Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Wireless protocol testing in embedded environment

Master's Thesis

Author
Dániel Ábrahám

Advisor
dr. András Vörös

December 17, 2023

Contents

Kivonat

Az Internet of Things (IoT) eszközök gyors elterjedése új korszakot nyitott az összekapcsolt rendszerekben, forradalmasítva a környezetünkkel való interakcióinkat és a környezetünk észlelését egyaránt. Diplomamunkámban kiemelt hangsúlyt fektetek az IoT-protokoll rendszerek összetett felépítésének és működésének vizsgálatába - különös tekintettel a Bluetooth-ba, a Zigbee-be, valamint a Thread-be -, emellett foglalkozom a rendszerek átjárhatóságával a robusztus IoT-tesztelés módszertanát érintő kritikus kihívásokkal.

A kutatás középpontjában az IoT-eszközök meglévő tesztelési technikáinak értékelése áll, figyelembe véve a Bluetooth, a Zigbee és a Thread által támasztott egyedi követelményeket. Megvizsgálom a különböző tesztelési stratégiák hatékonyságát - beleértve a funkcionális, teljesítmény- és biztonsági tesztelést -, annak érdekében, hogy feltárjam a különböző, az IoT-környezetek által támasztott kihívásokat.

A diplomamunkámban továbbá feltárom az IoT-tesztelés újonnan megjelenő módszereit, és javaslatot teszek egy egységes tesztelési keretrendszer létrehozására, amely alkalmazkodik a Bluetooth, a Zigbee és a Thread jellegzetes tulajdonságaihoz. Ezen protokollok egyetlen összefüggő tesztelési architektúrán belüli összehangolásával a kutatásom célja az IoT-ökoszisztémák átjárhatóságának javítása, elősegítve a megbízhatóbb és biztonságosabb összekapcsolt környezet létrehozását.

Abstract

The rapid proliferation of Internet of Things (IoT) devices has ushered in a new era of interconnected systems, revolutionizing the way we interact with and perceive our surroundings. This master's thesis delves into the intricacies of IoT protocols, with a specific focus on Bluetooth, Zigbee, and Thread, addressing the critical challenges associated with their interoperability and the methodologies employed for robust IoT testing.

The core of the research revolves around the evaluation of existing testing techniques for IoT devices, considering the unique requirements posed by Bluetooth, Zigbee, and Thread. Comprehensive testing strategies, including functional, performance, and security testing, are scrutinized to uncover their efficacy in addressing the challenges posed by diverse IoT environments.

Furthermore, the thesis explores the emerging methodologies for IoT testing and proposes a unified testing framework that accommodates the distinctive features of Bluetooth, Zigbee, and Thread. By aligning these protocols within a cohesive testing architecture, the research aims to enhance the overall interoperability of IoT ecosystems, fostering a more reliable and secure interconnected landscape.

Chapter 1

Introduction

1.1 Problem definition

With the rapid spread of internet-of-things (IoT) devices worldwide, finding defects in protocol implementations has become more critical. Trying to define test cases or suites that truly move all the inner workings of a device is a delicate balance of complexity and resources. More automated solutions are necessary to help QA engineers and testers focus on solving problems and exotic cases.

In the last decade, the IoT market experienced a massive boom in revenue. According to Statista[56], from 2020 to 2023, revenue increased by 61 percent. The market is expected to grow to $600 billion by the end of the current decade.

IoT is primarily used in large-scale sensor networks, making it suitable for Smart Cities, Utilities (energy management), and Smart Homes[2]. Healthcare remote monitoring is expected to be another market where IoT could become a significant technology. The automotive industry and agriculture[21, 22] (with the spread of precision agriculture) will also be great users.

According to Makhashari et al. [40], over 80 percent of developers face bugs at the device or the communication level. Out of these bugs, over 40 percent happen to be severe. They introduced taxonomy for bugs in IoT. For this thesis, the two main categories are device and connection bugs. Device bugs cover hardware and firmware issues. Connection bugs cover communication issues between multiple devices.

Previously, some studies discussed testing in the IoT realm. From functional testing[17, 52] through pattern-based techniques[49] to AI guided techniques[66, 24]. One of the studies discusses model-based techniques used in IoT testing[50]. There are two notable mentions of test ware setup in the literature [5, 6].

Most vendors' flagship chips can handle two or more protocols simultaneously. For this problem definition, I want to create a solution for testing a triple protocol setup that can run BLE, Thread, and Zigbee side-by-side.

Testing should cover all network-level relations but does not have to validate full conformance. The tested application has the capability of a BLE peripheral device, a Minimal Thread Device, and a Zigbee End Device.

The foundation of this idea is that many IoT devices on the market can be configured or initially set via Bluetooth by the user and then work on a different network. Smart lightbulbs are great examples for this use case as they have a constant power supply. Both Bluetooth and Zigbee have defined a light control profile, but there are quite a few Thread-compatible smart lights and controllers.

The tested application would be able to act as a lightbulb on every network that it supports and does not require the user to dedicate him-/herself to invest in another network. The users can use whatever existing environment they have.

The solution should only depend on open-source solutions, as investing in tooling was not an option. Also, since Python is emerging as a future-proof and popular language, I chose it to be the language of the main codebase.

For the implementation of this study, Silicon Labs provided me with some wireless hardware[55]. I was given four of their EFR32xG24 Pro kits. It consists of a radio board - where the MCU is located - and a wireless Pro mainboard that can communicate with the radio board.

The mainboard can connect to the outside world via USB or a network connection via ethernet. A user can attach a remote shell via telnet if the board connects to a network. One can pass commands to the radio board on the remote prompt if the application has integrated CLI features.

Furthermore, Silicon Labs provided the necessary applications to run the scenario. The Silicon Labs' Gecko Software Development Kit (GSDK) is their integrated software solution. It is open-source and published on Git Hub[54]. I was given a slightly modified version of the company's sample applications. All the SoC applications have a CLI interface compiled into the binary.

During this thesis, I will create an abstract model to represent a valid IoT use case. Then, I present a solution to generate abstract test cases and eventually attach the test environment to the embedded system running in real time. The abstraction layer will be connected to the test environment with an adaptor codebase. The adaptor will manage the test devices and handle the communication between the test executor and thee system under test.

1.2 Thesis Outline

This thesis has two parts: a literature review and a contribution. The literature review will consist of Chapter 2.

In Section 2.1, Wireless protocols will be shown. Section 2.2 will introduce the concept of multiprotocol networking, and Section 2.3 will introduce some software and system testing techniques.

My contributions will be described in Chapter 3. After the methodology, Chapter 4 summarizes the results evaluated from testing. Lastly, Chapter 5 summarises the thesis work and provides some future work.

Chapter 2

Background

2.1 Wireless protocols

In this chapter, I will describe and introduce some of the of protocols used in IoT. I will demonstrate each wireless protocol starting with Bluetooth Low Energy in Section 2.1.1, Zigbee in Section 2.1.3, and finally Thread/OpenThread in Section 2.1.4. Section 2.1.2 will provide an overview of the IEEE 802.15.4 radio specification, of which both Zigbee and Thread is based on.

2.1.1 Bluetooth Low Energy

Bluetooth is a low-power, short-range wireless technology developed initially for replacing cables when connecting devices like mobile phones, headsets, and computers. It has since evolved into a wireless standard for connecting electronic devices to form Personal Area Networks (PANs) and ad hoc networks. [20]

Bluetooth is one of the most popular commodity radios for wireless devices. As a representative of the frequency hopping spread spectrum radios, it is a natural alternative to broadcast radios in the context of sensor networks. [39]

The Bluetooth SIG specifies three (3) different types of devices, hence three specifications. Bluetooth Classic consists of Basic Rate and Enhanced Data Rate capable devices. On the other hand, Bluetooth Low Energy needs Low Energy capabilities.

Since BLE and Bluetooth Classic has different controller layer, they are not compatible with each other. This thesis will focus mainly on Bluetooth Low Energy (BLE) due to its similarity in functionality and utilization to Zigbee and Thread-/OpenThread.

The Bluetooth Special Interest Group (SIG) oversees the Bluetooth standard and has a membership of over 38,000 companies as of 2023. [10] The initial specification was made available in 1999. Since then, the SIG released five core specifications, 5.4 being the latest adoption in 2023. [12]

2.1.1.1 Introcution

The Bluetooth 4.0 Core Specification introduced Bluetooth Low Energy (BLE). It is tempting to represent itself as a smaller and highly optimized version of its bigger brother, classic Bluetooth. In reality, BLE has an entirely different lineage and design goals. Initially known as Wibree and created by Nokia, the Bluetooth SIG ultimately embraced the technology. The focus was to develop a radio standard with the lowest possible power consumption, specifically optimized for low cost, low bandwidth, low power, and low complexity. [58]

BLE uses frequency hopped spread spectrum (FHSS), just like Bluetooth Classic, to limitate transmission errors in the radio band. The physical layer uses a 2 MHz bandwidth. From Core Specification 5, BLE has a theoretical maximum throughput of 2 Mbps and a maximum output power of 100 mW (20 dBm).

2.1.1.2 Architecture

According to Townsend et al., the Bluetooth Low Energy device can be categorized into three primary blocks: controller, host, and application. Each block has one or more layers, as shown in Figure 2.1.

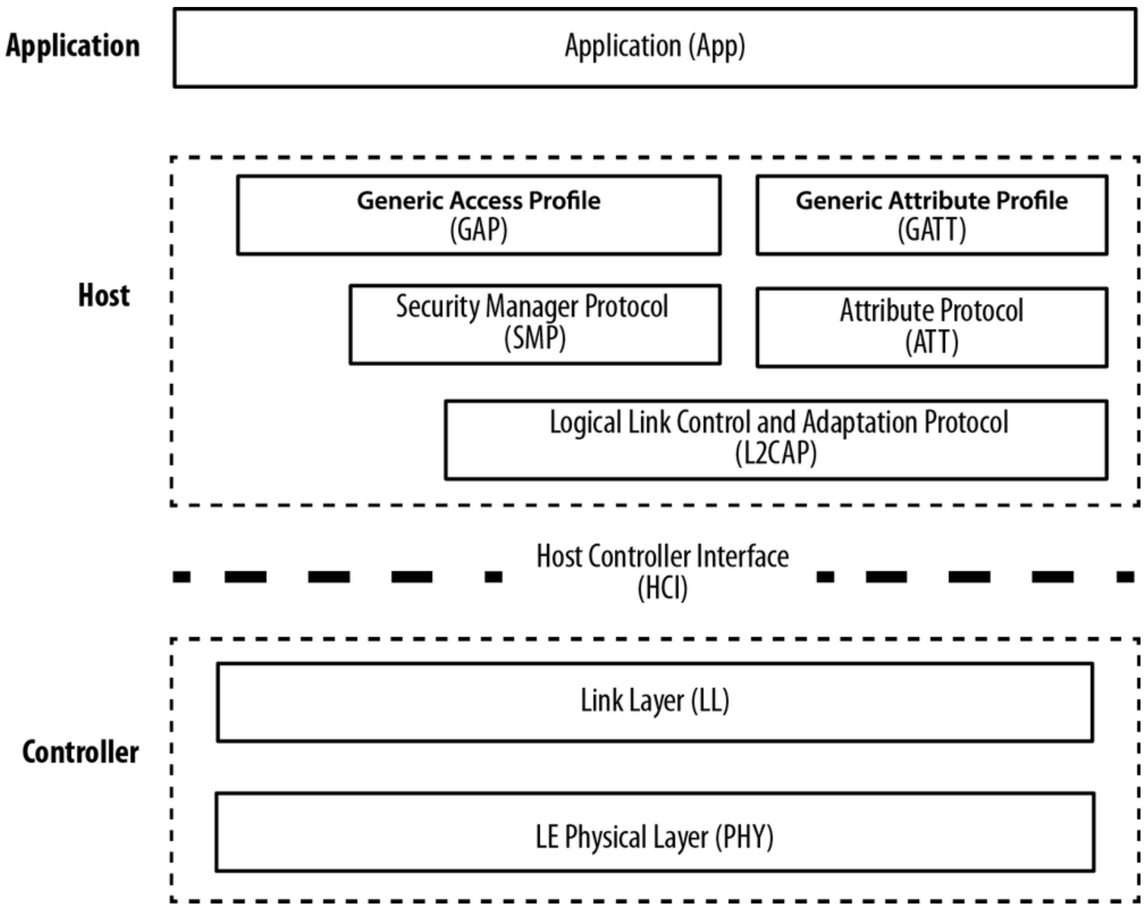
Application
Application (App)
Generic Access Profile (GAP)
Generic Attribute Profile (GATT)
Host
Security Manager Protocol (SMP)
Attribute Protocol (ATT)
Logical Link Control and Adaptation Protocol (L2CAP)
Host Controller Interface (HCI)
Link Layer (LL)
Controller
LE Physical Layer (PHY)

Figure 2.1: The BLE protocol stack. (Source: Figure 2-1 [58])

The layers defined in Figure 2.1 provide the required functionality to operate.

The Controller contains the Physical Layer (PHY) and the Link Layer (LL). These hold all the responsibility for operating the physical radio channel. Also, these layers are specialized by the LE standard keeping the Host layer more-or-less the same as Bluetooth Classic.

A device address and an address type identify a device. An address type indicates that an address can be either public or random. Both are 48 bits long. A device shall use at least one type of device address but can use both simultaneously.

In BLE, Generic Access Profile (GAP) defines specific roles for each device. These roles are: broadcaster, observer, peripheral, and central. A device may support multiple roles but only one at a given time. Each role specifies the requirements for the underlying Controller. Section 2.1.1.5 explains GAP in more detail.

2.1.1.3 LE Controller

The physical layer (PHY) contains all necessary analog communication circuitry. The radio uses the 2.4 GHz ISM band to communicate. This band is divided into 40 channels or subbands from 2.400 GHz to 2.4835 GHz, each subband using 2 MHz of bandwidth. (Yin et al., 2019) This division can be seen in Figure 2.2. There are three dedicated advertising subbands for connection setup and broadcast messages; the remaining 37 are known as general-purpose channels.

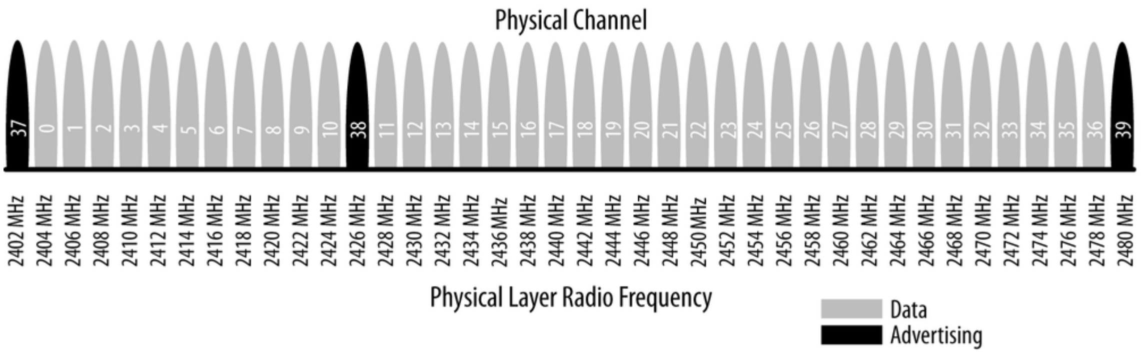

Figure 2.2: Channel allocation. (Source: Figure 2-2 [58])

The transmitter can use a maximum power of 100 mW. The LE Power Control Request can manage the power settings if a device supports this feature.

J. Yin et al. [65] showed that from Core Specification 5, a novel modulation scheme was introduced in the standard. This feature allowed radio modules to use up all 2 MHz of bandwidth, thus significantly increasing the maximum throughput.

Part B of the Core Specification's volume six (Low Energy Controller volume) defines the Bluetooth Low Energy Link Layer's specification.

The Link Layer is the component that directly communicates with the PHY. It is typically a blend of custom hardware and software solutions. [58]

The hardware part usually implements all computationally intensive tasks, such as CRC generation/validation, random number generation, and AES encryption.

Multiple State and Role		Advertising	Scanning	Initiating	Connection Master Role	Slave Role
Advertising		Prohibited	Allowed	Allowed	Allowed	Allowed
Scanning		Allowed	Prohibited	Allowed	Allowed	Allowed
Initiating		Allowed	Allowed	Prohibited	Allowed	Prohibited
Connection	Master Role	Allowed	Allowed	Allowed	Allowed	Prohibited
	Slave Role	Allowed	Allowed	Prohibited	Prohibited	Prohibited

Table 2.1: Link Layer State Machine limitations (obsolate)

The software part of this layer is the only hard real-time part of the whole stack because it has to match the radio hardware's timings based on the PHY layer's specifications.

The Link Layer can be thought of as a state machine in operation. In Figure 3, the representation of this state machine can be seen. The Link Layer may have multiple instances of this state machine but shall have at least one supporting Advertising or Scanning.[8]

The Core Specification 5.2 introduced two (2) new states: Synchronization and Isochronous Broadcasting.

The Air Interface protocol defines the inner working of each state. This interface consists of multiple access schemes, device discovery, and Link Layer connection methods.

In the case of multiple state-machine instances, the first version of the specification ([8]) prohibited certain combinations of states (and roles). Table 2.1 specifies the allowed and prohibited state machine combinations. From the release of Core Specification 5.2, an implementation may support any combination of states and roles. The only criterion is that any two (2) devices shall not have more than one connection between them. [11]

After the introduction in Core Specification 5, there are two algorithms for channel selection. The initial algorithm (called Channel Selection Algorithm #1) only supports channel selection for connection events. Algorithm #2 can be used for connection and periodic advertising events. J. Yin et al. showed that this update significantly enhanced the robustness of the protocol. [65]

The Link Layer Control protocol (LLCP) is used to control and negotiate aspects of the operation of a connection between two Link Layers. This includes procedures for control of the connection, starting and pausing encryption, and other link procedures. [8]

One of the most significant procedures is the Channel Map Update Procedure. This ensures that upon a physical channel update (initiated by frequency hopping), both the master and the slave stay in sync with the physical channel index.

BLE devices communicate via air interface packets. The specification differentiates two different formats; uncoded and coded packets. Figure 2.3 shows the differences between the formats. Higher-level protocols are transmitted in the PDU field. As

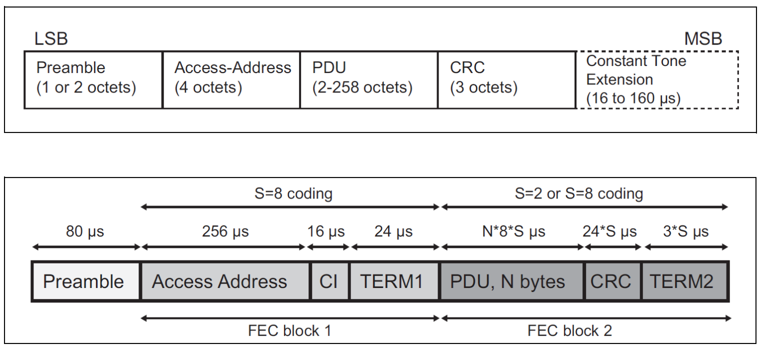

Figure 2.3: AIP formats. (Source: [11])

Figure 2.3 shows, the largest data transmitted in a single packet is limited to 258 bytes.

2.1.1.4 Host Controller Interface

The Host Controller Interface (HCI) connects the Controller (low-level) and Host (high-level) parts of the BLE stack. It contains a technology-independent logical command interface structure (HCI driver) and a Host Controller Transport Layer. The HCI provides a uniform interface for accessing a Bluetooth Controller's capability.

According to the standard, separating was necessary to keep the HCI driver independent of the underlying transport technology. Currently, in the latest standard (Core Specification 5.4), there are four (4) different technologies defined for the transport layer: UART, USB, Secure Digital (SD), and Three-wire UART.

The HCI uses a command-response communication format. The Host sends commands to the Controller, and the Controller responds in the form of events. If the Controller cannot execute the command immediately, it shall notify the Host that it received it.

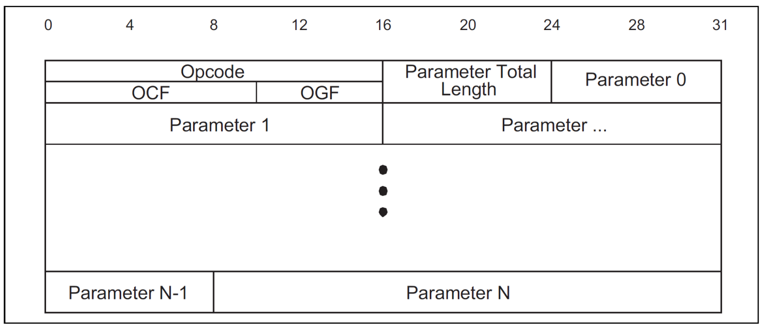

Figure 2.4: HCI command message structure. (Source: [9])

Figure 2.4 and Figure 2.5 show the data formats used in HCI communication.

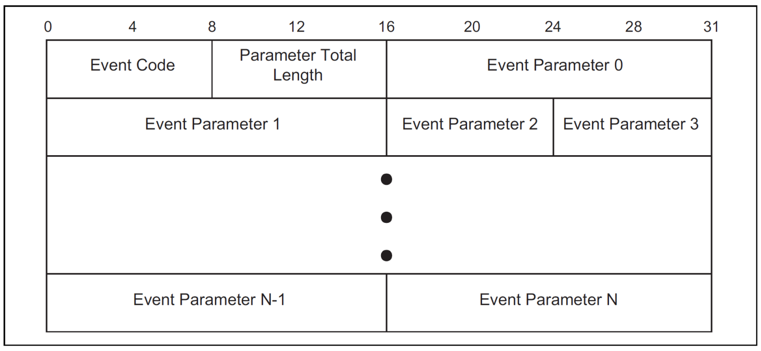

Figure 2.5: HCI command message structure. (Source: [9])

2.1.1.5 Host

The Bluetooth logical link control and adaptation protocol (L2CAP) supports higher-level protocol multiplexing, packet segmentation and reassembly, and the conveying of quality of service information.

Packet segmentation is necessary because data must fit into 27-byte maximum payload size of the BLE packets on the transmit side. [58] On the reception path, this procedure has to be reversed.

The Bluetooth logical link control and adaptation protocol (L2CAP) supports higher-level protocol multiplexing, packet segmentation and reassembly, and the conveying of quality of service information. The protocol state machine, packet format, and composition are described in this document. [58]

The Attribute Protocol (ATT) defines the peer-to-peer communication scheme. It defines client and server roles during communication. The ATT client communicates to an ATT server on a remote device. This communication uses a dedicated fixed L2CAP channel created by the initial connection.

The client can send commands, requests, and confirmations to a server. On the reversed path, the server can only send responses, notifications, and indications back to a client, meaning that a client can read and write the values of attributes on a device with an ATT server.

The Generic Access Profile (GAP) defines common functionalities all Bluetooth devices use. These functionalities, such as modes and access procedures, are used by transports, protocols, and application profiles. The GAP includes device discovery, connection modes, security, authentication, association models, and service discovery.

The Generic Attribute Profile (GATT) builds on the Attribute Protocol (ATT) and adds a hierarchy and data abstraction model on top of it. It can be considered the backbone of BLE data transfer because it defines how data is organized and exchanged between applications.

The GATT also provides an interface for discovering, reading, writing, and indicating service characteristics and attributes.

The Bluetooth SIG specifies over 250 standard profiles that must be supported to facilitate interoperability between developers. [13]

2.1.2 IEEE 802.15.4

The IEEE 802.15.4 standard defines the physical layer (PHY) and medium access control (MAC) sublayer specifications for low-rate wireless personal area networks (LR-WPANs). [33] LR-WPANs are designed to support low-power communication between devices with limited resources and short-range communication requirements.

The standard provides a framework for building wireless networks with low data rates, low power consumption, and low cost. It operates in the unlicensed Industrial, Scientific, and Medical (ISM) bands, such as the 2.4 GHz band used by Wi-Fi and Bluetooth devices, as well as the 868 MHz and 915 MHz bands. It supports a maximum data rate of 250 kbps.

2.1.2.1 Medium Access Control Layer (MAC)

The MAC layer describes essential message handling and congestion control. This layer includes mechanisms for forming and joining a network, a CSMA-CA mechanism, and a link layer for retires and acknowledgment for reliable communication [37].

For device identification, the layer uses 16-bit short addresses and 64-bit extended addresses. Devices can be assigned a unique short address by the network coordinator to enable efficient addressing in the network.

In MAC, two roles are defined: the network coordinator and the device. The network coordinator initiates and manages the network, while devices can join the network and communicate with the coordinator or other devices.

2.1.3 Zigbee

2.1.3.1 Introduction

Zigbee is a wireless communication protocol specifically designed for applications requiring low-power consumption and short-range communication. It operates on the IEEE 802.15.4 standard and provides a reliable and efficient way for devices to connect and communicate with each other. Zigbee is a frequently used technology in home automation, industrial control systems, and various other applications related to the Internet of Things (IoT). It supports mesh networking, allowing devices to relay data to extend the network coverage. Zigbee is known for its low power consumption, simple implementation, and ability to support a large number of devices in a network.

The Zigbee Alliance created the first version of the specification back in 2004. In 2022, the alliance renamed itself to Connectivity Standards Alliance. In 2023, more than 400 companies are actively involved in the alliance. [18]

The first Zigbee specification was released in 2004. There have been three iterations of the specification, namely Zigbee-2004 (now deprecated), Zigbee-2006, and Zigbee-2007. In 2023, the latest release is Zigbee-2007 Revision 23.

The Zigbee-2007 specification introduced a new stack called ZigBee Pro. The main difference is in the network layer. The ZigBee stack is formed around a central coordinator and uses tree addressing to establish a network. Meanwhile, the Pro version uses stochastic addressing to avoid limitations with a tree. [37]

The Pro stack implements new features in the routing protocol, such as network-level multicast, many-to-one, and source routing. Other features are fragmentation, asymmetric link handling, frequency agility, PAN ID conflict resolution, and multiple security levels. Silicon Labs states that using a Pro network improves stability and reliability.

Both the Pro and the original stack can interoperate but with limitations. If a Pro coordinator forms a network, the network can only contain Pro routers. Any stack can act as an end device. [37]

This thesis will be based on ZigBee Pro Revision 21. This version of the specification is also called ZigBee 3.0.

ZigBee 3.0 contains a new document called Base Device Behavior Specification (BDB). It possesses mandatory and optional application layer behavior for any ZigBee product. [29], [41]

2.1.3.2 Network

A Zigbee network is made up of three different types of nodes: Coordinator, Router, and End Device. [63]

A Zigbee network can be configured to support different topologies: star, tree, and mesh. Figure 2.6 shows the difference between the topologies.

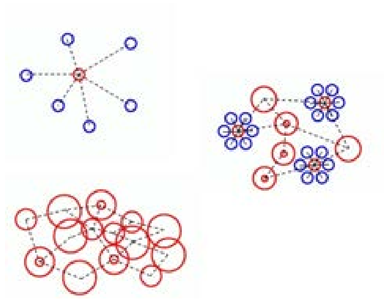

Figure 2.6: Star, Full Mesh, and Hybrid Mesh topologies. Source([37])

2.1.3.3 Architecture

Zigbee builds on top of the IEEE 802.15.4 physical layer defined in Section 3. The Zigbee architecture defines two more layers: the network layer (NWK) and the application layer (APL).

2.1.3.4 Network layer

The NWK layer is responsible for mesh networking, which includes broadcasting packets across the network, determining routes for unicasting packets, and ensuring packets are sent reliably from one node to another. The network layer also has a set of commands for security purposes, including secure joining and rejoining. ZigBee networks are all secured at the NWK layer, and the entire payload of the NWK frame is encrypted

2.1.3.5 Application layer

The ZigBee application layer consists of the APS sublayer, the ZDO, and the defined by manufacturer application objects. (Wang et al., 2016) [15]

The application support sub-layer (APS) provides an interface between the NWK and the application layer through services for ZDO and application objects.

The APS can be divided into two services: data service and management service. The data service is APS Data Entity (APSDE), and the management service is APS Management Entity (APSME).

The APSDE is responsible for payload conversion, address filtering, reliability, and fragmentation. Once a connection is built up, it is the transport layer between two or more devices.

The other service, the APSME, allows an application to interact with the stack. It is responsible for AIB, binding, security, and group management.

The application Framework provides a hosting environment for application objects on a Zigbee device. A developer can define up to 240 distinct application objects, each identified by an endpoint address from 1 to 240.

Index 0 is reserved for communication with ZDO, and index 255 is considered the broadcast address. For future use, the alliance reserved the index range from 241 to 254.

Zigbee device object interfaces the application objects, the device profile, and the APS. For the application objects, it provides device control and network functions. Other functions are address management of the device, discovery, binding, and security. (Wang et al., 2016)

The ZDO is responsible for initializing the APS sublayer, the NWK layer, and the Security Service Provider. Also, the ZDO defines the role of the device within a ZigBee network.

2.1.4 Thread

2.1.4.1 Introduction

Thread is intended to be an open standard, and the protocol is built by incorporating many current and proposed standards.[59] Anyone can download the complete specification from the Thread Group website. [27]

As mentioned, the Thread Group develops and maintains the protocol. Thread Group is a not-for-profit organization formed by seven companies: ARM (Softbank), Big Ass Fans, Freescale (NXP), Nest Labs (Google), Samsung, Silicon Labs, and Yale Locks. More than 200 companies, such as Apple, Google, and Amazon, are forming the Thread Group. [28]

This thesis uses the Thread specification, v1.3.0 2023.

2.1.4.2 Private Area Network

In a Thread network, there are two types of devices (nodes): Full Thread Device (FTD) and Minimal Thread Device (MTD). FTDs can communicate with each other and with their attached MTD children. MTDs can only communicate with the parent FTD.

A Thread Device can act in six different roles.

1. Active Router
2. Router.Eligible End Device (REED)
3. Full End Device (FED)
4. Minimal End Device (MED)
5. Sleepy End Device (SED)
6. Synchronized Sleepy End Device (SEED)

In order to create a network, the initial node must select an IEEE 802.15.4 channel and a PAN Identifier (PAN ID). The forming node becomes the PAN's first Router and selects a Rouder ID for itself.

The topology is based on the number of Routers in the Thread Network. If there is only one Router, then a basic star topology with a single Router is formed. A mesh topology is automatically formed if there is more than one Router. Figure 2.7 shows a basic topology.

The Mesh Commissioning Protocol (MshCoP) manages how a new device can join a Thread Network. A commissioner device is an elected authentication server. This device can be a cell phone, a cloud service, or an FTD. There can only be one commissioner node in a PAN, and the Thread Management Protocol manages it.

A new device that wants to join the network is called Joiner. A Joiner Router is a node that can act as a Router and is one hop away from the Joiner.

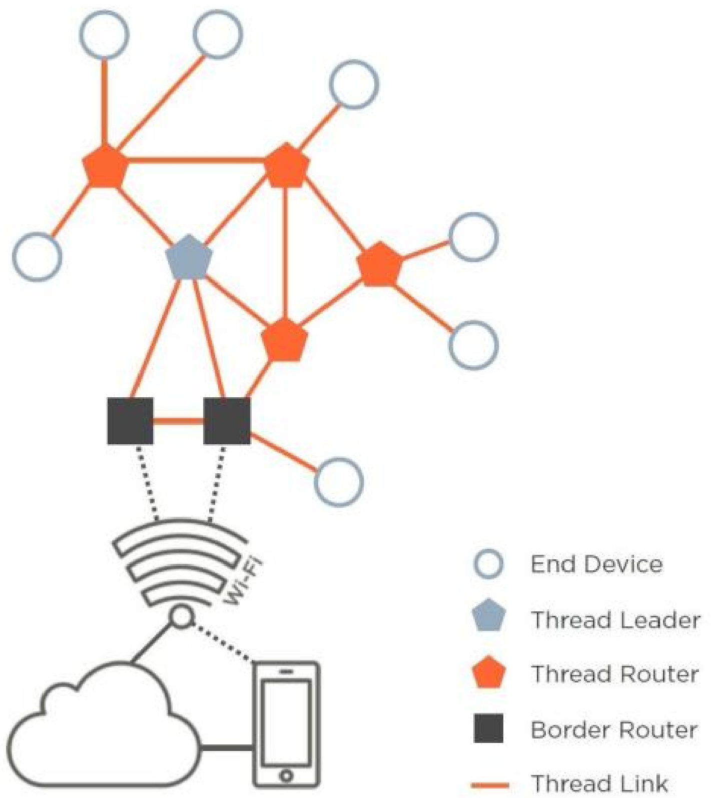

Figure 2.7: Channel allocation. (Source: Figure 3 [26])

There are two main approaches to commissioning: *external* if the commissioner is connected to the Thread Network via a Border Router or *native* if the commissioner is in the Thread Network. Different approaches are shown in Figure 2.8.

State machines can define the behavior of the Joiner (see Figure 2.9), the Joiner Router (see Figure 2.10), and the Commissioner (see **??**).

2.2 Multiprotocol

In this chapter I will define the multi-protocol networking in IoT.

Section 2.2.1 will give a brief overview of the technology. Section 2.2.2 will showcase some configurations and Section 2.2.3 will introduce some implementations that are available.

2.2.1 Overview

The IoT market continues to grow, developers and manufacturers need more integrated options for wireless connectivity[45]. With multiprotocol-enabled wireless solutions reduce design and development costs and open the path for IoT convergence where applications can leverage the best aspects of multiple connectivity standards.

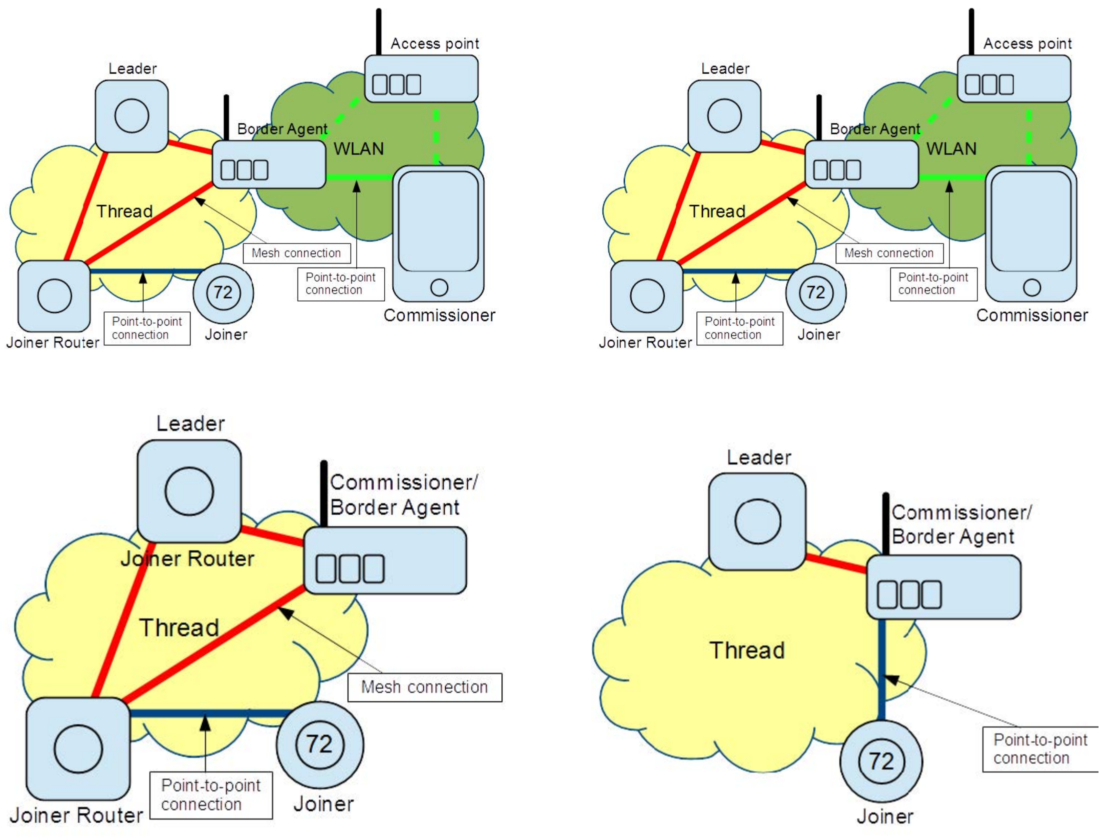

Figure 2.8: External and Nativ commissioning. (Source: [27])

As the current generation of chips on the market, a multiprotocol implementation usually contains some kind of operating system or a simple scheduler to manage concurrency between the protocols.

The industry specifies different types of multiprotocols. In a simple switched multiprotocol, the device's bootloader can switch between two or more protocols. This method does not require an operating system, as the switching happens on a predefined state (i.e. after device configuration). In switched mode, only one protocol can be loaded at any given time during operation. Figure 2.11 shows the operation of switched mode.

The other main type is concurrent multiprotocol. This mode allows the device to run multiple protocols on a single radio (see: Figure 2.12). The primary features of this type of solution are:

1. Switch between applications
2. Manage and configure the physical layer
3. Protocol prioritization
4. Real-time scheduling

These features usually call for an operating system. Available from the market, manufacturers do not create this operating system from the ground up. Rather they create a specialized Real-Time Operating System (RTOS).

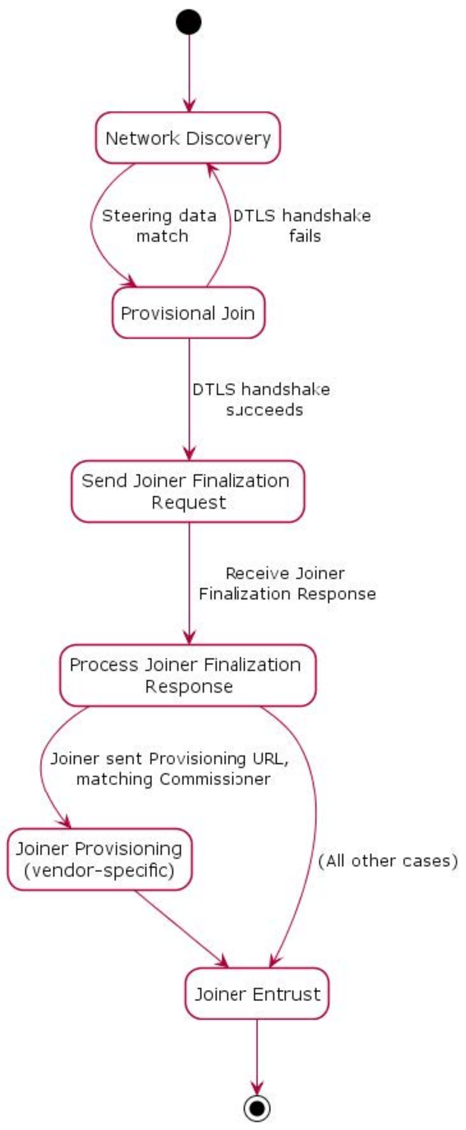
Network Discovery
Steering data match
DTLS handshake fails
Provisional Join
DTLS handshake succeeds
Send Joiner Finalization Request
Receive Joiner Finalization Response
Process Joiner Finalization Response
Joiner sent Provisioning URL, matching Commissioner
Joiner Provisioning (vendor-specific)
(All other cases)
Joiner Entrust

Figure 2.9: The Joiner's state machine (Source: [27])

Some manufacturers offer Linux-based solutions as well as embedded OS. With more powerful and capable embedded computers emerging on the market, devices can run more general operating systems such as Linux. These devices have the capacity to run multiple protocols alongside the OS and an additional IP stack as well. Devices suitable for this workload are being used as gateways or translators between protocols in the IoT realm.

2.2.2 Device configuration

In this section I will mention some device configurations. All stacks have a clear separation of what each layer does. This creates a possibility of different manufacturers can implement different layers.

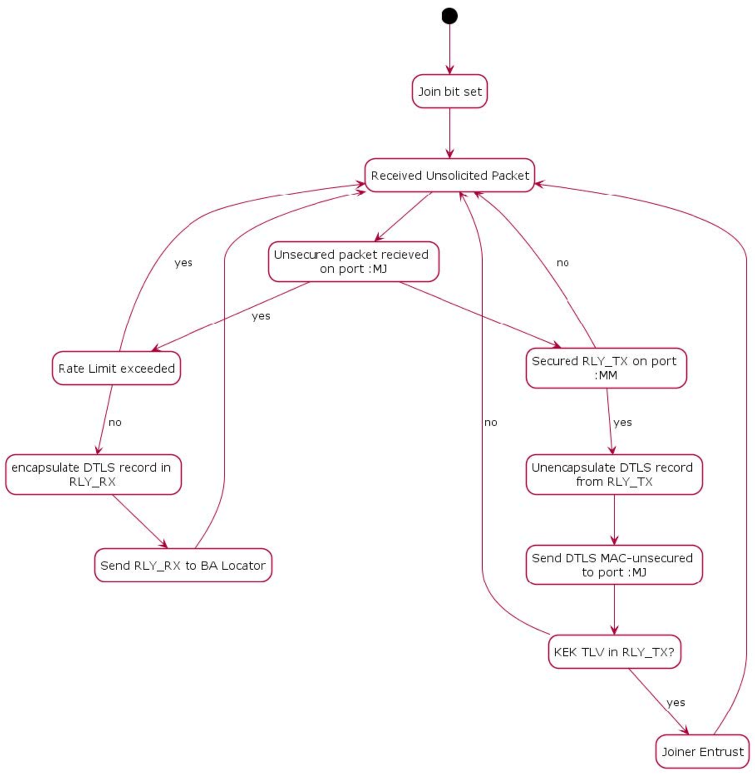
Join bit set
Received Unsolicited Packet
Unsecured packet recieved on port :MJ
yes
yes
no
Rate Limit exceeded
Secured RLY_TX on port :MM
no
no
yes
encapsulate DTLS record in RLY_RX
Unencapsulate DTLS record from RLY_TX
Send RLY_RX to BA Locator
Send DTLS MAC-unsecured to port :MJ
KEK TLV in RLY_TX?
yes
Joiner Entrust

Figure 2.10: The Joiner Router's state machine (Source: [27])

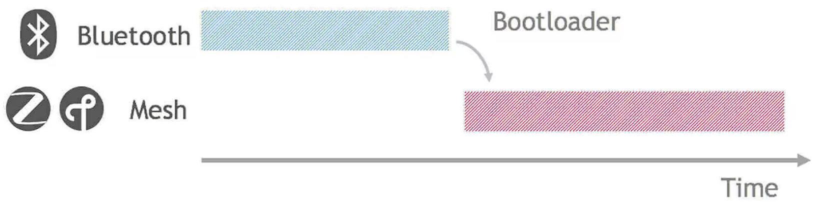

Figure 2.11: Switched multiprotocol. Source: [36]

Most manufacturers extend beyond their System-on-Chip (SoC) design and allow more interfaces so their network device to improve reusability. Figure 2.13 shows different approaches used in the market.

The SoC approach provides all stack-functionality and application-layer components within a single chip. With SoC, developers can write the most optimized solutions with the lowest power consumption. But the whole application is constrained to the actual hardware limitations. Figure 2.14 shows an example of a multiprotocol SoC approach.

In a Network Co-Processor (NCP) approach, the chip only runs the functions necessary for networking. The most common configuration is that the chip communicates with a second device called the ”host” processor via either SPI or UART. The host is responsible for the application management.

A Radio Co-Processor (RCP) approach is similar to the NCP. The chip only runs the physical layers of the stacks while the host runs the application and network layers.

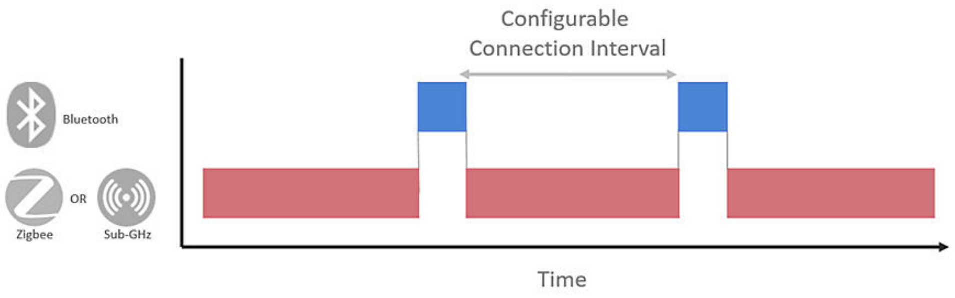

Figure 2.12: Concurent multiprotocol. Source: [36]

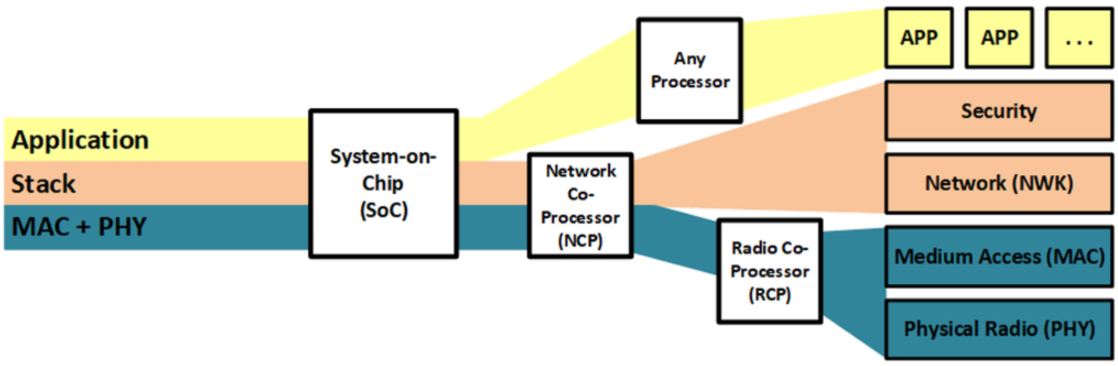

Figure 2.13: Stack designs. Source: [38]

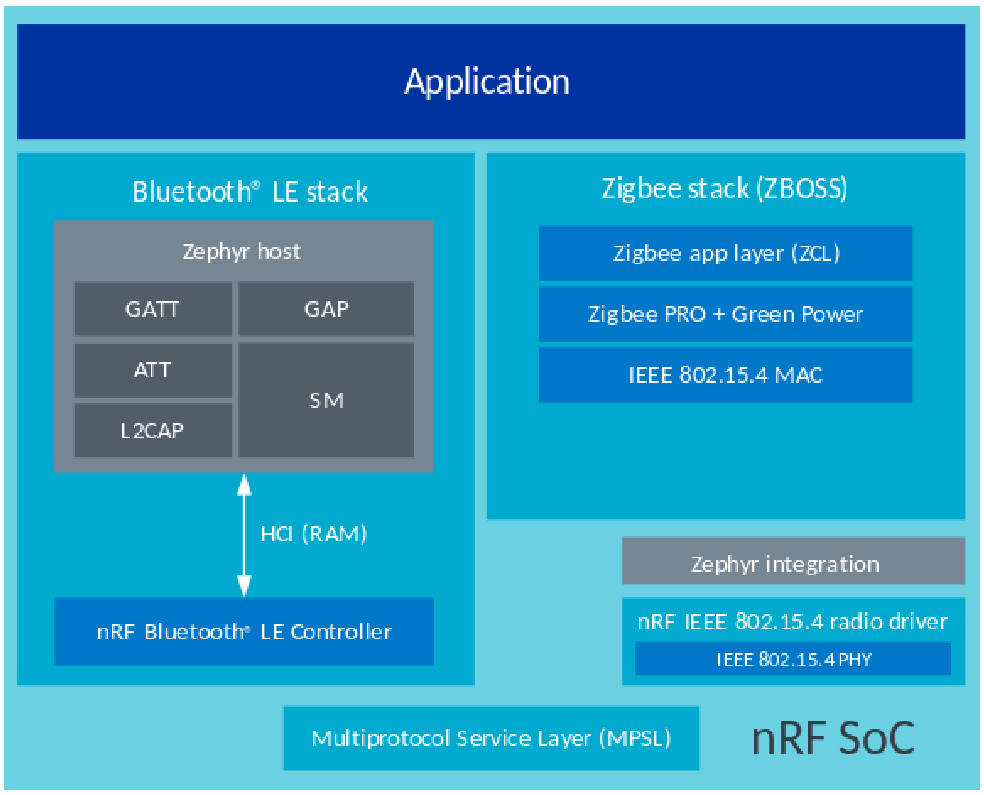

Figure 2.14: Nordic SoC design. Source: [42]

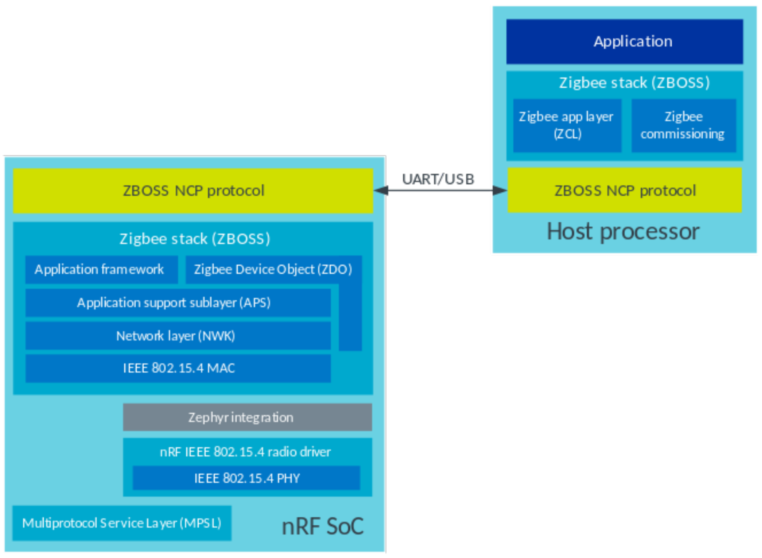

Figure 2.15: Zigbee Network Co-processor design. Source: [42]

With this architecture integration of multiple protocols that use the same radio band can be better achieved. In BLE, this is achieved with the host-controller-interface (see: Figure 2.17). Thread uses the spinnel protocol to control radio devices (see: Figure 2.18).

2.2.3 Implementation

Based on the Bluetooth, Thread, and Zigbee governing bodies[7, 16, 57], four companies are working on and promoting multiprotocol chips: Nordic Semiconductor, NXP Semiconductors, Espressif Systems, and Silicon Laboratories. They all have software development kits (SDKs) that support the development of multiprotocol applications on SoC, NCP, and RCP configuration.

Two distinct approaches could be differentiated between the four companies. Espressif's and NXP's solutions are achieving multiprotocol via having multiple RF transceivers—one for 802.15.4 and another one for Bluetooth. This way, they can use the full range of the 802.15.4 protocol with sub-GHz transmission.

Silicon Labs and Nordic went with a more packageable approach. Since Bluetooth and 802.15.4 can be operated on the 2,4 GHz frequency range, their solution implements only a single radio transceiver. With this approach, the overall footprint of the device could be smaller. Operating a single radio with multiple stacks could cause issues during communications due to concurrency.

All companies mentioned above implement solutions based on an RTOS-flavour operating system for SoC applications and support Linux-based host operating systems for NCP and RCP use cases.

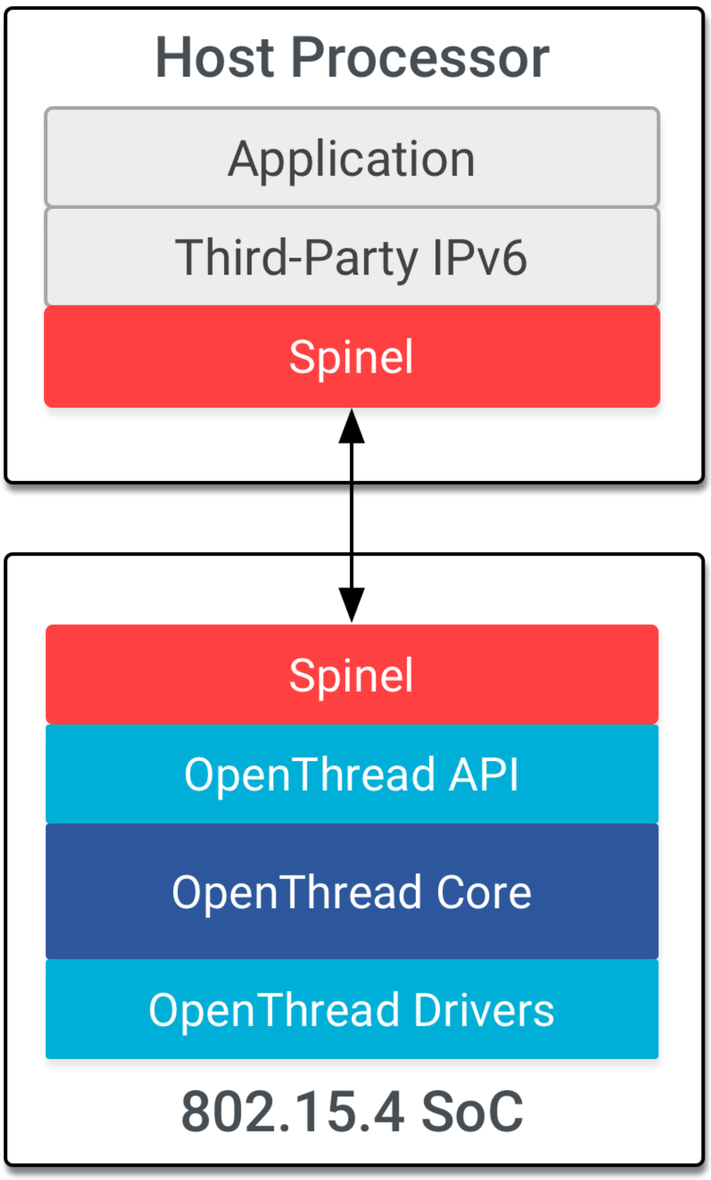

Figure 2.16: Thread Network Co-processor design. Source: [46]

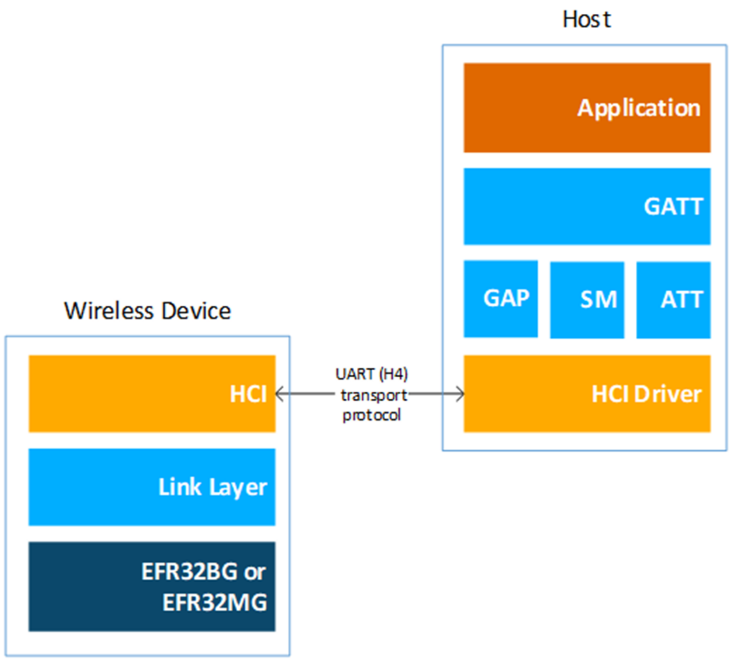

Figure 2.17: BLE Radio Co-processor design. Source: [35]

2.3 Software testing

2.3.1 Overview

A lot of different standard bodies and writers have defined software testing. Standardization organizations define software testing as follows: ”The process consisting of all lifecycle activities, both static and dynamic, concerned with planning, preparation, and evaluation of software products and related work products; to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose, and to detect defects.” [14]

Testing can have different goals, such as getting information about the quality, supporting decision-making, detecting defects, or preventing defects. In the process, there are two different approaches: test-as-information-provider and test-as-quality-accelerant.

In the test-as-information-provider approach, testing is usually executed last in development. There are independent test teams and separate test phases with fixed release cycles. Meanwhile, test-as-quality-accelerant follows a test-always strategy. In the latter approach, testers are quality assistants, and developers write tests alongside testers. This allows shorter and more fine-grade releases.

2.3.2 Testing levels

According to ISTQB, test levels are groups of test activities that are organized and managed together. Each test level is an instance of the test process performed

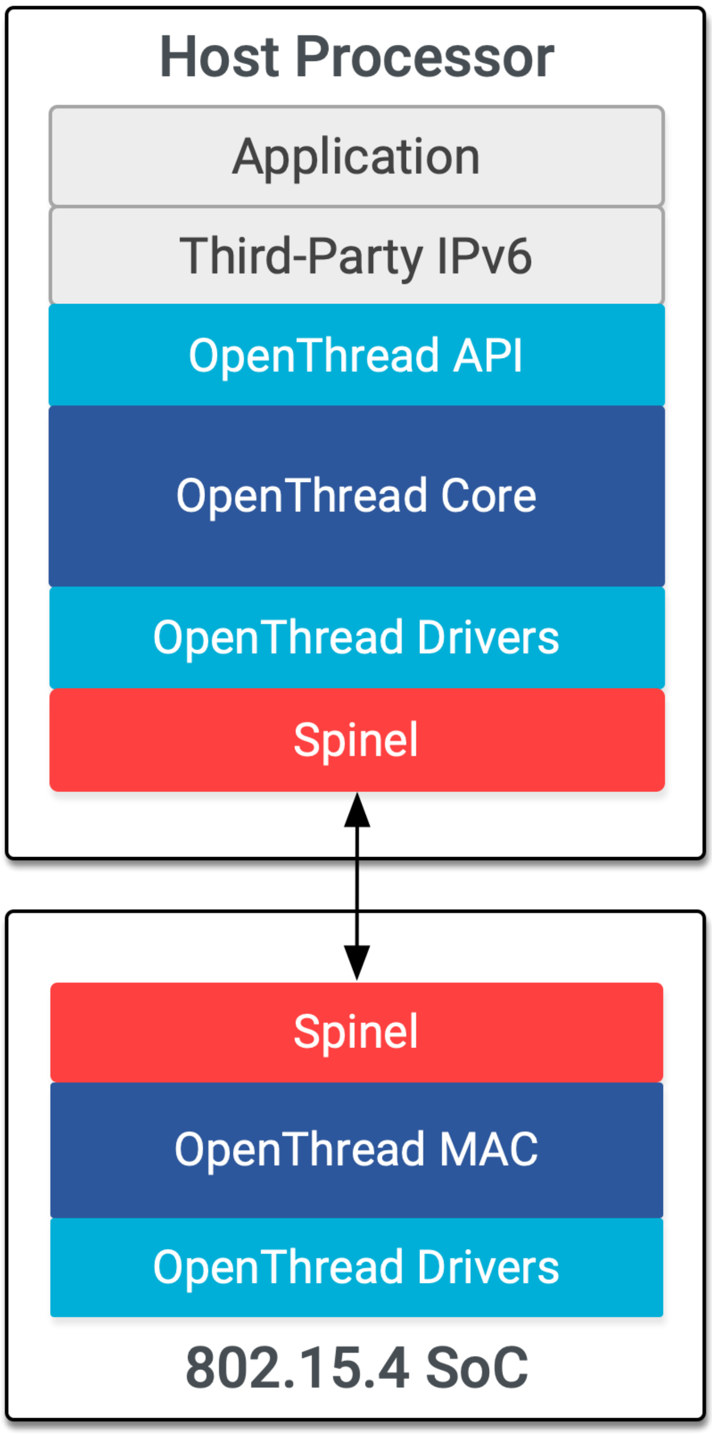

Figure 2.18: Thread Radio Co-processor design. Source: [46]

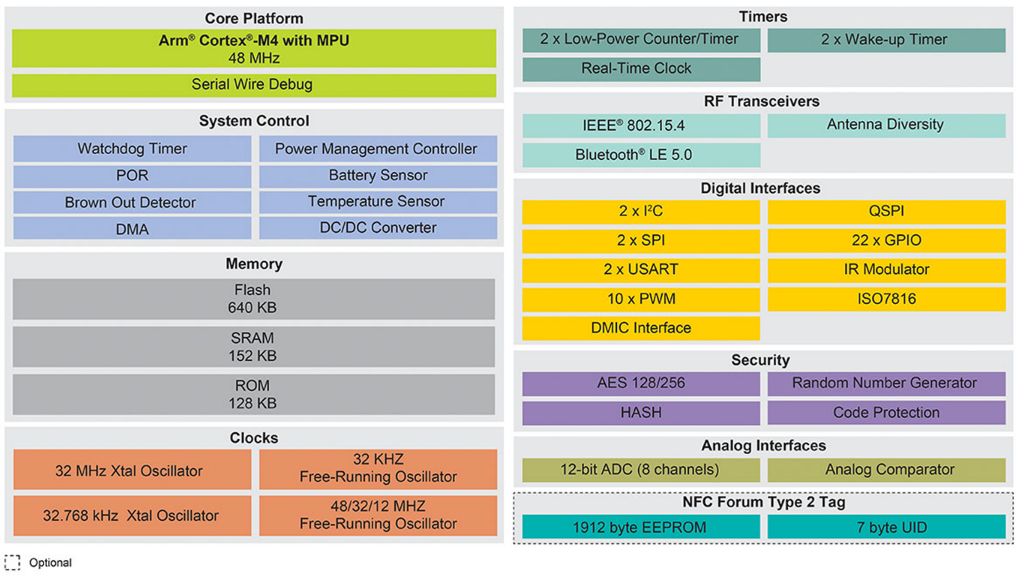

Figure 2.19: NXP k32w061 block diagram. Source: [44]

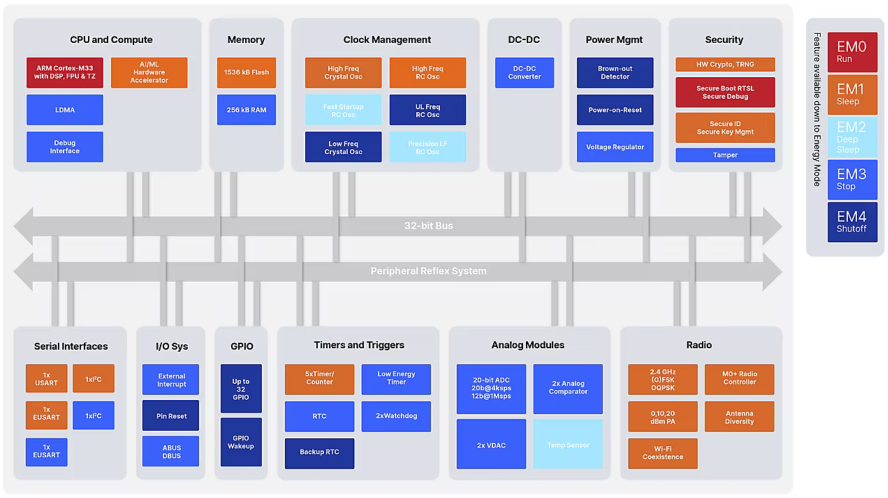

Figure 2.20: Silabs xg24 block diagram. Source: [53]

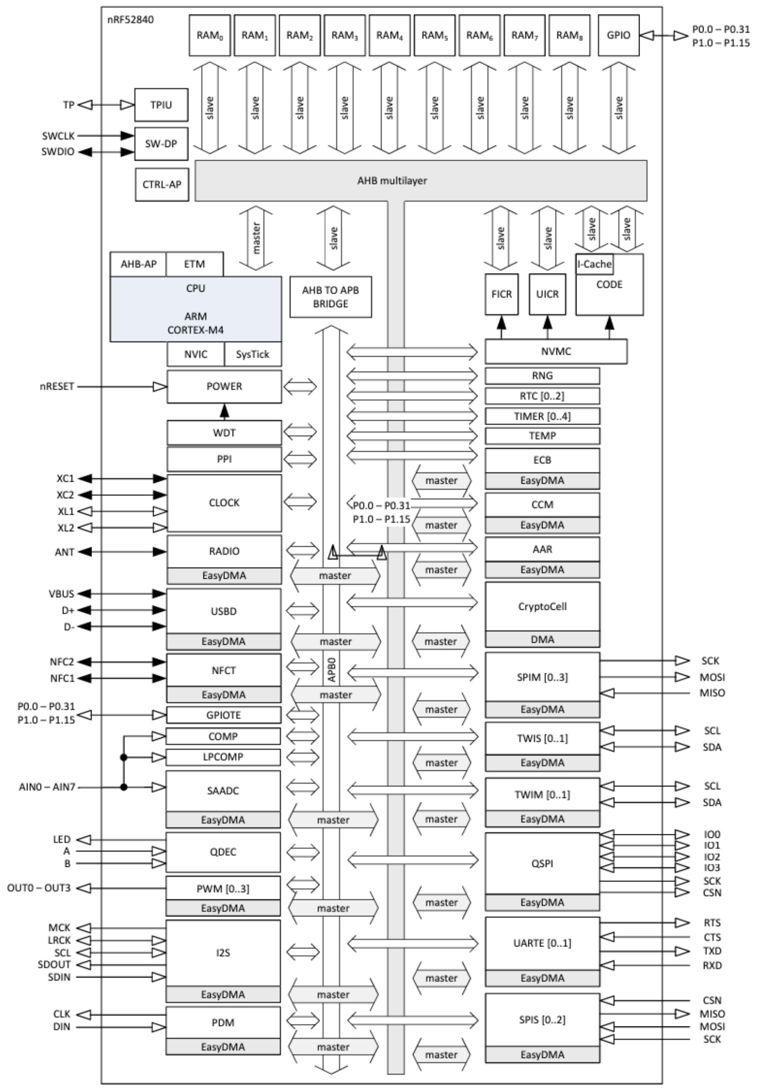

Figure 2.21: Nordic Semi nRF52840 block diagram. Source: [43]

	Small tests	Medium tests	Large tests
Time Goals	$\leq$ 100 ms	$\leq$ 1 sec	As quickly as possible
Time Limits	1 minute	5 minutes	1 hour

Table 2.2: Google's test levels. (Source: [64])

concerning software at a given stage of development, from individual components to complete systems or, where applicable, system of systems. [14]

The ISTQB differentiates the following levels: component (unit), component integration (integration), system, system integration, and acceptance testing. Unit testing focuses on a single component in isolation and is usually performed by developers. Integration testing focuses on testing the interfaces and interactions between components. System tests focus on the overall behavior and capabilities of the entire system or product.

Whittaker et al. show [64] a different approach to test levels used in Google. Test levels are defined in execution time; see Table 2.2. Google uses three different levels: small, medium, and large tests. Small tests cover a single unit of code in a faked environment. Medium tests cover two or more interacting features. Large tests represent real user scenarios and use real user data sources.

2.3.3 Test types

Test types are groups of test activities related to specific quality characteristics, and most of those test activities can be performed at every test level. [14]

Common types used in the industry are functional, non-functional, and regression testing.

Functional testing evaluates the functions that a system (or component) should perform. The main objective is to check the functional completeness and correctness of the system.

In non-functional testing, the goals are to test how well the system behaves. Commonly used software quality characteristics are defined by ISO/IEC 25010 standard[32]. These characteristics are performance, compatibility, usability, reliability, security, maintainability, and portability. Non-functional tests may need specific test environments.

Regression testing confirms that changes in the codebase did not inject new defects into the functionality. According to ISTQB, performing an impact analysis is advisable to optimize the extent of the regression testing.

2.3.4 Model-based test generation

Model-based testing (MBT) can best be defined as using models during the testing process [30]. Utting et al. [60] defined a complete taxonomy for MBT approaches. MBT has two distinct artifacts: a developed model and an adaptor component,

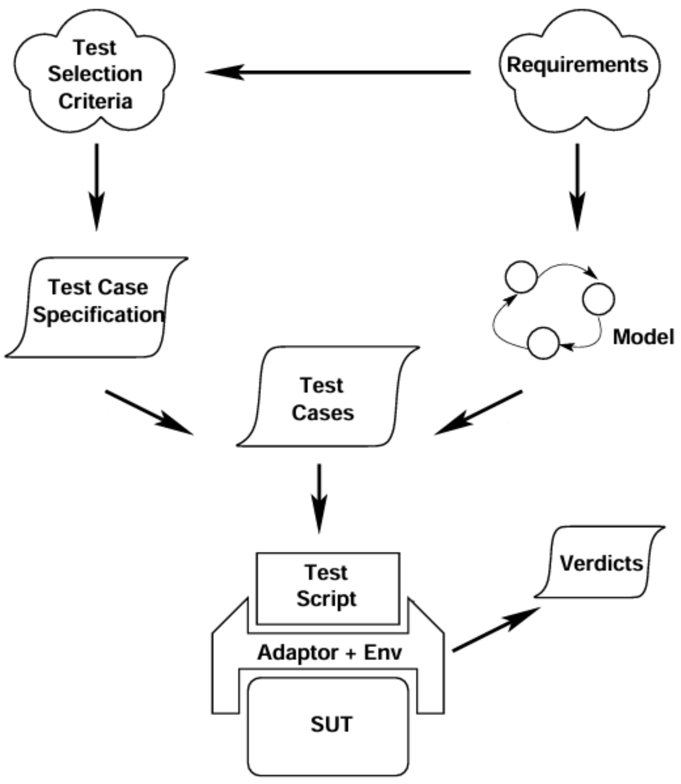

Figure 2.22: MBT process. (Source: [60])

which ties the model to the SUT. They defined that the approach has five major activities (see: Figure 2.22):

1. model creation
2. criteria definition
3. test case generation
4. creation of the adaptor component
5. test execution on the SUT

Testers use MBT from unit up to system level testing, but it is used primarily for system and integration testing.

Models can best describe the system's intended behavior under test or its environment. Several surveys show that there is only one preferred modeling approach [19, 3, 34], but state machine representations are the most popular. Then, state models get transformed into a directed graph. Modeling can be used to abstract different aspects of the SUT (function, data, and communication abstraction)

From the models - based on set criteria - automated tools could generate abstract test cases. The criteria could be in different forms: structural, requirement, or other coverage. Deriving test cases is usually based on random generation or graph search algorithms. Generation could be considered online or offline. In online generation concrete execution starts at the same time as generation. On the other hand, the offline mode can create abstract tests before the adaptor implementation. After the necessary codes are developed, the execution engine could run the offline-generated tests.

Execution engines create concrete test cases in conjunction with the adaptor component from the abstract test cases. The adaptor component wraps the SUT and handles the communication directly. It is used to control and monitor different aspects of the tested system. After the execution, the engine makes a verdict from the SUT responses and reports it back to the QA engineer.

Graphwalker[1] is a simple but powerful model-based testing tool. It can represent models as directed graphs. Model elements can be either vertexes or edges. Vertexes represent state or validation, and edges represent actions or transitions. The tool integrates a model editor (”studio”) and a test execution component. The executor supports many different generation strategies and coverage criteria.

2.3.5 Protocol testing

Nowadays, academia focuses on proving security issues on protocols and radio. Current studies work on finding vulnerabilities in the protocol itself or its implementation [4, 48, 51, 61, 41, 47, 1].

In terms of IoT testing, a more scalable approach is needed. The literature mentions two types of approaches: simulation and testbeds. Simulation is used when no physical device is available. It is great for validating the functional aspects of a protocol.

The SUT needs to be set up in a realistic environment to evaluate the non-functional properties. Werner-Allen et al. presented a concept for a real wireless network testbed. In their idea, a management layer is introduced for logging and debugging physical devices while users can interact with them. This way, managing and deploying large-scale tests on real hardware is possible[62].

According to Gluhak et al, more and more public testbeds are available[25]. More recently, Behnke et al.[6] proposed a new hybrid testbed solution that mixes real hardware with optionally simulated nodes.

A survey by Ferriera et al. and Ismail et al.[23, 31] showed that model-based techniques were emerging in the realms of IoT testing. Altough MBT is popular, the survey also showed that only a few papers used MBT to generate test cases automatically.

From the papers, I concluded that IoT mainly conducts system-level testing. The industry is using multiple setups to test software, but realistic results come from testing on testbeds. As mentioned above, MBT is an effective technique and is widely available. The deficiency of test case generation from models shows that the industry is not using the full capability of modern MBT tools.

[1]`https://graphwalker.github.io/`

Chapter 3

Methodology

3.1 Overview

The problem defined in Chapter 1 is heavily behavior-driven. Model-based testing has excellent usage in describing behavior. With state machines, I could create a complete state space of the multiprotocol device.

I chose graphwalker as my modeling tool. It has a limited toolset for creating directed graphs, but it is a mature open-source package that can be used for MBT. The software can handle shared states that can synchronize multiple models. At its core, graphwalker is implemented in Java and only supports Maven projects natively.

Since one of the goals was to create a project that runs on Python, more than graphwalker was needed. There is a Python package called altwalker, which acts as a Python wrapper around the executable and connects to graphwalker via internal sockets.

My proposed process for testing IoT application is summarized by Figure 3.1. I use graphwalker as the modeling tool of choice and then create a custom adaptor for the running environment. As Figure 3.2 shows, the test model is created with graphwalker. Then, the test execution environment generates abstract test cases with the help of graphwalker (via altwalker) and instantiates them with the adaptor. The adaptor runs concrete test steps on the embedded hardware. The execution creates a verdict based on the given test case.

Section 3.2 details the modeling process and final artifacts created during the thesis work. The adaptor design is described in Section 3.3, and Section 3.4 shows the entire test setup.

All the code and models developed for this thesis are publicly available on GitHub[1].

[1]`https://github.com/DAbraham2/Masters-Thesis`

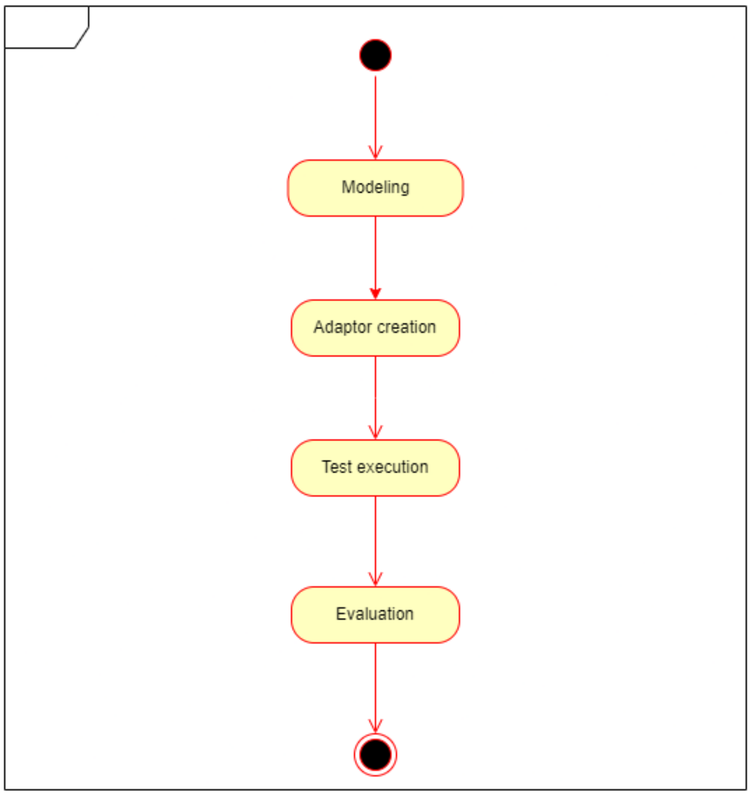

Figure 3.1: Proposed process

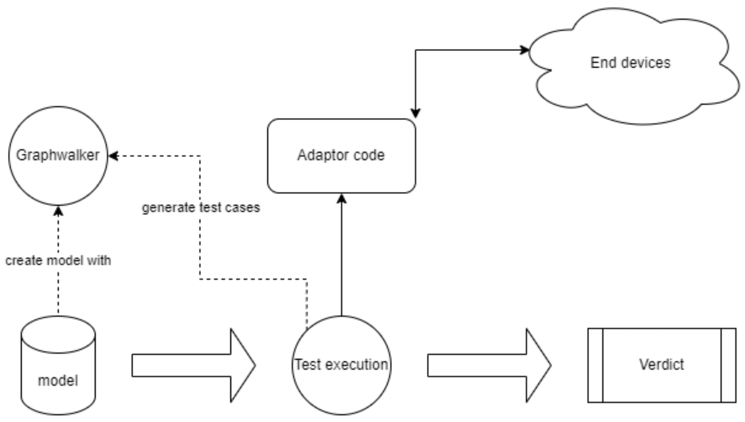

Figure 3.2: Testing flow

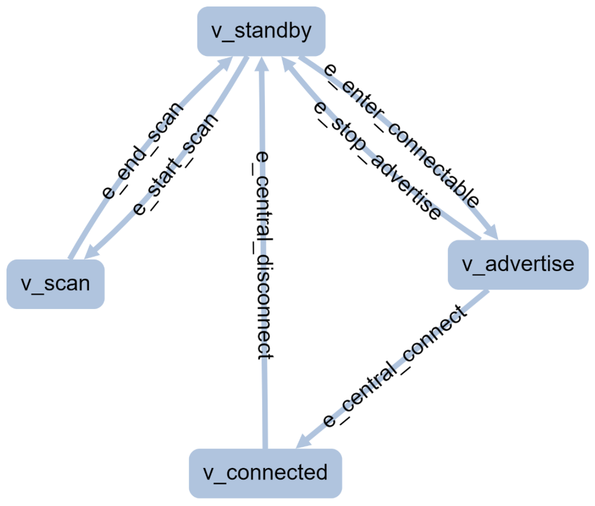

Figure 3.3: BLE model

3.2 System modeling

Graphwalker represents models as directed graphs. A graph node is called a vertex, and the edge is called an edge. Graphwalker recommends creating actions on edges and validating only on vertexes.

The Bluetooth Core specifies what a peripheral device shall be capable of. GAP defines discovery and connection mode as mandatory, leaving bonding, periodic advertising, and isochronous broadcast as excluded. Also, the LE Controller states that a minimal device shall be capable of scanning or advertising. After all, the set

$$\{standby, scan, advertise, connected\}$$

can describe the BLE device's state space. Since I am not testing the functionality of the profile - only the network capabilities - using any GATT services were not required.

As 802.15.4 defines the whole association procedure, the Zigbee states should be:

$$\{disconnected, scan, association, joined\}$$

Luckily, the application I was given had some plugins prepared for networking. The network-steering plugin automatically made the association process and gave its results. It tested the scanning, beaconing, and association procedures all in one. The final set of states is then reduced to just:

$$disconnected, connected$$

Another plugin- throughput- was used to test the network's ability to relay data. The complete graph of Zigbee can be found in the following.

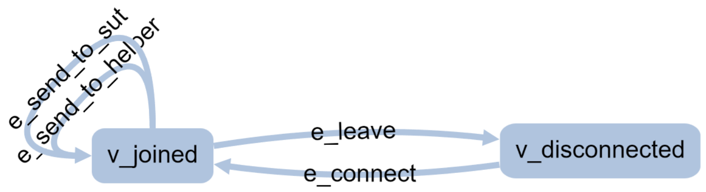

Figure 3.4: Zigbee model

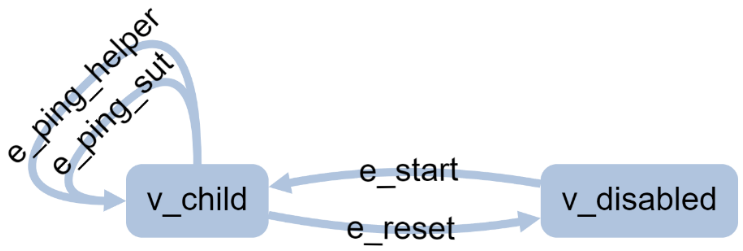

Figure 3.5: Thread model

For Thread, just as in Zigbee, the joining procedure is abstracted away. The complete state space of a device is:

$$\{disabled, detached, child, router, leader\}$$

One of the limitations is that the application cannot stop the thread network; it just resets the stack. Since I am just creating a single-node network, having the router state is not relevant for this type of testing. With those two statements, the Thread state space is:

$$\{disabled, child\}$$

ICMP ping was used in the child state to test the connection.

After creating the separate models, I could make the complete model for operating the three protocols side by side. Since graphwalker does not support regions or concurrently running multiple models, I combined the three models into one. The graph's vertexes could be described as

$$\{standby, scan, advertise, peripheral\} \times \{disconnected, joined\} \times \{disabled, child\}$$

Figure 3.6 shows the visual representation of the combined model. The final model consists of 18 vertexes and 98 edges.

3.3 Adaptation

An adaptor was needed to translate abstract test cases into concrete tests. Altwalker can create a template package from a given model. It has limitations regarding if a package already exists; it will fail to update the project. Creating models incrementally is beneficial. To solve this problem, I have created a simple utility

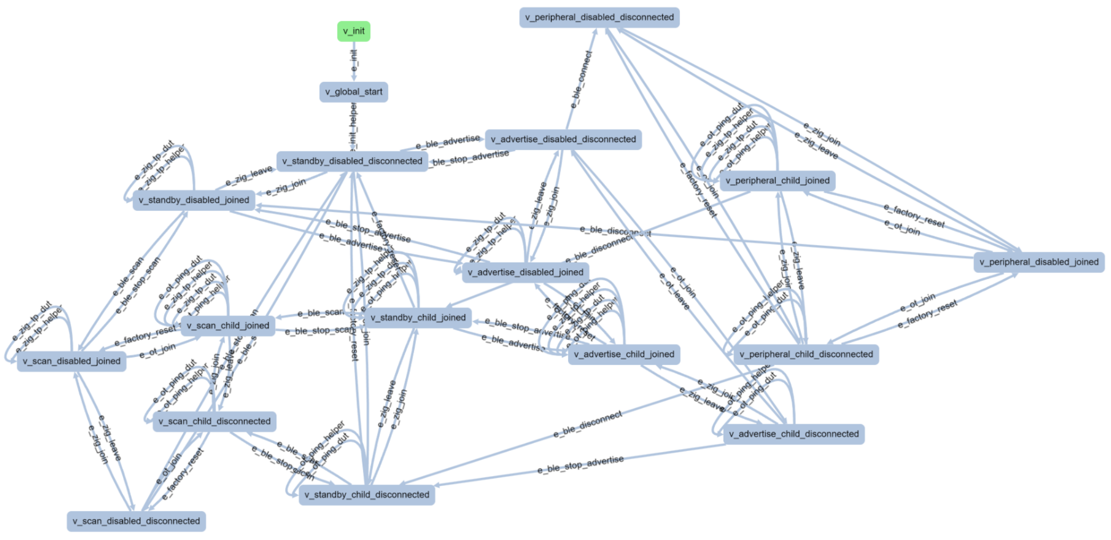
v_init
e_init
v_global_start
e_init_helper
v_standby_disabled_disconnected
v_advertise_disabled_disconnected
v_peripheral_disabled_disconnected
v_standby_disabled_joined
v_peripheral_child_joined
v_peripheral_disabled_joined
v_advertise_disabled_joined
v_scan_child_joined
v_standby_child_joined
v_scan_disabled_joined
v_advertise_child_joined
v_peripheral_child_disconnected
v_scan_child_disconnected
v_advertise_child_disconnected
v_standby_child_disconnected
v_scan_disabled_disconnected
e_ble_advertise
e_ble_stop_advertise
e_ble_connect
e_ble_disconnect
e_ble_scan
e_ble_stop_scan
e_zig_join
e_zig_leave
e_zig_tp_dut
e_zig_tp_helper
e_ot_join
e_ot_leave
e_ot_ping_dut
e_ot_ping_helper
e_factory_reset

Figure 3.6: Combined protocol model

that parses the package into a Python abstract syntax tree (AST). Then, the code will modify the AST to create missing methods. After the modifications, it will translate to Python syntax and save the file - leaving the old code intact.

To create the scenario defined in Chapter 1, I needed to communicate with four different devices. Of those four, three were SoC applications, and I could use Silicon Labs' pyBgApi solution for the BLE helper.

The software architecture is divided into four layers. The highest level is the abstract device represented with contents in the interface package. It contains all the necessary functions a BLE, a Zigbee, and a Thread device shall do. The application package contains the concrete implementations of the interfaces.

The plugins package is the middleware for each application. It functions as a transport-free abstraction for commands. This allows commands to be independent of the underlying transport layer. If there is a new channel - let's say ssh - commands can be ported without any changes to the plugins.

Transportation was placed on the lowest layer. It represents the connection with the test-ware, the Silicon Labs' WPK main board. A link is abstracted behind the BaseTransport class. Different transport implementations are kept internal within the package. To help create a suitable transport for a device, I made the transport_factory method that chose the matching transport for the given type. With this solution, it is possible to add new configurations for a device type easily.

A device can produce events asynchronously from the commands. For this, I made the telnet connection to listen on a separate thread. Furthermore, the plugins implement the observable pattern. It is possible to attach plugin handle methods to a transport, achieving asynchronous event handling.

The MpScenario class is responsible for initializing the applications. The package implements a singleton pattern. Test adaptors can access it via the get_mp_scenario function. This approach makes it possible to connect it to a multi-model test suite.

The whole architecture is shown in Figure 3.7

3.4 Test setup

I prepared a docker file that contains the adaptor code. It facilitates redistribution and ease of use. Docker is helpful because although altwalker is used, the test runner must also install graphwalker. Many companies have strict policies on what can be installed on company devices, and using docker is safe because it has a base image with graphwalker already installed.

The test setup consists of a test runner connected to a local network. Silicon Labs' WPKs are connected to the network as well. The test runner controls and monitors all the physical devices. Figure 3.8 shows the test setup overview.

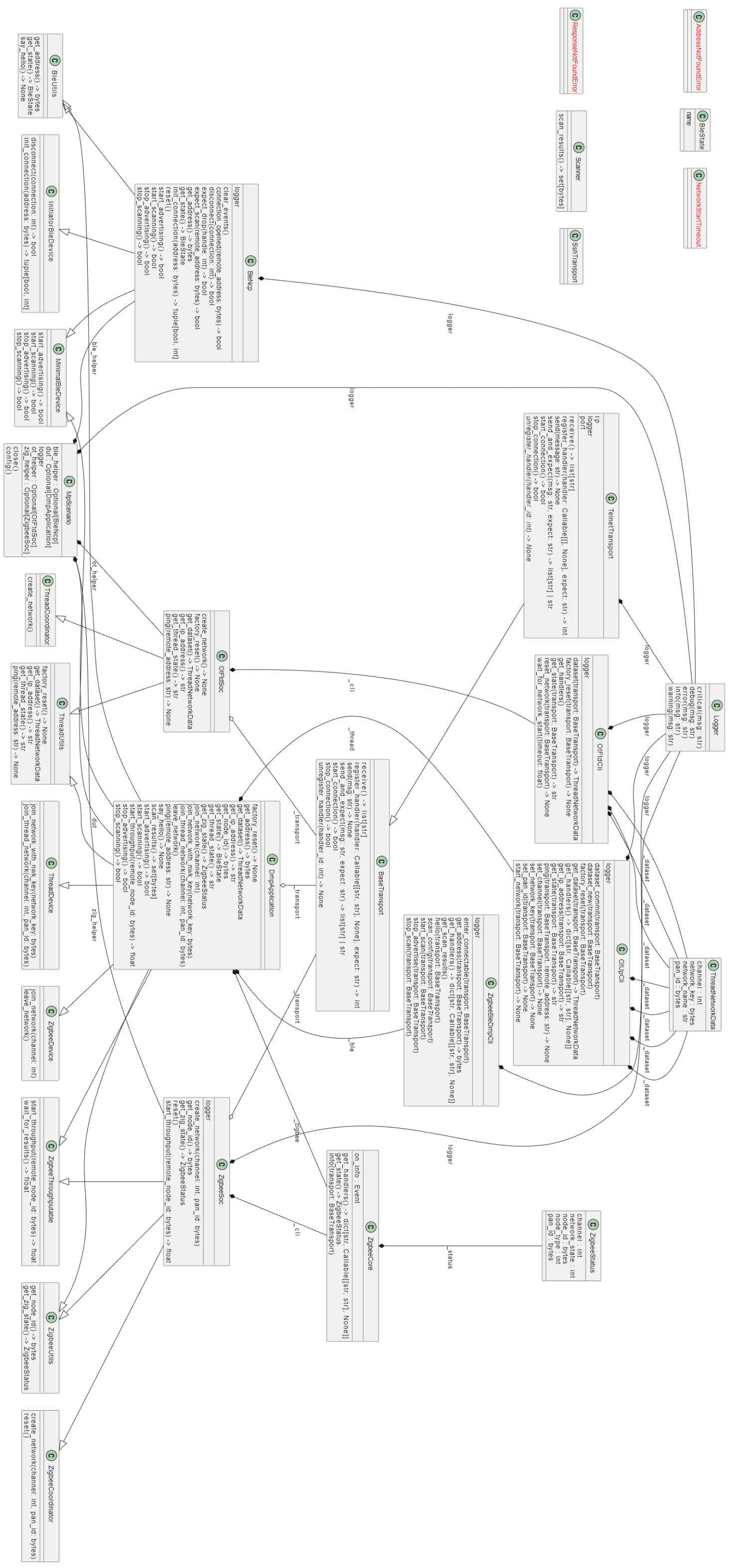

Figure 3.7: Adaptor architecture

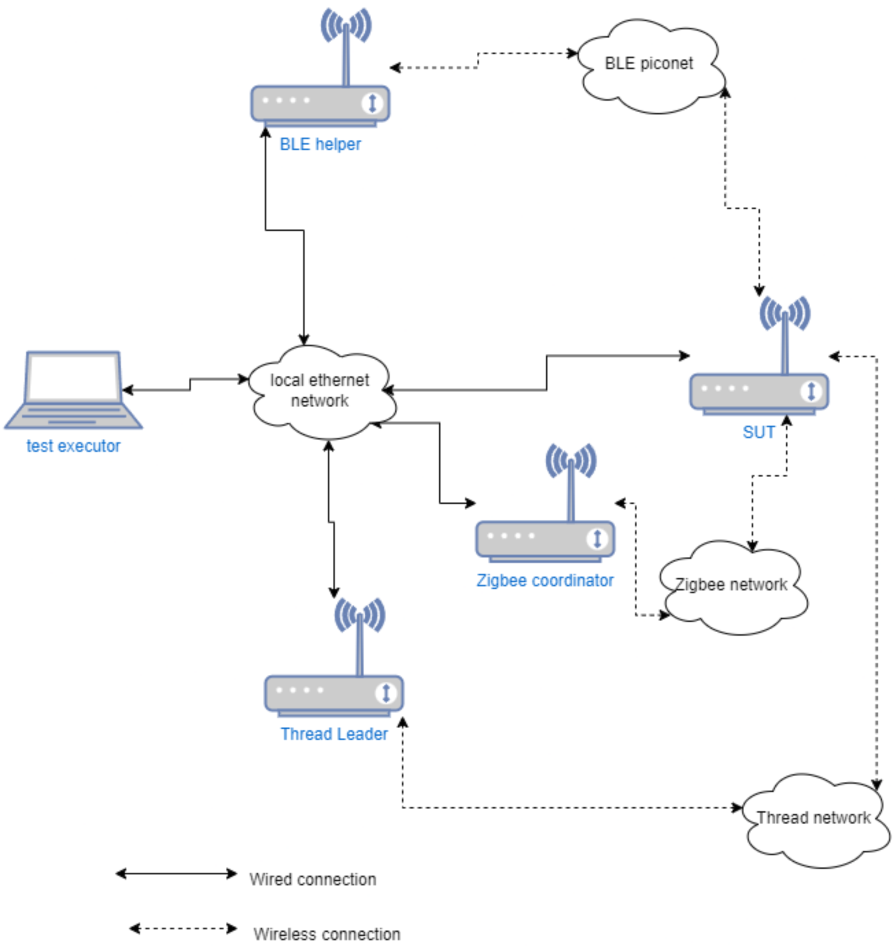

Figure 3.8: Overall test architecture

Chapter 4

Results

The main goal of testing was to find any functional defects in the multiprotocol application. In a production environment, test execution time can be crucial as well. The test runner was configured to achieve 100 percent vertex and at least 80 percent edge coverage. As the number of edges reached almost one hundred, the reduction of edge coverage helped the reduction of testing time.

For the first attempts, a defect started to form from the execution of the model. After running more and more times, I found out that resetting the thread stack of the application(thread factory_reset) caused the drop of a formed BLE connection.

This behavior is not desirable because if there is an ongoing operation on the BLE side when the reset comes, it could cause other malfunctions at the application level.

During the literature review phase of the thesis, I noticed that Nordic - another manufacturer - published a limiting factor for multiprotocol scenarios [1]. During BLE scanning, the 15.4 stack could expect packet drops. Considering this, I prepared the adaptor code to be flexible on Zigbee and Thread operations. Having a rigid code would cause false negative test results.

After modifying the test suite, I noticed a significant drop on success rate of Zigbee joins while the BLE stack was scanning. The successes dropped to roughly 70 percent. Anything less than 50 percent would be considered a failure.

All the test results and observations can be seen on Table 4.1. Overall, the testing conducted a state exploration through the given model. It used both quick random[2] and random path generation.

In conclusion, I have managed to find new bugs and performance degradations in the given application, achieving 60 percent vertex coverage. The edge coverage stayed below 30 percent during testing due to execution failure on defects in the SUT. Table 4.2 summarizes coverage data. During testing, Graphwalker was generated more than 100 different test cases for various states. I was able to generate a

[1] `https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/protocols/multiprotocol/index.html`

[2] `https://github.com/GraphWalker/graphwalker-project/wiki/Generators-and-stop-conditions#quick_random-some-stop-conditions-`

Type	Defect	Blocker
Functional	BLE drops on thread reset	True
Non-functional	Zigbee join fails occasionally while BLE scan	False
Non-functional	Zigbee throughput success drops on BLE scan	False
Non-functional	Thread ping packet loss increases on BLE scan	False

Table 4.1: Test results summary

Table 4.2: Model coverage summary

	visited	missed	coverage
Vertex	10	8	55%
Edge	21	77	21%

test execution that covered many requirements from the standards. The largest requirement coverage that was achieved without failing was 35%.

Chapter 5

Conclusions

5.1 Summary

In this thesis, I made a literature review on currently used IoT network protocols. After the protocols, I conducted a search for testing strategies applicable to use in IoT testing.

After finishing the literature review, I presented a small but valid use case. Then, I constructed an abstract model representing the scenario. I have managed to create abstract test cases automatically from the abstract model. I developed a Python library to map abstract test cases to concrete tests. This library controls and monitors the System Under Test and the surrounding helper nodes.

Lastly, I managed to containerize the running environment to be platform-independent. Next, I tested the provided application and summarized the test results.

5.2 Results

After conducting a thorough literature review on different protocols and testing methodologies, I concluded that using model-based techniques could be beneficial in testing IoT protocol implementations. It is a valuable technology that, when used correctly, can reduce test design time and costs. Model-based testing can automatically create test suites to cover different aspects of the system.

During the thesis, I managed to find new defects in an already existing and tested product. This shows yet again that model-based testing could discover new edge cases without a tester's manual intervention or supervision.

5.3 Future work

For future work, I would model more aspects of the application. Expand the model with the application-level logic, especially using BLE GATT or Zigbee Cluster services to test more profound and broader functionalities.

As graphwalker has limited capabilities in modeling, I recommend creating a translator that can transform more sophisticated models (i.e., UML state machine) into a directed graph model parsed by graphwalker. This way, the tester can use higher abstractions to model the problem with composite states.

Bibliography

[1] Dimitrios-Georgios Akestoridis, Vyas Sekar, and Patrick Tague. On the security of thread networks: Experimentation with openthread-enabled devices. In *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '22, page 233–244, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392167. DOI: `10.1145/3507657.3528544`. URL `https://doi.org/10.1145/3507657.3528544`.

[2] Shadi Al-Sarawi, Mohammed Anbar, Rosni Abdullah, and Ahmad B. Al Hawari. Internet of things market analysis forecasts, 2020–2030. In *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, pages 449–453, 2020. DOI: `10.1109/WorldS450073.2020.9210375`.

[3] Bruno Legard Anne Kramer and Rovert V. Binder. 2016/2017 model-based testing user survey: Results. `http://www.cftl.fr/wp-content/uploads/2017/02/2016-MBT-User-Survey-Results.pdf`, 2016.

[4] Daniele Antonioli. Bluffs: Bluetooth forward and future secrecy attacks and defenses. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 636–650, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400700507. DOI: `10.1145/3576915.3623066`. URL `https://doi.org/10.1145/3576915.3623066`.

[5] Mateusz Banaszek, Wojciech Dubiel, Jacek Łysiak, Maciej Dundefinedbski, Maciej Kisiel, Dawid Łazarczyk, Ewa Głogowska, Przemysław Gumienny, Cezary Siłuszyk, Piotr Ciołkosz, Agnieszka Paszkowska, Inga Rüb, Maciej Matraszek, Szymon Acedański, Przemysław Horban, and Konrad Iwanicki. 1kt: A low-cost 1000-node low-power wireless iot testbed. In *Proceedings of the 24th International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '21, page 109–113, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450390774. DOI: `10.1145/3479239.3485708`. URL `https://doi.org/10.1145/3479239.3485708`.

[6] Ilja Behnke, Lauritz Thamsen, and Odej Kao. Héctor: A framework for testing iot applications across heterogeneous edge and cloud testbeds. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion*, UCC '19 Companion, page 15–20, New York,

NY, USA, 2019. Association for Computing Machinery. ISBN 9781450370448. DOI: 10.1145/3368235.3368832. URL https://doi.org/10.1145/3368235.3368832.

[7] Bluetooth SIG. Member directory. https://www.bluetooth.com/develop-with-bluetooth/join/member-directory/, 2023.

[8] Bluetooth Special Intrest Group. Bluetooth core specification 4.0. Technical report, 2010.

[9] Bluetooth Special Intrest Group. Bluetooth core specification v5.0. Technical report, 2016. https://www.bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=421043.

[10] Bluetooth Special Intrest Group. Our history (2018. may 25.). https://web.archive.org/web/20180525083558/https://www.bluetooth.com/about-us/our-history, 2018.

[11] Bluetooth Special Intrest Group. Bluetooth core specification v5.2. Technical report, 2019. https://www.bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=421043.

[12] Bluetooth Special Intrest Group. Specifications and documents (2023. 05. 16). https://www.bluetooth.com/specifications/specs/?types=specs-docs&keyword=Core+Specification, 2023.

[13] Bluetooth Special Intrest Group. Gatt specification supplement 9. Technical report, 2023. https://www.bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=569770.

[14] International Software Testing Qualifications Board. Certified tester foundation level syllabus v4.0. https://istqb-main-web-prod.s3.amazonaws.com/media/documents/ISTQB_CTFL_Syllabus-v4.0.pdf, 2023.

[15] NXP B.V. *ZigBee 3.0 - Facilitating the Internet of Things*, 2016.

[16] Connectivity Standards Alliance. Members. https://csa-iot.org/members/, 2023.

[17] Rareş Cristea, Mihail Feraru, and Ciprian Paduraru. Building blocks for iot testing: A benchmark of iot apps and a functional testing framework. In *Proceedings of the 4th International Workshop on Software Engineering Research and Practice for the IoT*, SERP4IoT '22, page 25–32, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450393324. DOI: 10.1145/3528227.3528568. URL https://doi.org/10.1145/3528227.3528568.

[18] CSA. Csa members. https://web.archive.org/web/20230526073333/https://csa-iot.org/members/, 2023.

[19] Arilo C. Dias Neto, Rajesh Subramanyan, Marlon Vieira, and Guilherme H. Travassos. A survey on model-based testing approaches: A systematic review. In *Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies: Held in Conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*, WEASELTech '07, page 31–36, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595938800. DOI: 10.1145/1353673.1353681. URL https://doi.org/10.1145/1353673.1353681.

[20] Myra Dideles. Bluetooth: A technical overview. *XRDS*, 9(4):11–18, jun 2003. ISSN 1528-4972. DOI: 10.1145/904080.904083. URL https://doi.org/10.1145/904080.904083.

[21] Muhammad Shoaib Farooq, Shamyla Riaz, Adnan Abid, Kamran Abid, and Muhammad Azhar Naeem. A survey on the role of iot in agriculture for the implementation of smart farming. *IEEE Access*, 7:156237–156271, 2019. DOI: 10.1109/ACCESS.2019.2949703.

[22] Muhammad Shoaib Farooq, Osama Omar Sohail, Adnan Abid, and Saim Rasheed. A survey on the role of iot in agriculture for the implementation of smart livestock environment. *IEEE Access*, 10:9483–9505, 2022. DOI: 10.1109/ACCESS.2022.3142848.

[23] Vinícius Gomes Ferreira, Caio Guimarães Herrera, Simone Souza, Ricardo Ribeiro dos Santos, and Paulo Sérgio Lopes de Souza. Software testing applied to the development of iot systems: Preliminary results. In *Proceedings of the 8th Brazilian Symposium on Systematic and Automated Software Testing*, SAST '23, page 113–122, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400716294. DOI: 10.1145/3624032.3624049. URL https://doi.org/10.1145/3624032.3624049.

[24] Lavínia Freitas and Valéria Lelli. Using machine learning on testing iot applications: A systematic mapping. In *Proceedings of the Brazilian Symposium on Multimedia and the Web*, WebMedia '22, page 348–358, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394093. DOI: 10.1145/3539637.3558049. URL https://doi.org/10.1145/3539637.3558049.

[25] Alexander Gluhak, Srdjan Krco, Michele Nati, Dennis Pfisterer, Nathalie Mitton, and Tahiry Razafindralambo. A survey on facilities for experimental internet of things research. *IEEE Communications Magazine*, 49(11):58–67, 2011. DOI: 10.1109/MCOM.2011.6069710.

[26] Thread Group. *Thread Network Fundamentals White Paper*, 2022. https://portal.threadgroup.org/DesktopModules/Inventures_Document/FileDownload.aspx?ContentID=633.

[27] Thread Group. Thread 1.3.0 public specification. Technical report, 2023. https://www.threadgroup.org/ThreadSpec.

[28] Thread Group. Thread group - members. https://web.archive.org/web/20230525142814/https://www.threadgroup.org/thread-group, 2023.

[29] Texas Instruments. *What's new in Zigbee 3.0*, 2019.

[30] International Software Testing Qualifications Board. *ISTQB® Foundation Level Certified Model-Based Tester Syllabus*, 2015. https://istqb-main-web-prod.s3.amazonaws.com/media/documents/ISTQB-CT-MBT_Syllabus_v1.0_2015.pdf.

[31] Teeba Ismail and Ibrahim Hamarash. Model-Based Quality Assessment of Internet of Things Software Applications: A Systematic Mapping Study. *International Journal of Interactive Mobile Technologies (iJIM)*, 14:128, 06 2020. DOI: 10.3991/ijim.v14i09.13431.

[32] ISO/IEC 25010:2011. Systems and software Quality Requirements and Evaluation (SQuaRE). Standard, International Organization for Standardization, 2011.

[33] Adam Kozłowski and Janusz Sosnowski. Analysing efficiency of ipv6 packet transmission over 6lowpan network. page 104451J, 08 2017. DOI: 10.1117/12.2280699.

[34] Anne Kramer and Bruno Legard. 2019 model-based testing user survey: Results. https://www.cftl.fr/wp-content/uploads/2020/02/2019-MBT-User-Survey-Results.pdf, 2019.

[35] Silicon Labs. Qsg169: Bluetooth® quick-start guide for sdk v3.x and higher. https://www.silabs.com/documents/public/quick-start-guides/qsg169-bluetooth-sdk-v3x-quick-start-guide.pdf, 2022.

[36] Silicon Labs. Multiprotocol mcus. https://www.silabs.com/wireless/multiprotocol?tab=learn, 2023.

[37] Silicon Labs. *UG103.2: Zigbee Fundamentals*, 2023. https://www.silabs.com/documents/public/user-guides/ug103-02-fundamentals-zigbee.pdf.

[38] Silicon Labs. *UG103.3: Software Design Fundamentals*, 2023. https://www.silabs.com/documents/public/user-guides/ug103-03-fundamentals-design-choices.pdf.

[39] Martin Leopold, Mads Bondo Dydensborg, and Philippe Bonnet. Bluetooth and sensor networks: A reality check. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, page 103–113, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581137079. DOI: 10.1145/958491.958504. URL https://doi.org/10.1145/958491.958504.

[40] Amir Makhshari and Ali Mesbah. Iot bugs and development challenges. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 460–472, 2021. DOI: 10.1109/ICSE43902.2021.00051.

[41] Philipp Morgner, Stephan Mattejat, Zinaida Benenson, Christian Müller, and Frederik Armknecht. Insecure to the touch: Attacking zigbee 3.0 via touchlink commissioning. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '17, page 230–240, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350846. DOI: 10.1145/3098243.3098254. URL https://doi.org/10.1145/3098243.3098254.

[42] Nordic Semi. Zigbee architectures. https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/protocols/zigbee/architectures.html, 2023.

[43] Nordic Semi. nrf52840 product specification. https://infocenter.nordicsemi.com/index.jsp?topic=%2Fstruct_nrf52%2Fstruct%2Fnrf52840.html, 2023.

[44] NXP. K32w061/41. https://www.nxp.com/assets/block-diagram/en/K32W061_41.pdf, 2023.

[45] NXP. Multiprotocol mcus. https://www.nxp.com/products/wireless-connectivity/multiprotocol-mcus:MULTIPROTOCOL-MCUS, 2023.

[46] Openthread. Co-processor designs. https://openthread.io/platforms/co-processor, 2023.

[47] Jiradet Ounjai, Valentin Wüstholz, and Maria Christakis. Green fuzzer benchmarking. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2023, page 1396–1406, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702211. DOI: 10.1145/3597926.3598144. URL https://doi.org/10.1145/3597926.3598144.

[48] Travis Peters, Timothy J. Pierson, Sougata Sen, José Camacho, and David Kotz. Recurring verification of interaction authenticity within bluetooth networks. In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '21, page 192–203, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383493. DOI: 10.1145/3448300.3468287. URL https://doi.org/10.1145/3448300.3468287.

[49] Pedro Martins Pontes, Bruno Lima, and João Pascoal Faria. Test patterns for iot. In *Proceedings of the 9th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation*, A-TEST 2018, page 63–66, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450360531. DOI: 10.1145/3278186.3278196. URL https://doi.org/10.1145/3278186.3278196.

[50] Vaclav Rechtberger, Miroslav Bures, and Bestoun S. Ahmed. Alternative effort-optimal model-based strategy for state machine testing of iot systems. In *Proceedings of the 2nd World Symposium on Software Engineering*, WSSE

'20, page 141–145, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450387873. DOI: 10.1145/3425329.3425330. URL https://doi.org/10.1145/3425329.3425330.

[51] Mengfei Ren, Xiaolei Ren, Huadong Feng, Jiang Ming, and Yu Lei. Z-fuzzer: Device-agnostic fuzzing of zigbee protocol implementation. In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '21, page 347–358, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383493. DOI: 10.1145/3448300.3468296. URL https://doi.org/10.1145/3448300.3468296.

[52] Hassan Sartaj, Shaukat Ali, Tao Yue, and Kjetil Moberg. Testing real-world healthcare iot application: Experiences and lessons learned. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2023, page 2044–2049, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400703270. DOI: 10.1145/3611643.3613888. URL https://doi.org/10.1145/3611643.3613888.

[53] Silabs. Efr32mg24 series 2 multiprotocol wireless soc. https://www.silabs.com/wireless/zigbee/efr32mg24-series-2-socs, 2023.

[54] Silicon Labs. gecko sdk. https://github.com/SiliconLabs/gecko_sdk/tree/gsdk_4.3, 2023.

[55] Silicon Labs. xg24-pk6010a. https://www.silabs.com/development-tools/wireless/efr32xg24-pro-kit-20-dbm?tab=overview, 2023.

[56] Statista. Iot total annual revenue worldwide from 2020 to 2030. https://web.archive.org/web/20231205184608/https://www.statista.com/statistics/1194709/iot-revenue-worldwide/, 2023.

[57] ThreadGroup. Ourmembers. https://www.threadgroup.org/thread-group#OurMembers, 2023.

[58] Kevin Townsend, Carles Cufi, Akiba, and Robert Davidson. *Getting Started with Bluetooth Low Energy.* O'Reilly Media, Inc., 2014.

[59] Ishaq Unwala, Zafar Taqvi, and Jiang Lu. Thread: An iot protocol. In *2018 IEEE Green Technologies Conference (GreenTech)*, pages 161–167, 2018. DOI: 10.1109/GreenTech.2018.00037.

[60] Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing approaches. *Software testing, verification and reliability*, 22(5): 297–312, 2012.

[61] Xian Wang and Shuang Hao. Don't kick over the beehive: Attacks and security analysis on zigbee. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 2857–2870, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394505.

DOI: 10.1145/3548606.3560703. URL https://doi.org/10.1145/3548606.3560703.

[62] 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, 2005.*, pages 483–488, 2005. DOI: 10.1109/IPSN.2005.1440979.

[63] Lindsey N. Whitehurst, Todd R. Andel, and J. Todd McDonald. Exploring security in zigbee networks. In *Proceedings of the 9th Annual Cyber and Information Security Research Conference*, CISR '14, page 25–28, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450328128. DOI: 10.1145/2602087.2602090. URL https://doi.org/10.1145/2602087.2602090.

[64] James A. Whittaker, Jason Arbon, and Jeff Carollo. *How Google Tests Software*. Addison-Wesley Professional, 2012.

[65] Junjie Yin, Zheng Yang, Hao Cao, Tongtong Liu, Zimu Zhou, and Chenshu Wu. A survey on bluetooth 5.0 and mesh: New milestones of iot. *ACM Trans. Sen. Netw.*, 15(3), may 2019. ISSN 1550-4859. DOI: 10.1145/3317687. URL https://doi.org/10.1145/3317687.

[66] Cyrine Zid, Dmytro Humeniuk, Foutse Khomh, and Giuliano Antoniol. Double cycle hybrid testing of hybrid distributed iot system. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, ICSEW'20, page 529–532, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379632. DOI: 10.1145/3387940.3392218. URL https://doi.org/10.1145/3387940.3392218.