INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelor's Thesis

An IoT Test Bed with Heterogeneous Embedded Platforms and Network Technologies

Curt Polack

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelor's Thesis

An IoT Test Bed with Heterogeneous Embedded Platforms and Network Technologies

Curt Polack

Supervisor:	Prof. Dr. Dieter Kranzlmüller
Advisors:	Tobias Guggemos
	Jan Schmidt
	Annette Kostelezky
Date:	September 18th, 2019

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 18. September 2019

. . . . . .
(Unterschrift des Kandidaten)

Abstract

Small, efficient computers, known as Internet of Things (IoT) devices, have become increasingly important in the professional and academic worlds. In order to ensure that these devices can function under various and often limited conditions, both the software and the hardware must be rigorously tested. Test beds are, for reasons of convenience and consistency, frequently used to test such devices. IoT devices often employ a plethora of network technologies to communicate with one another as well as the outside world; ensuring that nodes can communicate effectively and securely is especially difficult in a heterogeneous environment.

This thesis examines different existing test beds as well as their limitations, and documents the design and implementation of a test bed at the University of Munich. The documented test bed was implemented using customized open source software created by the Future Internet of Things (FIT) Consortium in France and focuses on the interoperability of different network technologies and the associated challenges. In contrast to many existing test beds, it also provides the network infrastructure necessary for communication between devices using different network technologies. The finished test bed allows students and faculty to test their software on a number of devices and network technologies simultaneously without unnecessary overhead or needing to be physically present. The test bed is currently independently run by the University of Munich; an integration into the overarching FIT infrastructure is, however, planned for the near future.

Contents

1 Introduction

Internet of Things (IoT) devices have been growing in popularity and relevance for years. In 2010 devices connected to the internet outnumbered humans on a world-wide scale at a ratio of 1.84; by 2020, the ratio is expected to grow to 6.58 [Eva11]. With the growing number of IoT devices, which often collect sensitive data, there has also been a rise in concerns regarding privacy, security, and reliability. In order to ensure that both software and hardware of IoT devices properly function, they must be rigorously tested under different conditions; to this end, both industrial and academic organizations have developed a number of IoT test beds.

1.1 Motivation

The growing number of IoT devices is, however, also accompanied by challenges. The global IoT network is characterized by strong heterogeneity; there are a large number of unique network technologies used to create IoT networks [Pam16]. Many technologies were created to serve a specific purpose or to optimize an aspect of communication. The Bluetooth standard, for example, offers a personal area network with high data transfer rates and a very limited range; this standard is especially useful for connecting peripheral devices to cell phones or cars. On the other hand, the LoRa standard provides the means for low-power and long-range data transmission; the technology does, however, provide a much more limited data transfer rate. Among the technologies that have established themselves in the recent past are ZigBee, Wi-Fi, Bluetooth, Sigfox, LoRa [NBC17].

These varied network technologies / standards also differ in terms of the possible network topologies. Some allow only for a star topology, while others are capable of a mesh topology or a point to point topology. The dissimilarity of the many commonly used network technologies makes it difficult to test the interoperability of software on all platforms. Many IoT networks employ different network technologies and topologies to achieve various distinct yet interconnected goals. A Wireless Sensor Network (WSN) can, for example, receive smaller data packets from devices, which are geographically further away, using LoRa and receive larger packets from closer devices using ZigBee. The aggregated information from all sensors can then be sent to a central server over Wi-Fi or even Ethernet, in order to make use of the superior processing power of conventional computers, for example using the network topology depicted in Figure 1.1. This heterogeneity makes it difficult to test software using a conventional test bed consisting of separate and disparate networks / network technologies.

Communication between the devices of two networks which use distinct network technologies and / or topologies is especially challenging due to the lack of similarities and, more importantly, a common protocol. Recently, developers have begun implementing the most commonly used layer 3 protocol, Internet Protocol (IP), for a number of different IoT network technologies. The implementation of the IP protocol for a multitude of network technologies allows for direct communication between devices that use different (layer 1/2) technologies. Despite the rapid progress, most test beds today do not provide the necessary infrastructure to quickly and reliably test the communication between such networks. The

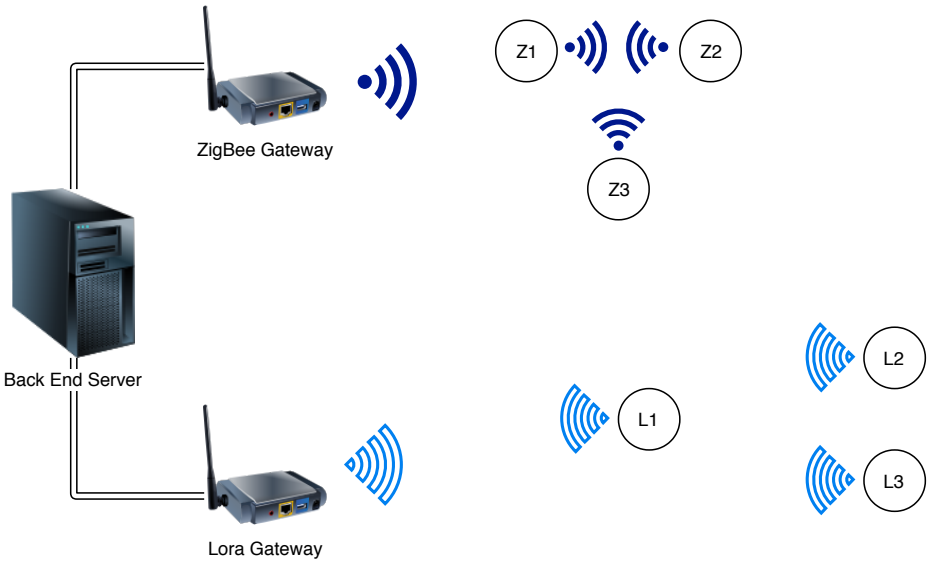


Figure 1.1: Heterogeneous IoT Set-up with a ZigBee Network (Mesh) a LoRa Network (Star)

goal of this thesis is to create an IoT test bed with the properly configured infrastructure required to allow for communication between otherwise dissimilar network technologies using the IP protocol.

1.2 Approach & Structure

This thesis elaborates the approach used to design and implement a heterogeneous test bed with a focus on the interoperability of selected network technologies. The second chapter provides an understanding of the most important terms and technologies which are later referenced in the thesis. The third chapter elaborates both the functional and non-functional requirements for the hardware and the software of the planned test bed for IoT devices. The fourth chapter examines existing open source test beds including their advantages, disadvantages, and common use cases based upon the requirements defined in the third chapter. The fifth chapter describes the design and architecture of both the hardware and software for the test bed and validates them using the the requirements defined in the third chapter as well as the configuration of the gateways and the network topology that allow for the interoperability between different technologies. The sixth chapter details the implementation of both the hardware and software for the test bed in addition to the overarching physical set-up, including casing, wiring, etc. More specifically, the sixth chapter includes the information necessary for the the addition of further, not yet supported devices and network technologies. This thesis concludes with a statement about the work thus far accomplished and the possibilities for further work / papers.

2 Background

The following chapter provides an introduction to the terminology and background information necessary to understand the thesis and the design of the testbed.

2.1 Vision & Use Cases

The goal of this thesis is to create a test bed which allows for the uncomplicated testing of software by reflecting real world scenarios in which an IoT network uses diverse hardware and network technologies to achieve a common goal. A smart city, defined as a new urban environment designed for performance through information and communication technologies [Dob18], is a concept that encompasses a number of industries and applications; many of which must communicate with one another in order to function ideally.

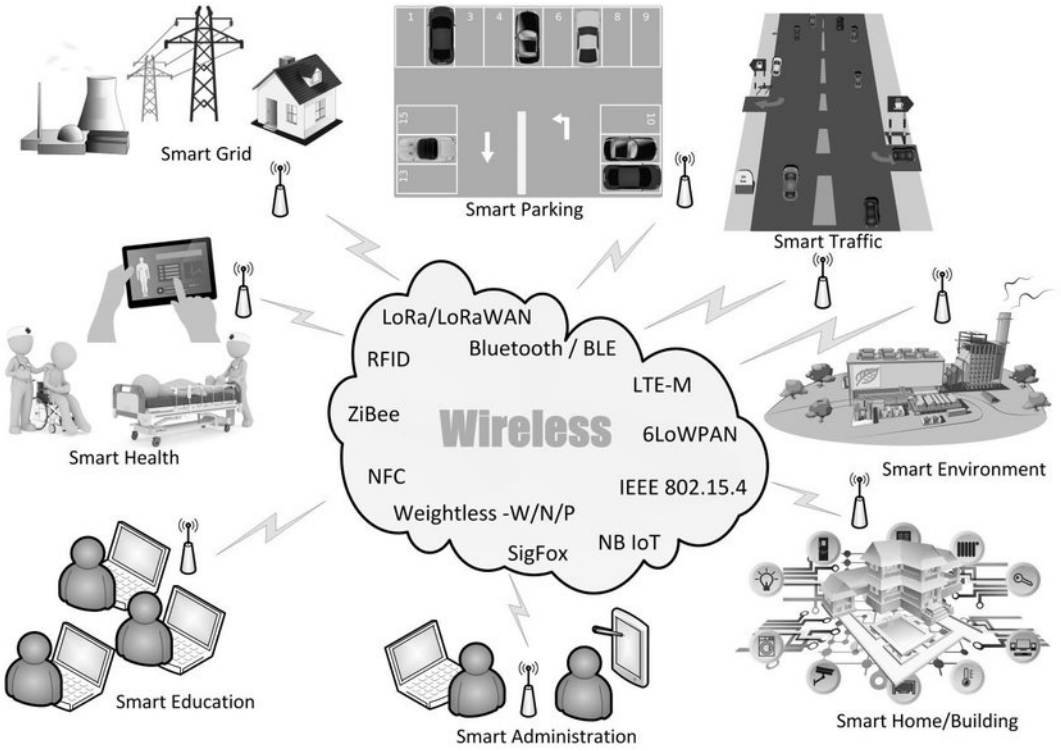


Figure 2.1: Overview of "Smart" Applications and Technologies [Dob18]

Appliances in a home, for example, should communicate with each other and a central system to create a smart home environment. A localized "smart home" IoT network allows users to control their appliances and keep track of their energy usage. The communication between appliances also opens the door to a plethora of opportunities; a door, which is outfitted with a sensor, can ensure that a security camera records when it is opened. Real-time

data from these appliances can, however, also be useful when implementing a smart grid system. Regional power providers / distributors could use this real-time data to increase or throttle energy production when needed as well as intelligently predict long-term energy consumption trends. The manufacturers of the appliances can also use this data to provide predictive maintenance directly to the end user. These largely different applications and devices, which may utilize different network technologies, must be able to communicate to achieve the desired benefits. These are just a few of the many applications of IoT technology that call for communication between diverse devices and separate networks / network technologies.

This thesis is a part of a larger effort by the University of Munich to create a test bed for IP Multicast communication between embedded systems. The end goal of this thesis in conjunction with other (future and current) theses is the creation of a test bed which allows for the management of dynamic IP Multicast groups.

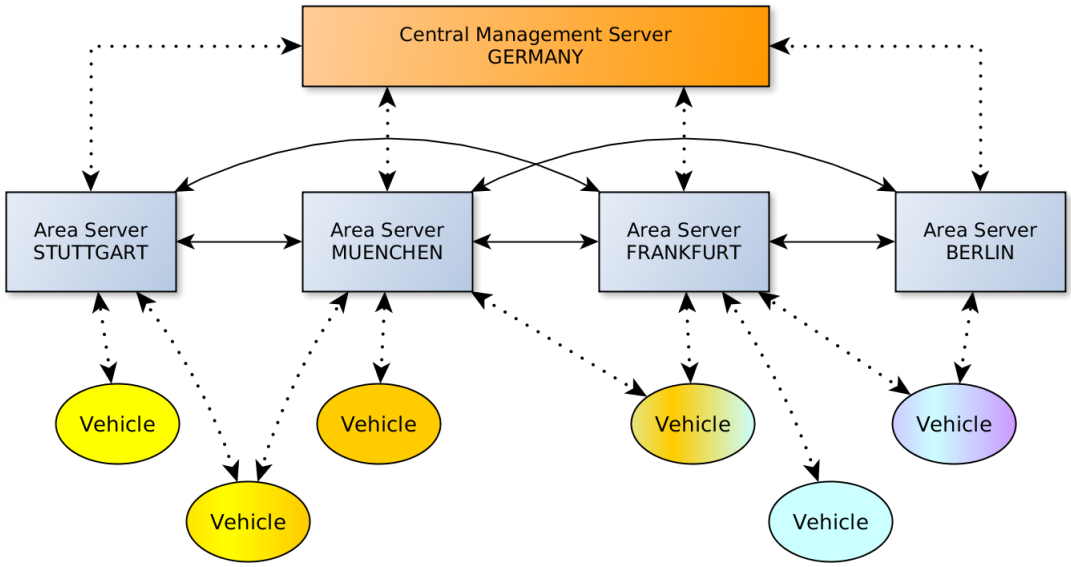


Figure 2.2: Overview of Multicast Group Scenario on German Highways [Eic17]

The scenario explained by Florian Eich in his report is set on two highways in Germany, the A7 and the A8, that run between Munich and Berlin [Eic17]. Each vehicle, regardless of its manufacturer, is connected with an area server using the closest wireless interface. Each vehicle wishes to receive the data which is relevant to its current situation, planned route, and immediate surroundings. The central management server controls the IP Multicast groups to ensure that vehicles receive information which is relevant to them. The central server can add and remove vehicles to/from IP Multicast groups to ensure that they receive all and only the information which is relevant to them.

A diverse test bed is, due to the heterogeneity of wireless network technologies that could be used by the vehicles in this scenario, the optimal environment in which to test such software. The interoperability of nodes and the use of a common protocol while using different servers and / or network technologies is a crucial advantage when testing such software when compared to traditional test beds.

2.2 Network Technologies

Multiple technologies and standards have established themselves in the IoT ecosystem. This section provides an overview of those most commonly used as well as their advantages, disadvantages, and possible use cases.

Technology	Standard	Frequency	Range	Data Transfer Rate	Topology
Bluetooth	IEEE 802.15.1	2.4GHz	1-10m	1 Mbps	Point-to-point, Star
802.15.4	IEEE 802.15.4	2.4GHz	10-100m	250 Kbps	Mesh, Star
LoRa	LoRa Alliance	867-869MHz(Europe)	2-5Km	290bps-50Kbps	Star
Wi-Fi	IEEE 802.11 (a/b/g/n)	2.4GHz,3.6GHz, 4.9GHz,5GHz, 5.9GHz	100m	1-54Mbps	Star
LTE	3GPP	2.5GHz, 5GHz,10GHz	30Km	75Mbps-300Mbps	Star

Table 2.1: Comparison of selected network technologies commonly used in IoT devices [YHM+17]

2.2.1 802.15.4

IEEE 802.15.4 is a standard developed by the Institute of Electrical and Electronics Engineers (IEEE) that specifies a protocol with a low data transfer rate, low power consumption, and low cost wireless networking [KAT05]. Although the standard was not specifically developed for IoT networks or WSNs, its specifications make it an ideal technology for such networks, especially those that have access to limited power or are battery driven. The protocol only specifies layer 1-2 in the ISO/OSI Model (physical and Media Access Control (MAC)); there are a number of protocols which build on this standard by defining the upper layers (network layer, etc.). The 802.15.4 standard and the protocols built based on it have been the subject of much research and are found in most modern IoT test beds.

2.2.2 Wi-Fi

Wi-Fi is arguably the most commonly used and well-known network technology used by mobile devices. The multiple Wi-Fi standards, IEEE 802.11(a/b/g/n), enable almost all of our mobile personal devices to reliably and securely connect to the internet wirelessly through close-by access points. The Wi-Fi standards provide much faster data transfer than 802.15.4 as well as a considerable range but require more complicated (and thus more expensive) transceivers and significantly more energy [FNS15]. These two hindrances make Wi-Fi unsuitable for many IoT-applications that value cost-effective and simple communication over speed; smart home applications, for example, are better served by the 802.15.4 standard than by Wi-Fi.

2.2.3 LTE

LTE is one of the technologies created by the 3GPP Mobile Competence Centre which also developed other well-known cellular standards, including GSM and 3G. LTE - much like the Wi-Fi Standards - provides a much faster data transfer rate than other IoT network technologies at the expense of transceiver complexity and power consumption [EH18]. LTE, in contrast to the Wi-Fi standards, allows for faster data transfer over a much larger range than other wireless network technologies. LTE is similar to the Wi-Fi standards in that it is unsuitable for many IoT-application due to its emphasis on speed and range over cost-efficiency and power consumption. The use of LTE in most countries requires, like other cellular technologies, a license from the responsible government agency to use a particular frequency [MM16]. This prerequisite makes it difficult for the average developer to legally use the LTE technology; LTE is instead most commonly used to provide mobile devices (cell phones, tablets, etc.) with high speed internet access.

2.2.4 Bluetooth

Bluetooth is a wireless network technology based on the IEEE 802.15.1 standard that was launched in 1994 by Ericsson, a Swedish telecommunications company [NBC17]. Bluetooth was designed to connect wireless peripheral devices (i.e. mouse, headset) to a larger device (i.e. phone, computer), in order to avoid unnecessary cables. Bluetooth for this reason offers a significantly slower data rate, a smaller range than Wi-Fi or LTE does, and enhanced power efficiency. In addition to the four conventional Bluetooth standards (1.0, 2.0, 3.0, 4.0), developers have also created Bluetooth Low Energy (BLE) which, in contrast to conventional Bluetooth, uses a free ISM band and provides significantly superior battery life [NBC17].

2.2.5 LoRa

LoRa is a proprietary standard for wireless networking created by Semtech, which is still today the only producer of integrated circuits for the LoRa Standard. LoRa uses a much wider band which makes it resistant to channel noise and fading but also results in less efficient transmission [NBC17]. The low power consumption and long range of LoRa devices makes it ideal for WSNs which send small packets of data from devices with access to limited power over a large distance. The LoRa standard defines only the physical layer and does not specify or implement any further layers.

LoRaWAN

LoRaWAN is a standard for wireless networking that builds on the LoRa standard and was created by the LoRa Alliance which specializes in the development of standards for low power wide area wireless networks (LPWANs). LoRaWan specifies layer 2 (MAC) as well as (partially) layer 3 and thus allows for the creation of networks of devices using the LoRa standard [MMAM17]. There are a number of standards which use LoRa for the physical layer; LoRaWan is, however, the most commonly used.

2.2.6 Ethernet

Ethernet is one of the oldest and most commonly used wired network technologies still used today [Ull12]. Ethernet creates networks using cables and devices, including switches, re-

peaters, hubs, and bridges. Ethernet supports a plethora of network topologies, including star, tree, and bus. The data transfer rate depends on the two clients as well as the cable being used. The CAT6 standard cable allows for speeds of up to 10Gbps and is thus significantly faster than any of the aforementioned wireless network technologies [Ull12].

2.2.7 Classification of Network Technologies

Network technologies can be classified using a number of criteria. They are often classified into groups based on the range they offer as seen in Figure 2.3. The layer 2 Maximum Transmission Unit (MTU) is also an important constraint when regarding network technologies for embedded devices. Many network technologies specify a MTU that is significantly smaller than the minimum size for IPv6 packets, 1280 bytes, and thus require a further layer of fragmentation (i.e. 6LoWPAN) [BEKG19]. Network technologies used by embedded systems are for that reason often categorized according to their MTU as seen in Table 2.2.

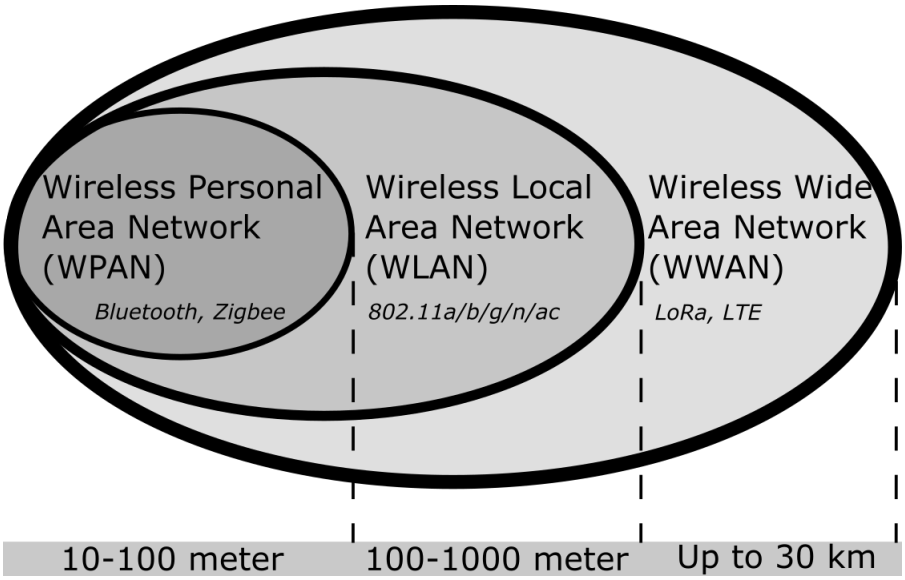


Figure 2.3: Categories for Network Technologies Based on Range [MM16]

Classification	MTU	6LoWPAN Fragmentation	Technologies
S0	3-12 Bytes	Not Possible	SigFox
S1	13-127 Bytes	Required	802.15.4 [LA18], LoraWAN (Data Rate 0-3) [80216]
S2	128-1279	Required	LoraWAN (Data Rate 4-7)[LA18], Bluetooth 4.2+ [TI16]
S3	$\geq$ 1280	Not Required	Ethernet, Wi-Fi, LTE

Table 2.2: Classification of Network Technologies Based on Layer 2 MTU [BEKG19]

2.3 Network Stack

There are a number of protocols / implementations used in the network stack of IoT devices which allow them to communicate despite the constraints of the underlying network technologies. A selection of those most commonly used are described in this section.

2.3.1 ZigBee

The ZigBee protocol was created by the ZigBee Alliance and is one of the most widely used networking technologies built upon the 802.15.4. standard [PD16]. Due to its use of the 802.15.4 standard, ZigBee is characterized by low power, low cost, and relatively slow wireless communication. ZigBee provides a simple to configure isolated network, making it ideal for a number of applications including smart home (light switches, climate control, etc.) as well as in WSNs [PD16].

2.3.2 6LoWPAN

IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) is a standard currently being developed by the Internet Engineering Task Force (IETF) which allows for IPv6 packets to be transmitted using low power wireless networks, especially 802.15.4. Although using IPv6 results in more overhead than proprietary protocols (e.g. LoRaWAN), it allows for simpler network topologies and direct communication between different network technologies [Mul07]. The IETF has also developed methods for reducing the overhead of IPv6 packets in order to achieve the increased efficiency often required by IoT devices. 6LoWPAN allows into account the constraints that wireless network technologies often have by providing an extra layer of fragmentation, since the smallest IPv6 Datagram must be 1280 bytes. 6LoWPAN is most commonly used in conjunction with the 802.15.4 standard; it can, however, be used with other underlying protocols on the physical layer.

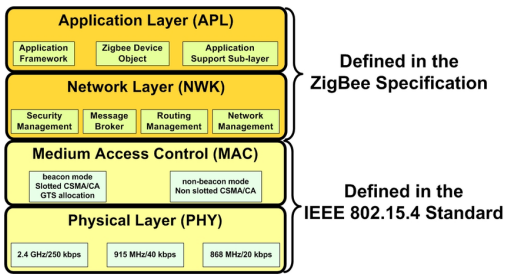


Figure 2.4: ZigBee Architecture [KAT19]

2.3.3 lwIP

Lightweight IP (lwIP) is an open source implementation of the TCP/IP Stack for embedded / constrained devices. [Dun01] The implementation supports a number of protocols, including IPv4, IPv6, PPPoE, TCP, UDP, and can be run using only ca. 40 KB of ROM and 10 KB of RAM which allows for it to be run on the vast majority of devices.

2.4 Embedded Systems

An embedded system is a computer system with a specific, dedicated function that is not designed so that it should ever need to be reprogrammed [Lam17]. Embedded systems are often used in everyday applications and devices to provide computing power for a specific

purpose which has been determined in advance. Although embedded systems lack the flexibility and computing power of traditional computers, they provide a superior form factor and power consumption.

2.4.1 Microcontroller

A microcontroller, the most common type of embedded system, is a small computer system on a single integrated circuit [Lam17]. Microcontrollers are often used to perform simple tasks, e.g. reading and transmitting sensor data.

2.4.2 Serial Communication

Serial communication uses the digital (binary) transmission of data and timing to allow embedded systems to communicate with other, often peripheral, devices [Lam17]. There are a number of standards that define interfaces and protocols for the digital communication used by embedded devices, including UART, SPI, I2C, CAN and One Wire. Almost all microcontrollers have at least one serial port.

SPI

Serial Peripheral Interface (SPI) is a standard for digital communication which allows a "master", normally the microcontroller, to handle multiple "slave" integrated circuits. [Lam17]. SPI is significantly faster than other traditional serial communication technologies, allowing a for speeds of up to of 500 MHz. The SPI standard defines 4 Pins: SCLK (Serial Clock), MOSI (Master Out Slave In), MISO (Master In Slave Out), and Slave Select which allows for multiple slaves to be connected simultaneously. SPI allows for full duplex communication and is synchronous, meaning that both devices use the same clock for communication.

UART

Universal Asynchronous Receiving and Transmitting (UART) sends all data as character bytes and is, for that reason, the universal standard; this method of communication also makes UART relatively inefficient and slow. [Lam17] UART allows for full duplex communication and is asynchronous, meaning that each device requires a buffer and internal clock.

2.4.3 Common Embedded Systems

Embedded systems / microcontrollers are becoming increasingly more important due to the rise of IoT technology. A number of companies produce microcontrollers for industrial and personal use, many of which are specially made to facilitate the testing of software.

Arduino

Arduino is the one of the world's leading open source hardware and software ecosystems and provides a number of software tools and hardware platforms which allow developers to create and test software for embedded systems [Arda]. The first Arduino board was created in 2005 to help design students with little experience use IoT platforms; Arduino now offers multiple distinct microcontrollers as well as a plethora of peripheral devices.

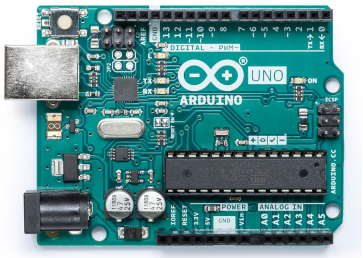


Figure 2.5: Arduno Uno Rev V3 [Ardb]

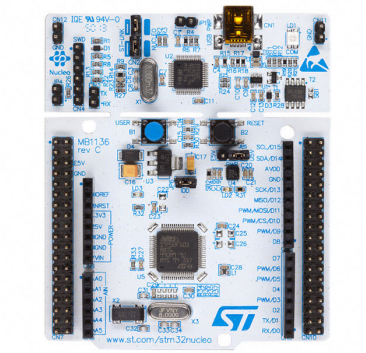


Figure 2.6: Nucleo F091RC [Mou]

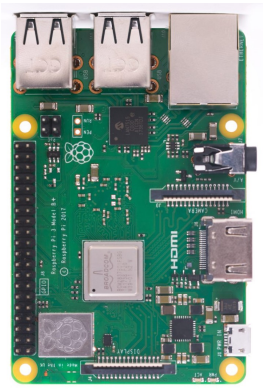


Figure 2.7: Raspberry Pi 3B+ [Ras]

Nucleo

Nucleo is a brand of microcontrollers developed and manufactured by STMicroelectronics, a world leader in the production of semiconductor solutions [STM]. The Nucleo microcontrollers / boards were created to allow developers to quickly test their applications using the hardware developed by STMicroelectronics.

Atmel

Atmel was founded in 1984 and is one of the worldwide leading manufacturers of microcontrollers and other embedded systems / peripheral devices [ASS]. Atmel offers a plethora of microcontrollers using a number of different architectures and powering a wide range of everyday and industrial devices. Atmel was acquired in 2016 by a competitor Microchip, which continues to produce chips using the Atmel brand.

Raspberry Pi

The Raspberry Pi Foundation is a charity based in the United Kingdom which aims to "put the power of computing and digital making into the hands of people all over the world" [FOUa]. The Raspberry Pi Foundation designs and sells low-cost and high-performance computers; the computers are not embedded systems as defined in this paper but instead compact and low-cost single-board computers. The Raspberry Pi Foundation also provides an open source distribution of the Linux distribution Debian, Raspbian, to facilitate use of and development of software for the hardware.

2.4.4 Classification of Embedded Systems

Embedded systems are often classified into groups based on the size of available RAM and Flash due to the significant impact they have on the capabilities of the system [BEKG19]. Class 1 devices are, for example, very constrained, making the implementation of "full stack" protocols (e.g. HTTP) impossible to fully implement. Class 3 devices, on the other hand, are significantly more powerful and capable of running more complex software / network stacks.

Classification	RAM Size	Flash Size
C0	<10 KB	< 100 KB
C1	≈ 10 KB	≈ 100 KB
C2	≈ 50 KB	500-1000 KB
C3	300-1000 KB	1000-2000 KB

Table 2.3: Classification of Embedded Systems [BEKG19]

2.5 Architectures for Embedded Systems

This section provides an overview of the most relevant architectures for embedded systems.

RISC

Reduced Instruction Set Computer (RISC) is an architecture for microprocessors which, in contrast to Complex Instruction Set Computing (CISC), only supports one layer of simple, standardized instructions. All instructions are completed within one processor cycle [Bha91]. RISC established itself in the 1990's as a superior alternative to CISC for use in embedded systems, due to its advantages regarding power consumption.

ARM

Advanced Risc Machines (ARM) is a modern architecture for microprocessors which is based on RISC and provides a simple, efficient architecture for constrained devices as well as many mobile devices (e.g. smartphones) [Zla16]. ARM is widely used today in a number of processors, especially those used by embedded systems.

2.6 Operating Systems for Embedded Systems

There are a number of operating systems used to create applications for embedded systems, which due to their limited resources are often unable to run traditional operating systems such as Linux. This section provides an overview of a few selected common operating systems used in the IoT context.

FreeRTOS

FreeRTOS (Free Real Time Operating System) is an open source project first developed in 2003 by Richard Barry and is currently actively developed by Amazon Web Services [Fre]. It aims to provide an easy to use and robust operating real-time operating system for embedded devices. FreeRTOS provides an easy to use API and is written in C.

TinyOS

TinyOS is an open source operating system specifically designed for low-power wireless devices and is written in nesC, a dialect of C. [Lev12] The nesC dialect was designed by the creators of TinyOS to better fit the requirements of development for embedded systems. The latest version of TinyOS (2.12) was, however, released in 2012.

RIOT OS

RIOT OS is an open source operating system designed for use with IoT devices while supporting multi-threading [BGH+18]. RIOT OS supports a large number of embedded systems, including multiple Arduino and almost all Nucleo boards. RIOT OS is built modularly to allow for the intuitive and quick development of software for IoT devices. RIOT OS also provides drivers for a number of wireless transceivers; some of these drivers were created by the active open source community that powers its development.

3 Requirement Analysis

This chapter outlines the requirements for both the software and hardware of the test bed. The requirements are separated into two categories, functional and non-functional. Functional requirements are statements of a piece of required functionality or a behavior that a system will exhibit under specific conditions [Wie03]. A functional requirement of a toaster, for example, would be that it heats bread when a certain button is pressed. A non-functional requirement is, on the other hand, a description of a property or characteristic that a software system must exhibit or a constraint that it must respect, other than an observable system behavior [Wie03]. Accessibility, compliance, performance, and fault tolerance are all typical non-functional requirements. A non-functional requirement of a toaster would be, for example, that it only uses a certain amount of power to heat bread.

Security requirements, such as the encryption of data, were largely not considered and could be a part of future papers / works. The requirements defined in this chapter will provide the basis for the evaluation and selection of the hardware and software for the test bed.

3.1 Software

This section details the functional and non-functional requirements for the software.

3.1.1 Functional Requirements

1. Node Management

   The IoT test bed will consist of a number of devices (otherwise known as nodes); the characteristics of these nodes needed to identify them as well as supplementary information must be stored and readily available to users. The software must provide users with a interface to add, change, view, and delete nodes. Users must also be able to view and change the configuration of nodes.

2. User Management / Access Control

   The software must provide an interface for adding users in at least two groups: normal users and administrators. Only authorized and authenticated users should be allowed to view and flash nodes; only administrators should be allowed to change the configuration of the system and its nodes.

3. Flashing of Software

   The users must be able to provide software for IoT devices using one or more common formats (i.e. bin, elf) and choose one or more nodes which should be programmed, commonly known as “flashing” when referring to embedded devices. The software should flash the provided software onto the selected node(s) and return the status of

the flashing. The user should then be prompted to start the software or it should be started automatically.

4. Serial Output

   Most embedded devices provide serial output in order to interact with the user and provide logs / debugging information. The software should be able to receive this serial output and return it to the user so that he can see whether or not the software is running as expected and better comprehend what can / must be changed to improve the software using debug messages.

5. Debugging of Nodes

   Many embedded devices provide a debugger (e.g. edbg) which allows the user to receive more detailed information from the node and reset the node, if it is no longer responding or cannot be done using the flashed software. The software must provide the user with the functionality of any available debuggers in order to allow the user to manipulate nodes using his software.

6. Routing of Network Technologies

   The software used by the IoT test bed and its components must allow for routing, especially between different network technologies. The configuration of the components, especially the component which routes packets between networks, must be able to be viewed and changed by an administrator.

7. Resource Management / Scheduler

   The IoT test bed will have limited resources, especially devices, at its disposal. The software must be able to centrally schedule the resources to allow for devices to be used uninterrupted for a certain amount of time by one user; the user should only be able to reserve resources for a limited amount of time so as not to block the system indefinitely for users. The software must provide an interface to schedule / reserve resources transparently and fairly.

8. Central Platform

   The software should provide the aforementioned functionality using a central interface which can be used remotely by users and administrators.

3.1.2 Non-Functional Requirements

1. Scalability

   Scalability is defined as how easily the system can grow to handle more users, transactions, servers, or other extensions [Wie03]. The software should provide the necessary infrastructure to add already supported nodes as well as new unsupported nodes with minimal overhead.

2. Availability

   Availability is defined as the extent to which the system's services are available when and where they are needed [Wie03]. The interface for the configuration and reservation

of nodes as well as the nodes themselves should be available 99% of the time. Maintenance should be able to be performed in batches at times during which the fewest users will want to use the platform. Updates of nodes (e.g. firmware) should be performed simultaneously on multiple units to avoid downtime.

3. Usability

   Usability is defined as how easy it is for people to learn, remember, and use the system [Wie03]. The interfaces provided by the software to meet the described functional requirements should be intuitive for users and administrators. The software should, in addition to intuitive interfaces, be well- documented to provide users, administrators and developers with the means to quickly and efficiently use and, if necessary, modify it.

4. Robustness & Reliability

   Robustness is defined as how well the system responds to unexpected operating conditions; reliability is defined as how long the system runs before experiencing a failure [Wie03]. The software should be able to handle improper user input, including non-functional or corrupt firmware for nodes, as well as other unexpected events.

5. Open Source

   The software should be freely available to the public and use by other organizations should be allowed. The software should be available on a code-sharing platform, e.g. GitHub, that allows anyone to contribute to the code.

3.2 Hardware

This section details the functional and non-functional requirements for the hardware.

3.2.1 Functional Requirements

1. Flashing of Software

   Both the hardware infrastructure as well as the nodes themselves must provide compatible interfaces to allow for the flashing of nodes. The hardware must be accessible and administrable using the centralized platform; thus, the hardware must be interconnected.

2. Routing of Network Technologies

   The hardware must provide the necessary interfaces and connections to allow for the communication and routing of IPv6 packets between different network technologies.

3. On-board Debugger

   The nodes should provide an on-board debugger to allow the user to receive verbose output and reset them as needed.

3.2.2 Non-Functional Requirements

1. Scalability & Modifiability / Modularity

   Modifiability is defined as how easy it is to maintain, change, enhance, and restructure the system [Wie03]. The overarching physical set-up and the hardware infrastructure should also allow for the addition of already supported nodes as well as new unsupported nodes with minimal overhead. The addition of further network technologies should also be considered in the design of the infrastructure.

2. Power / Thermal

   The amount of power should allow for all components to run at full capacity without failure; its use should be minimized, when possible, in order to save energy and reduce the amount of heat produced by components. No component should exceed its maximum operating temperature at any point and should automatically shut down, if this occurs, to prevent permanent damage.

3. Diversity

   The hardware of the test bed should also be sufficiently diverse to enable users to test their software on a number of platforms with different network technologies.

4 Related Work

This chapter describes four existing test beds which are used to test software for embedded systems in a number of countries. The test beds / systems described are all at least partially open source, allowing for the testing of a significant number of embedded devices, and are used by universities and / or companies to test IoT technologies.

4.1 FIESTA-IoT

The Federated Interoperable Semantic IoT Testbeds and Applications (FIESTA-IoT) Platform is an international confederation of heterogeneous test beds which provides a "meta-test bed IoT / cloud infrastructure" and allows users to submit experiments to a number of diverse interconnected test beds [IoT]. The FIESTA-IoT project is organized by the National University of Ireland Galway und funded by the European Commission which contributed approximately €5,000,000 between 2015 and 2018 to the development of the project and a number of confederated test beds as a part of the European Horizon 2020 Research Program [Com].

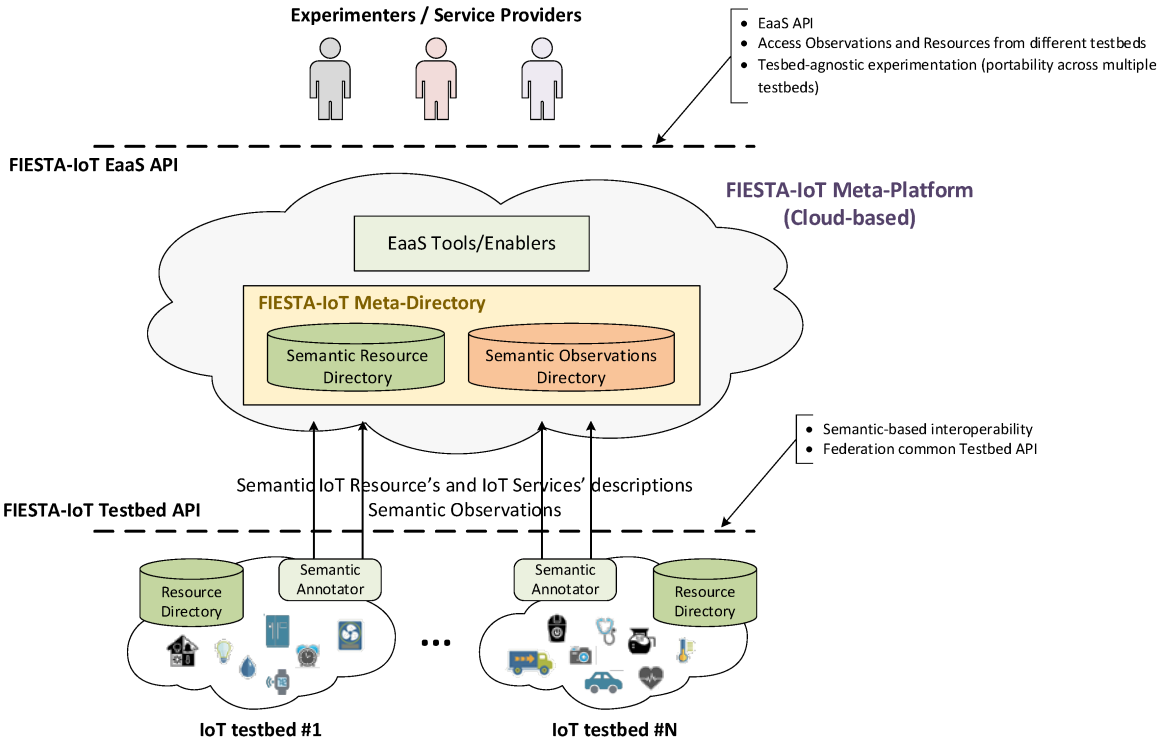


Figure 4.1: FIESTA-Iot Architecture [LSGF+16]

The FIESTA-IoT platform has had multiple rounds of "open calls" that called for researchers and industry professionals to submit their ideas for innovative IoT experiments and test beds. Currently, 10 separate heterogeneous test beds are part of the FIESTA-IoT platform; they include a test bed for large-scale city development, a test bed with hundreds of indoor sensor, and a "crowd-sensing" test bed that uses sensors on phones that measure noise, proximity, speed, and location. Each test bed is run by a separate university or corporation with the goal of testing a specific aspect of IoT technology. The FIESTA-IoT platform also offers registered users the ability to submit smaller IoT experiments to the test beds and to receive anonymized data collected by the nodes [IoT]. The FIESTA-IoT platform focuses mainly on collecting data from a number of sensors and its node are for that reason not designed to be flashed completely. The nodes can instead be sent instructions using an API and the resulting data sent to the user.

A large part of the software used by the FIESTA-IoT platform is open source and available on github; the software for the overarching portal which is used to administer and unify the different test beds is, however, not open source. The platform also does not provide the software for test beds themselves and instead only provides standards and "middleware adapters" for the integration of an existing test bed into the FIESTA-IoT architecture. The software has, however, not been updated since mid-2018 and the portal is not currently in service; this is most likely the result of the completion of the project / funding provided by the European Commission.

4.2 Eclipse IoT Open Testbed

The Eclipse Foundation is a non-profit organization in the United States that focuses on the development of mature, open source, and business-friendly software and projects and is widely known for its Eclipse IDE [EF]. Eclipse hosts and maintains a plethora of industry-friendly open source IoT software projects, which focus on "cross-stack functionality" and the interoperability of different platforms / devices. The provided open source software can also be used in conjunction with other open source solutions or even proprietary solutions that conform to open standards. The Eclipse Foundation does not run any test beds, it simply provides the necessary infrastructure / software to do so.

The Eclipse Foundation does, however, feature selected "Open Test Beds" on its website which were created using its software by companies looking to test specific use cases for IoT technology. A test bed was, for example, created to test asset tracking on a large scale; multiple larger corporations, including Red Hat and Samsung, work actively on the project [EF]. The goal of the test bed is to track cargo / goods in real time and to provide constant information about its status in order to predict and prevent damage, theft, or delay.

The open source software packages offer a wide range of features for dedicated test beds and are well-documented. In order to set up a fully functional test bed, it is necessary to evaluate and set up multiple different software components as well as configure them to work with each other. The software package (Eclipse HawkBit), which is responsible for the distribution of new firmware, requires the integration of an Application Program Interface (API) into the firmware itself to deliver updates.

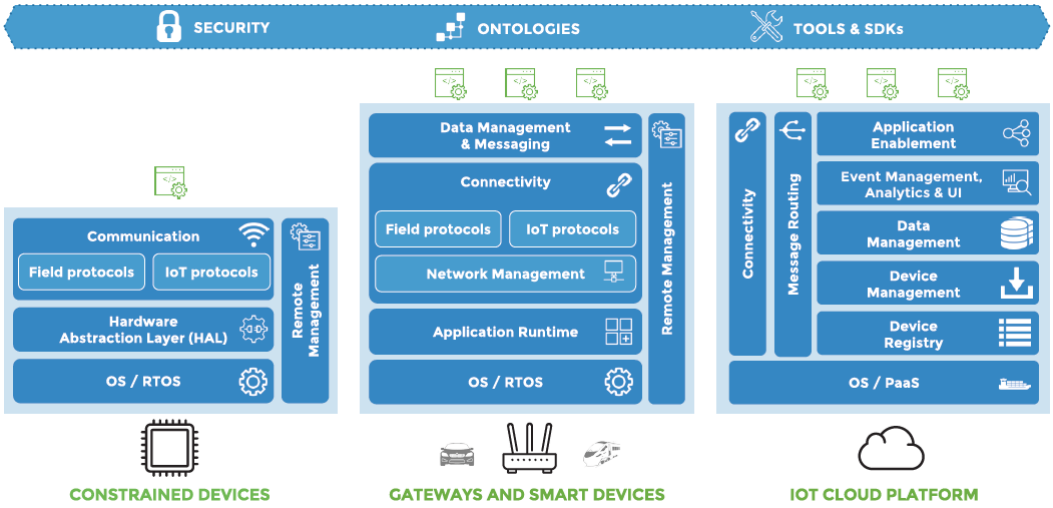


Figure 4.2: Structure of Eclipse IoT Open Source Software / Components [EF]

4.3 FIT IoT Lab

The FIT consortium consists of five French institutions of higher education and research, Pierre and Marie Curie University (UPMC), Inria, University of Strasbourg, Institut Mines Télécom, and French National Centre for Scientific Research (CNRS) which work primarily toward the development of test beds for network computer communications available to industries, scientific researchers, and educators [con]. The FIT IoT-Lab is a confederation of test beds that provide a large scale infrastructure for testing small wireless sensors devices and provide a central platform for the reservation and flashing of nodes [ABF+15]. The open test beds are comprised of 2,728 diverse low-power nodes as well as 117 mobile robots and are deployed across France at six separate sites. Each site consists of at least one server, which handles the low-power nodes and communicates with the "master site" using the provided REpresentational State Transfer (REST) infrastructure.

The central platform provided by the FIT consortium can be accessed using a web application, REST API or open source Command Line Interface (CLI) tools. The FIT consortium provides, in contrast to FIESTA-IoT, the software for the low-level infrastructure, which is split into three pieces, each responsible for a different set of functionalities.

1. The Open Node (ON) is the low-power device the user reprograms, and to which he/she has "bare-metal" access. This node's serial port is connected to the gateway. [ABF+15]

2. The Gateway (GW) is a small Linux computer connected to the serial port of both the ON and the control node. It is itself connected to the backbone over Ethernet. On top of monitoring (and reprogramming) the ON, it forwards the ON's serial activity to the back-end servers. [ABF+15]

3. The Control Node (CN) coordinates ON reprogramming, starts/stops/resets it, and selects its power source (battery or mains). In addition, the CN monitors the ON's

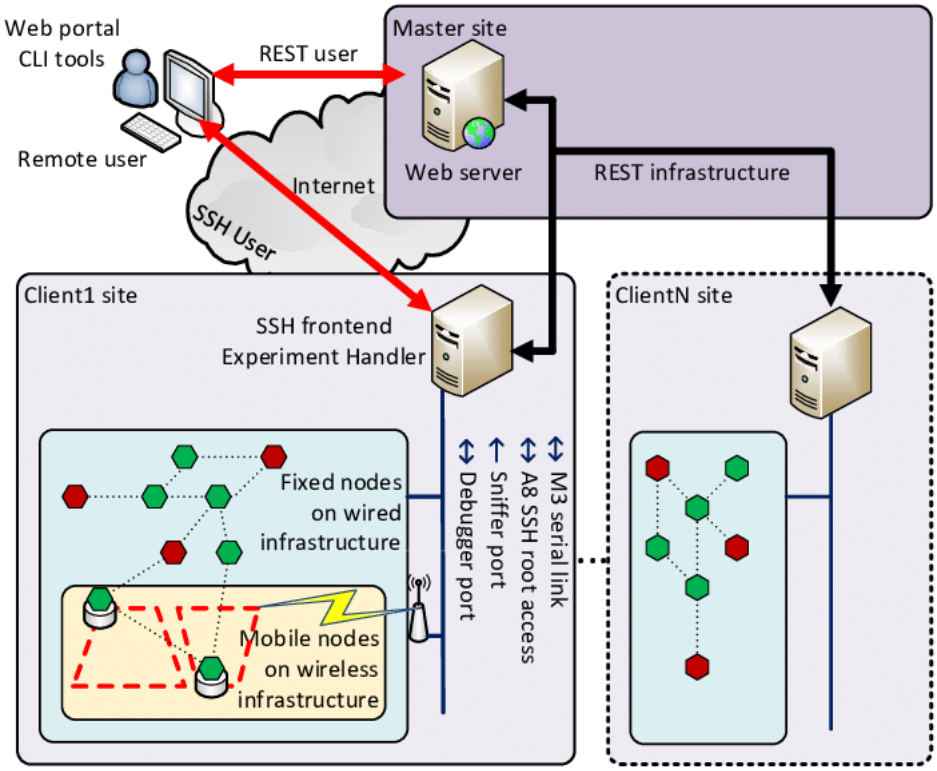


Figure 4.3: FIT-IoT Lab Architecture [ABF+15]

power consumption, drives its sensors, can serve as a wireless packet sniffer or inject arbitrary packets traffic. [ABF+15]

The GW and CN are usually run on the same small linux node, the Host Node (HN), and provide the user who was reserved the node direct access to the serial port of the ON as well as a debugger. The user can use the HN to reset, monitor and reprogram the node as required.

The FIT consortium provides drivers and other libraries for its open nodes to allow users to develop software with as little overhead as possible. In addition to drivers and libraries, the FIT consortium provides APIs, virtual development environments, and code snippets to help beginners learn quickly and hands-on.

4.4 Orange SensorLab2

SensorLab2 was created by Orange Labs, an academic organization in France that contributed to the founding of the FIT IoT lab [LB17]. SensorLab2 was created to provide IoT developers with an "in-the-field IoT Validation Platform". The low-level architecture of SensorLab2, like that of the FIT IoT Lab, consists of a HN which controls the experiment and an ON on which the experiment is run. In contrast to the FIT IoT Lab, the SensorLab2 nodes are designed to be portable in order to test different devices and network technologies

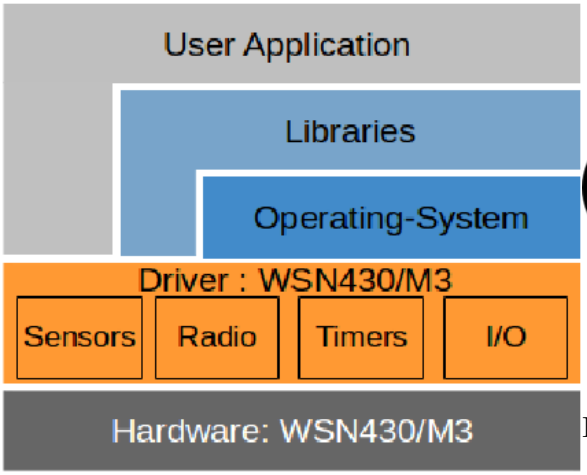


Figure 4.4: Structure of FIT-IoT Lab Tools / Components [ABF+15]

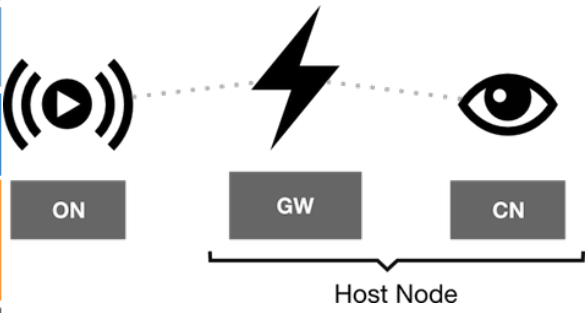


Figure 4.5: Low-Level FIT Architecture [ABF+15]

in diverse environments. The portable node includes a battery, in case there is no power source, and a Wi-Fi Module for the HN, so that it can be connected to a wireless network.

Figure 4.6: Orange SensorLab2 Low-Level Setup [LB17]

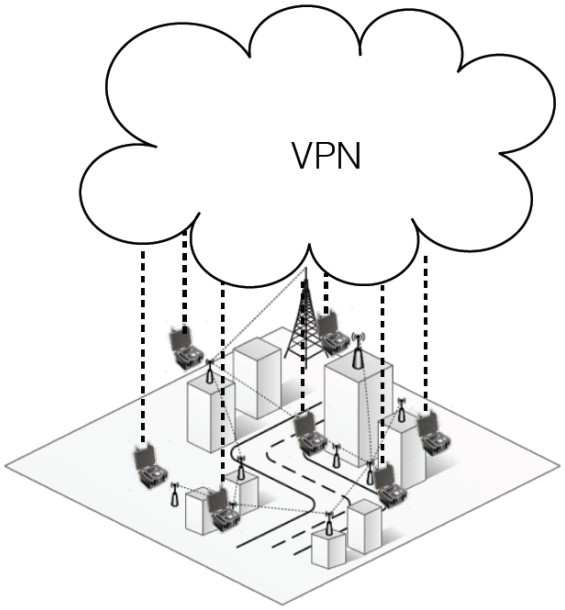


Figure 4.7: Orange SensorLab2 Over-Arching Setup [LB17]

The HN can be connected using Ethernet or Wi-Fi to a network with access to the internet so that it can use a Virtual Private Network (VPN) to communicate with other portable nodes. One or more users can than aggregate and use the data from the nodes, which are connected to the VPN, in different environments to draw conclusions. The SensorLab2 nodes can also be equipped with a 4G USB modem that would allow the HN to connect to other nodes and upload data from almost anywhere. SensorLab2 also provides an observation

toolchain to users to that they can view, analyze, and visualize networking data.

4.5 Comparison

The described open test beds were designed with different goals as well as parameters in mind and provide different levels of functionality. This section provides a summarized comparison of the different systems using the software requirements as defined in the third chapter. All platforms allow for the addition of new supported as well as unsupported nodes and validate user input. Eclipse IoT does not run any test beds and can thus not be compared with regards to hardware requirements. There is not enough information regarding the physical setup of the test beds to meaningfully compare them in terms of thermal and power requirements.

	FIESTA-IoT	Eclipse IoT	FIT IoT Lab	Orange SensorLab2
Software				
Node Management	✓	✓	✓	✗
Flashing of Software	✗	✗	✓	✓
Serial Output	✗	✓	✓	✓
Debugging Interface	✗	✗	✓	✓
Resource Management	✓	✓	✓	✓
User Management	✓	✗	✓	✗
Open Source	Partly	✓	Partly	✓
Central Platform	✓	✓	✓	✗
User Interface	Web REST API	Web	Web REST API CLI	REST API
Hardware				
Flashing of Software	✗	N/A	✓	✓
On-board Debugger	✗	N/A	✓	✓

Table 4.1: Comparison of Examined IoT Test Beds / Architectures

Scalability

The software created / used by the FIT IoT Lab, SensorLab2, and Eclipse all provide modular software which allows for the addition of unsupported nodes. The low-level architecture of the FIT IoT Lab and SensorLab2 both have an important constraint regarding physical scalability, namely that every embedded system must be connected to its own HN. The FIESTA-IoT platform does not provide software for the low-level architecture and the test beds that are part of the FIESTA-IoT platform are too heterogeneous to make a judgment regarding their scalability.

Availability

There is, unfortunately, not sufficient information to compare all test beds described in this chapter with regards to availability. Of the 4 test beds, only the FIT IoT bed is currently functional. The SensorLab2 test bed does not provide a public central interface;

the platforms used by FIESTA-IoT and the featured open test beds which use the software created by Eclipse are currently not available / offline.

Robustness & Reliability

The low-level structure of the FIT IoT Lab and the SensorLab2 allows for embedded systems to be flashed and debugged regardless of their current state; those two test beds are for that reason both robust. Not enough information is available regarding the robustness of the software provided by FIESTA-IoT and Eclipse to make accurate statements.

Diversity

The FIT IoT Lab offers support for at least 10 different embedded systems, which mostly use LoRa and 802.15.4 for communication with other systems. The FIESTA-IoT platform encompasses through its multitude of heterogeneous test beds at least 15 different types of sensors, which monitor a number of different environments.

Routing of Network Technologies

The FIT IoT Lab explicitly allows for the testing of the interoperability of different network technologies at their location Inria Lille but does not provide the necessary infrastructure to route IPv6 packets between them. The other test beds described in this chapter do not explicitly make provisions for these scenarios.

5 Design & Architecture

This chapter documents the design and the planned architecture of the test bed at the University of Munich.

5.1 Software

This section describes the decision process regarding the selection of software for the test bed and the extent of customization.

5.1.1 Selection & Structure of Base Software

The software used by the FIT IoT Lab fulfills all functional requirements for software as defined in the third chapter and is partially open source. The open source software as well as the closed source REST API are well-documented. The FIT IoT Lab also provides a number of different interfaces for interaction with the user. The robustness of the system is ensured through the unique low-level architecture which stipulates that each ON is directly connected to and controlled by a HN, which can reset and flash the ON regardless of its current state. The open source community currently working on the FIT IoT Lab is active; many University of Munich students and faculty members have already used the FIT IoT Lab to test their firmware. The test bed described in this thesis was for these reasons created using the open source software provided by the FIT consortium.

The source code of the software run on the HNs in the FIT IoT Lab is written in C++ and Python and structured as shown below; only relevant lower level folders are shown.

```
/
├── bin
│   ├── rules.d
│   └── ...
├── control_node_serial
├── doc
├── gateway_code
│   ├── open_nodes
│   │   └── common
│   ├── static
│   ├── utils
│   └── ...
├── tests
└── tests_utils
```

Each type of ON (e.g. Arduino Zero) must be defined in the following ways:

1. "node_" Python class (e.g. node_arduino_zero.py), which defines how a node is flashed and debugged, stored in the "open_nodes" folder.

2. udev rules, which help the software identify the node when connected and differentiate it from other nodes, stored in the "rules.d" folder. The software uses the rules to create a symbolic link (symlink) in the folder for all connected devices, /dev/iotlab.

3. firmwares, which are flashed by default to help save energy and test the nodes, stored in the "static" folder. A minimum of 2 standard firmwares, idle and autotest, are required.

The "utils" folder stores the Python interfaces for the software used to flash the ONs (openocd, edbg, etc.).

On start-up, the HN looks for the required configuration files in the configuration directory, stored in the environment variable, IOTLAB_GATEWAY_CFG_DIR (default: /var/local/-config). The configuration directory contains the following files:

1. board_type, which contains the name of the board as defined in the open_nodes folder.

2. control_node_type, which contains either iotlab or no depending on whether or not a custom FIT IoT Lab Node is connected.

3. hostname, which contains a unique name for the HN.

The HN does not store any information regarding the serial number of the attached ON, since it does not need to differentiate between different nodes. The port for the serial redirection of the connected ON is hard-coded (20000) and cannot be modified. The software provides a simple REST API using the Python bottle library, which allows a user to flash the connected ON. The gateway is a service that can be stopped and started using init.d. The REST API provides three interfaces for experiments / the flashing of software:

1. /exp/start/{exp_id}/{user}, which allows a user to start an experiment by providing an elf file as input for the flashing of the ON

2. /exp/stop, which allows a user to stop the currently running experiment by flashing the idle firmware.

3. /status, which allows a user to view the current status of the ON.

5.1.2 Necessity & Extent of Adjustments

1. Multiple ONs per HN

   The current structure of the FIT IoT Lab requires that each HN manages exactly one ON. However, under normal circumstances the small computers which normally serve as HNs have the ability to support multiple devices at the same time. In order to save space and reduce the costs of the test bed the software should be changed to allow for multiple ONs to be attached to one HN.

2. Redundant Storage / Memory

   The small computers that serve as HNs also require that each computer has its own storage device, which leads to large redundancies as well as unnecessary down time and bottlenecks when performing system updates. The software should be run using shared networked storage in order to reduce the amount of redundant data and increase the efficiency of updates. Each HN should have its own log and configuration folders to ensure that debugging information and configuration files can be separately saved.

3. Update / Compatibility of Software for HNs

   The open source software currently available for the low-level infrastructure created by the FIT consortium was designed to be run on the Raspberry Pi 2 using Ubuntu 14.04 LTS. The Raspberry Foundation has since released multiple newer, faster, and equally inexpensive versions of the Pi computer; multiple newer, stable versions of Ubuntu have also been released. The software used by the HNs must be updated for the new hardware and operating system.

4. Addition of Unsupported Nodes / Programmers

   The ONs currently supported by the software provided by the FIT consortium are, for the most part, custom-designed nodes. The software must be extended to support the ONs, which are listed later in this chapter.

5.1.3 Temporary Site Server

The software used for the site and master servers of the FIT IoT Lab are, unfortunately, not part of the open source software. The University of Munich has since contacted the FIT consortium and is currently taking part in the negotiation of a contract that would allow it to use the closed source software to integrate its test bed into the larger FIT ecosystem. In the meantime, in order to centrally and remotely manage / use the platform, a custom "site server" should be developed to provide the basic functionality required to use the nodes before the integration into the FIT ecosystem.

5.2 Hardware

This section lists necessary hardware components, the criteria used for their selection as well as the design of the over-arching setup, including enclosure and casing.

5.2.1 Criteria for & Selection of Open Nodes

This section lists criteria which should in addition to the hardware requirements described in the third chapter be used to select the ONs for the test bed.

Criteria

1. Supported Operating Systems

   Many operating systems for IoT devices only support specific devices and / or architectures. The selected nodes should be supported by as many operating systems as possible and must be supported by RIOT OS.

2. RAM

   The amount of Random Access Memory (RAM) available to the software that will be run on the node can greatly affect the speed of an application and the necessity of software optimizations. The amount of RAM should differ between nodes in order to provide users with a diverse test bed and thus the ability to test their software on embedded systems with different constraints.

3. Storage

   The amount of programmable storage available is important because it acts as a constraint for the development of software. The storage is due to the compact nature of embedded devices usually "flash" and provides the space for firmware to be programmed.

4. Processor

   The speed and architecture of the processor are important factors to consider in regards to the diversity of the test bed. Like the amount of RAM, the speed of the processors should differ in order to create a diverse test bed.

5. Available Connectors

   Each embedded system should provide at least one UART connection (using a USB port) so that it can be easily integrated into the architecture of the FIT IoT Lab.

6. Hat / Pin Layout

   The layout of the connectors for extensions / pins should conform to one standard to allow for all hats to be used by different devices in order to create a modular test bed. The Arduino Uno V3 Pin Layout has established itself in the IoT ecosystem and is used by a plethora of hats and embedded devices, many of which are produced by manufacturers other than Arduino.

7. Price

   The price of the ONs is, due to the limited budget of the University of Munich, also a factor. The University of Munich already owns a number of embedded systems that can be used in a test bed; these embedded systems should be, if possible, used to create the test bed.

Selection

Table 5.1 compares a number of embedded systems already owned by the University of Munich as well as embedded systems already used / supported by the FIT IoT Lab. All listed embedded systems support FreeRTOS and RIOT OS. Only Arduino Due and WSN43 also offer support for TinyOS. In order to create a diverse and cost-effective test bed, the following nodes were chosen: Nucleo-F091RC, Nucleo-F103RB, Nucleo-F411RE, Nucleo-F767ZI, Arduino M0 Pro. The choice of ONs includes embedded systems from C1, C2, and C3 to ensure that users can test software under a number of conditions. The selected nodes, although they greatly differ in terms of processing power and storage, all use the Arduino Uno V3 pin layout to ensure the modularity of the nodes with regards to peripheral devices. The Nucleo nodes all share a common debugging interface developed by their manufacturer, STMicroelectronics, in addition to the common pin layout.

Micro-controller	RAM	ROM	Class	Architecture / Processor	Connectors	Arduino Uno Layout	Price
Nucleo-F091RC	32 KB	256 KB	C2	ARM Cortex M0	USB	✓	€9.37
Nucleo-F103RB	20 KB	128 KB	C1	ARM Cortex M3	USB	✓	€9.37
Nucleo-F411RE	96 KB	512 KB	C2	ARM Cortex M4	USB	✓	€11.80
Nucleo-F767ZI	512 KB	2 MB	C3	ARM Cortex M7	USB, Ethernet	✓	€20.89
Arduino M0 Pro	32 KB	256 KB	C2	ARM Cortex M0	2 x USB	✓	€27.90
Arduino Due	96 KB	512 KB	C2	ARM Cortex-M3 Rev 2	2 x USB	✓	€35.00
WSN43	10 KB	48 KB	C0	RISC TI-MSP430-F1611	USB, IEEE 802.15.4 Radio	✗	Custom FIT Node
Atmel SAMR21	32 KB	256 KB	C2	ARM Cortex M0	2 x USB, IEEE 802.15.4 Radio	✗	€56.97
Atmel SAMR30	32 KB	256 KB	C2	ARM Cortex M0+	2 x USB, IEEE 802.15.4 Radio	✗	€58.88
nRF52840 DK	256 KB	1 MB	C3	ARM Cortex M4	USB, Bluetooth 5 / IEEE 802.15.4 Radio	✗	€42.85
TI CC3200	256 KB	1 MB	C3	ARM Cortex M4	USB, 802.11 b/g/n Radio	✗	€30.88
TI CC3220SF	256 KB	1 MB	C3	ARM Cortex M4	USB, 802.11 b/g/n Radio	✗	€51.47
ESP8266 DevKit	80 KB	1 MB	C2	RISC Tensilica L106	USB, 802.11 b/g/n Radio	✗	€7.20

Table 5.1: Comparison of Selected Embedded Systems[1][2]

[1]All Prices Brutto (Including VAT)
[2]Prices found by various distributors, including Mouser, Reichelt and Amazon

5.2.2 Selection of Host Nodes

Small Linux-capable computers are also required to create the necessary infrastructure for the test bed. The Raspberry Pi 3B+, the newest release of Raspberry Pi computer, was chosen for the following reasons:

1. Form Factor

   The dimensions of the Raspberry Pi 3B + are 82mm x 56 mm x 19.5mm. The computer is thus as small as most embedded systems and can be easily integrated into an environment with limited space.

2. Power over Ethernet (PoE) Capability

   The Raspberry Pi 3B+ is the first computer produced by the Raspberry Pi Foundation which supports PoE, allowing for it to be powered using an Ethernet cable and a PoE capable switch.

3. Network Booting

   The Raspberry Pi 3B+ is the first computer produced by the Raspberry Pi Foundation which is capable of networking booting (e.g. using Trivial File Transfer Protocol (TFTP)) out of the box (without modifications).

4. Documentation / Support

   The Raspberry Pi Foundation provides detailed documentation of the hardware, ready-to-use software packages and operating systems, and a popular forum which allows users to provide support to each other.

5. Ecosystem (Peripherals)

   The Raspberry Pi 3B+ is outfitted with a 40-pin General-Purpose Input/Output (GPIO) header which enables the Raspberry Foundation and other companies to produce a multitude of different peripheral devices / shields for it.

6. Connectors

   The Raspberry Pi 3B+ includes, among other, the following ports: Gigabit Ethernet, 2.4 / 5 GHz Wi-Fi, Bluetooth (4.2 und BLE), 4 2.0 USB Ports. The number of USB ports would allow the computer to connect simultaneously to 4 different ONs; the onboard Wi-Fi and Bluetooth Antennas would allow it to act as a gateway for ONs using those technologies.

7. Price

   The Raspberry Pi 3B+ is available commercially in Europe for €32.90 including VAT.

No other small linux-capable computer currently on the market offers the described features and value for money.

5.2.3 Selection of Network Technologies & Extensions / Hats

The diverse network technologies described in the second chapter provide a wide range of functionalities. ZigBee / 802.15.4 and LoRa were chosen as the 2 wireless technologies to be

use due to their many different characteristics. LoRa is used to send messages over large distances; ZigBee, on the other hand, is used to send messages over short distances. LoRa can, depending on the selected data rate (frequency), be classified as a S1 or S2 network technology, while ZigBee is a S1 network technology. Both technologies can, however, communicate using IP and the 6LoWPAN protocol. Ethernet was chosen as a traditional alternative to its younger wireless counterparts and is classified as a S3 network technology.

5.2.4 Design and Measurements of Overarching Set-up

The enclosure used to hold the test bed will be compact, secure, and modular. The enclosure will allow for two shelves; the HNs, and ONs to be placed as shown in Figure 6.6 and Figure 6.5. The HNs will be placed on either side of each shelf; theONs will be placed along the top and bottom of each shelf. Both HNs and ONs will be stacked in order to save space and minimize the required cabling. The gateways will be placed on the top shelf in the top left corner away from the other nodes to reduce the number of cables on the shelf.

5.2.5 Power & Thermal Requirements

The power and thermal requirements as described in the third chapter must also be considered in the design of the test bed.

Power

A Raspberry Pi 3B+ requires 1.9W when idling and 5.1W when at 400% CPU load [Dra]. Embedded devices use significantly less power, under most circumstances under .5 W. [Cos16]. The power source used by the nodes should be able to support all Raspberry Pi 3B+'s and attached embedded systems at full capacity.

Thermal

Both the Raspberry Pi 3B+ and the selected embedded systems have an official operating temperature range between -40° C and +85° C. The Raspberry Pi 3B+ thermally throttles its processor if the temperature nears its upper bound and, in the worst case, shuts down to prevent damage [Hea16]. Since the embedded systems are directly connected to the Raspberry Pi 3B+, this protection extends to them as well. The enclosure will be well-ventilated and allow for both active and passive cooling of the hardware to prevent overheating.

5.3 Network Topology

An IPv4 management network will be used to allow for communication between the switch, the site server, and the HNs. The management network will be separated from the ONs that use Ethernet by creating a separate Virtual Local Area Network (VLAN) for those nodes. The site server will be connected to both the management network and the internet using two interfaces in order to provide the management network with access to the internet. The gateways for LoRa and ZigBee will be connected to the management network as well as the network for ONs using a VLAN trunk to allow for the routing of IPv6 packets between their respective network technologies, ONs using Ethernet, and the site server. The topology used is shown in Figure 5.1.

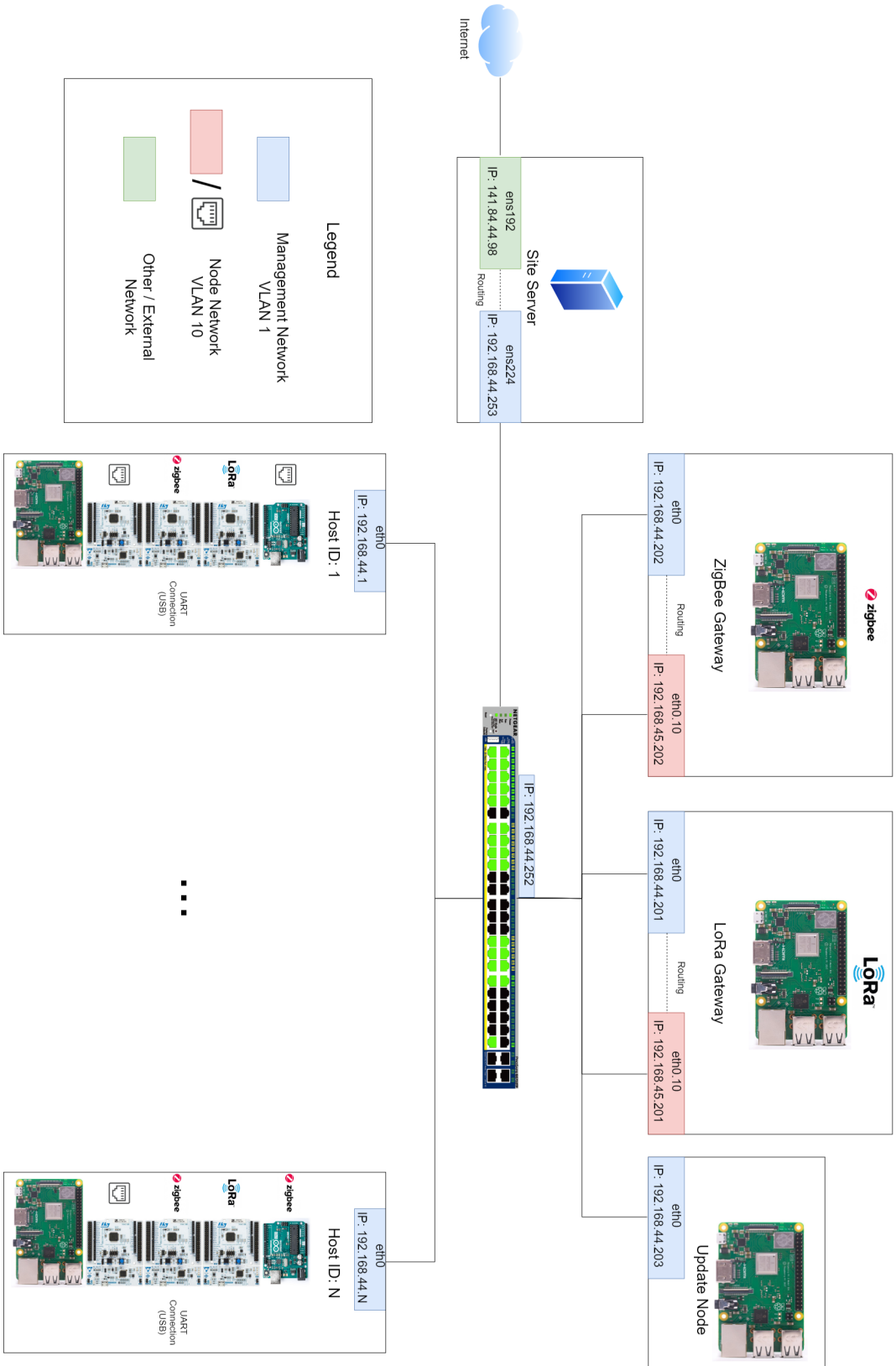


Figure 5.1: Network Topology / Setup

6 Implementation & Physical Set-up

This chapter documents the implementation of the design defined in the fifth chapter.

6.1 Implementation / Adjustment of Software

This section documents the customization of the open source software provided by the FIT consortium as well as the implementation of the temporary site server.

6.1.1 Multiple Open Nodes per Host Node / Update of Software

In order to run the software using newer versions of Debian/Ubuntu, which no longer use init.d to manage services, a systemd unit file had to be created. To allow for the Raspberry Pi 3B+ to take full advantage of all 4 of its USB ports simultaneously, 4 instances of the gateway service were created. There are a total of 4 unique sets of files and directories for each instance of the software running on each HN:

1. Systemd Unit File - /etc/systemd/system/gateway-server{“”,-2,-3,-4}.service

   The systemd unit file allows the user to start and stop the service and passes along the environment variables to the respective service.

2. Configuration File - /etc/gateway{“”,2,3,4}.conf

   The configuration file holds the environment variables to be used by the respective service.

3. Configuration Directory - /var/local/config{“”,2,3,4}

   The configuration directory contains the files described in 5.1.1. In addition to the files already stored in the configuration directory, there must be a file which stores the serial number of the connected ON as well as a file which stores the port over which a Transmission Control Protocol (TCP) socket provides the serial output of the ON.

4. Log Directory - /var/log/gateway-rest-server {“”,-2,-3,-4}

   The log directory contains the logs created by the respective service.

Systemd Unit File

Systemd passes along the listening address, port, and log directory to the REST server and ensures that the configuration directory exists before starting the service, using the following unit file:

```
[Unit]
Description=Gateway Server
After=nfs-mount.service
ConditionPathExists=/var/local/config
```

```
5
6  [Service]
7  Type=simple
8  User=www-data
9  Group=www-data
10 StandardOutput=syslog
11 StandardError=syslog
12
13 EnvironmentFile=/etc/gateway.conf
14 ExecStartPre=/bin/sleep 1
15 ExecStart=/usr/local/bin/gateway-rest-server $LISTEN $PORT $LOG
16 Restart=on-failure
17
18 [Install]
19 WantedBy=multi-user.target
```

The gateway.conf file for the first instance of the gateway server is as follows:

```
1 LISTEN=0.0.0.0
2 PORT=5060
3 LOG="--log-folder=/var/log/gateway-rest-server"
4 IOTLAB_GATEWAY_CFG_DIR=/var/local/config
```

board_config.py

Board_config.py is the interface used by the software to read the configuration files stored in the configuration directory. Two additional files, port and serial, have to be read during setup using the following 2 lines:

```
1         self.serial = config.read_config('serial')
2         self.port = config.read_config('port')
```

stlink.rules

The udev rules defined for each device / manufacturer create a symlink with a hard-coded name. This results in the creation of only one symlink when multiple boards of one type are connected. Stlink.rules provides the udev rules for all Nucleo boards and uses a shell script (serial.sh) to append the serial number of a node to the end of its symlink. A common Linux tool for embedded systems, udevadm, can be used to determine the information needed for udev rules.

```
1 KERNEL=="ttyACM*",SUBSYSTEM=="tty", ATTRS{idVendor}=="0483", ATTRS{idProduct
    }=="374b", SUBSYSTEMS=="usb", PROGRAM="/bin/serial.sh %k", SYMLINK+="
    iotlab/ttyON_STLINK%c"
2 SUBSYSTEM=="usb", ATTRS{idVendor}=="0483", ATTRS{idProduct}=="374b", MODE="
    0664", GROUP="dialout"
```

serial.sh

Serial.sh uses a single argument, the original dev path to an embedded system, and udevadm to return its serial number.

```
1 #!/bin/sh
2 udevadm info -a -n /dev/$1 | grep '{serial}' | head -n1 | cut -d'"' -f 2
```

node_st_link.py

Node_st_link.py is the super class used by all Nucleo devices. In order to pass on the serial number of the ON and the port to be used for serial redirection, both variables must be saved. The serial number is directly appended to the default dev path and saved in the variable "TTY". These parameters are then passed on to the programmer / debugger; in this case Open On-Chip Debugger (OpenOCD) is used to flash Nucleo devices.

```
    serial = gateway_code.config.read_config('serial')
    port = gateway_code.config.read_config('port')
    if serial:
        TTY += serial.upper()
    BAUDRATE = 115200
    DIRTY_SERIAL = True
```

openocd.py

Openocd.py provides the interface for the installed OpenOCD programmer. In order to use the serial number to flash a specific ON, it must be passed using the "hla_serial" parameter; the parameter has been added to the template used for OpenOCD commands.

```
    OPENOCD = ('{openocd_path} --debug=0'
               ' {config}'
               ' -c "hla_serial "{serial_port}""'
               ' -c "init"'
               ' -c "targets"'
               ' {cmd}')
```

The serial port must also be passed to the flasher. The serial number is then capitalized to ensure that it coincides with the serial number provided by the udev rules.

```
        if board_cfg.serial_number:
            self.serial_port = board_cfg.serial_number.upper()
```

serial_redirection.py

Serial_redirection.py creates a TCP socket to allow users to directly access the ON they have just flashed. The hard-coded port in the Socat command is replaced with a variable.

```
    SOCAT = ('socat -d'
             ' TCP4-LISTEN:{port},reuseaddr'
             ' open:{tty},b{baud},echo=0,raw')
```

This variable is accepted as a parameter by its constructor while retaining the original default of 20000. This argument is passed on by the respective node class.

```
    def __init__(self, tty, baudrate, port=20000):
        self.process_cmd = shlex.split(self.SOCAT.format(tty=tty,
                                                         baud=baudrate,
                                                         port=port))
```

6.1.2 Storage / Memory

The Raspberry Pi 3B+ can boot over a network using TFTP and Network File System (NFS) and thus does not require an SD Card; this process in described on the website

of the Raspberry Foundation [FOUb]. To that end, three services were installed on the virtual machine responsible for the temporary site server: Dynamic Host Configuration Protocol (DHCP) (isc-dhcp-server), TFTP (tftpd), NFS (nfs-kernel-server). The DHCP server provides the Raspberry Pi 3B+ with an IP Address and uses the "tftp-server-name" option to make it aware of the TFTP server which provides the files necessary to boot. The TFTP server allows the HN to boot and provides the configuration that allows it to mount an NFS repository from which the full operating system, Raspbian Stretch, can start. The operating system mounts the NFS store as a read-only file system and creates a temporary overlay file system using an open source script available on blockdev.io [Rid18].

The root file system created using OverlayFS allows for the temporary modification of read-only files and folders and thus enables multiple HNs to function normally while using one NFS store. These temporary changes are irrevocably lost when the HN is restarted. In order to provide each HN with unique and consistent configuration and log directories, two additional NFS stores per HN are created. These unique directories are named using the MAC address of each HN and are mounted during the boot process using a shell script, as shown below. This shell script is run on startup before the gateway services are started using a systemd unit file.

```
#!/bin/bash
fail(){
  echo -e "$1"
  /bin/bash
}
maca="$(ip a show dev eth0 | awk '/ether/ {print $2}')"
mount -t nfs 192.168.44.253:/nfs/clientconfig$maca /var/local/
if [ $? -ne 0 ]; then
    fail "ERROR: Could not mount IoT Lab Config from NFS Store"
fi
mount -t nfs 192.168.44.253:/nfs/clientlog$maca /var/log/
if [ $? -ne 0 ]; then
    fail "ERROR: Could not mount Log Directory from NFS Store"
fi
```

The final structure used to boot the HNs is structured as shown:

```
/
├── tftp
└── nfs
    ├── client2
    ├── configdefault
    ├── logdefault
    ├── clientconfig{MAC}
    ├── clientlog{MAC}
    └── ...
```

6.1.3 Addition of Unsupported Nodes

The process of adding new unsupported ONs to the existing FIT infrastructure is described in its GitHub wiki [1]. As described in 5.1.1, each ON requires at least 4 new files: a node_Python class, udev rules and 2 firmwares. The Arduino M0 Pro is already supported by the software

[1] https://github.com/iot-lab/iot-lab-gateway/blob/master/DEVELOPER.md

and can be flashed by being treated as an Arduino Zero, which shares almost all significant characteristics with the Arduino M0 Pro. The following section documents the process of adding the Nucleo F103RB to the FIT infrastructure. The other Nucleo nodes (F411RE, F091RC, F767ZI) were added analogously.

node_st_f103rb.py

```
class NodeStf103rb(NodeStLinkBase):
    """ Open node STM32 IOTNODE implementation """

    TYPE = 'st_f103rb'
    OPENOCD_CFG_FILE = static_path('st-f103rb.cfg')
    OPENOCD_PATH = '/opt/openocd/bin/openocd'
    FW_IDLE = static_path('st-f103rb_idle.elf')
    FW_AUTOTEST = static_path('st-f103rb_autotest.elf')
```

st-f103rb.cfg

The configuration file for the programmer must also be created; the parameters used by RIOT OS to flash a node can often be used.

```
source [find interface/stlink-v2-1.cfg]
transport select hla_swd
source [find target/stm32f1x.cfg]
reset_config srst_only
$_TARGETNAME configure -rtos auto
```

The firmwares can be created for a new board by compiling them using RIOT OS. The source code can be found in a GitHub repository[2] run by the FIT consortium. The firmwares should be compiled normally using RIOT OS and the resulting ELF files renamed to fit the naming convention, for example: st-f103rb_autotest.elf, and st-f103rb_idle.elf

6.1.4 Temporary Site Server

The temporary site server was created using python and the flask library and provides a simple, no-frills interface for administering and using the test bed. The site server can be started and stopped using a systemd service called site-server created in conjunction with gunicorn. The data required by the site server is stored in a MySQL database which is run on the same server. The data stored in the MySQL database encompass all necessary information in 3 tables as shown in Figure 6.1. The site server does not address security concerns or provide a user management system.

Node Management

The site server provides 5 interfaces for adding and managing nodes:

- Add HN (/addHostNode)

  This interface allows a user to input the MAC Address as well as an optional description of a new HN and returns its ID if successful. The following steps are executed by the site server when adding a HN:

[2]https://github.com/iot-lab/ci-firmwares/tree/master/firmwares

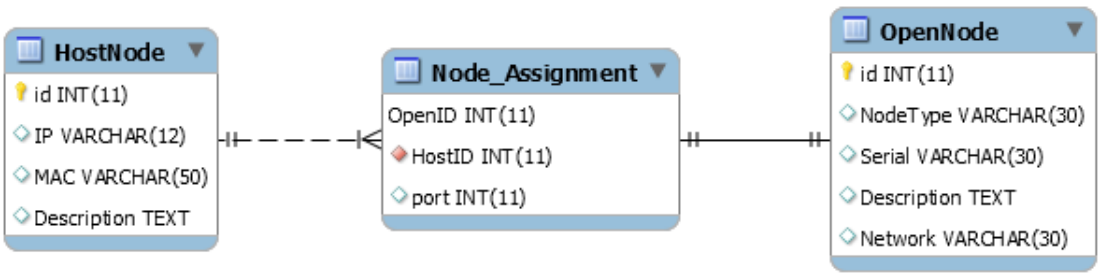


Figure 6.1: Database Structure as Entity Relationship Diagram (ERD)

1. A new entry in the HostNode SQL Table is created and a static IP address is assigned.
2. A new IP reservation entry is appended to the configuration for the DHCP server (/etc/dhcpd/dhcp.conf); the DHCP service is forced to reload the configuration after the change.
3. Unique log and configuration directories for the node are generated by copying pre-configured default directories.
4. The new unique directories are added to the configuration of the NFS server (/etc/exports); the NFS service is forced to reload the configuration after the change.
5. The ID of the new HN is returned. The HN can now be connected / restarted.

- Add ON (/addOpenNode)

  This interface allows the user to choose the type of node and network technology as well as enter the serial number and an optional description of a new ON and returns its ID when successful. The data is stored in the OpenNode SQL table and cannot be flashed until assigned to a HN.

- View HNs (/allHostNodes)

  This interface provides an list of all HNs and their characteristics.

- View ONs (/allOpenNodes)

  This interface provides an list of all ONs and their characteristics.

- Assign ONs to HNs (/connectNode)

  This interface allows the user to assign ONs to a HN by drag and drop. Both types of node are identified using their respective IDs. The following steps are executed by the site server when an ON is assigned to a HN:

  1. The assignment is saved as an entry in the Node_Assignment SQL table.
  2. The gateway server services are stopped and the configuration folder unmounted.
  3. The files in the configuration directory of the HN are modified and the port is assigned to the respective instance.
  4. The configuration folder is remounted and the gateway server services are restarted.

5. If the ON was previously assigned to another HN, steps 2-4 are repeated for that HN.

Flashing

The interface to flash ONs is located under /multFlash and allows a user to provide an ELF file and select up to 3 nodes to be flashed. The following steps are executed by the site server when one or more ONs should be flashed.

1. The site manager saves the uploaded firmware file in a temporary directory.
2. All 3 SQL tables are joined as to ascertain the correct IP address and port of the ONs
3. Each ON is flashed one after the other by accessing the REST API provided by the HN.
4. The temporary file is deleted and the raw output of the REST API calls is returned. If the value of the variable ret is 0, then the respective ON has been successfully flashed.

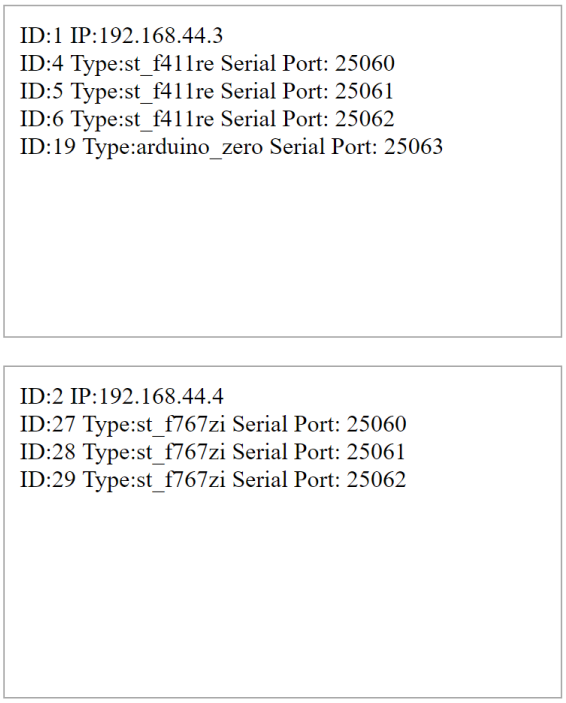


Figure 6.2: Assignment Web Interface

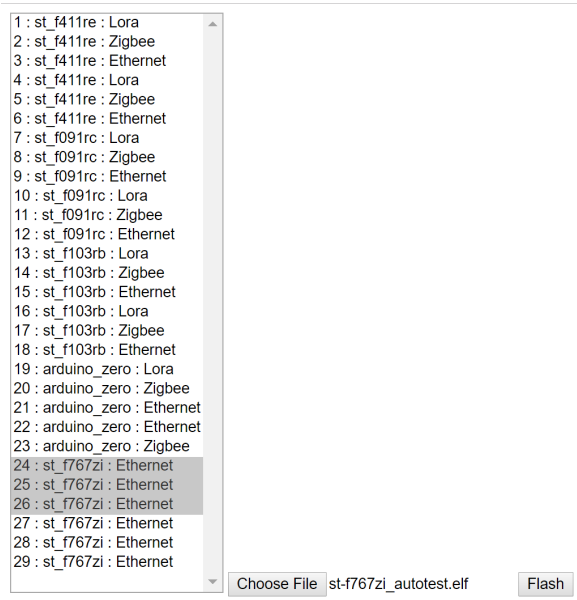


Figure 6.3: Flashing Web Interface

The output after the successful flashing of node 24 could look like the following:

```
Node 24:
(0, ' % Total % Received % Xferd Average Speed Time Time Time Current
 Dload Upload Total Spent Left Speed
0 0 0 0 0 0 0 0 --:--:-- --:--:-- --:--:-- 0
100 823k 0 0 100 823k 0 818k 0:00:01 0:00:01 --:--:-- 818k
100 823k 0 0 100 823k 0 410k 0:00:02 0:00:02 --:--:-- 410k
100 823k 0 0 100 823k 0 273k 0:00:03 0:00:03 --:--:-- 273k
```

```
100  823k  0  0  100  823k  0  205k  0:00:04  0:00:04  --:--:--  205k
100  823k  0  0  100  823k  0  164k  0:00:05  0:00:05  --:--:--  164k
100  823k  0  0  100  823k  0  145k  0:00:05  0:00:05  --:--:--  0
100  823k  100  10  100  823k  1  144k  0:00:10  0:00:05  0:00:05  0
{"ret": 0}')
```

6.2 Implementation of Gateways

This section documents the software used by the Raspberry Pi 3B+ gateways for the selected network technologies, LoRa and ZigBee / 802.15.4.

6.2.1 ZigBee / 802.15.4

The base image used for the ZigBee / 802.15.4 gateway was created using the guide[3] provided by the creators of RIOT OS. The configuration of the gateway is stored consistently using a package developed especially for the Raspberry Pi, which is available in the same repository[4]. In order to use the gateway with the same ZigBee modules that are used by the ONs, the configuration in the /boot/config.txt file must be changed as follows:

```
dtoverlay=at86rf233,reset=4,interrupt=24
```

6.2.2 LoRa

The LoRa Gateway was created using a pre-built image for the Raspberry Pi 3B+ called Waziup, which is available in a public GitHub repository[5]. The gateway can be configured using a web interface and per SSH. The gateway does not yet offer support for 6LoWPAN; this extension could be a part of future work.

6.3 Physical Set-up

This section details the physical set up of the test bed including the high- and low-level wiring and a full overview of all hardware used.

6.3.1 Overarching Set-up

A compact rack enclosure (600 x 505.1 x 600mm) produced by Digitus was purchased along with 2 shelves (40 x 32cm) to house the test bed. The rack enclosure can only be opened from the front and sides using a key and thus ensures that only authorized persons can access it. It also provides multiple slots for fans in order to allow for the use of active cooling when necessary. The designs for the shelves as shown in Figure 6.5 and Figure 6.6 were printed to scale and used to drill holes into the wooden boards so that the ONs can be properly secured.

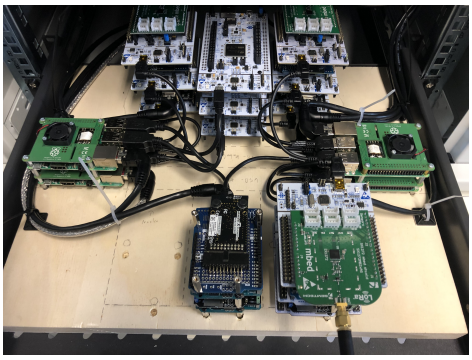
A Netgear Gigabit PoE-capable Ethernet switch with a total of 48 Ethernet ports was purchased; the switch can provide up to a total of 380W of power on all ports using PoE.

[3] https://github.com/RIOT-Makers/wpan-raspbian/wiki/Create-a-generic-Raspbian-image-with-6LoWPAN-support

[4] https://github.com/RIOT-Makers/wpan-raspbian

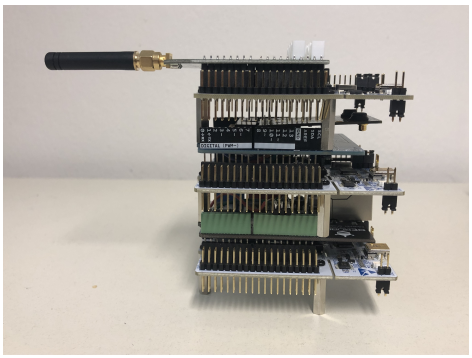
[5] https://github.com/CongducPham/LowCostLoRaGw

(a) Front View of Rack

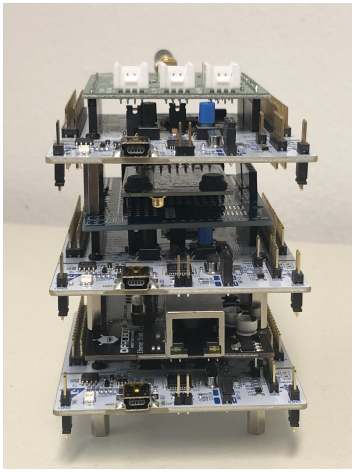


(b) Bottom Shelf Rack Shelf

(c) Top Rack Shelf



(d) ON Stack Side View



(e) ON Stack Front View

Figure 6.4: Physical Set-up & ON Stack

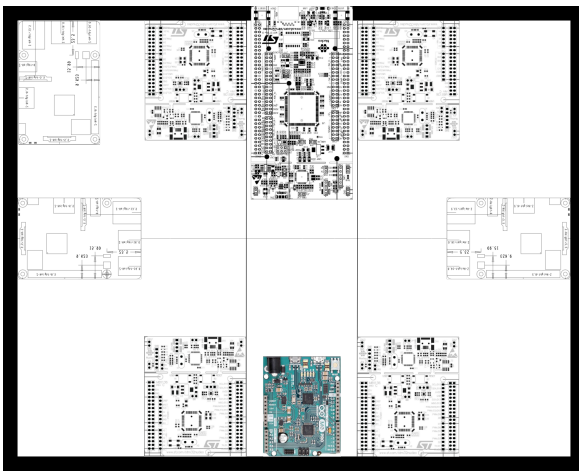


Figure 6.5: Design of Top Rack Shelf

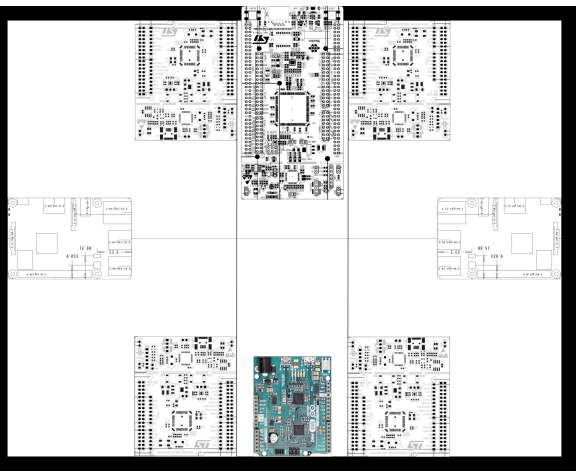


Figure 6.6: Design of Bottom Rack Shelf

This constellation enables the test bed to only require two cables to leave the enclosure, the power cord for the switch, and the Ethernet connection to the site server.

Short, angled USB Cables (10cm) were bought to reduce the "cable mess" and prevent unnecessary wear and tear that could be caused by bending the cables too strongly. Most of the Ethernet cables used are also angled for the same reason. The angled USB cables were used to connect the ONs directly next to the HNs; normal USB cables (15cm) were used to connect the ONs in the middle. Cable clips were also bought to ensure that all Ethernet cables remain stationary over time and if the setup is being moved. A small amount of slack was left to allow the shelves to be pulled out while the system is running in order to facilitate simple and non-interruptive maintenance.

The Ethernet cables were laid exclusively through the back of the enclosure and the top of the switch to provide an unencumbered view of the shelves and to allow for the shelves to be pulled out while in operation. The cables were, where possible, bound together using cable ties to ensure that the enclosure remains orderly.

6.3.2 Open Nodes

The ONs are clustered into stacks of the same board to ensure that all nodes in the stack conform to one form standard. Each Nucleo, except for the F767ZI, and Arduino tower encompasses 3 of the same ONs, each with one of the 3 selected network technologies.

The Ethernet Shield is the bottom shield to prevent unnecessary bending of the Ethernet cable; the LoRa Shield, on the other hand, blocks all holes which could be used to build the tower further and must therefore be placed at the very top of the tower. The ZigBee Shield is therefore placed in the middle of the tower. The Nucleo F767ZI's, which are already equipped with onboard Ethernet ports, do not make use of any shields.

6.3.3 Shields

The shields used by the ONs and gateways were selected due to their availability at the University of Munich and modified where needed to ensure compatibility with all nodes and modularity within the test bed.

Ethernet

The DFRduino Ethernet Shield V2.2 provides a standard RJ45 Ethernet jack for the Arduino Uno V3 pin layout and uses SPI to communicate with the microcontroller to which it is attached [DFR]. The shield is built with the W5100 Ethernet controller created by WIZnet and allows for speeds of up to 100Mbit/s. Due to its use of the In-Circuit Serial Programming (ICSP) header for communication with the microcontroller, the shield cannot be used out of the box with Nucleo boards, among others. In order for other boards to use the Ethernet shield, the following pins must be manually connected using an alternative method: MOSI, MISO, SLK, and Interrupt.

The pins using the ICSP header must be soldered as shown in Figure 6.8d and described in Table 6.1. The INT bridge, as shown in Figure 6.8d, must also be soldered shut to allow interrupt signals to be received by the node.

Pin	ICSP Pin	Arduino Pin
MOSI	4	11
MISO	1	12
SCK	3	13

Table 6.1: Pin Assignment DFRduino Ethernet Shield V2.2

Figure 6.7: ICSP Header (Orientation as in Figure 6.8c) [Ard19]

Zigbee / 802.15.4

The ATREB233-XPRO Zigbee Module is an 802.15.4 antenna created by Atmel created especially for use as a ZigBee transceiver. The ATREB233-XPRO was chosen due to the fact that multiple modules are already owned by the University of Munich. It does, unfortunately, not conform to the Arduino Uno V3 Pin Layout. In order to ensure consistent and modular devices / shields, Arduino prototyping shields were bought and soldered to create shields which conform to the standard. An angled box header is also required to connect the module to the shield, as seen in Figure 6.8f. The shield must be soldered as shown in Figure 6.8e and described in Table 6.2.

LoRa

The Semtech SX1272 LoRa Shield conforms to the Arduino Uno V3 pin layout and does not require any adjustments to function with Arduino or Nucleo embedded systems. The shield also permits the connection of an antenna to increase the range of the shield.

6.3.4 Linux Nodes

The HNs and gateways were stacked in order to save space on the shelves. Two Raspberry Pi to Arduino converter shields were bought so the gateways can use the same shields as the embedded systems. Two extensions headers were required so that both the PoE shield and the converter shield could be attached at the same time. This combination led to the gateway stack being the tallest stack in the setup, as shown in Figure 6.8b.

Pin	Arduino Layout Pin	Atmel ATREB233 Pin
VCC (5V)	5V	20
GND	GND	19
SPI_SCK	13	18
SPI_MISO	12	17
SPI_MOSI	11	16
SPI_SS_A	10	15
SPI_SS_B/GPIO	8	10
IRQ/GPIO	7	9
PWM(+)	9	7

Table 6.2: Pin Assignment for Atmel ATREB233-XPRO ZigBee Module

The normal Raspberry Pi 3B+'s are also outfitted with PoE shields which are equipped with fans to provide active cooling; sufficient space was left between the HNs to ensure for effective air flow, as shown in Figure 6.8a.

6.3.5 Hardware / Price Overview

The hardware used to build the test bed is listed in Table 6.3. Smaller items such as the tin and cables used to solder were omitted from the overview for the sake of clarity.

6.4 Current Configuration

This section provides an overview of the current state / configuration of the test bed. This overview is depicted in Figure 5.1, which also depicts the management network that connects gateways, HNs and the site server.

6.4.1 Network Topology

The switch is configured using two VLANs. The management VLAN encompasses ports 1-12 and 37-48. The VLAN used for the embedded systems outfitted with Ethernet modules, the node VLAN, can be accessed using ports 13-38. Port 1 is attached to the site server which provides the necessary services for the infrastructure of the test bed as well as access to the internet.

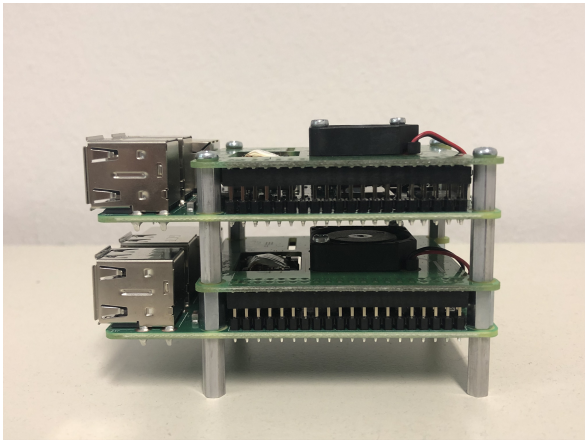
The management IPv4 network (192.168.44.0/24) is a class C network used by the HNs, switch, site server, and gateways. 192.168.44.1 - 192.168.44.100 is reserved for HNs, 192.168.44.201 - 192.168.44.255 is reserved for gateways and other management devices. 192.168.44.101-192.168.44.200 is currently used by the DHCP server to dynamically assign IP addresses to new devices. The site server and the gateways currently have only link-local IPv6 addresses.

All ONs which are equipped with Ethernet are connected to untagged ports (13-36) for the node VLAN, while the gateways are connected to both the management VLAN (untagged) and the node VLAN (tagged) using trunk ports (37-38), allowing them to route IP packets between their respective technologies, the node network, and the internet (over the site server).

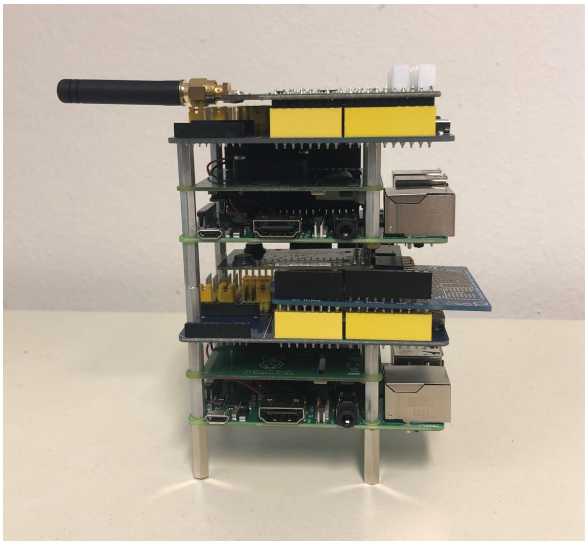
Component	Quantity	Price	Total
Embedded Systems			
Nucleo-F091RC	6	€9.37	€56.22
Nucleo-F411RE	6	€11.80	€70.80
Nucleo-F103RB	6	€9.37	€56.22
Nucleo-F767ZI	6	€20.89	€125.34
Arduino M0 Pro	6	€27.90	€167.40
Total Embedded Systems			€475.98
Host Nodes & Gateways			
Raspberry Pi 3 B+	11	€32.90	€361.90
Raspberry Pi PoE Shield	11	€23.20	€255.20
Arduino Adapter For Raspberry Pi	2	€26.20	€52.40
Raspberry Pi GPIO Stacking Header	2	€2.18	€4.36
Raspberry Pi PoE Stacking Header	2	€1.26	€2.52
Total Host Nodes & Gateways			€676.38
Network Modules / Shields			
Semtech SX1272 LoRa Shield	8	€22.14	€177.12
Atmel ATREB233-XPRO Zigbee Module	8	€27.69	€221.52
Arduino Prototyping Shield	8	€11.78	€94.24
DFRduino Ethernet Shield V2.2	8	€13.54	€108.32
Box Header	8	€0.14	€1.12
Total Network Modules / Shields			€602.32
Cabling			
Right-Angled USB Cable 10cm	6	€5.60	€33.60
Left-Angled USB Cable 10cm	12	€6.29	€75.48
USB Cable 15cm	12	€0.95	€11.40
Right-Angled Ethernet Cable 1m	7	€6.09	€42.63
Left-Angled Ethernet Cable 1m	2	€5.49	€10.98
Right-Angled Ethernet Cable 2m	6	€8.93	€53.58
Left-Angled Ethernet Cable 1.5m	2	€8.81	€17.62
Ethernet Cable 1m	8	€2.71	€21.68
Total Cabling			€266.97
Standoffs & Screws			
Standoff M2.5 x 20mm Female/Female	150	€0.314	€47.10
Standoff M2.5 x 14mm Male/Female	150	€0.429	€64.35
Standoff M2.5 x 15mm Male/Female	30	€0.454	€13.62
Standoff M2.5 x 10mm Male/Female	50	€0.408	€20.40
Screw M2.5 x 8mm	250	€0.087	€21.75
Total Standoffs / Screws			€167.22
Miscellaneous			
Netgear Gigabit Ethernet PoE+ Switch GS752TPv2	1	€630.00	€630.00
Digitus Network Rack DN-19 09U-6/6 incl. Shelves	1	€157.00	€157.00
Wooden Board 40cm x 32cm	2	€5.00	€10.00
Total Miscellaneous			€797.00
Total			€2,985.87

Table 6.3: Overview of All Purchased Hardware[6][7]

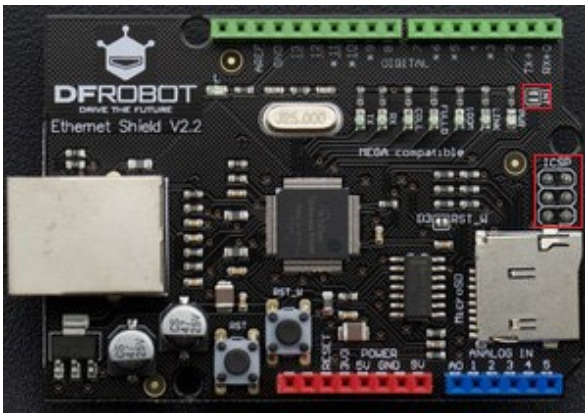
[6] All Prices Brutto (Including VAT)
[7] Prices found by various distributors, including Mouser, Reichelt and Amazon



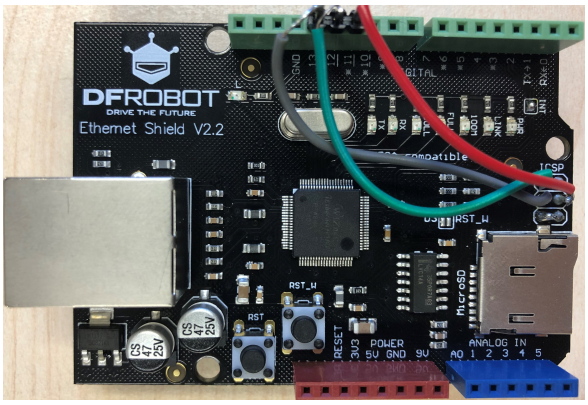
(a) HN Stack



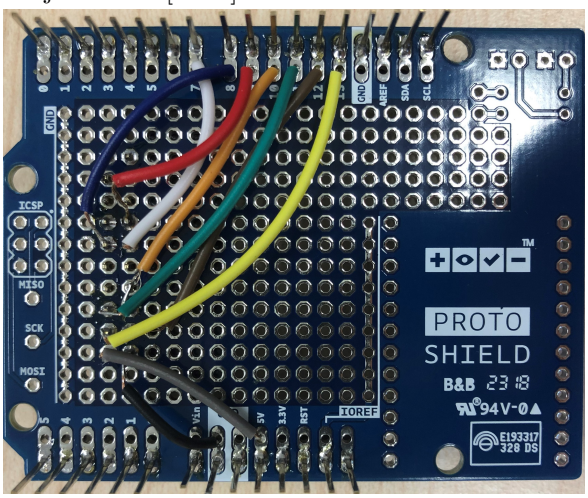
(b) Gateway Stack



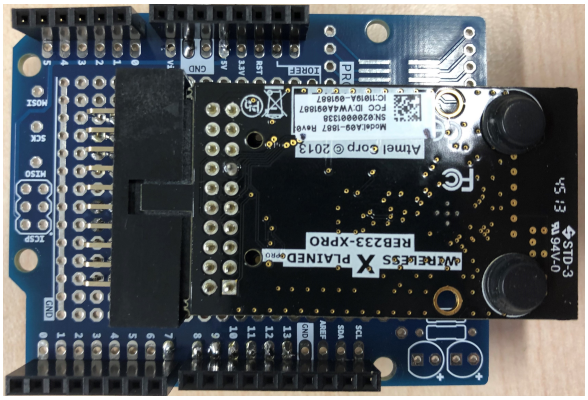
(c) Ethernet Shield with Position of Necessary Adjustments [DFR]



(d) Ethernet Shield After Adjustments



(e) Wiring of Arduino Prototype Shield



(f) Arduino Prototype Shield with Zigbee Transceiver

Figure 6.8: Modified Shields, HN Stack & Gateway Stack

Figure 6.9: Front View of Switch

6.4.2 Host Nodes & Open Nodes

Table 6.4 and Table 6.5 provide an overview of the current configuration of the test bed. This overview is especially useful when debugging the test bed / identifying defective nodes.

The switch can be reached from the management VLAN under 192.168.44.252 and the site server under 192.168.44.253.

6.5 Maintenance & Debugging

This section provides a non-exhaustive guide to performing maintenance and debugging with the test bed.

6.5.1 Host Node Update

Updating the software / operating system used by the HNs requires access to the site server, the update node, and the switch. In order to update the all HNs with minimum interruption to the operation of the test bed, the following steps are recommended:

1. Copy the current root directory (incl. permissions) on the site server (/nfs/client2/) to another directory and ensure that it is accessible per NFS.

```
#Execute on site server
cp -rp /nfs/client2 /nfs/client3
/nfs/client3 *(rw,sync,no_subtree_check,no_root_squash) >> /nfs/exports
service nfs-kernel-server restart
```

2. Login into the update node and mount the newly created directory.

```
mount -t nfs 192.168.44.253:/nfs/client3 /mnt/client3
```

3. Use chroot to execute all necessary update commands in that folder.

4. Unmount the new root directory.

```
unmount /mnt/client3
```

5. Shutdown all HNs using the switch PoE interface found under: System → PoE → PoE Configuration.

6. Delete the old root directory (make a backup, if needed) and rename the new one.

```
rm -rf /nfs/client2
mv /nfs/client3 /nfs/client2
```

7. Copy all boot files except cmdline.txt und config.txt into /tftpboot if they were changed (for example, when updating the kernel).

8. Start all HNs using the same interface mentioned in step 5.

HN ID / IP	Location HN	ON ID	ON Board	Network Technology
6 / 192.168.44.8	Left Tower Top Node	24	Nucleo F767ZI	Ethernet
		25	Nucleo F767ZI	Ethernet
		26	Nucleo F767ZI	Ethernet
7 / 192.168.44.9	Left Tower Bottom Node	22	Arduino M0 Pro	Ethernet
		16	Nucleo F103RB	LoRa
		17	Nucleo F103RB	Zigbee
		18	Nucleo F103RB	Ethernet
3 / 192.168.44.5	Right Tower Top Node	7	Nucleo F091RC	LoRa
		8	Nucleo F091RC	ZigBee
		9	Nucleo F091RC	Ethernet
4 / 192.168.44.6	Right Tower Bottom Node	23	Arduino M0 Pro	ZigBee
		1	Nucleo F411RE	LoRa
		2	Nucleo F411RE	ZigBee
		3	Nucleo F411RE	Ethernet

Table 6.4: Overview of Bottom Shelf

HN ID / IP	Location HN	ON ID	ON Board	Network Technology
1 / 192.168.44.3	Left Tower Top Node	19	Arduino M0 Pro	LoRa
		4	Nucleo F411RE	LoRa
		5	Nucleo F411RE	ZigBee
		6	Nucleo F411RE	Ethernet
2 / 192.168.44.4	Left Tower Bottom Node	21	Arduino M0 Pro	Ethernet
		27	Nucleo F767ZI	Ethernet
		28	Nucleo F767ZI	Ethernet
		29	Nucleo F767ZI	Ethernet
8 / 192.168.44.10	Right Tower Middle Node	13	Nucleo F103RB	LoRa
		14	Nucleo F103RB	ZigBee
		15	Nucleo F103RB	Ethernet
5 / 192.168.44.7	Right Tower Bottom Node	20	Arduino M0 Pro	ZigBee
		10	Nucleo F091RC	LoRa
		11	Nucleo F091RC	ZigBee
		12	Nucleo F091RC	Ethernet
192.168.44.202	ZigBee Gateway	-	-	-
192.168.44.201	LoRa Gateway	-	-	-
192.168.44.203	Update Node Right Tower Top Node	-	-	-

Table 6.5: Overview of Top Shelf

6.5.2 Identification & Debugging of Defective Nodes

The following section provides a short, non-exhaustive guide for debugging low-level components in the test bed.

HNs

If a host node cannot be reached, the first step should be to restart that node using the switch. The following debugging steps are recommended:

1. View all active PoE ports using the interface under System → PoE → Advanced → Port Overview and the MAC addresses of the connected ports using the interface under Switching → Address Table.

2. If a PoE port is active and does not show a MAC address or the node is simply non-responsive, then restarting the node often solves the problem. The nodes can be restarted by turning the corresponding PoE ports off and back on again using the interface under System → PoE → PoE Configuration.

3. Should the node continue to be unresponsive, then it should be removed from the test bed and examined for possible hardware defects.

4. If the node is responsive and reachable (ping and ssh) but unable to flash ONs, the unique log directory of that HN should be examined to obtain further information.

ONs

If an ON cannot be flashed or does not respond to serial commands and the HN to which it is attached is running, it is recommended that the ON is removed from the test bed and separately debugged. The location of the ON can be determined using the overview in 6.4.2

6.6 Fulfillment of Requirements

This section examines the test bed at the University of Munich with regards to the requirements described in the third chapter.

Scalability

The adjustments to the low-level architecture of the FIT IoT Lab removes the limiting constraint of requiring one HN for every ON. This allows for the test bed at the University of Munich to take advantage of the modularity and robustness of the software developed by the FIT consortium while limiting the amount of necessary hardware and thus enhancing its scalability.

Availability

The test bed, including the central platform, is currently available using the temporary site server. The maintenance process described in 6.5.1 allows for the HNs to be updated efficiently and with minimal downtime, thereby achieving 99% availability.

	FIT IoT Lab	University of Munich
Software		
Node Management	✓	✓
Flashing of Software	✓	✓
Serial Output	✓	✓
Debugging Interface	✓	✓
Resource Management	✓	✗
User Management	✓	✗
Open Source	Partly	✓
Central Platform	✓	✓
User Interface	Web REST API CLI	Web
Hardware		
Flashing of Software	✓	✓
On-board Debugger	✓	✓

Table 6.6: Fulfillment of Requirements by FIT IoT Lab and University of Munich Test Bed

Robustness & Reliability

The low-level structure of the FIT IoT Lab allows for embedded systems to be flashed and debugged regardless of their current state; the test bed at the University of Munich takes advantage of this existing robustness.

Diversity

The test bed at the University of Munich offers 5 distinct embedded systems, which are widely spread in terms of processing power and other resources, allowing users to test their software under a number of different constraints. The nodes are outfitted with 3 distinct network technologies that provide the opportunity to test software and its interoperability.

Thermal / Power

The enclosure for the test bed is well-ventilated and allows for active and passive cooling. All HNs and gateways were outfitted with fans to prevent overheating and, in case of overheating, shutdown automatically to prevent damage. ONs can be flashed with an idle firmware when not in use to reduce the amount of energy used by the test bed. The 380W switch provides more than sufficient power for all HNs as well as ONs to run at full capacity.

Routing of Network Technologies

Both the hardware and the software used in the test bed allow for the routing of IPv6 packets between different network technologies. The gateways act as routers for their respective technology and the site server for the nodes that use Ethernet. The configuration of these devices can be changed by administrators as needed.

7 Conclusion & Future Work

The original impetus for this thesis was the cooperation between the FIT IoT Lab, Orange, the creators of SensorLab2, and the University of Munich. The goal was to design and build a test bed for the purpose of testing software / firmware for embedded systems using a number of network technologies. The heterogeneity of the test bed should reflect the realities of the current IoT ecosystem / landscape. In this thesis, a number of requirements regarding the necessary hardware and software for the test bed were established. A number of a test beds were then evaluated and compared on the basis of these requirements.

Based on the evaluation of a number of platforms, the software created by the FIT consortium was chosen to provide a basis for the creation of the test bed at the University of Munich. I would like to thank the FIT consortium for providing the open source software as well as support throughout the course of the thesis. The open source software was then modified in order to optimize it with regards to the unique requirements and the scalability of the test bed. The necessary infrastructure components were selected and purchased, including Raspberry Pi 3B+'s to serve as HNs, a PoE switch, and a secure enclosure. A diverse selection of ONs and shields were also selected and purchased to allow for the testing of software under a different constraints and using various network technologies. I would like to thank Annette Kostelezky for her help with the purchase of the components as well as the set-up of the test bed.

The existing test bed functions with the help of the temporary site server which can be used to manage and flash the ONs. The temporary site server, unfortunately, does not fulfill all requirements as defined in this thesis, especially those regarding user and resource management (e.g. the scheduling of nodes). Although not essential to a functioning test bed, these features would enable a larger group of people to actively and simultaneously use it without interrupting each other's work as well as ensure that the configuration is not changed by unauthorized users. These limitations can be removed either by using the site server created by the FIT consortium or by expanding the functionality of the temporary site server.

As mentioned in the second chapter, this thesis serves as the foundation for further work involving the interoperability of different embedded systems and network technologies. There are a number of further modifications and additions that can facilitate the testing of software and its interoperability.

- 6LoWPAN with LoRa

  The test bed already provides a gateway with 6LoWPAN support with the 802.15.4 standard being used on layer 1 - 2. In order to allow LoRa devices to send and receive IPv6 packets, a 6LoWPAN implementation for LoRa must be developed for the Raspberry Pi 3B+ acting as a gateway as well as the firmware to be used by individual nodes.

- Integration into the FIT Infrastructure

Although the contract negotiations with the FIT consortium were not completed in a manner timely enough to be used in the course of this thesis, it would be advantageous to both parties to integrate the test bed at the University of Munich into the existing overarching FIT infrastructure. It is unclear to what extent the software for the FIT site server would have to be modified to conform with the changes made to the low-level infrastructure. In particular the assignment of multiple ONs to a single HN will most likely require adjustments to the site server with regards to node management, the flashing of nodes, and serial redirection. Alternatively it is possible to extend the functionality currently offered by the temporary site server to match that of the FIT site server although this option is significantly more time-consuming.

- Addition of Further Network Technologies

  The selection of ZigBee, LoRa, and Ethernet as the three initial technologies to be used in the test bed provides a diverse basis for testing the interoperability of software. The addition of further network technologies would allow for the test bed to better reflect more diverse real world scenarios. Bluetooth and Wi-Fi would not require the addition of any further gateways due to the on-board Bluetooth and Wi-Fi transceiver already available on the Raspberry 3B+. The addition of new network technologies would, however, require the purchase of shields for the ONs and the configuration of new gateways. The use of multiple wireless network technologies in the small enclosure used for the test bed can also lead to interference and unreliable connections if they use the same frequency / band.

- Dynamic IP Multicast Groups

  The ultimate goal of this thesis in conjunction with other (future) theses is the creation of a test which allows for embedded systems using a plethora of network technologies to be dynamically added and removed from IP Multicast groups. Ralf Weidner is currently developing this extension / functionality for the RIOT OS operating system and use within the test bed at the University of Munich.

- Security / Privacy Concerns

  This thesis does not consider the security required by an open test bed or the privacy concerns of IoT technology. Future works could examine security and privacy concerns and implement measures to monitor and ensure their observance.

Glossary

6LoWPAN IPv6 over Low-Power Wireless Personal Area Networks.

API Application Program Interface.

ARM Advanced Risc Machines.

BLE Bluetooth Low Energy.

CISC Complex Instruction Set Computing.

CLI Command Line Interface.

CN Control Node.

DHCP Dynamic Host Configuration Protocol.

ERD Entity Relationship Diagram.

FIESTA-IoT Federated Interoperable Semantic IoT Testbeds and Applications.

FIT Future Internet of Things.

GPIO General-Purpose Input/Output.

GW Gateway.

HN Host Node.

ICSP In-Circuit Serial Programming.

IEEE Institute of Electrical and Electronics Engineers.

IETF Internet Engineering Task Force.

Internet of Things A network of items - each embedded with sensors - which are connected to the Internet[CMR14].

IoT Internet of Things.

IP Internet Protocol.

IP Multicast A method of sending IP packets to a group of clients using a single transmission.

LPWAN low power wide area wireless network.

lwIP Lightweight IP.

MAC Media Access Control.

mesh topology A topology in which each device is interconnected with one another, allowing for most transmissions, to be distributed even if one of the connections goes down [SA13].

MTU Maximum Transmission Unit.

NFS Network File System.

ON Open Node.

OpenOCD Open On-Chip Debugger.

PoE Power over Ethernet.

point to point topology A direct connection between two devices (nodes), the value of a permanent point-to-point network is unimpeded communications between the two end-points [SA13].

RAM Random Access Memory.

REpresentational State Transfer an simple architectural style for developing web services based on the Hypertext Transfer Protocol (HTTP).

REST REpresentational State Transfer.

RISC Reduced Instruction Set Computer.

SPI Serial Peripheral Interface.

star topology One of the most common network topologies in which each of the devices or nodes in a network connects to a central hub [SA13].

symlink symbolic link.

TCP Transmission Control Protocol.

TFTP Trivial File Transfer Protocol.

TMPFS Temporary File System.

UART Universal Asynchronous Receiving and Transmitting.

VLAN Virtual Local Area Network.

VPN Virtual Private Network.

WSN Wireless Sensor Network.

List of Figures

Bibliography

[80216] IEEE Standard for Low-Rate Wireless Networks. In: *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)* (2016), April, S. 1–709. http://dx.doi.org/10.1109/IEEESTD.2016.7460875. – DOI 10.1109/IEEESTD.2016.7460875

[ABF+15] Adjih, Cedric ; Baccelli, Emmanuel ; Fleury, Eric ; Harter, Gaetan ; Mitton, Nathalie ; Noel, Thomas ; Pissard-Gibollet, Roger ; Saint-Marcel, Frederic ; Schreiner, Guillaume ; Vandaele, Julien ; Watteyne, Thomas: FIT IoT-LAB: A large scale open experimental IoT testbed, 2015

[Arda] Arduino: *About Us: Arduino*. https://www.arduino.cc/en/Main/AboutUs

[Ardb] Arduino: *Arduino Uno Rev V3*. https://store-cdn.arduino.cc/uni/catalog/product/cache/1/image/1040x660/604a3538c15e081937dbfbd20aa60aad/a/0/a000066_featured_1_.jpg. – [Online; accessed August 21, 2019]

[Ard19] Arduino: *ICSP Header*. 2019 https://www.arduino.cc/en/uploads/Reference/ICSPHeader.jpg

[ASS] ASSOCIATION, SILICON VALLEY H.: *Atmel Corporation*. https://www.siliconvalleyhistorical.org/atmel-history/

[BEKG19] Bormann, Carsten ; Ersue, Mehmet ; Keranen, Ari ; Gomez, Carles: Terminology for Constrained-Node Networks / IETF Secretariat. Version: March 2019. http://www.ietf.org/internet-drafts/draft-bormann-lwig-7228bis-04.txt. 2019 (draft-bormann-lwig-7228bis-04). – Internet-Draft. – http://www.ietf.org/internet-drafts/draft-bormann-lwig-7228bis-04.txt

[BGH+18] Baccelli, Emmanuel ; Gündogan, Cenk ; Hahm, Oliver ; Kietzmann, Peter ; S. Lenders, Martine ; Petersen, Hauke ; Schleiser, Kaspar ; Schmidt, Thomas ; Wahlisch, Matthias: RIOT: an Open Source Operating System for Low-end Embedded Devices in the IoT. In: *IEEE Internet of Things Journal* PP (2018), 03, S. 1–1. http://dx.doi.org/10.1109/JIOT.2018.2815038. – DOI 10.1109/JIOT.2018.2815038

[Bha91] Bhandarkar, Dileep: Performance From Architecture: RISC vs CISC, 1991

[CMR14] Chebudie, Abiy B. ; Minerva, Roberto ; Rotondi, Domenico: *Towards a definition of the Internet of Things (IoT)*, Diss., 08 2014

[Com] Comission, European: *Federated Interoperable Semantic IoT/cloud Testbeds and Applications.* https://cordis.europa.eu/project/rcn/194117/factsheet/en

[con] consortium, FIT: *About us: The FIT consortium.* https://www.iot-lab.info/about-us/

[Cos16] Costachioiu, Teodor: *Power guzzlers: testing some Arduino boards.* https://electronza.com/power-guzzlers-testing-arduino-boards/. Version: 2016

[DFR] DFRobot: *DFRduino Ethernet Shield.* https://www.dfrobot.com/product-455.html

[Dob18] Dobrilovic, D: Networking Technologies for Smart Cities: An Overview. In: *Interdisciplinary Description of Complex Systems* 16 (2018), 09, S. 408–416. http://dx.doi.org/10.7906/indecs.16.3.13. – DOI 10.7906/indecs.16.3.13

[Dra] Dramble, Raspberry P.: *Power Consumption Benchmarks Raspberry Pi.* https://www.pidramble.com/wiki/benchmarks/power-consumption

[Dun01] Dunkels, Adam: Design and implementation of the lwIP TCP/IP stack. In: *Swedish Institute of Computer Science* 2 (2001), 03

[EF] Eclipse Foundation, Inc.: *Open Source for IoT.* https://www.eclipse.org

[EH18] El Hakim, Ahmed: *Internet of Things (IoT) System Architecture and Technologies, White Paper.*

[Eic17] Eich, Florian: *Tools for Managing and Testing IP Multicast Networks.* 2017

[Eva11] Evans, D: The Internet of Things: How the Next Evolution of the Internet is Changing Everything. In: *Cisco Internet Business Solutions Group (IBSG)* 1 (2011), 01, S. 1–11

[FNS15] Firdaus, Firdaus ; Nugroho, Eko ; Sahroni, Alvin: ZigBee and wifi network interface on Wireless Sensor Networks. In: *Proceeding - 2014 Makassar International Conference on Electrical Engineering and Informatics, MICEEI 2014* (2015), 03, S. 54–58. http://dx.doi.org/10.1109/MICEEI.2014.7067310. – DOI 10.1109/MICEEI.2014.7067310

[FOUa] FOUNDATION, RASPBERRY P.: *About Us Raspberry Pi Foundation.* https://www.raspberrypi.org/about/

[FOUb] FOUNDATION, RASPBERRY P.: *Network boot your Raspberry Pi.* https://www.raspberrypi.org/documentation/hardware/raspberrypi/bootmodes/net_tutorial.md

[Fre] FreeRTOS: *About FreeRTOS.* https://www.freertos.org/RTOS.html

[Hea16] Heath, Nick ; ZDNet (Hrsg.): *Your Raspberry Pi 3 won't overheat in everyday use.* https://www.zdnet.com/article/no-your-raspberry-pi-3-wont-overheat-in-everyday-use-says-its-creator/. Version: 2016

[IoT] IoT, Fiesta: *FIESTA-IoT provides a Blueprint Experimental Infrastructure for Heterogeneous IoT Technologies*. `http://fiesta-iot.eu/index.php/iot-experiments-as-a-service/`

[KAT05] Koubaa, Anis ; Alves, MÃ¡rio ; Tovar, Eduardo: IEEE 802.15.4 for Wireless Sensor Networks: A Technical Overview. (2005), 01

[KAT19] Koubaa, Anis ; Alves, Mario ; Tovar, Eduardo: IEEE 802.15.4: a Federating Communication Protocol for Time-Sensitive Wireless Sensor Networks. (2019), 08

[LA18] LoRa Alliance, Inc.: *LoRaWAN 1.1 Regional Parameters*, 2018

[Lam17] Lambert, Tyler: *Introduction to Microcontrollers and Embedded Systems*. `https://www.researchgate.net/publication/317399328_Introduction_to_Microcontrollers_and_Embedded_Systems`. Version: 07 2017

[LB17] Lampin, Quentin ; Barthel, Dominique: Sensorlab2: A monitoring framework for IoT networks. In: *2017 International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN)* (2017), S. 1–6

[Lev12] Levis, Philip: Experiences from a decade of TinyOS development, 2012, S. 207–220

[LSGF+16] Lanza, Jorge ; Sanchez, Luis ; Gomez Fernandez, David ; Elsaleh, Tarek ; Steinke, Ronald ; Cirillo, Flavio: A Proof-of-Concept for Semantically Interoperable Federation of IoT Experimentation Facilities. In: *Sensors* 16 (2016), 06, S. 1006. `http://dx.doi.org/10.3390/s16071006`. – DOI 10.3390/s16071006

[MM16] Mahmoud, Mahmoud ; Mohamad, Auday: A Study of Efficient Power Consumption Wireless Communication Techniques/ Modules for Internet of Things (IoT) Applications. In: *Advances in Internet of Things* 06 (2016), 01, S. 19–29. `http://dx.doi.org/10.4236/ait.2016.62002`. – DOI 10.4236/ait.2016.62002

[MMAM17] Marais, Jaco ; Malekian, Reza ; Abu-Mahfouz, Adnan: LoRa and LoRaWAN testbeds: A review, 2017

[Mou] Mouser Electronics, Inc.: *Nucleo Board 64*. `https://www.mouser.de/images/stmicroelectronics/lrg/Nucleo_Board_64_DSL.jpg`. – [Online; accessed August 21, 2019]

[Mul07] Mulligan, Geoff: The 6LoWPAN Architecture. In: *Proceedings of the 4th Workshop on Embedded Networked Sensors*. New York, NY, USA : ACM, 2007 (EmNets '07). – ISBN 978–1–59593–694–3, 78–82

[NBC17] Noreen, U. ; Bounceur, A. ; Clavier, L.: A study of LoRa low power and wide area network technology. In: *2017 International Conference on Advanced Technologies for Signal and Image Processing (ATSIP)*, 2017, S. 1–6

[Pam16] Pamar, Jekishan K.: IoT: Networking Technologies and Research Challenges. In: *International Journal of Computer Applications* (2016)

[PD16] Parmar, Jekishan ; Desai, Ankit: IoT: Networking Technologies and Research Challenges. In: *International Journal of Computer Applications* 154 (2016), 11, S. 1–6. http://dx.doi.org/10.5120/ijca2016912181. – DOI 10.5120/ijca2016912181

[Ras] Raspberry Pi Foundation: *Rasbperry Pi 3B+*. https://www.raspberrypi.org/homepage-9df4b/static/a77ea87d78ae0a97e7f3491c8ee14ca4/bc3a8/c680f7e0fae35f6cd7fd83123a66e4a3c3af1f1e_770a4970-1-1.jpgg. – [Online; accessed August 21, 2019]

[Rid18] Ridgway, Paul ; blockdev.io (Hrsg.): *Raspberry Pi, overlayfs read-write root, read-only NFS base*. https://blockdev.io/read-only-rpi/. Version: 2018

[SA13] Santra, Santanu ; Acharjya, Pinaki: A Study And Analysis on Computer Network Topology For Data Communication. In: *International Journal of Emerging Technology and Advanced Engineering* (2013)

[STM] STMicroelectronics: *Who We Are STMicroelectronics*. https://www.st.com/content/st_com/en/about/st_company_information/who-we-are.html

[TI16] Texas Instruments, Inc.: *LE Data Length Extension (DLE)*. http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_40_00_45/docs/blestack/ble_user_guide/html/ble-stack-3.x/data-length-extensions.html. Version: 2016

[Ull12] Ullah, Zobair: Use of Ethernet Technology in Computer Network. In: *Global Journal of Computer Science and Technology Network, Web & Security* (2012)

[Wie03] Wiegers, Karl E.: *Software Requirements*. 2. Redmond, WA, USA : Microsoft Press, 2003. – ISBN 0735618798, 9780735618794

[YHM+17] Yaqoob, Ibrar ; Hashem, Ibrahim ; Mehmood, Yasir ; Gani, Abdullah ; Mokhtar, Salimah ; Guizani, Sghaier: Enabling Communication Technologies for Smart Cities. In: *IEEE Communications Magazine* 55 (2017), 01. http://dx.doi.org/10.1109/MCOM.2017.1600232CM. – DOI 10.1109/MCOM.2017.1600232CM

[Zla16] Zlatanov, Nikola: *ARM Architecture and RISC Applications*. 02 2016