POLITECNICO DI TORINO

Master degree course in Computer engineer

Master Degree Thesis

Firmware update for 6LoWPAN networks of OMA-LwM2M IoT devices

Supervisors

Prof. Andrea Acquaviva

Ing. Marco Grella (STMicroelectronics)

Candidates

Davide Mirisola

Academic year 2017-2018

To my parents and grandparents

Acknowledgements

Before I start with the necessary acknowledgments, I would first and foremost like to dedicate my whole university and academic career along with my Thesis to my grandparents, who unfortunately are no longer with us. I know my grandfather, Giuseppe would have been immensely proud to have an engineer within the family.

Next I would like to express my thanks to Professor Andrea Acquaviva of the Politecnico di Torino who as my supervisor inspired and instructed me throughout my Thesis work and without whom, this achievement would not have been possible.

I am also deeply grateful to both my Manager Ing. Fabian Castanier and my mentor Marco Grella of STMicroelectronics. I thank them for having faith in me and allowing me the opportunity to further develop experience within my chosen field. The company provided an enriching and stimulating work environment. It was an absolute honour to work with the company.

A special thanks goes to Marco Grella who despite his many work commitments, always found the time to provide me with guidance and support in helping me develop my Thesis and allowing me to learn important lessons from him going forward with my professional career.

Finally, I am deeply indebted to both my parents who make me proud to be their son. I want to thank them for always believing in me and continually providing support and encouragement in my journey through life and my many years of study.

It has truly been an honour to have known you all in various capacity and again thank you all from the bottom of my heart.

Abstract

In an IoT architecture we can often find a number of devices that can easily exceed the hundreds, usually located in remote or inaccessible areas where manual intervention is generally cost prohibitive and sometimes really difficult. Since these devices have long lifetime requirements, it is very likely that they need to be upgraded or reprogrammed. In this context, providing techniques to remotely update their firmware is of critical importance to achieve the desired system functionality and performance. The solution to this particular problem is called Firmware Over-The-Air (FOTA) and it consists in supporting an efficient and timely firmware update for IoT devices. A new research area is then now evolving to design innovative FOTA solutions.

In this thesis, a new FOTA method is evaluated with the purpose of guaranteeing the firmware update process in heterogeneous Low Power IoT networks, exploring the implementation details and quantifying the costs in terms of energy, throughput and time required to complete the operation.

This method uses the OMA-LwM2M Device Management and Data Model standard along with the CoRE CoAP application protocol over the 6LoWPAN communication protocol to enabling M2M communication. This kind of implementation gives to a node the possibility of updating its firmware remotely, either receiving (Push mode) or fetching (Pull mode) it from a server. In addition to that, a recovery mode in case of failures has been added in order to restore and boot the old and stable version if necessary. In the end, the DTLS protocol has been enabled with the purpose of securing the data transfer in order to quantify its impact on the firmware update process.

Contents

List of Figures

Chapter 1

Introduction

Several years have been passed since we started to use Internet for communications and it has been more than just a way to communicate, but it represented a starting point for a new way of thinking and moreover of living.

Nowadays, in a world where people and devices are strongly linked together, thanks to technological advances in the IT world and how connections have become a first need, we have already entered in a new era where not only people, but also objects are connected, where everything is smart and magic is becoming something real, this era is called "Internet of Things".

Internet of Things (IoT) is the result of two concepts "Internet" and "thing". The former represents the connection among "things", while the latter indicates every entity able to interact both with themselves and with the environment where they are located, anytime and anywhere. In an IoT ecosystem, it is important to update or reprogram all the remotely connected devices, in order to upgrade their functionalities or add new features. Moreover, due to the fact that the firmware might be transferred over a lossy low bit-rate network and that limited hardware resources devices are used, a FOTA technology is necessary in this context and it has to be developed in order to guarantee the firmware update.[12]

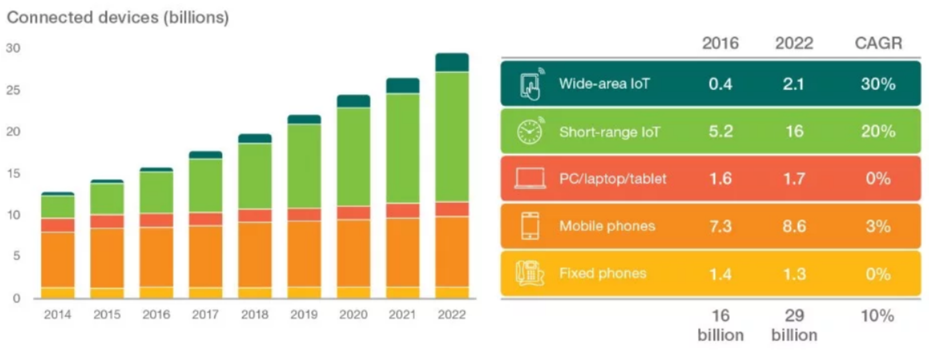
1.1 IoT trend

In the last five years the computers tendency has changed silently from mainframe computers and static machines to always more portable and tiny devices. This tendency is more and more fast evolving, so now we can easy find devices that are implantable, wearable, mobile and portable.

From now we are going to live surrounded by systems with the aim of support and sustain us in our homes, offices, cars, gardens, stations, gyms, and other daily actions and moments. They will understand and elaborate our primary needs in order to obtain the rightful services and give them to the right person in the exact moment in which they are requested.

All the environments where we live are already filled up with a variety of electronic elements including environment monitoring sensors, actuators, monitors, cameras and so on, but in the next future, all the physical and concrete items will become empowered with computation and communication-enabled components by embedding specially made electronics onto them in order to make these items more and more useful and usable.

In this panorama the value of the IoT market is expected to hit $1423.09 billion by 2020 [3] and according to the "Ericsson" forecast [7] around 18 billion of IoT devices will be connected by 2022 as it is shown in the Figure 1.1.



	2016	2022	CAGR
Wide-area IoT	0.4	2.1	30%
Short-range IoT	5.2	16	20%
PC/laptop/tablet	1.6	1.7	0%
Mobile phones	7.3	8.6	3%
Fixed phones	1.4	1.3	0%
	16 billion	29 billion	10%

Figure 1.1. IoT trend in terms of number of devices.

1.2 IoT domains

IoT environment is constantly growing both in number of devices and applications, so it is very easy now to find it in several and different domains.[3][16]

Hereafter the main sectors wherein the IoT is bound are listed:

Automotive: vehicles are heavily equipped with sensors, actuators and infotainment systems in order to help drivers to have a safe, comfortable and smart experience.

Energy: smart metering solutions are possible through the IoT in such a way that consumers, and utility providers get the information they need to better manage services, infrastructures and power consumptions minimizing energy costs and the environmental impact.

Health care: thanks to a big number of specific and wearable devices for measuring and managing different health parameters of humans, IoT is changing remote patient monitoring, allowing personalized treatment, reduced health care costs and real time assistance to patients.

Industrial: industrial machines are made smart so they can speed up the production, lower the process costs and analyze data from every link of the manufacturing chain with the aim of increase efficiency and keep production running smoothly.

Retail: IoT aims to gather and organizing data, improve logistics, supply chain management and inventory management in such way that retailers can provide personalized and immersive experiences that make shoppers enthusiastic of the service. “Amazon” is a good example of what IoT, in retail sector, can do.

Smart building: a smart building is the fusion of building, technology, and energy systems. In a smart building, IoT aims to provide building automations, life safety, telecommunications, user systems, facility management systems and security. For example, because of IoT, is possible to provide remote alarm control, remote heating, ventilation, and air conditioning (HVAC) controls for homes and businesses through mobile phones.

Smart city: a smart city is an urban area that uses IoT devices to gather information that will be processed in order to monitor and manage traffic and transportation systems, water supply networks, law enforcement and other services enhancing both the quality of government services and citizen welfare.

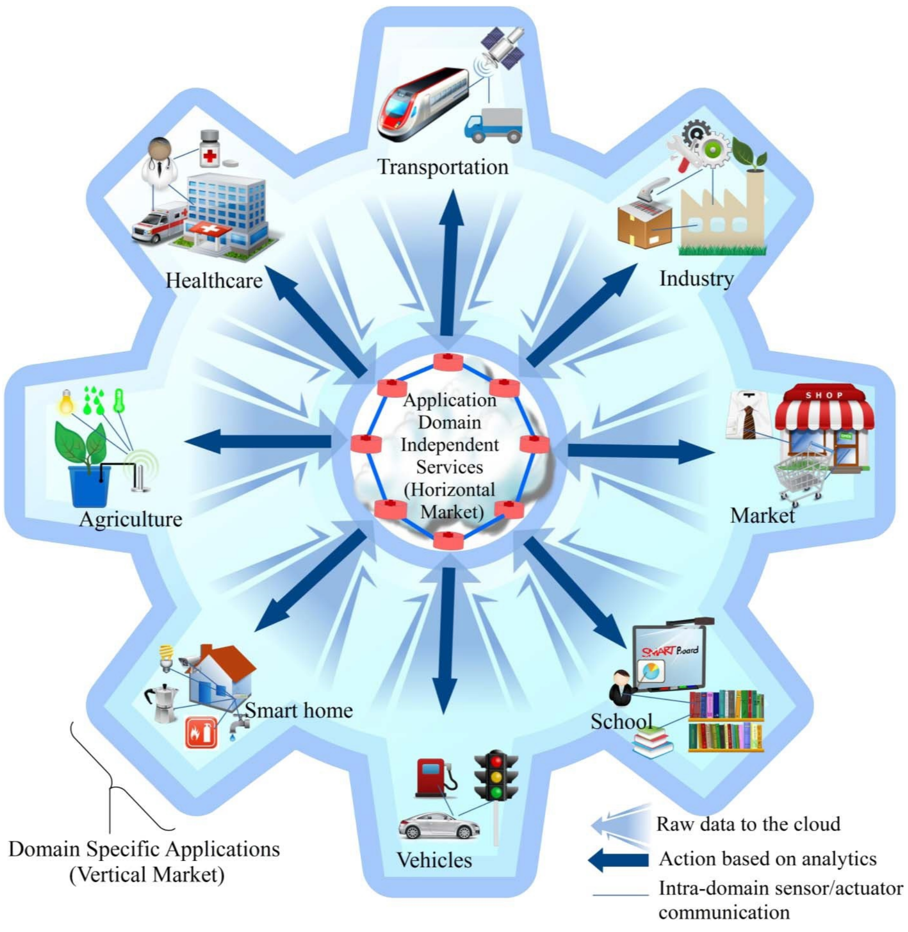

Transportation
Industry
Market
School
Vehicles
Smart home
Agriculture
Healthcare
Application Domain Independent Services (Horizontal Market)
SHOP
SMART Board
Domain Specific Applications (Vertical Market)
Raw data to the cloud
Action based on analytics
Intra-domain sensor/actuator communication

Figure 1.2. IoT domains representation.

1.3 Firmware Over-The-Air (FOTA)

As has been said before, "Things" are elements able to communicate among them and with the surrounding environment, but to be more specific, a "Thing" consists in an embedded system composed of sensors, actuators and communication interfaces. Embedded systems need a software in order to operate in a specific and determined manner, this software is called "Firmware". This firmware sometime needs to be upgraded or updated for different reasons, for instance there could be some bugs in, new features need to be installed on the device or some parameters need to be adjusted and so on.

Often, in an IoT architecture, a large number of devices is present and all of that can be located in remote or inaccessible areas where manual intervention is cost prohibitive or otherwise difficult, so since these devices have long lifetime requirements, firmware update plays a critical role.

These considerations are valid in particular for battery powered devices that are in fact used in ultra-Low Power scenarios like Low Power IoT sensors networks.

The solution for this problem is called Firmware Over-The-Air (FOTA) that consists in different methods able to provide efficient and timely firmware updates or upgrade for a device remotely allowing to:

- Continuously reduce the number of bugs and improve product behavior even after the device is in the hands of your consumers;
- Test new features by sending updates to one or multiple devices;
- Save costs by managing the firmware across their fleet of devices from a seamless, unified interface;
- Augments scalability by adding new features and infrastructure to products after they are released.

Because all these reasons, FOTA represents one of the most critical component for IoT success and to be successful it must meet various requirements related to the application for which it is built.

For instance, thinking of an IoT architectures designed for an inaccessible area, power consumption become a significant constraint to be faced. Regarding the geographic area covered by devices, distance between nodes is another factor that requires a scrupulous selection of the specific network technologies to adopt in order to guarantee connectivity within devices in constrained environment. Other two important requirements are the QoS that is a crucial in several domains such as in the health care domain and the security, that is even more important because it aims to secure data in each layer of the IoT protocol stack.

In this extent, all the elements designed for the IoT architectures have limitations in terms of power consumption, size, costs and provided features, for example IoT devices have small flash memory and reduced clock frequency, communication protocols have reduced headers, security protocols are light to avoid weighing down communications and so on. All these reasons make the development of FOTA methods a difficult job. Moving a big amount of data such as a firmware image in a constraint environment meeting all the requirements mentioned before is always difficult and onerous, thus building a FOTA architecture is as challenging as important.

For this reason, there is a new research area that is evolving trying to build optimal solutions for this particular scenario, born from the need of creating new firmware update methods for the IoT new paradigm where not a large number of solutions have been proposed yet.

Nowadays, there is a wide and diversified collection of hardware and software that can be integrated in order to create an ad hoc firmware update solution for each application. These elements include network technologies, low power devices, energy harvesting mechanisms, cloud computing systems and so on, that even if in different way, ensure benefits and compliance with the main applications requirements.

Furthermore, considering that IoT devices are different from each other and their number is growing, it is useful to exploit the benefits of the IoT in which the devices are considered as network objects, in order to facilitate the device management and ensure an efficient and reliable remote firmware update.

1.4 Proposal

In this thesis a new FOTA method is evaluated with the purpose of guaranteeing the firmware update process in heterogeneous Low Power IoT networks, exploring the implementation details and quantifying the costs in terms of energy, throughput and time required to complete the operation.

This work has been developed for STMicroelectronics company at the office located in the "Istituto Superiore Mario Boella" (ISBM) in Turin. The idea for this project comes from the willing to offer a FOTA method able to meet the low power requirement maintaining the compliance with the data model standards already used in IoT applications.

The FOTA method refers to the ST hardware and software systems, but it can be adopted by every IoT infrastructure after an appropriate porting process.

It implements a specific protocol stack that is illustrated in the Figure 1.3 where the OMA Lightweight M2M standard is located as device management over the CoAP internet application protocol and the 6LoWPAN protocol occupies the network layer to enable M2M communication exploiting IPv6 internet protocol.

This implementation gives a node the possibility to update its firmware remotely both fetching a firmware image from a file server (Pull mode) or receiving it directly from a server (Push mode), additionally it provides a solid management of the entire process and ensures a secure transfer based on DTLS protocol.

Moreover, the FOTA method offers a recovery mechanism in case of failures that could happen during the downloading and updating phases. This mechanism, thanks to the watchdog peripheral, is able to run the new firmware or to restore

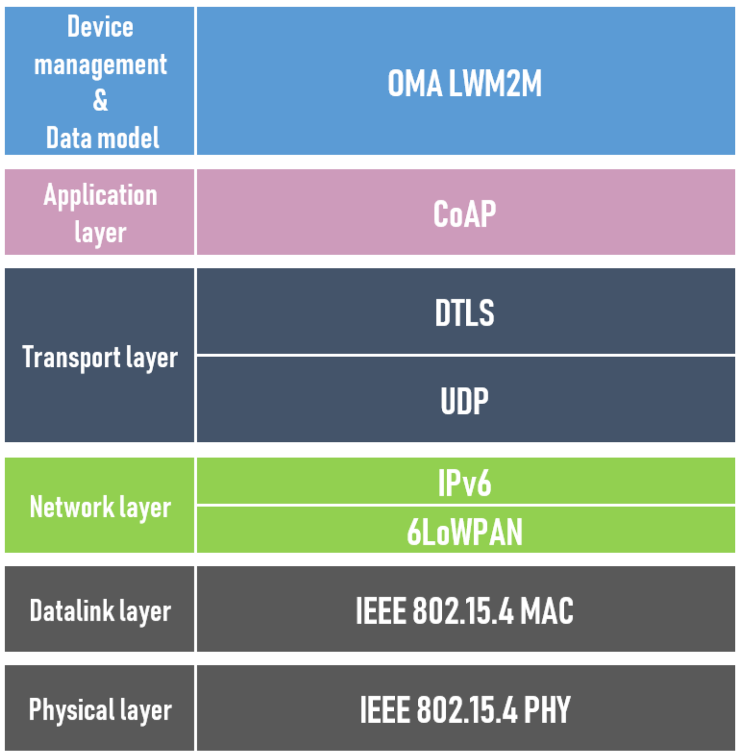


Figure 1.3. Proposed FOTA method protocol stack.

and boot the old and stable one according to the integrity of the new firmware image received from the server.

The proposed FOTA method architecture consists in many components that can be summarized in the Figure 1.4

This FOTA method has been developed for two different devices that are the STM32F401RE and STM32L476RG boards.

The starting point of the work has been the FP-SNS-6LPNODE1 function pack software that provides a basic OMA-LWM2M client solution running on STM32F401RE board. This basic implementation has been enhanced by adding the OMA-LwM2M Firmware Object stub and the internal logic. This first step led to a complete implementation of the FOTA process when the process is successful, leaving open the problem of system recovery in case of firmware failure.

In order to accomplish this, the software package has been ported to run on the STM32L476RG that provides more flexibility due to a bigger flash memory, that is also more fine-grained partitioned. Having a flash that is divided into smaller sectors results in more degrees of freedom in the firmware storage design.

Thanks to STM32L476RG features, the dual boot has been implemented along

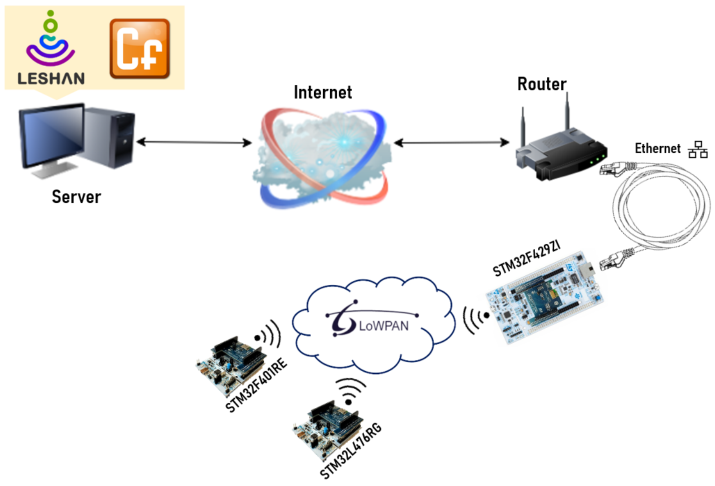


Figure 1.4. Proposed FOTA method architecture.

with a reliable recovery mechanism that starts in case of downloading or updating failures.

Many difficulties associated to the STM32F401RE flash memory, the STM32L476RG porting, the data format used by the standards and the characteristics of the servers adopted have been encountered and solved during the development process.

Since the FOTA method will be adopted by future developers in order to build dependable IoT applications or improve the procedure itself, taking some measurements in terms of power consumption, transfer time and throughput has been necessary in order to characterize the firmware update process. According to this, many tests have been performed with different context and distinct scenarios such as the one hop and two hops network, in which the different performances between the two boards, the behaviour of the forwarding node with respect to the final one, the characteristics of the Push and Pull transfer modes and the overhead that DTLS secure algorithm has on the firmware update process have been highlighted.

The end of the development process has led to a reliable, robust and portable new low power FOTA method that take the lead for new developments and future works.

A more detailed description of the FOTA method implementation, in addition to a background about IoT protocols and FOTA architectures will be discussed in the next chapters.

1.5 Thesis structure

This thesis is structured as follow:

- **Chapter 2**: gives a detailed description of an IoT layered architecture where each level's function along with every element of which it is composed is described. Then the STMicroelectronics company, its products' portfolio and IoT solutions are presented;
- **Chapter 3**: describes a typical FOTA architecture starting from its requirements. The main elements of the architecture, an explanation of the main firmware update techniques and an example of a FOTA protocol are analysed and illustrated. Moreover, some examples of FOTA solutions are presented;
- **Chapter 4**: presents the three main standards adopted in the thesis FOTA system architecture. In order, the structure of the 6LoWPAN protocol is described including its main features; then CoAP application layer protocol is discussed highlighting its message format and communication methods. To conclude the OMA-LwM2M device management protocol is presented including its data model and structure;
- **Chapter 5**: gives a detailed description of both hardware and software used to design, build, program and test the thesis FOTA system architecture;
- **Chapter 6**: gives a step by step description of the FOTA method development. Here all the design choises along with the algorithm definition and architecture description are explained in detail;
- **Chapter 7**: illustrates the tests that have been performed, describing the structural conditions and system parameters. To conclude, tests results are presented and discussed;
- **Chapter 8**: concludes the thesis work giving an overview of the results achieved and gives an overview of possible future work;

Chapter 2

IoT architecture

To meet the growing IoT trend and guarantee the capability to connect billions of different and heterogeneous devices, the need to have a standardized infrastructure arises.

In addition to this, IoT devices are typically interacting 24x7 thus a highly-available (HA) approach is needed to allow disaster recovery (DR) across data centres and again devices may not have UIs so a remote management is required in order to interact with them.

Moreover, IoT devices are very commonly used for collecting and analyzing personal data, so even in this case, a model for managing the identity and access control for IoT devices is a key requirement. For all this reasons and more, it is essential to have a flexible and scalable layered architecture in place in order to support integration between systems and devices.

IoT architectures must comply some specific requirements according to their components and the environments that they support, including:

- Scalability;
- Security;
- Privacy;
- Highly-availability (HA);
- Integration;
- Transparency;
- Energy efficiency;
- Data reliability.

2.1 Layered architecture for IoT

There is no standard architecture for the IoT ecosystem and it can vary according to requirements and applications. For this reason, to be exhaustive a complete and robust IoT layered architecture is presented and discussed.[3]

The reference architecture in Figure 2.1 consists in a set of five layers plus three cross-cutting/vertical additional layers that due to their enforcement in different level, are presented independently in this architecture.

The layers are:

1. Device layer;
2. Communication layer;
3. Integration layer
   - Aggregation layer;
   - Resource management.
4. Data management and analytics;
5. Apps and software;

The cross-cutting layers are:

- Security;
- Device management;
- Identity and access management.

All these layers are described in more detail throughout this chapter.

2.1.1 Device layer

Starting from the bottom, the first layer of the architecture is the devices layer. It comprises the physical devices that are used to collect and process information from the environment.

IoT ecosystem include different types of physical devices, but to be considered as such, they must be composed of:

MCU: the brain of the device that aim to take the input from sensors, elaborate it and produce output;

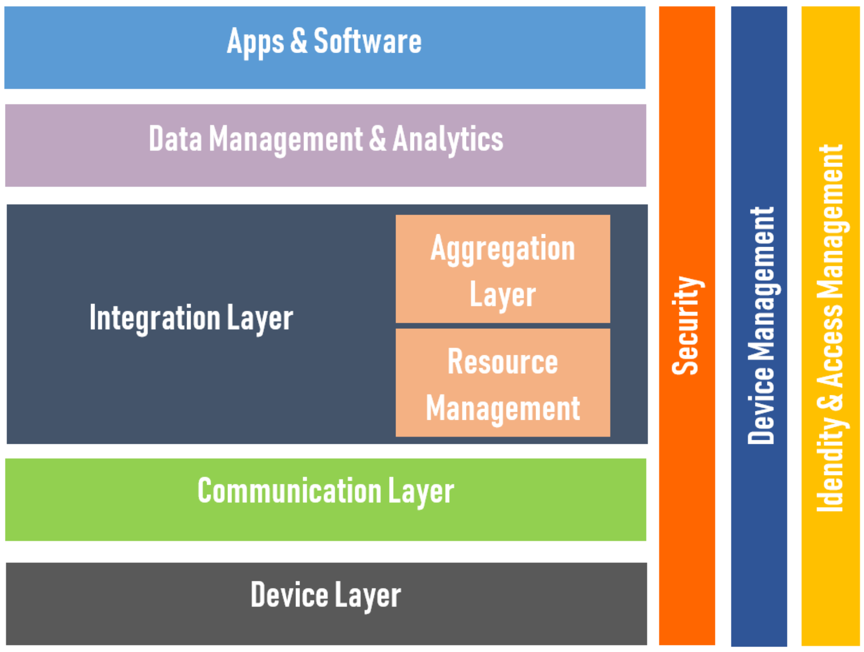


Figure 2.1. IoT layered architecture.

Sensors: physical elements that convert a non-electrical input into an electrical signal that can be sent to an electronic circuit;

Actuator: physical elements that convert an electrical signal into action, often by converting the signal to nonelectrical energy, such as motion;

Internet communication interface: hardware and software elements that ensure internet access to the device in order to send or receive data from upper layer.

Types of sensors

Sensors, based on their power sources can be categorized in:

Active sensors: emit energy of their own and then sense the response of the environment to that energy. They can be used in a wide range of environmental conditions;

Passive sensors: simply receive energy that is produced external to the sensing device. They require less energy to operate.

There is a wide selection of sensors with different functionality such as light sensors, gesture and proximity sensors, humidity sensors, touch and fingerprint sensors, temperature sensors, pressure sensors, velocity and acceleration sensors and more.

A special type of sensor-actuator devices are Microelectromechanical systems (MEMS) that represent the technology of microscopic devices, particularly those with moving parts.

Besides the functionality of the sensors, there are several generic factors that determine their suitability for a specific application and any of these can impact the reliability of the received data and therefore the value of the data itself.

Here the main factors are presented:

Accuracy: a measure of how precisely a sensor reports the signal;

Repeatability: a sensor's performance in consistently reporting the same response when subjected to the same input under constant environmental conditions;

Resolution: the smallest incremental change in the input signal that the sensor requires to sense and report a change in the output signal;

Noise: the fluctuations in the output signal resulting from the sensor or the external environment;

Range: the band of input signals within which a sensor can perform accurately;

Selectivity: the sensor's ability to selectively sense and report a signal.

2.1.2 Communication layer

This layer transfers data that are collected from devices layer to integration layer using secure transmission channels.[17]

In order to support the connectivity of the devices, the communication layer implements a specific IoT protocol stack based on the TCP/IP stack using multiple potential protocols designed for constraint systems.

The Figure 2.2 shows the layered stack highlighting the most common used protocols for each layer.

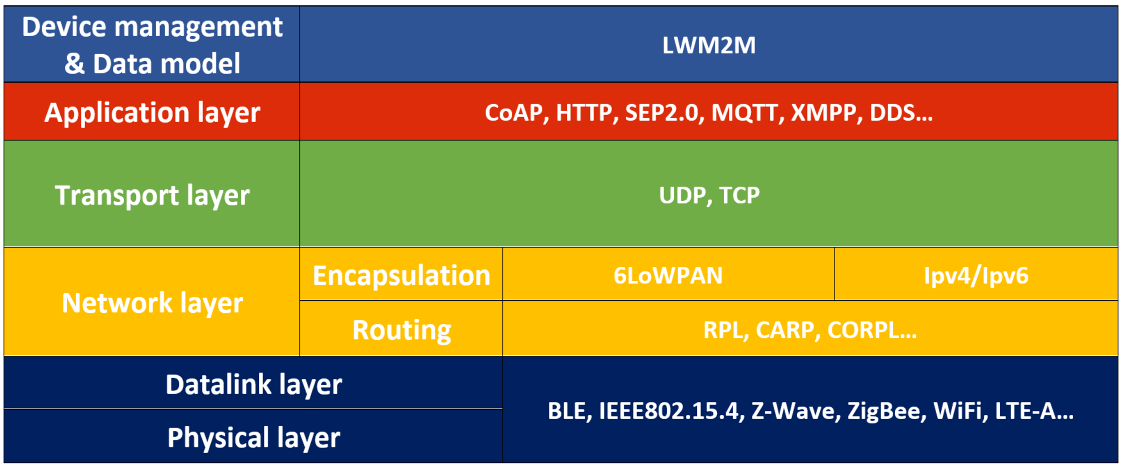


Figure 2.2. IoT layered protocol architecture.

2.1.3 Aggregation layer

The aggregation layer has the important task of aggregating, combining and bridging communications from different devices and protocols. Furthermore, aggregation layer adapts legacy protocols.

In this layer, data collected from multiple devices come in different formats and at different sampling rates, thus in order to aggregate, process, and store data in a format that can be used for analytics applications, ETL tools are used.

These tools operate in three different phase:

Extraction: refers to acquiring data from multiple sources and multiple formats and then validating to ensure that only data that meet a criterion are included;

Transformation: includes operations such as splitting, sorting, merging and transforming the data into a desired format;

Loading: refers to the process of loading the data into a database that can be used for analytics applications.

2.1.4 Resource management

Resource management involves discovering and identifying all available resources, partitioning them to maximize a utility function which can be in terms of cost, energy, performance, etc., and, finally, scheduling the tasks on available physical resources.

It performs identification and location of a device by storing and indexing metadata information about each object and then it discovers the target service that needs to be invoked. An effective discovery algorithm ensures a minimized delay and a good user experience.

It is a very important element in distributed systems because selecting and provisioning the resources, greatly impacts Quality of Service (QoS) of the IoT applications.

2.1.5 Data management & analytics

This layer collects data from the integration layer and provides the ability to process and act upon it. For this purpose, related procedures such as data acquisition, filtering, transmission, and analysis beside machine-learning algorithms are used to extract knowledge and patterns from the raw data in order to archive these records for future reference.[16]

Focusing on the 3Vs characteristic of Big Data namely velocity, volume, and variety, different data-processing approaches have been introduced.

The three main approaches to elaborate Big Data are described to following.

CLOUD computing

Cloud computing, due to its on-demand processing and storage capabilities, can be used to analyze data generated by IoT objects in batch or stream format and creating a streamlined process for building IoT applications.

However, integrating cloud computing with an IoT middleware limits the practicality and full utilization of cloud computing in scenarios where minimizing end-to-end delay is the goal.

Real Time computing

Real time analytics is referred to the process of analyzing large volume of data at the moment it is produced or used.

To fulfill its task, Real-time relies on the Edge Computing paradigm that aims to optimize cloud computing systems by taking the control of computing applications closer to the end user.

Processing and storage capability of these devices can be utilized to extend the advantages of using cloud computing by creating another cloud, known as Edge Cloud, near application consumers, in order to:

- Decrease networking delays;

- Save processing or storage cost;
- Perform data aggregation;
- Prevent sensitive data from leaving the local network.

FOG computing

Fog Computing aims to keep the same features of Cloud, such as networking, computation, virtualization, and storage, but also meets the requirements of applications that demand low latency and specific QoS requirements.

Fog Computing can be considered as a hybrid between Cloud computing and Real time computing.

2.1.6 Apps & software layer

This layer provides the diverse kinds of services requested by the customers that depend on the specific use case that customers want to adopt.

To provide a way for these services to communicate with the device-oriented system a modular front-end architecture is needed. This architecture can support existing Web server-side technology, such as Java Servlets/JSP, PHP, Python, Ruby, etc.

Services and device-oriented system communication are achieved following three main approaches:

- Create web-based front-ends and portals that interact with devices and with the event-processing layer;
- Create dashboards that offer views into analytics and event processing;
- Using machine-to-machine communications (APIs).

2.1.7 Device management

Device management (DM) allows to integrate, organize, monitor and remotely manage IoT devices, offering features critical to maintaining the health, connectivity and security of the them along their entire lifecycles. Such features include:

- Devices configuration;
- Devices provisioning;

- Devices monitoring and diagnostics;
- Devices troubleshooting.

Device management consists of a server-side system that communicates with devices via various protocols providing both individual and bulk control and a management agent that runs on a managed node offering an interface to manage it.

Available standardized device management protocols include the Open Mobile Alliance's Device Management (OMA-DM) and Lightweight Machine-to-Machine (OMA-LwM2M).

2.1.8 Identity and access management

This layer provides a set of rules and mechanisms for access control in order to identify devices, sensors, monitors, and manage their access to sensitive and non-sensitive data.

At this scope, there are various mechanisms able to uniquely identify devices or objects in IoT ecosystem, such as:

Ucode: which generate 128-bit codes and can be used in active and passive RFID tags;

Electric Product Code (EPC): which creates unique identifiers using Uniform Resource Identifier (URI) codes.

Methods for identifying objects and setting their access level play an important role in the whole ecosystem because on one side they affect both the amount of time required to establish trust and the degree of confidence, on the other, being able to globally and uniquely identify and locate objects decreases the complexity of expanding the local environment.

2.1.9 Security

Security management is essential to ensure security and privacy in the IoT environment. It operates in each layer of the stack using both hardware and software mechanisms. It firstly aims to secure communications channels and informations from unauthorized access to avoid identity theft and to protect privacy.

In an IoT environment, resource constraints are the key barrier for implementing standard security mechanisms in embedded devices so specific mechanisms based on identification, authentication and authorization of user and cryptography have to be implemented.

2.2 IoT solutions

Several IoT solutions have been proposed today to fulfill all the user's needs. Starting from the fact that IoT environment requires a huge collection of elements to operate correctly as has been said earlier through this chapter, a huge number of IoT solutions can exist.

So, in this direction, big IT companies such as Intel, Bosch, IBM, NXP, Google etc. make available a diversified portfolio of products in order to express at best the potential of their solutions exploiting the IoT paradigm.

For example, Intel offers a complete IoT solution called "Intel IoT Platform" that includes end-to-end reference architectures model, developer kits, tools and devices management [9]. Siemens proposes "MindSphere" a cloud-based, open IoT operating system that connects devices and enables to harness the wealth of data generated by the environment with advanced analytics [19]. NXP as well as Texas Instruments provide finished solutions integrating all the necessary IoT infrastructure building blocks.[15][35]

These are just some of the existing IoT solutions.

Among the biggest IT companies, one of the most successful and innovative that invests a lot in the IoT domain is STMicroelectronics.

2.2.1 STMicroelectronics

STMicroelectronics, commonly called ST, is a French-Italian multinational electronics and semiconductor manufacturer.

It was formed in 1987 by the merger of semiconductor companies SGS Microelettronica (Società Generale Semiconduttori) of Italy and Thomson Semiconducteurs, the semiconductor arm of France's Thomson. At the time of the merger the company was known as SGS-THOMSON but took its current name in May 1998.

By 2005, STMicroelectronics was ranked fifth, behind Intel, Samsung, Texas Instruments and Toshiba, but ahead of Infineon, Renesas, NEC, NXP, and Freescale.

STMicroelectronics is Europe's largest semiconductor chip maker based on its revenues (US$ 8.35 billion in 2017) with about 45.000 employees worldwide and 80 sales and marketing offices in 35 countries.

2.2.2 ST product portfolio

Offering one of the industry's broadest product portfolios, ST develops products for a wide range of market applications, including automotive products, computer peripherals, telecommunications systems, consumer products, industrial automation, control systems and the Internet of Things.[29]

The company operates through two segments:

> ***Semiconductors***: it designs, develops, manufactures and markets a broad range of products, including discrete and standard commodity components, application-specific integrated circuits, full custom devices and semi-custom devices and application-specific standard products for analog, digital, and mixed-signal applications;
>
> ***Subsystems***: it designs, develops, manufactures and markets subsystems and modules for the telecommunications, automotive and industrial markets including mobile phone accessories, battery chargers, ISDN power supplies and in-vehicle equipment for electronic toll payment.

Figure 2.3. STMicroelectronics logo

MCU platforms

STMicroelectronics produces a varied and complete range of MCU platforms from high-performance to ultra-low-power, including:

- Robust low-cost 8-bit MCUs
- 32-bit Arm MCU based on:
  - Cortex-M0 and M0+;
  - Cortex-M3;
  - Cortex-M4;
  - Cortex-M7.

The STM32 MCU platforms collection is illustrated in Figure 2.4.

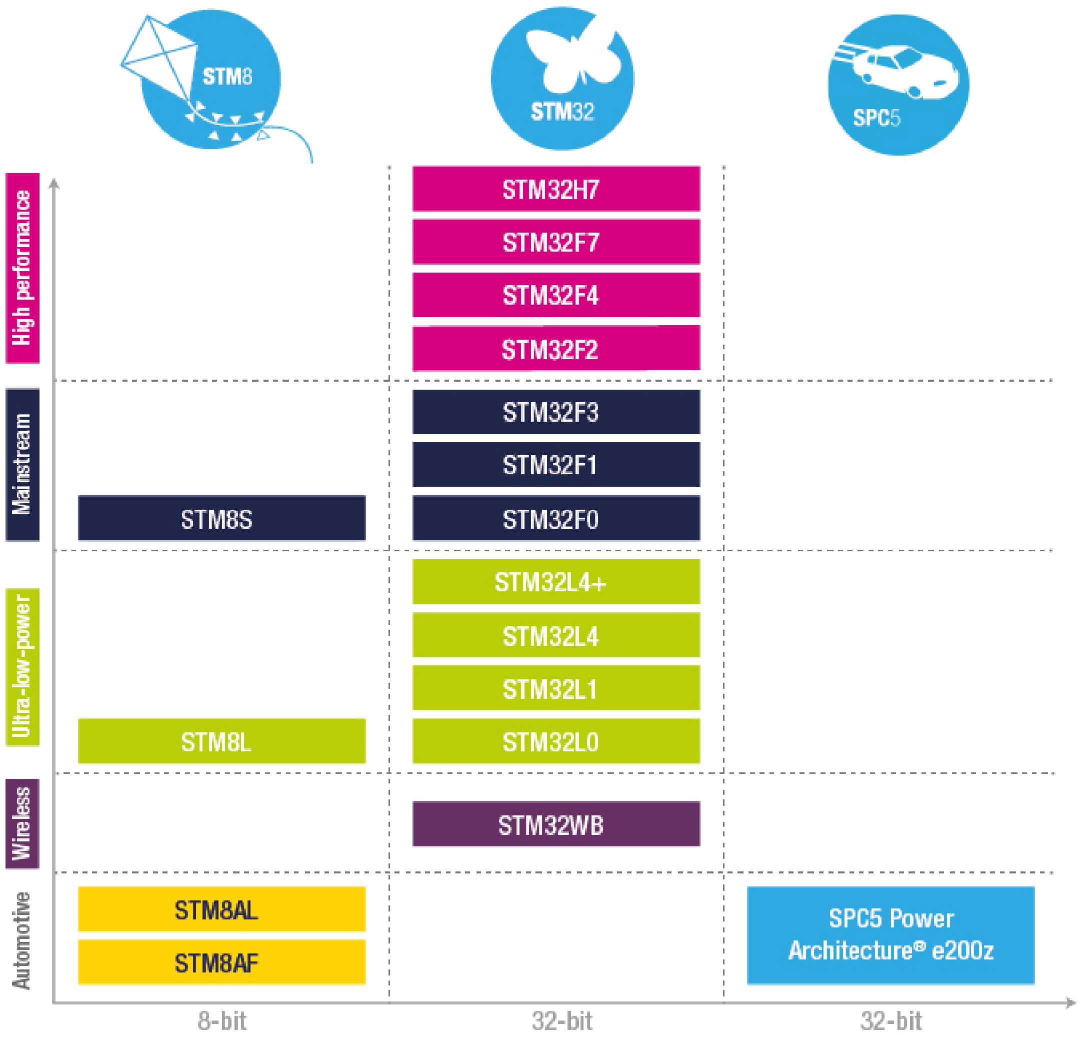


Figure 2.4. STMicroelectronics microcontrolers portfolio.

STM32 MCU Nucleo

The highly affordable STM32 Nucleo boards permit to quickly create prototypes with any STM32 MCU and since they are supported by a wide choice of IDEs and direct access to mbed online resources www.mbed.org they allow to easy develop applications.

STM32 Nucleo boards integrate an ST-Link debugger/programmer and sharing the same connectors, they can easily be extended with a large number of specialized application hardware including Arduino Nano for Nucleo-32 and Arduino Uno rev3 & ST morpho connectors for Nucleo-64. The STM32 Nucleo boards portfolio is illustrated in Figure 2.5.

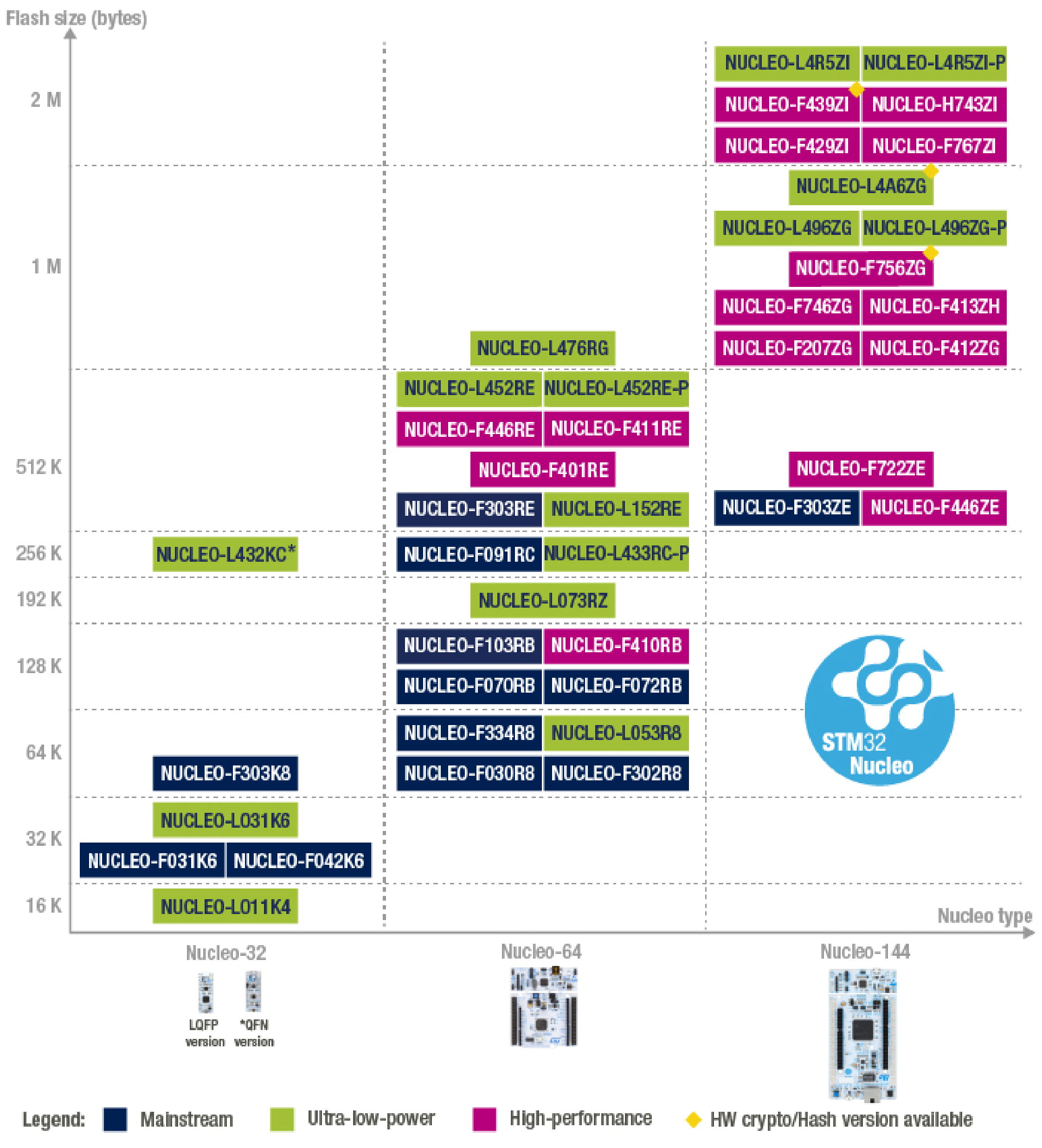


Figure 2.5. STMicroelectronics nucleo boards.

Embedded software

ST and a large group of 3rd party partners provide an extensive range of embedded software including libraries, snippets, middleware, codecs and protocol stacks to offer an assisted software development process with a certain abstraction from the hardware. The STM32 embedded software portfolio is illustrated in Figure 2.6.

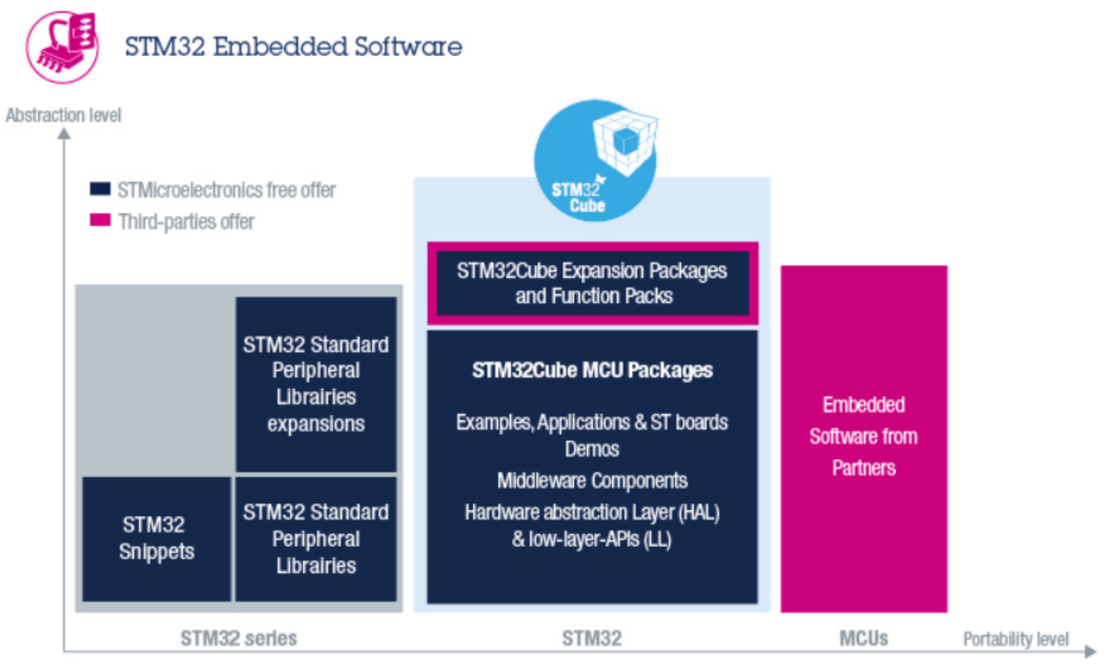

STM32 Embedded Software
Abstraction level
STMicroelectronics free offer
Third-parties offer
STM32 Cube
STM32Cube Expansion Packages and Function Packs
STM32 Standard Peripheral Libraires expansions
STM32Cube MCU Packages
Examples, Applications & ST boards Demos
Middleware Components
Hardware abstraction Layer (HAL) & low-layer-APIs (LL)
Embedded Software from Partners
STM32 Snippets
STM32 Standard Peripheral Librairies
STM32 series
STM32
MCUs
Portability level

Figure 2.6. STMicroelectronics embedded software.

STM32Cube Packages

STM32Cube is a comprehensive software tool, that allow to significantly reducing development efforts, time and cost and consists of:

STM32Cube MX: a graphical software configuration tool that allows the generation of C initialization code such as clock tree, peripherals and middleware setup;

STM32Cube MCU: a comprehensive embedded software platform specific, which includes the STM32Cube HAL embedded abstraction-layer software and a consistent set of middleware components such as RTOS, USB, TCP/IP and many others.

STM32Cube Expansion Packages

STM32Cube Expansion Packages is a basic set of software components like HAL, LL APIs and middleware.

Thanks to the compatibility with the functionalities of the STM32Cube embedded software libraries and tools, it permits to enable the usage of a multitude of ST devices together with the most appropriate STM32 MCUs in such way to ensure a rapid application development based on proven and validated software elements.

STM32 Function Packs

STM32 Function Packs are a combination of low-level drivers, middleware libraries and sample applications assembled into a single software package that leverage the modularity and interoperability of ST devices and expansion board.

Function packs may also include additional libraries and frameworks not present in the original X-CUBE packages, in order to enable new functions and creating more pertinent and usable systems for developers.

Used together, function Packs help jump-start the implementation and the development of applications in different domains aiming to best satisfy the final user application requirements.

ST IoT solutions

ST offers the simplest, fastest and most robust way to develop applications for IoT with a complete hardware and software portfolio covering all the necessary building blocks in such way that developers can easily find solutions to create Smart Things for a number of IoT application domains.

Particular attention is given to wearable devices and smart home solutions where ST's products offers high precision sensing, low power consumption, tiny form factor and outstanding performance. Example of ST IoT solutions are:

SL-DEV-MON-IBM: used to a Smart Appliances monitoring connecting an IoT node with motion and environmental sensors to the IBM Watson IoT platform. This IoT solution uses the MQTT protocol to transmit sensor data and receive commands connecting through a Wi-Fi network

SL-NET-UTIL: used to create a Smart Utility sensor node over 6LoWPAN sub GHz mesh network that integrates proximity sensor, motion and environmental sensors.

SL-TMON-MS: used to create a Smart Thermostat connecting an IoT node with environmental sensors to the Microsoft Azure IoT platform. This IoT solution uses the MQTT protocol for bidirectional communication connecting through a Wi-Fi network.

These are just three of the many existing solutions proposed by ST, whom full list is provided on the ST website [23].

Chapter 3

FOTA architectures

FOTA, as has been said in the first chapter, consists in different methods that aim to update the firmware of a device without any kind of physical access, so since the firmware is powering the reliability and scalability of connected devices, FOTA represents one of the most critical component for IoT architectures.

To guarantee a successful firmware update process, it has to be compliant with the various types of networks and moreover with the application requirements.

For this reason, a vast and diversified selection of protocols and architectures exist and all of them, even if in different ways, ensure benefits and the compliance with specifications.

In this chapter requirements and architectures of a firmware update are discussed with particular attention to the main components and protocols.

3.1 Requirements

During firmware update process several issues can be observed. These issues might happen during download phase or during update phase inside the device itself and they can be broadly categorized as safety related issues and security related issues.

To succed in a correct and efficient firmware update there are many requirements that FOTA methods have to satisfy, including:

Safety: an image can be downloaded into a device only if it has been authenticated and verified, for instance by checksum or CRC techniques.

Security: data should be sent utilizing secure protocol and encryption mechanisms in order to slow down the risk to be attacked. For this reason, at least these considerations must be taken:

  - Firmware signature;
  - Integrity for signature checking Mechanisms;
  - Secure communication channel;
  - Reliable switching mechanisms between of the old and new firmware.

  Reliability:firmware should be sent with no loss of data and the system must guarantee to install firmware updates in a robust fashion so the update does not break the device functionality and furthermore that a power failure at any time must not cause a failure of the device.

  Scalability: FOTA should be design to support large number of devices by providing not only unicast but even multicast capabilities.

  Flexibility: FOTA should support different devices providing an efficient mechanism.

  Power consumption: FOTA should take into account energy efficiency and battery lifetime considerations in order to consume the minimum quantity of energy.

Besides requirements, FOTA methods must ensure different important functionalities such as:

- Manage large update packages;
- Manage corrupted update packages;
- Accept/Reject packages;
- Operate with a small bootloader because hardware constraints;
- Guarante a robust wireless connection;
- Minimize update time.

3.2 Components and architecture

Before developing FOTA methods, in order to satisfy all requirements including those specific to the application, it is very important to choose the correct architecture to adopt and which components to use.[14]

For these reasons, it is necessary take into account:

- Firmware size;

- Trigger methods;
- Devices parameters;
- Device dependencies.

A typical FOTA architecture is composed of:

Server: the entity that stores and transmits the firmware image;

Node: a device with enough flash memory to store the firmware image and all the data required for the update process;

Firmware: a binary containing the complete software of a device or a subset of it;

Wireless communication channel: medium used to transfer the firmware image from server to the node;

Bootloader: a binary code able to make decision before running a firmware.

Before presenting firmware updating techniques and protocols a more detailed description of Flash memory, Bootloader and wireless communication channel are shown in the next subsections.

3.2.1 Flash memory

Flash memory is a non-volatile memory chip used for store data permanently including firmware, useful data for the system and the boot loader. During the firmware update process, flash memory also store metadata that bootloader uses to make decisions about booting the system, validate data or swap firmwares.

Often, especially for embedded system, flash memory is chosen considering different constraint regarding:

Dimension: it depends on firmware dimensions and on the functionality that the whole system provides;

Flexibility: it depends on how it is partitioned, writable and erasable in order to permit an easy and efficient memory management;

Cost: it depends on its dimension, flexibility and technology.

With this in mind, firmware update process in embedded systems requires a deep knowledge of the flash memory.

3.2.2 Bootloader

The bootloader is a small segment of code that runs before any operating system is running, which handles many operations, like:

- verifying the correctness of available firmwares;
- booting of a firmware;
- replacing the old firmware.

There are two bootloader behavioural models, the first is when it automatically detects new update and manages its own flashing process while the second is when it needs an external source command to enable the flashing process whenever there is a new update available. So a communication interface is needed for interaction between software application and the embedded system.[10]

3.2.3 Wireless communication channel

There are several wireless communication channel to convey the firmware images from server to nodes, and the choice of the right one depends directly on network topologies, devices features and system requirements. In any case the specified mechanism needs to be agnostic to the distribution of the firmware. The most common communication protocol for FOTA are:

- ZigBee;
- WiFi;
- Bluethooth Low Energy(BLE);
- 6LoWPAN;

A typical FOTA architecture is depicted in Figure 3.1:

3.3 Firmware update techniques

In general, different firmware update techniques exist, but in field, there are two main techniques used for this scope called firmware replacement and firmware patching. Both are described here:

Firmware replacement: firmware replacement is the simplest update technique used for firmware updates over the air. To perform an update using

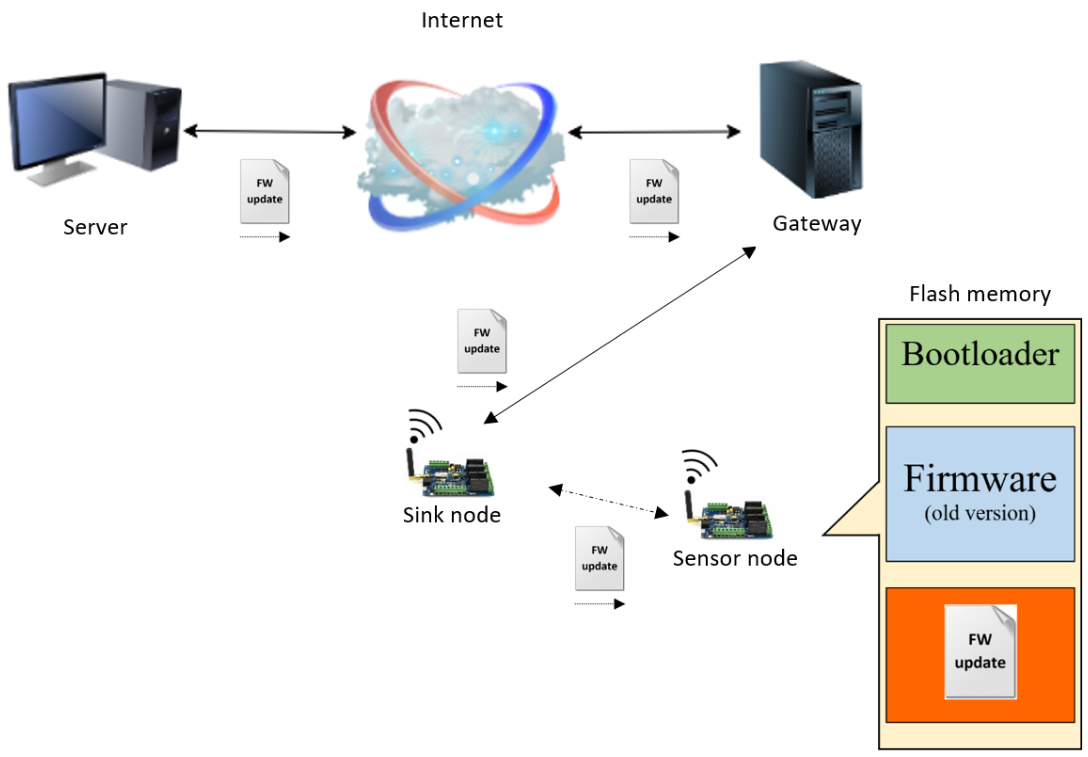


Figure 3.1. Example of a FOTA architecture.

firmware replacement, the firmware binary file has to be downloaded completely first and then updated;

Firmware patching: because of a lot of the devices in the field has to guarantee specific lifetimes or are restricted in the time being online, or just in resources being available on the devices, resource-saving concepts are required. Firmware patching is a technique to reduce the data which has to be transmitted. So having the knowledge of the currently installed firmware version and which are the changes to be made, a patch can be created and then sent to the device. In this case, a patch contains only the differences between two versions and the position where they are located in the present firmware. During the download of the patch, the device can stay operable and switch to update mode as soon as the download is complete. Due to the fact, that the firmware is manipulated directly there is still a high risk to have a bricked device in case of an unexpected reset, which has to be handled properly.

In both cases, in between of the downloading and update process, it is important to understand where to store the new firmware and in which mode the firmware is running during download.

Here often used concepts are presented:

- The device updates the firmware on the fly running in a specific update mode (typically in the bootloader). Since every problem during the update process can lead to a bricked device, this represents the most dangerous concept. The risk can be minimized adopting rollback criteria.
- The device writes the data in a specific part of the non-volatile flash memory, running in a normal mode. When all the data is received and stored, the operation mode is changed to update mode, which is often part of the bootloader. Due to the fact that both the new firmware and the old one are stored in the flash memory at the same time, at least double of the memory space is needed and if a read-only firmware for factory reset is required then the flash memory required space is three times the firmware size. Despite the huge amount of memory it requires, this is the most robust method that ensure a successful firmware update.

3.4 Typical firmware update protocol

To meet different IoT architectures and several requirements, disparate management protocols are present today. Management protocols are a list of rules intended to enable device management and service enablement including the firmware update process.

In order to manage the bootloader and establish a communication between the bootloader itself and the system, some additional information are required that in this scope are called “Metadata”.

The metadata format encodes the information that the bootloader and the device need in order to make decisions.

So in case of a firmware update, several decisions are made for the succeeding of the operation, such as:

- Check if the source of the firmware is reliable;
- Check if the firmware has been corrupted;
- Check if the new firmware is older than the active firmware;
- Decide when the device should start the update process;
- Decide how the device should apply the update;

- Decide where the firmware should be stored.

Hereafter, a typical FOTA protocol is described.

The protocol Figure 3.2 illustrates that an author creates a trustable firmware image in order to send it to a node.

Depending on existing device management protocols the firmware image can be pushed to the device or fetched by the device and typically a file server is involved.

The node stores the firmware into the flash memory and thanks to the bootloader it will understand that a new firmware is ready to run.

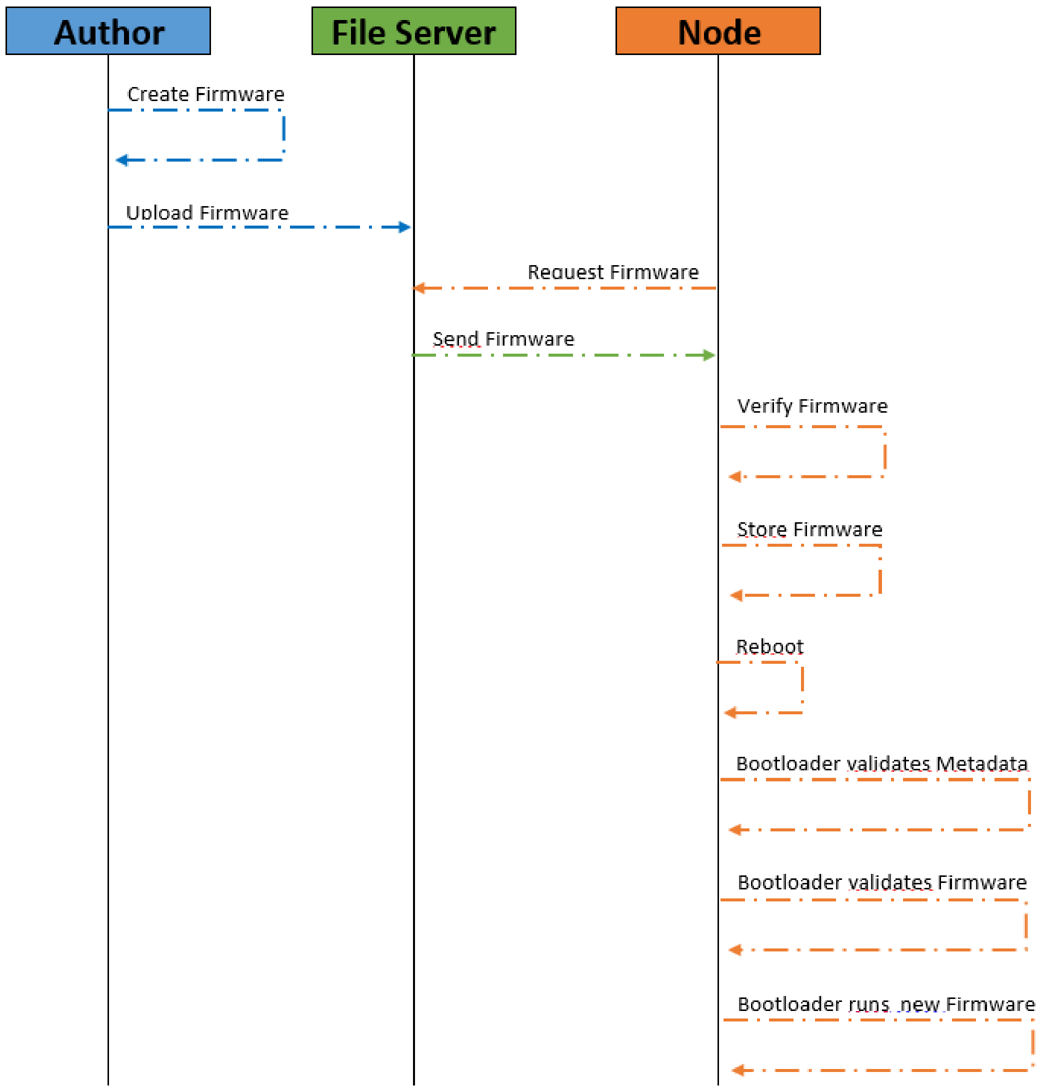


Figure 3.2. Example of a FOTA protocol.

3.5 FOTA solutions

Since the IoT scenario is very diversified and IoT architectures are strictly correlated with the demanding application requirements, there are no optimal solutions that can manage the firmware update. To face with this lack, a new research area has born from the need of creating new firmware update methods for the IoT new paradigm. Even though a large number of solutions have not yet been proposed, some FOTA method already exists.

For example, a FOTA method has been proposed by Erkki Moorits and Gert Jervan that formalize a method for firmware update in memory constrained low-power controllers used in marine aids to navigation (AtoN) and telematics systems[13]. Another solution based on IEEE 802.11s over a SG AMI network has been proposed by Samet Tonyali, Kemal Akkaya and Nico Saputro. In their work, they address the problem of multicasting the firmware update securely in a SG AMI network and developing a secure and reliable multicast-over-broadcast protocol[36].

Again, a FOTA method has been proposed by Goran Jurković and Vlado Sruk. The two authors implemented the system software upgrade using a GPRS module to enable communication with the server and a software agent to enter directly into the program memory, on a system based on ATMEL processor[11].

Also STMicroelectronics provides a set of firmware update solutions where in some cases regard complete FOTA methods implementation, while the others focus on particular phases of the process. Here two solutions are considered.

The first solution, called FP-CLD-AZURE1 function pack[20], besides allowing to safely connect an IoT node to Microsoft Azure IoT, transmit sensor data and receive commands from Azure cloud applications, it includes a sample implementation for firmware update over the air (FOTA) built on Wi-Fi and Ethernet connectivity links.

The second solution, called X-CUBE-SBSFU expansion pack[31], focuses only on the booting and updating phases adding security features. The update process is performed in a secure way to prevent access to confidential on-device data such as secret code and firmware encryption key and unauthorized updates.

In addition, before every execution, a security mechanism namely Secure Boot checks the authenticity and integrity of user application code to ensure that invalid or malicious code cannot be run.

Chapter 4

Standards and protocols

The FOTA method proposed in this thesis is based on 6LoWPAN protocol in order to enable M2M communication exploiting IPv6 internet protocol and uses OMA-LwM2M as Device Management and Data Model protocol, that is specified in the CoRE CoAP application protocol.

Before going into the practical implementation of the system, a detailed description of these standards and protocol is presented in this chapter.

4.1 6LoWPAN

6LoWPAN, acronym of IPv6 over Low-Power Wireless Personal Area Networks, is a networking technology or adaptation layer defined by IETF standards that allows IPv6 packets to be carried efficiently within constrained link, such as IEEE 802.15.4. It has a limited frame size and bandwidth and requires a minimal use of code and memory. [18]

6LoWPAN also enables several features, including:

- Bootstrapping;
- Network autoconfiguration using neighbour discovery;
- Fragmentation and reassembly;
- Compression;
- Security;
- Mobility.

4.1.1 6LoWPAN architecture

Defining a stub network as a network which IP packets are sent from or destined to, but which doesn't act as a transit to other networks, the 6LoWPAN architecture can be represented by a union of Low-power Wireless Area Networks (LoWPANs), which are IPv6 stub networks.

A LoWPAN consists of nodes, which may play the role of host or router, along with one or more edge routers (or border router). The latter plays an important role as it is in charge of connecting LoWPANs to other IP networks including IPv4 and IPv6 ones. Moreover, it routes traffic in and out of the LoWPAN, while handling 6LoWPAN compression and NeighborDiscovery for the LoWPAN.

The overall 6LoWPAN architecture is presented in Figure 4.1

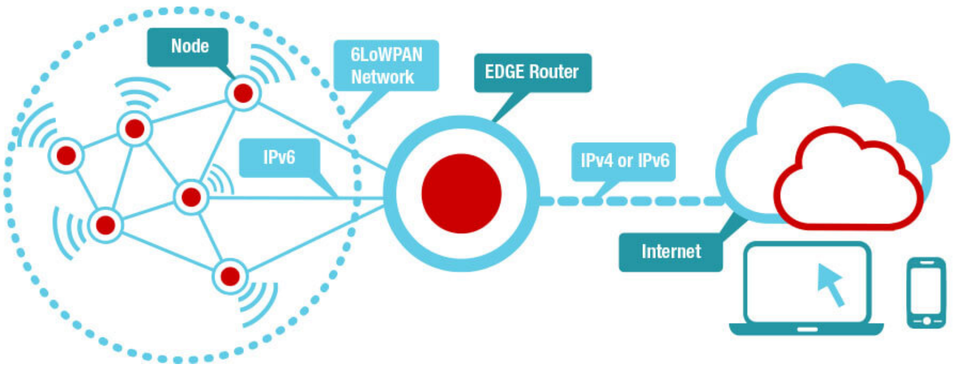


Figure 4.1. 6LoWPAN overall architecture.

4.1.2 Protocol stack

6LoWPAN is a "substrate" of the Network layer that allows to adapt IPv6 to IEEE 802.15.4 standard. Referring to the standard TCP/IP Protocol Stack, the 6LoWPAN is located with the IEEE 802.15.4 in the physical and Data link layers and with 6LoWPAN Adaptation Layer in the network layer.

The 6LoWPAN protocol stack is shown in Figure 4.2

IEEE 802.15.4

Standard for wireless communication that defines the Physical (PHY) and Media Access Control (MAC) layers. Standardized by the IEEE (Institute for Electrical and Electronics Engineers) the 802.15.4 focusses on communication between devices in constrained environment with low memory, low power and low bandwidth.

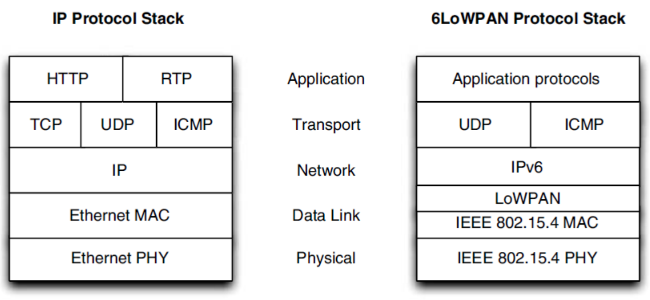


Figure 4.2. TCP/IP and 6LoWPAN protocols stacks.

The Physical Layer (PHY) allows to transmit in a range between 10 to 30 meters with three different modes:

- 20 kbps at 868 MHz;
- 40 kbps at 915 MHz;
- 250 kbps at 2.4 GHz (DSSS).

IEEE 802.15.4 supports:

- Simple addressing but no routing;
- Star, mesh and Peer-to-Peer topologies;
- Auto configuration with neighbour discovery;
- 16-bit short or IEEE 64-bit extended media access control addresses;
- Header and payload compressions;
- Beaconless mode (Simple CSMA algorithm) and Beacon mode with superframe (Hybrid TDMA-CSMA algorithm);
- Up to 64k nodes with 16-bit addresses;
- Small packet size;
- Low power, typically battery operated;
- AES block cypher in several modes (AES-CCM-64 mandatory).

Adaptation layer

The 6LoWPAN adaptation layer sits between Data-link and original Network layer. It effectively becomes part of the Network layer, but only on the specified Data-Link layers.

Since a IPv6 packet is too large to fit into a single 802.15.4 frame, the main task of the 6LoWPAN adaptation layer is to compress all headers (adaptation, network and transport layers) down to a few bytes eliding header fields in order to fit an IPv6 packet in 802.15.4 frame. This is possible by means of fragmentation, compression and reassembly techniques.

4.1.3 Fragmentation and compression

IPv6 requires underlying links to support Minimum Transmission Units (MTUs) of at least 1280 bytes, thus in order to enable the transmission of IPv6 frames over IEEE 802.15.4 radio links, the IPv6 frames need to be divided into several smaller segments. For this reason, fragmentation, header compression and reassembly are required.[34]

In this extent, 6LoWPAN uses stacked headers and extension headers Figure 4.3 to define these capabilities as follow:

Mesh address header: is used to forward packets over a 6LoWPAN multi hops network. Its length is between 5 and 17 bytes, depending on the addressing mode in use and includes three fields:

- ***Hop limit***: used to limit the number of hops for forwarding;
- ***Source address***: used to indicate the IP start point;
- ***Destination address***: used to indicate the IP end point.

Fragmentation header: is used when the payload is too large to fit in a single IEEE 802.15.4 frame thus needs to be divided. The fragment header contains three fields:

- ***Datagram size***: describes the total (un-fragmented) payload;
- ***Datagram tag***: identifies the set of fragments and is used to match fragments of the same payload;
- ***datagram offset***: identifies the fragment's offset within the un-fragmented payload.

The fragment header length is 4 bytes for the first header and 5 bytes for all subsequent headers. The fragmentation header is elided for packets that fit into one single IEEE 802.15.4 frame

Compression header: is used to reduce the size of the IPv6/UDP address header by means of header compression techniques based on some network default values such as flat address spaces and unique MAC addresses. 6LoWPAN compresses IPv6 addresses by:

- Eliding the IPv6 prefix;
- Compressing the IID;
- Compressing with a well-known “context”;
- Compressing multicast addresses;
- Removes duplicated information that can be derived from other layers.

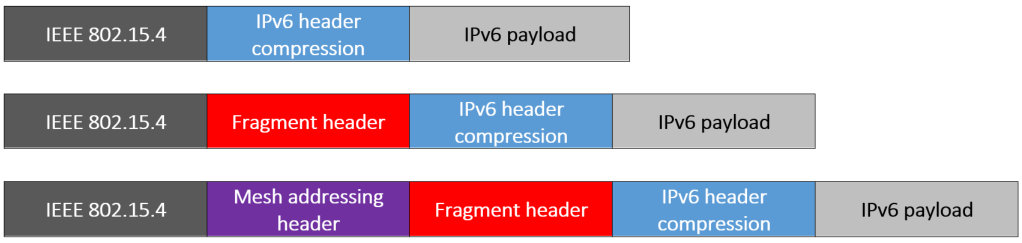


Figure 4.3. 6LoWPAN stacked headers.

4.1.4 Bootstrapping

Bootstrapping represents a set of operations that aim to establish a network, joining new nodes and re-organize them. In particular its main tasks are:

- Link layer connection between nodes;
- Network layer address configuration, discovery of neighbours and node registration;
- Route initialization;
- Network maintenance and re-organization.

Neighbour Discovery and Node Registration

During this step, nodes perform the *stateless autoconfiguration* that allows devices inside the 6LoWPAN network to automatically generate their own IPv6 address. Then, nodes register with their neighbouring routers exchanging NR/NC messages. To maintain node registrations, routers keep a binding table of registered nodes that is updated every time nodes, by means of new NR messages, refresh their registration.

Furthermore, node registration enables:

- Host/router unreachability detection;
- Address resolution;
- Duplicate address detection (DAD).

Routing and mobility

6LoWPAN have techniques to deal with address changes caused by nodes moving between LoWPANs, and to mitigate the effects of network mobility. Since the Internet Protocol (IP) does not calculate routes, routing protocol are used at this scope.

There are two main classes of routing protocols useful for 6LoWPAN:

Distance-vector: links are associated with cost, used to find the shortest route. Each router along the path store local next-hop information about its route table;

Link-state: each node acquires complete informations about the network, typically by flooding, then it calculates a shortest-path tree for each destination.

4.1.5 Security

6LoWPAN offers security sessions both in Datalink and Network layers. In particular, in the Datalink layer, IEEE 802.15.4 provides a built-in encryption and an integrity check mechanism based on the 128-bit Advanced Encryption Standard (AES) and a counter with CBC-MAC mode (CCM).

Moreover, an end-to-end security is provided by the Datagram Transport Layer Security (DTLS) that provides privacy and data integrity .

4.2 CoAP

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for machine-to-machine (M2M) applications. It is based on the REST model over UDP and it is designed to use minimal resources, both on the device and on the network.

This protocol has the characteristics of keeping the message overhead small, limiting the fragmentation and supporting multicast, having a 4-byte fixed header and a compact encoding of options which enables small messages that cause no or little fragmentation on the link layer.

CoAP supports XML, JSON, CBOR plus many other data format and actually uses DTLS based on PSK, RPK and Certificate security in order to provide strong protection. It is located in the application layer and interfaces with UDP layer which formats the received data into a datagram that is then sent to the lower layers. Similar to HTTP, CoAP also uses a request-response model with a reduced set of methods. Request message is sent by a client using a *Method Code* to request an action on a URI identifiable resource and the server replies with a *Response Code* which may include a resource representation. During this communication, all CoAP messages are exchanged asynchronously between the end-points.

CoAP also supports a URI identification and by means of the Resource Directory (RD), which hosts descriptions of resources, provides a way to discover the properties of the nodes on the network.

4.2.1 Block-wise transfer

CoAP supports large data exchange to allow that a huge amount of data is sent in the network. When a resource representation is larger than the size that can be comfortably transferred in the payload of a single CoAP datagram, a Block option can be used to indicate a block-wise transfer.[2]

As payloads can be sent both with requests and with responses messages, CoAP provides two separate options for each transfer direction:

Block1: indicates the transfer of the resource representation that refers to the request;

Block2: indicates the transfer of the resource representation that refers to the response.

4.2.2 Methods

CoAP, compared to HTTP, provides a reduced set of methods that can be used:

> ***GET***: is used to retrieve the representation of the resource identified by the request URI;
>
> > ***OBSERVE***: is a new option for the GET method used to have the current representation of a resource over a period of time.
>
> ***POST***: is used to request the server to create a new subordinate resource under the requested parent URI;
>
> ***PUT***: is used to request the server that the resource identified by the request URI be updated or created with the enclosed message body;
>
> ***DELETE***: is used to request that the resource identified by the request URI be deleted.

4.2.3 Message types

The types of messages that CoAP offers are:

> ***Confirmable message (CON)***: a confirmable message requires a response, either a positive acknowledgement or a negative acknowledgement. In case acknowledgement is not received, retransmissions are made until all attempts are exhausted;
>
> ***Non-confirmable message (NON)***: a non-confirmable request is used for unreliable transmission. Such a message is not generally acknowledged or it does not require an acknowledgement;
>
> ***Acknowledgement message (ACK)***: an acknowledgement message acknowledges that a specific confirmable message arrived;
>
> ***Reset message (RST)***: a reset message indicates that a specific message (Confirmable or Non-confirmable) was received, but some context is missing to properly process it. Reset message is also useful as an inexpensive check of the liveness of an endpoint (CoAP ping).

To make implementation simpler, the number of response code has also been reduced.

Message format

The CoAP messages are encoded in a simple binary format and they starts with a fixed-size 4-byte header followed by a sequence of optional fields.[8] The header of a CoAP message Figure 4.4 contains the following parameters:

> ***Ver (Version)***: 2-bit unsigned integer.Indicates the CoAP version number;
>
> ***T (Message Type)***: 2-bit unsigned integer. Indicates the if the message is Confirmable, Non-Confirmable, Acknowledgement or Reset;
>
> ***TKL (Token Length)***: 4-bit unsigned integer. Indicates the length of the variable-length Token field (0-8 bytes);
>
> ***Code***: 8-bit unsighted integer. 3-bit class (most signification bits). 5-bit detail (least significant bits). Request Method (1-10) or Response Code (40-255);
>
> ***Message ID***: 16-bit identifier used for matching responses and detect message duplication;
>
> ***Token***: optional field (0 to 8 bytes) is used to correlate requests and responses;
>
> ***Option***:optional field that indicates the options that can be included in a message;
>
> ***Payload***: optional field.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| T |  TKL  |      Code     |          Message ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Token (if any, TKL bytes) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4.4. CoAP message format.

4.3 OMA-LwM2M

Open Mobile Alliance Lightweight Machine to Machine (OMA-LwM2M) is a device management protocol designed for remote management of M2M devices and related service enablement. It features a modern architectural design based on REST, defines an extensible resource and data model based and provides several interfaces built on top of CoAP.[1]

It allows to decouple devices from the platform services using standard interfaces and defines provisioning and bootstrapping functionalities.

DTLS is used to provide a secure channel between the LwM2M Server and the LwM2M Client for all the messages exchange. In order to support very limited and more capable embedded devices, three DTLS security modes have been defined:

- Pre-Shared Key;
- Raw Public Key;
- Certificate mode.

Furthermore, LwM2M provides:

- Key management;
- Service provisioning;
- Access Control;
- Setting change;
- Devices error reporting.

4.3.1 LwM2M architecture

The LwM2M defines the application layer communication protocol between a server and a client. It is composed of:

LwM2M Server: persistent endpoint through which devices and app interact;

LwM2M Client: resides on the device and hosts resources (objects) that represent the physical device. It is typically integrated as a software library or a built-in function of a module or device.

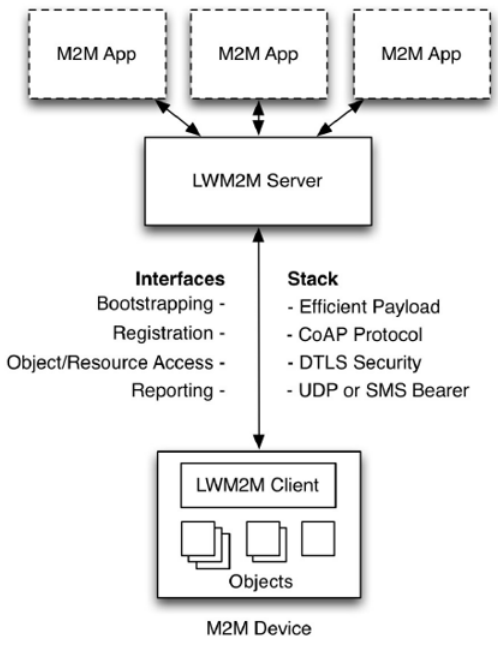


Figure 4.5. LwM2M architecture.

Four logical interfaces are defined between server and client namely:

Bootstrap interface: is used to initialize the needed Object(s) for the LwM2M Client to register with one or more LwM2M Servers. There are four bootstrap modes supported by the LwM2M Enabler:

- Factory Bootstrap;
- Bootstrap from Smartcard;
- Client Initiated Bootstrap;
- Server Initiated Bootstrap.

Device discovery and registration interface: is used by a LwM2M Client to register with one or more LwM2M Servers, maintain each registration and de-register from a LwM2M Server. The registration is based on the Resource Model with a lifetime indicated by the *Lifetime Resource* of that LwM2M Server Object Instance.

Device management and service enablement interface: is used by the LwM2M Server to access Object Instances and Resources available from a registered LwM2M Client. The interface provides this access through the use of Object operations.

Information reporting interface: is used by a LwM2M Server to observe any changes in a Resource on a registered LwM2M Client receiving notifications when new values are available. This observation relationship is initiated by sending an "Observe" operation.

4.3.2 Data model

The OMA-LwM2M Objects model Figure 4.6. describes an Object by three elements:

Objects: represent typed containers of a resources collection;

Objects instances: represent specific object types at runtime;

Resources: specify a particular view or active property of an object. They can contain a value or trigger an action in the LwM2M Client depending on their specification.

Objects, instances and resources are implicitly mapped into the URI path hierarchy with the following format:

/Object ID/Object Instance/Resource ID

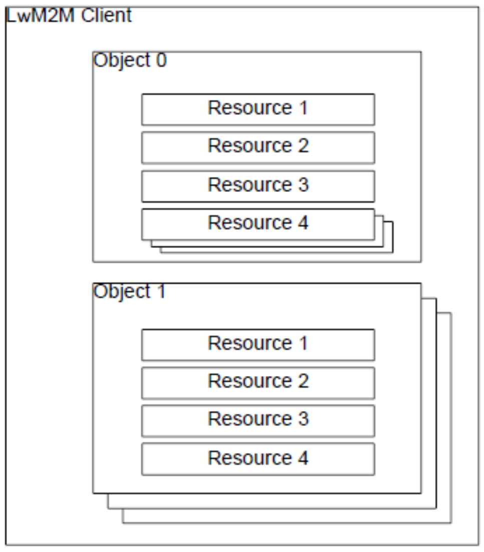


Figure 4.6. LwM2M Object model.

Objects and Resources can have single or multiple instances that can be instantiated using JSON, TLV, TEXT or OPAQUE format and can be managed through the LwM2M interfaces that define the following types of possible operations:

Resource values: read, write and execute;

Object Instances: create and delete;

Objects and their instances: read and write;

Attributes: set and discover.

4.3.3 Object and resource definition

According to the OMA-LwM2M specifications the required parameters to define an Object are:

Name: specifies the Object name;

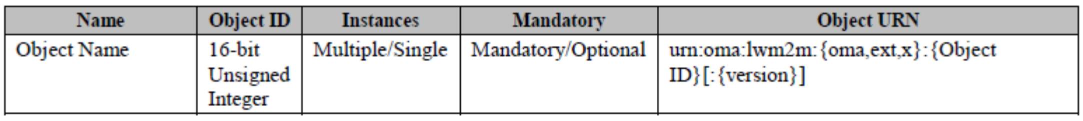
Object ID: specifies the Object ID;

Instances: indicates whether this Object supports multiple Object Instances or not;

Mandatory: if this field is “Mandatory”, then the LwM2M Client MUST support this Object. If this field is “Optional”, then the LwM2M Client should support this Object.

Object Uniform Resource Name (URN): specifies the Object URN. The format of the Object URN is urn:oma:lwm2m:oma,ext,x:Object ID”.

This Object definition template is shown in Figure 4.7.



Name	Object ID	Instances	Mandatory	Object URN
Object Name	16-bit Unsigned Integer	Multiple/Single	Mandatory/Optional	urn:oma:lwm2m:{oma,ext,x}:{Object ID}[:{version}]

Figure 4.7. OMA LwM2M Object definition template.

Regarding Resources definition, the required parameters are:

ID: specifies the Resource ID which is unique within Object;

Name: specifies the Resource name;

Operations: indicates which operations the Resource supports such as Read, Observe, Discover or Execute;

Instances: indicates whether this Resource supports multiple Resource Instances or not;

Mandatory: if this field is "Mandatory", then the LwM2M Server and the LwM2M Client MUST support the Resource. If this field is "Optional", then the LwM2M Server and the LwM2M Client should support the Resource;

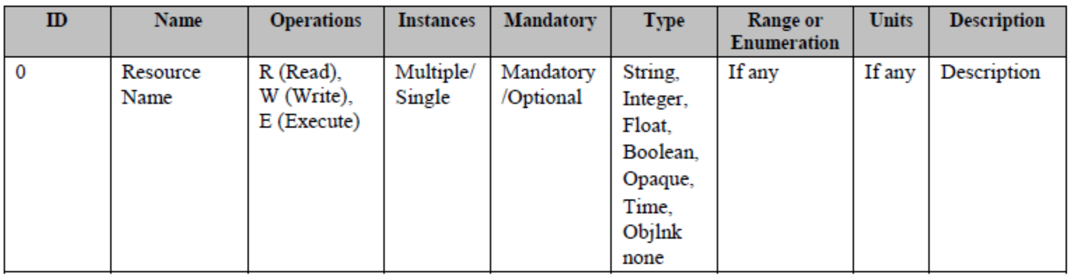
Type: Data Type indicates the type of Resource value;

Range or Enumeration: this field limits the value of Resource;

Units: specifies the unit of the Resource value;

Description: specifies the Resources description.

This Resource definition template is shown in Figure 4.8.



ID	Name	Operations	Instances	Mandatory	Type	Range or Enumeration	Units	Description
0	Resource Name	R (Read), W (Write), E (Execute)	Multiple/ Single	Mandatory /Optional	String, Integer, Float, Boolean, Opaque, Time, Objlnk none	If any	If any	Description

Figure 4.8. OMA LwM2M Resource definition template.

Define new Object

OMA make possible to define new Objects in a straightforward way. In order to register a new object is necessary first to write a specification filling out the Object template tables including Object Name, Description and if it can have Multiple Instances and second specify the list of resources the Object defines such as Resource Name, ID, Operations, Multiple Instances, Mandatory, Data Type, Range, Units and Description.

The Object IDs are registered with the OMA Naming Authority (OMNA) by OMA working groups or 3rd party organizations.

IPSO Alliance has created additional objects divided in two packs:

Starter Pack: describes 18 Smart Objects, including a temperature sensor, a light controller, an accelerometer, a presence sensor, and other types of sensors and actuators;

Expansion Pack: adds 16 common template sensors, 6 special template sensors, 5 actuators and 6 control switch types.

4.3.4 LwM2M standard objects

The LwM2M v1.0 standard defines 8 objects:

Security: handles security aspects between management servers and the LwM2M client on the device;

Server: provides data related to a LwM2M Servers the device is registered with;

Device: provides a range of device related information which can be queried by the LwM2M Server, including device reboot and factory reset function;

Firmware: enables management of firmware which is to be updated. This Object includes installing firmware package, updating firmware and performing actions after updating firmware;

Access control: supports the need to determine which operation on a given Object Instance is authorized for which LwM2M Server;

Location: provides information about the current location of an M2M device including latitude, longitude, altitude and velocity;

Connectivity monitoring: assists in monitoring the status of a network connection;

Connection statistics: holds statistical information about an existing network connection enabling clients to collect statistical information and enables the LwM2M Server to retrieve this information.

Chapter 5

System hardware and software

In **Chapter** 2 and **Chapter** 3 both IoT protocol stack and a generic FOTA architecture have been characterized in detail. In this chapter, referring explicitly to the FOTA method proposed in this thesis, all the relevant elements, both hardware and software, are listed and described in this chapter.

5.1 Devices description

5.1.1 STM32F401RE

The STM32F401RE device [26] is based on the high-performance ARM®Cortex® M4 32-bit RISC core operating at a frequency of up to 84 MHz.

It incorporates high-speed embedded memories (512 Kbytes of Flash memory, 96 Kbytes of SRAM). Up to 11 timers, two watchdog timers (independent and window) and a SysTick timer. STM32F401RE device offers one 12-bit ADC, a low-power RTC, six general-purpose 16-bit timers including one PWM timer for motor control, two general-purpose 32-bit timers. It also features standard and advanced communication interfaces including 81 I/O ports with interrupt capability, up to 3 x I2 C interfaces, up to 3 USARTs, up to 4 SPIs and USB 2.0 full-speed.

This device allows debug by means of serial wire debug (SWD) or JTAG interfaces, and offers power consumption features that allow itself to operate in four power modes varying consumption from 146 µA/MHz (Run mode with peripheral off) to Standby 2.4 µA (Standby mode).

5.1.2 STM32L476RG

The STM32L476RG device Figure 5.1 [28] is based on the ultra-low-power microcontroller based on the high-performance Arm® Cortex®-M4 32-bit RISC core operating at a frequency of up to 80 MHz.

It embeds high-speed memories (Flash memory up to 1 Mbyte, up to 128 Kbyte of SRAM) and also several protection mechanisms for embedded Flash memory and SRAM such as the memory protection unit (MPU) which enhances application security in order to guarantee readout protection, write protection, proprietary code readout protection and Firewall.

The device offers up to three fast 12-bit ADCs, two comparators, two operational amplifiers, two DAC channels, two general-purpose 32-bit timer, two 16-bit PWM timers dedicated to motor control, seven general-purpose 16-bit timers, a low-power RTC, and two 16-bit low-power timers.

It also features standard and advanced communication interfaces, including: USB OTG 2.0 full-speed, LPM and BCD, 2x SAIs (serial audio interface), IRTIM (Infrared interface), 1x LPUART (Stop 2 wake-up), 3x SPIs (and 1x Quad SPI), SWPMI single wire protocol master I/F, 3x I2C FM+(1 Mbit/s), SMBus/PMBus, 5x USARTs (ISO 7816, LIN, IrDA, modem), CAN (2.0B Active) and SDMMC interface.

The device offers development support by means of serial wire debug (SWD), JTAG, Embedded Trace Macrocell™ and includes a comprehensive set of power-saving modes, including Ultra-low-power with FlexPowerControl that ensure a power consumption between 39 µA/MHz (Run mode) to 30 nA (Shutdown mode).

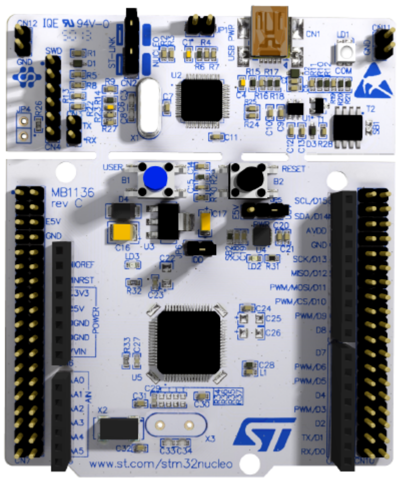


Figure 5.1. STM32L476RG nucleo board.

5.1.3 STM32F429ZI

The STM32F42ZI device Figure 5.2 [27] is based on the high-performance Arm Cortex-M4 32-bit RISC core operating at a frequency of up to 180 MHz.

It embeds high-speed embedded memories (Flash memory up to 2 Mbyte, up to 256 Kbytes of SRAM), up to 4 Kbytes of backup SRAM.

This device offers three 12-bit ADCs, two DACs, a low-power RTC, twelve general-purpose 16-bit timers including two PWM timers for motor control, two general-purpose 32-bit timers and an extensive range of enhanced I/Os and peripherals.

It also features standard and advanced communication interfaces, including up to three I2Cs, Six SPIs, two I2Ss full duplex (clocked via a dedicated internal audio PLL or via an external clock to allow synchronization), four USARTs plus four UARTs, an USB OTG full-speed and a USB OTG high-speed with full-speed capability, two CANs and the Ethernet RJ45 compliant with IEEE-802.3.

Furthermore, thanks to the ST Zio connector, which extends the Arduino Uno V3 connectivity, and the ST morpho headers, STM32F42ZI device provides an easy means of expanding the functionality of the Nucleo open development platform with a wide choice of specialized expansion boards.

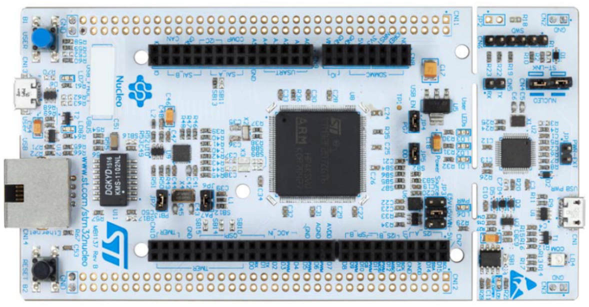


Figure 5.2. STM32F429ZI nucleo board.

5.1.4 X-NUCLEO-LPM01A

The X-NUCLEO-LPM01A device Figure 5.3 [32]is based on STM32L496VGT6 microcontroller featuring Arm Cortex-M4 core at 80 MHz and offers a 1.8 V to 3.3 V programmable power supply source with advanced power consumption measurement capability.

It performs consumption averaging (static measurement up to 200 mA) as well as real-time analysis (dynamic measurement up to 50 mA with 100 kHz bandwidth).

The X-NUCLEO-LPM01A operates either in standalone mode (using its LCD, joystick and button to display static measurements), or in controlled mode connected to host PC via USB (using a graphical user interface).

It can be used to supply and measure the consumption of STM32 Nucleo-32, Nucleo-64 or Nucleo-144 boards, using Arduino connectors. Alternatively, it supplies and measures the consumption of any target connected by wires via the basic connector. The device can be powered by USB micro-B (VBUS) and Arduino Uno and Nano connectors (pin 5 V).

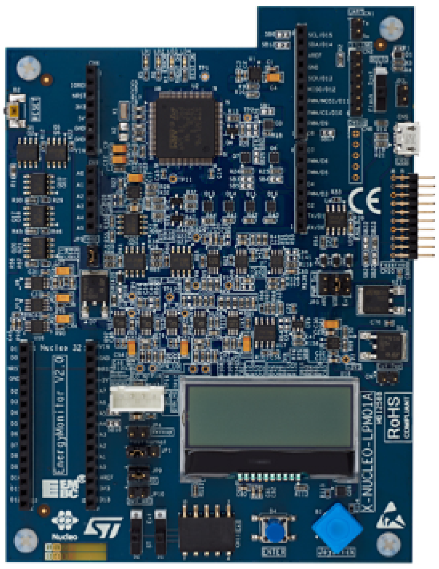


Figure 5.3. X-NUCLEO-LPM01A board.

5.2 Expansion board

5.2.1 SPIRIT1

The SPIRIT1 Figure 5.4 [22] is a very low-power RF transceiver, intended for RF wireless applications in the sub-1 GHz band. It is designed to operate both in the license-free ISM and SRD frequency bands at 169, 315, 433, 868, and 915 MHz, but can also be programmed to operate at other additional frequencies in the 300-348 MHz, 387-470 MHz, and 779-956 MHz bands.

The SPIRIT1 can be used in systems with channel spacing of 12.5/25 kHz, complying with the EN 300 220 standard and it supports different modulation schemes, including: 2-FSK, GFSK, OOK, ASK, and MSK.

It can operate as configurable baseband modem, which supports data management, modulation, and demodulation. Transmitted/received data bytes are buffered in two different three-level FIFOs, accessible via the SPI interface for host processing and its air data rate is programmable from 1 to 500 kbps.

The SPIRIT1 supports an embedded CSMA/CA engine and an AES 128-bit encryption co-processor is available for secure data transfer. Moreover, it provides an optional automatic acknowledgement, retransmission, and timeout protocol engine in order to reduce overall system costs by handling all the high-speed link layer operations.

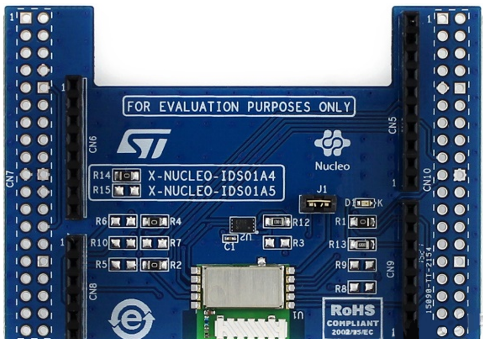


Figure 5.4. SPIRIT1 low-power RF transceiver.

5.3 Software

5.3.1 FP-SNS-6LPNODE1

FP-SNS-6LPNODE1 [21] is an STM32 ODE function pack which allows to connect an IoT node to a 6LoWPAN Wireless Sensor Network (WSN) and exposes the sensor and actuator resources using standard application layer protocols.

It also contains middleware libraries that can support a 6LoWPAN communication stack and permit to write applications based on a RESTful IoT node access using the OMA-LwM2M standard and CoAP over UDP along with the former data representation for the access on sensor and actuator resources.

The software includes drivers for the SPIRIT1 based sub-1GHz RF communication modules (SPSGRF-868 or SPSGRF-915), as well as the motion, environmental, and time-of-flight sensors.

The package is based on the STM32CubeHAL hardware abstraction layer and extends STM32Cube by providing a board support package (BSP) for the sub-1GHz RF communication expansion boards.

FP-SNS-6LPNODE1 software features:

- Middleware library with Contiki OS and Contiki 6LoWPAN protocol stack 3.x;
- Support for mesh networking technology by the means of the standard RPL protocol;
- Complete firmware to connect an IoT node with sensors and actuators to a 6LoWPAN network, using sub-1GHz RF communication technology.

The figure Figure 5.5 shows the overall FP-SNS-6LPNODE1 STM32 ODE function pack architecture.

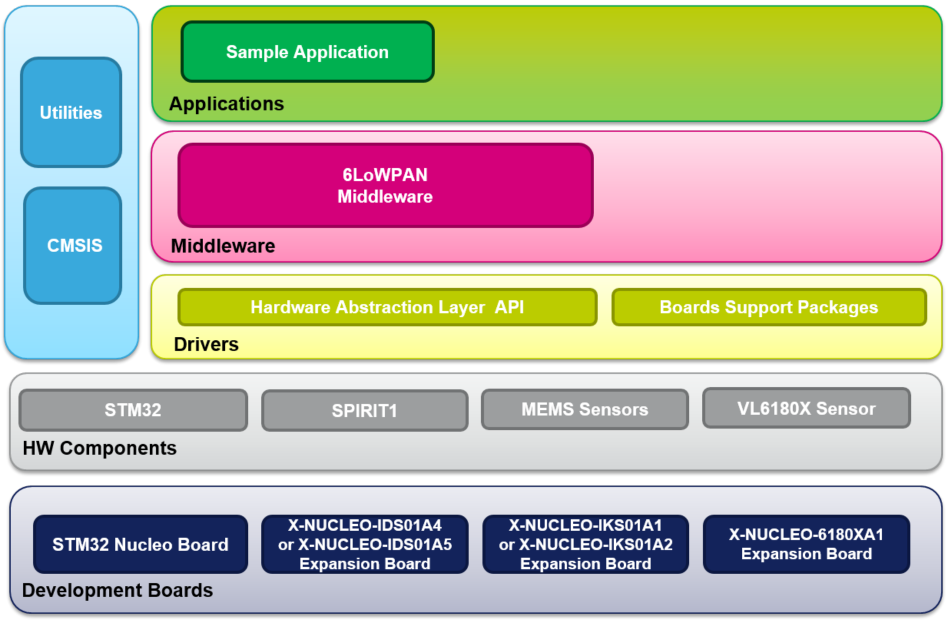


Figure 5.5. FP-SNS-6LPNODE1 STM32 ODE function pack layered architecture.

Contiki OS

Contiki OS [4] is an open source operating system for the Internet of Things that connects tiny low-cost and low-power microcontrollers to the Internet.

It is designed to run in small amounts of memory providing a set of mechanisms for memory allocation and it runs on a wide range of low-power wireless devices ensuring an easy and fast development process.

Contiki provides powerful low-power Internet communication with standard IP protocols such as IPv6, IPv4, UDP, TCP, and HTTP along with the recently standardized IETF protocols for low-power IPv6 networking including the 6LoWPAN, RPL and the CoAP RESTful application-layer protocol.

In addition, other important features are:

Coffee flash file system: a lightweight flash file system that allows application to operate on the external flash, without having to worry about flash sectors needing to be erased before writing or flash wear-leveling.(appendice)

Power Awareness:Power awareness Contiki provides mechanisms for estimating the system power consumption and for understanding where the power is spent to assist the development of low-power systems.

Sleepy router: relay nodes can be battery-operated thanks to the ContikiMAC radio duty cycling mechanism which allows them to sleep between each relayed message.

Protothreads: protothreads is a mixture of the event-driven and the multi-threaded programming mechanisms that guarantees to save memory and to provide a nice control flow in the code. [http://www.contiki-os.org/index.html]

5.3.2 Leshan

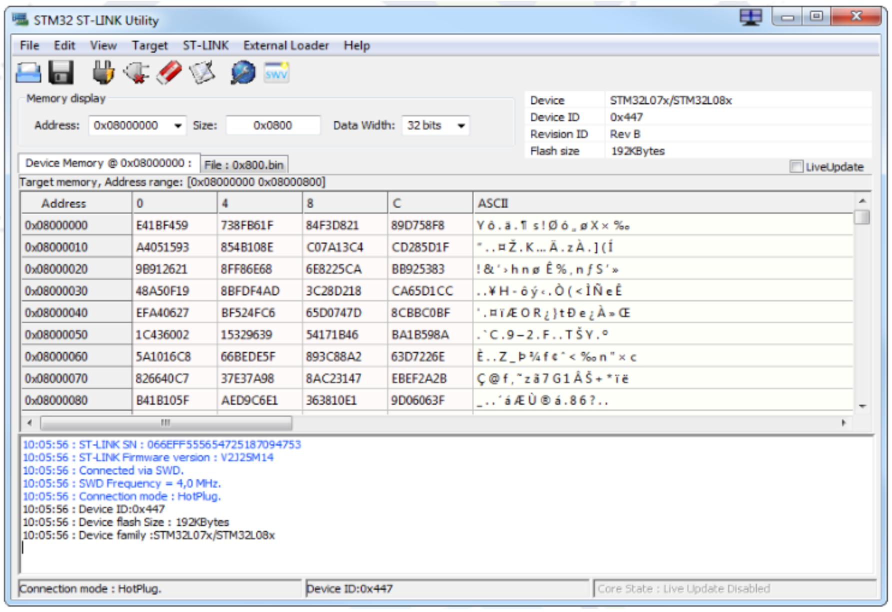
Eclipse Leshan is an OMA Lightweight M2M server and client Java implementation that provides a set of modular java libraries which help people to develop their own Lightweight M2M server and client. It is based on Californium CoAP and Scandium DTLS implementations and supports IPSO objects.[6]

5.3.3 STM32 ST-LINK Utility

STM32 ST-LINK Utility [24] is a free and full-featured software interface for programming STM32 microcontrollers.

It provides an easy-to-use and efficient environment for reading, writing, programming, verifying and erasing a memory device.

STM32 ST-LINK Utility supports Motorola S19, Intel HEX and binary formats and is delivered as a graphical user interface (GUI) Figure 5.6 with a command line interface (CLI). It also allows to load, edit and save executable generated by the Assembler/Linker or C compilers and to compare file with target memory.



Memory display

Address: 0x08000000 Size: 0x0800 Data Width: 32 bits

Device	STM32L07x/STM32L08x
Device ID	0x447
Revision ID	Rev B
Flash size	192KBytes

Device Memory @ 0x08000000 : File : 0x800.bin

Target memory, Address range: [0x08000000 0x08000800]

Address	0	4	8	C	ASCII
0x08000000	E41BF459	738FB61F	84F3D821	89D758F8	Y ô . ä . ¶ s ! Ø ó „ ø X × ‰
0x08000010	A4051593	854B108E	C07A13C4	CD285D1F	" . . ¤ Ž . K … Ä . z À .] (Í
0x08000020	9B912621	8FF86E68	6E8225CA	BB925383	! & ' › h n ø Ê % , n ƒ S ' »
0x08000030	48A50F19	8BFDF4AD	3C28D218	CA65D1CC	. . ¥ H - ô ý ‹ . Ò (< Ì Ñ e Ê
0x08000040	EFA40627	BF524FC6	65D0747D	8CBBC0BF	' . ¤ ï Æ O R ¿ } t Ð e ¿ À » Œ
0x08000050	1C436002	15329639	54171B46	BA1B598A	. ` C . 9 – 2 . F . . T Š Y . º
0x08000060	5A1016C8	66BEDE5F	893C88A2	63D7226E	È . . Z _ Þ ¾ f ¢ ˆ < ‰ n " × c
0x08000070	826640C7	37E37A98	8AC23147	EBEF2A2B	Ç @ f , ˜ z ã 7 G 1 Â Š + * ï ë
0x08000080	B41B105F	AED9C6E1	363810E1	9D06063F	_ . . ´ á Æ Ù ® á . 8 6 ? . .

```
10:05:56 : ST-LINK SN : 066EFF555654725187094753
10:05:56 : ST-LINK Firmware version : V2J25M14
10:05:56 : Connected via SWD.
10:05:56 : SWD Frequency = 4,0 MHz.
10:05:56 : Connection mode : HotPlug.
10:05:56 : Device ID:0x447
10:05:56 : Device flash Size : 192KBytes
10:05:56 : Device family :STM32L07x/STM32L08x
```

Connection mode : HotPlug. | Device ID:0x447 | Core State : Live Update Disabled

Figure 5.6. ST-Link graphical interface.

5.3.4 Californium

Californium (Cf) is an open source implementation of the Constrained Application Protocol (CoAP) and provides a convenient API for RESTful Web services that support all of CoAP's features like observe operations and blockwise transfers. It is written in Java and targets unconstrained environments such as back-end service infrastructures.[5]

The project is divided into five sub-projects:

Californium (Cf): is the core that provides the central framework with the protocol implementation.

Scandium (Sc): provides security for Californium implementing DTLS 1.2 to secure applications through ECC with pre-shared keys, certificates, or raw public keys.

Actinium (Ac): is the app-server for Californium. JavaScript apps become available as RESTful resources and can directly talk to IoT devices using our CoAPRequest object API.

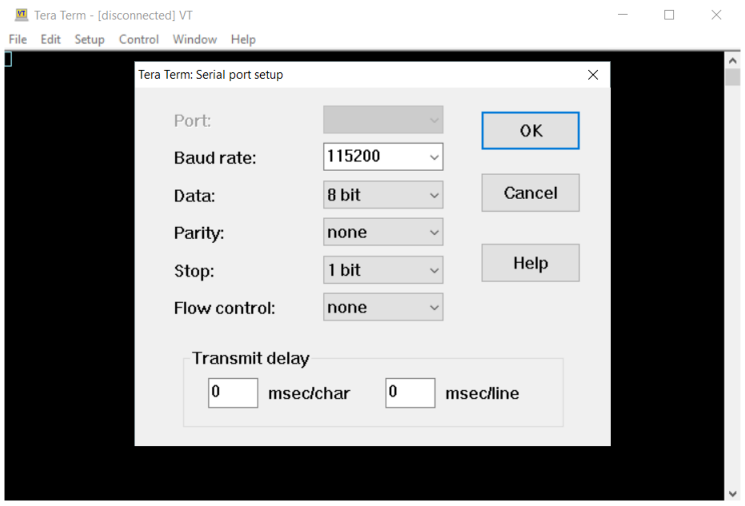
CoAP tools: provides CoAP tools like CoAPBench for benchmarking or the cf-client to interact with devices from the command line.

Connector: the element-connector abstracts from the different transports CoAP can use.

Californium (Cf) focuses on service scalability for large-scale Internet of Things applications and thanks to CoAP's low overhead, it allows to handle millions of IoT devices with a single service instance

5.3.5 Tera Term

Tera Term Figure 5.7 [33] is an open-source, free, software terminal emulator (communications) program. It emulates different types of computer terminals, supports different TCP/IP connections including telnet, SSH1, SSH2, serial port connections, Ipv6 communication, a proprietary script language called Tera Term Language and many file transfer protocols such as Kermit, XMODEM and Quick-VAN. Also it provides multiple languages and UTF-8 character sets.



Port:
Baud rate: 115200
Data: 8 bit
Parity: none
Stop: 1 bit
Flow control: none
Transmit delay
0 msec/char 0 msec/line

Figure 5.7. Tera Term graphic interface.

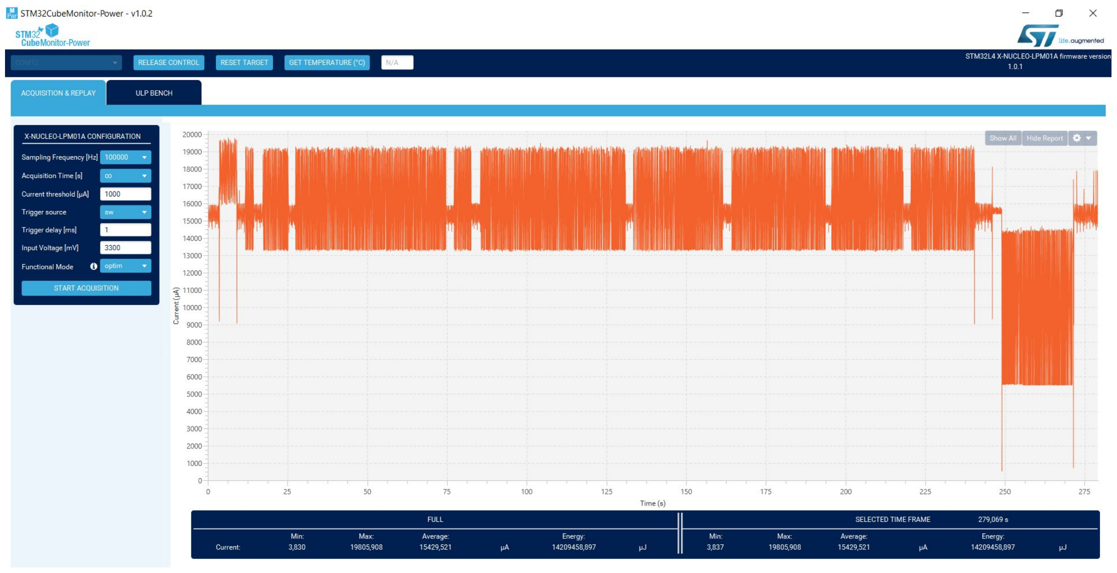
5.3.6 STM32CubeMonPwr

STM32CubeMonitor-Power is a multi platforms tool that allows to acquire power measurements through the X-NUCLEO-LPM01A specialized intermediate board in order to analyze the low-power performance of target boards.[25]

Dynamic measurement of current covers a range from 100 nA to 50 mA at up to 100 kHz with 2% accuracy, while STM32CubeMonitor-Power allows updating of acquisition parameters, and data rendering in real time.

Thanks to an intuitive graphical interface with mouse-based data navigation Figure 5.8, this software can display measurements and control all X-NUCLEO-LPM01A functions such as acquisition frequency, supply voltage, triggers, and others.

It also has the capability to compute the consumed energy, save acquired data and to load previously saved data.



X-NUCLEO-LPM01A CONFIGURATION

Sampling Frequency [Hz]	100000
Acquisition Time [s]	∞
Current threshold [µA]	1000
Trigger source	sw
Trigger delay [ms]	1
Input Voltage [mV]	3300
Functional Mode	optim

	FULL						SELECTED TIME FRAME 279,069 s					
	Min:	Max:	Average:		Energy:		Min:	Max:	Average:		Energy:	
Current:	3,830	19805,908	15429,521	µA	14209458,897	µJ	3,837	19805,908	15429,521	µA	14209458,897	µJ

Figure 5.8. STM32CubeMonPwr graphic interface.

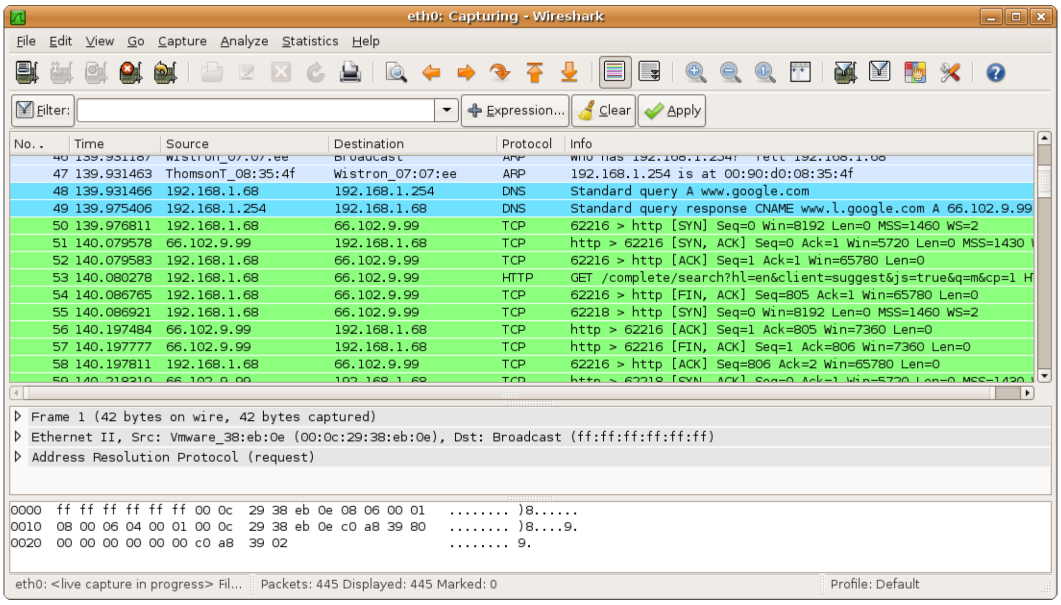
5.3.7 Wireshark

Wireshark is a free and open source packet analyzer able to "understands" the structure of different networking protocols exploiting pcap API.[37]

It is used mainly for network troubleshooting, analysis, software and communications protocol development, and education. Wireshark has a rich feature set which includes the following:

- Live capture and offline analysis;
- Runs on Windows, Linux, macOS, Solaris, FreeBSD and NetBSD;
- Wireshark has a graphical front-end Figure 5.9, plus some integrated sorting and filtering options ;

- Captured network data can be browsed via a GUI, or via the TTY-mode TShark utility;
- Capture files compressed with gzip can be decompressed on the fly;
- Live data can be read from Ethernet, IEEE 802.11, Bluetooth, USB and many others (depending on the platform);
- Colouring rules can be applied to the packet list for quick, intuitive analysis;
- Output can be exported to XML, PostScript®, CSV, or plain text;
- Plug-ins can be created for dissecting new protocols.


eth0: Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression...
Clear
Apply

```
No. .  Time        Source             Destination       Protocol  Info
    46 139.931187  Wistron_07:07:ee   Broadcast         ARP       Who has 192.168.1.254?  Tell 192.168.1.68
    47 139.931463  ThomsonT_08:35:4f  Wistron_07:07:ee  ARP       192.168.1.254 is at 00:90:d0:08:35:4f
    48 139.931466  192.168.1.68       192.168.1.254     DNS       Standard query A www.google.com
    49 139.975406  192.168.1.254      192.168.1.68      DNS       Standard query response CNAME www.l.google.com A 66.102.9.99
    50 139.976811  192.168.1.68       66.102.9.99       TCP       62216 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
    51 140.079578  66.102.9.99        192.168.1.68      TCP       http > 62216 [SYN, ACK] Seq=0 Ack=1 Win=5720 Len=0 MSS=1430
    52 140.079583  192.168.1.68       66.102.9.99       TCP       62216 > http [ACK] Seq=1 Ack=1 Win=65780 Len=0
    53 140.080278  192.168.1.68       66.102.9.99       HTTP      GET /complete/search?hl=en&client=suggest&js=true&q=m&cp=1 H
    54 140.086765  192.168.1.68       66.102.9.99       TCP       62216 > http [FIN, ACK] Seq=805 Ack=1 Win=65780 Len=0
    55 140.086921  192.168.1.68       66.102.9.99       TCP       62218 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=2
    56 140.197484  66.102.9.99        192.168.1.68      TCP       http > 62216 [ACK] Seq=1 Ack=805 Win=7360 Len=0
    57 140.197777  66.102.9.99        192.168.1.68      TCP       http > 62216 [FIN, ACK] Seq=1 Ack=806 Win=7360 Len=0
    58 140.197811  192.168.1.68       66.102.9.99       TCP       62216 > http [ACK] Seq=806 Ack=2 Win=65780 Len=0
    59 140.218319  66.102.9.99        192.168.1.68      TCP       http > 62218 [SYN, ACK] Seq=0 Ack=1 Win=5720 Len=0 MSS=1430

▷ Frame 1 (42 bytes on wire, 42 bytes captured)
▷ Ethernet II, Src: Vmware_38:eb:0e (00:0c:29:38:eb:0e), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▷ Address Resolution Protocol (request)

0000  ff ff ff ff ff ff 00 0c  29 38 eb 0e 08 06 00 01   ........ )8......
0010  08 00 06 04 00 01 00 0c  29 38 eb 0e c0 a8 39 80   ........ )8....9.
0020  00 00 00 00 00 00 c0 a8  39 02                     ........ 9.

eth0: <live capture in progress> Fil...   Packets: 445 Displayed: 445 Marked: 0        Profile: Default
```

Figure 5.9. Wireshark graphic interface.

5.3.8 System Workbench toolchain

The System Workbench toolchain [30], is a free multi-OS software development environment created by AC6 which supports the full range of STM32 microcontrollers and associated boards.

It is based on Eclipse thus it is compatible with Eclipse plug-ins and supports multiple OS support, including Windows, Linux and OSX.

Moreover, the System Workbench toolchain has no code size limit and supports ST-LINK utility, GCC C/C++ compiler and GDB-based debugger.

Chapter 6

System implementation

The FOTA method implementation refers to the ST hardware and software systems. It implements the specific protocol stack illustrated in the Figure 6.1 where the OMA-LwM2M standard is located as Device Management over the CoAP internet application protocol and the 6LoWPAN protocol occupies the network layer to enabling M2M communication exploiting IPv6 internet protocol.

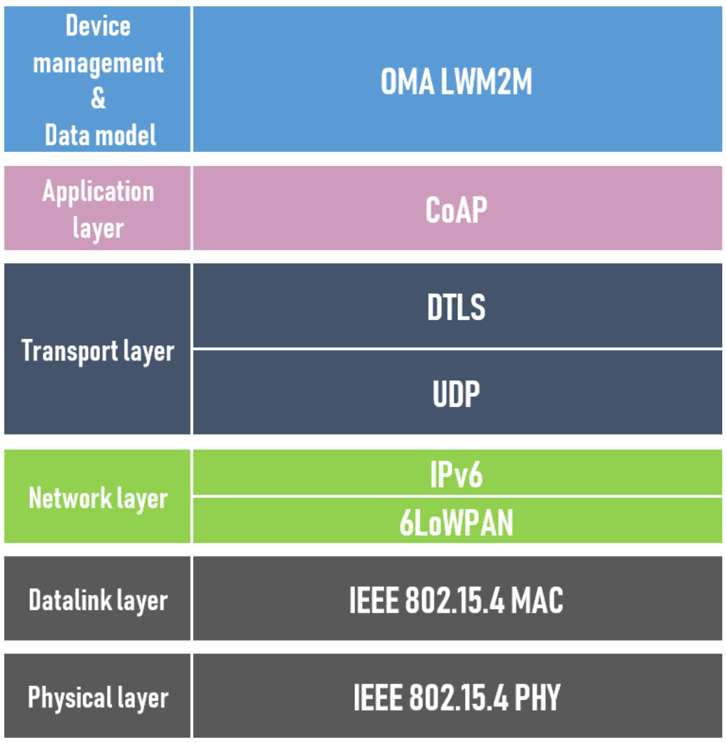


Figure 6.1. Proposed FOTA method protocol stack.

This FOTA method has been developed for two different devices: the STM32F401RE and STM32L476RG boards.

The implementation process has been divided into two phases.

Since the FP-SNS-6LPNODE1 function pack supports the STM32F401RE, the first phase has been to implement the FOTA method for this board in order to perform a stable firmware update that complies with the OMA LWM2M standard.

The second phase has been to make the porting of the FP-SNS-6LPNODE1 function pack on the STM32L476RG boards in order to develop the dual boot along with a reliable recovery mechanism starting from a stable firmware update.

The FOTA method supports the remote firmware update both using the Push and Pull transfer modes. In particular, two different servers have been used to deliver the firmware image to the client: Leshan server and Californium file server.

The Leshan one supports the OMA-LwM2M standard and the CoAP application protocol so it has been used for managing and delivering the firmware in addition to the enabling of security since the FOTA method supports DTLS. But apart from that, Leshan can not be used as a file server, so for this reason it has been necessary to adopt a second server that can behave as one of that, which is the Californium.

For both these servers some parameters have been modified in order to comply with the protocol stack requirements such as MAX_MESSAGE_SIZE, BLOCK_SIZE, ports and security parameters.

Moreover, the FOTA architecture adopts an Ethernet type router connection because even if the FOTA implementation works correctly also with WiFi and Border router, the Ethernet one offers a better and more stable performance. So in the end, the final FOTA method architecture is shown in Figure 6.2.

The implementation of this FOTA method required several steps, that are discussed hereafter and can be summarized in:

1. Definition of the OMA-LwM2M Firmware Object and implementation of the Update State Machine;
2. Implementation of the block-wise resource level code;
3. Implementation of the Push and Pull transfer modes;
4. Implementation of the Update process for both STM32F401RE and STM32L476RG devices;
5. Implementation of the recovery mechanism.

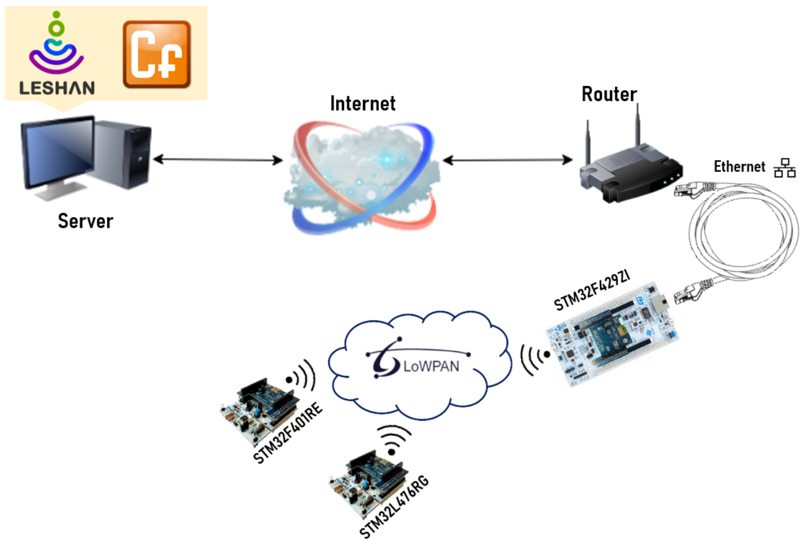


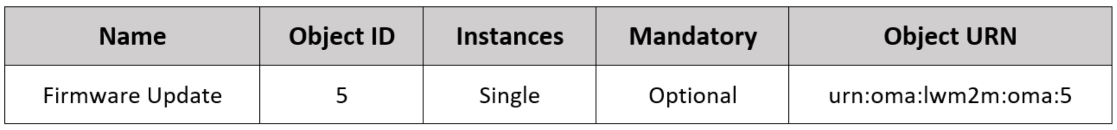
Figure 6.2. Proposed FOTA method architecture.

6.1 OMA-LwM2M Firmware Object

The LwM2M Firmware Object [1] enables management of the firmware which is to be updated allowing a LwM2M Client to connect to any LwM2M Server. This Object includes installing firmware package, updating firmware, and performing actions after updating firmware.

6.1.1 Object Firmware definition

Starting from the FP-SNS-6LPNODE1 function pack, in order to define the OMA LwM2M Firmware object, the file lwm2m-firmware.c has been added into the Contiki oma-lwm2m, while the Object itself has been defined into the OMA-LwM2M engine.h file, according to the parameters defined by the standard as shown in th Figure 6.3.

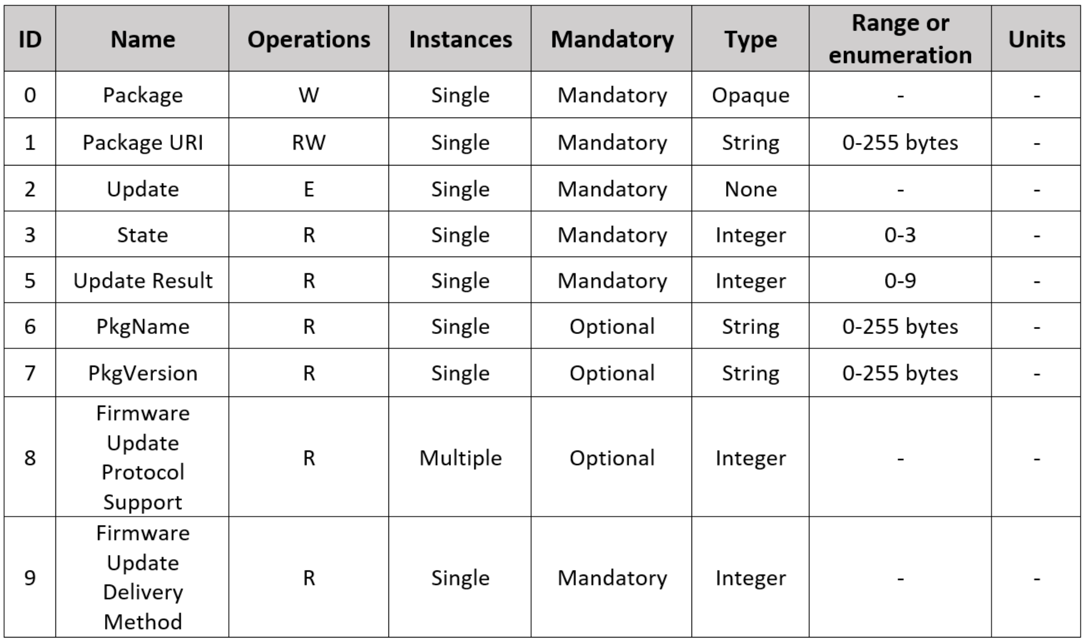


Name	Object ID	Instances	Mandatory	Object URN
Firmware Update	5	Single	Optional	urn:oma:lwm2m:oma:5

Figure 6.3. OMA-LwM2M Firmware Object.

6.1.2 Firmware Resources definition

After the object definition, according to the application requirements, all the resources provided by the standard Figure 6.4, have been defined and implemented in the lwm2m-firmware.c file.



ID	Name	Operations	Instances	Mandatory	Type	Range or enumeration	Units
0	Package	W	Single	Mandatory	Opaque	-	-
1	Package URI	RW	Single	Mandatory	String	0-255 bytes	-
2	Update	E	Single	Mandatory	None	-	-
3	State	R	Single	Mandatory	Integer	0-3	-
5	Update Result	R	Single	Mandatory	Integer	0-9	-
6	PkgName	R	Single	Optional	String	0-255 bytes	-
7	PkgVersion	R	Single	Optional	String	0-255 bytes	-
8	Firmware Update Protocol Support	R	Multiple	Optional	Integer	-	-
9	Firmware Update Delivery Method	R	Single	Mandatory	Integer	-	-

Figure 6.4. OMA-LwM2M Firmware Resources.

Package: is used to enable Push method firmware transfer where the server sends the firmware image to the client;

Package URI: enables Pull method firmware transfer where the client fetches the firmware image from a file server specified in the URI by the node;

Update: starts the update process using the firmware image already stored in the flash;

State: indicates the current state with respect to this firmware update;

Update Result: contains the result of the operation plus the report of any error generated by the downloading or updating process;

PkgName: indicates the name of the Firmware Package;

PkgVersion: indicates the version of the Firmware package;

Firmware Update Protocol Support: indicates what protocols the LwM2M Client implements to retrieve firmware images;

Firmware Update Delivery Method: indicates the supported transferring method for firmware images. This includes Push, Pull or both.

According to this FOTA method, both Push and Pull transfer modes are supported.

6.1.3 Firmware update state machine

After the object and resources definition, the second step has been to implement the firmware update mechanism proposed by the standard. The state diagram in Figure 6.5 shows which are the 4 states in which the process consists:

IDLE: indicates the starting state of the state diagram. The state is reachable before downloading or after a successful updating;

DOWNLOADING: indicates that the firmware transfer between client node and server is started but not yet concluded;

DOWNLOADED: indicates that the downloading is concluded and the firmware image is stored in the flash memory;

UPDATING: indicates the operation of replacing the old firmware image with the updated one and then rebooting.

The state diagram uses two variable represented by the STATE and UPDATE RESULT object Resources. The former indicates the current state by number that goes from 0 to 4, while the latter is used to report results and possible errors that can be met during the entire firmware update process.

In this FOTA method, the UPDATE RESULT Resource can assume values from 0 to 9, that are related to specific meanings such as:

0: initial value. Once the updating process is initiated this Resource is reset to this initial value;

1: firmware updated successfully;

2: not enough flash memory for the new firmware package;

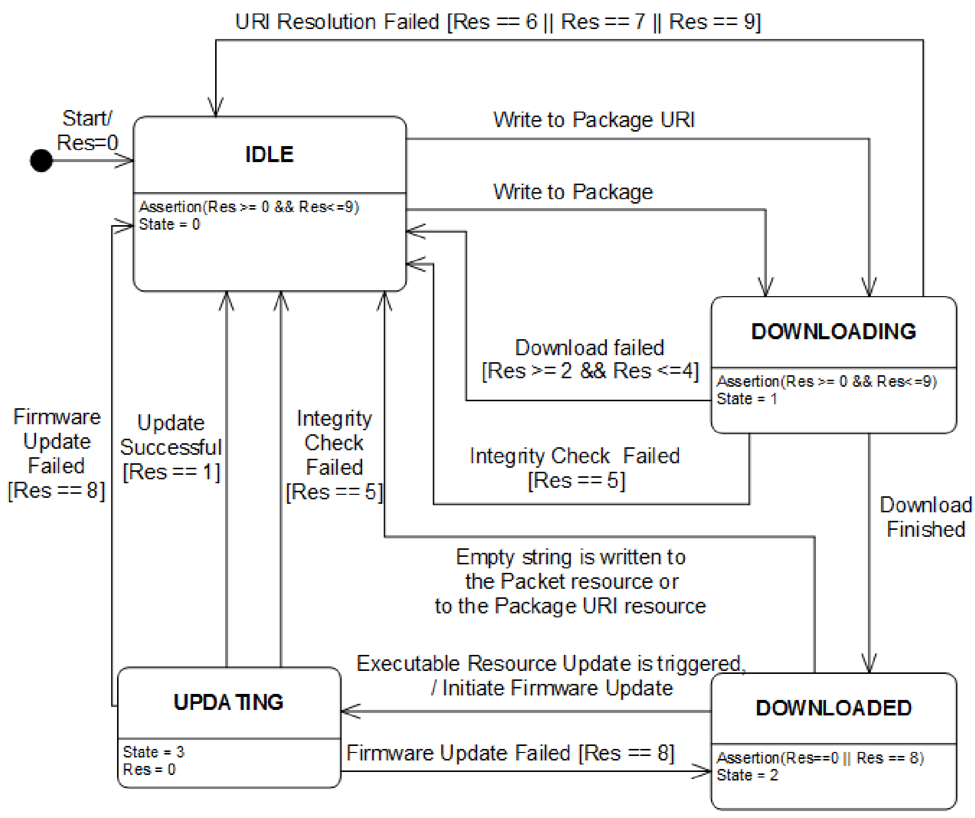

URI Resolution Failed [Res == 6 || Res == 7 || Res == 9]
Start/ Res=0
IDLE
Assertion(Res >= 0 && Res<=9)
State = 0
Write to Package URI
Write to Package
DOWNLOADING
Assertion(Res >= 0 && Res<=9)
State = 1
Download failed [Res >= 2 && Res <=4]
Integrity Check Failed [Res == 5]
Download Finished
Firmware Update Failed [Res == 8]
Update Successful [Res == 1]
Integrity Check Failed [Res == 5]
Empty string is written to the Packet resource or to the Package URI resource
Executable Resource Update is triggered, / Initiate Firmware Update
UPDATING
State = 3
Res = 0
Firmware Update Failed [Res == 8]
DOWNLOADED
Assertion(Res==0 || Res == 8)
State = 2

Figure 6.5. OMA-LwM2M Firmware update state machine.

4: connection lost during downloading process;

5: integrity check failure for new downloaded package;

7: invalid URI;

8: firmware update failed;

9: unsupported protocol.

Since the firmware mechanism depends on the STATE and UPDATE RESULT values, it is necessary to store these data in the flash memory in order to preserve the correct information after a firmware update, reboot or recovery processes.

6.2 Block-wise transfer

Since the FP-SNS-6LPNODE1 function pack integrates Contiki that uses CoAP as transfer protocol, it has been possible to use block-wise transfer options to enable the firmware transfer in both Push and Pull mode. By the reason that CoAP specifies block-wise transfer, but this was not yet supported in the LwM2M Resource model implementation of Contiki, the third step has been to add the block-wise resource level code. At this scope, the *lwm2m_resource* structure has been modified in order to add a new type of LwM2M Resource format writer called LWM2M_RESOURCE_TYPE_CALLBACK_BW1 that aims to handle the Push block-wise transfer mode.

With the block-wise transfer the firmware that has to be delivered is first divided in blocks with a selectable size that can be set as a power of two and then sent one by one.

Since this FOTA method relies on the 802.15.4 physical layer, the maximum payload dimension allowed is 96 Bytes, so to avoid fragmentation, the block size has been fixed at 64 Bytes in since it is the maximum power of two that fits into the 802.15.4 payload. To meet this requirement, both the properties of Leshan and Californium servers have been modified.

In order to send them, the server encapsulates every block in a CoAP message where each of that contains, in addition to the payload of the firmware, the CoAP Block parameters that are:

Block_num: indicates the relative number of the block within a sequence of blocks with a given size;

Block_more: indicates if more blocks are following or if it is the last one transmitted;

Block_size: indicates the size of the block.

These parameters refer both to the Block1 and Block2 options and can be used to manage the firmware transfer between Server and Client. In addition to that, they are used to validate the package received, by checking informations such as knowing if the firmware has the right size to fit into the flash memory, if there have been retransmissions or if a packet loss has happened, letting us understand if the firmware file is corrupted or not after it has been received.

Moreover, during the download phase, unexpected events such as a timeout of connection or a bad request form server are handled by the Push and Pull mode algorithms, that immediately report the eventual error using the UPDATE RESULT Resource.

6.2.1 Push transfer mode

Push transfer mode refers to the Package Resource of the OMA-LwM2M standard. In this scenario, the server starts the transfer of the firmware image packet by packet.

Thanks to the LWM2M_RESOURCE_TYPE_CALLBACK_BW1 format writer, it is possible to handle the PUT server requests using the Block1 option.

An example of the Push mode message exchange is represented in the Figure 6.6.

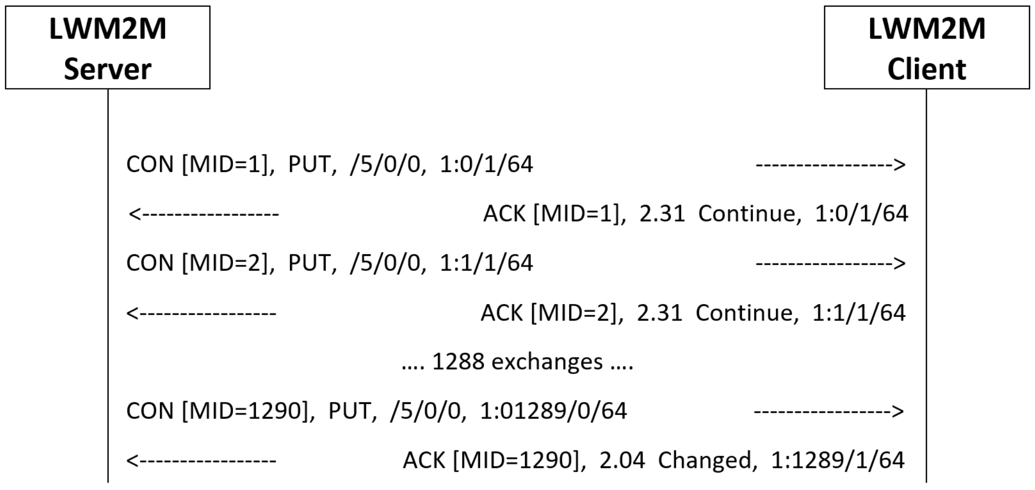

LWM2M Server
LWM2M Client
CON [MID=1], PUT, /5/0/0, 1:0/1/64 ---------------->
<----------------- ACK [MID=1], 2.31 Continue, 1:0/1/64
CON [MID=2], PUT, /5/0/0, 1:1/1/64 ---------------->
<----------------- ACK [MID=2], 2.31 Continue, 1:1/1/64
…. 1288 exchanges ….
CON [MID=1290], PUT, /5/0/0, 1:01289/0/64 ---------------->
<----------------- ACK [MID=1290], 2.04 Changed, 1:1289/1/64

Figure 6.6. Example of a LwM2M Push transer mode messages exchange.

In this case, when the server starts to send the firmware, a PUT request from the server, with the Block1 option, arrives to the client, the Contiki LwM2M engine recognizes that is a block-wise transfer and calls the Package resource which is able to handle the Block1 option PUT request using the *Package* function.

This function implements the mechanism that allows to manage the Push transfer mode which receives packets from the server and after validating them, it checks the Block1_more flag that indicates whether there are more packets to send or not, in order to understand when the transfer has to be considered concluded and set the STATE Resource according to the transfer process.

The Block1_more flag indicates indeed if more blocks are following in the communication and still need to be received, or not, if true, an Acknowledgment message with a Continue response code is sent to the server. When the last packet arrives, the Block1_more flag shows that no more blocks are following so the client answers with the changed response to stop the transfer.

The Push transfer mode mechanism is described in Figure 6.7.

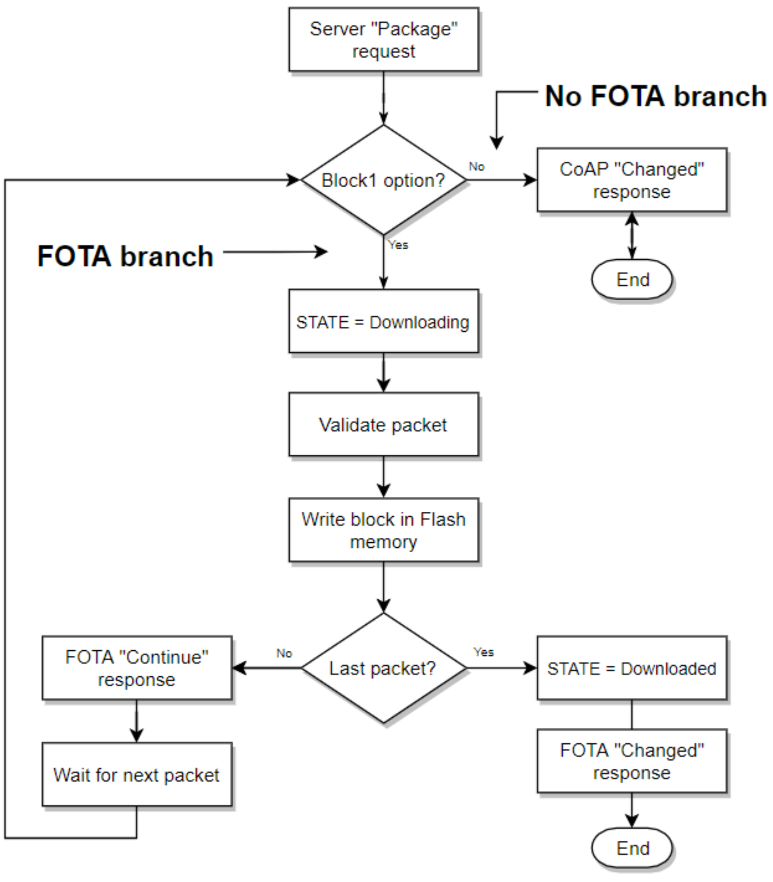


Figure 6.7. Push transfer mode flow chart.

Since the public server Leshan does not allow to send files using the interface, because of the UI limitation, has been necessary to use the Leshan REST API in order to deliver the firmware to the client. To do so the script Bin2Json has been created with the aim to convert the firmware image in an opaque format and then use the cURL tool to send it as a JSON data format. The final cURL command used is:

Curl -T $json_{f}ile_{p}ath - H"Content - Type : application/json" - XPUThttp : //localhost : 8080/api/clients/$client_name/5/0/0?format=Opaque

The Bin2Json script sequence chart is shown in Figure 6.8.

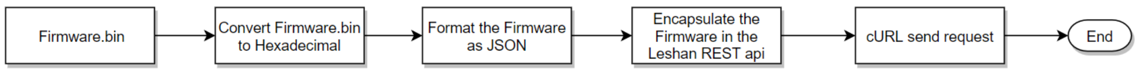

Firmware.bin
Convert Firmware.bin to Hexadecimal
Format the Firmware as JSON
Encapsulate the Firmware in the Leshan REST api
cURL send request
End

Figure 6.8. Bin2Json script sequence chart.

Leshan though, does not recognize, the OPAQUE data type format on which the OMA-LwM2M standard is based, so we had the need to implement a local Leshan server version where some modifications have been made in order to support that particular format so as to enable the push method transfer.

6.2.2 Pull transfer mode

Pull transfer mode refers to the Package URI Resource of the OMA-LwM2M standard. In this case the client fetches the firmware image directly from a file server. Before the transfer begins, the LwM2M server communicates the firmware URI to the client in order to let the him fetches the packages from the file server indicated in the firmware URI.

An example of the Pull transfer mode message exchange is shown in the Figure 6.9.

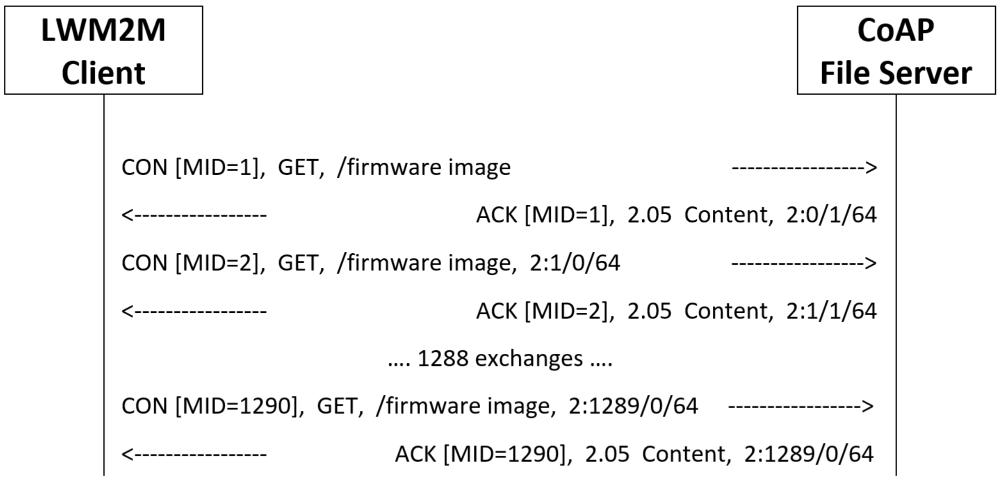
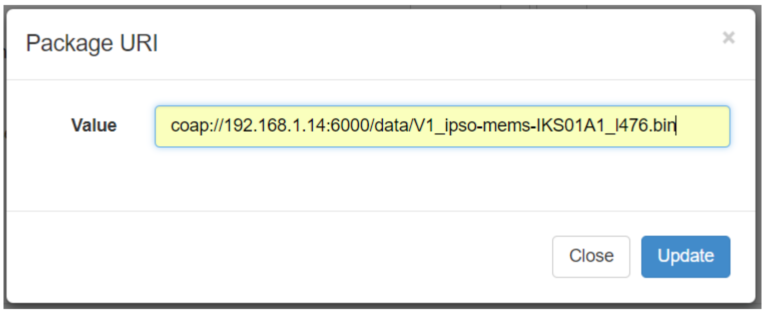

LWM2M Client
CoAP File Server
CON [MID=1], GET, /firmware image ----------------->
<----------------- ACK [MID=1], 2.05 Content, 2:0/1/64
CON [MID=2], GET, /firmware image, 2:1/0/64 ----------------->
<----------------- ACK [MID=2], 2.05 Content, 2:1/1/64
…. 1288 exchanges ….
CON [MID=1290], GET, /firmware image, 2:1289/0/64 ----------------->
<----------------- ACK [MID=1290], 2.05 Content, 2:1289/0/64

Figure 6.9. Example of a LwM2M Pull transer mode messages exchange.

In this case Californium has been used as file server, while Leshan server has been used to send the URI of the file server to the client as shown in Figure 6.10.

To handle the Pull transfer mode mechanism, the Package URI Resource calls the *package_uri_write* function.

This function, when a firmware URI arrives, first parses it in order to obtain and validate the protocol scheme, the host name, the port and the path needed to issue the request message to the file server. After the parsing operation, the function analyses the host name in order to define the IP address of the file server. It can indeed understand if the host name contains or not a domain name and in that case it can resolve it by contacting the DNS.


Package URI

Value: coap://192.168.1.14:6000/data/V1_ipso-mems-IKS01A1_l476.bin

Figure 6.10. Leshan Package URI GUI with an example of CoAP firmware URI.

When a valid address has been obtained, the client starts to request for packets to the file server and the latter responds with messages containing Block2 option.

The *package_uri_write* function receives packets from the server and after validating them it checks the Block2_more flag that indicates whether there are more packets to send or not, in order to understand when the transfer has to be considered concluded and set the STATE Resource according to the transfer process.

The Block2_more flag, as the Block1_more flag indicates if more blocks are following in the communication and still need to be fetched or not, if true, an Acknowledgment message with a continue response is sent to the file server. When the last packet arrives the client answers with the changed response to stop the transfer.

The Pull transfer mode mechanism is described in the Figure 6.11.

6.3 Update process

The update process, independently from the specific mechanism that is used and the device in which is executed, has the general aim to replace the old firmware with new one. This operation is performed by the bootloader that accomplishes the procedure according to the information related to the download process that have been stored by the application running on the device before the starting of the update process.

The information mentioned before, which are used in the communication between the two main actors of this process, take the name of METADATA for this particular FOTA method and are stored in specific addresses of the flash memory, being accessible by both the bootloader and the firmware that is currently running. They are used in order to perform different actions, such as the validation of the firmware or the recovery of the system, and are represented by: STATE Resource, UPDATE RESULT Resource and MAGIC WORD. In particular, the latter

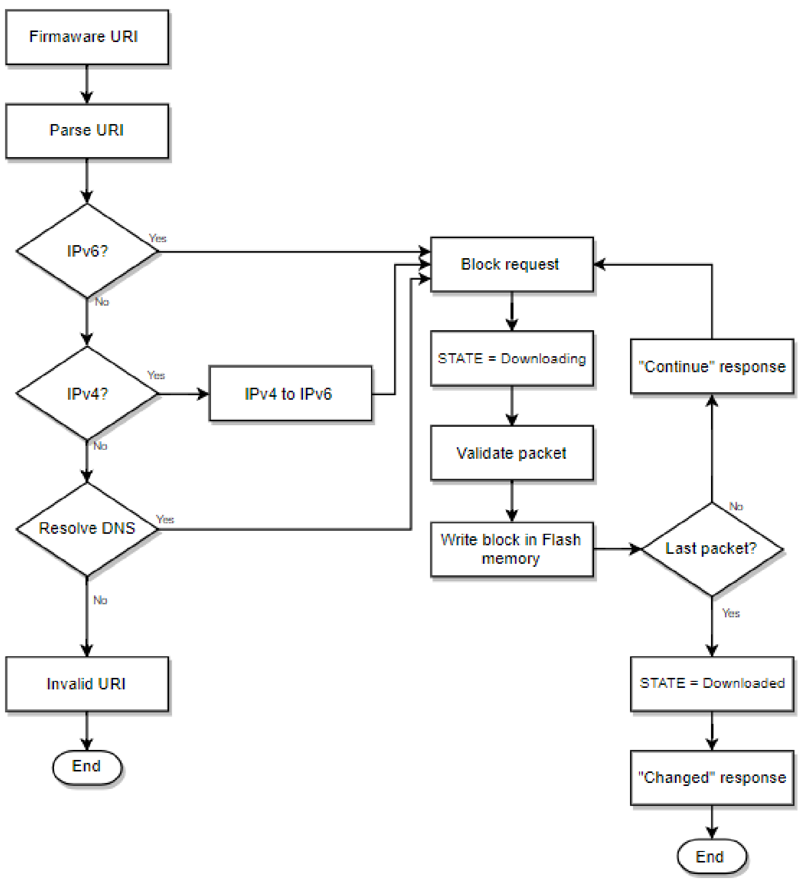


Figure 6.11. Pull transfer mode flow chart.

is a safeguard (known) hexadecimal sequence that is used to guarantee that the following data actually contains FSM metadata and the new firmware.

6.3.1 STM32F401RE update process

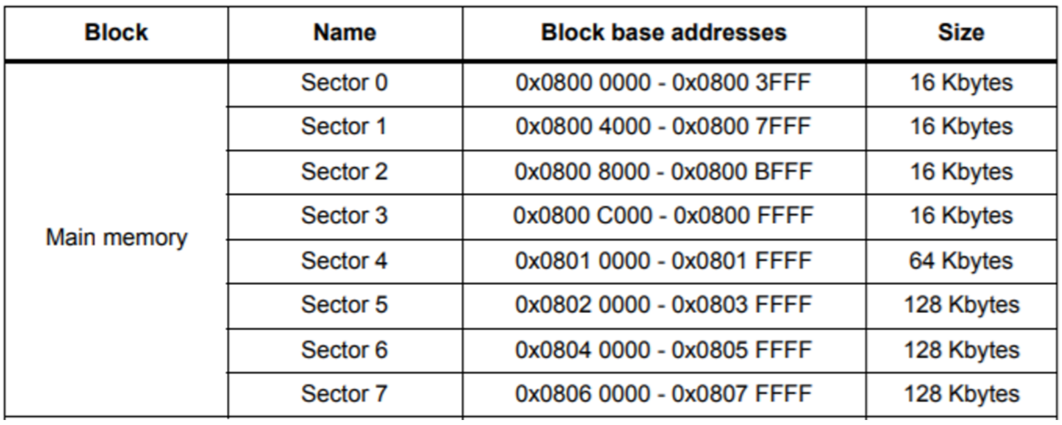
The STM32F401RE update phase consists in three main steps:

1. Checking if the download process was successful or not;
2. Replacing the old firmware image with the new one after the validation;
3. Booting the new firmware image.

In order to accomplish the STM32F401RE update process, due to the various elements required by the FOTA method, in which the bootloader, the metadata and the firmware itself have to be stored in the flash memory, it has been necessary to perform a preliminary mapping of the latter.

Flash memory mapping

The STM32F401RE has a 512 KB memory flash divided into 7 sectors as shown in the figure Figure 6.12.



Block	Name	Block base addresses	Size
Main memory	Sector 0	0x0800 0000 - 0x0800 3FFF	16 Kbytes
	Sector 1	0x0800 4000 - 0x0800 7FFF	16 Kbytes
	Sector 2	0x0800 8000 - 0x0800 BFFF	16 Kbytes
	Sector 3	0x0800 C000 - 0x0800 FFFF	16 Kbytes
	Sector 4	0x0801 0000 - 0x0801 FFFF	64 Kbytes
	Sector 5	0x0802 0000 - 0x0803 FFFF	128 Kbytes
	Sector 6	0x0804 0000 - 0x0805 FFFF	128 Kbytes
	Sector 7	0x0806 0000 - 0x0807 FFFF	128 Kbytes

Figure 6.12. STM32F401RE flash memory sectors partition.

Since STM32F401RE memory partitioning is not so fine grained, and because the flash technology forces to erase and write sector by sector, only a minimal firmware update mechanism has been implemented.

The STM32F401RE flash memory mapping is shown in Figure 6.13.

STM32F401RE bootloader

To manage the update process, the STM32CubeFunctionPack_AZURE1_V3.2.2 bootloader [20], after adapting it for the STM32F401RE device, has been used. It represents a simplified version that recognises if a new firmware is present or not in the flash memory and if yes, it erases the first firmware image located in the Region 3, copies the new one from Region 4 to Region 3 and finally boot the system.

The STM32F401RE update mechanism starts when an Update request arrives from the server and it consists in checking whether the STATE Resource value is equal to "Downloaded", which means that the downloading of the new firmware has been completed. Following this step, the current running firmware sets the STATE Resource equal to "Update" and then, before restarting the system, the metadata is stored into the Region 2 of the memory flash and the MAGIC WORD is added to new firmware image.

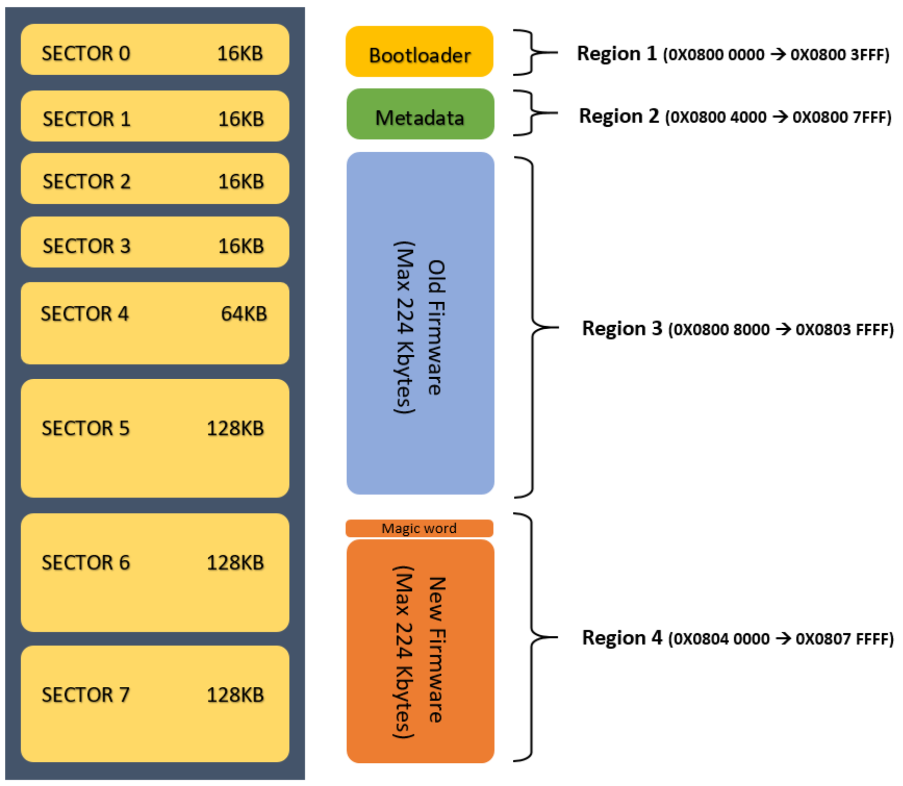

SECTOR 0 16KB
SECTOR 1 16KB
SECTOR 2 16KB
SECTOR 3 16KB
SECTOR 4 64KB
SECTOR 5 128KB
SECTOR 6 128KB
SECTOR 7 128KB
Bootloader
Metadata
Old Firmware (Max 224 Kbytes)
Magic word
New Firmware (Max 224 Kbytes)
Region 1 (0X0800 0000 → 0X0800 3FFF)
Region 2 (0X0800 4000 → 0X0800 7FFF)
Region 3 (0X0800 8000 → 0X0803 FFFF)
Region 4 (0X0804 0000 → 0X0807 FFFF)

Figure 6.13. STM32F401RE flash memory mapping.

After that, when the system restarts, the bootloader runs first and checks if the MAGIC WORD, added to the new firmware and stored in the Region 4 of the flash memory, is present, in order to validate the firmware and begin the replacement process.

Following the validation, if the new firmware can run, the bootloader:

1. Erases Region 3;
2. Copies the new firmware stored in Region 4 into Region 3;
3. Runs the new firmware.

The new firmware, during its initialization stage, validates the metadata and updates the STATE Resource equal to *Idle* and UPDATE RESULT Resource equal to *Successful*.

The STM32F401RE updating process is shown in the Figure 6.14.

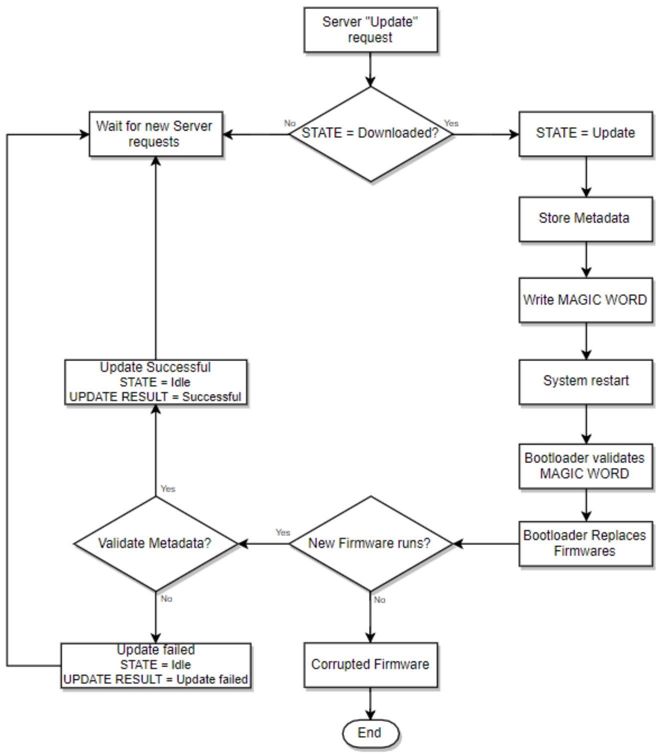


Figure 6.14. STM32F401RE update mechanism flow chart.

6.3.2 STM32L476RG update process

STM32L476RG unlike STM32F401RE has a bigger flash memory with a fine grain partitioning, thus it offers the possibility of implementing more complex update mechanism. In the proposed FOTA method, in particular, the STM32L476RG guaranties a dual boot and recovery mechanism that aims to restore the old firmware in case of failure during the download or update phases.

The STM32L476RG update phase consists in four main steps:

1. Checking if the download process was successful or not;
2. Validating the new firmware and swapping the two firmware images;
3. Booting the new firmware;
4. Checking if the new firmware runs correctly or if a recovery operation is required.

As it has been made for the STM32F401RE board, in order to accomplish the STM32L476RG update process, due to the various elements required by the FOTA method, even in this case it has been necessary to perform a preliminary mapping of the STM32L476RG flash memory.

Flash memory mapping

STM32L476RG has a 1MB memory flash divided into 512 sectors of 2 KB, so it offers more freedom in mapping it. Because of its finely partitioned structure, it has been possible to implement an advanced firmware update mechanism.

The STM32L476RG flash memory mapping is shown in Figure 6.15.

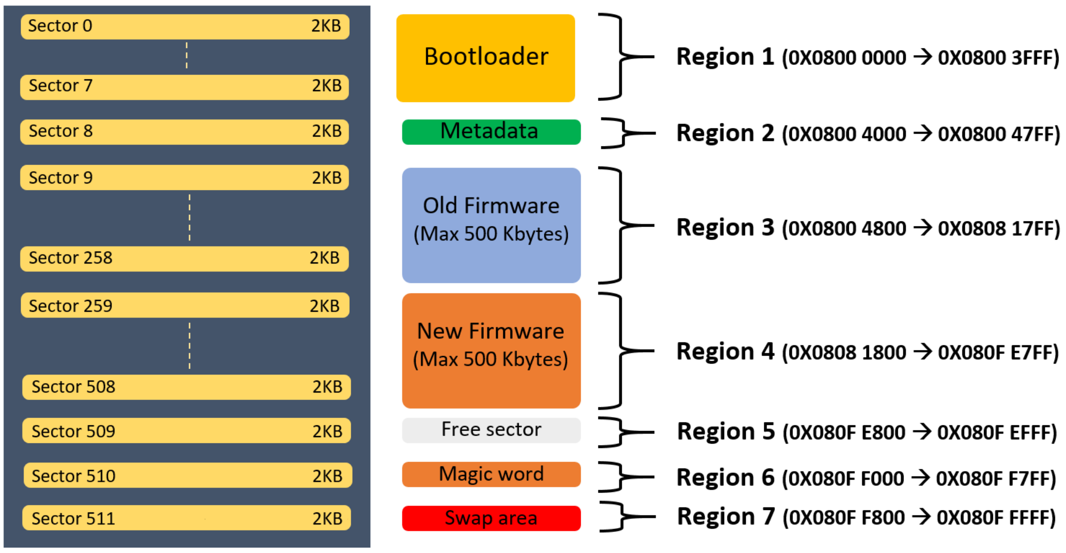

Sector 0 2KB
Sector 7 2KB
Sector 8 2KB
Sector 9 2KB
Sector 258 2KB
Sector 259 2KB
Sector 508 2KB
Sector 509 2KB
Sector 510 2KB
Sector 511 2KB
Bootloader
Metadata
Old Firmware (Max 500 Kbytes)
New Firmware (Max 500 Kbytes)
Free sector
Magic word
Swap area
Region 1 (0X0800 0000 → 0X0800 3FFF)
Region 2 (0X0800 4000 → 0X0800 47FF)
Region 3 (0X0800 4800 → 0X0808 17FF)
Region 4 (0X0808 1800 → 0X080F E7FF)
Region 5 (0X080F E800 → 0X080F EFFF)
Region 6 (0X080F F000 → 0X080F F7FF)
Region 7 (0X080F F800 → 0X080F FFFF)

Figure 6.15. STM32L476RG flash memory mapping.

STM32L476RG bootloader

For this platform, the bootloader has been improved, introducing the possibility of swapping the new and the old firmware in the flash memory, without erasing (and losing) the previous one. A watchdog mechanism has been added, enabling the system to reboot with the previous firmware in case the new one is not running.

So, as it has been made for the STM32F401RE board, the STM32L476RG update mechanism starts when an Update request arrives from the server and it consists in checking if the STATE value is equal or not to “Downloaded”, which means that the download of the new firmware has been completed and if true, setting the STATE value equal to “Update”. Before restart the system and beginning the swap process, the metadata are stored into Region 2 while the magic word is stored into Region 6 of the flash memory.

After the restarting of the system, the bootloader runs first and checks if the MAGIC WORD, stored in Region 6, is present in order to validate the firmware and start the swap phase.

Regarding this particular phase, two different swap strategy have been implemented in order to add versatility to the system. The first one consists in using the flash swap area represented by the Region 7 of the flash memory, while the second one uses 2 KB of memory RAM for the swapping.

These two methods are similar in terms of performance, but the possibility to choose whether to use one or another, adds some degree of freedom for next developers in order to meet the application requirements.

Apart from the particular adopted strategy, in order for the new firmware to run, the bootloader swaps the two firmware images, 2KB at a time, iterating the following steps:

1. Copy Region 3 into Swap area;
2. Erase Region 3;
3. Copy Region 4 into Region 3;
4. Erase Region 4;
5. Copy Swap area into Region 4.

At the end of the swap phase, the bootloader runs the new firmware, that during its initialization phase validates metadata and updates the STATE Resource equal to "Idle" and UPDATE RESULT equal to "Successful".

The STM32L476RG updating process is shown in the Figure 6.16.

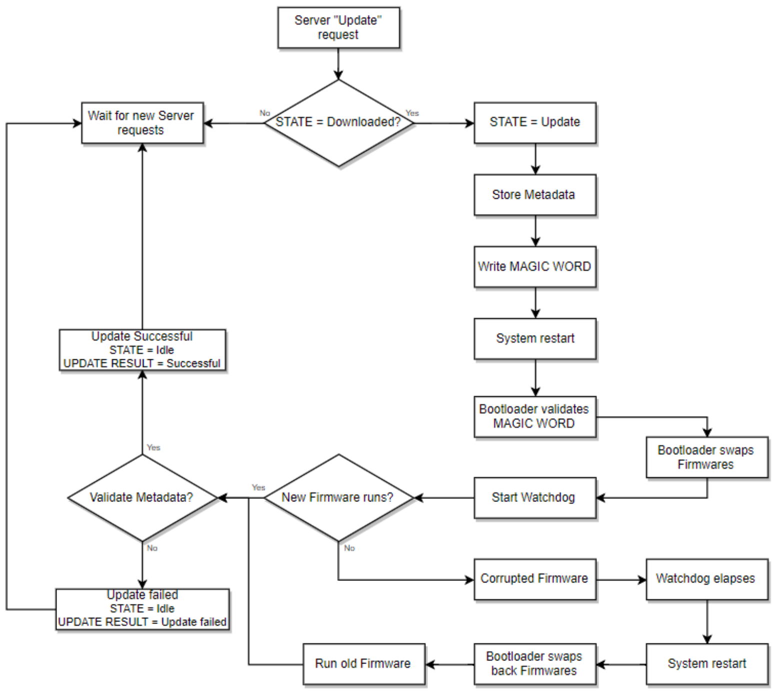

Server "Update" request
STATE = Downloaded?
No
Yes
Wait for new Server requests
STATE = Update
Store Metadata
Write MAGIC WORD
System restart
Bootloader validates MAGIC WORD
Bootloader swaps Firmwares
Start Watchdog
New Firmware runs?
Yes
No
Validate Metadata?
Yes
No
Update Successful STATE = Idle UPDATE RESULT = Successful
Update failed STATE = Idle UPDATE RESULT = Update failed
Corrupted Firmware
Watchdog elapses
System restart
Bootloader swaps back Firmwares
Run old Firmware

Figure 6.16. STM32L476RG update mechanism flow chart.

Recovery mode

Since STM32L476RG supports dual boot operation, a recovery technique has been implemented in order to add dependability to the system, aiming to restore the system in case of download or update failures.

The recovery mode starts after the reboot phase in which, before the new firmware is booting, the watchdog is activated with its timer set at the value of 762 ms. If the watchdog timer elapses and reaches the zero value, a reset command is triggered, so when the bootloader starts again, checks if the reset has been caused by the watchdog reading the RCC_FLAG_IWDGRST watchdog flag, and if that is confirmed, it starts to swap back the two firmwares stored in the flash memory in order to restore the system.

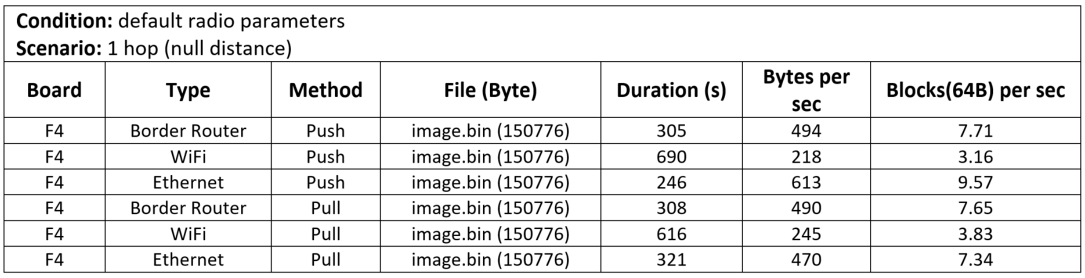
The watchdog flag is set when the firmware does not run correctly and this could happen because the firmware is corrupted or there has been a problem during the Update phase. In the opposite case, if the new firmware starts to run normally, it takes care of resetting the watchdog timer to its initial value in order to never reach the zero, which means that the new firmware update was performed successfully.

Chapter 7

Tests and results

In order to characterize how the FOTA method impacts the firmware update process, it has been necessary to perform many tests, experiencing different contexts and scenario, with the goal of collecting some measurements about power consumption, transfer time and throughput.

All the tests have been performed using an Ethernet router. This choice has been made according to the fact that some previous testing on the firmware update using WiFi router and the border router (with serial line interface) gave in output worse throughput results as in shown in the Figure 7.1.



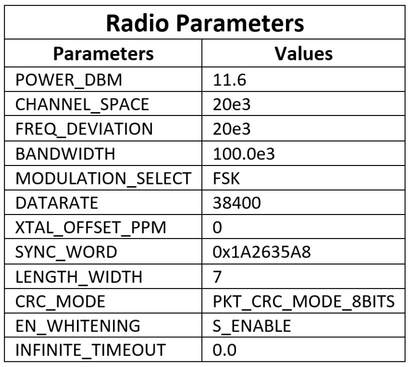
Condition: default radio parameters
Scenario: 1 hop (null distance)

Board	Type	Method	File (Byte)	Duration (s)	Bytes per sec	Blocks(64B) per sec
F4	Border Router	Push	image.bin (150776)	305	494	7.71
F4	WiFi	Push	image.bin (150776)	690	218	3.16
F4	Ethernet	Push	image.bin (150776)	246	613	9.57
F4	Border Router	Pull	image.bin (150776)	308	490	7.65
F4	WiFi	Pull	image.bin (150776)	616	245	3.83
F4	Ethernet	Pull	image.bin (150776)	321	470	7.34

Figure 7.1. Routers comparison table.

Test results have been obtained after normalizing all the different firmwares images to a given dimension of 100KB in order to make a consistent comparison. Moreover, all the tests have been performed keeping for each device the same radio parameters that are listed shown in Figure 7.2.

All the tests have been performed with two different scenarios, a one hop and a two hops network. In the first one, the final node is located close to the router, so that there was no significant distance between the two of them, while in the second one instead, the intermediate node is in charge to forward the firmware image to



Radio Parameters	
Parameters	**Values**
POWER_DBM	11.6
CHANNEL_SPACE	20e3
FREQ_DEVIATION	20e3
BANDWIDTH	100.0e3
MODULATION_SELECT	FSK
DATARATE	38400
XTAL_OFFSET_PPM	0
SYNC_WORD	0x1A2635A8
LENGTH_WIDTH	7
CRC_MODE	PKT_CRC_MODE_8BITS
EN_WHITENING	S_ENABLE
INFINITE_TIMEOUT	0.0

Figure 7.2. Radio parameters values used for testing.

the final node. In this case, the nodes were distant 30 meters between themselves, among a noisy environment.

Despite of the huge amount of tests carried out, the most significant ones are related to the difference between the performance of STM32F401RE and STM32L476RG devices, the behaviour of the forwarding node with respect to the final node in term of power consumption, the Push and Pull transfer modes comparison and the overhead that DTLS secure algorithm has on a firmware update process.

The particular measurements collected for each tests are:

- Firmware transfer time [s];
- Bytes per sec [bytes/s];
- Packets per sec [Packets(64B)/s];
- Avarage current [mA];
- Energy consumption [mJ].

While the Firmware transfers time, the Bytes per sec and the Packets per sec have been detected by the node itself, the X-NUCLEO-LPM01A with its STM32CubeMonPwr tool have been used in order to measure the power consumption and current. This board is indeed able to measure current and energy consumption using just a direct wire linking to the device. Since we had the need of measuring both MCU and SPIRIT1 radio, that are in a matter of fact the main actors of the firmware update process, these particular components have been tested separately.

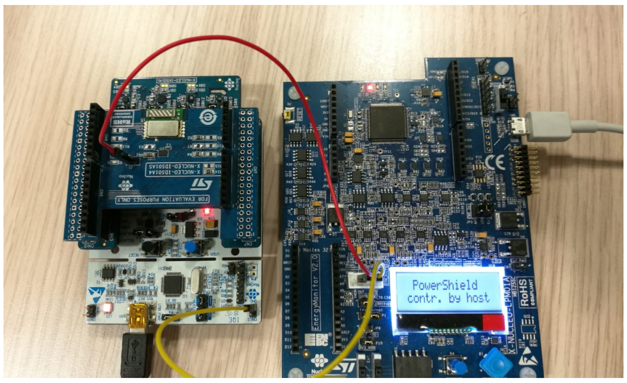


Figure 7.3. Measurement of energy consumption of the node (on the left) through the X-NUCLEO-LPM01A board (on the right).

7.1 Graphs analysis

Even though several tests have been performed, only the most significant graphs are reported here in order to give a consistent characterization of the FOTA method behaviour in the contests that we want to analyze. Moreover, it is important to stress that these results do not represent absolute values but they want just to be a guideline for future works.

The first bar graph in Figure 7.4 shows the comparison in terms of throughput [Bytes/s] between STM32F401RE and STM32L476RG boards that operates in Push and Pull transfer mode in a single hop and two hops scenario.

As we can see, in one hop scenario the throughput is about twice the two hops one in all the cases. This result depends on the idle time of the node introduced by the distance in the two hops scenario.

According to the graph, moreover, it is clear that both for STM32F401RE and STM32L476RG boards, the Push transfer mode offers a higher throughput value with respect to the Pull transfer mode.

Considering the two boards separately, we can appreciate how they present the same throughput both regarding Push and Pull transfer modes.

The second bar graph reported in Figure 7.5 shows the comparison between the STM32F401RE and STM32L476RG in terms of energy [mJ]. Even in this case, the boards operate in Push and Pull transfer modes but with three different roles:

- ***1 Hop***: the node that performs the firmware update is directly connected to the router with a negligible distance;

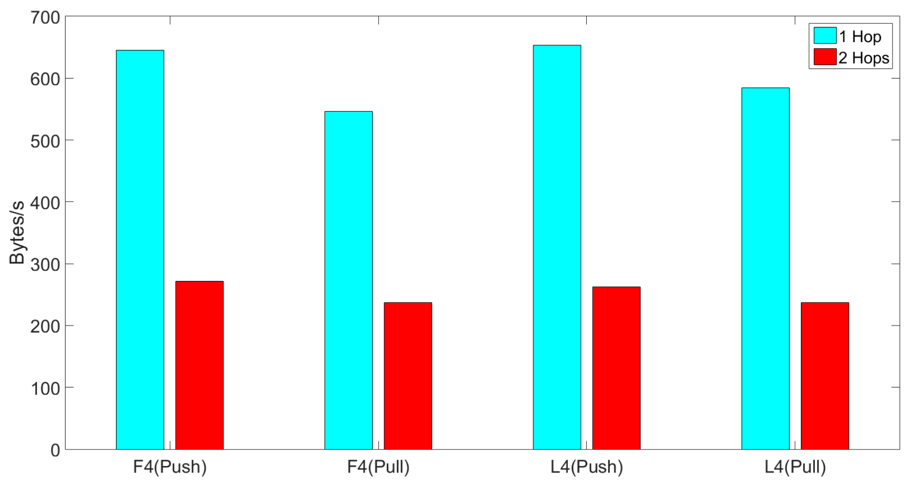


Figure 7.4. Throughput comparison between STM32F401RE and STM32L476RG boards in a one hop and two hops scenarios.

Forwarding: the node that is in charge of forwarding packets to the final node, which receives the firmware update, is located between the router and the latter;

Final node 2 hops: the node that performs the firmware update is connected to the router through a forwarding node.

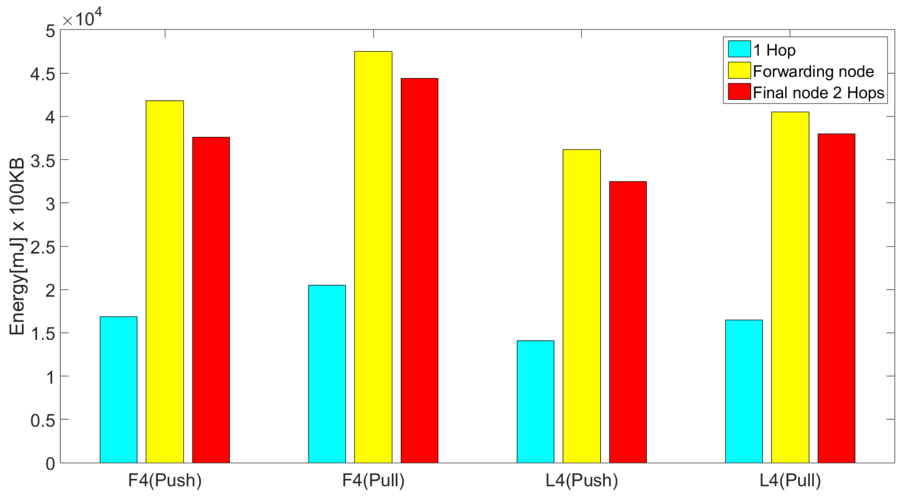


Figure 7.5. Energy comparison between the forwarding node and the final node.

From the analysis of the graph is highlighted that in all the cases, the forwarding node has the highest value in terms of energy consumption. In particular, comparing the two nodes that operate in the same scenario, which are the forwarding node and the final node, it is possible to notice a small but considerable energy consumption overhead due to a higher power consumption in the transmission phase of the SPIRIT1 radio.

In each scenario the STM32F401RE device consumes more energy than the STM32L476RG one and this is explained by the fact that the STM32L476RG relies on a low power internal architecture with a MCU that has a clock frequency of 80 MHz which is lower than the STM32F401RE 84 MHz clock frequency one. Regarding the transfer modes instead, the Push one gives a lower power consumption level for both STM32F401RE and STM32L476RG devices.

The graph also shows that in every case a 2 hops firmware update results in a much more energy consumption with respect to a 1 hop scenario even if lower differences about the power can be appreciated. This trend highlights that the network topology plays an important role for a firmware update.

The last two bar graphs in Figure 7.6 and in Figure 7.7 regard the DTLS secure protocol overhead during a firmware update using the Push transfer mode both with STM32F401RE and STM32L476RG in terms of throughput [Bytes/s] and energy consumption [mJ].

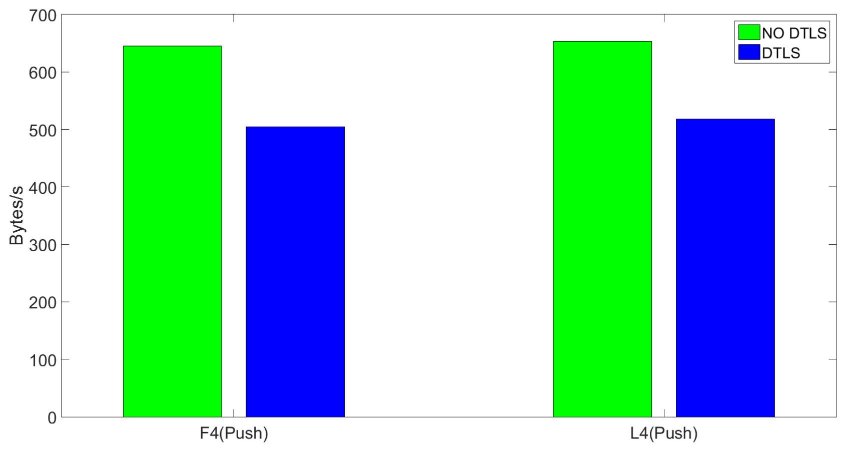


Figure 7.6. DTLS secure protocol throughput overhead.

Studying the graph, we can see that the DTLS security protocol affects the throughput and the energy consumption of the firmware update, due to the messages overhead and the big effort required by the devices in order to decrypt the data.

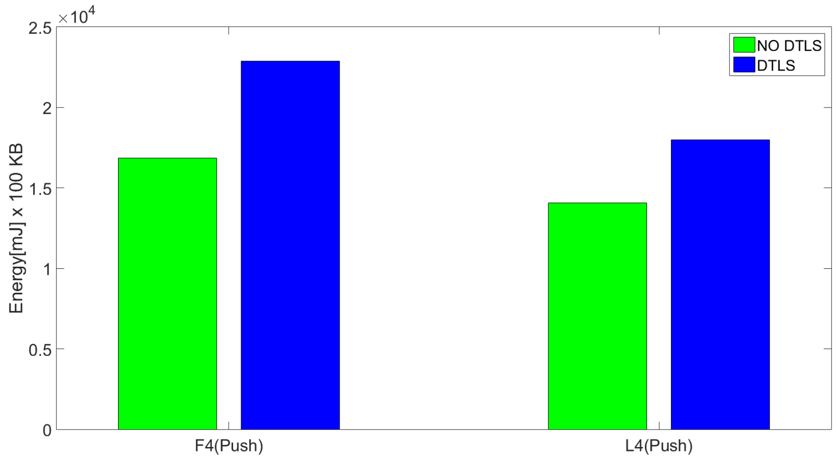


Figure 7.7. DTLS secure protocol energy overhead.

Indeed, we can observe that the DTLS adds approximately an overhead of about 20-25% with respect to the firmware update with no DTLS enabled and that the throughput values are directly proportional to the energy consumption.

The reported data refer to a normalized firmware size of 100KB, so they highlight the overhead intrinsic to DTLS processing only. It is important to note that, without changing other parameters, a DTLS enabled firmware will be bigger than a plain one, so a further overhead due to the major number of blocks has to be considered as well.

7.2 Outcomes

In conclusion, from what we can observe, the STM32L476RG board, thanks to its internal architecture, ensures the same performance as the STM32F401RE board in terms of throughput, although offering a lower energy consumption in order to perform the firmware update.

Concerning the transfer mode, it is clear instead that the Push transfer mode gives, in all the scenarios, the best performance in terms of throughput and energy consumption, thanks to the fact that a higher throughput allows to lower the idle time of the node and reduce the energy consumption itself.

In the two hop scenario we can observe that the forwarding node consumes more energy than the final node which receives the firmware. This overhead shows that the SPIRIT1 radio is more power hungry during the transmission phase highlighting that forwarding operation require more energy compared to the validating and writing operations.

Furthermore, the comparison between one and two hop scenarios, indicates how the network topology is one of the most important elements for the firmware update process in a real 6LoWPAN IoT application. In fact, even if the nodes show different power consumption values depending on the scenarios, the adopted devices and the transfer method, the largest impact in terms of energy consumption is due to the network topology, that in the performed tests, represents the most critical parameter to manage in order to reduce the energy waste in a firmware update.

Regarding security, the test proves that in this context an overhead of 20-25% in terms of throughput and energy consumption has to be taken into account during a node firmware updating process.

Chapter 8

Conclusions

In this dissertation, a new FOTA method has been evaluated because of the need to guarantee a firmware update process in heterogeneous Low Power IoT networks, exploring the implementation details and quantifying the energy costs, the throughput and the time required to complete this specific operation. It is based on the 6LoWPAN communication protocol and uses the OMA-LwM2M as Device Management and Data Model standard along with the CoAP application layer protocol to enable M2M communications.

The description of the FOTA method that has been presented, whose architecture consists in many components based on ST devices, reported first the definition of the OMA-LwM2M Firmware Object along with its Resources and the implementation of the update mechanism defined by the standard. In addition to that, a solid management of the entire process has been provided.

Regarding the firmware transfer, the block-wise CoAP option has been used to enable the firmware transfer in both Push and Pull mode on two different boards, the STM32F401RE and the STM32L476RG, where for the latter, in particular, the dual boot mechanism has been implemented. Moreover, the FOTA method, in order to add dependability to the system, offers a recovery mechanism that is able to identify cases of failures arising from the downloading or updating phases, thanks to a watchdog peripheral, and to run the new firmware or restore the old one.

After the implementation phase, several tests have been executed in different contexts and scenarios with the purpose of characterizing the FOTA method in terms of throughput, required time and power consumption, also quantifying the influence of the DTLS security protocol on the firmware update process.

In conclusion, from what we observed, the STM32L476RG board ensures the same performance as the STM32F401RE board in terms of throughput, although offers a lower energy consumption. Moreover, the Push transfer mode gave in all the scenarios the best performance with respect to the Pull one. Regarding the network architecture, in the two hop scenario we observed that the forwarding node consumes more energy than the final one which receives the firmware. Furthermore, tests proved that in this context a conspicuous overhead was given by the DTLS security protocol used during the firmware transfer. Eventually, the tests indicated how the network topology is one of the most important element that has to be taken into account for a firmware update process in real 6LoWPAN IoT applications.

Starting from this developed and tested FOTA method, there are several possible future works that can be carried on for different part of the project that has been presented, aiming to:

- Evaluate HTTP/HTTPS as application layer protocols;
- Create new firmware update strategies based on the network topology, such as code propagation;
- Implement firmware compression or patch based techniques to optimize the transfer process.
- Reduce the board power consumption using low power strategies, exploiting the device's low power features and the particular network topology where the device will be adopted;

Bibliography

[1] Open Mobile Alliance. “Lightweight Machine to Machine Requirements”. In: *Candidate Version* 1 (2013).

[2] Carsten Bormann and Zach Shelby. *Block-Wise Transfers in the Constrained Application Protocol (CoAP)*. Tech. rep. 2016.

[3] Rajkumar Buyya and Amir Vahid Dastjerdi. *Internet of Things: Principles and paradigms*. Elsevier, 2016.

[4] Contiki. *Contiki*. URL: http://www.contiki-os.org/index.html.

[5] Eclipse. *Californium*. URL: https://projects.eclipse.org/projects/technology.californium.

[6] Eclipse. *Leshan*. URL: https://github.com/eclipse/leshan/wiki.

[7] Ericsson. *Ericsson IoT forecast*. URL: https://www.ericsson.com/en/mobility-report/internet-of-things-forecast/.

[8] Klaus Hartke. “Observing resources in the constrained application protocol (CoAP)”. In: (2015).

[9] Intel. *Intel IoT Platform*. URL: http://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/stm32-nucleo-expansion-boards/x-nucleo-lpm01a.html.

[10] Neha Jain, Swapnil G Mali, and Suhas Kulkarni. “Infield firmware update: Challenges and solutions”. In: *Communication and Signal Processing (ICCSP), 2016 International Conference on*. IEEE. 2016, pp. 1232–1236.

[11] Goran Jurkovic and Vlado Sruk. “Remote firmware update for constrained embedded systems”. In: *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014 37th International Convention on*. IEEE. 2014, pp. 1019–1023.

[12] Adrian McEwen and Hakim Cassimally. *Designing the internet of things*. John Wiley & Sons, 2013.

[13] Erkki Moorits and Gert Jervan. “Low resource demanding FOTA method for remote AtoN site equipment”. In: *OCEANS 2010*. IEEE. 2010, pp. 1–5.

[14] Brendan Moran, Milosch Meriac, and Hannes Tschofenig. *A Firmware Update Architecture for Internet of Things Devices*. Internet-Draft draft-moran-suit-architecture-03. Work in Progress. Internet Engineering Task Force, Mar. 2018. 30 pp. URL: https://datatracker.ietf.org/doc/html/draft-moran-suit-architecture-03.

[15] NXP. *IoT solutions*. URL: https://www.nxp.com/applications/solutions/internet-of-things:Internet-of-Things-IoT.

[16] Pethuru Raj and Anupama C Raman. *The Internet of Things: Enabling Technologies, Platforms, and Use Cases*. CRC Press, 2017.

[17] Tara Salman and Raj Jain. "Networking Protocols and Standards for Internet of Things". In: *Internet of Things and Data Analytics Handbook (2015)* (2015), pp. 215–238.

[18] Zach Shelby and Carsten Bormann. *6LoWPAN: The wireless embedded Internet*. Vol. 43. John Wiley & Sons, 2011.

[19] Siemens. *MindSphere*. URL: https://www.siemens.com/global/en/home/products/software/mindsphere.html.

[20] STMicroelectronics. *FP-CLD-AZURE1*. URL: https://my.st.com/content/ccc/resource/technical/document/user_manual/group0/07/bb/bb/fe/ce/90/4c/a0/DM00280570/files/DM00280570.pdf/jcr:content/translations/en.DM00280570.pdf.

[21] STMicroelectronics. *FP-SNS-6LPNODE1*. URL: http://www.st.com/en/embedded-software/fp-sns-6lpnode1.html.

[22] STMicroelectronics. *Spirit1*. URL: http://www.st.com/content/ccc/resource/technical/document/datasheet/68/6c/7b/ec/b2/6b/49/16/DM00047607.pdf/files/DM00047607.pdf/jcr:content/translations/en.DM00047607.pdf.

[23] STMicroelectronics. *ST IoT solutions*. URL: http://www.st.com/en/applications/internet-of-things-iot.html.

[24] STMicroelectronics. *STM32 ST-LINK Utility*. URL: http://www.st.com/en/development-tools/stsw-link004.html.

[25] STMicroelectronics. *STM32CubeMonPwr*. URL: http://www.st.com/content/st_com/en/products/development-tools/software-development-tools/stm32-software-development-tools/stm32-performance-and-debuggers/stm32cubemonpwr.html.

[26] STMicroelectronics. *STM32F401RE*. URL: http://www.st.com/content/ccc/resource/technical/document/datasheet/30/91/86/2d/db/94/4a/d6/DM00102166.pdf/files/DM00102166.pdf/jcr:content/translations/en.DM00102166.pdf.

[27] STMicroelectronics. *STM32F429ZI*. URL: http://www.st.com/content/ccc/resource/technical/document/datasheet/03/b4/b2/36/4c/72/49/29/DM00071990.pdf/files/DM00071990.pdf/jcr:content/translations/en.DM00071990.pdf.

[28] STMicroelectronics. *STM32L476RG*. URL: http://www.st.com/content/ccc/resource/technical/document/datasheet/c5/ed/2f/60/aa/79/42/0b/DM00108832.pdf/files/DM00108832.pdf/jcr:content/translations/en.DM00108832.pdf.

[29] STMicroelectronics. *STMicroelectronics*. URL: http://www.st.com/content/st_com/en.html.

[30] STMicroelectronics. *System Workbench toolchain*. URL: http://www.st.com/en/development-tools/sw4stm32.html.

[31] STMicroelectronics. *X-CUBE-SBSFU*. URL: http://www.st.com/content/ccc/resource/technical/document/user_manual/group0/33/ee/5b/6b/c7/43/44/3e/DM00414687/files/DM00414687.pdf/jcr:content/translations/en.DM00414687.pdf.

[32] STMicroelectronics. *X-NUCLEO-LPM01A*. URL: http://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/stm32-nucleo-expansion-boards/x-nucleo-lpm01a.html.

[33] Tera Term. *Tera Term*. URL: https://ttssh2.osdn.jp/manual/en/.

[34] Pascal Thubert and Jonathan W Hui. "Compression format for IPv6 datagrams over IEEE 802.15. 4-based networks". In: (2011).

[35] TI. *TI IoT*. URL: http://www.ti.com/ww/en/internet_of_things/iot-products.html#.

[36] Samet Tonyali, Kemal Akkaya, and Nico Saputro. "An attribute-based reliable multicast-over-broadcast protocol for firmware updates in smart meter networks". In: *Computer Communications Workshops (INFOCOM WKSHPS), 2017 IEEE Conference on*. IEEE. 2017, pp. 97–102.

[37] Wireshark. *Wireshark*. URL: https://www.wireshark.org/.