping (TSCH)

The MAC protocol Time-Slotted Channel Hopping (TSCH) [6] is a combined TDMA and FDMA MAC protocol. It uses 10 ms long time slots with up to 16 frequency channels at each time slot. All active channels follow a pseudo-random hopping sequence that is cycled through, using a different channel at each timeslot to counteract narrow-band interference.

TSCH groups communication slots in continuously repeated slot-frames. All slot-frames together form the TSCH schedule. A TSCH schedule is generated by a centralized, autonomous, or distributed scheduler.

Centralized Scheduling: Central schedulers use global knowledge about the network topology (esp. wireless link quality) to build a schedule and disseminate the schedule into the network. Many early ones, such as TASA [11] and others [10,16], assume interference-free wireless channels without lossy links and, therefore, do not include retransmissions in their schedules. Later work focuses on increasing reliability in the presence of fading channels while ensuring end-to-end latency requirements of each flow. They achieve this by adding retransmissions, i.e., slot-based retransmissions, as used by AMUS [119], to the schedule. As interference can rarely be linked to a specific location beforehand, some recent works by Brummet et al. [131], and MASTER [240] introduce a new approach to retransmissions in TSCH scheduling: they introduce flow-based retransmissions achieving lower latency and a higher degree of adaptability to local interference level.

Autonomous/Distributed Scheduling: Next to these centralized TSCH protocols, a significant amount of work concentrates on autonomous scheduling, a concept introduced by Orchestra [8] and extended by others [148, 259].

Distributed scheduling on the other hand builds on 6TiSCH with its default scheduling function MSF [134], as well as, LLSF [135] and LDSF [136], focusing on improving latency in distributed TSCH.

Multipath TSCH: For multi-path communication in TSCH, several algorithms [166, 168] were studied for distributed and centralized scheduling scenarios. To some extent, these works propose similar ideas as AUTOBAHN, yet they clearly stay within the specifications of TSCH and do not apply opportunistic routing or synchronous transmissions. Moreover, their evaluation results are solely based on simulation.

2.2 Opportunistic Routing

Opportunistic routing is a routing approach to improve network throughput, communication reliability and efficiency in wireless multi-hop mesh networks. Instead of performing unicast communication as established TSCH schedulers do, opportunistic routing builds upon anycasts. By this, opportunistic routing sends each packet to a set of receivers. If any of them receives the packet, the transmission is successful. As multiple receivers might receive the packet, opportunistic routing has to overcome the challenge of selecting a unique forwarder. This forwarder selection has to wait until after the transmission [171–173].

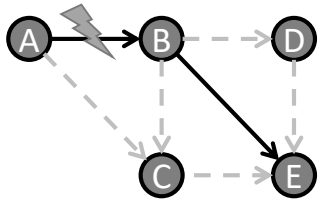
While initial works do not use duty-cycled, low-power wireless networking, later works such as as ORW [174] and ORPL [175] bring opportunistic routing to these. Nonetheless, these protocols are not built for TSCH. Huynh et al. [180], Hermeto et al. [181], and Hosni et al. [182] study the use of opportunistic routing or anycasts in TSCH and propose changes to TSCH to allow non-colliding acknowledgments from multiple receivers. BOOST [183] introduces forwarder selection through sending delays with carrier sense in TSCH. In contrast to these approaches, AUTOBAHN does not use any preferred forwarder selection method. Instead, we overcome this challenge by using synchronous transmissions.

2.3 Synchronous Transmissions

Synchronous transmission protocols allow multiple nodes to transmit packets simultaneously. With precise timing, these packets do not collide destructively, allowing protocols to achieve high communication reliability [9, 65]. As a result, protocols employing synchronous transmissions do not maintain routes by selecting parent nodes, announcing routing metrics, discovering neighbors, and maintaining routing tables as traditional routing protocols.

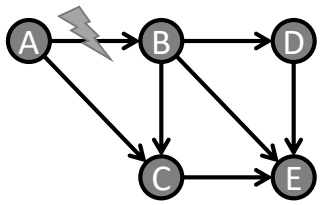
For receiving such a packet, the senders must not significantly differ in timing. One common option of receiving synchronous transmissions is the so-called Capture Effect [57]. According to the capture effect in IEEE 802.15.4, a stronger signal must not arrive later than $160\mu s$ after the first signal [59]. When sending the same data, non-destructive interference is achievable if the time offset between multiple senders is within a bound of $0.5\mu s$ [9].

Synchronous transmissions are well studied. Glossy [9] laid the foundation for synchronous transmissions in wireless sensor networks. Since Glossy's introduction, many protocols including Chaos [59] and LWB [65] followed.



	1	2	3	4
A	TX	TX		
B	RX	TXRX	TX	
C				
D				
E		RX	RX	

(a) Established central scheduling approaches employ a single routing path. Their schedule will fail if one of the links fails, such as the link between nodes A and B in this example.



	1	2	3	4
A	TX	TX		
B	RX	TXRX	TX	
C	RX	TXRX	TX	
D		RX	RXTX	TX
E		RX	RX	RX

(b) Autobahn utilizes multi-path routing and thereby provides redundant options in case routes fail. In this example, packets can travel via node C to destination E.

Figure B.1: Autobahn compared to established centralized TSCH scheduling approaches. In this example, we assume a topology of five nodes, with node A as source and node E as destination. We show both the scheduled paths and the TSCH schedule, using RX, RXTX, and TX slots as typical for flow-based retransmission schemes (with a retransmission window of two). Grayed-out slots present slots where reception and transmission are not possible due to previously failed interfered receptions.

They all are protocols that use network-wide flooding without a concept of routing. Protocols like WSNShape/Sparkle [67], CXFS [68] and LaneFlood [69] divert from network-wide flooding and use flooding with some notion of routing along a path of forwarders.

All of these protocols operate without a routing layer, whereas AUTOBAHN follows the principle of combining synchronous transmissions and TSCH as envisioned by Chang et al. [260]. Gomes et al. [261] study an initial approach of flooding-based routing in TSCH. This approach relies fully on broadcasts (no acknowledgements) and uses shorter TSCH slots. Baddeley et al. [193] present a hybrid between TSCH and synchronous transmissions by replacing some TSCH slots with synchronously transmitted BLE packets for exchanging control information.

While some protocols explore the field of combining TSCH and synchronous transmissions, AUTOBAHN explores it further by combining synchronous transmissions with TSCH, including both synchronous transmissions as well as synchronous acknowledgments in combination with opportunistic routing.

3 Design

We continue with the design of AUTOBAHN. We begin with a simple example to present the basic idea of AUTOBAHN. Then, we introduce (1) general node selection requirements, (2) the forwarder selection through node ranks, and (3) the active nodes in each slot. After discussing these main points of the design, we present the system design, including the Contiki/TSCH extensions to allow anycast communication, and AUTOBAHN's routing layer adaptations.

3.1 Autobahn: General Idea

As an example, we assume the network of five nodes in Figure B.1 where Node A communicates with node E. Further, let us assume that the link between nodes A and B fails due to interference.

To illustrate the benefits of AUTOBAHN, we first discuss how established centralized scheduling approaches suffer from link failures. Established approaches commonly employ a single routing path. Their schedule will fail if one of the links fails, such as the link between nodes A and B in this example, see Figure B.1a. Retransmissions, as scheduled in the example, usually happen on a different channel and thereby protect the protocol against narrow-band interference. Wide-band interference, however, can break links and result in packet loss. Eventually, the scheduler has to deploy an updated schedule. If this is done frequently, this adds significant overhead to the communication scheme.

The general idea behind AUTOBAHN is to add redundancy to the routing path, see Figure B.1b. In the example of AUTOBAHN, node A sends a packet that will be received by nodes B and C. These two forward the packet synchronously to nodes D and E, which receive one of the two transmissions due to the capture effect. Lastly, node D sends the packet to node E as well. In case of interference, node B is not reachable. That means that only node C receives the packet from node A. Node C then forwards the message to node E. Redundant routing paths in AUTOBAHN add non-neglectable overhead to the duty cycle of the network. In our evaluation, we, however, show that this overhead is justifiable in the interference-free case, and in the case of interference, it is essential for reliable communication.

3.2 Routing Set

Centralized schedulers have global knowledge over the network topology through long-term link quality metrics. They commonly route traffic along a single path, using single forwarders. In contrast to that, AUTOBAHN addresses packets to multiple forwarders (anycast). To achieve this, we employ a routing set with redundancy instead of a single path.

We define a routing set to consist of all nodes we use for end-to-end communication. A routing set $\{rs\}$ is a set of nodes $n_1, ..., n_k \in \{rs\}$ responsible for routing a data packet from the source node n_1 to the sink node n_k. This routing set contains the nodes forming the shortest path from source to sink as well as additional nodes used for opportunistic, anycast routing in AUTOBAHN, which adds path redundancy.

To build a routing set, we start with the shortest path from source to destination employing the ETX-metric [48] and Dijkstra's shortest-path algorithm [49]. Next, we add routing redundancies by including neighboring nodes along the path. For this, we introduce three schemes: (i) neighbor-based, (ii) hop-based, and (iii) cost-based selection of routing sets. Especially in dense networks, the number of these neighboring nodes for each of the three schemes is likely to be high and leads to the inclusion of massive parts of the network. Therefore, we specify a node overhead factor (scaling the number of additional nodes) and a cost overhead factor (scaling the max. allowed ETX cost of nodes of an end-to-end path).

Neighbor-based participant selection starts with determining all nodes neighboring at least one node of the shortest path. From these nodes, we continue with three different subsets: (a) all selected nodes that do not exceed the cost overhead, (b) a subset of (a) forming a second shortest path, and (c) a subset of (a) forming a shortest path from each node of the original path to the destination node.

After selecting the respective nodes, we check whether the node overhead is too large. If so, we refine our selection only to include the allowed number of nodes with the lowest cost.

The hop-based selection possibility includes additional nodes with a similar combined distance to the shortest path's source and destination while not exceeding the cost overhead. Equation B.1 shows the general idea of this selection strategy, that the combined hop distances from source to forwarder (d_{sf}) and forwarder to destination (d_{df}) must not exceed the direct distance d_{sd} plus a slack value s. The slack value has to be a natural number. If the node overhead is too large, nodes with the lowest hop count are preferred.

$$d_{sf} + d_{df} \leq d_{sd} + s \tag{B.1}$$

The cost-based selection possibility follows the same equation. However, instead of hop-based distances, we use ETX-based distances and an ETX-based slack (s), which can be a positive real number. However, the maximum slack value for this strategy equals the maximum cost overhead allowed according to the cost overhead factor. If, after node selection, the node overhead is too large, we rank the nodes regarding their cost and take those with the lowest cost. In addition, we exclude all other nodes that have the same cost as one of the already excluded nodes.

3.3 Anycast forwarding in Autobahn

In anycast routing, we address a packet to a set of neighboring nodes, i.e., the ones making sufficient progress towards the destination. Thus, for each transmission in Autobahn, this set of possible recipients listens for the packet. Practically, we introduce node ranks: Each node has a rank according to its distance to the destination of a flow from source to destination. The sender of a packet has rank 0, and the rank increases towards the receiver, with the receiver having the highest rank.

If a node receives a packet, it compares its rank to the sender's rank, which we include in the packet header. If the own rank is higher, it acknowledges the packet and forwards it. Otherwise, it stays silent and acknowledges for

itself that the packet has passed. The sender of a packet performs a similar action. If it receives an acknowledgment from a node with a higher rank, it concludes that the opportunistic anycast succeeded and stops forwarding this packet. This way, we ensure that only nodes closer to the packet's destination acknowledge the reception of the packet and forward the packet; thus, we avoid loops and packets stuck mid-flow.

In traditional opportunistic routing, packet duplicates are often a challenge [171, 174, 175]: There is always a risk that multiple forwarders receive a packet, and each individually forwards the packet, adding additional load on the network. In AUTOBAHN, all packets – including duplicates – are forwarded synchronously, and their spatial diversity is the basis for the reliability of our design in the presence of interference. Thus, duplicates are (i) inherently part of the design and (ii) do not add the overhead as in traditional designs.

3.4 Active slots in Autobahn

AUTOBAHN uses flow-based retransmission schemes such as Sliding Windows introduced by MASTER [240], with multiple nodes possibly active in one slot. AUTOBAHN extends this by activating all nodes in a slot that are reachable by any previously active node.

In the first slot of a flow, the sender of a packet and all receivers in range are active. For each of the following slots, we include all additional nodes reachable by any previously active node. From this, we derive the first active slot of a node, i.e., the first point in time a packet in a flow can reach a node along one of the different paths employed by AUTOBAHN. We determine a node's last active slot based on the node's hop distance to the flow's receiver and the flow's number of total transmissions.

Due to the opportunistic nature of AUTOBAHN, a high network duty-cycle is expectable. Nevertheless, to still keep the energy consumption as low as possible, each node stays only active until we no longer need it for forwarding the packet. As we explain above, a node determines whether it is still needed through the received rank of other participants.

The schedules in Figure B.1 illustrate the difference between the active slots of a flow-based central scheduler (Figure B.1a) without opportunistic routing and AUTOBAHN (Figure B.1b).

3.5 System Integration

For the design of AUTOBAHN, we devise a TSCH implementation with support for opportunistic anycasts and a good enough time synchronization for synchronous transmissions. In our evaluation, we show that the TSCH implementation of Contiki-NG is sufficient for synchronous transmissions. However, as it does not support anycasts, we have to realize these ourselves. AUTOBAHN itself can be implemented on top of any centralized scheduler. We choose MASTER, a centralized scheduler implemented for Contiki-NG as our basis. We implement AUTOBAHN to replace MASTER's central routing and retransmission logic while keeping its scheduling module. Below, we discuss the integration into MASTER's Contiki routing layer and the extension of Contiki-NG/TSCH to allow opportunistic anycasts.

3.6 Integration in Master's routing layer

We extend MASTER's routing layer to have access to a node's rank and relay a packet back to the correct flow address instead of a neighbor address.

The routing layer is also responsible for the routing-specific header. In addition to the existing 7-byte routing header, AUTOBAHN requires one additional byte. The existing 7 bytes are a flow identifier (1 byte), a sequence number (2 bytes), the time-to-live (TTL) (2 bytes), and the earliest transmission slot (2 bytes) of a packet. AUTOBAHN adds the node's rank to the packet to allow the receiver to make its forwarding decision according to our description above.

3.6.1 Contiki-NG/TSCH extensions

The TSCH implementation of Contiki-NG does not support anycast communication. To add support for anycasts, we extend it with (1) the capability of accepting packets from any neighbor of a flow, as well as (2) using this flow as a sender and receiver simultaneously. Moreover, we (3) define a flow-specific sequence number to accept acknowledgments successfully.

With our modification, TSCH accepts packets from a flow address if the receiving node is a member of the respective flow. As we no longer need the sender's and receiver's addresses, we replace them with the flow address.

Besides accepting packets from any flow participant, TSCH needs the capability to accept acknowledgments from any possible forwarder of the flow. Therefore, we include the receiver's rank in the acknowledgment. If a node receives an acknowledgment acknowledging a different synchronous sender, it still needs to be accepted. Therefore, our routing layer replaces the TSCH sequence number with a flow and packet-specific end-to-end sequence number.

4 Evaluation

In this section, we evaluate AUTOBAHN's performance and compare it to the state-of-the-art. We start by showing the feasibility of synchronous transmissions in the context of TSCH. After that, we evaluate AUTOBAHN's different routing set selection choices and compare those to MASTER in scenarios with and without interference. Afterward, we compare AUTOBAHN's best routing selection algorithm against Orchestra, the default autonomous scheduler in Contiki-NG. We conclude our evaluation with long-term stability analysis of schedules in AUTOBAHN.

4.1 Evaluation Setup

4.1.1 Testbed and Platform

We run our experiments on a 20-node testbed at our local university. This testbed (Figure B.2) covers the top floor of a university building with offices and student lab rooms and thus shares the wireless spectrum with WiFi and Bluetooth communication outside of our control.

4.1.2 Metrics, Comparison, and Duration

We evaluate AUTOBAHN in terms of end-to-end reliability, end-to-end latency, and network energy consumption (network duty cycle). We measure these metrics for different routing and retransmission approaches for MASTER and AUTOBAHN under different interference levels. Moreover, we compare AUTOBAHN with Orchestra according to these metrics. We include six flows we give in Figure B.2. The duration of each experiment in sections 4.5, and 4.4 is 75 minutes, with each flow sending 100 packets per minute. In section 4.6 we run 75-minute experiments with 60 packets per minute and flow. For the long-term evaluations from Section 4.8 onward, we specify the duration as part of the specific experiment.

4.1.3 Implementation

We implement AUTOBAHN for Contiki-NG [256] and target the Zolertia Firefly Platform. This platform features a CC2538 Cortex-M3 CPU (32-bit, 32 MHz) with 32 KB of RAM, 512 KB flash storage, and an IEEE 802.15.4 compatible radio.

4.1.4 Channels and Interference

We perform most of our experiments under interference. To ensure comparable levels of interference for all tested protocols, we generate these ourselves in a repeatable manner using JamLab [248]. If not stated otherwise, we use an interference level of 10% channel occupancy. We use five interference sources depicted in Figure B.2. Two of the interference sources are in a central position surrounded by several nodes, while the other three are each in close vicinity to a forwarding node in the network. As our testbed only provides the capability of generating interference on one channel at a time, we use only a single channel (channel 26) for all experiments. As we target networks susceptible to wide-band interference, evaluating on only one channel is not a problem. Wide-band interference, such as WiFi, would cover multiple IEEE 802.15.4 channels, eliminating channel hopping advantages. Therefore, it is more realistic to use one channel with interference than multiple channels with interference on only one of them. Moreover, using only one channel lets us compare the worst-case performance of the discussed protocols.

4.1.5 Application Payload and Overhead

We send packets with a 64-byte randomly generated payload for all experiments, a medium packet size for TSCH. Additionally to this data payload, we include 7-byte and 8-byte routing headers for MASTER and AUTOBAHN, respectively. Orchestra uses IPv6 headers instead and requires additional network layer control traffic.

4.1.6 Routing Sets

We include three AUTOBAHN routing sets marked as neighbor-based, hop-based, and cost-based. The neighbor-based one is option (c) of the neighbor-based routing sets in Section 3.2, the one with an alternative path from each node

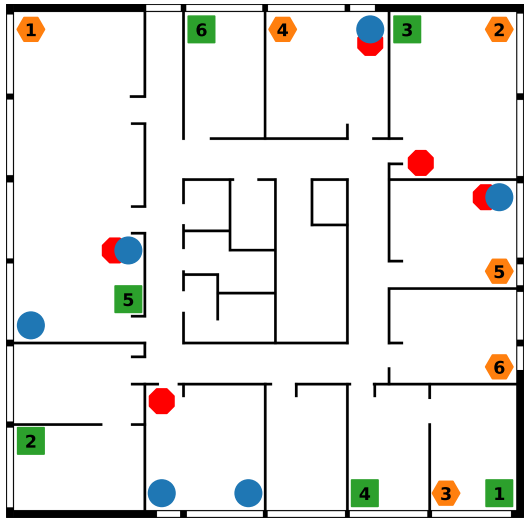


Figure B.2: Local testbed of 500m^2. Source nodes: orange hexagons; Sink nodes: green squares; Relay-only nodes: blue circles; Red octagons: interferer; Numbers: corresponding flow

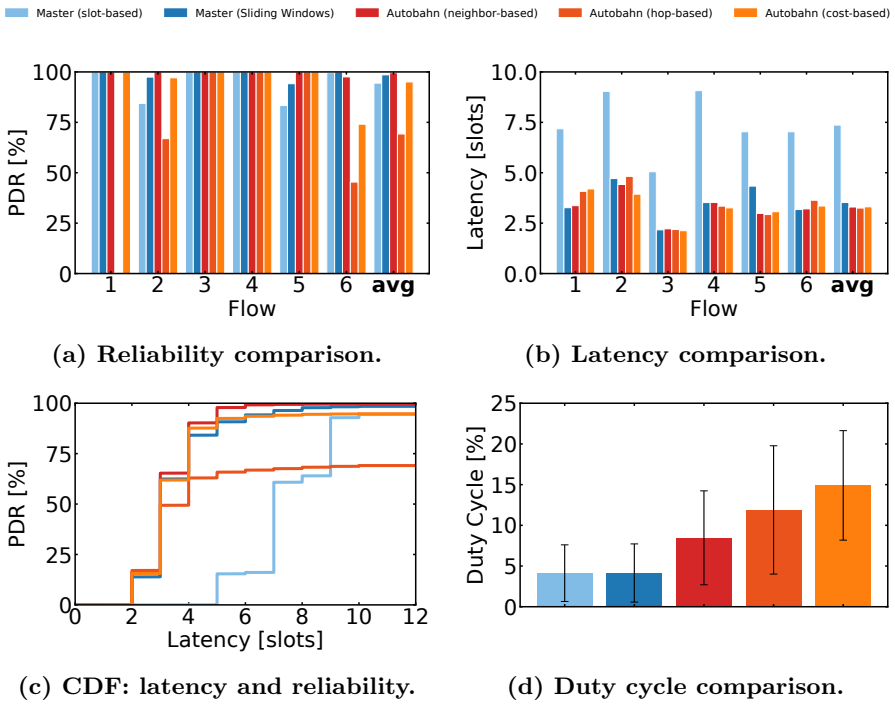
through all neighbors. For the hop-based routing set, we use a slack value of 2. For the cost-based routing set, we use the maximum possible slack value, equaling the maximum cost overhead. This slack value is potentially different for each flow. This value ensures that we include all nodes, with an end-to-end ETX value not exceeding the cost overhead factor. We use overhead factors of 2 and 2.5 as node-overhead factor and cost overhead factor, respectively.

4.2 Baselines

We compare AUTOBAHN's routing-set algorithms to three other TSCH scheduling policies. Two of these are MASTER's slot-based retransmission strategy and MASTER's flow-based transmission strategy called Sliding Windows. MASTER's slot-based retransmission strategy follows the traditional concept of replicating slots of single hops, done in several recent publications, including AMUS [119]. We use MASTER, as it provides us an implementation for Contiki-NG. As the last baseline we use Orchestra, to set AUTOBAHN into relation to a well-known protocol. Orchestra [8] is an autonomous scheduler for TSCH included in Contiki-NG [256]. It autonomously maps links to resources, e.g., determines a node's send or receive slot based on a hash function.

4.3 Possibility of Synchronous Transmissions in TSCH

Before starting our main evaluation, we investigate the quality of synchronization in TSCH for synchronous transmissions. With a desk setup of 4 nodes, we can identify the feasibility of synchronous transmissions. Our data shows an average offset between two synchronously transmitting nodes of 16.4μs with a standard deviation of 16.8μs and a maximum offset of 65.7μs. This



(a) Reliability comparison.

(b) Latency comparison.

(c) CDF: latency and reliability.

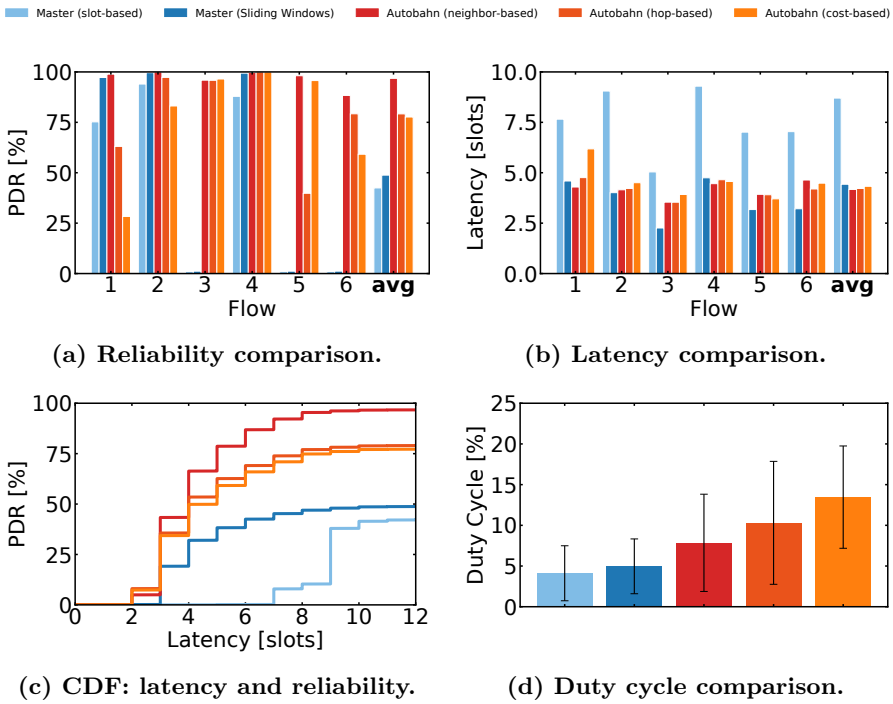
(d) Duty cycle comparison.

Figure B.3: Autobahn and Master without interference. Autobahn's neighbor-based strategy outperforms Master while increasing the duty cycle by 4.3 percentage points.

offset clearly shows that the degree of synchronization in Contiki's TSCH implementation is by far not good enough for constructive interference (offset bound of $0.5\mu s$ [9]). However, the offset stays below the maximum offset for capture effect of $160\mu s$ [59]. TSCH generally does not require synchronization as strict as Glossy and therefore does not include additional physical layer time synchronization measures. Nonetheless, our results show that synchronous transmissions are possible due to the capture effect.

4.4 Performance without Interference

We begin our evaluation by comparing the performance of Autobahn's different routing sets with the performance of Master's retransmission strategies. For this evaluation, we do not generate any interference. When comparing the reliability of the different routing sets (see Figure B.3a), we see a generally better performance of the neighbor-based routing set in comparison with the hop-based or cost-based ones. Especially the difference between the neighbor-based and hop-based routing sets is visible for flows 1 and 6. The hop-based routing set has too many simultaneously active nodes at similar distances to a forwarder or receiver. With this number of active nodes, no signal is strong enough for reception through the capture effect. For flow 1, we even see a destruction of the signal. The cost-based strategy has better reliability but



(a) Reliability comparison.

(b) Latency comparison.

(c) CDF: latency and reliability.

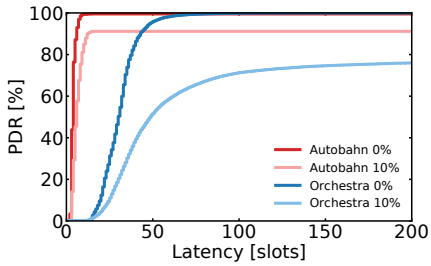
(d) Duty cycle comparison.

Figure B.4: Autobahn and Master under interference. Autobahn has a much better performance than Master with the best performance using the neighbor-based routing set.

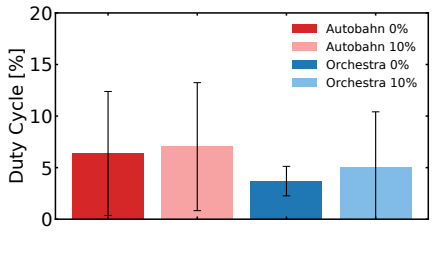
generally does not achieve the high reliability of the neighbor-based routing set. The baseline strategies are not exposed to in-flow interference and achieve almost the reliability of neighbor-based AUTOBAHN, with a slight advantage for Sliding Windows over the slot-based retransmission strategy. However, if the network's link qualities are not perfect for a flow (flows 2 and 5), MASTER is more strongly affected. Latency-wise (see Figure B.3b, B.3c), AUTOBAHN has a small advantage over MASTER, while both AUTOBAHN and Sliding Windows are better than the slot-based retransmission strategy. The reliability improvement comes at a cost of a higher network duty cycle (see Figure B.3d). With more active nodes, the duty cycle increases significantly. Nevertheless, the best performing routing set of AUTOBAHN leads to the least increase in duty cycle by, on average 4.3 percentage points.

4.5 Performance under Interference

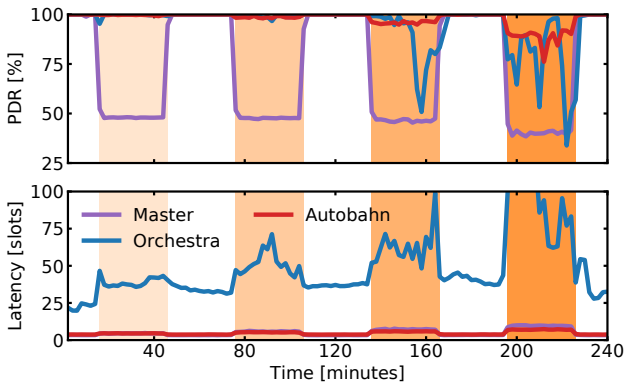
Next, we compare the same strategies as before under induced interference. From Figure B.4a it is visible that the neighbor-based routing set once again performs best. The other routing sets still offer average reliability of around 80%. However, for some flows, these routing sets achieve very low reliability (e.g., flow 1). This low reliability indicates that too many nodes are active simultaneously plus the additional interference heavily impacting a successful



(a) Latency and reliability comparison between Orchestra and Autobahn with and without interference.



(b) Network duty cycle of Orchestra and Autobahn with and without interference.



(c) Performance under and recovery from interference over time. (Interference levels from left to right: 5%, 10%, 15%, and 25%)

Figure B.5: Comparison of Autobahn and Orchestra with and without interference. Figure B.5c includes the recovery performance of Master's Sliding Windows strategy as an additional baseline.

capture effect. Comparing AUTOBAHN to MASTER shows that all routing sets of AUTOBAHN have higher average reliability than MASTER's strategies. This is due to MASTER being heavily impacted by interference in certain flows (flows 3, 5, and 6). In these flows, the shortest path passes closely to an interference source. Thus, we see that additional nodes offered by AUTOBAHN are necessary to route traffic around interference sources opportunistically. The latency differences (see Figure B.4b) follow the same trend as without interference, just slightly higher. As the latency comparison only includes received packets, we also show the combination of latency and reliability in Figure B.4c. AUTOBAHN requires a latency of 4 slots to reach a 50% reliability, while MASTER cannot reach this network-wide reliability at all. The higher overall reliability comes at the cost of a higher network duty cycle (see Figure B.4d). The cost is similarly high as for the interference-free case. However, especially in the presence of interference, an increase of network duty cycle of 2.89 percentage points for the best routing set should be acceptable if reliability has high priority.

4.6 Autobahn vs. Orchestra

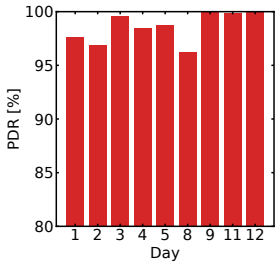
Next, we compare AUTOBAHN's best-performing routing set (neighbor-based) with another baseline, the autonomous scheduler Orchestra. As Orchestra is a best-effort protocol and is therefore not limited to a deadline, i.e., a certain number of slots to successfully transmit a packet, we, therefore, relax these limitations for AUTOBAHN as well. However, as we send a new packet every second, AUTOBAHN's schedule should not exceed this value. We achieve this by using a higher scaling factor of three instead of two to determine the number of transmission slots. That means that we use for each flow 50% more transmissions than in the previous experiments. The results of this comparison (see Figure B.5a-B.5b) show that Orchestra achieves slightly higher reliability than AUTOBAHN without the presence of interference (99.96% vs. 99.51%). However, in the presence of interference, AUTOBAHN clearly outperforms Orchestra with a 13.89 percentage points higher packet delivery rate (PDR). Latency-wise, AUTOBAHN clearly outperforms Orchestra. In case of interference, by a factor 9. However, we need to attribute some of that to the fact that all flows communicate actively simultaneously in Orchestra, while in AUTOBAHN's schedule, the flows communicate one after another. Energy-wise, we can see a similar trend as with the comparison of AUTOBAHN and MASTER (Figure B.3d and Figure B.4d). AUTOBAHN uses more active nodes and therefore has a higher duty cycle. In contrast both MASTER and Orchestra follow a single path, with MASTER's duty cycle being the lowest, while Orchestra occupies the middle ground.

4.7 Recovery from interference

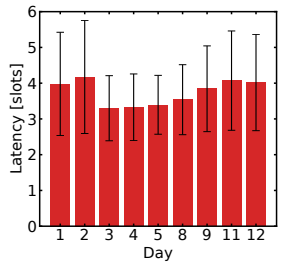
Next, we compare how well MASTER, AUTOBAHN, and Orchestra perform under interference over time and how good they are at recovering from interference. Figure B.5c shows that all three algorithms are influenced by interference but are successful at recovering independently of the interference level. MASTER and AUTOBAHN have similar latency responses, while Orchestra has a less uniform curve as the rerouting in case of interference takes some time. Reliability-wise, Orchestra keeps high reliability for quite a long time but has to drop packets towards the end of an interference block. MASTER is generally hit the hardest by interference, while AUTOBAHN clearly performs best under interference.

4.8 Long-term stability of Autobahn

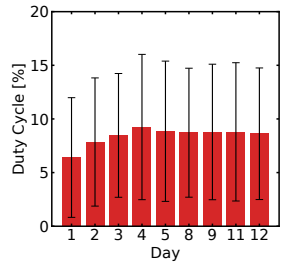
After comparing AUTOBAHN to various baselines under different interference levels, we evaluate AUTOBAHN's long-term stability. As before, we use AUTOBAHN's neighbor-based routing set. For this section, we use a schedule generated with a neighbor discovery on day one and run this schedule almost daily over 12 days. Moreover, after 12 days, we run a final 25 hour experiment with three-hour blocks of 5%, 10%, 15%, and 25% interference, respectively. While the performance varies within the days, reliability always stays above 96% (see Figure B.6a). Latency fluctuates slightly, yet it remains around the level of the first days, whereas the duty cycle shows an upwards trend in the beginning (see Figure B.6b-B.6c). The performance analysis over 25 hours (see



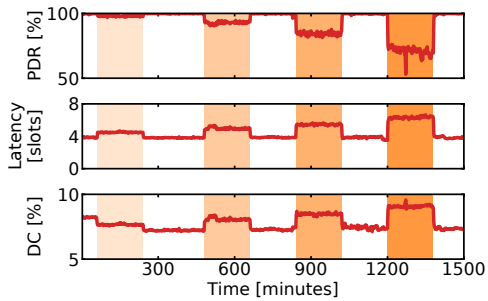
(a) Reliability over the course of 12 days.



(b) Latency over the course of 12 days.



(c) Energy consumption (duty cycle) over the course of 12 days.



(d) Performance over 25 hours with 3 hour long interference blocks (interference levels from left to right: 5%, 10%, 15%, 25%).

Figure B.6: Long-term stability evaluation of Autobahn. Figures B.6a - B.6c present (almost) daily runs of Autobahn with the same schedule over the course of 12 days. Figure B.6d shows the performance of the same 12-day old schedule under different interference levels and its recovery from interference.

Figure B.6d) shows that even an almost two-week-old AUTOBAHN schedule is still able to perform under interference and quickly recover from it.

5 Conclusion

Centrally scheduled networks are sensitive to wireless link dynamics, esp. wideband interference. AUTOBAHN addresses this by adding spatial redundancies via combining TSCH with synchronous transmissions and opportunistic routing.

We show that AUTOBAHN offers reliability of 95% and more under interference while mildly increasing the duty cycle by 4.3 percentage points. Moreover, experiments over 12 days show the long-term stability of AUTOBAHN's schedules with 98.6% reliability.

C

BlueSeer: AI-Driven Environment Detection via BLE Scans

Valentin Poirot, **Laura Harms**, Hendric Mertens, Olaf Landsiedel

Proceedings of the ACM/IEEE Design Automation Conference (DAC), 2022, pp. 871–876.

Abstract

IoT devices rely on environment detection to trigger specific actions, e.g., for headphones to adapt noise cancellation to the surroundings. While phones feature many sensors, from GNSS to cameras, small wearables must rely on the few energy-efficient components they already incorporate. In this paper, we demonstrate that a Bluetooth radio is the only component required to accurately classify environments and present BlueSeer, an environment-detection system that solely relies on received BLE packets and an embedded neural network. BlueSeer achieves an accuracy of up to 84% differentiating between 7 environments on resource-constrained devices, and requires only ~12 ms for inference on a 64 MHz microcontroller-unit.

1 Introduction

Smartphones, wireless peripherals, and small wearables constantly accompany us in our daily life: at home, in transit, while shopping, or even at work. The ability to detect our surrounding environments, known as Environment Detection or Classification [235], is a desirable trait in mobile systems: a smartphone can automatically enter into silent mode when it detects that we enter a theater or airplane mode once we board a plane; wireless headphones can adapt their degree of noise cancellation to match the current environment: total cancellation in offices, but limited cancellation in streets to ensure that the user can still hear oncoming traffic and emergency vehicles; fitness trackers can distinguish between indoor and outdoor activities by checking the user's surroundings; and smart speakers can control their initial volume if they detect they are in public spaces. Environment detection also enables informed decisions for co-located systems: the performance of distance estimation and localization algorithms relying on radio signal propagation can suffer from multi-path fading indoors [262, 263], environment detection allows such algorithms to select the best propagation model based on the current environment. While environment detection can aid localization systems [264], it does not aim at pinpointing a device position within a given area; instead, it classifies the surrounding environment into general categories such as shopping center, office, home, or street.

Modern smartphone systems feature many sensors that we can use to infer surroundings, such as Global Navigation Satellite Systems (GNSS, e.g., GPS), cameras, microphones, or ultra-wideband radios. In contrast, small wearables such as fitness trackers or in-ear earphones often lack this luxury. While GNSS chips offer precise localization and, with access to maps, provide environment detection with high accuracy, they remain the most energy-hungry on-device sensors [265], and would drain an unacceptable share of the energy budget of a small IoT platform. Microphones are a more energy-friendly alternative to perform environment detection [235], but the presence of microphones on fitness trackers for the sake of environment detection raises concerns on privacy.

Yet, most embedded wearables incorporate one common component: a Bluetooth radio. With 4 billion new devices shipped in 2020 alone [266], Bluetooth and Bluetooth Low Energy (BLE) are the go-to solutions for wireless communication on the most modest IoT wearables. Although it seems incongruous at first that we can use a BLE radio to infer its surroundings, this paper demonstrates that we can classify environments with high accuracy solely from received BLE packets. Specifically, we show that the density, diversity, and dynamics of mobile devices, which we can infer from listening to their periodic packet broadcasts, form a wireless fingerprint that can be relied on to categorize surroundings. With this insight, we are able to provide a new approach to accurate and energy-efficient environment detection; any device, from the smallest BLE-enabled sensors up to smartphones, can accurately infers the environment by simply turning on its BLE radio periodically. If a device already scans for BLE transmissions, no additional radio-on time is required; the received packets are used as the basis for the classification.

Challenges. By default, many BLE devices announce their services and thereby their presence so that other devices can connect to them via packets

called advertisements. BLE Advertisements contain the different services offered by each equipment, for example volume control for smart speakers or temperature for thermometers. The presence of specific services influences the classification: for example, more keyboards are found in offices than restaurants, more smart assistants are located in homes than in transport. The number of devices in proximity also discriminates between environments: we receive many more BLE signals in a busy street than deep in a forest. However, this wireless fingerprint highly evolves with the time of day: crowded transport at rush-hour shares more similarities with a concert venue than riding an empty bus. As the number of advertisements and the number of services within packets highly vary with time and place, feature engineering is required. We need to extract meaningful features over all received advertisements to be able to pass it as input to, e.g., a neural network classifier.

Approach. We present *BlueSeer*, an environment-detection system specifically tailored for resource-constrained IoT platforms. To work, BlueSeer only requires a BLE radio on the device: periodically, BlueSeer scans for BLE advertisements from nearby devices, performs feature extraction from the raw data, and employs an embedded neural network to predict the current surrounding environment. We show that the knowledge extracted from BLE packets is sufficient to infer the environment accurately.

Contributions. This paper makes the following contributions:

(1) We show that it is possible to categorize environments exclusively from received BLE advertisements;

(2) We present BlueSeer, an Environment Detection system able to classify environments solely using received BLE packets. BlueSeer distinguishes between 7 categories: home, office, shopping, transport, nature, street, and restaurant;

(3) We carry out extensive feature engineering and identify 23 features from BLE advertisements, ranging from number of devices in proximity and RSS measurements, to the diversity of offered services;

(4) We devise a neural network and show that its quantized, embedded version classifies its environment with up to 84% accuracy on a low-power platform with a 64 MHz MCU, and uses 65 KB of memory. We make BlueSeer's implementation and its dataset open-source[1].

Outline. The paper is structured as follows: §2 provides background on Bluetooth Low Energy advertisements, §3 dives into the design of BlueSeer, §4 provides an in-depth evaluation of BlueSeer and its embedded neural network, §5 summarizes the related literature and §6 concludes this paper.

2 Background: Bluetooth LE

BLE. Introduced as part of Bluetooth 4.0 in 2010 [7], Bluetooth Low Energy (BLE) is a short-range wireless technology in the 2.4 GHz ISM band. BLE targets direct, one-hop communication and provides datarates of up to 2 Mbit/s

[1]Available at: github.com/ds-kiel/blueseer

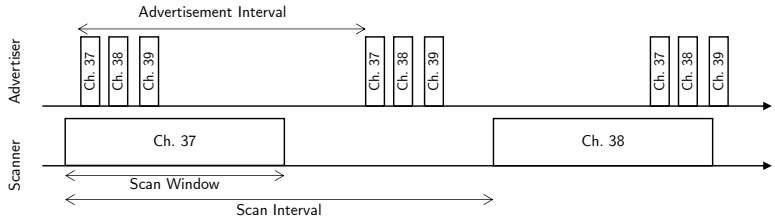


Figure C.1: BLE Advertisements. The advertiser pseudo-periodically sends advertisement packets on all three advertising channels. The scanner periodically listens for advertisements.

within 10-50 meters, typically. It uses 40 2-MHz wide frequency channels, 37 reserved for connection exchanges, and 3 for advertisements and broadcasts.

Advertising. BLE has two operation modes: connected and non-connected mode. The non-connected mode is used to disclose the presence of connectable devices (for example after turning on headphones) or broadcast data to nearby devices (such as COVID contact tracing keys). The advertiser, such as a small wearable, operates by broadcasting an advertisement packet on all advertising channels (cf. Fig. C.1) and repeats the packet pseudo-periodically at a fixed interval, plus a random delay to avoid collisions. On the receiving end, a scanner, such as a smartphone, scans the medium for advertisements by listening to a specific channel during a scan window. The scanner iterates over all advertising channels by switching channels after a period called scan interval. A device initiates a connection by responding to an advertisement: data exchange then proceed on the remaining 37 channels.

Advertising data. The payload of BLE advertisements consists of a list of optional fields called Advertising Data (AD). Advertisers include AD to define, e.g., their device name, address, the list of offered services (such as sound or heart rate sensor), or data related to its specific manufacturer. Each AD is identified by its universally unique identifier (UUID), defined in the standard [7]. For example, the COVID Exposure Notification AD has the UUID 0xFD6F [38]. The presence of specific AD fields (e.g., announcing keyboard capabilities), the variety of advertised services, as well as the variety of advertisements received within one scan form a wireless fingerprint that we use to infer the current environment.

3 Design: BlueSeer

With BlueSeer, we demonstrate that a BLE radio is the only component required for a device to detect its environment. All BLE advertisements received by a device compose a wireless fingerprint of the surroundings. This fingerprint builds the basis for inferring the environment's category. We present the system architecture of BlueSeer in §3.1, dive deep into the data collection and feature extraction processes in §3.2, describe the embedded neural network used by BlueSeer in §3.3, and precise implementation-specific details in §3.4.

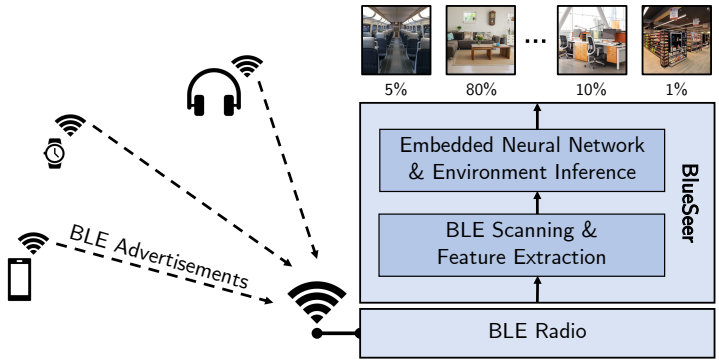


Figure C.2: BlueSeer: System architecture. BlueSeer scans the wireless medium for BLE packets and extract features from them. An embedded neural network classifies the environment between 7 categories.

3.1 Overview

Wireless Fingerprints. Different environments often exhibit different characteristics: multiple wireless keyboards are often found nearby in offices, smart home assistants at home; buses driving around town pass by many people and devices for a short duration; and the occasional wireless speaker can be found in city parks when the weather allows. We show in this paper that the devices found within an environment, their density, variety, the motion of those devices as well as the mobility of the receiver, all form a unique composition specific to each environment. Further, since many of these devices voluntarily disclose their proximity and services via BLE advertisements, each environment has its own wireless fingerprint that we exploit for classifying the environment. Thus, the sole presence of a BLE radio is the only requirement for IoT platforms to classify their surroundings.

BlueSeer. BlueSeer consists of two main components: (1) the scanning and feature extraction block, which produces data for a classifier, and (2) the classification process that relies on an embedded neural network, as we depict in Fig. C.2. Periodically, BlueSeer scans the wireless medium for BLE advertisements. From the raw packets' data, BlueSeer extracts features related to the device density, variety, as well as the dynamics of the environment (see §3.2). We then feed these features into an embedded neural network, whose weights are quantized to fit in the memory of small, constrained IoT platforms (see §3.3). Since many wearables do not feature constant internet connectivity, we cannot rely on a complex classifier on the cloud or nearby edge-device to infer the category and must rely on a memory-efficient model. With its neural network, BlueSeer discriminates between 7 different environments:

- Home (house, apartment)
- Office
- Shopping (such as supermarkets, malls)
- Transport (e.g., car, bus, train)

- Nature (city parks, forests, countryside)
- Street (both walking and standing)
- Restaurant

3.2 Feature Extraction

Raw data. Rather than pass the raw data to a classifier, we perform feature extraction on the received packets to produce 23 informative features that we pass onto the embedded neural network. As BLE scans return anything from a few results up to a hundred packets in crowded spaces such as restaurants, and the packet length evolves with the number of services advertised, raw packets are bad candidates as input features. In the following, we present a selection of the most notable features produced by BlueSeer.

Dynamic environments. Since the device's density and dynamics are markers of an environment, we extract the number of unique devices we encounter within a scan as our first feature. The number of devices that left our vicinity since the previous scan, which we call lost devices, and the number of new devices that we hear for the first time serve as our second and third features. These metrics represent both the density and dynamics of the environment.

During a single scan, a receiver might receive more than one packet from the same source since every advertiser chooses its own advertisement interval (see §2). This fact gives two kinds of information about our surroundings: (1) the type of devices in proximity, as low-power platforms tend to select long intervals to save energy; and (2) how long devices stay nearby: we should receive many packets from devices with short intervals unless they quickly leave our vicinity. Therefore, we measure the average interval between two transmissions from the same source, and how long devices stay in our vicinity on average, and use both as additional features.

Signal strength. Whenever the BLE radio receives a new advertisement, it records the Received Signal Strength (RSS). We use the RSS as a rough approximation of the distance between a sender and receiver [267]. Successive measures also produce a simple estimation of a device's motion. When combined, the RSS datapoints represent the estimated device density of the current environment. We collect the following statistics from raw RSS values and use them as features: the highest and lowest RSS measures received as well as the average RSS over all devices.

Device diversity. Along with the device density and environment dynamics, the device variety is an important factor in distinguishing between environments. While BLE lacks the device class categories used by Bluetooth Classic, we deduce approximate categories from the list of services advertised. For example, the TX Power AD exposes the transmit power used by the sender; larger devices such as laptops tend to transmit at high power, while battery-powered sensors rely on low levels to operate longer. We collect the following metrics as features: the lowest, highest, average, and count of TX Powers advertised by nearby devices to infer information on their types.

We can further take advantage of the diversity of AD options advertised by BLE packets. By counting how many unique services are offered, we can infer the diversity of devices. By counting the total number of services, we can

infer the homogeneity. The presence of manufacturer data is also a marker of specific environments; devices from different manufacturers send specific AD, and different devices by the same manufacturer also send unique data. We extract the average manufacturer data length to use it as an additional marker for environment detection.

3.3 Embedded Neural Network

The feature extraction process produces a total of 23 unique features that encompass the device density, variety, the radio conditions, as well as the dynamics of the environment. As each scan generates the same number of features, we directly feed them as input to an embedded neural network.

Sliding-window. Yet, using the features from a single scan may lead to inaccuracies, especially in highly dynamic environments. For example, a car driving along a shopping street sees many advertisements, but receives almost no packets a few seconds later as it enters a residential district. We keep track of the last scans within a sliding-window and concatenate the result of several BLE scans together. We feed the resulting vector as input to BlueSeer's embedded neural network. Because we use several consecutive scans, the neural network's decisions are less subject to fluctuations in highly dynamic environments. Importantly, BlueSeer performs inference after each individual scan: BlueSeer does not require to refill the window with fresh scans between two inferences entirely; instead, earlier scans remain part of the sliding window for a number of scans equal to its capacity. We set the sliding-window's capacity to the last 5 scans, see §4.2.

Neural architecture. With BlueSeer, we target constrained devices, such as fitness trackers or in-ear earphones that feature a <100 MHz MCU and ~200 KB memory. Thus, we devise a two-layer dense neural network: the input features are fed into (1) a 500-neuron dense layer with Rectified Linear Unit (ReLU); that feeds (2) the dense output layer with softmax function and 7 output classes (see §4.1). To ensure that the model's weights do not monopolize all memory, we employ quantization, fixing weights to integer representations [268]. Our neural network takes 65 KB of memory and requires ~12 ms to infer the environment on a 64 MHz MCU.

Collection and training. We collect 70 600 datapoints from 7 environments, using 62 000 for training and 8 600 for the test dataset for the final evaluation. For each environment, we collect data from different physical locations at different time, within the same city and in nearby locations. For example, for Home, the training set contains samples from 6 locations: three individual houses and three apartments. For Shopping, we feature 5 environments: three different grocery stores, one clothing store, and a shopping mall. The test dataset consists of physically different recordings: we do not separate one supermarket recording into training and test data, but collect two recordings each at a different time, one for training, one for testing. We train the network for 20 epochs, with an exponentially decreasing learning rate initially set to 0.01. Once the network is trained, we quantize its weights and retrain it for 10 additional epochs to mitigate the accuracy drop due to quantization.

3.4 Implementation

We implement BlueSeer for the Zephyr RTOS and use its open-source BLE stack implementation [269]. We use Tensorflow for training and employ quantization-aware retraining for the second training step to ensure that the quantized model is on-par with the original performance. We rely on Tensorflow Lite for Microcontrollers for on-device inference [268].

BlueSeer is platform-independent; we use the nRF52840 SoC (nRF52) to collect data and evaluate the performance of the embedded neural network. The nRF52 features a 64 MHz Cortex-M4 CPU with FPU, 256 KB of RAM, and a Bluetooth 5.2 radio supporting BLE. The nRF52 is a well-representative platform and depicts a performance comparable to commercial fitness trackers.

4 Evaluation

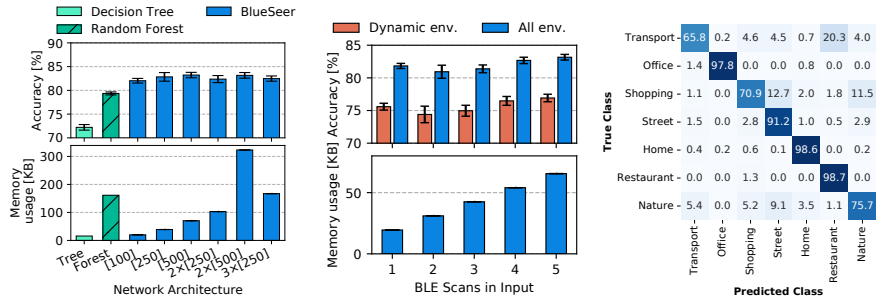
In this section, we experimentally demonstrate the effectiveness of BlueSeer. We first evaluate the impact of the neural network architecture in terms of accuracy and memory usage. Then, we assess the importance of the features produced by the feature extraction process, as well as the impact of the number of scans used as input. Finally, we evaluate the overall performance of BlueSeer on unseen data and its on-device computation, memory, and energy footprint.

Metrics. We evaluate the following set of metrics: (1) Accuracy: Top-1 accuracy obtained from the quantized neural network; (2) Memory: Memory usage to store all weights and temporary computations, both in RAM and ROM (flash); (3) Energy: Energy consumption of running BlueSeer, on the embedded device, including neural network computation and BLE scanning; and (4) Compute-time: Execution time of the neural network on the resource-constrained device.

4.1 Neural Architecture

Scenario. We investigate the impact of the number of layers and neurons on the accuracy of the system and the memory consumed by the model. We evaluate different neural network architectures and compare them to decision trees (DT), a memory and compute-efficient machine learning alternative, as well as against random forests, that are known to improve accuracy compared to DTs while remaining compute-efficient. We limit the DT to a maximum depth of 40 and the random forest to contain a maximum of 10 concurrent DTs. We average results over 10 models using k-fold cross-validation.

Results. Fig. C.3a depicts the accuracy and memory usage of different neural network models after quantization, as well as the performance of decisions trees and random forests. The baseline decision tree achieves 72.2% accuracy and consumes 15 KB of memory, while the random forest composed of ten DTs achieves 79.4% accuracy with 161 KB memory usage, roughly 10× the size of a single DT. An embedded neural network with a single hidden layer achieves 82%, 82.8%, and 83.2% for 100, 250, and 500 neurons, respectively. Memory-wise, these models consume 20 KB, 39 KB, and 70 KB, respectively, where 5 KB are assumed for intermediary results storage. Adding more layers does not improve accuracy further but induces a significant memory overhead:



(a) Accuracy and memory costs of different models.

(b) Number of BLE scans as input.

(c) Confusion matrix.

Figure C.3: Evaluating BlueSeer. (a) A single hidden layer with 500 neurons is sufficient to classify environments accurately and easily fits into memory-constrained devices. (b) By including multiple scans as input to the neural network, BlueSeer reduces the risks of fluctuation in highly-dynamic environments. (c) BlueSeer is able to accurately classify restaurant, home and office samples, but environments with high-mobility are harder to classify. The best out of 10 models achieves 85.5% accuracy.

82.4% accuracy and 103 KB of memory when using two layers of 250 neurons, 83.2% and 318 KB with 2 layers of 500 neurons, and 82.5% and 167 KB for three layers of 250 neurons. We select a neural network comprising one hidden layer of 500 neurons for BlueSeer. For the most memory-constrained hardware, the model with 100 neurons is the best trade-off between accuracy and memory.

4.2 Feature Analysis

Scenario: Features importance. The feature extraction process produces 23 distinct features representing device density, variety, and environment dynamics. We evaluate the importance of the produced features and how they affect the performance of BlueSeer. We withhold a subset of features from the training dataset and compare the achieved accuracy with the model using all features. For all models, we use a single BLE scan as input. We average results over 10 models using k-fold cross-validation.

Results. With all features present and one BLE scan, the baseline model achieves 81.5% accuracy. Removing features related to the environment's variety, such as the number of services, decreases the accuracy down to 77.6%, while removing RSS-related features drops the accuracy to 77.1%, and removing the TX Power information drops to 79.2%. Some redundancy is present in the feature set: removing the number of devices only decreases the accuracy to 80.7%, while removing the number of advertisements received achieves 80.8% accuracy. Device variety, represented by the available services and TX Power, as well as the environment dynamics, with the RSS measures, are important factors to distinguish between environments. The combination of all features provided by the feature extraction process enables the high accuracy

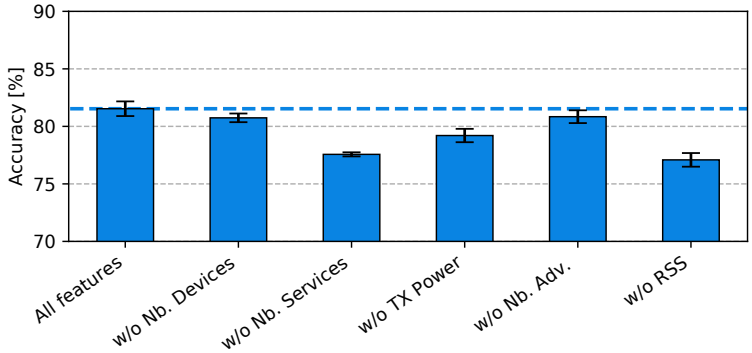


Figure C.4: Feature analysis. The number of services and RSS measures play an important role in distinguishing environments. Some features provide redundancy in the input. The dotted line represents the accuracy when all features are present.

of BlueSeer.

Scenario: Number of scans. We investigate how the number of BLE scans used as input to the model affects the overall accuracy, see §3.3. More scans should avoid fluctuations due to highly dynamic environments, but induce a larger input and slightly larger memory footprint. We average results over 10 models using k-fold cross-validation.

Results. Fig. C.3b depicts the classification accuracy based on the number of BLE scans used as input. The model achieves 82%, 80.7%, 81.2%, 82.5%, and 83.2% accuracy for the input ranging from 1 up to 5 scans, respectively. As the number of scans increases, the model more accurately distinguishes between environments with high dynamics such as transport. Interestingly, the street and shopping categories perform slightly better with one scan than with two scans (street drops from 86% with one scan to 82% with two) before increasing again with three or more scans. Other environments (nature, home, restaurant) always benefit from more scans. Over all environments, it is more beneficial to include more scans; we therefore always include 5 scans for BlueSeer.

4.3 Overall Performance

Scenario: Per-class accuracy. We dissect the performance of BlueSeer's quantized neural network and look at the per-class accuracy on the test dataset. Each class in the test set comprises 1200 elements. We take the model that produces the best accuracy out of 10 trained models using k-fold cross-validation.

Results. Fig. C.3c depicts the confusion matrix of the quantized model's inference over the test set. BlueSeer is able to accurately classify environments with low dynamics such as home, restaurant, and office with 98% or more accuracy. However, BlueSeer is less accurate when it comes to dynamic environments with devices in motion. Shopping shares many similarities with a busy street, where many passersby come and go. As transport covers trains, buses and cars, the class might be too general and could be split into sub-categories. Similarly, nature contains both city parks, where many people visit on sunny

Table C.1: On-device requirements for BlueSeer.

	Feat. Extr.	Model	Others	Total
Flash	1.8 KB	65 KB	46.2 KB (TFLite)	113 KB
RAM	11.9 KB	2 KB	1 KB	15 KB
CPU	<1 ms	~12 ms	3 sec (1 scan)	13 ms
Energy	6.4 µJ	111.3 µJ	56.8 mJ (1 scan)	57.9 mJ

days, with forests, where meeting a passerby is less likely. Higher granularity in the categories could improve accuracy but would require more training data.

Scenario: On-device execution. We now measure BlueSeer's footprint on resource-constrained devices. We use the Tensorflow Lite for Microcontrollers for on-device inference. We measure the ROM and RAM used by the neural network and the library, the feature extraction and inference time, as well as the overall energy cost for BlueSeer.

Results. Table C.1 summarizes all resources used up when performing BlueSeer's inference on the nRF52 SoC (cf. §3.4). BlueSeer uses 113 KB of storage: 65 KB to save the weights, 46 KB for Tensorflow Lite's library. 2 KB of RAM are reserved for dynamic allocation for the input, temporary results, and output, 1 KB for the library, and 12 KB for the packet parsing. It takes less than 1 ms to extract features and ~12 ms to run the inference step on the 64 MHz MCU. Energy-wise, BlueSeer requires 57.9 mJ to perform one scan and inference. Assuming BlueSeer executes one scan and inference every 10 sec and uses an AAA battery with a capacity of 800 mAh, we can run over 62 000 BlueSeer inferences. This represents over a week of constant operation on a single AAA battery.

5 Related Work

Audio-based detection. Several works establish acoustic sensing as an accurate enabler for environment detection. Ma et al. rely on microphones and Hidden Markov Model classifiers to distinguish between 12 environments such as bus, car, street, and office, from 3-second long audio recordings and achieve up to 93% accuracy [235]. However, the authors do not discuss the problems of privacy arising from relying on microphones to infer surroundings. Heittola et al. represent audio fingerprints as histograms and compare new recordings with previous histograms to distinguish between ten environments [237]. Choi et al. combine microphone and camera inputs to detect the user position and activity [238]. Acoustic-based systems are also proven to improve Human Activity Recognition (HAR) [270].

Sensor-based detection. Accelerometers play a notable role in transportation mode detection, as well as HAR. Liang and Wang introduce a CNN to parse a smartphone's accelerometer data and distinguish which transport (such as bus, car, bike) the user is using [239]. Kern et al. distribute accelerometers over the body to perform HAR [271]. Sankaran et al. show that a barometer can also help distinguish between idle, walking, and in-vehicle users [272]. Yang et al. rely on channel state information to detect building occupancy with only

a wifi radio [273]. However, they only detect if people are present in a room and how many, but do not detect the users' activities.

Wireless-enabled localization. Several works investigate the use of Bluetooth and BLE packets as a driver for localization and distance estimation [214, 262]. Bertuletti et al. demonstrate that RSS measures are noisy and lead to a 30% distance estimation error [262]. Zhuang et al. achieve <3m localization using BLE beacons [214]. Ultra-wideband radios are much more accurate and typically achieve <10 cm localization error, sometimes down to the centimeter [274, 275]. BlueSeer does not aim at solving localization but rather classifies the general environment of the device.

6 Conclusion

Environment detection allows devices to react to their environments: smartphones can automatically switch to silent mode when entering a theater, while headphones can adapt their noise cancellation to their surroundings. Although modern phones can rely on many sensors to infer their environments, such as GNSS chips and cameras, small IoT wearables lack this diversity and must rely on the few, energy-efficient components they already incorporate. This paper shows that a Bluetooth radio is the unique component required to classify the current environment with high accuracy: the BLE packets received by a device form a wireless fingerprint, unique enough to infer the correct environment. We present BlueSeer, an AI-driven environment-detection system specifically tailored to resource-constrained wearables: from BLE packets, BlueSeer extract 23 unique features that are fed to a quantized, embedded neural network. Periodically, BlueSeer scans the wireless medium for BLE advertisements and executes an embedded neural network to classify between 7 categories: home, office, shopping, transport, nature, street, and restaurant. We show that BlueSeer achieves up to 84% accuracy on resource-constrained devices, while requiring only 65 KB of memory, and takes ~12 ms to execute on a 64 MHz microcontroller-unit.

D

Grace: Low-Cost Time-Synchronized GPIO Tracing for IoT Testbeds

---

Laura Harms, Christian Richter, Olaf Landsiedel

Elsevier Computer Networks, vol. 228, 2023, p. 109746.

Abstract

Testbeds have become a vital tool for evaluating and benchmarking applications and algorithms in the Internet of Things (IoT). IoT testbeds commonly consist of low-power IoT devices augmented with observer nodes providing control, debugging, logging, and often also power-profiling capabilities. Today, the research community operates numerous testbeds, sometimes with hundreds of IoT nodes, to allow for detailed and large-scale evaluation. Most testbeds, however, lack opportunities for tracing distributed program execution with high accuracy in time, for example, via minimally invasive, distributed GPIO tracing. And the ones that do, like Flocklab, are built from custom hardware, which is often too complex, inflexible, or expensive to use for other research groups.

This paper closes this gap and introduces Grace, a low-cost, retrofittable, distributed, and time-synchronized GPIO tracing system built from off-the-shelf components, costing less than €20 per node. Grace extends observer nodes in a testbed with (1) time-synchronization via wireless sub-GHz transceivers and (2) logic analyzers for GPIO tracing and logging, enabling time-synchronized GPIO tracing at a frequency of up to 8 MHz. We deploy Grace in a testbed and our evaluation shows that it achieves an average time synchronization error between nodes of 1.53 µs using a single time source, and 15.3 µs between nodes using different time sources, sufficient for most IoT applications.

1 Introduction

With more than 10 billion connected IoT devices deployed today, and an estimated 30 billion devices by the beginning of the next decade [276], the Internet of Things enables new applications in our connected and data-driven society. Their connected and often distributed nature makes extensive testing, evaluation, and benchmarking a must to ensure proper functionality and performance of applications, algorithms, and protocols before their actual deployment.

Simulation [80, 277] allows for high-level insights into protocols and algorithms. It makes it possible to inspect these in a controlled environment and evaluate their general correctness. However, simulation cannot capture all details of a real environment, nor is it capable of evaluating the performance of protocols on real hardware. Yet, such evaluation in real-world environments is necessary to ensure the correct functionality in nondeterministic environments. Therefore, the research community commonly uses testbeds: deployments of (low-power) IoT devices co-located with observer infrastructure, typically an edge device – like a Raspberry Pi – for instrumentation, logging, and deployment control.

While testbeds provide real-world insights and are today's established tool for evaluating distributed IoT applications, most lack one essential capability: The capability to non-intrusively – or with minimal intrusiveness – track the execution of distributed protocols and algorithms. And the ones that do offer these capabilities use custom hardware that is not easily replicable and integrable into existing testbeds. For example, for debugging and evaluating (real-time) protocols, we often need insight into the execution and states within the hardware. Without this insight, we can only treat the hardware system as a black box. For non-distributed settings such as traditional software development, one commonly uses debuggers, with which one can halt program execution and inspect the system's state. In distributed settings, we cannot halt the operation of nodes as both the environment continues to change, and all other nodes will also continue their operation. Another common way is printing messages during operation, usually through a serial interface. However, printing takes several hundreds of microseconds, which leads to side effects on program executions and limits accurate timestamping. It might even break the timing in timing-critical sections of a program, leading to missed deadlines. Or the removal of the print statements after evaluating the system might change the timing that much that it introduces bugs not previously present.

The third way of gaining insight is through tracing the General-Purpose Input/Output (GPIO) pins of a processor or microcontroller. Toggling GPIO pins offers a minimally intrusive way of communicating timing-correct information on the operation to the outside world. A logic analyzer can record the GPIO traces to evaluate these later. While logic analyzers provide us with a time-accurate trace of the execution of a program, they commonly only provide insights into one device due to the physical distance between devices. However, in a distributed communication system, it is essential to know how multiple devices interact with each other and at what exact point in time, or how much time passes between the same operation on multiple devices. For example, Time-Division Multiple Access (TDMA) protocols like Glossy [9] or Time-Slotted

Channel Hopping (TSCH) [6] are time-critical protocols that synchronize their communication; and, among others, LWB [65] and Chaos [59] enable multiple devices to send data concurrently in a time-synchronized fashion. To evaluate the synchronization of protocols like these and the interaction between multiple devices, we need an external system that itself is time-synchronized. To achieve this, we require a GPIO tracing system, which performs a time-synchronized tracing on all devices.

Many means of time-synchronization exist, including the Network Time Protocol (NTP) and the Global Positioning System (GPS). However, none of them offers a low-cost, low complexity solution that is both available at indoor testbed locations and offers the required accuracy. For example, the accuracy of GPS would be favorable, however, GPS requires direct line of sight to several satellites, making it only usable outside or close to a window. NTP on the other hand is available anywhere where a device has internet access, including on observer devices in testbeds. However, it only offers accuracy in the order of milliseconds, which is not sufficient for precisely timestamping events in IoT protocols.

There are distributed, time-synchronized GPIO logging systems implemented in existing testbeds [84, 86, 93]. However, they use custom hardware with FPGAs and require a specific testbed observer platform throughout the testbed. This limits their adoption into other testbeds, especially those that already exist and use different hardware and observer platforms.

In this paper, we present Grace, a low-cost, retrofittable, distributed, and time-synchronized GPIO tracing system using off-the-shelf components. Grace extends observer nodes with (1) time-synchronization via wireless sub-GHz (433 MHz) transceivers and (2) logic analyzers for GPIO tracing and logging. Using sub-GHz wireless, Grace enables building-wide time-synchronization from a single central node performing unidirectional single-hop RBS-like time-synchronization. Extending Grace and using multiple synchronization nodes even extends the covered area to larger buildings or offers even campus-wide time-synchronization while keeping the single-hop nature of our synchronization system. Further, we devise a software framework to enable extensive tracing capabilities using this hardware. In our evaluation, we show that Grace is capable of continuously logging sparse data as commonly produced when debugging IoT systems, such as wireless protocols, at a rate of 8 MHz. Moreover, we show that we achieve a time-synchronization of on average 1.53 µs between nodes using the same time source, which, as we argue, is sufficient for most applications. Moreover, we extend our initial work on Grace [251] to function in larger networks by introducing multiple timesources which provide an average time-synchronization of 15.3 µs between nodes and sub-deployments using different time sources.

This paper is an extension of [251], which made the following contributions:

- We present Grace, a low-cost time-synchronized GPIO tracing system for IoT testbeds.
- We implement Grace using off-the-shelf hardware to enable easy adoption in other building-wide testbeds and make both the software and the hardware setup openly[1] available.

[1]Available as open-source at: `https://github.com/ds-kiel/grace`

- We show Grace's low cost of less than €20 per node.
- We evaluate Grace, showing its degree of time-synchronization between nodes of on average 1.53 µs, while not exceeding a worst-case synchronization of 3.75 µs.

Now, this paper adds the following new contributions:

- We significantly extend the discussion of our design, its algorithms, and deepen the discussion of Grace throughout the paper. We add a discussion of the necessity of a system like Grace, a more comprehensive explanation of the design, including additional algorithms, as well as a discussion of the scalability of Grace.
- We introduce multiple types of synchronization nodes, enabling time-synchronization for both building-wide and campus-wide testbeds.
- We discuss and evaluate the intrusiveness of GPIO tracing on IoT platforms.
- We evaluate the time synchronization performance of Grace when using multiple time sources and show that its degree of time-synchronization is on average around 15.3 µs between nodes using different time sources.

The remainder of this paper is organized as follows. Section 2 gives necessary background information, followed by a discussion of related testbeds and GPIO tracing testbed systems in Section 3. Section 4 introduces Grace's design, and Section 5 presents our experimental evaluation. We conclude our paper in Section 6.

2 Background

This section provides the necessary background for the remainder of this paper. We introduce (1) the concept of time synchronization with a focus on (2) the Network Time Protocol (NTP) and (3) the Reference Broadcasting System (RBS). Afterward, we provide general background on (4) Logic Analyzers.

2.1 Time Synchronization

Most electronic computing devices use a crystal oscillator as a basis for their clock. These oscillators operate at a certain frequency, but usually do not perfectly hold their nominal frequency. No oscillator is perfect, and physical variations like temperature or air pressure add to the oscillators' frequency variation. While these drifts are commonly negligible in stand-alone single computer setups, they impose a challenge on distributed computing and communication systems. These systems require a tight synchronization of the individual clocks. For time-sensitive applications, these clocks have to fulfil one or both of these metrics: *precision* and *accuracy*. The notion of *precision* (β) defines the maximum time error between two clocks (p, q) of a system:

$$\forall t, \forall p, q : |C_p(t) - C_q(t)| \leq \beta \tag{D.1}$$

The notion of *accuracy* (α) describes a clock's difference towards a reference timescale [71]:

$$\forall t, \forall p : |C_p(t) - t| \leq \alpha \tag{D.2}$$

A common reference timescale is UTC, which the Network Time Protocol (NTP) (see Section 2.3) uses.

Depending on the importance of accuracy or precision, different synchronization approaches like NTP (see Section 2.3) or the Reference Broadcasting System (RBS) (see Section 2.4) are suitable. Within the next sections, we describe these two in more detail.

2.2 Global Positioning System (GPS)

The Global Positioning System (GPS) is a global navigation satellite system (GNSS) that provides positional information on earth. Moreover, it contains atomic clocks and is thus capable of providing accurate timekeeping functionalities. Common GPS receiver modules are capable of generating a precise 1 pulse per second (1-PPS) signal from the received data. Moreover, they output NMEA 0183 [278] sentences containing additional information. NMEA 0183 is a standard for communication between marine electronics including GPS controlled by the National Marine Electronics Association (NMEA). For example, the RMC sentence [279] (Recommended minimum specific GPS/Transit data) includes, i.a., time, location, and date.

2.3 Network Time Protocol (NTP)

The most widely used time synchronization protocol for distributed systems is the Network Time Protocol (NTP) [71]. It is the default protocol used by computers and most devices directly connected to the Internet, and builds the baseline for other protocols. For time synchronization, a device contacts an NTP server to receive the server's local time. From the received timestamps and the device's local timestamps of sending the request and receiving the response, the device can compute the round trip latency and thus determine its offset from the reference clock of the server.

In NTP, clocks are synchronized to UTC. As not each NTP server can be equipped with a reference clock, e.g., a GPS receiver, NTP builds a hierarchical structure of servers. This structure uses so-called stratum levels. A stratum-1 server is equipped with a reference clock. Each server is a stratum-$(k+1)$ server if the server it contacts to synchronize to is a stratum-k server. NTP is known to achieve accuracy between 1 and 50 ms.

2.4 Reference Broadcasting System (RBS)

The reference broadcasting system (RBS) [71] differs significantly from methods like NTP. It does not assume the existence of an accurate clock (e.g., a UTC clock) within the network. Instead, it merely has the goal of network-internal clock synchronization. RBS is a wireless, physical layer time synchronization method. Moreover, contrary to other methods, where a node contacts a timeserver, in RBS, a time source broadcasts a reference signal to all nodes within the network. Every node generates a timestamp with its local clock on

reception of the synchronization signal. As RBS only has a single sender that reaches all receivers, most parts of the critical path are eliminated. In a wireless network, the transmission time to all receivers is roughly the same, with a negligible offset. Thus, the critical part is only the reception (and timestamping) of the broadcast packet at the receivers. To reduce jitter introduced upon reception, RBS performs multiple broadcast rounds, and nodes exchange each other's delivery times to estimate their mutual, relative offset.

2.5 Logic Analyzer

A logic analyzer records the physical state of one or more signals over time. Logic analyzers commonly trace digital signals. Some logic analyzers can even record analog waveforms (tracing signals voltage level), like oscilloscopes. Thus, logic analyzers commonly replace an oscilloscope, especially when working with digital electronic components. To process the trace of a logic analyzer, several software solutions working with logic analyzers have built-in features to not only display the recorded traces, but even decode protocols like the Serial Peripheral Interface (SPI) communication protocol to use a logic analyzer to debug communication between electronic components [280].

3 Related Work

In the past 18 years, starting with MoteLab [99], the research community proposed several testbed architectures and currently operates several testbeds for testing, debugging, evaluating, and benchmarking low-power IoT protocols, with tens to thousands of nodes. A selection of these are Flocklab [84], Flocklab 2 [85, 86], D-Cube [87], FIT IoTLab [98], Indriya [88], Indriya2 [89], OpenTestBed [90], WUSTL testbed [91], TWIST [281], Kansei [282], SmartSantander [283], VIADUCT [92], Tracelab [93], Aveksha [94], Minerva [95], HATBED [96], as well as the previous version of our own IoT testbed [97]. While these testbeds and testbed architectures have different goals, they all are usable for testing and evaluating IoT protocols. For example, OpenTestBed [90] is a movable testbed which can easily be placed wherever needed by using a node design that doesn't require any fixed infrastructure. SmartSantander [283] proposes a city-scale testbed with up to 20000 devices. VIADUCT [92] bridges the gap between testing infrastructure and real-world IoT deployments. FIT IoTLab [98] offers a wide range of different IoT platforms and in addition to nodes with a fixed location they also offer moving testbed nodes mounted on robots. D-Cube [87] takes a different approach by building a testbed intended for benchmarking IoT protocols in a wireless environment with a controllable amount of interference.

While all of these testbeds offer the logging of serial output, only a subset has GPIO interfacing capabilities. To our knowledge, the first IoT testbed offering GPIO tracing capabilities is Flocklab [84]. Flocklab's GPIO tracing system directly uses the observer for GPIO tracing and can trace up to 5 GPIO pins at a sampling rate of up to 10 kHz. For time-synchronization, the system uses NTP, reaching a precision of 40 µs. Next to GPIO tracing, Flocklab also allows GPIO actuation as well as power profiling. With Tracelab [93], Lim et

al. extend Flocklab by a more capable GPIO acquisition system based on an FPGA. They achieve a short-term sampling frequency of up to 100 MHz, and a continuous sampling frequency of around 285 kHz. For time synchronization, they use Glossy on 868 MHz with an FPGA-based clock correction control loop, achieving a maximum time-synchronization error of 1.5 µs.

Other testbed architectures like Aveksha [94], Minerva [95], and HATBED [96] use different J-Link tracing methods, including tracing the program counter, or watchpoint tracing in a non-intrusive way. Aveksha [94] uses an On-Chip Debug Module (OCDM) to non-intrusively observe the inner workings of embedded processors. It can trace specific program counter addresses, the entry, and exit of tasks and interrupt service routines as well as user-defined events. It uses a method of polling the JTAG interface, allowing a polling period of 30 µs. While Aveksha can trace internal events of embedded processors, it is not suitable for distributed debugging, as it does not perform synchronized timestamping. Minerva [95] is closely related to Aveksha, as it also offers non-intrusive tracing in testbeds using a JTAG interface. However, Minerva additionally offers time-synchronized tracing, synchronous stopping of the execution to collect memory snapshots and network-wide assertions based on the traced data. Minerva uses NTP for time synchronization and timestamping reaching a millisecond precision. Thus, it is not suitable for tracing time-critical applications. HATBED [96] uses the on-chip debugging capabilities of ARM Cortex-M3/M4 processors. It supports `printf` logging using the Instrumentation Trace Macrocell (ITM) and watchpoint-logging using the Data Watchpoint and Trace Unit–(DWT). HATBED uses a logic analyzer (the same chipset we use for our work) for tracing the ITM and DWT output.

A more recent work is Flocklab 2 [85, 86], which uses the programmable real-time unit (PRU) of a Beaglebone Green for GPIO tracing. For time synchronization, the system uses GNSS with an accuracy of approx. 50 ns where available, and the Precision Time Protocol (PTP) with an accuracy of approx. 1 µs at all other locations. Next to GPIO tracing, it also supports Serial Wire Debug (SWD) tracing through a J-Link debug probe.

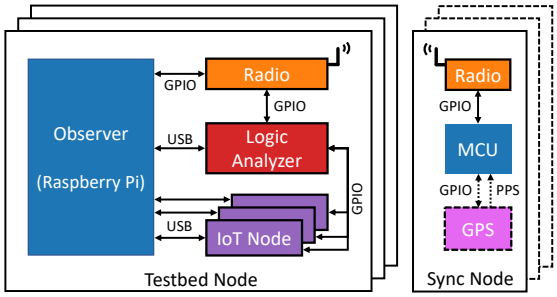
Regarding time synchronization, Grace has the highest similarity with Tracelab. However, instead of performing the time synchronization in hardware on an FPGA, we enable it at a higher precision in software on off-the-shelf hardware, a Raspberry Pi, when processing the logic analyzer's traces. Regarding GPIO tracing, we differ from all these solutions in that we use low-cost logic analyzers and do not depend on a specific observer platform.

4 Design

Our design of Grace enables a time-synchronized use of GPIO tracing in building-wide or campus-wide testbeds while only using off-the-shelf components. Actuating GPIO pins is minimally intrusive and thus has commonly negligible influence on a system's timing.

4.1 Design Overview

The diagram in Figure D.1a illustrates the general idea of Grace. For building-wide testbeds, a single synchronization node repeatedly sends out a time signal.



(a) Grace's node types. The testbed node consists of an observer platform and one or more IoT nodes. We add a logic analyzer and a radio for time-synchronized GPIO tracing. The synchronization node consists of a microcontroller and a radio generating the time signal. In case of larger deployments, we can use multiple synchronization nodes and add an otherwise optional GPS receiver. The GPS receivers on each synchronization node synchronously generate a signal once a second (1-PPS) that we use for time-synchronizing the synchronization nodes.

(b) A testbed node. We see from left to right a Zolertia Firefly, a Raspberry Pi with our custom HAT connecting the CC1101 radio on top and the logic analyzer (top of the box) to the Pi, a Telosb mote, and a Nordic nRF52840-DK.

Figure D.1: Design Overview of Grace.

Each testbed node, equipped with a receiver, receives this time signal. Using the signal, each observer synchronizes its GPIO tracing clock. This synchronization of the tracing clocks enables the post-processing to match the GPIO traces of the different devices and build a common trace over all distributed devices. Once a testbed spans a wider area, such as like multiple buildings or a campus, a single synchronization node, will not be able to reach all nodes. For such large testbeds, Grace relies on multiple synchronization nodes, that synchronously send out the same timestamp.

Grace's design concerns and extends solely the testbed's observer infrastructure, and it does not impact the program execution on our target platforms (see IoT nodes in Figure D.1a). All we need on the IoT nodes are available GPIO pins, which almost every hardware has. Through our design, the IoT nodes gain the option to output information through state changes of their GPIO pins in addition to their serial logging capabilities over USB.

Within the following sections, we motivate the need of GPIO tracing for low-intrusive time stamping, and describe the design of the individual components of Grace in detail. For the synchronization node, we split our design into two distinct parts; one using a single synchronization node and one using multiple synchronization nodes following Grace's extended design. Next, we discuss our time-error correction algorithm and discuss the system's integration into an existing testbed.

4.2 Low-intrusive time stamping

Before discussing our design in detail, we motivate the necessity of GPIO tracing to achieve low-intrusive time stamping. For example, if we are developing a protocol for wireless communication of IoT nodes, we might come to the point that we want to identify which nodes turn on their radio at what time, e.g., to evaluate the accuracy of the protocol's time synchronization algorithm. For this, we require a system that precisely timestamps the event and does not delay the processing of received data. A time-synchronized serial logging system using `print`-statements could enable us to timestamp such an event. However, the duration it takes to output anything, even a single byte over a serial interface, has a significant influence on the program's timing and on the accuracy of the timestamp. For example, transmitting a single byte takes 86.8 μs at a baud rate of 115200 baud/s (8 data bits, 1 start bit, 1 stop bit). Changing a state on a GPIO pin is much faster and takes only a few clock cycles, resulting in an overhead of no more than tens or hundreds of nanoseconds. For example, the nRF52840 [192] takes 140.5 ns for switching the state of a GPIO pin (cf. Section 5.2).

In Section 5.2, we evaluate the exact timing differences for one of our target platforms. Moreover, precisely time synchronizing logic analyzers that are fully under our control, is simpler than precisely time synchronizing general purpose Linux operating systems. Therefore, a time-synchronized GPIO tracing system, like the one we present in this paper, can offer precisely time-synchronized, low-intrusive time stamping for evaluation purposes of IoT systems and protocols.

4.3 Synchronization Node

A synchronization node in Grace is a node (physically) independent of the testbed and its observers and IoT nodes. It consists of a microcontroller and a 433 MHz radio (see Figure D.1a). In case of multiple synchronization nodes, or in need of an accurate time source, it also contains a GPS receiver. We use the 433 MHz band, which is an ISM band available in the International Telecommunication Union's (ITU) region 1 (i.a., Europa and Africa) [284]. Creating a design with 433 MHz radios allows the use of only one synchronization node for a typical building-scale testbed, due to the extended range of 433 MHz radios compared to the range of radios using higher frequencies. Moreover, the 433 MHz band is well outside the bands usually used for IoT research: the 2.4 GHz and 868/915 MHz bands. Nonetheless, please note that the design of Grace is generic and independent of the 433 MHz band. In countries where it is not available as open ISM band, one can use other frequency ranges. For larger testbeds beyond building scale, we extend our initial design and use multiple of our synchronization nodes communicating on different channels. As the testbed nodes all have to receive a timestamp at the same time, the synchronization nodes have to send the same synchronization timestamp at the same time. To synchronize the synchronization nodes, we have to discipline them with an external clock. We choose to use GPS receivers and their 1-PPS (1 pulse per second) signal for this purpose.

Single Synchronization Node In case of a single synchronization node, the node generates and transmits a timestamp at a configurable time interval.

Algorithm D.1 Timestamping and transmission on PPS interrupt

Input: $timestamp$, $skipped_{pps}$, $state_{radio}$, $timer$, $count_{timer}$
Ensure: $timestamp$, $timer$

```
1: if timestamp = 0 then
2:    initialize time signal
3: else
4:    stop timer
5:    count_timer ← 0
6:    if skipped_pps = 1 then
7:       execute initialize time signal algorithm
8:       skipped_pps ← 0
9:    else
10:      timestamp ← timestamp + 1
11:      if state_radio = TX_MODE then
12:         transmit timestamp
13:         busy wait for transmission to finish
14:      end if
15:      start timer //1 second timer
16:   end if
17: end if
```

In our case, we use an interval of one second, inspired by the one pulse per second signal generated by GPS receivers. The timestamp we send contains a counter value that represents the time that has passed since turning on the synchronization node. With only one synchronization node, we do not require a globally accurate clock, but rather a single time source within the vicinity of the network. Thus, the synchronization node can use its microcontroller's clock to generate the timestamps. Even variations in the time between two generated timestamps are not a problem as we merely need a common time reference and not a globally accurate clock.

GPS-disciplined Synchronization Node If we nonetheless require a globally accurate clock, or in case of multiple synchronization nodes, we require a different design, extending Grace's initial design. This node type also transmits a time signal once a second. However, instead of generating a timestamp locally, we send the Unix timestamp [285] and each of the synchronization nodes transmits the timestamp at the same time. We compute this timestamp from the GPS data we receive and use the one pulse per second signal (1-PPS) as a trigger to send the timestamp. Algorithm D.1 shows the procedure of timestamping and transmitting the timestamp. On the first PPS signal, when the timestamp is 0, we initialize our Unix timestamp. We use the GPS NMEA RMC string [279], containing both the current date and UTC time, as a basis for calculating the initial timestamp. We compute the timestamp according to Algorithm D.2. In case the synchronization nodes misses or skips a 1-PPS signal due to, e.g., the lack of a GPS fix, we reinitialize the Unix timestamp according to the same procedure. We detect this skipping of a signal, using a local 1-second timer. If we did not skip a signal, we increment the timestamp by 1 and transmit the time signal. Using the GPS time and the 1-PPS signal,

Algorithm D.2 UTC to Unix timestamp conversion

Input: *year*, *month*, *day*, *hour*, *minute*, *second*
Ensure: *timestamp*

$timestamp_{reference} \leftarrow 1640995200$ {2022-01-01T00:00:00+00:00}
$days_{previous\_months} \leftarrow [0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334]$
$leap\_years \leftarrow 0$
$y \leftarrow 2022$
while $y < year$ **do**
  if $y \bmod 4 = 0$ **then**
    $leap\_years \leftarrow leap\_years + 1$
  end if
  $y \leftarrow y + 1$
end while
$days \leftarrow (year - 2022) * 365 + leap\_years$
  $+days_{previous\_months}[month - 1] + day - 1$
if $(month > 2) \wedge (year \bmod 4 = 0)$ **then**
  $days \leftarrow days + 1$ {current year is a leap year}
end if
$timestamp \leftarrow timestamp_{reference} + (days * 86400) + (hour * 3600) + (minute * 60) + second$
return *timestamp*

we can ensure that all synchronization nodes generate the same timestamp at the same time. By only sending the timestamp if we received at least two consecutive 1-PPS signals, we do not need to read the timestamp from the GPS module, but can instead increase the timestamp locally and can transmit this, ensuring a minimal compute time before transmitting the timestamp. Moreover, performing the same instructions on each synchronization nodes and not firstly communicating with the GPS module, ensures that the actual time we transmit the signal does not vary significantly between the synchronization nodes.

Regardless of the type of synchronization node, the node's radio broadcasts the timestamp to all testbed nodes in range. As – in the case of multiple synchronization nodes – all send out the timestamp at the same time, we can expect wireless interference at testbed nodes close to more than one synchronization node. We avoid this interference by using different frequency channels for different synchronization nodes. Each testbed node is preconfigured to listen on one of these channels.

The approach of sending a timestamp at a regular interval (e.g., once a second) to all testbed nodes in a single(-hop) wireless broadcast follows the approach of the reference broadcasting system (RBS) (see Section 2.4). Through the single broadcast and the close distance of all nodes to its synchronization node, we have a low signal propagation delay to all testbed nodes. It is low enough that all nodes will receive the time signal with a negligible time offset during the same logic analyzer sampling period or latest with an offset of a few sampling periods for physically large testbeds. For example, when using a logic analyzer with a sampling frequency of 8 MHz, and a sampling period of 125 ns, the possible distance between two nodes to receive the timestamp on

Algorithm D.3 Process Bulk Data

Input: $data$, $state_{prev}$ (previous GPIO state), $mask_{active}$ (active channels), $channel_{time}$ (time signal channel)

```
1:  for each sample ∈ data do
2:    if sample ≠ state_prev then
3:      changed ← (sample ⊕ state_prev) ∧ mask_active {bitwise}
4:      if changed then
5:        for each channel do
6:          if changed ∧ (1 << channel) then
7:            if channel = channel_time ∧ sample[channel] = 1 then
8:              execute reference time signal algorithm (Algorithm D.5)
9:            else
10:             execute GPIO signal algorithm (Algorithm D.4)
11:           end if
12:         end if
13:       end for
14:     end if
15:   end if
16:   state_prev ← sample
17:   execute tick algorithm (Algorithm D.6)
18: end for
```

the same sample is about 37.5 meters. Any two nodes which have a distance offset from the time source of less than half that distance (18.7 meters) will receive the timestamp in the same logic analyzer sample. Note that we do not intend to synchronize the clocks of testbed observers, but rather synchronize the timestamps of the logic analyzers.

4.4 Testbed node

A testbed node consists of a controller (e.g., a Raspberry Pi), often also denoted as observer, and one or more low-power IoT devices as target platforms (see Figure D.1a). The target platforms expose GPIO pins that are to be traced. The system we describe here concerns solely the controller and does not pose any overhead on the IoT platforms. To enable this tracing, we devise a system consisting of a USB logic analyzer, and a radio that together can be retrofitted to any testbed by connecting them directly to the controller node. In addition, we devise a small software library for data acquisition and control of these devices for deployment on the controller node. We use the logic analyzer for the GPIO tracing and the radio for receiving the timestamp from a synchronization node. We reserve one of the logic analyzer's pins for the radio. All other GPIO pins are available for tracing the target platforms' GPIO pins. Once the radio receives a signal from the synchronization node, it turns on its GPIO pin connected to the logic analyzer. This notifies the process, running on the controller node and handling the logic analyzer's input of a new timestamp. Moreover, it pinpoints the reception of the time signal to an exact tick of the logic analyzer. In other words, we can match the reception time of the synchronization signal to a local timestamp of the GPIO tracing system. This

allows us to perform error correction on the local time and thus have a notion of synchronization for combining the recorded traces of different devices in post-processing.

Algorithm D.4 Handle GPIO signal

Input: TICKS_PER_NANOSECOND, $state_{clock}$, $seconds$, $accumulator$, $state_{gpio-pin}$, $channel_{event}$
Output: trace object (to be written to file)
1: **if** $state_{clock} = \text{FREQ}$ **then**
2: $timestamp \leftarrow 10^9 * seconds + \frac{accumulator}{\text{TICKS\_PER\_NANOSECOND}}$
3: **create** trace object containing $timestamp$, $state_{gpio-pin}$, and $channel_{event}$
4: **return** trace object
5: **end if**

Within the following sections, we describe the different components of the GPIO tracing and the time synchronization.

4.5 GPIO Tracing

For GPIO tracing, Grace employs a USB-driven logic analyzer. This logic analyzer has to be able to trace sparse amounts of data on multiple GPIO pins and write the traces without prior processing to the USB buffer.

4.6 Trace Data Processing

The algorithms at the observer processes the incoming data in bulks. The starting point for the bulk data processing is Algorithm D.3. We perform different actions based on the changes present in each data sample. Each data sample is one recording of the logic analyzer. We compare each sample to its previous one. If there are changes present, we identify the corresponding channels of the logic analyzer. Depending on the channel's role, we perform further actions. If the state of the channel corresponding to the radio changed and that pin turned on, we know that we received a new time signal. We describe the algorithm for processing this time data in the following section (Section 4.6.1). If we detect a change on one of the traced GPIO pins, we timestamp the event (if we previously received at least two global timestamps) according to Equation D.3 and Algorithm D.4 and hand it over for further processing. Lastly, we perform one clock tick of the logic analyzer updating its timestamp for the next sample (see Section 4.6.2).

$$timestamp \leftarrow 10^9 * seconds + \frac{accumulator}{ticks\ per\ ns} \tag{D.3}$$

For simplicity and without losing generality, we assume a timestamp interval of one second for the algorithms. The timestamp interval is easily adjustable to a different timescale.

Algorithm D.5 Handle reference time signal

Input: WEIGHT, TICKS_PER_SECOND, $state_{clock}$, $ticks_{nominal}$, $ticks_{actual}$, $seconds$, $freq_{nominal}$, $accumulator$
Output: $state_{clock}$, $freq_{adj}$, $offset$, $offset_{adj}$, $error_{remaining}$, $seconds$, $seconds_{previous}$, $ticks_{actual}$

1: $seconds_{ref} \leftarrow$ read timestamp from radio
2: **if** $state_{clock} =$ WAIT **then**
3: $seconds \leftarrow seconds_{ref}$
4: $state_{clock} \leftarrow$ OFFSET
5: **else**
6: $factor \leftarrow \frac{(seconds_{ref} - seconds_{previous}) * ticks_{nominal}}{ticks_{actual}}$
7: $freq_{adj} \leftarrow freq_{nominal} * factor$
8: **if** $seconds_{ref} = seconds + 1$ **then**
9: $error_{remaining} \leftarrow$ TICKS_PER_SECOND $- accumulator$
10: $offset \leftarrow accumulator -$ TICKS_PER_SECOND
11: **else if** $seconds_{ref} = seconds$ **then**
12: $error_{remaining} \leftarrow accumulator$
13: $offset \leftarrow accumulator$
14: **end if**
15: $offset_{adj} \leftarrow offset / ticks_{nominal}$
16: $ticks_{actual} \leftarrow 0$
17: **end if**
18: $seconds_{previous} \leftarrow seconds_{ref}$
19: **return**

4.6.1 Time Error Correction

Once the observer's bulk data processing algorithm (Algorithm D.3) detects the reception of a new time signal, we execute Algorithm D.5. This algorithm essentially determines the time increment added for each sample recorded by the logic analyzer. At first, it reads the received data (timestamp) from the radio and sets the radio back into receive mode, which prepares the radio for receiving the next timestamp and turns off the radio's GPIO pin. Now we have the global timestamp and the exact tick it was received on. The processing of it differs depending on the state the GPIO tracing clock-correction system is in. The clock correction has two different states, `WAIT`, and `OFFSET`. Initially, we start in state `WAIT` until we process our first time signal.

When receiving the first time signal, the algorithm saves the received timestamp as the current time with respect to the current sample, and as the previous timestamp for the algorithm's next iteration. Moreover, it changes the clock's state to `OFFSET`.

When the system is in state `OFFSET`, we start by calculating a factor after receiving and reading the reference time ($seconds_{ref}$):

$$factor \leftarrow \frac{(seconds_{ref} - seconds_{previous}) * ticks_{nominal}}{ticks_{actual}} \tag{D.4}$$

This factor determines how much faster or slower the logic analyzer clock ran since receiving the previous timestamp. Please note, that it is not necessary

Algorithm D.6 Tick

Input: TICKS_PER_SECOND, $state_{clock}$, $freq_{adj}$, $offset_{adj}$, $error_{remaining}$, $seconds$, $ticks_{actual}$, $accumulator$
Output: $offset_{adj}$, $error_{remaining}$, $seconds$, $ticks_{actual}$, $accumulator$

```
1: if state_clock = OFFSET then
2:    accumulator ← accumulator + freq_adj − offset_adj
3:    if offset_adj < 0 then
4:       error_remaining ← error_remaining + offset_adj
5:    else
6:       error_remaining ← error_remaining − offset_adj
7:    end if
8:    if error_remaining ≤ 0 then
9:       offset_adj = 0
10:   end if
11:   if accumulator > TICKS_PER_SECOND then
12:      seconds ← seconds + 1
13:      accumulator ← accumulator − TICKS_PER_SECOND
14:   end if
15:   ticks_actual ← ticks_actual + 1
16:   return
17: end if
```

that we receive every timestamp. We just need any timestamp after the previous one. The factor uses both the current timestamp ($seconds_{ref}$) and the previously received timestamp ($seconds_{previous}$), as well as the nominal number of ticks ($ticks_{nominal}$) that should pass within a second (e.g., 8 000 000 at 8 MHz) and the actual number of ticks passed since the previous reception of a timestamp ($ticks_{actual}$). If this factor is 1, the clock of the logic analyzer is running at its nominal frequency. If the factor is <1, the logic analyzer's sampling frequency is too high and if the factor is >1, its frequency is too low. Using this factor, we can adjust the frequency value:

$$freq_{adj} \leftarrow freq_{nominal} * factor \tag{D.5}$$

Phase offset: Moreover, the algorithm performs an additional phase offset adjustment. If the logic analyzer's frequency is not exactly the nominal frequency and neither a multiple of it, there will remain an offset of ticks the logic analyzer's clock is ahead or behind the reference clock. This offset is a phase offset, which we also need to handle when increasing our clock. To be able to correct this phase offset, we calculate the error to the closest second and an adjusted offset to adjust the phase when increasing the tick counter for each logic analyzer sample. Lastly, we reset the tick counter ($ticks_{actual}$) to zero.

Independent of the clock's state, we save the previous timestamp for the algorithm's next iteration.

4.6.2 Clock Tick

Algorithm D.6 performs the frequency and phase adjustments computed in Algorithm D.5 and discussed above. This algorithm only executes once the system has received an initial time value and thus is either in state `OFFSET`. The algorithm design follows closely the precision system clock design of NTP [75]. On each tick, we increase an accumulator value by a frequency value deduced by a phase offset value. With this, we increase the logic analyzer timestamp (cf. Equation D.3) by a value close to its actual frequency while removing the phase error over time. This method of error correction corrects the error evenly spread over the course of one second. If we do not have a (remaining) phase error, the timestamp increases with the logic analyzer's frequency. Once the accumulator reaches the value corresponding to a second, we increase the second counter and reduce the accumulator by the respective value corresponding to one second. Lastly, we increase the tick counter.

This method of performing a tick at each sample has the advantage that it allows us to precisely adjust the clock of the logic analyzer. Moreover, the advantage of doing these adjustments in software is the granularity with which we can adjust phase and frequency. Instead of performing a correction every 10 ms (cf. NTP [71]) and having to do a rather large adjustment, adding 9 or 11 ms, we can have much smaller changes by adjusting the clock slightly for every data sample and thus approximately 8×10^6 times a second (if the logic analyzer runs at 8 MHz).

4.7 Post-processing

Each of the testbed's nodes independently collects traces and timestamps these. Yet, as the system is intended to run distributed on several devices, we need to aggregate and process the traces, eventually. We perform this aggregation centrally on the testbeds central server. Therefore, the observers transmit their traces and serial logs to the central server using their network connections. As all timestamps are based on the same global clock, we can just merge all timestamped traces into a common trace. When using one or multiple GPS-disciplined synchronization nodes, the timestamps in this common trace follow a global reference time (UTC). If we want to have the same global duration of a second, when using the single, non-GPS synchronization node, we can trace a GPS 1-PPS (1 pulse per second) signal at one of the testbed nodes. With this signal, we can stretch or unstretch the recorded time between two GPS time signals for all traces, matching our traces to a more accurate timescale (UTC). As long as testbed server has enough storage, we can scale this post-processing and thus our system to infinitely many testbed nodes while linearly increaing the post-processing time.

4.8 Implementation

After presenting the design of Grace, we discuss its implementation. We also discuss its integration into our existing testbed [97], to illustrate how Grace can be retrofitted and integrated into existing testbeds.

Synchronization node: For implementing the synchronization node, we use an STM32F401 microcontroller with 86 MHz clock speed and a CC1101

433 MHz radio [286]. In case of the GPS-disciplined synchronization node, we additionally use the MTK3339 GPS chipset [287] and an external GPS antenna with a gain of 30 ± 3 dBi. We send a timestamp once a second. Either triggered by the GPS 1-PPS signal, or by a local timer. The timestamp we send once a second is a 4 byte value and the sole payload of the packet. This payload size is sufficient to send Unix timestamps until the beginning of the next century.

CC1101 Packet Format: The total structure of the packet we send after the preamble consists of 2 bytes sync word, one byte each for packet length and address, 4 bytes payload (timestamp) and a 2-byte checksum [286]. As we use an unsigned integer as our payload, it is large enough for any Unix timestamp in this century.

Testbed node: As an observer, we use a Raspberry Pi 3B+ in our testbed due to its low price and wide availability at the time of building the testbed. For the GPIO tracing, we use an eight-channel logic analyzer featuring a Cypress EZ-USB FX2LP microcontroller [288]. The FX2 consists of an 8051 microcontroller, a USB interface, and i.a. the General Programmable Interface (GPIF). The GPIF allows specifying custom communication protocols via a finite state-machine. It is used to constantly sample data from the Logic Analyzer's input pins into the USB buffer. All of these components work independently of each other, allowing a deterministic tracing operation without being interrupted by the microcontroller or the USB interface. We build a custom firmware for the FX2, enabling us to focus on tracing sparsely occurring events continuously for the full duration of an experiment. The logic analyzer's state machine samples the state of its (eight) inputs at a frequency of 8 MHz (one sample every 125 ns). Internally, it passes these samples to the USB interface and writes them to the USB buffer. To one pin of the logic analyzer, we connect a radio, the same one mentioned above (CC1101 433 MHz radio). To notify the logic analyzer of a successfully received packet, the CC1101 asserts the successful reception of a reference signal on one of its GPIO pins [286]. We implement the described trace collection and time error correction functionalities and algorithms as part of an application running in user space on the observer (Raspberry Pi). For having a high granularity for the timestamps, we choose a value of (2^{60}) as a value for the constant `TICKS_PER_SECOND` in our algorithms. For ease of wiring the logic analyzer and the radio to the Raspberry Pi, and positioning the radio, we design a (non-essential) custom PCB HAT for the Raspberry Pi (see Figure D.1b). This HAT consists of a 2-layer PCB, headers, and a ribbon cable to connect the logic analyzer.

Post-processing: We implement the post-processing to aggregate the GPIO traces into a common human-readable CSV file. Moreover, we also convert it into a VCD file to be able to easily, visually analyze the combined traces with a software like GTKWave.

Cost: As we argue that Grace is a low-cost GPIO tracing system, we present in Table D.1 the cost for the different components, at the time of writing. The table shows, that the hardware for a testbed node stays below €20. The cost of a synchronization node depends on the presence or absence of a GPS module and cost either €19 or €56 without or with GPS, respectively. Equipping a full testbed of 20 nodes with this GPIO tracing system and a single time source costs less than €450.

Table D.1: Cost of components for Grace.

Component	Cost (€)
CC1101 Radio Module	9
STM32F401 Development Board	9
8-Channel Logic Analyzer	8
GPS module	26
GPS Antenna	9
Antenna adapter cable	2
PCB	0.50
PCB headers and cables	<2
Jumper Wires	<1
Total: Synchronization Node	19
Total: GPS-disciplined synchronization node	56
Total: Testbed Node	18
Total: Testbed node with PCB	19.50

4.9 Discussion

After introducing the design and its implementation, we additionally discuss the system's scalability and limitations, as well as its applicability for other applications.

Generally, we can scale our system to infinitely many testbed nodes. The only requirement is that every testbed node is in single-hop distance from one synchronization node. Thus, on the side of the testbed nodes, the limit is the availability of single-hop time signals, which we can easily extend by using more synchronization nodes. The only real limitation is the testbed server that has to handle the post-processing. The more nodes we have in the testbed, the more storage it needs to store all the traces and the more time the post-processing takes to merge these traces. Yet, as the server is not constraint to a specific hardware, this should not be an issue in a deployment. Moreover, as our system solely concerns the observer of a testbed node, the behavior of the IoT devices like their uptime and availability is of no concern for this system. The only thing a platform like the IoT nodes we use has to have is the capability to toggle traceable GPIO pins. Thus, we can use the system easily in combination with other hardware.

5 Evaluation

In this section, we experimentally evaluate our time-synchronized GPIO tracing system Grace. We evaluate the GPIO tracing intrusiveness, Grace's degree of time-synchronization, and its stability within an actual testbed deployment. We set our system into perspective to other time synchronization approaches, discussing its advantages and disadvantages. We first evaluate our system when using a single time source. For this, we evaluate the different sub-systems individually before finally looking at the system as a whole. Afterward, we look at the extension of using more than one synchronization node and look at its influence on the system's degree of time synchronization.

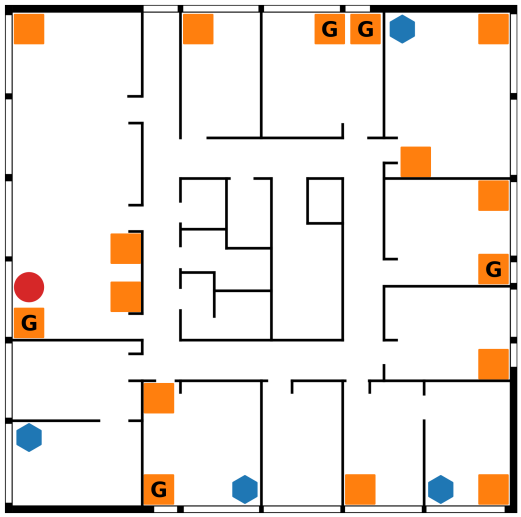


Figure D.2: Local testbed of 500 m^2. Red circle: synchronization node; Orange squares: Nodes equipped with our GPIO tracing system (Grace); Blue hexagons: other nodes; Marker G: nodes equipped with GPS.

5.1 Evaluation Setup

Before our evaluation, we discuss our evaluation environment, namely our testbed, with the location of the different node types, the metrics we evaluate, and the clock references we use for accurately evaluating Grace's timing.

5.1.1 Testbed

We evaluate Grace on our local testbed of 20 nodes (see Figure D.2), spanning the top-most floor of a university building with an area of 500 m^2. The floor was mostly unoccupied during the experiments, yet, as we use 433 MHz for communication, we expect occupation to have minimal impact on the evaluation results.

In our initial deployment, we equip 16 of the 20 nodes, with a logic analyzer and a radio (marked with orange squares in Figure D.2). Additionally, we place the synchronization node in close vicinity of one of the testbed nodes (marked with a red circle in Figure D.2). In the latest state of testbed, all nodes are part of our GPIO tracing system Grace.

For our additional evaluation, extending the original version of Grace of the multiple synchronization nodes scenario, we place three synchronization nodes next to each other beside the node marked with 'G' on the right side of the testbed map. Due to a required close proximity of testbed nodes to the same logic analyzer for evaluating their time offsets, we place three nodes close to the location of the synchronization node (red circle) we use for the other experiments.

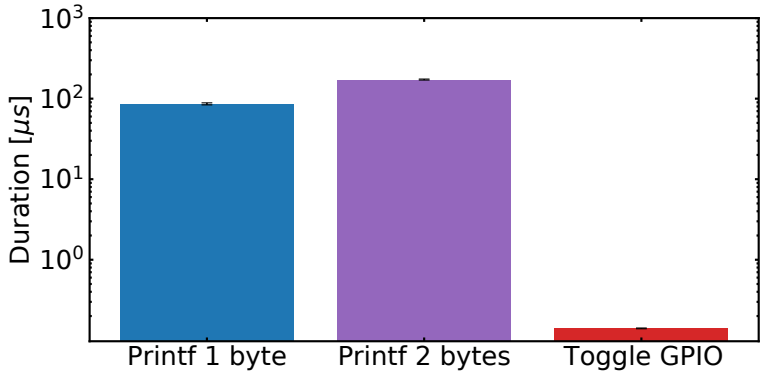


Figure D.3: Comparison of the duration of different logging outputs. We compare the duration of printing 1 byte/character *(avg: 86.2 µs)*, or 2 bytes *(avg: 172.6 µs)* (1 character and a newline character) through the serial interface, with the duration of the state change of 1 GPIO pin *(avg: 140.5 ns)*. Please note that we use a logarithmic scale to display the drastic duration differences between the output through GPIO and through the serial interface.

5.1.2 Metrics

We evaluate Grace in terms of the logic analyzer's frequency stability, the synchronization node's frequency stability, timing offset between testbed nodes, timing offset between time sources, and system-wide time-synchronization stability. We measure frequency deviations and time offsets.

5.1.3 Reference Clock

To measure and evaluate the exact timing of events in Grace, we require systems with a more accurate clock than the system's clock we evaluate. Therefore, we either use an external logic analyzer (Saleae Logic Pro 8 [289]) or the 1-PPS signal output of GPS receivers as a reference clock. The error of our reference clock is up to 50 $\mu s/s$ (50 ppm) for the Saleae logic analyzer [290] and up to tens of nanoseconds between two GPS receivers. For the experiments on synchronization node stability (Section 5.4), receiver stability (Section 5.5), and multiple time sources (Section 5.7) we use the logic analyzer, as we interface at least two nodes at once. For the remaining experiments, we use the GPS receivers. The GPS receivers (marked *'G'* in Figure D.2) as well as the ones we use for the evaluation of multiple time sources (Section 5.7) have an external antenna mounted on the outside of the building, to keep the 1-PPS synchronization at the specified accuracy.

5.2 Output intrusiveness

To explain the advantage of GPIO tracing for time stamping, we evaluate the intrusiveness of it in comparison with the standard output solution of serial logging using `print`-statements. We evaluate it using the nRF52840-DK [291], one of the target platforms present in our testbed, running Contiki-NG [256]. To

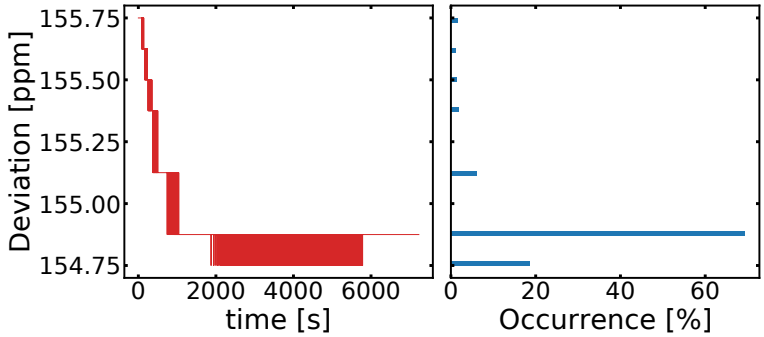


Figure D.4: Stability of a deployed logic analyzer over time. On the Y-axis, we display the relative deviation of the logic analyzer from its nominal frequency of 8 MHz in ppm.

access the GPIO pins of the nRF52840 [192], we use the nRF52840's hardware abstraction layer (HAL) included in Contiki-NG. In Figure D.3 we compare the duration of logging one character with and without inserting a newline through the serial interface at a baud rate of 115200 baud/s and the duration of a state change of a GPIO pin.

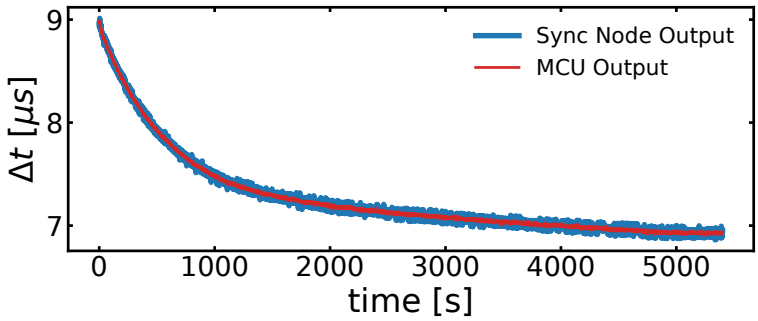
The average duration for printing one byte, e.g., 1 character, is 86.2 µs, and for printing two bytes, e.g., 1 character and a newline character for better readability, is 172.6 µs. The duration for changing the state of a GPIO pin is significantly smaller at 140.5 ns, equaling 9 clock cycles. Thus, even a pattern of up to 613 GPIO changes is still faster than printing a single character through the serial interface. Therefore, we can conclude that GPIO tracing is a low-intrusive logging method.

5.3 Logic Analyzer Frequency Stability

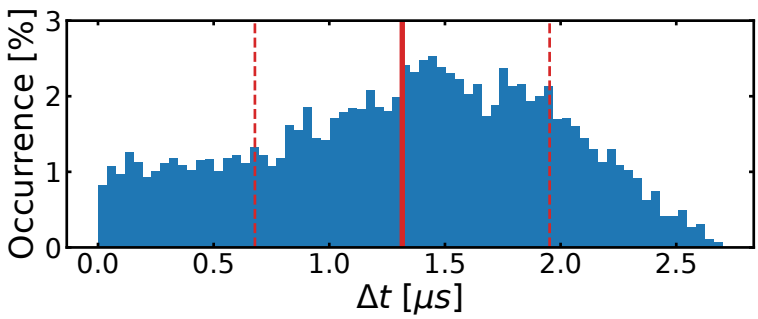
We continue our evaluation by analyzing the frequency stability of a logic analyzer. We, therefore, trace the 1-PPS signal of a GPS receiver with a logic analyzer. In Figure D.4, we show the frequency stability over two hours and the relative occurrence of the different deviations, exemplary for one logic analyzer. This logic analyzer had during the tracing of the GPS signal an average deviation from the nominal frequency of 154.9 ppm. Generally, we expect the frequency stability of the logic analyzers to be within ±200 ppm [290]. As 200 ppm equals a time difference of 200 µs within a second, and thus, a maximum time difference of 400 µs between two logic analyzers, this clearly underlines the need for a system that time-synchronizes logic analyzers.

5.4 Frequency Stability Of Synchronization Node

Next, we investigate the frequency stability of the synchronization node. We configure the synchronization node to send a packet once a second according to its internal clock. With an external logic analyzer, we record the exact sending times over a period of 90 minutes. For that, we record the completion of creating a timestamp at the microcontroller by tracing the chip select line.



(a) Stability of the synchronization node's signal over 90 minutes. The radio output of the synchronization node follows the microcontroller with a slight jitter of on average 18.7 ns.



(b) Offset distribution of a radio's input signal from the radio output signal of the synchronization node. Avg offset (solid red line): 1.32 µs, Std (dashed red lines): 637 ns.

Figure D.5: Stability of the microcontroller output, the synchronization node's radio output, and the testbed node's radio input.

Moreover, we trace the state of the GPIO pin the radio turns off once it is done sending a packet.

Figure D.5a shows the offset of the microcontroller's second from the reference second over the time of the experiment. This offsets starts at 9 µs and over time reduces to 7 µs. Figure D.5a also shows the resulting offset of the reference time signal from the reference time. Generally, the transmission times of the radio precisely follow the microcontroller's signal generation times/offset from the reference time with a slight jitter of on average 18.7 *ns* and a maximum of 78 *ns*. While the offset between two timestamps is on average 7.1 µs, it is of no concern as the offset will be evenly present on all testbed nodes and thus have at most a minor influence on the distributed time-synchronization. The added jitter in the nanosecond range is no concern for our system's requirements.

After investigating the synchronization node's stability, we next look at the stability of the testbed nodes. For this, we firstly look at the deviation of the input signal at one testbed node from the output signal of the synchronization node. For that, we interface both nodes with our external logic analyzer

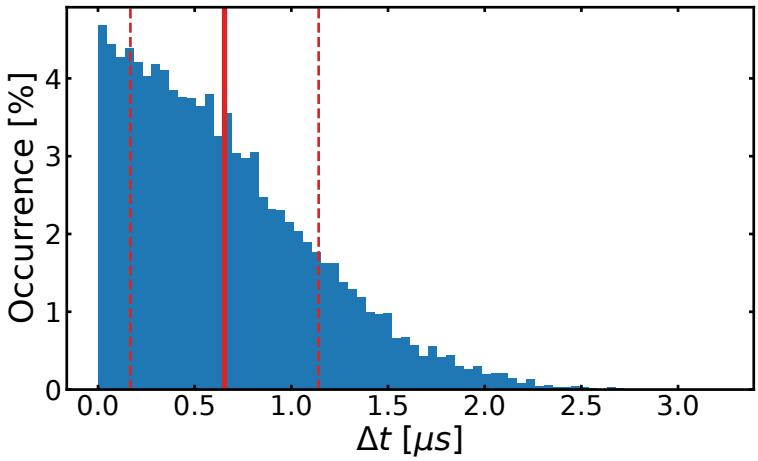


Figure D.6: Histogram showing the distribution of offsets between two radio receivers. We show the mean value (0.65 µs) as a solid red line, and the standard deviation (487 ns) as dashed red lines.

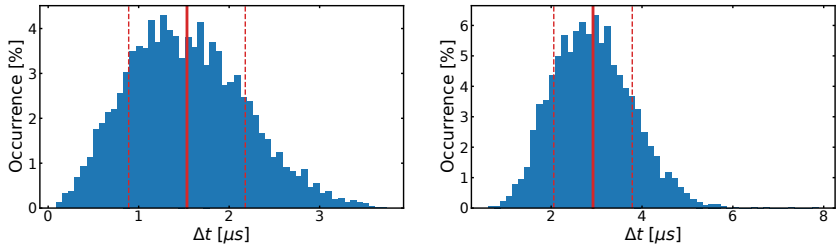
simultaneously. On the synchronization node, we trace the GPIO pin, the radio turns off once it is done sending a packet, and on the testbed node, we trace the GPIO pin of the radio that also notifies our time-synchronization system of the availability of a new timestamp. We initially synchronize these two timestamps, to analyze the receiver's variation in offset. Figure D.5b shows an average offset of 1.32 µs with a standard deviation of 637 ns from the synchronization node's time signal.

5.5 Receiver Stability

Next, we look at two nodes in the testbed, close to each other and the signal received by the radios. We once again use the logic analyzer and trace the GPIO pin of the radio that also notifies our time-synchronization system of the availability of a new timestamp. We run several experiments with a total duration of almost 5 hours. When looking at the reception time differences between the two radios, we see the distribution shown in Figure D.6. The difference between the radio's reception times (without the time correction system) is on average 654 ns (median: 562 ns) with a standard deviation of 487 ns. The maximal measured offset between the two radios is 3.22 µs. 77.4% of the measurements have an offset of less than 1 µs. Even the maximum value of 3.22 µs is sufficient for evaluating the timing of many IoT protocols, including Time-Slotted Channel Hopping (TSCH) [6].

5.6 Clock Correction

Next, we focus on the full system, including the time error correction. To evaluate this, we include the nodes that have a GPS receiver, and we use the 1-PPS GPS signal traced by the testbed node's logic analyzers. We analyze the time differences of the timestamps associated with the 1-PPS GPS signals.



(a) Offsets between any two traced GPS signals. Avg: 1.53 µs, Std: 644 ns.

(b) Offsets between any two GPIO tracing nodes within our testbed. Avg: 2.92 µs, Std: 864 ns.

Figure D.7: Histograms showing the distribution of offsets between multiple nodes using the full time-error correction system. We show the mean value as a solid red line, and the standard deviation as dashed red lines.

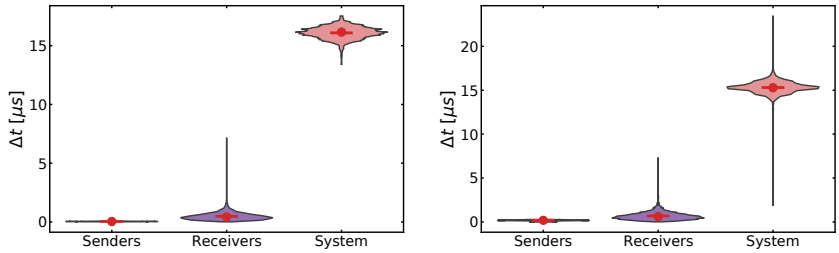
Figure D.7a shows the distribution of the time stamping error of Grace. On average, the system has an error of 1.53 µs with a standard deviation of 644 ns, and a maximum error of 3.75 µs. This clearly shows the advantage of Grace over NTP with a thousandfold higher precision. Moreover, this synchronization is sufficient for analyzing timing in many IoT protocols, including time-critical communication protocols like Time-Slotted Channel Hopping (TSCH) [6] and Chaos [59].

After comparing the system's stability with a GPS reference, we can also compare it to the reference signal our synchronization node sends out. Therefore, we compare the deviation of the local timestamps based on the synchronization signal. Figure D.7b shows similar results to the GPS-based experiment. However, when tracing the synchronization node's time signal, all the offsets between the different receivers get accumulated (cf. Figure D.6). Overall, the offset, when including all these errors, between any two nodes is on average 2.92 µs with a maximum offset of 7.9 µs.

5.7 Multiple Time Sources

After evaluating our system using a single time source, we will next evaluate the performance of our extended system using multiple synchronization nodes. This is necessary when we need to span larger buildings or want to time-synchronize campus-wide networks. For the synchronization nodes (senders), and the receivers, we trace the end of sending and the start of the reception with a logic analyzer, respectively. For the full system, we trace with the internal system the GPS 1-PPS signal of a GPS receiver, the same one for all involved testbed nodes.

Figure D.8a shows the offset distributions for synchronization nodes (senders), radio receivers, and the full time-synchronization system using a single GPS signal. We feed the same GPS signal from a single GPS receiver to all three of the senders. After passing the sender, the sent-out signals have on average an offset of 39 ns to each other with a standard deviation of 19.5 ns. The receivers have a significantly higher timing error of on average 480 ns (standard



(a) Three sender-receiver pairs. Time signal generated using one GPS receiver.

(b) Three sender-receiver pairs. Time signal generated using three GPS receivers.

Figure D.8: Time offsets of different components of Grace. Time offsets between the generated time signals at the sender side, between the received time signals at the receiver side, and between the nodes of the full system. Red line: average time offset, Red dot: median time offset

deviation: 352 ns) with some outliers reaching an offset of up to 7.2 µs. As in the case of one synchronization node, we see that the full system has a significantly higher timing offset of on average 16.1 µs and with a standard deviation of 530 ns.

When using multiple time sources at different locations, we need to use different GPS modules for them. Thus, next, we compare how the use of three different GPS receivers changes the behavior of our system. Figure D.8b shows the offset distributions for the same subsystems as above. Here, we see a similar trend as above with slightly higher timing errors between the senders and the receivers, while the full system is not affected much, excluding some outliers. The senders have an average timing offset of 173.5 ns to each other with a standard deviation of 63.7 ns and the receivers have a timing offset of 680.5 ns with a standard deviation of 453.5 ns. While these numbers are slightly higher than above, this is expected, as this setting also includes the time offset introduced by using multiple GPS receivers. The full systems achieves a time synchronization error of 15.3 µs with a standard deviation of 855.8 ns.

5.8 Summary

From the results, we present in this evaluation, we can conclude that Grace achieves a building-wide time-synchronization using a single time source in the range of a few microseconds. This does not fully reach the degree of time-synchronization offered by custom solutions which use specific hardware or FPGA's for GPIO tracing [86, 93]. However, our system is easily and cost efficiently retrofittable to existing testbeds and offers a sufficient time synchronization for tracing in many application fields. For testbeds larger than a single building, we have to make a trade-off between coverage and accuracy. With multiple synchronization nodes, our time-synchronization system has a ten times higher error of 15.3 µs in comparison with the system using a

single synchronization node. While this is significantly worse than the time-synchronization we achieve when using only a single time source, it still offers a significantly higher degree of clock synchronization than NTP and is still good enough for measuring and evaluating the time synchronization in wireless IoT communication protocols like Time-Slotted Channel Hopping (TSCH) [6]. Moreover, the time-synchronization between the nodes using the same time source is not affected by this decrease in synchronization. Thus, Grace offers low microsecond accuracy between nodes using the same time source and tens of microseconds accuracy between nodes using different time sources.

6 Conclusion

Testbeds are an important tool for developing and evaluating IoT protocols. While there are many testbeds used by the research community, most of them lack the capabilities to accurately evaluate the timing of time-critical IoT systems.

With Grace, we present an easily retrofittable system capable of tracing the timing of IoT devices by proposing a low-cost GPIO tracing system. Grace uses only low-cost off-the-shelf components, making it retrofittable to existing testbeds for less than €20 per node. We show that Grace can cover a building-wide testbed with a single time source and can synchronize GPIO events with an average time synchronization error of 1.53 µs, and a worst-case time synchronization error of 3.75 µs. Moreover, Grace can extend time synchronization to larger areas like campus-wide testbeds using multiple time sources. It achieves an average time synchronization error between nodes using different time sources of 15.3 µs.

E

TSCH meets BLE: Routed Mesh Communication over BLE

Laura Harms, Olaf Landsiedel

Proceedings of the 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT), 2023, pp. 187–195.

Abstract

Bluetooth Low Energy (BLE) is the prevalent communication protocol for the Internet of Things. However, for time-critical applications requiring time-synchronized multi-hop networks with often multiple node exchanging data at the same time slot, BLE lacks a solution. Instead, we commonly see IEEE 802.15.4 being used with its Time-Slotted Channel Hopping (TSCH) MAC layer.

In this work, we build TBLE, which brings the established TSCH protocol to BLE, enabling BLE to be used for time-synchronized routed mesh communication. We show that in experimental testbed deployments, TBLE achieves similar performance to TSCH, with the possibility for lower average latencies of up to 20%. Moreover, due to the higher spectral efficiency of BLE compared with IEEE 802.15.4 (40 vs. 16 channels), more parallel routed communications are possible with TBLE, further reducing latency and increasing throughput.

1 Introduction

With more than 5 billion ($5 * 10^9$) BLE devices estimated to be shipped in 2023 alone [266] and a continuing rise in popularity, Bluetooth Low Energy (BLE) is the prevalent standard for communication in low-power wireless networks. Due to its wide availability, low-cost and energy efficiency, BLE is supported by practically all smart devices we interact with today. In contrast, most wireless industrial devices and many smart home devices use IEEE 802.15.4 instead of BLE. Of the two protocols, the physical layer of BLE is the less complex one using a GFSK modulation scheme in contrast to the O-QPSK modulation scheme of IEEE 802.15.4, allowing for cheaper radios.

Both protocols have limited range and thus rely on, i.e., multi-hop mesh networking for communicating over longer distances. In the field of IEEE 802.15.4, there are several established protocols, including those for flooding-based communication [9, 65] as well as routing-based communication [6]. For routing-based communication, an established technique is Time-Slotted Channel Hopping (TSCH), the standard MAC layer protocol in IEEE 802.15.4. For BLE, the standard mesh protocol is Bluetooth Mesh [25,26], a protocol using managed flooding on top of BLE advertisements. While flooding-based protocols usually use the entire network for sending a message, routing-based protocols allow multiple parallel communications in the same network at the same point in time. Yet, to our knowledge, there is no time-synchronized routing-based mesh communication protocol for BLE.

Several works looked at the combination of IEEE 802.15.4 TSCH and BLE, either in terms of coexistence [190, 191] or by using a single radio for both BLE and IEEE 802.15.4 and communicating TSCH control information using concurrent BLE transmissions [193]. Especially the latter work raises the question of why there is the need to continue using IEEE 802.15.4 for TSCH communication while only communicating control information over BLE.

In this paper, we combine the BLE PHY with the MAC layer protocol TSCH and call this combination TBLE. TBLE sends standard TSCH packets as part of time-synchronized BLE advertisements, enabling routed mesh communication over BLE with TSCH and replacing IEEE 802.15.4. TBLE enables the use of well established protocols including deadline-based real-time communication protocols on top of BLE. We design TBLE for the use with any BLE radio. We exemplarily implement a BLE driver for the nRF52840 DK [291] for Contiki-NG [256] and adapt it to allow the transmission of valid IEEE 802.15.4 TSCH frames within BLE packets. We study both the coded (125 kbps/500 kbps) and the uncoded (1 Mbps/2 Mbps) PHYs of BLE and experimentally evaluate TBLE's performance on a low-power wireless testbed using the well established autonomous scheduler Orchestra [8] which is included in Contiki-NG and compare its performance to TSCH over IEEE 802.15.4.

Our evaluation on a testbed shows, that especially the coded BLE modes achieve a similar connectivity within a deployment as IEEE 802.15.4. TBLE achieves similar performance to TSCH, with the possibility for lower average latencies of up to 20%. Moreover, due to the higher spectral efficiency of BLE compared with IEEE 802.15.4 (40 vs. 16 channels), more parallel routed communications are possible with TBLE, further reducing latency and increasing throughput.

Overall, we make the following contributions:

- We present TBLE, a protocol closing the gap of routed mesh-communication in BLE. TBLE extends the established TSCH standard.
- We design and implement a BLE driver for the Nordic nRF52840 DK for Contiki-NG and adjust it to be compatible with the Contiki-NG IEEE 802.15.4 TSCH and 6TiSCH stack. We make it publicly available[1] as open source.
- We are the first to run TSCH over BLE, demonstrating TBLE as a practical routed mesh-protocol for BLE.
- We experimentally evaluate TBLE and compare it to IEEE 802.15.4 TSCH, showing its feasibility and a possible performance increase over TSCH without modifying any upper-layer protocols.

We structure the remainder of this paper as follows. Section 2 gives the necessary background information on IEEE 802.15.4, BLE and TSCH, followed by a detailed dissection of TSCH in Section 3. In Section 4 we introduce the design of TBLE, followed by our experimental evaluation of TBLE and its comparison to IEEE 802.15.4 TSCH in Section 5. In Section 6 we discuss a selection of works related to this topic, and conclude our work in Section 7.

2 Background

In this section, we introduce the required background information on IEEE 802.15.4, Time-Slotted Channel Hopping (TSCH), and Bluetooth Low Energy (BLE).

2.1 IEEE 802.15.4

IEEE 802.15.4 is a low-power personal area radio protocol introduced in 2003 [27], initially for the 2.4 GHz (2400 – 2483.5 MHz) ISM band. It operates at a data rate of 250 kbps and uses a robust modulation scheme of O-QPSK with DSSS (direct-sequence spread spectrum). IEEE 802.15.4 specifies 16 channels that are 2 MHz wide and 5 MHz apart. On the physical layer, an IEEE 802.15.4 packet consists of 4 preamble bytes, a 1 byte start of frame delimiter (SFD), a 1 byte packet length, and up to 127 bytes of payload.

Besides the physical layer, the IEEE 802.15.4 standard also defines the medium access control (MAC) layer. One defined MAC layer is Time-Slotted Channel Hopping (TSCH).

2.2 Time-Slotted Channel Hopping (TSCH)

Time-Slotted Channel Hopping (TSCH) [6] is a standardized MAC layer protocol (IEEE 802.15.4e) for low-power wireless mesh networks. TSCH combines Time-division multiple access (TDMA) with Frequency-division multiple access (FDMA) and a pseudo-random channel hopping mechanism. Communication

[1] Available as open-source at: `https://github.com/ds-kiel/TBLE`.

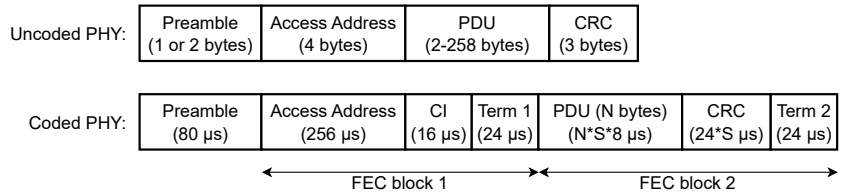


Figure E.1: BLE PHY packet formats

occurs in distinct time-frequency-slots, with as many concurrent communications as channels included in the hopping sequence (maximally 16). The channel hopping allows TSCH to withstand narrowband interference.

Slots in TSCH have a standard length of 10 ms and allow the transmission of a single IEEE 802.15.4 packet followed by an optional acknowledgement upon successful reception. Slots can be reserved for sending and receiving Enhanced Beacons (EB). Beacons are sent regularly containing control information for nodes to associate to the TSCH network and to keep the network in sync and alive.

2.3 Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE) [7] is a short-range and low-power communication protocol in the 2.4 GHz ISM band targeting single-hop communication between two devices. BLE uses 40 2-MHz wide frequency channels, which use Gaussian Frequency Shift Keying (GFSK) as a modulation scheme. Three of these channels are reserved for (primary) advertisements and broadcasts, while the other 37 are reserved for connected communication and secondary advertisements. BLE offers an uncoded PHY with data rates of 1 Mbps (standard data rate) and 2 Mbps, and since Bluetooth 5.0 even a long-range coded PHY with data rates of 125 kbps and 500 kbps.

PHY packet format. The physical layer packet format of BLE differs between the uncoded PHY and the coded PHY. The uncoded PHY packet starts with a 1 or 2 byte preamble of alternating ones and zeros, for a data rate of 1 Mbps and 2 Mbps, respectively. It is followed by the 4-byte access address, identifying packets belonging to a connection. For advertisement packets, the advertisement address is fixed to `0x8E89BED6`. Afterward, the packet contains between 2 and 258 bytes payload (PDU) and a 3-byte cyclic redundancy check (CRC) code for error correction. The coded PHY packet generally contains the same components, however, with additional fields for error correction (see Figure E.1). The preamble is uncoded, consisting of 10 repetitions of `0x3C` transmitted with a data rate of 1 Mbps. The first forward error correction (FEC) block is always transmitted with a data rate of 125 kbps containing the access address and the coding indicator (CI). The CI indicates the coding of the second FEC block, deciding whether it uses a data rate of 125 kbps or 500 kbps. The PDU and the CRC are then transmitted with the indicated coding afterward.

Advertisements. BLE has two operation modes: connected and non-connected. In the non-connected mode, BLE devices disclose their presence and advertise their services to nearby devices. These services include, i.e.,

Table E.1: IEEE 802.15.4e TSCH timeslot timings.

Name	Time offset / duration (μs)
CCAOffset	1800
CCA	128
TxOffset	2120
MaxTx	4256
RxOffset	1020
RxWait	2200
RxAckDelay	800
TxAckDelay	1000
AckWait	400
MaxAck	2400
Sum	9776
Timeslot Length	10000

media control services or weather information (e.g., temperature data). For some of these services, a receiver has to connect to the advertising device and communicate in connected mode.

Advertisement data. In non-connected mode, an advertiser sends data (PDU) consisting of a 2-byte header, followed by one or multiple advertising data (AD) structures. The first byte of the header contains a 4-bit PDU type, 1 bit reserved for future use, a 1-bit flag whether the advertiser supports the LE channel selection algorithm, two 1-bit flags whether the advertiser's and the target device's addresses are random or public, respectively. The second header byte contains the length of the subsequent advertising payload. An AD structure consists of a 1-byte length, n bytes AD type (e.g., an identifier that a list of service UUIDs follows) and $lenght - n$ bytes AD data (e.g., the list of service UUIDs).

3 Dissecting TSCH

After a general introduction of Time-Slotted Channel Hopping (TSCH) in the background, we analyze the inner workings of TSCH. For that, we study the timings within a TSCH timeslot and the time synchronization mechanism of TSCH, especially in the context of the implementation of TSCH in Contiki-NG.

3.1 TSCH Timeslot Timing

A TSCH timeslot allows the transmission of one packet with a subsequent acknowledgement in case the packet was received. We illustrate the timing within a timeslot for both the sender and receiver in Figure E.2, and provide the timing offsets and durations in Table E.1.

At the sender's end, a timeslot starts with the transmission offset (`TXOffset`). During this offset, the sender configures its radio, turns it on, and potentially performs Clear Channel Assessment (CCA). Moreover, the sender prepares the packet (i.e., adding headers) and starts transmitting. By the end of the `TXOffset`, the preamble and the Start of Frame Delimiter (SFD) should be

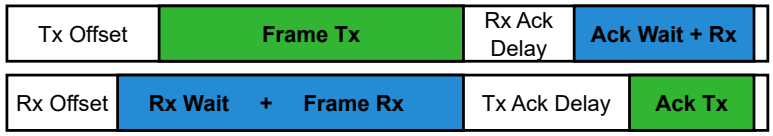


Figure E.2: Simplified TSCH timeslot timing. We omit the optional `CCAOffset` and the `CCA`, which happen during `TxOffset`, if enabled. Please note: the illustrated timing is not to scale. For the correct timing, see Table E.1.

transmitted. Following the SFD, the radio transmits the TSCH frame standardized to a maximum length of 128 bytes including 1 length byte. TSCH reserves a time of 4256 μs equating to 133 bytes for this. After the transmission, the sender turns off its radio, and in case of an expected acknowledgement (ACK), it reconfigures the radio into receive mode and turns it on by the end of the `RxAckDelay`. If the sender, which is waiting for the ACK does not receive anything by the end of `AckWait`, it turns off its radio. Otherwise, if it detects an ACK, it receives it and turns off the radio afterward.

The receiver starts with an `RxOffset` during which it configures the radio and turns it on to listen for incoming packets. If it does not start receiving a packet within `RxWait` it turns off its radio. If the receiver receives a valid packet that requires acknowledgement, it prepares its radio for transmission during `TxAckDelay` and by the end of `TxAckDelay` it should have transmitted the packet's synchronization header. The full acknowledgment of maximally 69 bytes (plus 1 length byte) has to be transmitted within the time given by `MaxAck` of 2400 μs (the duration of 75 bytes).

For both the sender and the receiver, we see a mismatch of a time equaling 5 bytes between the reserved transmission times and the maximum packet lengths that can be transmitted. We assume that the given times include the times for transmitting the synchronization header, even though the synchronization header should have been transmitted already during `TxOffset` or `TxAckDelay` for frame and ACK, respectively.

3.2 TSCH packet duration

The TSCH timeslot timing provides us with the maximum time a transmission might take (`MaxTx`). However, during operation, we do not wait until the end of `MaxTx` before continuing with the next field of the timeslot if we send packets shorter than the maximum length. Instead, we directly continue once the transmission is over. Instead of probing the radio for the end of the transmission, the implementation of TSCH in Contiki-NG computes the transmission time of the packet. The implementation defines the TSCH packet duration as $duration_{packet} \leftarrow airtime_{byte} * (len + overhead_{PHY})$, with a PHY overhead of 3 for the nRF52840.

3.3 TSCH time synchronization

On the physical level, a radio can precisely timestamp certain events related to the radio packet, which we can use as markers for time synchronization. For

example, in IEEE 802.15.4 mode, the nRF52840 chipset can timestamp events including *framestart*, *Address sent or received*, *Packet payload sent or received*, and *Packet sent or received*. The TSCH implementation in Contiki-NG uses the *start of frame delimiter (SFD)* as the timestamp to synchronize on. While the radio cannot timestamp the start of the SFD, it can timestamp the *framestart*, which is the timestamp right after transmitting or receiving the SFD. Thus, Contiki-NG can derive the necessary timestamp easily from the *framestart* event timestamp. As the SFD is 1 byte long (and takes 32 μs), we can compute the SFD timestamp by subtracting 32 μs from the recorded timestamp.

3.4 Hopping sequences

For counteracting narrowband interference, TSCH uses channel hopping according to a pseudo-random hopping sequence. TSCH defines a 9-bit linear feedback shift register to determine these hopping sequences. Common hopping sequences include a single-channel hopping sequence (channel 20), a four-channel hopping sequence (channels 15, 20, 25, and 26), and a 16-channel hopping (all IEEE 802.15.4 channels).

4 Design

In this section, we discuss our design enabling TSCH on top of BLE advertisements, which closes the gap of routed mesh communication in BLE. We discuss the TSCH timeslot timings regarding the four different BLE data rates and show how we achieve time synchronization. Moreover, we discuss our BLE packet format and the channel hopping sequences for TBLE.

4.1 Overview

The general idea behind TBLE is to replace the IEEE 802.15.4 PHY with a BLE PHY and send standard TSCH packets as part of BLE advertisements. For other BLE devices, these appear like standard BLE packets, while devices running TBLE can recognize them and form a standard TSCH network using a BLE PHY instead of the IEEE 802.15.4 PHY for communication. For that, we have to change the TSCH timing to work with different data rates and embed the TSCH payload into BLE packets. Moreover, BLE offers significantly more radio channels in the same spectrum (40 instead of 16), due to a lower channel spacing, and thus, we can use different and longer hopping sequences allowing both more possibilities to avoid interference and a higher total bandwidth with more parallel communications. Lastly, a BLE radio does not offer the same timestamping capabilities as an IEEE 802.15.4 radio; thus we have to identify a different timestamp for time synchronization.

4.2 Derived Timing

In Section 3.1, we explore the timing of the standard 10 ms TSCH slot (cf. Figure E.2). The different offsets and wait times in the standard TSCH slot are partly dependent on the radio's data rate. The ones dependent on the radio's data rate are the maximum frame length (`MaxTX`), the maximum ACK

length (`MaxAck`), the `TxOffset`, and the `AckDelay`. The former two values depend on a packet's maximum time on air, thus on the maximum number of bytes and the radio's data rate. The latter two depend only to a minor extent on the radio as they contain mainly processing and wait times and, in addition, the transmission time of the PHY synchronization header. The `CCA` duration might be dependent on the radio, yet it does not take more time than standardized on the radio we tested it on, which supports both IEEE 802.15.4 and BLE. All other times seem to be independent of the radio. Instead, some of them are dependent on the device's CPU speed. As we only change the physical layer (from IEEE 802.15.4 to BLE) and otherwise keep the same processor capabilities, we only change the times strongly affected by the physical data rate: `MaxTX` and `MaxAck`. As the PHY synchronization header takes less time for either BLE mode than for IEEE 802.15.4, we keep `TxOffset` and `AckDelay` unchanged, which increases the times for data processing by 152 μs and 80 μs for the uncoded and coded BLE modes, respectively. We also keep the guard times for correctly receiving the beginning of a data packet (`RxOffset` + `RxWait`) the same as in IEEE 802.15.4. Those guard times do not have any effect on the total slot length for IEEE 802.15.4 and do not exceed the derived timeslot lengths for any of the BLE modes.

Contrary to our expectations, our experiments show, that the guard time for beginning to receive an acknowledgment (`AckWait`) is too low to successfully receive an acknowledgment in coded BLE. Thus, we increase `AckWait` from 400 to 1000 μs for coded BLE.

From our dissection of TSCH (cf. Section 3.1), we know that we need to reserve the time equivalent to 133 bytes and 75 bytes for `MaxTX` and `MaxAck`, respectively. While BLE would allow for packets with a longer payload, we stick to the maximum payload size of IEEE 802.15.4, as this does not require major changes in TSCH. Moreover, packets with a longer time on air are more susceptible to short bursts of interference.

4.2.1 Packet Format

After setting the basis for the payload length of a packet and of an acknowledgment, we next discuss the BLE packet structure to be able to calculate the timing. This packet structure differs significantly between the coded and uncoded modes of BLE. While we want to send valid BLE packets in any case, we also want to enable a radio of a device that is not running TBLE to discard the packet as quickly as possible.

For the uncoded modes of BLE (1 Mbps and 2 Mbps), we choose a custom, application-specific access address and transmit the TSCH packet as is as the BLE advertisement packet's PDU. This should enable other BLE devices to discard the packet, as it is not using the standard advertisement access address. Moreover, it is very unlikely that a device has a connection with the same access address, we use for TBLE.

For the coded modes (125 kbps and 250 kbps), we cannot use a custom access address, as the radio we use is not able to receive coded BLE packets with an access address besides the standard advertisement access address. Instead, we have to choose a different way to create a BLE compliant but easily discardable packet. Thus, we create a standard advertisement packet with a

Table E.2: TSCH/TBLE timeslot timings and effective data rates.

Mode	IEEE 802.15.4	BLE 125k	BLE 500k	BLE 1M	BLE 2M
MaxTX (μs)	4256	9532	2590	1088	548
MaxAck (μs)	2400	5520	1662	624	316
AckWait (μs)	400	1000	1000	400	400
Sum (μs)	9776	7372	18172	4832	3984
Timeslot (μs)	10000	7500	18500	5000	4000
Effective data rate (kbps)	101.6	135.5	54.9	203.2	254

PDU with one advertisement data (AD) structure containing the TSCH frame. We set the first byte of the advertisement header to `0x90`, using an undefined PDU type, which might enable other radios to discard the packet right away. The AD structure we send uses the AD type `0xFF`, identifying the subsequent data as manufacturer specific.

Thus, for both modes, we deviate slightly from sending correct BLE packets to enable only devices running TBLE to further process the received data.

4.2.2 TBLE Timing

After the considerations regarding the BLE packet structure and the payload length, we can derive the timing for `MaxTX` and `MaxAck`, which we show in Table E.2. Similarly, to IEEE 802.15.4, we round up the timeslot lengths to the next multiple of 500 μs. The timeslot lengths vary between 4 and 18.5 ms. In addition to the timeslot lengths, we also show the effective data rates, which we calculate for a packet of maximum size (127 data bytes) in kbps with $\frac{127*8}{TimeslotLength}$.

4.3 Packet duration

To enable TSCH to start the `RxAckDelay` and `TxAckDelay` at the correct time, we need to adjust the TSCH packet duration computation (cf. Section 3.2). Setting the byte air time for the payload is straight-forward and can be directly derived from the bitrate. However, the radio PHY overhead is only easily identifiable for the uncoded mode (10 bytes). For the coded modes of BLE, we identify 4 bytes overhead in the payload plus certain fixed time overheads. These are 296μs for FEC block 1, and 54μs or 216μs for CRC and Term 2 at the end of FEC block 2 for a BLE data rate of 500 kbps or 125 kbps, respectively (cf. Figure E.1).

4.4 Time Synchronization

As we discuss in Section 3.3, TSCH uses time stamping functionalities of the radio, for precise time synchronization, especially the timestamp of the start of frame delimiter (SFD). As BLE PHY packets do not contain an SFD between the preamble and the start of the frame, we cannot use the same timestamp for time synchronization as in the case of IEEE 802.15.4. The time stamping

points the nRF52840's radio offers in BLE mode are *address sent/received*, *payload sent/received*, and *packet sent/received*. When comparing the accuracy of the different time stamps between two devices located next to each other and connected to a logic analyzer, our results strongly suggest that the timestamp of the *address sent/received* event is most suitable for TSCH time synchronization. Similar to IEEE 802.15.4, we use this timestamp to compute the timestamp of the end of the preamble. For that, we subtract the transmission time for the address from the timestamp. Moreover, our experiments show, that contrary to our assumptions, the radio is not done transmitting its preamble by the end of `TxOffset`. Thus, to synchronize to the intended time, we introduce an additional (negative) timing offset initiating the transmission in TSCH earlier to ensure the preamble being transmitted exactly at the assumed time.

4.5 Hopping Sequences

For TBLE, we need hopping sequences using the BLE channels. For comparison with IEEE 802.15.4, we replicate the 1, 2, 4, and 16 channel hopping sequences using the BLE channels that best match the used IEEE 802.15.4 channels. In addition, we generate a 40 channel hopping sequence containing all BLE channels and an intermediate one using 32 channels. For the 40 channel hopping sequence, we use the 9-bit linear feedback shift register defined by the TSCH standard. For the 32 channel hopping sequence, we take the 40 channel hopping sequence and remove every fifth channel.

4.6 Standard-compliance Discussion

To end the description of our design, we want to discuss the standard compliance of TBLE with both TSCH and BLE. To our knowledge, we are fully compliant with TSCH as long as we use the maximum 40-channel hopping sequence. None of the timeslot timing numbers exceed 2 bytes and thus fit into the TSCH Information Elements (IEs) to transmit the timeslot timing as part of an Enhanced Beacon (EB). Moreover, the longer hopping sequence also doesn't exceed the length allowed in IEs of EBs.

For BLE, we deviate somewhat from the standard. The individual packets use an access address not standard for advertisements or use an undefined PDU type. Moreover, we use all BLE channels and do not stick to the primary advertisement channels while sending advertisement packets. Our reasoning is to enable BLE radios to discard our packets as quickly as possible after reception, but still enabling BLE radios running TBLE to join the TSCH network.

5 Evaluation

After discussing the design of TBLE and the necessary adaptions and extensions towards TSCH, we experimentally evaluate its performance. We split the evaluation into two parts. Firstly, we study the connectivity of the same physical network using different PHYs. We compare the four BLE PHY configuration with the IEEE 802.15.4 PHY as our baseline. Afterward, we evaluate and compare the performance of TSCH multi-hop communication over

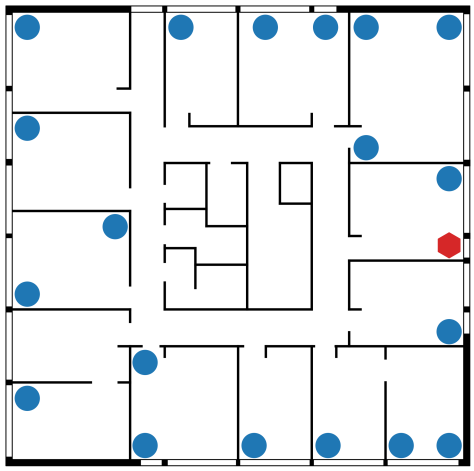


Figure E.3: Local testbed of 500 m^2. Red hexagon: TSCH PAN coordinator/Orchestra root node; Blue circles: network participants

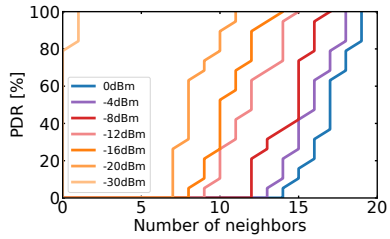
BLE and IEEE 802.15.4. For that, we use the well established autonomous TSCH scheduler Orchestra [8]. Orchestra is integrated into Contiki-NG and is a standard benchmarking solution for Contiki-NG. Orchestra builds upon the routing protocol RPL [53] that builds a routing tree on top of TSCH.

Setup. For our evaluation, we use our local testbed of 20 nodes (see Figure E.3). The testbed spans the top most floor of a university building (500 m^2) with offices and lab rooms and shares the wireless spectrum with other networks including Wi-Fi. Each node is equipped with, i.a., a nRF52840 DK board [291] and a Raspberry Pi 3B+ as observer for collecting and processing our evaluation logs. The nRF52840, which our design targets, is a Cortex-M4 microcontroller with 64 MHz clock speed, 1 MB flash and 256 KB RAM and a radio supporting, i.a., both IEEE 802.15.4 and BLE 5.

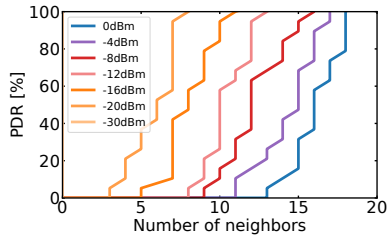
Metrics. Throughout this evaluation, we look at the following metrics. For the connectivity, we compare the single-hop reachability of nodes and the *Expected Transmission Count (ETX)* [48] for a node's neighbors. The *ETX* is the inverse of the *Packet Reception Rate (PRR)*. For our performance evaluation with Orchestra, we look at reliability (*Packet Delivery Rate (PDR)*), latency and radio duty cycle for the different modes.

5.1 Reachability

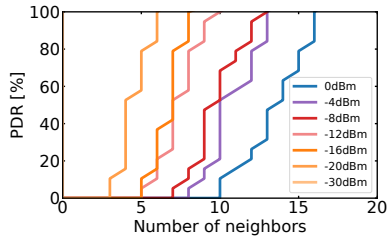
Scenario. We investigate the reachable number of nodes for each node in our testbed for the four BLE PHYs and compare it to the IEEE 802.15.4 PHY. We quantify the performance both in terms of neighbors and in terms of average neighbor ETX. We perform this evaluation for 7 different transmit powers between −30 dBm and 0 dBm. For each transmit power we run a 15-minute experiment using one of the available hopping sequences (1 channel, 4 channel, and 16 channels for all PHYs, plus 32, and 40 channels for the BLE PHYs). We use the neighbor discovery mode of the centralized TSCH scheduler MASTER [240].



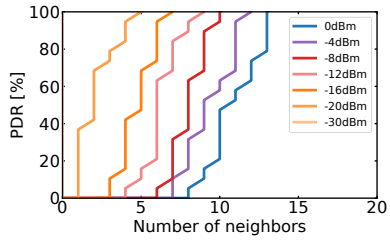
(a) Connectivity CDF for BLE 125 kbps.



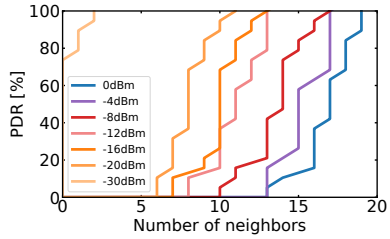
(b) Connectivity CDF for BLE 500 kbps.



(c) Connectivity CDF for BLE 1 Mbps.



(d) Connectivity CDF for BLE 2 Mbps.

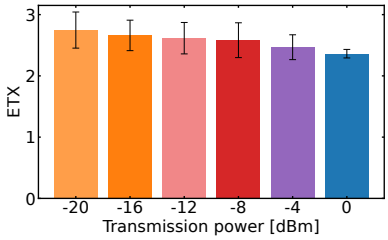


(e) Connectivity CDF for IEEE 802.15.4.

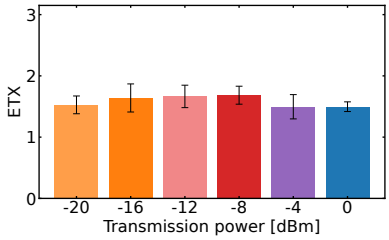
Figure E.4: Evaluation of the nodes' reachability (median reachability CDF for the different hopping sequences). The bitrate has a clearly visible influence on the communication range and thus the number of neighbors.

Results. Figure E.4 and Figure E.5 show the evaluation results for Figure E.4 shows the median number of nodes reachable for the different transmit powers. The plots show a CDF for each of the transmit powers. Each line shows the percentage of nodes in the testbed that has a certain number of neighbors. For example, the left most visible line in Figure E.4d, shows that for the 2 Mbps BLE mode at a transmit power of −20 dBm, 7 nodes can reach a single other node, 6 nodes can reach 2 other nodes, and 2, 3, and 1 nodes can reach 3, 4, and 5 nodes, respectively.

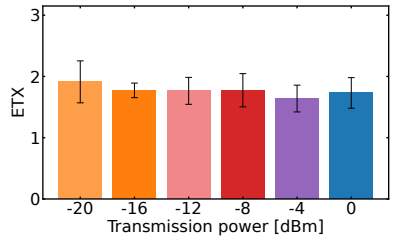
Both the BLE mode with a bitrate of 125 kbps and the IEEE 802.15.4 mode have nodes at a transmit power of 0 dBm that can reach all other 19 nodes in the testbed. Moreover, these are the only modes that can reach any neighbors at all, at a transmit power of −30 dBm. This means that only for



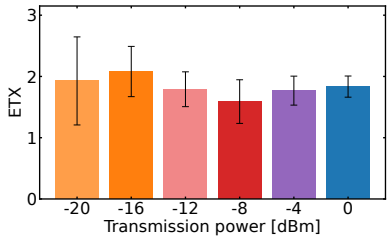
(a) Average ETX to neighbors for BLE 125 kbps.



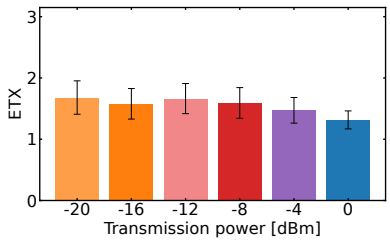
(b) Average ETX to neighbors for BLE 500 kbps.



(c) Average ETX to neighbors for BLE 1 Mbps.



(d) Average ETX to neighbors for BLE 2 Mbps.



(e) Average ETX to neighbors for IEEE 802.15.4 (250 kbps).

Figure E.5: Evaluation of the nodes' reachability showing the average ETX to a node's neighbors. IEEE 802.15.4 has the best connectivity to its neighbors.

these two modes, a TSCH network forms at −30 dBm. Generally speaking, the BLE mode with a bitrate of 125 kbps has a similar number of reachable neighbors as the IEEE 802.15.4 mode. The 500 kbps mode has a slightly lower number of reachable nodes than the 802.15.4 mode. The 2 Mbps BLE mode has with 13 possible neighbors a much lower maximum number of reachable nodes. Moreover, at a transmit power of −20 dBm, TSCH using the 2 Mbps BLE mode can just barely form a mesh network.

Figure E.5 shows the average ETX and the standard deviation for a node's transmission to all of its reachable neighbors for 6 out of the 7 transmit powers. We exclude the transmit power of −30 dBm as it is hardly usable at all. Contrary to the number of reachable nodes, the average ETX to the neighbors is significantly higher for 125 kbps BLE (Figure E.5a) than for IEEE 802.15.4 (Figure E.5e). We suspect this behavior to come from the robustness of the

modulation scheme. While both modes can reach similar numbers of neighbors, the BLE modulation scheme (GFSK) should be more affected by interference, leading to the higher expected number of transmissions to successfully reach its neighbors. The other BLE modes (Figure E.5b – E.5d) reach a lower number of nodes, but have a better connection to those.

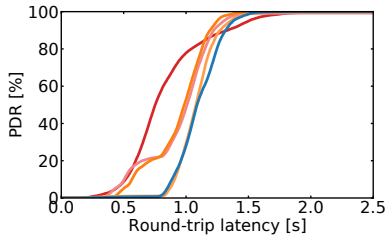
From these results, we derive that −16 dBm (dark orange line) is the lowest usable transmit power to form a proper mesh network for all PHYs. Moreover, −8 dBm (dark red line) is the transmit power with a medium number of neighbors for all PHYs. Therefore, we will use these two transmit powers for our performance evaluation below.

5.2 Performance Evaluation

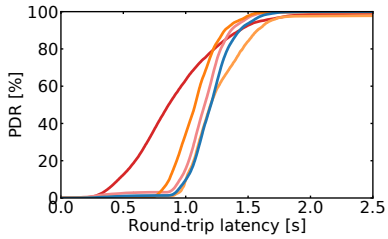
Scenario. We evaluate the performance of mesh multi-hop communication of TBLE in comparison with our baseline TSCH over IEEE 802.15.4 using the autonomous scheduler Orchestra at transmit powers of −8 and −16 dBm. We run multiple experiments with a total runtime of 4 hours for each transmit power and mode. As Orchestra builds upon the *Routing Protocol for Low-Power and Lossy Networks (RPL)* [53], we compare different routing networks with each other. Thus, while using the same physical deployment for our experiments, RPL builds a routing network outside our control, optimized for the prevalent situation. Therefore, the results might differ from our intuitive ideas. Yet, using an autonomous scheduler and a routing protocol like RPL assures us to use the best routes for each protocol and mode. This allows us to compare the effective performance of each protocol. Orchestra sends once a second a packet to a random node of the network, and the node sends back a reply on reception of the packet. We evaluate round-trip and one-way performance of this communication using latency and reliability as our metrics. For the round-trip latency, we use the node's measured time between sending and receiving the packet. For the one-way latency we use slot counts, which we convert to time. We base the latter on the slot count, as the timestamping of the serial interfaces of our testbed are not time synchronized to the required degree. Lastly, we also compare the radio duty cycle of the different PHY modes.

Results. Figure E.6 shows the performance of Orchestra for all modes. Figure E.6a and E.6b show that the round-trip latency for most BLE modes is most of the time better than for IEEE 802.15.4 with an average improvement of 10 to 20%. For 125 kbps BLE (dark red line) this is most visible. While this mode has the longest slot lengths, it seems that RPL uses fewer hops for the routing, which benefits the latency in most cases, but also leads to an increased latency for some packets. We can see a similar routing benefit for 500 kbps BLE and 1 Mbps BLE in Figure E.6a.

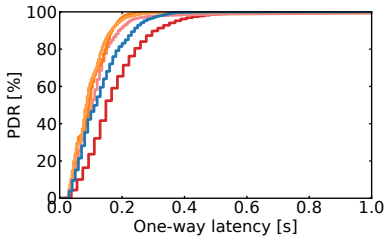
Overall, reliability is comparable for most modes. However, at a transmit power of −16 dBm, both the 125 kbps BLE mode and the 2 Mbps BLE mode do not reach maximum reliability. While we can expect that for the former the routing is not favorable to reach highest reliability, this should not be the case for the latter. For the latter (2 Mbps), we observe that the network formation takes already half an hour and therefore, we can expect the network to be not as stable as in the other modes and thus is unable to achieve maximum



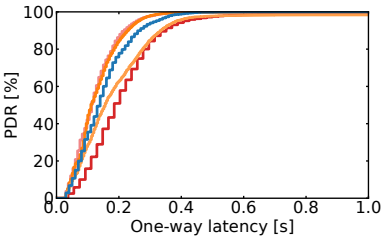
(a) Round-trip performance for a transmit power of −8 dBm.



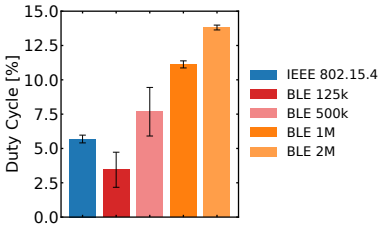
(b) Round-trip performance for a transmit power of −16 dBm.



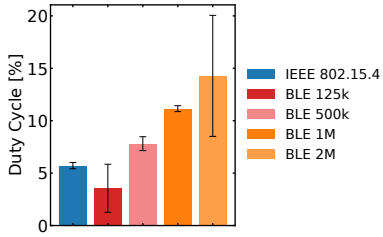
(c) One-way performance for a transmit power of −8 dBm.



(d) One-way performance for a transmit power of −16 dBm.



(e) Duty cycle for a transmit power of −8 dBm.



(f) Duty cycle for a transmit power of −16 dBm.

Figure E.6: Evaluation of the performance of Orchestra for all PHY layers. We show the performance for two transmit powers (−8 dBm and −16 dBm) and the respective radio duty cycles during operation. We show the legend for all plots in Figure E.6e and E.6f. The BLE modes usually achieve better round-trip latency performance with a slight instability of the highest bitrate (2 Mbps) at a transmit power of −16 dBm.

reliability.

In Figure E.6e and E.6f we compare the duty cycle of TBLE and TSCH over IEEE 802.15.4. We can see that the duty cycle increases with the bitrate. This can be expected, as a higher bitrate allows for a shorter slot duration to transmit the same amount of data. In Orchestra, a receiver listens in every slot. If the receiver does not encounter a packet, it keeps its radio on for the `RxWait` guard time. Therefore, if we have twice the number of slots (e.g., 1 Mbps BLE vs. IEEE 802.15.4), we have a doubling in radio-on-time for the majority of slots, those in which no communication takes place. When we multiply

the duty cycle with the respective slot length, the resulting values are almost the same. When comparing the radio duty cycle between a network with a transmit power of −8 dBm (Figure E.6e) and a transmit power of −16 dBm (Figure E.6f), we see a minor increase for all modes of 0.03 to 0.5 percantage points. This confirms that the predominant factor for the duty cycle are slots without communication.

6 Related Work

In recent years, several works looked into extending the use of Time-Slotted Channel Hopping (TSCH) into the field of other radio PHYs. Brachmann et al. [184] study the possibility of using TSCH with subGHz PHYs. The authors show its feasibility when adapting the TSCH timeslot timings. Moreover, they show the possibility of combining multiple PHYs in the same TSCH schedule. Rady et al. [186] build with g6TiSCH another work combining multiple PHYs in a single TSCH network. They perform modifications along the 6TiSCH stack to allow an intelligent choice which PHY to use in a TSCH slot. Carhacioglu et al. [190] study the co-existence of TSCH and BLE and propose a system with a common TSCH and BLE orchestrator to overcome cross-technology interference. Hajizadeh et al. [191] build a simulation framework analyzing the coexistence and amount of expectable collisions for coexisting BLE and TSCH networks.

Baddeley et al. [193] take an approach of combining BLE and TSCH. They propose 6TiSCH++ which uses the standard TSCH slots over IEEE 802.15.4 for data communication, but replaces the beaconing slots with concurrent transmissions over BLE. In 6TiSCH++, multiple subsequent concurrent BLE transmissions fit into one TSCH slot and allow for a faster transmission of control information for the TSCH network. Concurrent Transmissions (CT) are a well explored communication paradigm in IEEE 802.15.4 and their feasibility for multi-hop communication on top of BLE were shown by BlueFlood [60, 292].

On the side of pure BLE networking, Patti et al. [194] devise a connection-oriented protocol for real-time mesh communication on top of BLE that uses subnetworks, each with a central node and several peripheral nodes. The networks are linked through peripheral nodes shared between two subnetworks. Leonardi et al. [195] extend and implement that solution. In contrast, we build a single mesh network allowing communication between any two nodes. With Bluetooth Mesh [25, 26], the Bluetooth SIG standardized a mesh networking protocol on top of BLE using managed flooding. Aijaz et al. [41] experimentally study its performance using the same hardware we use for TBLE. Leonardi et al. [196] propose RESEMBLE, a protocol for Bluetooth Mesh enabling TDMA-based communication with time slots and clock synchronization over Bluetooth Mesh to allow for real-time communication in Bluetooth Mesh networks. Contrary to that solution, we create a routing-enabled solution utilizing a well established time-slotted and time-synchronized MAC layer protocol. Petersen et al. [197] extend BLE to enable efficient multi-hop IPv6 over BLE and Lee et al. [198] bring the RPL routing protocol to BLE.

Our approach to mesh networking on top of BLE is to some extent in line with these networking approaches, but differs in certain aspects. On the one

hand, we bring TSCH to another PHY and study a mesh networking approach on BLE. On the other hand, our approach differs from the approaches above in that we combine an established MAC layer (TSCH) for multi-hop routing with a widespread radio communication technology (BLE), enabling routed mesh communication on top of BLE.

7 Conclusion

Bluetooth Low Energy is a widely used communication protocol in the IoT. For advanced communication systems covering larger areas, mesh communication is necessary. While BLE offers Bluetooth Mesh, it lacks a routed mesh communication protocol. With this work, we introduce TBLE, a combination of BLE and TSCH, and a replacement for IEEE 802.15.4. We show that TSCH and the 6TiSCH network stack are a viable candidate for routed mesh communication over BLE. Moreover, with the larger amount of available frequency slots in comparison with IEEE 802.15.4, and the possibility for shorter time slots due to higher bit rates, BLE and TBLE might even be favorable over IEEE 802.15.4 and TSCH in latency-critical applications. Moreover, BLE supports longer packets, which could increase the effective bit rate even further and lead to an even more efficient use of the wireless spectrum.

Bibliography

[1] P. Valdesolo, "Scientists Study Nomophobia—Fear of Being without a Mobile Phone," Scientific American, Oct. 2015. [Online]. Available: https://www.scientificamerican.com/article/scientists-study-nomophobia-mdash-fear-of-being-without-a-mobile-phone/

[2] UpKeep Technologies, Inc., "What is the difference between Industry 3.0 and Industry 4.0?" [Online]. Available: https://upkeep.com/learning/industry-3-0-vs-industry-4-0/

[3] A. El-Askalany, "The difference between #Industry 3.0 and #Industry 4.0," Feb. 2019. [Online]. Available: https://www.linkedin.com/pulse/difference-between-industry-30-40-ahmed

[4] M. Crnjac, I. Veza, and N. Banduka, "From concept to the introduction of industry 4.0," *International Journal of Industrial Engineering and Management*, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:59481432

[5] R. Galin and R. Meshcheryakov, "Automation and robotics in the context of Industry 4.0: the shift to collaborative robots," *IOP Conference Series: Materials Science and Engineering*, vol. 537, no. 3, p. 032073, May 2019.

[6] "IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1: MAC sublayer," Tech. Rep., 2012. [Online]. Available: http://ieeexplore.ieee.org/document/6185525/

[7] Bluetooth SIG, "Bluetooth Core Specification v5.2," 2019.

[8] S. Duquennoy, B. Al Nahas, O. Landsiedel, and T. Watteyne, "Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '15. ACM, Nov. 2015.

[9] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with Glossy," in *ACM/IEEE IPSN*, 2011.

[10] A. Saifullah, Y. Xu, C. Lu, and Y. Chen, "Real-Time Scheduling for WirelessHART Networks," in *2010 31st IEEE Real-Time Systems Symposium*. IEEE, Nov. 2010.

[11] M. Palattella, N. Accettura, M. Dohler, L. Grieco, and G. Boggia, "Traffic-Aware Time-Critical Scheduling in Heavily Duty-Cycled IEEE 802.15.4e for an Industrial IoT," in *Proceedings of IEEE Sensors 2012*, 2012.

[12] HART Communication Foundation, *WirelessHART Specification 75: TDMA Data-Link Layer. HCF_SPEC-75.* HART Communication Foundation, 2008.

[13] 10.3182/20090520-3-KR-3006.00019, "Thread Network Fundamentals White Paper," Thread Group, Tech. Rep., Sep. 2022.

[14] "Matter Specification Version 1.0," Connectivity Standards Alliance, Inc., Tech. Rep., Sep. 2022. [Online]. Available: https://csa-iot.org/wp-content/uploads/2022/11/22-27349-001_Matter-1.0-Core-Specification.pdf

[15] M. Ojo and S. Giordano, "An efficient centralized scheduling algorithm in IEEE 802.15.4e TSCH networks," in *2016 IEEE Conference on Standards for Communications and Networking (CSCN).* IEEE, Oct. 2016.

[16] D. Gunatilaka and C. Lu, "Conservative Channel Reuse in Real-Time Industrial Wireless Sensor-Actuator Networks," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS).* IEEE, Jul. 2018.

[17] Apple Inc., "iPhone 15 Pro," 2024, archived at https://archive.is/I56AO. [Online]. Available: https://www.apple.com/iphone-15-pro/specs/

[18] M. Mallick, *Mobile and Wireless Design Essentials.* Wiley Publishing, Inc., Indianapolis, Indiana, 2003, ch. Wireless Networks, pp. 46–52.

[19] "IEEE Standard for Low-Rate Wireless Networks," *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pp. 1–709, 2016.

[20] T. M. W. 1.0, "LoRaWAN™ What is it? - A technical overview of LoRa® and LoRaWAN™," LoRa Alliance, Tech. Rep., 2015. [Online]. Available: https://lora-alliance.org/wp-content/uploads/2020/11/what-is-lorawan.pdf

[21] Sigfox, "What is Sigfox 0G Technology," accessed: 2023-12-09. [Online]. Available: https://www.sigfox.com/what-is-sigfox/

[22] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, "A comparative study of LPWAN technologies for large-scale IoT deployment," *ICT Express*, vol. 5, no. 1, pp. 1–7, Mar. 2019.

[23] accent systems, "Differences between NB-IOT and LTE-M," accessed: 2023-12-09. [Online]. Available: https://accent-systems.com/differences-nb-iot-lte-m/

[24] A. S. Tanenbaum, *Computer Networks*, 5th ed., D. Wetherall, Ed. Boston: Prentice Hall, 2011, includes bibliographical references and index. - Description based on print version of record.

[25] Bluetooth SIG, "Mesh Profile 1.0," 2017. [Online]. Available: https://www.bluetooth.com/specifications/specs/mesh-profile-1-0/

[26] ——, "Mesh Model 1.0," 2017. [Online]. Available: https://www.bluetooth.com/specifications/specs/mesh-model-1-0/

[27] "IEEE Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN Specific Requirements - Part 15: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPAN)," *IEEE Std 802.15.4-2003*, 2003.

[28] Connectivity Standards Alliance, "zigbee - The Full-Stack Solution for All Smart Devices," 2022, accessed: 2023-12-09. [Online]. Available: https://csa-iot.org/all-solutions/zigbee/

[29] everything RF, "What is OQPSK Modulation?" Nov. 2022. [Online]. Available: https://www.everythingrf.com/community/what-is-oqpsk-modulation

[30] D. Torrieri, *Principles of Spread-Spectrum Communication Systems*. Springer International Publishing, 2015.

[31] M. I. Benakila, L. George, and S. Femmam, "A Beacon-Aware Device For The Interconnection of Zig Bee Networks," *IFAC Proceedings Volumes*, vol. 42, no. 3, pp. 123–130, 2009.

[32] D. De Guglielmo, S. Brienza, and G. Anastasi, "IEEE 802.15.4e: A survey," *Computer Communications*, vol. 88, pp. 1–24, Aug. 2016.

[33] A. G. Ramonet and T. Noguchi, "IEEE 802.15.4 Now and Then: Evolution of the LR-WPAN Standard," in *2020 22nd International Conference on Advanced Communication Technology (ICACT)*. IEEE, Feb. 2020.

[34] N. Choudhury, R. Matam, M. Mukherjee, and J. Lloret, "A Performance-to-Cost Analysis of IEEE 802.15.4 MAC With 802.15.4e MAC Modes," *IEEE Access*, vol. 8, pp. 41 936–41 950, 2020.

[35] everything RF, "What is GFSK Modulation?" Dec. 2022. [Online]. Available: https://www.everythingrf.com/community/what-is-gfsk-modulation

[36] I. Microchip Technology, "Bluetooth® Low Energy (BLE) Link Layer Packet Types," Nov. 2023. [Online]. Available: https://microchipdeveloper.com/xwiki/bin/view/applications/ble/introduction/bluetooth-architecture/bluetooth-controller-layer/bluetooth-link-layer/Packet-Types/

[37] M. Afaneh, "What's the Maximum Data Size you can send in a Bluetooth Advertising Packet?" Apr. 2022. [Online]. Available: https://novelbits.io/maximum-data-bluetooth-advertising-packet-ble/

[38] Google and Apple, "Exposure Notification - Bluetooth Specification v1.2," Apr. 2020.

[39] M. Baert, J. Rossey, A. Shahid, and J. Hoebeke, "The Bluetooth Mesh Standard: An Overview and Experimental Evaluation," *Sensors*, vol. 18, no. 8, p. 2409, Jul. 2018.

[40] M. Woolley, "Bluetooth Mesh Networking - An Introduction for Developers," bluetooth.com, Tech. Rep., 2020. [Online]. Available: https://www.bluetooth.com/wp-content/uploads/2019/03/Mesh-Technology-Overview.pdf

[41] A. Aijaz, A. Stanoev, D. London, and V. Marot, "Demystifying the Performance of Bluetooth Mesh: Experimental Evaluation and Optimization," in *2021 Wireless Days (WD)*. IEEE, Jun. 2021.

[42] Silicon Laboratories, "AN1424: Bluetooth Mesh 1.1 Network Performance," 2023. [Online]. Available: https://www.silabs.com/documents/public/application-notes/an1424-bluetooth-mesh-11-network-performance.pdf

[43] ISA, *ISA-100.11a-2011 - Wireless Systems for Industrial Automation: Process Control and Related Applications*. ISA, 2011.

[44] G. Oikonomou, S. Duquennoy, A. Elsts, J. Eriksson, Y. Tanaka, and N. Tsiftes, "The Contiki-NG open source operating system for next generation IoT devices," *SoftwareX*, vol. 18, p. 101089, Jun. 2022.

[45] R. Teles Hermeto, A. Gallais, and F. Theoleyre, "Scheduling for IEEE802.15.4-TSCH and slow channel hopping MAC in low power industrial wireless networks: A survey," *Computer Communications*, vol. 114, pp. 84–105, Dec. 2017.

[46] A. Urke, Ø. Kure, and K. Øvsthus, "A Survey of 802.15.4 TSCH Schedulers for a Standardized Industrial Internet of Things," *Sensors*, vol. 22, no. 1, p. 15, Dec. 2021.

[47] A. Tabouche, B. Djamaa, and M. R. Senouci, "Traffic-Aware Reliable Scheduling in TSCH Networks for Industry 4.0: A Systematic Mapping Review," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 4, pp. 2834–2861, 2023.

[48] D. S. J. De Couto, "High-Throughput Routing for Multi-Hop Wireless Networks," Ph.D. dissertation, Massachusetts Institute of Technology, 2004. [Online]. Available: https://pdos.lcs.mit.edu/papers/grid:decouto-phd/thesis.pdf

[49] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Section 24.3: Dijkstra's algorithm," in *Introduction to Algorithms*, 2nd ed. MIT Press and McGraw-Hill, 2001.

[50] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[51] B. Li, Y. Ma, T. Westenbroek, C. Wu, H. Gonzalez, and C. Lu, "Wireless Routing and Control: A Cyber-Physical Case Study," in *2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, Apr. 2016.

[52] C. Wu, D. Gunatilaka, M. Sha, and C. Lu, "Real-Time Wireless Routing for Industrial Internet of Things," in *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, Apr. 2018.

[53] R. Alexander, A. Brandt, J. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, and T. Winter, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550, Mar. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6550.txt

[54] P. Thubert, "An Architecture for IPv6 over the Time-Slotted Channel Hopping Mode of IEEE 802.15.4 (6TiSCH)," RFC 9030, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9030

[55] X. Vilajosana, T. Watteyne, T. Chang, M. Vucinic, S. Duquennoy, and P. Thubert, "IETF 6TiSCH: A Tutorial," *IEEE Communications Surveys &Tutorials*, vol. 22, no. 1, pp. 595–615, 2020.

[56] T. Watteyne, X. Vilajosana, B. Kerkez, F. Chraim, K. Weekly, Q. Wang, S. Glaser, and K. Pister, "OpenWSN: a standards-based low-power wireless development environment," *Transactions on Emerging Telecommunications Technologies*, vol. 23, no. 5, pp. 480–493, Aug. 2012.

[57] K. Leentvaar and J. Flint, "The Capture Effect in FM Receivers," *IEEE Transactions on Communications*, vol. 24, 1976.

[58] P. Dutta, S. Dawson-Haggerty, Y. Chen, C.-J. M. Liang, and A. Terzis, "Design and Evaluation of a Versatile and Efficient Receiver-Initiated Link Layer for Low-Power Wireless," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys10. ACM, Nov. 2010.

[59] O. Landsiedel, F. Ferrari, and M. Zimmerling, "Chaos: versatile and efficient all-to-all data sharing and in-network processing at scale," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '13. ACM, Nov. 2013.

[60] B. A. Nahas, A. Escobar-Molero, J. Klaue, S. Duquennoy, and O. Landsiedel, "BlueFlood: Concurrent Transmissions for Multi-hop Bluetooth 5—Modeling and Evaluation," *ACM Transactions on Internet of Things*, vol. 2, no. 4, pp. 1–30, Jul. 2021.

[61] T. Istomin, A. L. Murphy, G. P. Picco, and U. Raza, "Data Prediction + Synchronous Transmissions = Ultra-low Power Wireless Sensor Networks," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '16. ACM, Nov. 2016.

[62] M. Wilhelm, V. Lenders, and J. B. Schmitt, "On the Reception of Concurrent Transmissions in Wireless Sensor Networks," *IEEE Transactions on Wireless Communications*, vol. 13, no. 12, pp. 6756–6767, Dec. 2014.

[63] C.-H. Liao, Y. Katsumata, M. Suzuki, and H. Morikawa, "Revisiting the So-Called Constructive Interference in Concurrent Transmission," in *2016 IEEE 41st Conference on Local Computer Networks (LCN)*. IEEE, Nov. 2016.

[64] M. Baddeley, C. A. Boano, A. Escobar-Molero, Y. Liu, X. Ma, V. Marot, U. Raza, K. Römer, M. Schuss, and A. Stanoev, "Understanding Concurrent Transmissions: The Impact of Carrier Frequency Offset and RF Interference on Physical Layer Performance," *ACM Transactions on Sensor Networks*, vol. 20, no. 1, pp. 1–39, Oct. 2023.

[65] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Low-power wireless bus," in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '12. ACM, Nov. 2012.

[66] M. Zimmerling, L. Mottola, P. Kumar, F. Ferrari, and L. Thiele, "Adaptive Real-Time Communication for Wireless Cyber-Physical Systems," *ACM Transactions on Cyber-Physical Systems*, vol. 1, no. 2, pp. 1–29, Feb. 2017.

[67] D. Yuan, M. Riecker, and M. Hollick, "Making 'Glossy' Networks Sparkle: Exploiting Concurrent Transmissions for Energy Efficient, Reliable, Ultra-Low Latency Communication in Wireless Control Networks," in *Wireless Sensor Networks*, 2014, vol. 8354.

[68] D. Carlson, M. Chang, A. Terzis, Y. Chen, and O. Gnawali, "Forwarder Selection in Multi-transmitter Networks," in *2013 IEEE International Conference on Distributed Computing in Sensor Systems*. IEEE, May 2013.

[69] M. Brachmann, O. Landsiedel, and S. Santini, "Concurrent Transmissions for Communication Protocols in the Internet of Things," in *2016 IEEE 41st Conference on Local Computer Networks (LCN)*. IEEE, Nov. 2016.

[70] S. Frattasi and F. D. Rosa, Eds., *Mobile positioning and tracking*, 2nd ed. Hoboken: IEEE Press, Wiley, 2017, includes bibliographical references and index.

[71] M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 3rd ed. distributed-systems.net, 2017.

[72] P. Horowitz and W. Hill, *The art of electronics*, 2nd ed. Cambridge [u.a.]: Cambridge Univ. Press, 2008.

[73] E. Kaplan and C. Hegarty, *Understanding GPS/GNSS: Principles and Applications*, 3rd ed., 2017.

[74] D. L. Mills, "Network Time Protocol (Version 3) Specification, Implementation and Analysis," RFC 1305, Mar. 1992. [Online]. Available: https://www.rfc-editor.org/info/rfc1305

[75] ——, *Computer Network Time Synchronization: The Network Time Protocol*. CRC Press, 2006, p. 286.

[76] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 147–163, Dec. 2002.

[77] nsnam, "ns-3 Network Simulator," 2023. [Online]. Available: https://www.nsnam.org/

[78] Z. Bojthe, L. Meszaros, G. Szászkő, R. Hornig, A. Varga, and A. Török, "INET framework," 2023. [Online]. Available: https://inet.omnetpp.org/

[79] OpenSim Ltd., "OMNeT++ Discrete Event Simulator," 2023. [Online]. Available: https://omnetpp.org/

[80] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-Level Sensor Network Simulation with COOJA," in *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*. IEEE, Nov. 2006.

[81] antmicro, "RENODE." [Online]. Available: https://renode.io/

[82] E. Municio, G. Daneels, M. Vučinić, S. Latré, J. Famaey, Y. Tanaka, K. Brun, K. Muraoka, X. Vilajosana, and T. Watteyne, "Simulating 6TiSCH networks," *Transactions on Emerging Telecommunications Technologies*, vol. 30, no. 3, Aug. 2018.

[83] A. Elsts, "TSCH-Sim: Scaling Up Simulations of TSCH and 6TiSCH Networks," *Sensors*, vol. 20, no. 19, p. 5663, Oct. 2020.

[84] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel, "FlockLab: a testbed for distributed, synchronized tracing and profiling of wireless embedded systems," in *ACM/IEEE IPSN*, ser. IPSN '13, Philadelphia, Pennsylvania, USA, 2013, p. 153–166. [Online]. Available: https://doi.org/10.1145/2461381.2461402

[85] R. Trüb, R. D. Forno, L. Daschinger, A. Biri, J. Beutel, and L. Thiele, "Non-Intrusive Distributed Tracing of Wireless IoT Devices with the FlockLab 2 Testbed," *ACM TIOT*, vol. 3, no. 1, pp. 1–31, feb 2022.

[86] R. Trüb, R. Da Forno, L. Sigrist, L. Mühlebach, A. Biri, J. Beutel, and L. Thiele, "FlockLab 2: Multi-Modal Testing and Validation for Wireless IoT," in *CPS-IoTBench*. ETH Zurich, 2020.

[87] M. Schuß, C. A. Boano, M. Weber, and K. Römer, "A Competition to Push the Dependability of Low-Power Wireless Protocols to the Edge," in *Proceedings of the* 14^{th} *International Conference on Embedded Wireless Systems and Networks (EWSN)*. Uppsala, Sweden: Junction Publishing, Feb. 2017, pp. 54–65.

[88] M. Doddavenkatappa, M. C. Chan, and A. L. Ananda, "Indriya: A low-cost, 3D wireless sensor network testbed," in *Testbeds and Research Infrastructure. Development of Networks and Communities: 7th International ICST Conference, TridentCom 2011, Shanghai, China, April 17-19, 2011, Revised Selected Papers 7*. Springer, 2012, pp. 302–316.

[89] P. Appavoo, E. K. William, M. C. Chan, and M. Mohammad, "Indriya 2: A heterogeneous wireless sensor network (wsn) testbed," in *Testbeds and Research Infrastructures for the Development of Networks and Communities: 13th EAI International Conference, TridentCom 2018, Shanghai, China, December 1-3, 2018, Proceedings 13*. Springer, 2019, pp. 3–19.

[90] J. Munoz, F. Rincon, T. Chang, X. Vilajosana, B. Vermeulen, T. Walcarius, W. van de Meerssche, and T. Watteyne, "OpenTestBed: Poor Man's IoT Testbed," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, apr 2019.

[91] D. Gunatilaka, "The WUSTL Wireless Sensor Network Testbed." [Online]. Available: https://cps.cse.wustl.edu/index.php?title=The_WUSTL_Wireless_Sensor_Network_Testbed

[92] P. Sommer and F. Sutton, "VIADUCT: Bridging the Gap between Testbeds and Real-World Cyber-Physical Systems," in *Proceedings of the 2021 International Conference on Embedded Wireless Systems and Networks (EWSN)*, 2021.

[93] R. Lim, B. Maag, B. Dissler, J. Beutel, and L. Thiele, "A testbed for fine-grained tracing of time sensitive behavior in wireless sensor networks," in *IEEE LCN Workshops*, 2015.

[94] M. Tancreti, M. S. Hossain, S. Bagchi, and V. Raghunathan, "Aveksha: a hardware-software approach for non-intrusive tracing and profiling of wireless embedded systems," in *ACM SenSys*, Seattle, Washington, 2011, p. 288–301.

[95] P. Sommer and B. Kusy, "Minerva: distributed tracing and debugging in wireless sensor networks," in *ACM SenSys*, Roma, Italy, 2013, pp. 1 –14.

[96] Y. Li, J. Ma, and T. Zhang, "HATBED: Hardware Assisted Tracing Testbed for Embedded Networked Sensor Systems," in *ACM SenSys*, Shenzhen, China, 2018, p. 327–328.

[97] IoT Chalmers, "IoT-Testbed," 2021. [Online]. Available: https://github.com/iot-chalmers/iot-testbed

[98] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne, "FIT IoT-LAB: A large scale open experimental IoT testbed," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. IEEE, dec 2015.

[99] G. Werner-Allen, P. Swieskowski, and M. Welsh, "MoteLab: a wireless sensor network testbed," in *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks*. IEEE, 2005.

[100] M. Campbell, A. Hoane, and F.-h. Hsu, "Deep Blue," *Artificial Intelligence*, vol. 134, no. 1–2, pp. 57–83, Jan. 2002.

[101] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[102] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun. 2016.

[103] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[104] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[105] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," 2018.

[106] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[107] M. Schreiner, "GPT-4 architecture, datasets, costs and more leaked," the decoder, Jul. 2023. [Online]. Available: https://the-decoder.com/gpt-4-architecture-datasets-costs-and-more-leaked/

[108] M. Singh, J. Cambronero, S. Gulwani, V. Le, C. Negreanu, and G. Verbruggen, "CodeFusion: A Pre-trained Diffusion Model for Code Generation," 2023.

[109] "Keras Applications." [Online]. Available: https://keras.io/api/applications/

[110] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," 2017.

[111] Edge Impulse, "FOMO: Object detection for constrained devices," 2023. [Online]. Available: https://docs.edgeimpulse.com/docs/edge-impulse-studio/learning-blocks/object-detection/fomo-object-detection-for-constrained-devices

[112] Arduino, "Nicla Vision," 2024. [Online]. Available: https://docs.arduino.cc/hardware/nicla-vision

[113] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[114] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/d8330f857a17c53d217014ee776bfd50-Paper.pdf

[115] S. Bhattacharya and N. D. Lane, "Sparsification and Separation of Deep Learning Layers for Constrained Resource Inference on Wearables," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '16. ACM, Nov. 2016.

[116] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, and T. Abdelzaher, "Deep compressive offloading: speeding up neural network inference by trading edge computation for network latency," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, ser. SenSys '20. ACM, Nov. 2020.

[117] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "SPINN: Synergistic Progressive Inference of Neural Networks over Device and Cloud," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '20. ACM, Sep. 2020.

[118] M. Ojo, S. Giordano, G. Portaluri, D. Adami, and M. Pagano, "An energy efficient centralized scheduling scheme in TSCH networks," in *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, May 2017.

[119] Y. Jin, P. Kulkarni, J. Wilcox, and M. Sooriyabandara, "A centralized scheduling algorithm for IEEE 802.15.4e TSCH based industrial low power wireless networks," in *2016 IEEE Wireless Communications and Networking Conference*. IEEE, Apr. 2016.

[120] G. Gaillard, D. Barthel, F. Theoleyre, and F. Valois, "High-reliability scheduling in deterministic wireless multi-hop networks," in *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. IEEE, Sep. 2016.

[121] F. Dobslaw, T. Zhang, and M. Gidlund, "End-to-End Reliability-Aware Scheduling for Wireless Sensor Networks," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 2, pp. 758–767, Apr. 2016.

[122] E. Khorov, A. Lyakhov, and R. Yusupov, "Scheduling of Dedicated and Shared Links for Fast and Reliable Data Delivery in IEEE 802.15.4 TSCH Networks," in *2019 International Conference on Engineering and Telecommunication (EnT)*. IEEE, Nov. 2019.

[123] M. R. Palattella, N. Accettura, L. A. Grieco, G. Boggia, M. Dohler, and T. Engel, "On Optimal Scheduling in Duty-Cycled Industrial IoT Applications Using IEEE802.15.4e TSCH," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3655–3666, Oct. 2013.

[124] A. Darbandi and M. K. Kim, "Path Collision-aware Real-time Link Scheduling for TSCH Wireless Networks," *KSII Transactions on Internet & Information Systems*, vol. 13, no. 9, 2019.

[125] J. P. G. Rugamba, D. L. Mai, and M. K. Kim, *Implementation of a Centralized Scheduling Algorithm for IEEE 802.15.4e TSCH.* Springer International Publishing, 2019, pp. 118–129.

[126] K.-H. Choi and S.-H. Chung, *A New Centralized Link Scheduling for 6TiSCH Wireless Industrial Networks.* Springer International Publishing, 2016, pp. 360–371.

[127] K. Choi and S.-H. Chung, "Enhanced time-slotted channel hopping scheduling with quick setup time for industrial Internet of Things networks," *International Journal of Distributed Sensor Networks*, vol. 13, no. 6, p. 155014771771362, Jun. 2017.

[128] E. Mozaffari Ahrar and M. Nassiri, "T2AS: Topology/Traffic Aware Scheduling to Optimize the End-to-end Delay in IEEE802.154e-TSCH Networks," *TABRIZ JOURNAL OF ELECTRICAL ENGINEERING*, vol. 51, no. 1, pp. 129–137, 2021. [Online]. Available: https://tjee.tabrizu.ac.ir/article_13330.html

[129] G. Portaluri and S. Giordano, "Gambling on fairness: a fair scheduler for IIoT communications based on the shell game," in *2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD).* IEEE, Sep. 2020.

[130] M. G. Gaitán, L. Almeida, P. M. D'orey, P. M. Santos, and T. Watteyne, "Minimal-Overlap Centrality for Multi-Gateway Designation in Real-Time TSCH Networks," *ACM Transactions on Embedded Computing Systems*, vol. 23, no. 1, pp. 1–17, Jan. 2024.

[131] R. Brummet, D. Gunatilaka, D. Vyas, O. Chipara, and C. Lu, "A Flexible Retransmission Policy for Industrial Wireless Sensor Actuator Networks," in *2018 IEEE International Conference on Industrial Internet (ICII).* IEEE, Oct. 2018.

[132] A. Tinka, T. Watteyne, and K. Pister, *A Decentralized Scheduling Algorithm for Time Synchronized Channel Hopping.* Springer Berlin Heidelberg, 2010, pp. 201–216.

[133] X. Vilajosana, K. Pister, and T. Watteyne, "Minimal IPv6 over the TSCH Mode of IEEE 802.15.4e (6TiSCH) Configuration," RFC 8180, May 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8180.txt

[134] T. Chang, M. Vučinić, X. Vilajosana, S. Duquennoy, and D. R. Dujovne, "6TiSCH Minimal Scheduling Function (MSF)," RFC 9033, May 2021. [Online]. Available: https://rfc-editor.org/rfc/rfc9033.txt

[135] T. Chang, T. Watteyne, Q. Wang, and X. Vilajosana, "LLSF: Low Latency Scheduling Function for 6TiSCH Networks," in *2016 International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, May 2016.

[136] V. Kotsiou, G. Z. Papadopoulos, P. Chatzimisios, and F. Theoleyre, "LDSF: Low-Latency Distributed Scheduling Function for Industrial Internet of Things," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8688–8699, Sep. 2020.

[137] N. Accettura, M. R. Palattella, G. Boggia, L. A. Grieco, and M. Dohler, "Decentralized Traffic Aware Scheduling for multi-hop Low power Lossy Networks in the Internet of Things," in *2013 IEEE 14th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*. IEEE, Jun. 2013.

[138] M. Domingo-Prieto, T. Chang, X. Vilajosana, and T. Watteyne, "Distributed PID-Based Scheduling for 6TiSCH Networks," *IEEE Communications Letters*, vol. 20, no. 5, pp. 1006–1009, May 2016.

[139] M. R. Palattella, T. Watteyne, Q. Wang, K. Muraoka, N. Accettura, D. Dujovne, L. A. Grieco, and T. Engel, "On-the-Fly Bandwidth Reservation for 6TiSCH Wireless Industrial Networks," *IEEE Sensors Journal*, vol. 16, no. 2, pp. 550–560, Jan. 2016.

[140] S. Jeong, H.-S. Kim, J. Paek, and S. Bahk, "OST: On-Demand TSCH Scheduling with Traffic-Awareness," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. IEEE, Jul. 2020.

[141] J. Jung, D. Kim, T. Lee, J. Kang, N. Ahn, and Y. Yi, "Distributed Slot Scheduling for QoS Guarantee over TSCH-based IoT Networks via Adaptive Parameterization," in *2020 19th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, Apr. 2020.

[142] H. Nguyen-Duy, T. Ngo-Quynh, F. KOJIMA, T. Pham-Van, T. Nguyen-Duc, and S. Luongoudon, "RL-TSCH: A Reinforcement Learning Algorithm for Radio Scheduling in TSCH 802.15.4e," in *2019 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, Oct. 2019.

[143] Y. H. Pratama and S. Chung, "RL-SF: Reinforcement Learning based Scheduling Function for Distributed TSCH Networks," in *2022 IEEE 12th International Conference on Electronics Information and Emergency Communication (ICEIEC)*. IEEE, Jul. 2022.

[144] O. Tavallaie, S. M. Zandavi, H. Haddadi, and A. Y. Zomaya, "GT-TSCH: Game-Theoretic Distributed TSCH Scheduler for Low-Power IoT Networks," 2023.

[145] A. Elsts, X. Fafoutis, J. Pope, G. Oikonomou, R. Piechocki, and I. Craddock, "Scheduling High-Rate Unpredictable Traffic in IEEE 802.15.4 TSCH Networks," in *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, Jun. 2017.

[146] J. Shi, M. Sha, and Z. Yang, "Distributed Graph Routing and Scheduling for Industrial Wireless Sensor-Actuator Networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1669–1682, Aug. 2019.

[147] S. Oh, D. Hwang, K.-H. Kim, and K. Kim, "Escalator: An Autonomous Scheduling Scheme for Convergecast in TSCH," *Sensors*, vol. 18, no. 4, p. 1209, Apr. 2018.

[148] S. Kim, H.-S. Kim, and C. Kim, "ALICE: autonomous link-based cell scheduling for TSCH," in *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*, ser. IPSN '19. ACM, Apr. 2019.

[149] S. Jeong, J. Paek, H.-S. Kim, and S. Bahk, "TESLA: Traffic-Aware Elastic Slotframe Adjustment in TSCH Networks," *IEEE Access*, vol. 7, pp. 130 468–130 483, 2019.

[150] J. Jung, D. Kim, J. Hong, J. Kang, and Y. Yi, "Parameterized slot scheduling for adaptive and autonomous TSCH networks," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, Apr. 2018.

[151] S. Rekik, N. Baccour, M. Jmaiel, K. Drira, and L. A. Grieco, "Autonomous and traffic-aware scheduling for TSCH networks," *Computer Networks*, vol. 135, pp. 201–212, Apr. 2018.

[152] X. Cheng and M. Sha, "ATRIA: Autonomous Traffic-Aware Scheduling for Industrial Wireless Sensor-Actuator Networks," in *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. IEEE, Nov. 2021.

[153] ——, "Autonomous Traffic-Aware Scheduling for Industrial Wireless Sensor-Actuator Networks," *ACM Transactions on Sensor Networks*, vol. 19, no. 2, pp. 1–25, Feb. 2023.

[154] M. Osman and F. Nabki, "OSCAR: An Optimized Scheduling Cell Allocation Algorithm for Convergecast in IEEE 802.15.4e TSCH Networks," *Sensors*, vol. 21, no. 7, p. 2493, Apr. 2021.

[155] S. Kim, H.-S. Kim, and C.-k. Kim, "A3: Adaptive Autonomous Allocation of TSCH Slots," in *Proceedings of the 20th International Conference on Information Processing in Sensor Networks (co-located with CPS-IoT Week 2021)*, ser. IPSN '21. ACM, May 2021.

[156] A. R. Urke, Ø. Kure, and K. Øvsthus, "Experimental Evaluation of the Layered Flow-Based Autonomous TSCH Scheduler," *IEEE Access*, vol. 11, pp. 3970–3982, 2023.

[157] ——, "Autonomous Flow-Based TSCH Scheduling for Heterogeneous Traffic Patterns: Challenges, Design, Simulation, and Testbed Evaluation," *IEEE Open Journal of the Communications Society*, vol. 4, pp. 2357–2372, 2023.

[158] P. Thubert, M. R. Palattella, and T. Engel, "6TiSCH centralized scheduling: When SDN meet IoT," in *2015 IEEE Conference on Standards for Communications and Networking (CSCN)*. IEEE, Oct. 2015.

[159] G. Exarchakos, I. Oztelcan, D. Sarakiotis, and A. Liotta, "plexi: Adaptive re-scheduling web-service of time synchronized low-power wireless networks," *Journal of Network and Computer Applications*, vol. 81, pp. 62–73, Mar. 2017.

[160] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, Apr. 2015.

[161] M. Baddeley, R. Nejabati, G. Oikonomou, M. Sooriyabandara, and D. Simeonidou, "Evolving SDN for Low-Power IoT Networks," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, Jun. 2018.

[162] P. Du and G. Roussos, "Adaptive time slotted channel hopping for wireless sensor networks," in *2012 4th Computer Science and Electronic Engineering Conference (CEEC)*. IEEE, Sep. 2012.

[163] R. Tavakoli, M. Nabi, T. Basten, and K. Goossens, "Enhanced Time-Slotted Channel Hopping in WSNs Using Non-intrusive Channel-Quality Estimation," in *2015 IEEE 12th International Conference on Mobile Ad Hoc and Sensor Systems*. IEEE, Oct. 2015.

[164] P. H. Gomes, T. Watteyne, and B. Krishnamachari, "MABO-TSCH: Multihop and blacklist-based optimized time synchronized channel hopping," *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 7, Aug. 2017.

[165] A. Elsts, X. Fafoutis, R. Piechocki, and I. Craddock, "Adaptive channel selection in IEEE 802.15.4 TSCH networks," in *2017 Global Internet of Things Summit (GIoTS)*. IEEE, Jun. 2017.

[166] P. Minet, I. Khoufi, and A. Laouiti, "Increasing reliability of a TSCH network for the industry 4.0," in *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*. IEEE, Oct. 2017.

[167] T. Lagos Jenschke, R.-A. Koutsiamanis, G. Z. Papadopoulos, and N. Montavont, "Multi-path Selection in RPL Based on Replication and Elimination," in *Ad-hoc, Mobile, and Wireless Networks*, 2018, vol. 11104.

[168] E. Mozaffari Ahrar, M. Nassiri, and F. Theoleyre, "Multipath aware scheduling for high reliability and fault tolerance in low power industrial networks," *Journal of Network and Computer Applications*, vol. 142, pp. 25–36, Sep. 2019.

[169] A. C. Estrin, T. Lagos Jenschke, G. Z. Papadopoulos, J. Ignacio Alvarez-Hamelin, and N. Montavont, "Thorough Investigation of multipath Techniques in RPL based Wireless Networks," in *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, Jul. 2020.

[170] G. Gaillard, D. Barthel, F. Theoleyre, and F. Valois, *Kausa: KPI-aware Scheduling Algorithm for Multi-flow in Multi-hop IoT Networks*. Springer International Publishing, 2016, pp. 47–61.

[171] S. Biswas and R. Morris, "ExOR: opportunistic multi-hop routing for wireless networks," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, pp. 133–144, Aug. 2005.

[172] S. Chachulski, M. Jennings, S. Katti, and D. Katabi, "Trading structure for randomness in wireless opportunistic routing," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, pp. 169–180, Aug. 2007.

[173] R. R. Choudhury and N. H. Vaidya, "MAC-layer anycasting in ad hoc networks," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 1, pp. 75–80, Jan. 2004.

[174] O. Landsiedel, E. Ghadimi, S. Duquennoy, and M. Johansson, "Low Power, Low delay: Oportunistic Routing meets Duty Cycling," in *2012 ACM/IEEE 11th International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, Apr. 2012.

[175] S. Duquennoy, O. Landsiedel, and T. Voigt, "Let the tree Bloom: scalable opportunistic routing with ORPL," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '13. ACM, Nov. 2013.

[176] J. So and H. Byun, "Load-Balanced Opportunistic Routing for Duty-Cycled Wireless Sensor Networks," *IEEE Transactions on Mobile Computing*, vol. 16, no. 7, pp. 1940–1955, Jul. 2017.

[177] A. Hawbani, X. Wang, Y. Sharabi, A. Ghannami, H. Kuhlani, and S. Karmoshi, "LORA: Load-Balanced Opportunistic Routing for Asynchronous Duty-Cycled WSN," *IEEE Transactions on Mobile Computing*, vol. 18, no. 7, pp. 1601–1615, Jul. 2019.

[178] W. Wu, X. Wang, A. Hawbani, P. Liu, L. Zhao, and A. Al-Dubai, "FLORA: Fuzzy Based Load-Balanced Opportunistic Routing for Asynchronous Duty-Cycled WSNs," *IEEE Transactions on Mobile Computing*, vol. 22, no. 1, pp. 253–268, Jan. 2023.

[179] M. U. Farooq, X. Wang, A. Hawbani, L. Zhao, A. Al-Dubai, and O. Busaileh, "SDORP: SDN Based Opportunistic Routing for Asynchronous Wireless Sensor Networks," *IEEE Transactions on Mobile Computing*, vol. 22, no. 8, pp. 4912–4929, Aug. 2023.

[180] T. Huynh, F. Theoleyre, and W.-J. Hwang, "On the interest of opportunistic anycast scheduling for wireless low power lossy networks," *Computer Communications*, vol. 104, pp. 55–66, May 2017.

[181] R. T. Hermeto, A. Gallais, and F. Theoleyre, "Is Link-Layer Anycast Scheduling Relevant for IEEE 802.15.4-TSCH Networks?" in *2019 IEEE 44th LCN Symposium on Emerging Topics in Networking (LCN Symposium)*. IEEE, Oct. 2019.

[182] I. Hosni and F. Théoleyre, *Adaptive k-cast Scheduling for High-Reliability and Low-Latency in IEEE802.15.4-TSCH.* Springer International Publishing, 2018, pp. 3–14.

[183] Y. Jin, U. Raza, and M. Sooriyabandara, "BOOST: Bringing Opportunistic ROuting and Effortless-Scheduling to TSCH MAC," in *2018 IEEE Global Communications Conference (GLOBECOM).* IEEE, Dec. 2018.

[184] M. Brachmann, S. Duquennoy, N. Tsiftes, and T. Voigt, "IEEE 802.15.4 TSCH in Sub-GHz: Design Considerations and Multi-band Support," in *2019 IEEE 44th Conference on Local Computer Networks (LCN).* IEEE, Oct. 2019.

[185] D. Van Leemput, J. Bauwens, R. Elsas, J. Hoebeke, W. Joseph, and E. De Poorter, "Adaptive multi-PHY IEEE802.15.4 TSCH in sub-GHz industrial wireless networks," *Ad Hoc Networks*, vol. 111, p. 102330, Feb. 2021.

[186] M. Rady, Q. Lampin, D. Barthel, and T. Watteyne, "g6TiSCH: Generalized 6TiSCH for Agile Multi-PHY Wireless Networking," *IEEE Access*, vol. 9, pp. 84 465–84 479, 2021.

[187] M. Haubro, C. Orfanidis, G. Oikonomou, and X. Fafoutis, "TSCH-over-LoRA: Long Range and Reliable IPv6 Multi-hop Networks for the Internet of Things," *Internet Technology Letters*, vol. 3, no. 4, May 2020.

[188] D. M. King, B. G. Nickerson, and W. Song, "Evaluation of ultra-wideband radio for industrial wireless control," in *2017 IEEE 38th Sarnoff Symposium.* IEEE, Sep. 2017.

[189] M. Charlier, B. Quoitin, and D. Hauweele, "Challenges in using time slotted channel hopping with ultra wideband communications," in *Proceedings of the International Conference on Internet of Things Design and Implementation*, ser. IoTDI '19. ACM, Apr. 2019.

[190] O. Carhacioglu, P. Zand, and M. Nabi, "Cooperative Coexistence of BLE and Time Slotted Channel Hopping Networks," in *2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC).* IEEE, Sep. 2018.

[191] H. Hajizadeh, M. Nabi, M. Vermeulen, and K. Goossens, "Coexistence Analysis of Co-Located BLE and IEEE 802.15.4 TSCH Networks," *IEEE Sensors Journal*, vol. 21, no. 15, pp. 17 360–17 372, Aug. 2021.

[192] Nordic Semiconductor, "nRF52840." [Online]. Available: https://www.nordicsemi.com/products/nrf52840

[193] M. Baddeley, A. Aijaz, U. Raza, A. Stanoev, Y. Jin, M. Schuß, C. A. Boano, and G. Oikonomou, "6TiSCH++ with Bluetooth 5 and Concurrent Transmissions," in *Proceedings of the 2021 International Conference on Embedded Wireless Systems and Networks (EWSN)*, 2021.

[194] G. Patti, L. Leonardi, and L. Lo Bello, "A Bluetooth Low Energy real-time protocol for Industrial Wireless mesh Networks," in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*. IEEE, Oct. 2016.

[195] L. Leonardi, G. Patti, and L. Lo Bello, "Multi-Hop Real-Time Communications Over Bluetooth Low Energy Industrial Wireless Mesh Networks," *IEEE Access*, vol. 6, pp. 26 505–26 519, 2018.

[196] L. Leonardi, L. Lo Bello, and G. Patti, "RESEMBLE: A Real-Time Stack for Synchronized Mesh Mobile Bluetooth Low Energy Networks," *Applied System Innovation*, vol. 6, no. 1, p. 19, Jan. 2023.

[197] H. Petersen, T. C. Schmidt, and M. Wählisch, "Mind the gap: multi-hop IPv6 over BLE in the IoT," in *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, ser. CoNEXT '21. ACM, Dec. 2021.

[198] T. Lee, M.-S. Lee, H.-S. Kim, and S. Bahk, "A Synergistic Architecture for RPL over BLE," in *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, Jun. 2016.

[199] C. Bae, S. Yang, M. Baddeley, A. Elsts, and I. Haque, "BlueTiSCH: A Multi-PHY Simulation of Low-Power 6TiSCH IoT Networks," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*. IEEE, Dec. 2022.

[200] BeagleBoard.org Foundation, "SeeedStudio BeagleBone® Green." [Online]. Available: https://www.beagleboard.org/boards/seeedstudio-beaglebone-green

[201] S. Subedi and J.-Y. Pyun, "A Survey of Smartphone-Based Indoor Positioning System Using RF-Based Wireless Technologies," *Sensors*, vol. 20, no. 24, p. 7230, Dec. 2020.

[202] P. Bahl and V. N. Padmanabhan, "RADAR: an in-building RF-based user location and tracking system," in *Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, ser. INFCOM-00. IEEE, 2000.

[203] I. Guvenc, C. Abdallah, R. Jordan, and O. Dedeoglu, "Enhancements to RSS based indoor tracking systems using Kalman filters," in *GSPx & International Signal Processing Conference*, 2003, pp. 91–102.

[204] D. Fox, J.Hightower, L. Liao, D. Schulz, and G. Borriello, "Bayesian filtering for location estimation," *IEEE Pervasive Computing*, vol. 2, no. 3, pp. 24–33, Jul. 2003.

[205] J. Hightower and G. Borriello, "Particle Filters for Location Estimation in Ubiquitous Computing: A Case Study," in *UbiComp 2004: Ubiquitous Computing*, N. Davies, E. D. Mynatt, and I. Siio, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 88–106.

[206] A. S. Paul and E. A. Wan, "Wi-Fi based indoor localization and tracking using sigma-point Kalman filtering methods," in *2008 IEEE/ION Position, Location and Navigation Symposium.* IEEE, 2008.

[207] C.-L. Wu, L.-C. Fu, and F.-L. Lian, "WLAN location determination in e-home via support vector classification," in *IEEE International Conference on Networking, Sensing and Control, 2004.* IEEE, 2004.

[208] J. Pan, W.-J. Zheng, Q. Yang, and H. Hu, "Transfer learning for WiFi-based indoor localization," in *AAAI Conference on Artificial Intelligence*, 2008. [Online]. Available: https://cdn.aaai.org/Workshops/2008/WS-08-13/WS08-13-008.pdf

[209] K.-H. Chow, S. He, J. Tan, and S.-H. G. Chan, "Efficient Locality Classification for Indoor Fingerprint-Based Systems," *IEEE Transactions on Mobile Computing*, vol. 18, no. 2, pp. 290–304, Feb. 2019.

[210] W. Zhang, K. Liu, W. Zhang, Y. Zhang, and J. Gu, "Deep Neural Networks for wireless localization in indoor and outdoor environments," *Neurocomputing*, vol. 194, pp. 279–287, Jun. 2016.

[211] A. Thaljaoui, T. Val, N. Nasri, and D. Brulin, "BLE localization using RSSI measurements and iRingLA," in *2015 IEEE International Conference on Industrial Technology (ICIT).* IEEE, Mar. 2015.

[212] A. I. Kyritsis, P. Kostopoulos, M. Deriaz, and D. Konstantas, "A BLE-based probabilistic room-level localization method," in *2016 International Conference on Localization and GNSS (ICL-GNSS).* IEEE, Jun. 2016.

[213] R. Faragher and R. Harle, "Location Fingerprinting With Bluetooth Low Energy Beacons," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 11, pp. 2418–2428, Nov. 2015.

[214] Y. Zhuang, J. Yang, Y. Li, L. Qi, and N. El-Sheimy, "Smartphone-Based Indoor Localization with Bluetooth Low Energy Beacons," *Sensors*, vol. 16, no. 5, p. 596, Apr. 2016.

[215] A. Jimenez and F. Seco, "Finding objects using UWB or BLE localization technology: A museum-like use case," in *2017 International Conference on Indoor Positioning and Indoor Navigation (IPIN).* IEEE, Sep. 2017.

[216] A. Koutris, T. Siozos, Y. Kopsinis, A. Pikrakis, T. Merk, M. Mahlig, S. Papaharalabos, and P. Karlsson, "Deep Learning-Based Indoor Localization Using Multi-View BLE Signal," *Sensors*, vol. 22, no. 7, p. 2759, Apr. 2022.

[217] B. Perez, T. J. Pierson, G. Mazzaro, and D. Kotz, "Identification and Classification of Electronic Devices Using Harmonic Radar," in *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT).* IEEE, Jun. 2023.

[218] V. Brik, S. Banerjee, M. Gruteser, and S. Oh, "Wireless device identification with radiometric signatures," in *Proceedings of the 14th ACM international conference on Mobile computing and networking*, ser. MobiCom08. ACM, Sep. 2008.

[219] K. Sankhe, M. Belgiovine, F. Zhou, S. Riyaz, S. Ioannidis, and K. Chowdhury, "ORACLE: Optimized Radio clAssification through Convolutional neuraL nEtworks," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. IEEE, Apr. 2019.

[220] L. Xie, L. Peng, J. Zhang, and A. Hu, "Radio frequency fingerprint identification for Internet of Things: A survey," *Security and Safety*, vol. 3, p. 2023022, Sep. 2023.

[221] A. Al-Shawabka, F. Restuccia, S. D'Oro, T. Jian, B. Costa Rendon, N. Soltani, J. Dy, S. Ioannidis, K. Chowdhury, and T. Melodia, "Exposing the Fingerprint: Dissecting the Impact of the Wireless Channel on Radio Fingerprinting," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. IEEE, Jul. 2020.

[222] D. Zhang, J. Ma, Q. Chen, and L. M. Ni, "An RF-Based System for Tracking Transceiver-Free Objects," in *Fifth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom'07)*. IEEE, 2007.

[223] C. Xu, B. Firner, R. S. Moore, Y. Zhang, W. Trappe, R. Howard, F. Zhang, and N. An, "SCPL: indoor device-free multi-subject counting and localization using radio signal strength," in *Proceedings of the 12th international conference on Information processing in sensor networks*, ser. IPSN 2013. ACM, Apr. 2013.

[224] Y. Chen, W. Dong, Y. Gao, X. Liu, and T. Gu, "Rapid: A Multimodal and Device-free Approach Using Noise Estimation for Robust Person Identification," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 3, pp. 1–27, Sep. 2017.

[225] M. F. R. M. Billah, N. Saoda, J. Gao, and B. Campbell, "BLE Can See: A Reinforcement Learning Approach for RF-based Indoor Occupancy Detection," in *Proceedings of the 20th International Conference on Information Processing in Sensor Networks (co-located with CPS-IoT Week 2021)*, ser. IPSN '21. ACM, May 2021.

[226] H. Choi, M. Fujimoto, T. Matsui, S. Misaki, and K. Yasumoto, "Wi-CaL: WiFi Sensing and Machine Learning Based Device-Free Crowd Counting and Localization," *IEEE Access*, vol. 10, pp. 24 395–24 410, 2022.

[227] W. Wang, F. Nikseresht, V. G. Rajan, J. Gao, and B. Campbell, "Enabling Ubiquitous Occupancy Detection in Smart Buildings: A WiFi FTM-Based Approach," in *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*. IEEE, Jun. 2023.

[228] S. Sigg, U. Blanke, and G. Troster, "The telepathic phone: Frictionless activity recognition from WiFi-RSSI," in *2014 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, Mar. 2014.

[229] Y. Wang, J. Liu, Y. Chen, M. Gruteser, J. Yang, and H. Liu, "E-eyes: device-free location-oriented activity identification using fine-grained WiFi signatures," in *Proceedings of the 20th annual international conference on Mobile computing and networking*, ser. MobiCom'14. ACM, Sep. 2014.

[230] Z. Chen, C. Cai, T. Zheng, J. Luo, J. Xiong, and X. Wang, "RF-Based Human Activity Recognition Using Signal Adapted Convolutional Neural Network," *IEEE Transactions on Mobile Computing*, vol. 22, no. 1, pp. 487–499, Jan. 2023.

[231] M. Zhao, T. Li, M. A. Alsheikh, Y. Tian, H. Zhao, A. Torralba, and D. Katabi, "Through-Wall Human Pose Estimation Using Radio Signals," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, Jun. 2018.

[232] T. Li, L. Fan, M. Zhao, Y. Liu, and D. Katabi, "Making the Invisible Visible: Action Recognition Through Walls and Occlusions," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2019.

[233] J. Geng, D. Huang, and F. De la Torre, "DensePose From WiFi," 2023.

[234] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, "Deep learning for sensor-based activity recognition: A survey," *Pattern Recognition Letters*, vol. 119, pp. 3–11, Mar. 2019.

[235] L. Ma, B. Milner, and D. Smith, "Acoustic Environment Classification," *ACM Trans. Speech Lang. Process.*, vol. 3, no. 2, p. 1–22, Jul. 2006.

[236] M. A. Qamhan, H. Altaheri, A. H. Meftah, G. Muhammad, and Y. A. Alotaibi, "Digital Audio Forensics: Microphone and Environment Classification Using Deep Learning," *IEEE Access*, vol. 9, pp. 62 719–62 733, 2021.

[237] T. Heittola, A. Mesaros, and T. Virtanen, "Audio context recognition using audio event histograms," in *European Signal Processing Conference*, 2010, pp. 1272–1276.

[238] W.-H. Choi, S.-I. Kim, M.-S. Keum, W. Han, H. Ko, and D. K. Han, "Acoustic and visual signal based context awareness system for mobile application," in *IEEE Intl. Conf. on Consumer Electronics (ICCE)*, 2011, pp. 627–628.

[239] X. Liang and G. Wang, "A Convolutional Neural Network for Transportation Mode Detection Based on Smartphone Platform," in *IEEE Intl, Conf, on Mobile Ad Hoc and Sensor Systems (MASS)*, 2017.

[240] L. Harms and O. Landsiedel, "MASTER: Long-Term Stable Routing and Scheduling in Low-Power Wireless Networks," in *2020 16th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, May 2020.

[241] P. T. Michalski, "Design and Evaluation of Deadline-Based Scheduling Algorithms for the Industrial Internet of Things," Bachelor's Thesis, Kiel University, 2020. [Online]. Available: https://www.ds.informatik.uni-kiel.de/en/teaching/bachelor-and-master-theses/completed-master-and-bachelor-theses/2020 Design and Evaluation of Deadline-Based Scheduling Algorithms for the Industrial Internet of Things Patrik Thomas Michalski.pdf

[242] V. Paskal, "Design and Evaluation of a Neighbor Data Collection and Schedule Distribution System for Centrally Scheduled TSCH," Master's Thesis, Kiel University, 2022. [Online]. Available: https://www.ds.informatik.uni-kiel.de/en/teaching/bachelor-and-master-theses/completed-master-and-bachelor-theses/2022_Master_Viktor_Paskal.pdf

[243] L. Harms and O. Landsiedel, "Opportunistic Routing and Synchronous Transmissions Meet TSCH," in *2021 IEEE 46th Conference on Local Computer Networks (LCN)*. IEEE, Oct. 2021.

[244] V. Poirot, L. Harms, H. Martens, and O. Landsiedel, "BlueSeer: AI-driven environment detection via BLE scans," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. ACM, Jul. 2022.

[245] Tensorflow, "TensorFlow Lite for Microcontrollers," 2023. [Online]. Available: https://www.tensorflow.org/lite/microcontrollers

[246] M. Pöhls, "AI-Driven Precise Location Fingerprinting from Bluetooth Low Energy Transmissions," Master's Thesis, Kiel University, 2022. [Online]. Available: https://www.ds.informatik.uni-kiel.de/en/teaching/bachelor-and-master-theses/completed-master-and-bachelor-theses/2022_Master_Mika_Poehls.pdf

[247] M. Becker, "Location Fingerprinting with Bluetooth Low Energy: A Supervised Learning Approach," Master's Thesis, Kiel University, 2023. [Online]. Available: https://www.ds.informatik.uni-kiel.de/en/teaching/bachelor-and-master-theses/completed-master-and-bachelor-theses/2023_Master_Marcel_Becker.pdf

[248] C. A. Boano, T. Voigt, C. Noda, K. Römer, and M. A. Zúñiga, "JamLab: Augmenting Sensornet Testbeds with Realistic and Controlled Interference Generation," in *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*, 2011.

[249] M. Schuß, C. A. Boano, M. Weber, M. Schulz, M. Hollick, and K. Römer, "JamLab-NG: Benchmarking Low-Power Wireless Protocols under Controllable and Repeatable Wi-Fi Interference." in *Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks (EWSN)*, 2019, pp. 83–94.

[250] C. Richter, "Design and Implementation of GPIO Sensing for Minimally Intrusive Tracing of Wireless Sensor Networks,"

Bachelor's Thesis, Kiel University, 2020. [Online]. Available: https://www.ds.informatik.uni-kiel.de/en/teaching/bachelor-and-master-theses/completed-master-and-bachelor-theses/2020 Design_and_Implementation_of_GPIO_Sensing_for_Minimally_Intrusive_Tracing_of_Wireless_Sensor_Networks Christian Richter.pdf

[251] L. Harms, C. Richter, and O. Landsiedel, "Grace: Low-Cost Time-Synchronized GPIO Tracing for IoT Testbeds," in *2022 18th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, may 2022.

[252] ——, "Grace: Low-Cost Time-Synchronized GPIO Tracing for IoT Testbeds," *Computer Networks*, vol. 228, p. 109746, Jun. 2023.

[253] Semtech Corporation, "SX1278," 2024. [Online]. Available: https://www.semtech.com/products/wireless-rf/lora-connect/sx1278

[254] L. Harms and O. Landsiedel, "TSCH Meets BLE: Routed Mesh Communication Over BLE," in *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*. IEEE, Jun. 2023.

[255] A. S. Tanenbaum and H. Bos, "Section 2.4.2: Scheduling in Batch Systems - Shortest Job First," in *Modern Operating Systems*, 4th ed. Pearson Education, Inc., 2015, pp. 157–158.

[256] "Contiki-NG: The OS for Next Generation IoT Devices," 2020. [Online]. Available: http://www.contiki-ng.org/

[257] C. A. Boano and M. Schuß, "EWSN 2019 Dependability Competition Logistics Information, rev. 1," Jan. 2018. [Online]. Available: https://iti-testbed.tugraz.at/fileupload/static/fileupload/EWSN2019_DC_Logistics_1.pdf

[258] L. Råde and B. Westergren, *Mathematics Handbook for Science and Engineering*. Lund: Studentlitteratur AB, 2004.

[259] J. Shi, M. Sha, and Z. Yang, "DiGS: Distributed Graph Routing and Scheduling for Industrial Wireless Sensor-Actuator Networks," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, Jul. 2018.

[260] T. Chang, T. Watteyne, X. Vilajosana, and P. H. Gomes, "Constructive Interference in 802.15.4: A Tutorial," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 217–237, 2018.

[261] P. H. Gomes, T. Watteyne, P. Gosh, and B. Krishnamachari, "Competition: Reliability through Timeslotted Channel Hopping and Flooding-Based Routing," in *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks (EWSN)*, 2016.

[262] F. Ben Abdesslem, A. Phillips, and T. Henderson, "Indoor distance estimated from Bluetooth Low Energy signal strength: Comparison of regression models," in *IEEE Sensors Applications Symposium (SAS)*, 2016, pp. 1–5.

[263] C. Falsi, D. Dardari, L. Mucchi, and M. Z. Win, "Time of arrival estimation for UWB localizers in realistic environments," *EURASIP J. on Advances in Signal Processing*, vol. 2006, pp. 1–13, 2006.

[264] P. Lazik, N. Rajagopal, O. Shih, B. Sinopoli, and A. Rowe, "ALPS: A Bluetooth and Ultrasound Platform for Mapping and Localization," in *ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, 2015, p. 73–84.

[265] F. Ben Abdesslem, A. Phillips, and T. Henderson, "Less is More: Energy-Efficient Mobile Sensing with Senseless," in *ACM Workshop on Networking, Systems, and Applications for Mobile Handhelds (MobiHeld)*, 2009, p. 61–62.

[266] Bluetooth SIG, "Bluetooth 2021 Market Update," 2021. [Online]. Available: https://www.bluetooth.com/wp-content/uploads/2021/01/2021-Bluetooth_Market_Update.pdf

[267] B. Etzlinger, B. Nußbaummüller, P. Peterseil, and K. A. Hummel, "Distance Estimation for BLE-based Contact Tracing – A Measurement Study," in *Wireless Days (WD)*, 2021.

[268] R. David, J. Duke, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang, P. Warden, and R. Rhodes, "TensorFlow Lite Micro: Embedded Machine Learning for TinyML Systems," in *Machine Learning and Systems (MLSys)*, vol. 3, 2021, pp. 800–811.

[269] Zephyr Project, "Zephyr Real-Time Operating System," 2016.

[270] Y. Zhan and T. Kuroda, "Wearable sensor-based human activity recognition from environmental background sounds," *J. of Ambient Intelligence and Humanized Computing*, vol. 5, no. 1, pp. 77–89, 2014.

[271] N. Kern, B. Schiele, and A. Schmidt, "Multi-sensor Activity Context Detection for Wearable Computing," in *Ambient Intelligence*. Springer, 2003, pp. 220–232.

[272] K. Sankaran, M. Zhu, X. F. Guo, A. L. Ananda, M. C. Chan, and L.-S. Peh, "Using Mobile Phone Barometer for Low-Power Transportation Context Detection," in *ACM Conf. on Embedded Network Sensor Systems (SenSys)*, 2014, p. 191–205.

[273] J. Yang, H. Zou, H. Jiang, and L. Xie, "Device-Free Occupant Activity Sensing Using WiFi-Enabled IoT Devices for Smart Homes," *IEEE Internet of Things J.*, vol. 5, no. 5, 2018.

[274] B. Kempke, P. Pannuto, B. Campbell, and P. Dutta, "SurePoint: Exploiting Ultra Wideband Flooding and Diversity to Provide Robust, Scalable, High-Fidelity Indoor Localization," in *ACM Conf. on Embedded Network Sensor Systems (SenSys)*, 2016, p. 137–149.

[275] Y. Ma, N. Selby, and F. Adib, "Minding the Billions: Ultra-Wideband Localization for Deployed RFID Tags," in *Intl. Conf. on Mobile Computing and Networking (MobiCom)*, 2017, p. 248–260.

[276] A. von See, "Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2030," 2021. [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[277] "BabbleSim - A physical layer simulator." [Online]. Available: https://babblesim.github.io/

[278] "Publications and Standards from the National Marine Electronics Association (NMEA) / NMEA 0183," Tech. Rep., Nov. 2008. [Online]. Available: https://web.archive.org/web/20131021183159/http://www.nmea.org/content/nmea_standards/nmea_0183_v_410.asp

[279] G. Baddeley, "GPS - NMEA sentence information," 2001. [Online]. Available: http://aprs.gids.nl/nmea/#rmc

[280] Electronic Notes, "What is a Logic Analyzer." [Online]. Available: https://www.electronics-notes.com/articles/test-methods/logic-analyzer/basics-tutorial.php

[281] V. Handziski, A. Köpke, A. Willig, and A. Wolisz, "TWIST: a scalable and reconfigurable testbed for wireless indoor experiments with sensor networks," in *Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality.* ACM, may 2006.

[282] E. Ertin, M. Nesterenko, A. Arora, R. Ramnath, V. Naik, S. Bapat, V. Kulathumani, M. Sridharan, H. Zhang, and H. Cao, "Kansei: A Testbed for Sensing at Scale," in *Proceedings of the fifth international conference on Information processing in sensor networks - IPSN '06.* ACM Press, 2006.

[283] L. Sanchez, J. A. Galache, V. Gutierrez, J. M. Hernandez, J. Bernat, A. Gluhak, and T. Garcia, "Smartsantander: The meeting point between future internet research and experimentation and the smart cities," in *2011 Future Network & Mobile Summit.* IEEE, 2011, pp. 1–8.

[284] Wikipedia contributors, "Itu region — Wikipedia, the free encyclopedia," 2022, [Online; accessed 1-October-2022]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=ITU_Region&oldid=1082115274

[285] ——, "Unix time — Wikipedia, the free encyclopedia," 2022, [Online; accessed 1-October-2022]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Unix_time&oldid=1112784820

[286] *CC1101 - Low-Power Sub-1 GHz RF Transceiver*, Texas Instruments. [Online]. Available: https://www.ti.com/lit/ds/symlink/cc1101.pdf

[287] Adafruit, "Ultimate GPS Module - 66 channel w/10 Hz updates - PA1616S - MTK3339 Chipset." [Online]. Available: https://www.adafruit.com/product/790

[288] *EZ-USB® Technical Reference Manual*, Cypress Semiconductor, Cypress Semiconductor, 198 Champion Court, San Jose, CA 95134-1709, 2019, 001-13670 Rev. *G. [Online]. Available: https://www.infineon.com/dgdl/Infineon-EZ-USB_TECHNICAL_REFERENCE_MANUAL-AdditionalTechnicalInformation-v08_00-EN.pdf

[289] "Saleae Logic Pro 8 USB Logic Analyzer," 2023. [Online]. Available: https://downloads.saleae.com/specs/Logic Pro 8 Product Fact Sheet.pdf

[290] Saleae Support, "Time Measurement Error," 2021. [Online]. Available: https://support.saleae.com/faq/technical-faq/time-measurement-error

[291] Nordic Semiconductor, "nRF52840 DK." [Online]. Available: https://www.nordicsemi.com/Products/Development-hardware/nrf52840-dk

[292] B. Al Nahas, S. Duquennoy, and O. Landsiedel, "Concurrent Transmissions for Multi-Hop Bluetooth 5," in *Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks (EWSN)*, 2019, pp. 130–141.