UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Araz Heydarov

Developing a Testing Framework for Internet of Things Systems using IoTempower as an Example

Master's Thesis (30 ECTS)

Supervisor(s): Prof. Ulrich Norbisrath

Tartu 2024

Developing a Testing Framework for Internet of Things Systems using IoTempower as an Example

Abstract:

The Internet of Things (IoT) has made technology much more powerful, enabling smart features to be built into every aspect of daily life. However, this involves a lot more than developing a device that connects to the internet – IoT development is complicated and difficult. This can be especially daunting for beginners and even industry professionals as well. One challenge faced by developers, rather than users, is building a reliable IoT framework, as testing is not a common practice across these platforms. IoTempower is no exception. IoTempower was created as an accessible framework for engaging with IoT that could be used by anyone from tinkerers and programmers all the way up to students, artists, or professionals–and everyone in between! It is also great for schools because not only does it teach about home automation systems with real-world applications, but it also serves as a powerful teaching tool for higher-level concepts surrounding internet-enabled devices.

This thesis focuses on regression testing and hardware management within the IoTempower framework, with the broader aim of studying how to effectively test hardware frameworks. The main aim is to develop a complete test suite that allows new features to integrate seamlessly without breaking existing ones while adding new hardware support. Enhancing these areas provides a better user experience, makes the framework more reliable overall, and advances testing methods adopted in various IoT development frameworks, thus increasing reliability in different IoT scenarios.

Keywords: IoT, Hardware components, IoT deployment, Testing, Regression-testing, Testing framework

CERS: P170, Computer Science

Testraamistiku loomine asjade interneti süsteemi testimiseks, IoTempoweri näitel

Lühikokkuvõte:

Asjade internet (IoT) on muutnud tehnoloogia palju võimsamaks, võimaldades nutikaid funktsioone integreerida igapäevaelu igasse aspekti. See hõlmab aga palju enamat kui Interneti-ühendusega seadme väljatöötamist – asjade Interneti arendamine on keeruline ja raske. See võib olla eriti hirmutav algajatele ja isegi valdkonna professionaalidele. Üks väljakutse, millega arendajad silmitsi seisavad, on usaldusväärse IoT raamistiku loomine, kuna testimine pole nendel platvormidel tavaline praktika. IoTempower pole erand. IoTempower loodi ligipääsetava raamistikuna asjade Internetiga suhtlemiseks, mida saaksid kasutada kõik alates algajatest kuni professionaalideni. See sobib suurepäraselt ka koolidele, kuna sellega on võimalik õpetada koduautomaatika süsteemide tööpõhimõtteid reaalsete rakenduste näidel ning aitab mõista ka keerukamaid kontseptsioone IoT seadmetes.

See lõputöö keskendub regressioonitestimisele ja riistvarahaldusele IoTempoweri raamistikus, mille laiem eesmärk on uurida, kuidas riistvararaamistikke tõhusalt testida. Peamine eesmärk on välja töötada täielik testkomplekt, mis võimaldaks lisadauusi funktsionaalsusi ja riistvaratuge ilma olemasolevat funktsionaalsust rikkumata. Testimise täiustamine võimaldab tulemusena paremat kasutuskogemust, muudab raamistiku üldiselt usaldusväärsemaks ja aitab edendada olemasolevaidasjade interneti arendusraamistike, suurendades nende töökindlust.

Võtmesõnad: Asjade internet, Riistvarakomponendid, Asjade interneti juurutamine, Testimine, Regressioonitestimine, Testimisraamistik

CERCS: P170, Arvutiteadus

1. Introduction 4
2. Background 5
2.1. Evolution of IoT Technology 6
2.2. Existing IoT Development Frameworks 8
2.3. Raspberry Pi as a Development Platform 10
2.3.1. Key Features and Benefits of Raspberry Pi: 10
2.3.2. Why We Use Raspberry Pi: 11
2.4. Containerization Technologies, Docker 12
2.4.1. Project-Specific Use of Docker 13
2.5. Introduction to IoTempower Framework 14
3. Related Work and Literature Review 18
3.1. Tasmota and ESPHome 20
3.2. Microsoft Azure IoT Testing Services 23
4. Case Study 25
4.1. Case Application 25
4.2. Design 25
4.3. Test Cases Implementation 29
4.3.1. Test Case 1 - Installation Testing 29
4.3.2. Test Case 2 - Compilation Testing 31
4.3.3. Test Case 3 - Deployment Testing 34
4.3.4. Test Case 4 - Hardware Testing 39
5. Evaluation 43
6. Discussion 45
7. Conclusion and Future Work 46
8. Acknowledgements 47
9. References 48
10. Appendix 55
10.1. Interview Questions 55
10.2. Testing IoTempower Documentation 56
License 62

1. Introduction

The Internet of Things (IoT) has transformed everyday objects into connected devices capable of communicating over the Internet, leading to automation across different fields. The growth of this technology calls for development frameworks that are able to efficiently support the exploration, creation, and deployment phases required by any given solution based on such platforms. Testing becomes critical at this point since functionalities need verification before being deployed on workable environments which can be achieved through proper testing procedures. (Murad et al., 2018) The study, therefore, explores specialized testing environment development within the IoTempower system to make the installation process more manageable and improve hardware utilization for sustainable IoT solutions.

IoTempower is mainly adopted in educational settings where it acts as a useful instrument for teaching practical applications of the Internet of Things. (ulno, 2024) Students in classrooms often experiment with various configurations and hardware modules which may lead to unintended bugs or system failures that can take a lot of time to trace back. In addition, these disruptions not only interrupt learning but also indicate the need for stability tests on IoTempower, even during modifications. Updating codes frequently, together with changes made on different devices used within the same system, usually leads to unforeseen errors during the development stages of any typical IoT project. The challenge lies in ensuring continuous integration of new functions into the existing framework without compromising its stability and, at the same time, adding support for additional peripheral interfaces as required by evolving technological trends. This scenario underlines the importance of an immediate fault detection capability through reliable testing environment design, which saves time by preventing system collapse, hence improving user satisfaction.

This chapter gives an overview of what will be discussed within this paper, including each section's purpose and content. The "Background" section lays the groundwork by discussing the history of IoT technology, current IoT development frameworks, and how computing platforms such as Raspberry Pi contribute to this field. It also presents the IoTempower framework, describes what it can do, and where research in this area is headed. It then progresses to the idea

generation and implementation of the testing framework, beginning with software testing and continuing with tests that integrate both software and hardware. This process includes various types of tests to ensure that the devices function well together under different operational conditions. The following "Case Study" section examines a specific application scenario, describing the experimental methodology used to implement the testing framework. The "Design" and "Implementation" sections follow, where the practical analysis and implementation of the thesis objectives is presented, focusing on evaluating the testing framework's capabilities to quickly identify and address issues within IoTempower in software and hardware contexts. Moreover, to complement the technical analysis of IoTempower and its testing frameworks, semi-structured interviews were conducted with key stakeholders involved in IoT development to gather qualitative insights on the practical challenges and effectiveness of the proposed solutions. The "Evaluation" section reports the results of these tests, providing a baseline assessment and detailed results for each test case. In the "Discussion" section, these findings are analyzed, discussing the broader implications of the research and suggesting potential directions for future research and development. The thesis concludes with the "Conclusion" section, which summarizes the research findings and highlights the contributions made to IoT development, offering practical insights for practitioners and researchers.

2. Background

The background part presents the key concepts and technologies that are relevant to the thesis. It starts with the development of Internet of Things (IoT) technology, pointing out important milestones and improvements in this field. Following, it reviews existing frameworks for IoT application development, considering what each framework does and its features. After that, it looks at Raspberry Pi as a platform for developing IoT projects by highlighting its most useful features as well as benefits for such projects. Also included here are containerization technologies like Docker, which are used to handle dependencies when working on different parts of an application concurrently, thus speeding up the whole process of development management. Lastly, the IoTempower framework is presented, outlining its features and capabilities. The background section helps set the stage for understanding related work sections and design phases, which will be discussed later on.

2.1. Evolution of IoT Technology

The Industrial Revolution has undergone several phases, each marked by transformative technological advancements. The first Industrial Revolution began with the invention of the modern coal-powered steam engine that changed transportation means and manufacturing processes completely. Secondly, the wide-ranging adoption of electricity occurred during the Second Industrial Revolution alongside the introduction of mass production techniques, i.e., assembly line methods (Sharma and Jit Singh, 2020) .

The third phase, also known as the digital revolution, saw computers being used widely while other forms of information sharing took place digitally, too, thus changing forever how we process information itself.This era laid the groundwork for the fourth Industrial Revolution, which is defined by integrating digital, physical, and biological systems and prominently features the Internet and the Internet of Things (IoT) (Sharma and Jit Singh, 2020).

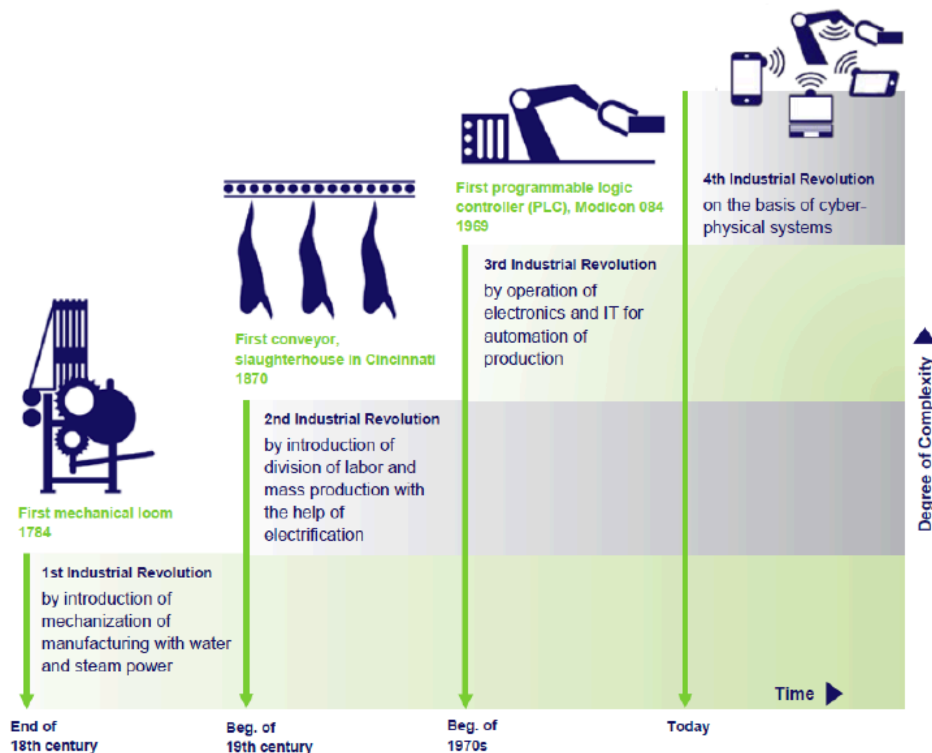


Figure 1. The four Industrial Revolutions (Kagermann et al., 2013)

Kevin Ashton introduced the Internet of Things concept in 1999 during a presentation on the advantages of Radio Frequency Identification (RFID) technology for the P&G company (Foote, 2022). However, the first documented instance of an Internet of Things application dates back to 1991 at Cambridge University, where a group of academics shared images of a coffee machine over the Internet using a camera system (PetaPixel, 2013). This system remained operational until August 22, 2001 (www.cl.cam.ac.uk, n.d.).

Human needs and perspectives on technology have also changed with the development of technology, and the Internet has now begun to be defined as a crucial tool for communication. Today, it has evolved from the classical Internet to a platform where billions of devices are controlled. These devices produce large amounts of data from the sensors they hold and the Internet they are connected to. The Internet of Things is a technology that brings together billions of devices that can communicate over the Internet and enables the relevant devices to perceive the physical world, send and receive observation data, and act in parallel with this data. (Fizza et al., 2021) The Internet of Things (IoT) has developed as a new digital transformation wave, enabling real-time detection, data collection, and sharing.

In sum, The Internet of Things is a network of intelligent communication systems that allows all devices/objects connected to a network to communicate data among themselves with a specific protocol without human intervention and data entry (Sinan Ameen Noman et al., 2023) . As mentioned before, IoT can collect billions of data and also make decisions with this data. Today, IoT technology is used in different areas such as smart homes, smart cities, energy measurement, construction, health, agriculture, animal tracking, the public sectors, logistics, and transportation (Chataut, Phoummalayvane and Akl, 2023).

According to the report published by Statista Online Services in June 2021, the number of Internet of Things devices worldwide was 8.74 billion in 2020, and it is estimated that this number will exceed 23.4 billion by 2030. On an industrial scale, Internet of Things devices are used extensively in electricity, gas, steam, water supply and waste management, retail and wholesale sales, transportation, and storage business lines. It is estimated that the number of

Internet of Things devices used on an industrial scale will exceed 8 billion in 2030 (Statista, 2021).

The development of the Internet of Things (IoT) has followed general technological progress in terms of better connectivity, smaller sensors, improved data analysis, and increased cloud computing. These advancements have enabled IoT's use across different sectors to enhance efficiency, automation, and innovation in operations. Continuous investigations are keeps pushing IoT forward further with concepts like AI, edge computing, and blockchain being among the areas of interest (Ahmed et al., 2023).

The evolution of IoT is changing people's relationship with technology as well as their environment (Elgazzar et al., 2022). It provides new perspectives for individuals when it becomes more advanced and widely deployed since it can reveal how things work better or serve humanity well. However, it should be used responsibly.

2.2. Existing IoT Development Frameworks

IoT development frameworks form one of the vital elements contributing to creativity within this extended domain; they allow users to create diverse IoT applications by shaping them differently.

IoT development frameworks present various hardware platforms that come along with software tools, libraries, and resources necessary for easy building up IoT solutions (Jose Reena K, 2023) .They follow a structured way of creating an IoT application where they provide essential functionalities and features needed during such processes, thus making work easier for developers.

Frameworks also provide reflexive interfaces, clear documentation, and user-friendly development environments that simplify things for all levels while at the same time giving room for customization through third-party integration that seamlessly fits specific requirements (Hasan Derhamy et al., 2015).

Several main IoT development frameworks have emerged, each delivering to different user needs and preferences:

One such framework is Node-RED, which acts as a flow-based programming tool designed for the visual creation of IoT applications through function nodes wiring together. The idea behind this method was to make it simple, even for those who might not know much about coding, but still powerful when dealing with complex system integration since many pre-built nodes are available in its library accordingly, making work much easier, especially among beginners. Also, due to continuous contribution by the community, it grows fast and, therefore, becomes popular among other systems, too (Clerissi et al., 2018).

Another significant player in the IoT framework landscape is Home Assistant, an open-source home automation platform that emphasizes privacy and local control.Home Assistant is compatible with numerous devices and can easily integrate into existing home infrastructure allowing users monitor their entire living environment from single dashboard. What sets Home Assistant apart further from other platforms lies in its processing ability which is done locally meaning no personal data has to be shared with external servers thus more secure particularly for privacy-conscious users (Home Assistant, 2024).

Several notable IoT development frameworks have emerged, each catering to different needs and preferences. **ESPHome** can be used to control ESP8266 and ESP32-based devices through simple configuration files, which also integrate well with Home Assistant (ESPHome, n.d.). Its simplicity and flexibility are its strengths, but it has a limited GPL license. Apache NiFi acts as a platform for data logistics, which allows automation of data movement between systems, thus making it suitable for IoT that require complex data processing. **OpenHAB** is an open-source home automation software that works with many systems and devices and provides support for various communication protocols, among others (www.openhab.org, n.d.). **FHEM** is a highly configurable Perl server for house automation but needs more technical skills during installation and maintenance processes.**Tasmota**, being an open-source firmware for ESP8266 and ESP32, enables control via MQTT, HTTP, Web UI, etcetera, with extensive community support being one of its pros (tasmota.github.io, n.d.). **Apple HomeKit, Amazon Alexa,** and **Google Assistant** are voice-controlled platforms for smart home automation with broad device integration capabilities (Writer, 2020). Thread, on the other hand, is a low-power wireless networking protocol designed specifically for IoT applications where energy saving is critical, as well as having strong mesh networks (www.threadgroup.org, n.d.).

AWS Iot Core , Google Cloud IoT Core , Microsoft Azure IoT can be referred to as cloud service management frameworks by major providers which allow scalable secure infrastructures where devices can connect control and explore (Pierleoni et al., 2020).

To sum up, these frameworks can integrate via MQTT, web requests, and CoAP among others to provide flexible and comprehensive IoT solutions. Choosing the right tools for particular IoT projects becomes easier when one understands the strengths and weaknesses of each framework; moreover, it also highligts the need for a specialized testing framework for IoTempower system.

2.3. Raspberry Pi as a Development Platform

The Raspberry Pi is a small, affordable computer that has greatly impacted the world of embedded systems and IoT (Internet of Things) development (Johnston and Cox, 2017). It was designed specifically for educational purposes in order to make learning about computers more accessible to people of all ages around the world. Since its introduction back in 2012, this credit card-sized single-board PC has become extremely popular not only among hobbyists but also among professionals who use it as an affordable tool for rapid prototyping within various fields such as robotics or home automation systems integration due its great community support base. (Hosny et al., 2023).

At its core, the Raspberry Pi has an ARM-based processor and various input/output (I/O) interfaces like USB, HDMI, GPIO (General Purpose Input/Output) pins, and network connections. This makes it suitable for various applications, from simple DIY projects to complex IoT systems (Johnston and Cox, 2017).

2.3.1. Key Features and Benefits of Raspberry Pi:

- **Affordable:** Generally, the Raspberry Pi was known as a cheap option, which makes it accessible to many, including students and hobbyists (Johnston and Cox, 2017). However, with the recent price increases, it remains relatively affordable compared to other computing solutions, and it is essential to question its affordability in the current market context.

- **Versatile:** The most commonly used operating system for Raspberry Pi is Raspberry Pi OS but there are several others that can run on it. This flexibility allows developers to run a wide range of applications from simple programming tools through all-in-one multimedia setups enabling them satisfy different project requirements as well as learn new things.
- **Connectivity:** With built-in Ethernet and Wi-Fi, the Raspberry Pi can easily connect to the internet and other devices. This connectivity makes it an effective edge gateway for networked applications, although using it as an access point can present challenges like limiting the number of sensors that can be connected at the same time. Specifically, using the Raspberry Pi, in our case, as an IoT gateway allows effective communication between IoT devices (Kumar and Geetha, 2017).
- **Expandable:** With its GPIO pins, a wide range of sensors, motors and other hardware can be attached by users. Nevertheless, it is worth mentioning that utilizing a Raspberry Pi to directly control simple sensors may not be efficient. A better option would be using it as the coordinator that communicates with simpler microcontrollers controlling individual sensors hence optimizing resource utilization (www.flux.ai, n.d.).
- **Community Support:** The Raspberry Pi has a big and active community around it with extensive documentation, tutorials and forums for help and ideas. This means that there is enough support from the community which ensures users could always find assistance or resources for many different kinds of projects to improve their learning or development process (Jolles, 2021).

2.3.2. Why We Use Raspberry Pi:

Raspberry Pi has been chosen to be used in this project because it comes preinstalled with all necessary software tools required to set up a gateway including deployment tools. Moreover, the device management system makes it easier than installing everything on regular PCs, which might pose complexity issues (Simadiputra and Surantha, 2021). It also provides an easy environment where developers and students can deploy their IoT applications quickly, thus allowing them to manage their projects based on IoT needs.

The main reason for selecting the Raspberry Pi is its widespread availability among end-users of the IoTempower framework; thus, our solutions will undergo real-life testing within the context of IoT situations (ut-teaching-ulno, 2023). This method simplifies testing while increasing the overall dependability of Internet of Things projects.

2.4. Containerization Technologies, Docker

The use of containers has become a revolutionary approach in modern software design as it allows developers to be more flexible and efficient in deploying apps across multiple environments. Docker is a platform that changed the way we build, deploy, and manage applications (Chung et al., 2016).

Containerization is not a new concept, but it has always been difficult due to its complexity and specific tooling requirements. Developers used to have many tools to perform different tasks, which made containers impractical in most cases. However, in 2013 Docker brought container usage to an unprecedented level of simplicity (Bhatia, 2017).

Docker's arrival had removed the complexity associated with traditional forms of containerization, making them easier to understand and therefore adopted by developers at all levels of experience within their field. Moreover, businesses could also confidently introduce containers into their development cycle without fear of wasting resources or time overruns. (James Turnbull, 2014).

At its core, Docker is a changeable containerization platform designed to facilitate containerized applications' creation, deployment, and management. What sets Docker apart is its user-friendly interface and vital ecosystem, which have moved it to one of the leading position of containerization technologies. (Bhatia, 2017).

Docker provides an entire ecosystem that caters to various containerization requirements. The foundation of Docker is the Docker Engine, which creates and runs containers. Docker Hub is a cloud-based registry service that facilitates the sharing and discovery of containerized apps by providing a platform where they can be published, pulled down, or searched. Multi-container applications are made easy to handle with Docker Compose, which defines complex

architectures and manages their lifecycle through simple commands. These components create an environment where developers have all the tools necessary to use containers effectively (James Turnbull, 2014).

In addition to this, there are some key principles behind Docker's success as a platform (Chung et al., 2016):

- **Portability:** By encapsulating applications and their dependencies within containers, Docker ensures portability across different environments, allowing applications to run consistently regardless of the underlying infrastructure.
- **Environmental Independence:** Docker containers abstract away the underlying environment, enabling applications to run seamlessly across disparate platforms without dependency issues.
- **Streamlining Development and Testing Processes:** Docker simplifies development and testing workflows by providing lightweight, reproducible environments for application testing, enabling early detection of issues and accelerating time to market.
- **Efficient Resource Management:** Docker optimizes resource utilization through container isolation by enabling multiple applications to run on a shared kernel, reducing overhead and enhancing efficiency.
- **Ideal for DevOps Applications:** Docker's compatibility with continuous integration and deployment (CI/CD) processes facilitates seamless automation and collaboration, driving agility and innovation in DevOps practices.

To sum up, Docker has appeared as a transformative force in containerization technologies, empowering developers and organizations to embrace agility, scalability, and innovation in application deployment. With its robust interface and ecosystem, Docker persists in shaping software development and infrastructure management.

2.4.1. Project-Specific Use of Docker

Many challenges were met during this project because of all the required and optional dependencies that need to be installed system-wide in order to run the application. For instance, creating a Raspberry Pi OS image with pre-installed dependencies solved this problem on

Raspberry Pis but not for PC operating systems due to additional overhead and complexity. Moreover, our work required using older JavaScript environments which added more confusion into dependency management.

To avoid system-wide installations and keep development agility, Docker has provided an effective solution by enabling us create isolated containers that encapsulate all necessary dependencies. It ensures that our application works consistently across different environments without dependency conflicts which makes it perfect for this project.

Although IoTempower functions properly in WSL, virtual machines or even on Mac command line; Docker is especially helpful when dealing with complex dependencies found in larger software systems. This approach simplifies setup process, guarantees functional development environment and solves dependency issues that we face (iotempire, 2019).

2.5. Introduction to IoTempower Framework

The Internet of Things (IoT) has entered a new phase marked by interconnectivity and automation across many industries and domains. IoT technology affects everything from smart homes and wearables to industrial automation and smart cities – opportunities for innovation and efficiency are limitless. However, developing and managing such applications are highly complicated, thus calling for robust frameworks/tools to ease the development process.

In light of these requirements, the IoTempower framework is a versatile solution simplify the building, deployment, and management of internet-based devices. It provides tools like libraries alongside protocols that help tame the complex nature involved while creating IoT applications so that developers can focus more on innovation rather than getting lost in infrastructure concerns. It is also designed for educational purposes, where students can learn about connecting things easily through its reflexive design coupled with documentation meant for beginners who may have never had any experience before touching such technologies but wish they could start somewhere practicable instead of only theory (ulno, 2024). This feature shows how adaptable IoTempower will likely be adopted in different IoT development scenarios given its practical deployment orientation and educational usage.

Device management is a key strength of IoTempower, facilitating seamless integration between various devices and sensors. This capability allows developers to connect and manage all components effortlessly, ensuring a smooth and efficient development process. With support for multiple communication protocols and various hardware platforms, IoTempower offers unparalleled flexibility, allowing users to exploit the potential of their IoT ecosystems without limitations fully (iotempire, 2019).

It also prioritizes simplicity and ease of use by providing user-friendly interfaces together with documentation that leads users through each step during the development process until completion in addition to setting up communication channels, defining device configurations, and deploying firmware updates, reducing complexity while taking less time in so doing improving efficiency (ulno, 2024).

Another aspect worth mentioning about IoTempower concerns scalability and extensibility whereby regardless of whether somebody wants to create small scale project or large-scale industrial implementation then, IoTempower should adapt accordingly since it has been built such that way; not forgetting modular architecture enables easy incorporation into existing systems thereby ensuring compatibility across wide range environments comprising interoperability among others too (iotempire, 2019).

Moreover, integration with PlatformIO further extends what can be done using IOtemp – here we have an all-inclusive development platform for embedded systems as well as IoT applications where lots more libraries besides development tools have been made available under one roof courtesy of platform.io, thereby speeding up entire cycle from concept design stage upto final product realization hence lowering time-to-market while boosting output at the same time (iotempire, 2019).
The IoTempower framework is a great step forward in the world of IoT development. It provides developers with an all-inclusive set of tools to address the challenges associated with creating and managing applications for the Internet of Things (ulno, 2024). IoTempower empowers

developers to express their creativity and innovate in the rapidly evolving IoT landscape by providing a solid foundation, user-friendly interfaces, and documentation.

2.6. Features and Capabilities of IoTempower

As mentioned in the previous chapter, IoTempower is an adaptable framework designed to simplify the development and deployment of Internet of Things (IoT) applications. It is designed for a broad audience, including tinkerers, makers, programmers, hobbyists, students, teachers, artists and professionals making it suitable to be used both for teaching IoT as well as real-world IoT deployments (ulno, 2024). The framework focuses on accessibility and support for learning purposes which makes it an ideal tool for teaching IoT and home automation. It has also added features like over-the-air (OTA) updates and automatic multi-device deployment to boost existing IoT implementations.

The architecture of IoTempower revolves around gateways and nodes. An IoTempower system contains one or many IoTempower gateways that are the central hubs (edge) where the software required to configure or manage an IoT system resides. These gateways can also manage other services such as Wi-Fi routers or MQTT brokers, enabling complete dataflow within the network along with configuration management (iotempire, 2019).

Nodes in the IoTempower system are typically microcontrollers like the Wemos D1 mini or NodeMCU, which connect wirelessly to the gateway. These nodes interact directly with the physical world through connected devices—sensors or actors (also called "things" within the IoT architecture) (iotempire, 2019). This setup allows for scalable and flexible IoT applications ranging from simple home automation systems to complex, multi-device configurations.

IoTempower is built to work on various hardware platforms but mainly Linux-based ones and wireless microcontrollers (ut-teaching-ulno, 2023). It supports several single-board Linux computers including Raspberry Pi (models 1-4 & Zero W), besides being compatible with microcontrollers like ESP8266/ESP32 etc. This wide range of supported hardware makes IoTempower a flexible choice for different IoT applications and environments.

IoTempower comes with multiple utils and scripts that make deploying & managing IoT systems much more manageable. These tools include setting up access points, performing OTA updates, and initializing new nodes. Some notable commands include (iotempire, 2019):

- **iot upgrade**: Updates IoTempower to the latest version.
- **iot install**: Installs or reinstalls the IoTempower environment.
- **console_serial**: Opens a serial console to view debug output from a connected node.
- **deploy**: Handles OTA updates for IoTempower nodes after software changes.

These tools are designed to be user-friendly and are accessible via a simple text-based menu, making the framework accessible even for those with limited technical background.

For Installing IoTempower, the scripts provided for different platforms such as Linux, MacOS, and even Android (Termux) are used. The installation process typically consists of running some basic commands that will install the necessary tools like Git or Curl first and then execute a script that sets up the IoTempower environment (ulno, 2017).

Furthermore, due to its design and features, IoTempower fits very well within educational settings where students or teachers can use it to construct their own IoT systems and learn about them. The framework's ability to embrace different hardware types as well as management tools ensures that users can experiment with scaling from small classroom projects through large interconnected IoT ecosystems (ut-teaching-ulno, 2023).

In outline, IoTempower provides:

- A user-friendly framework for IoT development.
- Combining powerful architecture.
- Broad hardware support.
- A suite of tools designed to simplify IoT deployment.

Its focus on education and ease of use makes it a valuable resource for anyone looking to delve into the world of IoT.

In this paper, we will study targeting of the test cases that will be useful in developing and implementing IoT frameworks with a special focus on the IoTempower framework. Testing is very important for Internet-of-Things (IoT) systems since they are complicated and can have various drawbacks and inefficiencies (Malik, B.H. et al, 2019). Our analysis for testing purposes has to be more detailed so as to identify and solve these issues that will lead to increased reliability of the entire IoTempower. IoTempower is a popular framework in education circles where students frequently manipulate different settings as well as hardware components. Nonetheless, it usually disrupts learning when such experiments produce system failures at class times, according to Ulrich Norbisrath.

The IoTempower system is commonly used in educational settings, where students tend to play around with settings and hardware. But this can lead to a lot of system crashes which are disruptive when they happen during class time. Our goal is not just to decrease the number of these failures, but also to detect and address them before they impact users.

For instance, according to Ulrich Norbisrath, there have been instances where a temperature sensor was completely broken and went unnoticed until it caused significant issues. Similarly, other components like LED setups and compilation processes have faced problems that needed fixing. Through systematic testing, we aim to experience these breaks earlier than the users do, allowing us to implement solutions proactively.

This analysis and corresponding testing aim to identify and address errors within systems and their inefficiencies, thus improving the overall reliability of IoTempower.The developers will have a better understanding of how full capability should be utilized after solving systematically. Thus, this thesis is meant to ensure that IoTempower remains powerful for both educational and practical IoT applications.

3. Related Work and Literature Review

The Internet of Things (IoT) has brought great automation levels, data-sharing capabilities and smart functions through the interconnection between devices and systems, which traditionally had no common ground for communication, let alone interaction (Udoh and Kotonya, 2018).

This rapid growth in technology therefore requires the creation of different frameworks or methods that would make it easy to develop, deploy, and manage IoT solutions. These frameworks and methods streamline the complexities of integrating different hardware and software components, ensuring seamless communication and operation (Uviase and Kotonya, 2018).

This section delves into the current research landscape and technologies for testing IoT frameworks. It covers aspects used by different entities towards realizing reliable functional performance systems under IoT environments, including regression testing commonly used in the software development process where there is a need to confirm changes made on current functionality do not negatively affect other parts; Also emphasizes on hardware integration testing which verifies compatibility among devices within an ecosystem — especially sensors connected via wireless networks or wired connections.

Additionally, some popular development frameworks, such as Tasmota and ESPHome, will be discussed; these are commonly used for device management automation purposes within the IoT community. For instance, Tasmota provides a powerful, flexible way of managing various devices through MQTT, HTTP, or Web UI protocols. At the same time, ESPHome enables automatic firmware creation for ESP8266/ESP32 boards based on YAML configs (tasmota.github.io, 2017), (ESPHome, 2024).

Moreover, the discussion extends to the advanced IoT testing services provided by Microsoft Azure. Azure's broad suite of tools and services supports the scalable and secure testing and management of IoT devices, facilitating real-time monitoring and automated testing processes. This ensures that IoT solutions can be developed and deployed highly, reliably, and efficiently (Microsoft.com, 2024).

By examining these frameworks and testing methodologies, this section aims to provide a detailed understanding of the current state of IoT testing technologies. It also aims to identify best practices and potential areas for improvement, ultimately contributing to enhancing the IoTempower framework's capabilities.

3.1. Tasmota and ESPHome

Tasmota and ESPHome are two popular frameworks used for IoT device management and automation (athom, 2024). Each of these frameworks has its own unique features that make them most suitable for testing and integration in various Internet of Things environments.

Tasmota is an open-source firmware that allows you to control lots of different types of IoT devices over multiple communication protocols such as MQTT, HTTP or web UI (Alhazmi, Khulud Alawaji and OConnor, 2022). It can work with sensors, switches, relays, etc., so it's great for many home automation projects. The framework is well-regarded for its robustness, mainly due to its extensive community support and frequent updates, continuously enhancing its functionality and reliability (Tasmota.github.io, 2024). Some researches conducted on Tasmota have proved that this particular software is really effective in terms of managing device communication and automation processes because it is very easy to use, even when dealing with various applications pertaining to the field (Alhazmi, Khulud Alawaji and OConnor, 2022)

Regarding testing, Tasmota currently lacks any formal testing procedures or frameworks. While investigating their repository revealed no evident testing mechanisms. This absence of testing even has been acknowledged in a GitHub issue (arendst, 2018).

Another quite powerful framework, which is also explicitly designed for ESP8266/ESP32 boards only, is called ESPHome. This one focuses heavily on simplicity and ease of use, such that users can define their device configurations using simple YAML files (ESPHome, 2024). The main advantage of this approach is that people can quickly set up and manage their IoT devices without spending much time on complicated software configurations. Home Assistant integration, among other things, makes it possible to create complete smart home systems where all components work together seamlessly within a single ecosystem (Debauche et al., 2020). Some research carried out so far found that ESPHome reduces development time while improving device interoperability; hence, it is highly recommended by many professional enthusiasts working within different areas related to the Internet of Things (Himeur, Y et al., 2021).

For testing new device configurations, ESPHome uses a structured and automated process. When a new device is added, its configuration is defined in a YAML file, specifying necessary parameters such as GPIO pin assignments, communication protocols, and sensor types. ESPHome offers detailed documentation and a variety of pre-defined device components to simplify the configuration process. Once the YAML configuration is complete, the firmware is compiled and uploaded to the device using the ESPHome command-line tool or through integration with platforms like Home Assistant (ESPHome, 2024).

The testing in ESPhome has two different levels, which, in this study, we are lacking one. Since ESPhome core functionalities were written in Python, using Pytest to write unit tests is easier (ESPHome, 2024). That is technically very hard to test shell scripts, which cover most of the IoTempower framework. The only possible way to write an individual test for each shell script is to test the expected outcome, which can not be generalized easily since each shell script acts differently and has different purposes. It can be done in the future by targeting iotempower specifically. ESPHome is lucky in this scenario. The next level of testing is similar to our compilation testing. Since ESPHome is configured by YAML files, each test case is saved as a separate YAML file. Example code has been presented below (ESPHome, 2024):

```
esphome:
 name: test
 platform: ESP8266
 board: d1_mini_lite

wifi:
 ssid: SomeNetwork
 password: SomePassword

button:
 - platform: wake_on_lan
   target_mac_address: 12:34:56:78:90:ab
   name: wol_test_1
   id: wol_1
   internal: true
 - platform: wake_on_lan
```

```yaml
    target_mac_address: 12:34:56:78:90:ab
    name: wol_test_2
    id: wol_2
    internal: false
```

Then they use their core functionality to generate CPP files and assert that some lines are present, example code is below (ESPHome, 2024):

```python
def test_button_sets_mandatory_fields(generate_main):
    """
    When the mandatory fields are set in the yaml, they should be set in main
    """
    # Given

    # When
    main_cpp = generate_main("tests/component_tests/button/test_button.yaml")

    # Then
    assert 'wol_1->set_name("wol_test_1");' in main_cpp
    assert "wol_2->set_macaddr(18, 52, 86, 120, 144, 171);" in main_cpp
```

```python
@pytest.fixture
def generate_main():
    """Generates the C++ main.cpp file and returns it in string form."""

    def generator(path: str) -> str:
        CORE.config_path = path
        CORE.config = read_config({})
        generate_cpp_contents(CORE.config)
        print(CORE.cpp_main_section)
        return CORE.cpp_main_section

    yield generator

    CORE.reset()
```

It should also be mentioned that ESPHome completely lacks any hardware testing and depends on the community to report bugs in a new release (esphome.io, 2024).

In summary, Tasmota and ESPHome provide strong IoT device management and automation solutions (athom, 2024). Tasmota's adaptable and essential community backing make it suitable for various applications. Together, these frameworks contribute significantly to advancing IoT technologies by simplifying device integration and management, thereby supporting the rapid growth and innovation within the IoT ecosystem.

3.2. Microsoft Azure IoT Testing Services

Microsoft Azure offers testing services for IoT applications, giving a secure foundation for connecting, monitoring and managing IoT devices. The IoT testing framework of Azure is broad such as device simulation, automated testings or real-time tracking that are necessary to ensure the reliability and performance of the Internet of Things deployments. Apart from its compatibility with multiple programming languages and integration with other Microsoft services make it even more attractive for large-scale enterprise IoT projects (Microsoft.com, 2024).

One important feature is device simulation which allows developers to create virtual representations of physical devices. These simulations can imitate real-world conditions and scenarios thus providing a controlled environment where to test device interactions, data flow or system responses without involving physical hardware. This particularly comes in handy when doing stress tests or validating behavior of the system under different operational loads.

In Azure IoT automated testing involves implementing continuous integration and continuous deployment (CI/CD) pipelines using tools like Azure DevOps. These pipelines automate building, testing and deploying IoT applications so that any changes made or updates done on the system are checked thoroughly before being released into production environment.Automated tests can be unit tests, integration tests or end-to-end tests which cover all aspects of an application ranging from individual components up to the full system (Microsoft.com, 2024).

Azure IoT services provide strong support through remote management capabilities together with real-time monitoring when adding new hardware components. For instance new devices can be registered and configured through Azure IoT Hub which acts as a central point for managing all IOT devices (Forsström and Jennehag, 2017). Developers can use IOT hub to push firmware updates ,configure device settings as well as monitor device health & performance in real-time.

The Azure IoT Device Provisioning Service (DPS) facilitates the secure and scalable deployment of new devices. DPS automates the process, ensuring that devices are securely connected to the IoT Hub and configured according to predefined policies and settings (Karmakar et al., 2022). This process includes validating device identities, assigning devices to appropriate IoT Hubs, and applying initial configurations, all of which are critical for maintaining the security and integrity of the IoT system (Microsoft.com, 2024).

To further validate the integration of new hardware, edge testing is utilized using Azure IoT Edge. IoT Edge extends cloud capabilities to local devices thereby enabling data processing at the edge and running machine learning models on-premises. This allows for real time testing and validation of new hardware within operational environment (Jensen, 2019).

Azure IoT solutions provide a unique testing method that focuses on evaluating the solutions built on top of its service rather than the framework itself. Given the complexity of these solutions, Azure IoT offers the ability to simulate sensor readings, allowing developers to test their code and ensure the program functions as intended. These simulated readings may continue for hours, days, or even weeks until they stop manually. This type of test may come in handy in the future when the user completes testing the framework itself; then, they can focus on creating a framework that is meant to be used by end-users (Microsoft.com, 2024).

Overall, this section underscores the importance of reliable testing processes in IoT development, highlighting how Tasmota, ESPHome, and Microsoft Azure IoT Testing Services contribute to reliable and efficient IoT deployments. These insights are crucial for advancing the IoTempower framework, supporting its goal to simplify IoT development and deployment.

4. Case Study

This chapter focuses on implementing a test framework for IoTempower system in an actual, real-life situation. In this case study, we will look at some application scenarios specifically explaining how did I apply it and what were the results when testing performance based on different criteria. In addition to the technical evaluations, this study employed semi-structured interviews as a qualitative research method. The interviews were conducted with professionals who have experience in IoT development and testing, specially in the IoTempower framework. The questions were designed to elicit detailed feedback on specific aspects of the framework, such as installation, compilation, deployment, and hardware testing. (see Appendix)

4.1. Case Application

The primary aim of this research was to study testing frameworks in the context of IoT development, exploring how testing is generally conducted and how these practices can be applied to real-world scenarios. IoTempower emerged as an ideal candidate for implementing and evaluating a testing framework due to its existing reliability challenges and the opportunity to enhance its testing capabilities. While ESPHome provided a good starting point with its approach to testing, it was not without its shortcomings—particularly in the area of hardware testing, which remained a significant gap in their framework (esphome.io, 2024). This made IoTempower an excellent subject for exploring how comprehensive testing, including both software and hardware aspects, could be implemented to improve overall reliability. Moreover, the supervisor of this research is one of the main authors of IoTempower which led us to know about the framework and its needed improvements.

4.2. Design

In order to solve the challenges mentioned above, docker containers were used as an experimental method for encapsulating IoTempower environments. Through virtualization and containerization, Docker simplifies software dependency management by allowing developers package applications together with all the necessary components into images that will run anywhere (Moreau, D. et al., 2021). With this approach, every supported platform becomes a

uniform environment for testing thus reducing bugs caused by differences in systems and increasing test reproducibility.

When I started analyzing the possible test cases for the IoTempower framework, it became apparent that the installation process was more complex than it initially seemed. Typically, for other software or frameworks, there is a single point of entry for installation. However, IoTempower requires multiple layers and steps for installation. Initially, I failed by missing some dependencies, highlighting the need for tests to ensure all chosen dependencies are correctly installed.

Although I planned testing the installation process, it remained challenging. There was a risk of IoTempower's dependencies conflicting with the computer's dependencies, despite efforts to keep IoTempower as a local environment. Consequently, I decided to create a Docker image to facilitate testing and increase overall reliability. After the initial Docker versions, Ulrich Norbisrath, continued this approach, providing a single point of entry. Docker proved user-friendly, allowing for easy recovery by starting a new container if something was accidentally deleted or changed. Moreover, previously, we were not able to install IoTempower natively on Apple and Termux devices; it was only supported by Debian and Arch distributions of Linux. With the introduction of Docker to the system, this limitation was solved.

I observed that, like most frameworks, IoTempower involves several steps: creating configuration files, compiling them into binaries, and flashing them to the hardware device. I decided to create an initial compilation test, as this does not require any external hardware and can be done within the Docker container.

To perform this, I first needed to create a node locally and generate configuration files for the appropriate sensors. In the configuration process, I specify the board's name in the `node.conf` file (Figure 2), using one of the boards supported by IoTempower. In the `setup.cpp` file (Figure 3), the necessary code for the sensors is written. My analysis showed that most sensors documented can be used on all boards, but some devices like mfrc522 (RFID reader), rgb_single, etc., are either not supported by all boards or supported by only one board. Additionally, different

boards have varying PIN layouts, complicating the configuration. To address this, we used IoTempower's core functionality to name the pins (e.g., IOT_TEST_DIGITAL) and define different pins for different boards in their relevant source codes. This allows us to write a general syntax for testing all boards.

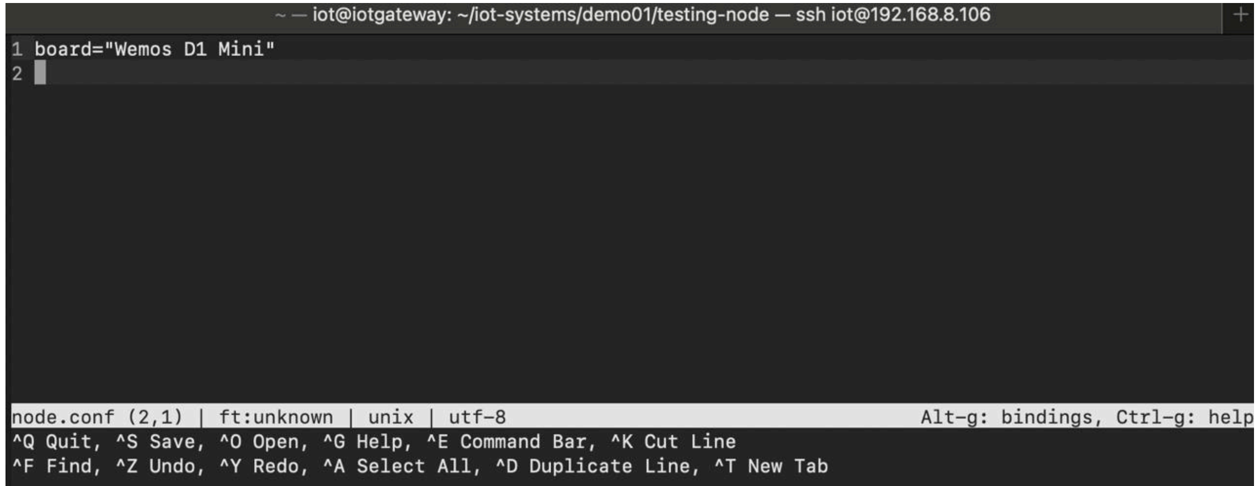


Figure 2. Content of the `node.conf` file

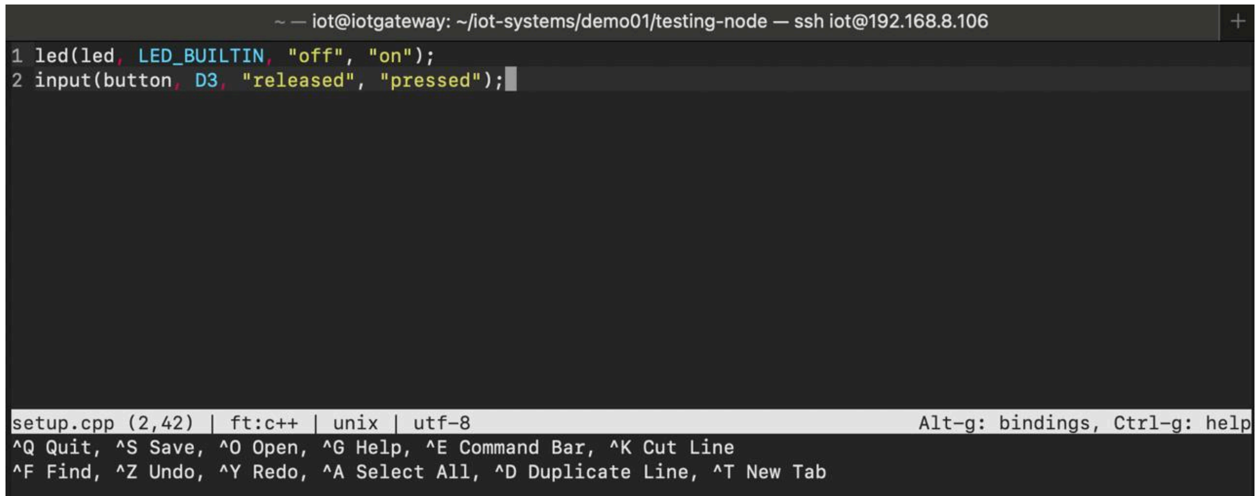


Figure 3. Content of the `setup.cpp` file

Compiling typically takes around five seconds and it doesn't require any additional external devices. However, for deployment testing, the conditions change. Deploying without Docker does not require an external device, but we previously discussed the issues with this approach. When using Docker, accessing the host machine's USB devices is challenging, making deployment nearly impossible. To resolve this, I decided to use a Raspberry Pi. The Raspberry Pi

is well-suited for IoT development and testing because it is small, uses little power, and is very flexible. However it's very slow compared to a personal computer setup.

In this project, the Raspberry Pi is crucial for its role as an edge gateway in testing the combination of new hardware and software setups. Using the Raspberry Pi, we can simulate real-world IoT situations and check how well our system parts work under different conditions. Additionally, its ease of setup with a pre-configured Raspberry Pi image simplifies the process considerably, though it is not the focus of testing. Despite its pros and cons, it is a reliable solution. The Raspberry Pi will be connected to the local network, and a tester will connect to it via SSH to execute commands. The SSH connection must be configured in advance using SSH key exchange, avoiding exposure of security details in the tests and simplifies remote orchestration. The IP/DNS address of the gateway should be configurable, as the Raspberry Pi registers itself under the DNS name "iotgateway" in the network, but this might not work with some routers. After establishing the SSH connection, the tester device (personal computer) was able to deploy programmatically by sending SSH commands. It can be confirmed the deployment's success by listening to the MQTT channel, as the deployed nodes publish a standard status payload. Since the Raspberry Pi does not expose its MQTT port externally, the local port of the Pi should be bound to our device's local port using SSH tunneling. Only after this the code may connect to MQTT channel externally. By capturing the MQTT payload, it can be asserted that the deployment is successful.

Even though we assert that the deployment successfully finished, the user can not be sure that the deployed code is functional within the hardware. For this, hardware testing is needed. The hardware's functionality can be checked using another hardware device. For this test, at least two pieces of hardware are needed: one for testing and one being tested. Deploy is done to both devices as defined in the test case, with initial steps similar to deployment testing. Then, we subscribe to the topic of the tested device via the MQTT channel, record the initial status, send a command to the tester device, and observe if the status of the tested device changes. This allows us to directly assert that the flashed code affects the hardware.

4.3. Test Cases Implementation

As it was mentioned earlier testing is a critical phase of this project, and it is for ensuring that all components of the IoTempower framework function as expected and that new features integrate smoothly without introducing errors. A well-structured testing framework is essential to catch potential issues early before impacting the end-user and maintain the reliability of the system.

Before starting the tests, I needed to decide on which testing framework to use. Since IoTempower is primarily written in shell scripts and also uses Python in many areas, it was reasonable to assume that its contributors are familiar with Python. Moreover, Python is a relatively easy programming language to learn and is already known by many. Given this, Pytest, which is the most popular and widely-used testing framework in Python, stood out as an ideal choice for my testing needs.

4.3.1. Test Case 1 - Installation Testing

Implementing a robust testing process requires a clear understanding of the dependencies and the tools involved. Before initiating the test, it was essential to gather all relevant information about the dependencies to ensure accurate testing.

To start the test, I needed the names of the dependencies, the modules they belong to, and the corresponding package managers. This information is stored in the `packages` variable in `tests/conf_data.py`. The `packages` variable is a collection of dictionaries, each containing three key-value pairs. The `name` key specifies the package's name, the `package_manager` key identifies the source from which the package is obtained, and the `module` key groups packages that can be selected during installation and subsequently verified.

```
# The list of the packages for installation test
packages = [
    {"name": "git", "package_manager": "binary", "module": "general"},
    {"name": "jq", "package_manager": "binary", "module": "general"},
    {"name": "make", "package_manager": "binary", "module": "general"},
    {"name": "curl", "package_manager": "binary", "module": "general"},
```

```
    {"name": "mosquitto_sub", "package_manager": "binary", "module": "general"},
    {"name": "mosquitto_pub", "package_manager": "binary", "module": "general"},
    {"name": "node", "package_manager": "binary", "module": "general"},
    {"name": "terminal-kit", "package_manager": "npm", "module": "general"},
    {"name": "g++", "package_manager": "binary", "module": "cloud_commander"},
    {"name": "gritty", "package_manager": "npm", "module": "cloud_commander"},
    {"name": "cloudcmd", "package_manager": "npm", "module": "cloud_commander"},
    {"name": "node-red", "package_manager": "npm", "module": "node_red"},
    {"name": "caddy", "package_manager": "binary", "module": "caddy"},
    {"name": "mosquitto", "package_manager": "binary", "module": "mosquitto"},
]
```

End-user choices are saved as a JSON file during installation to accurately track the installation dependencies. After the installation, this JSON file is read to verify that only the dependencies selected by the end-user are correctly installed. Additionally, the packages in the "general" module are non-optional and must be present regardless of user selection.

```
with open(installation_options_file, "r") as f:
    installation_options = json.load(f)

packages_to_test = []
for key, value in installation_options.items():
    if value.lower() == "1":
        packages_to_test += [tuple(package.values()) for package in packages if
package["module"] == key]
```

Since there are only two package managers in use at the moment, I use `which` command to check binary packages, and `npm list` for node js packages. During the implementation of this test there was not any generalized way to check all the packages, however currently it is developed and functional by Ulrich Norbisrath.

```
@pytest.mark.parametrize("package_name, package_manager, module",
packages_to_test)
def test_eval(package_name, package_manager, module) -> None:
    if package_manager == "binary":
```

```
        command = f"which {package_name}"
    elif package_manager == "npm":
        command = f"cd {local_dir}/nodejs && npm list {package_name}"
    else:
        raise NotImplementedError("Only installed binaries and npm packages are
supported for now")
    result = subprocess.run(command, shell=True)
    assert result.returncode == 0, f"Cannot find {package_name} in your system
which is needed for {module}"
```

4.3.2. Test Case 2 - Compilation Testing

For the compilation tests, I used IoTempower's `compile` command. The process began with creating the node folder, followed by editing the `{node_directory}/setup.cpp` and `{node_directory}/node.conf` files to cover various test cases. These files required specific combinations for testing, as IoTempower supports a limited number of IoT boards. The supported board lists are stored in the `conf_data.py` file.

List of supported boards:

```
boards = [
    "wemos_d1_mini",
    "sonoff",
    "nodemcu",
    "olimex",
    "m5stickc",
    "wroom_02",
    "esp-m",
    "sonoff_pow",
    "esp8266",
    "esp32",
    "esp32minikit",
]
```

Most sensors are compatible with all boards, so these sensors have been listed as well:

```
devices = [
    {"device_name": "input", "example_syntax": 'input(lower, IOT_TEST_INPUT, 
"released", "pressed");'},
    {"device_name": "output", "example_syntax": 'led(yellow, IOT_TEST_OUTPUT, 
"turn on", "turn off");'},
    {"device_name": "analog", "example_syntax": "analog(example_name);"},
    {"device_name": "bmp180", "example_syntax": "bmp180(example_name);"},
    {"device_name": "bmp280", "example_syntax": "bmp280(example_name);"},
    {"device_name": "dallas", "example_syntax": "dallas(example_name, 
IOT_TEST_DIGITAL);"},
    {"device_name": "gyro6050", "example_syntax": "gyro6050(example_name);"},
    {"device_name": "gyro9250", "example_syntax": "gyro9250(example_name);"},
    {"device_name": "gesture_apds9960", "example_syntax": 
"gesture_apds9960(example_name);"},
    {"device_name": "sgp30", "example_syntax": "sgp30(example_name);"},
    {"device_name": "edge_counter", "example_syntax": 
"edge_counter(example_name, IOT_TEST_DIGITAL);"},
    {"device_name": "dht", "example_syntax": "dht(example_name, 
IOT_TEST_DIGITAL);"},
    {"device_name": "ds18b20", "example_syntax": "ds18b20(example_name, 
IOT_TEST_DIGITAL);"},
    {"device_name": "servo", "example_syntax": "servo(example_name, 
IOT_TEST_DIGITAL, 800);"},
    {"device_name": "acoustic_distance", "example_syntax": "hcsr04(distance, 
IOT_TEST_DIGITAL, IOT_TEST_DIGITAL_2);"},
]
```

To generate all the necessary combinations for testing, I utilized Python to automate the process.

```
isolated_combinations_to_test = [
    (board, device["device_name"], device["example_syntax"]) for device in 
devices for board in boards
]
```

However, some hardware is supported by only a limited number of boards, so I defined these specific combinations manually.

```
isolated_combinations_to_test += [("wemos_d1_mini", "rgb_single",
"rgb_single(r0, D6, D5, D0, true);")]
```

As we discussed previously, not all boards share the same pin layout. For this, we defined testing pins in the source file. For example, this is the pin definitions of “nodemcu” board:

```
// test pins for IoTempower
#define IOT_TEST_INPUT D3
#define IOT_TEST_OUTPUT D4
#define IOT_TEST_DIGITAL D1
#define IOT_TEST_DIGITAL_2 D2
```

For testing this theory several discussions have been made with Ulrich Norbisrath and the solution has been presented the result of these discussions. The result is writing some initial pin definitions and testing different boards and sensors. During this process, I discovered that some sensors and boards were already broken. For instance, both the `m5stickc_plus` and `m5stickc` boards had missing header issues, which partially has been fixed. Additionally, we found a simple typo in the name of `esp_m`, which was incorrectly listed as `esp-m`, and we corrected it. Another issue involved the `dev-gyro.h` file, which was mistakenly imported as `dev-gyro.h.h`. We also found that logging functions were called with a float parameter, which needed to be cast to a string first. Finally, we corrected the names of both `dev_gesture_apds9960.h` and `dev_gesture_apds9960.cpp`. With Ulrich Norbisrath’s assistance, these issues were resolved, making IoTempower more reliable.

I then created the necessary `node.conf` and `setup.cpp` files from the test cases and compiled all the examples one by one using IoTempower's "compile" command, ensuring the successful completion of each compilation.

```
@pytest.mark.parametrize("board_name, device_name, example_syntax",
isolated_combinations_to_test)
def test_compilation_isolated(board_name: str, device_name: str,
example_syntax: str):
```

```
    node_directory = f"{test_dir}/node"
    os.makedirs(node_directory, exist_ok=True)
    with open(f"{node_directory}/node.conf", "w") as f:
        f.write(f'board="{board_name}"')
    with open(f"{node_directory}/setup.cpp", "w") as f:
        f.write(example_syntax)
    try:
        subprocess.check_call(["compile"], cwd=node_directory)
    except subprocess.CalledProcessError:
        assert False, f"Failed to compile {device_name} for {board_name}"
    finally:
        shutil.rmtree(node_directory, ignore_errors=True)
        if remove_cache_flag.lower() == "true":
            shutil.rmtree(cache_dir, ignore_errors=True)
```

4.3.3. Test Case 3 - Deployment Testing

To accomplish this testing the Raspberry Pi will be connected to the local network, and I will connect to it via an SSH connection to execute commands. To establish the SSH connection, I used Python's `fabric` library, setting it up as a `fixture` for deployment testing.

```
@pytest.fixture(scope="module", autouse=False)
def ssh_client():
    conn = Connection(host=f"{default_username}@{gateway_host}",
connect_timeout=5)
    try:
        conn.run("echo Hello IoTempower Testing")
        yield conn
        conn.close()
    except socket.timeout:
        raise Exception(f"Failed to connect to {gateway_host}")
    except AuthenticationException:
        raise Exception(
            f"Authentication failed for {gateway_host}, "
            f"please make sure you have right configuration,
https://www.ssh.com/academy/ssh/copy-id"
```

```
        )
```

First, I created a node on the Raspberry Pi device using IoTempower's `create_node_template` command:

```
@pytest.fixture
def new_node(ssh_client):
    ssh_client.run(f"rm -rf {nodes_folder_path}/testing-node")
    ssh_client.run(f"cd {nodes_folder_path} && iot x create_node_template
testing-node")
```

nodes_folder_path - is a configurable path for node creation.

As with all our tests, I needed specific test cases for this deployment test. Similar to other test cases, the deployment testing cases are stored in the `conf_data.py` file.

```
cases_for_deployment: list[tuple[str, list[str], list[tuple[str, str]]]] = [
    # Example:
    # (
    #      "device_name",
    #      ["example_syntax_line1", "example_syntax_lineN"],
    #      [("{mqtt_topic_name1}", "{default_message1}"), ("{mqtt_topic_nameN}",
"{default_messageN}")],
    # ),
    ("Wemos D1 Mini", ['led(led, LED_BUILTIN, "off", "on");'],
[("testing-node/led", "on")]),
    ("Wemos D1 Mini", ['input(button, D3, "released", "pressed");'],
[("testing-node/button", "released")])]
```

The test cases are list of tuples where each tuple represents a separate test. Each tuple contains:

- The board name for the `node.conf` file.
- A list of lines for the `setup.cpp` file.
- A list of tuples, each containing an MQTT topic and the expected payload from that topic.

Sometimes content of `setup.cpp` file can be too complex, that's why we create that file locally then transferring to raspberry pi using sftp client.

```
sftp_client.putfo(generate_file(lines=commands),
f"{nodes_folder_path}/testing-node/setup.cpp")
ssh_client.run(f"echo 'board=\"{device_name}\"' >
{nodes_folder_path}/testing-node/node.conf")
```

After the configuration files are prepared, user is ready for deployment. For this purpose, I use IoTempower's `deploy serial` command. This command also accepts an argument specifying the address of the connected device. The Raspberry Pi has four USB ports, each with a specific name. The tester must define the address of the target device in the `conf_data.py` file. The default value is:

```
deploy_device_address =
"/dev/serial/by-path/platform-fd500000.pcie-pci-0000:01:00.0-usb-0:1.1:1.0-port
0"
```

Here is representation of raspberry pi USB ports and their addresses:

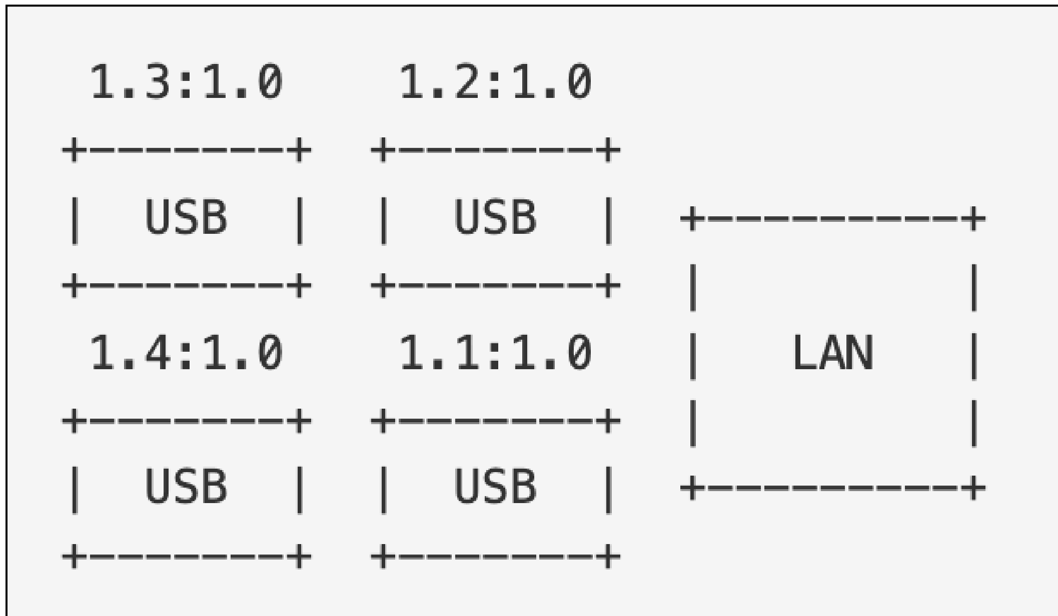


Figure 4. Layout of Raspberry Pi USB Ports and Their Corresponding Addresses

This is real-life view of Raspberry Pi USB ports and their Addresses:

Figure 5. Real-Life View of Raspberry Pi USB Ports and Their Addresses

Another essential item in deployment testing is MQTT client for python. We used mqtt client `paho` library. As it's stated before, Pi doesn't expose it's MQTT port. We used `sshtunnel` library of Python to make this map internal port to our configurable local port:

```
@pytest.fixture(scope="module", autouse=False)
def mqtt_client():
   mqttc = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
   with sshtunnel.open_tunnel(
       gateway_host,
       ssh_username=default_username,
       remote_bind_address=("127.0.0.1", 1883),
       local_bind_address=("127.0.0.1", local_bind_mqtt_port),
   ):
       mqttc.connect("127.0.0.1", local_bind_mqtt_port, 60)
       yield mqttc
       mqttc.disconnect()
```

After MQTT client is created I am subscribing to the test case topics to receive messages. To do that, I listen to mqtt topics for configurable amount of time then assert that the I have received expected status messages which are defined in test cases and assigned IP by raspberry PI.

The whole code look like this:

```
@pytest.mark.parametrize("device_name, commands, expected_messages",
cases_for_deployment)
def test_deploy(new_node, ssh_client, sftp_client, mqtt_client, device_name,
commands, expected_messages):
   sftp_client.putfo(generate_file(lines=commands),
f"{nodes_folder_path}/testing-node/setup.cpp")
   ssh_client.run(f"echo 'board=\"{device_name}\"' >
{nodes_folder_path}/testing-node/node.conf")
   ssh_client.run(f"cd {nodes_folder_path}/testing-node && iot x deploy serial
{deploy_device_address}")

   # Subscribe to node topics for status messages
   for topic, message in expected_messages:
       mqtt_client.subscribe(topic)
   # Subscribe to IP definition topic
   mqtt_client.subscribe(topic="iotempower/_cfg_/testing-node/ip")

   # Listen for status messages of tested node and write them to messages list
   messages = mqtt_listen(mqtt_client)
   assert check_for_presence(all_messages=messages,
expected_messages=expected_messages)
   assert check_ip_presence(all_messages=messages, node_name="testing-node")
```

4.3.4. Test Case 4 - Hardware Testing

The implementation of hardware testing uses the same functionalities as the deployment test, with only minor logical adjustments that differentiate it from the deployment tests. The primary difference lies in the structure of the test cases:

```
cases_for_hardware = [
    # Example:
    # (
    #     ("tested_device_name", "tester_device_name"),
    #     (
    #         ["example_syntax_tested_line1", "example_syntax_tested_lineN"],
    #         ["example_syntax_tester_line2", "example_syntax_tester_lineN"],
    #     ),
    #     (
    #         [
    #             ("mqtt_topic_tested1_name",
"mqtt_tested1_initial_state_message"),
    #             ("mqtt_topic_testedN_name",
"mqtt_testedN_initial_state_message"),
    #         ],
    #         [
    #             ("mqtt_topic_tester1_name", "mqtt_tester1_command"),
    #             ("mqtt_topic_testerN_name", "mqtt_testerN_command"),
    #         ],
    #         [
    #             ("mqtt_topic_tested1_name", "mqtt_tested1_message"),
    #             ("mqtt_topic_testedN_name", "mqtt_testedN_message"),
    #         ],
    #     ),
    # ),
    (
        ("Wemos D1 Mini", "Wemos D1 Mini"),
        (
            ['input(input_tested, D2, "high input", "low input");'],
            ['output(output_tester, D5, "on", "off");'],
        ),
        (
            [("hardware-testing/tested_node/input_tested", "low input")],
            [("hardware-testing/tester_node/output_tester/set", "on")],
            [("hardware-testing/tested_node/input_tested", "high input")],
        ),
    )
```

As shown in the example, I now have two different configurations for two different nodes. Therefore, I create two separate nodes, configure them individually, and deploy them to the connected devices, which are defined by address using the same logic as in the compilation tests. After successful deployments, I subscribe to the node topics and listen for a configurable amount of time to assert the initial state message of the tested node. Next, I publish a trigger to the tester node and start listening again for the status change in the tested node. Finally, I assert that the status of the tested device has indeed changed in response to the tester node's input. The hardware configuration of the example above is illustrated the picture below. It's visible that D5 pin of the tester device Wemos D1 mini, is connected to D2 pin of the tested device, which is another Wemos D1 mini board.

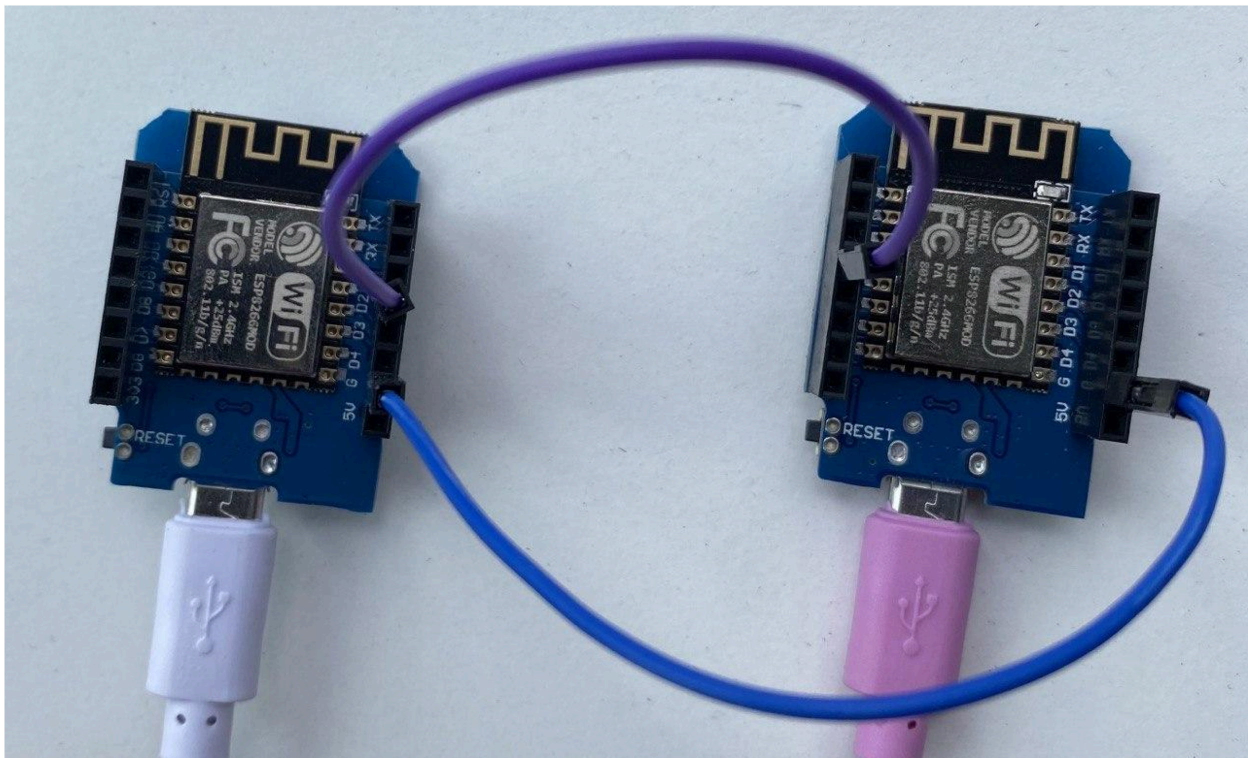


Figure 6. Hardware Testing Example Configuration: Wemos D1 Mini Board

The complete code:

```
@pytest.mark.parametrize("device_names, example_syntaxes, mqtt_messages",
cases_for_hardware)
def test_hardware(mqtt_client, new_nodes, ssh_client, sftp_client,
device_names, example_syntaxes, mqtt_messages):
    # Extracting configurations from tuple objects
    tested_device_name, tester_device_name = device_names
    example_syntax_tested, example_syntax_tester = example_syntaxes
    initial_tested_status, tester_messages_to_send, tested_messages_to_receive =
mqtt_messages

    tested_node_path =
f"{nodes_folder_path}/hardware-testing/{tested_node_name}"
    tester_node_path =
f"{nodes_folder_path}/hardware-testing/{tester_node_name}"

    # Assigning device names of tested and tester nodes
    ssh_client.run(f"echo 'board=\"{tested_device_name}\"' >
{tested_node_path}/node.conf")
    ssh_client.run(f"echo 'board=\"{tester_device_name}\"' >
{tester_node_path}/node.conf")

    sftp_client.putfo(generate_file(lines=example_syntax_tested),
f"{tested_node_path}/setup.cpp")
    sftp_client.putfo(generate_file(lines=example_syntax_tester),
f"{tester_node_path}/setup.cpp")

    # Deployment
    ssh_client.run(f"cd {tested_node_path} && iot x deploy serial
{tested_device_address}")
    time.sleep(3)
    ssh_client.run(f"cd {tester_node_path} && iot x deploy serial
{tester_device_address}")

    # Subscribe to tested node topic for initial status message
    for topic, message in initial_tested_status:
        mqtt_client.subscribe(topic)
```

```
    # Listen for initial status message of tested node and write them to
messages list
    messages = mqtt_listen(mqtt_client)
    assert check_for_presence(all_messages=messages,
expected_messages=initial_tested_status)

    # Sending command through MQTT
    for topic, payload in tester_messages_to_send:
        mqtt_client.publish(topic, payload)

    # Listening for expected status message of tested node after command
    # Subscription to the topics and on_message functionality was written in
upper lines
    messages = mqtt_listen(mqtt_client)
    assert check_for_presence(all_messages=messages,
expected_messages=tested_messages_to_receive)
```

5. Evaluation

The primary goal of this project was to implement a testing framework that is both configurable and user-friendly. While working on the project, it became evident that the installation and management of dependencies for IoTempower were more complicated than predicted. Installing through several entry points with multiple layers of dependencies called for a robust testing framework to ensure everything works well after installation. This prompted the development of a test suite that has been integrated directly into the IoTempower codebase.

Documentation was created to make the testing framework usable. (see Appendix) It offers an overview for new and existing contributors of IoTempower, guiding them through their changes in functionalities or the addition of new ones as they navigate through tests.

Different test cases were created during the implementation phase and stored in the conf_data.py file. These test cases cover installation, compilation, deployment, and hardware. This process was documented so that users who come across it later will be able to understand easily how these tests can be used or extended.

For instance, there is a script called test_installations.py, which checks if all selected dependencies have been installed correctly using an installation.json file generated during the installation process, thus making sure only necessary dependencies required by IoTempower are installed while ignoring others.

To test the compilation process, we wrote another script named test_compile.py , which checks whether node configurations can be compiled into binary files for devices. It uses lists of boards and devices to create all possible combinations, even when dealing with the most complex configurations. During this phase, I worked with Ulrich Norbisrath, where we tested initial pin definitions and different board/sensor combinations; this collaboration brought out some issues related to certain sensors/boards, which were fixed, hence improving the reliability of IoTempower.

In order to do deployment testing, I used Raspberry Pi as my test environment, whereby I connected it to the local network over SSH. The script requires Raspberry Pi to be configured with the right SSH settings plus hostname so that once it is deployed on target devices, nodes will start communicating with each other immediately without any further configuration. After deploying, the script subscribes to the MQTT channel to verify whether expected messages are published under the correct topics.

Hardware tests followed the same pattern but involved more meant to confirm devices' physical operation devices. The test_hardware.py script tests if the deployed code works on hardware by sending commands to a tester device and checking for an expected response from the tested device. The script subscribes to the MQTT channel and monitors status changes, thus ensuring that hardware responds correctly to commands.

During my work on this testing framework, there were several challenges, including the inability to access USB devices from within the Docker container, among others, related to the Raspberry Pi setup, but they were all addressed, leading to the creation of a reliable and practical test suite for IoTempower across different environments.

The testing framework implemented in this project has proven to be a valuable addition to IoTempower. It ensures thorough application testing, which includes installation, compilation, deployment, and hardware functionality. The documentation created with the framework guides future users, making it easier for them to maintain and expand the testing process as IoTempower grows.

IoTempower reliability and user-friendliness were enhanced through the knowledge acquired from this process and working together to solve problems and better frameworks. The project has demonstrated the importance of a structured testing approach, and the tools and methods developed here will continue to support the ongoing development of IoTempower.

6. Discussion

As in all software frameworks the are no wrong or right implementations but more intuitive and reliable ones. Our interviews supported the idea that some of our design decision are open to discussion for improvement from different aspects. During discussions with Ulno, it was noted that the computational resources of the Raspberry Pi are limited, which can extend the overall testing time. One suggestion was to compile the code locally and then transfer the output files to the Raspberry Pi, potentially speeding up the testing process. This approach could be explored in future work to determine whether it offers significant time savings.

In the design phase, it was decided that the most effective way to verify successful deployment was by listening to the MQTT channel for status messages. This method was chosen because MQTT messages are structured, standardized, and easy to parse. However, while serial data provides more detailed information about the deployment process, it also presents several challenges, such as complexity in parsing and managing the data.

During our discussion with Ulrich Norbisrath, another point that emerged was that the test cases for hardware and deployment tests could be more straightforward if we unify all node configurations into a single tuple instead of logically grouping board names, syntaxes, etc. The

proposed tuple structure would be: [({board_name}, {example_syntax}, {expected_mqtt_message}), ({second_board_name}, {second_syntax}, {second_message})].

The choice of containerization tool was another topic. Even though Docker proved to be one of the best tools in the market, there are still competitors which can be more secure considering that Docket daemon needs to be run as a root process. Podman is favored for its daemonless architecture, which improves security by eliminating the need for a background service running as root. It also offers Docker-compatible commands, making it easier to switch without significant retooling.

7. Conclusion and Future Work

This study has examined the role of testing and testing frameworks within the realm of the Internet of Things (IoT), with a specific focus on the IoTempower framework. Through the case study of IoTempower, I explored the challenges and intricacies involved in developing robust and effective testing frameworks for IoT applications. The research highlighted the critical need for comprehensive testing strategies to ensure the reliability, scalability, and functionality of IoT systems, which are becoming increasingly complex.

The findings emphasize the importance of structured testing methodologies in the IoT domain, particularly when integrating new hardware and software components into existing systems. By leveraging Docker for containerization and Raspberry Pi for real-world simulation, the study demonstrated how these tools can significantly enhance the testing process, leading to more stable and dependable IoT frameworks. The insights gained from this research not only contribute to the understanding of testing frameworks in IoT but also provide valuable guidance for future improvements in IoTempower and similar systems. The following part outlines specific areas for future work, focusing on expanding and refining these frameworks to keep pace with the evolving demands of IoT development.

- **Optimizing Testing Time:** The ultimate goal of these tests is to catch bugs before they are merged into the main branch. To achieve this, relevant CI/CD pipelines could be established. Modern Git repositories allow for the use of agent-client setups, where a separate machine runs the pipelines. Hardware configurations could be set up in a fixed location, allowing the CI/CD pipelines to test the code on actual hardware. However, this approach may introduce additional overhead, as testing times are already lengthy and will likely increase with the addition of more test cases. This could lead to delays in the development process. Additionally, hardware reliability is a concern, as physical issues could be mistaken for software problems. Since the testing setup would be remote, fixing such issues could take time.

- **Expanding Test Coverage:** Future work could involve adding more test cases for deployment and hardware testing, as well as completing the remaining sensor and device tests in the compilation phase.
- **Improving Efficiency:** During interviews, a suggestion was made to reduce testing time by listening to the MQTT topic only until the expected message is received, rather than for the entire configurable period. This approach could allow the testing process to stop earlier if the desired result is already achieved.

- **User-Friendly Interface:** Creating a user interface, whether a command-line interface (CLI) or a graphical user interface (GUI), could make the testing framework more accessible and user-friendly for developers.

8. Acknowledgements

This thesis would not have been possible without the support and contributions of many individuals, to whom I am deeply grateful. I, Araz Heydarov, would like to express my heartfelt thanks to everyone who has played a part in the completion of this research.

First and foremost, I extend my sincere gratitude to my supervisor, Ulrich Norbisrath, who is also the founder of IoTempower. His guidance, expertise, and encouragement have been invaluable throughout the entire process. Despite his busy schedule, he was always available to provide support, from the early stages of defining the research direction to the detailed discussions we had during our interviews. His insights were crucial in shaping the content and structure of this thesis, particularly in the areas of testing frameworks and IoT development.

I would also like to acknowledge the University of Tartu, specifically the Faculty of Science and Technology, the Institute of Computer Science, for providing an exceptional academic environment. The knowledge and skills I have gained during my studies here have been foundational in conducting this research and writing this thesis.

Additionally, I am grateful to all the participants who contributed to my research. Their insights and experiences were vital to the empirical aspects of this study, enriching the content of this thesis significantly.

Throughout this journey, I have relied on Grammarly to help refine my writing and ensure clarity and precision in my sentences. This tool has been instrumental in correcting and shaping the language of this thesis, allowing me to present my ideas more effectively.

This thesis is the result of a collaborative effort, and I am truly thankful to everyone who has contributed to its successful completion. Thank you all for your invaluable support.

9. References

Ahmed, S.F., Alam, Md.S.B., Hoque, M., Lameesa, A., Afrin, S., Farah, T., Kabir, M., Shafiullah, G. and Muyeen, S.M. (2023). Industrial Internet of Things enabled technologies, challenges, and future directions. *Computers and Electrical Engineering*, [online] 110, p.108847. doi:https://doi.org/10.1016/j.compeleceng.2023.108847.

Alhazmi, A., Khulud Alawaji and OConnor, T. (2022). MPO: MQTT-Based Privacy Orchestrator for Smart Home Users. *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*. doi:https://doi.org/10.1109/compsac54236.2022.00152.

arendst (2018). *How to automate testing in TASMOTA? · Issue #4018 · arendst/Tasmota*. [online] GitHub. Available at: https://github.com/arendst/Tasmota/issues/4018#issuecomment-428305980 [Accessed 8 Aug. 2024].

athom. (2024). *athom*. [online] Available at: https://www.athom.tech/ [Accessed 12 Aug. 2024].

Bhatia, G. (2017). THE ROAD TO DOCKER: A SURVEY. *International Journal of Advanced Research in Computer Science*, 8(8), pp.83–87. doi:https://doi.org/10.26483/ijarcs.v8i8.4618.

Chataut, R., Phoummalayvane, A. and Akl, R. (2023). Unleashing the Power of IoT: A Comprehensive Review of IoT Applications and Future Prospects in Healthcare, Agriculture, Smart Homes, Smart Cities, and Industry 4.0. *Sensors*, [online] 23(16), p.7194. doi:https://doi.org/10.3390/s23167194.

Chung, M.T., Quang-Hung, N., Nguyen, M.T. and Thoai, N., 2016, July. Using docker in high performance computing applications. In 2016 IEEE Sixth International Conference on Communications and Electronics (ICCE) (pp. 52-57). IEEE.

Clerissi, D., Leotta, M., Reggio, G. and Ricca, F. (2018). Towards an approach for developing and testing Node-RED IoT systems. doi:https://doi.org/10.1145/3281022.3281023.

Debauche, O., Rachida Ait Abdelouahid, Mahmoudi, S., Yahya Moussaoui, Abdelaziz Marzak and Manneback, P. (2020). RevoCampus: a Distributed Open Source and Low-cost Smart Campus. doi:https://doi.org/10.1109/commnet49926.2020.9199640.

Elgazzar, K., Khalil, H., Alghamdi, T., Badr, A., Abdelkader, G., Elewah, A. and Buyya, R. (2022). Revisiting the internet of things: New trends, opportunities and grand challenges. *Frontiers in the Internet of Things*, 1. doi:https://doi.org/10.3389/friot.2022.1073780.

ESPHome. (2024). *ESPHome — ESPHome*. [online] Available at: https://esphome.io/index.html#esphome-automations [Accessed 11 Aug. 2024].

ESPHome. (n.d.). *ESPHome*. [online] Available at: https://esphome.io/index.html./ [Accessed 7 Aug. 2024].

Fizza, K., Banerjee, A., Mitra, K., Jayaraman, P.P., Ranjan, R., Patel, P. and Georgakopoulos, D. (2021). QoE in IoT: a vision, survey and future directions. *Discover Internet of Things*, 1(1). doi:https://doi.org/10.1007/s43926-021-00006-7.

Foote, K. (2022). *A Brief History of the Internet of Things - DATAVERSITY*. [online] DATAVERSITY. Available at: https://www.dataversity.net/brief-history-internet-things/.

Forsström, S. and Jennehag, U. (2017). *A performance and cost evaluation of combining OPC-UA and Microsoft Azure IoT Hub into an industrial Internet-of-Things system*. [online] IEEE Xplore. doi:https://doi.org/10.1109/GIOTS.2017.8016265.

Frequently Asked Questions (2024). *Frequently Asked Questions*. [online] ESPHome. Available at:

https://esphome.io/guides/faq.html#:~:text=While%20we%20try%20to%20test%20ESPHome/YAML%20as%20much%20as%20we%20can%20using%20our%20available%20hardware [Accessed 8 Aug. 2024].

Github.io. (2024). *For Developers - Tasmota*. [online] Available at: https://tasmota.github.io/docs/For-Developers/ [Accessed 12 Aug. 2024].

Hasan Derhamy, Eliasson, J., Jerker Delsing and Priller, P. (2015). A survey of commercial frameworks for the Internet of Things. *Emerging Technologies and Factory Automation*. doi:https://doi.org/10.1109/etfa.2015.7301661.

Himeur, Y. and Bensaali, F., Endorsing Energy Efficiency Through Accurate Appliance-Level Power Monitoring, Automation and Data Visualization.

Home Assistant (2024). *Securing*. [online] Home Assistant. Available at: https://www.home-assistant.io/docs/configuration/securing/ [Accessed 7 Aug. 2024].

Hosny, K.M., Magdi, A., Salah, A., Osama El-Komy and Lashin, N.A. (2023). Internet of things applications using Raspberry-Pi: a survey. *International Journal of Power Electronics and Drive Systems*, 13(1), pp.902–902. doi:https://doi.org/10.11591/ijece.v13i1.pp902-910.

initiative Industrie 4.0: Final report of the Industrie 4.0 Working Group.

iotempire (2019). *GitHub - iotempire/iotempower: IoTempower is a framework and environment for making the Internet of Things (IoT) accessible for everyone*. [online] GitHub. Available at: https://github.com/iotempire/iotempower [Accessed 11 Aug. 2024].

Jensen, D. (2019). *Beginning Azure IoT Edge Computing : Extending the Cloud to the Intelligent Edge*. Berkeley, Ca Apress Imprint, Apress.

Johnston, S. and Cox, S. (2017). The Raspberry Pi: A Technology Disrupter, and the Enabler of Dreams. *Electronics*, 6(3), p.51. doi:https://doi.org/10.3390/electronics6030051.

Jolles, J.W. (2021). Broad‐scale applications of the Raspberry Pi: A review and guide for biologists. *Methods in Ecology and Evolution*, 12(9), pp.1562–1579. doi:https://doi.org/10.1111/2041-210x.13652.

Jose Reena K (2023). IoT Programming Toolkits and Frameworks: Building a Foundation for Smart Connectivity at Home. *INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH AND ANALYSIS*, 06(10). doi:https://doi.org/10.47191/ijmra/v6-i10-12.

Kagermann, H., Wahlster W., Helbig, J. (2013). Recommendations for implementing the strategic

Karmakar, K.K., Varadharajan, V., Speirs, P., Hitchens, M. and Robertson, A. (2022). SDPM: A Secure Smart Device Provisioning and Monitoring Service Architecture for Smart Network Infrastructure. *IEEE Internet of Things Journal*, pp.1–1. doi:https://doi.org/10.1109/jiot.2022.3195227.

Kumar, P.C.P. and Geetha, G. (2017). Gateway Pi-Design and Implementation of Smart and Secure Internet of Things Gateway Integrating with Raspberry Pi. *Journal of Computational and Theoretical Nanoscience*, 14(9), pp.4448–4453. doi:https://doi.org/10.1166/jctn.2017.6760.

Malik, B.H., Khalid, M., Maryam, M., Nauman, M., Yousaf, S., Mehmood, M. and Saleem, H., 2019. IoT Testing-as-a-Service: A new dimension of automation. International Journal of Advanced Computer Science and Applications, 10(5), pp.364-371.

Moreau, D., Wiebels, K. and Boettiger, C., 2023. Containers for computational reproducibility. Nature Reviews Methods Primers, 3(1), p.50.

Murad, G., Aalaa Badarneh, Abdallah Quscf and Fadi Almasalha (2018). Software Testing Techniques in IoT. doi:https://doi.org/10.1109/csit.2018.8486149.

PetaPixel. (2013). *The First Webcam Was Invented to Check Coffee Levels Without Getting Up*. [online] Available at: https://petapixel.com/2013/04/03/the-first-webcam-was-invented-to-check-coffee-levels-without-getting-up/ [Accessed 7 Aug. 2024].

Pierleoni, P., Concetti, R., Belli, A. and Palma, L. (2020). Amazon, Google and Microsoft Solutions for IoT: Architectures and a Performance Comparison. *IEEE Access*, 8, pp.5455–5470. doi:https://doi.org/10.1109/access.2019.2961511.

Sharma, A. and Jit Singh, B. (2020). Evolution of Industrial Revolutions: A Review. *Regular*, 9(11), pp.66–73. doi:https://doi.org/10.35940/ijitee.i7144.0991120.

Simadiputra, V. and Surantha, N. (2021). Rasefiberry: Secure and efficient Raspberry-Pi based gateway for smarthome IoT architecture. *Bulletin of Electrical Engineering and Informatics*, 10(2), pp.1035–1045. doi:https://doi.org/10.11591/eei.v10i2.2741.

Sinan Ameen Noman, Haitham Ameen Noman, Qusay Al-Maatouk and Atkison, T. (2023). Internet of Things Communication, Networking, and Security: A Survey. *International Journal of Computing and Digital Systems*, 13(1), pp.923–936. doi:https://doi.org/10.12785/ijcds/130173.

tasmota.github.io (2017). *MQTT - Tasmota*. [online] Available at: https://tasmota.github.io/docs/MQTT/ [Accessed 11 Aug. 2024].

tasmota.github.io. (n.d.). *News - Tasmota*. [online] Available at: https://tasmota.github.io/docs / [Accessed 6 Aug. 2024].

The Docker Book: Containerization is the new virtualization (2014). Brooklyn: James Turnbull

Udoh, I.S. and Kotonya, G. (2018). Developing IoT applications: challenges and frameworks. *IET Cyber-Physical Systems: Theory & Applications*, 3(2), pp.65–72. doi:https://doi.org/10.1049/iet-cps.2017.0068.

ulno (2017). *Development Kits for the Internet of Things (IoT)*. [online] ulno.net. Available at: https://ulno.net/iot/devkits/ [Accessed 11 Aug. 2024].

ulno (2024). *Teaching Internet of Things (IoT) the IoTempower Way*. [online] ulno.net. Available at: https://ulno.net/teaching/iot/ [Accessed 7 Aug. 2024].

ut-teaching-ulno (2023). *IoTempower Under The Hood - 4/6 - Natively on Linux*. [online] YouTube. Available at: https://www.youtube.com/watch?v=HM-J361gW_c [Accessed 11 Aug. 2024].

ut-teaching-ulno (2023). *IoTempower Under The Hood - 5/6 - Tools and Node Types*. [online] YouTube. Available at: https://www.youtube.com/watch?v=RXwJZ0HTpPQ&list=PLlppUpfgGsvkfAGJ38_mzQc1-_Z7bNOgq&index=28 [Accessed 8 Aug. 2024].

Uviase, O. and Kotonya, G. (2018). IoT Architectural Framework: Connection and Integration Framework for IoT Systems. *Electronic Proceedings in Theoretical Computer Science*, [online] 264, pp.1–17. doi:https://doi.org/10.4204/eptcs.264.1.

Writer, G. (2020). *Alexa, Google Assistant and Apple HomeKit: Your Guide to Smart Home Ecosystem Options*. [online] IoT For All. Available at: https://www.iotforall.com/comparing-smart-home-ecosystem-options-alexa-google-assistant-apple-homekit./ [Accessed 6 Aug. 2024].

www.cl.cam.ac.uk. (n.d.). *The Trojan Room Coffee Pot Timeline*. [online] Available at: https://www.cl.cam.ac.uk/coffee/qsf/timeline.html/ [Accessed 7 Aug. 2024].

www.flux.ai. (n.d.). *Arduino Vs Raspberry Pi: Which Is the Best Board for You*. [online] Available at: https://www.flux.ai/p/blog/arduino-vs-raspberry-pi-comparison./ [Accessed 6 Aug. 2024].

www.openhab.org. (n.d.). *Introduction*. [online] Available at: https://www.openhab.org/docs/ [Accessed 7 Aug. 2024].

www.statista.com. (2017). *Topic: Internet of Things*. [online] Available at: https://www.statista.com/topics/2637/internet-of-things/ [Accessed 7 Aug. 2024].

www.threadgroup.org. (n.d.). *Thread Benefits*. [online] Available at: https://www.threadgroup.org/What-is-Thread/Thread-Benefits / [Accessed 6 Aug. 2024].

10. Appendix

10.1. Interview Questions

1. "Would you like to take a look at the documentation and give me some feedback?"
2. "Do you think the documentation is sufficient enough to introduce the user to the framework?"
3. "From your perspective, how intuitive and user-friendly is the framework?"
4. "Are there any areas where you feel users might struggle or need additional guidance?"
5. "Are there any features you think are missing or could be enhanced to improve the testing functionality?"

10.2. Testing IoTempower Documentation

Overview

The Testing IoTempower project focuses on ensuring the robustness and reliability of IoTempower applications. This documentation provides details on the structure and contents of the testing folder, along with instructions on running the tests.

Contents

The testing folder for the IoTempower project is located at `iot/tests`. The files that start with the `test_` prefix are actual test files. Other files contain shared functionalities and configuration for the tests.

File Descriptions

- **conf_data.py**: Contains configurations and test cases used by the test files. This file includes several variables:

- **packages**: A list of dictionaries holding values for dependency names and the nature of the dependency. Important for `test_installations`.
- **boards**: A list of devices to test compilation. All boards and IoT devices are paired individually to test the compilation of all combinations.
- **devices**: A list of sensor names and syntaxes. All boards and IoT devices are paired individually to test the compilation of all combinations.
- **isolated_combinations_to_test**: Combinations of sensors and boards that can only run on specific boards.
- **gateway_host**: Hostname for the Raspberry Pi in the local network, typically `iotgateway`. This is important for `test_deploy` and `test_hardware` since these tests should be run on an actual Raspberry Pi device.
- **default_username**: The default username for SSH connection into the Raspberry Pi.
- **mqtt_listen_period**: Number of seconds to listen to the MQTT channel to verify that the expected message is published under the expected topic.
- **local_bind_mqtt_port**: Since we are connecting to the Raspberry Pi as a device in a local network, it doesn't expose its MQTT port (1883 by default). That's why we use SSH tunneling to bind that port to one of our local ports.
- **private_key_file_path**: The Raspberry Pi should be configured before running tests to recognize our private key for a smooth SSH connection experience. Change this value if you have configured a different key with the Pi.
- **nodes_folder_path**: The full path of the folder used for running deployment and hardware testing.
- **tested_node_name**: The folder name of the node being tested.
- **tester_node_name**: The folder name of the node used for testing.
- **cases_for_deployment**: A list of tuples where each tuple represents a separate test. Each tuple contains:
  - The board name for the `node.conf` file.
  - A list of lines for the `setup.cpp` file.

        - A list of tuples, each containing an MQTT topic and the expected payload from that topic.
    - **cases_for_hardware**: A list of tuples where each tuple represents a separate test. Each tuple contains:
        - A pair of board names for the tested and tester nodes, which go into the `node.conf` file of each node, respectively.
        - A pair of lists of syntax lines for the tested and tester nodes, which go into the `setup.cpp` file of each node, respectively.
        - A tuple containing three lists of tuples:
            - The tested node topic and initial status messages emitted by the tested node.
            - The set topic and corresponding set commands to trigger the tester node.
            - The tested node topic and the expected status messages after the tester trigger.
- **deploy_device_address, tester_device_address, tested_device_address**: These variables are explained in the Hardware Configuration section below.
- **test_installations.py**: Checks if your installations are complete. This test ensures that all the selected dependencies are correctly installed and configured so that IoTempower works properly. The input file for this test is `installation.json`, which is generated during the installation period and saves your selection of dependencies. Some dependencies are not crucial and are only installed for specific features. If you are using a Docker image or Raspberry Pi image, there is no need to run these tests as they have already been checked.
    - **Purpose**: To verify that all dependencies are installed and configured correctly.
    - **Input**: `installation.json`, generated during installation.
    - **Note**: Not necessary if using Docker or Raspberry Pi images.
- **test_compile.py**: Tests the compilation process. The compilation is a process where the framework takes node configurations and generates binary files to upload to the actual

device. This process can be done in a fully isolated manner. In `conf_data.py`, there is an example where:

- The **boards** variable lists all devices.
- The **devices** variable lists sensor names and syntaxes that can be compiled with all boards.
- The **isolated_combinations_to_test** variable contains combinations of sensors and boards that can only run on specific boards.
- **Purpose**: To verify that node configurations can be compiled into binary files for devices.
- **Details**:
  - Uses `boards` and `devices` lists to create all possible combinations.
  - `isolated_combinations_to_test` contains specific sensor-board combinations.
- **Example**: Refer to `conf_data.py` for how devices and sensors are combined.

- **test_deployment.py**: Requires a Raspberry Pi device to be connected to the network. Normally, the Pi registers itself in the local network under the name `iotgateway`. If this is not the case for you, change the `gateway_host` variable in `conf_data.py`. You should configure the SSH connection before running tests (see details at SSH Academy). The actual testing parameters are assigned to the `cases_for_deployment` variable, which is a list of tuples. Each tuple represents a separate test and holds three values:
  - The board name for the `node.conf` file.
  - A list of lines for the `setup.cpp` file.
  - A list of tuples where each tuple contains an MQTT topic and the expected payload from that topic.
  - **Purpose**: To verify that the deployment process works correctly on a Raspberry Pi.
  - **Details**:
    - Change `gateway_host` in `conf_data.py` if `iotgateway` is not used.

        - Configure SSH as described at SSH Academy.
        - `cases_for_deployment` lists test cases, each with a board name, setup lines, and MQTT topic-payload expectations.
- **test_hardware.py**: The statements for `test_deployment.py` are also applicable for this test. You should have at least two dongles connected to the Raspberry Pi for tester and tested roles.
    - **Purpose**: To verify that after the deployment process, the physical changes also take effect, rather than just seeing those changes in serial or MQTT channels.
    - **Details**: `cases_for_hardware` lists test cases, and new test cases should be added to that list.

Running Tests

To run the tests:

1. Navigate to the `tests` folder.
2. Activate the IOT environment by typing `iot` (if it hasn't been activated already).
3. Use the pytest testing library to run the tests. The command to run all tests is:

```
pytest -s -v
```

To run a specific test, append the test file name to the command. For example, to run the deployment tests:

```
pytest -s -v test_deployment.py
```

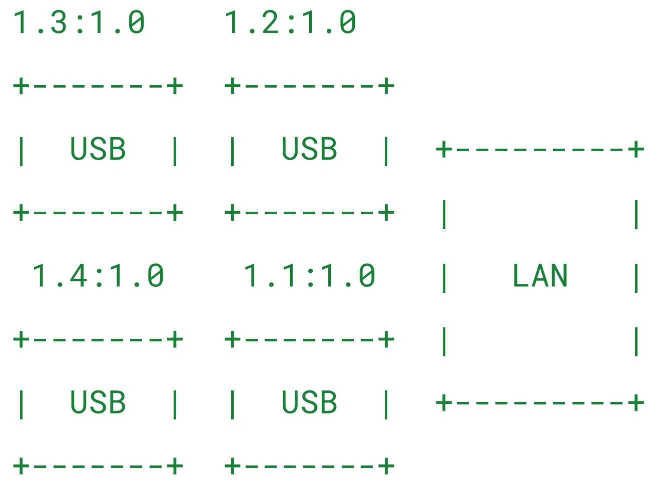
Hardware Configuration

To set up deployment and hardware testing, you should define the address of the node device in the configuration file. You can easily list the connected devices by executing the following command on the Raspberry Pi:

```
ls -l /dev/serial/by-path/
```

Device Layout

This is the device layout of the Raspberry Pi:



License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Araz Heydarov**,

(*author's name*)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:
   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and
   1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

Developing a Testing Framework for Internet of Things Systems using IoTempower as an Example

(title of thesis)

supervised by Ulrich Norbisrath,

(supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains these rights.

4. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Araz Heydarov

13.08.2024