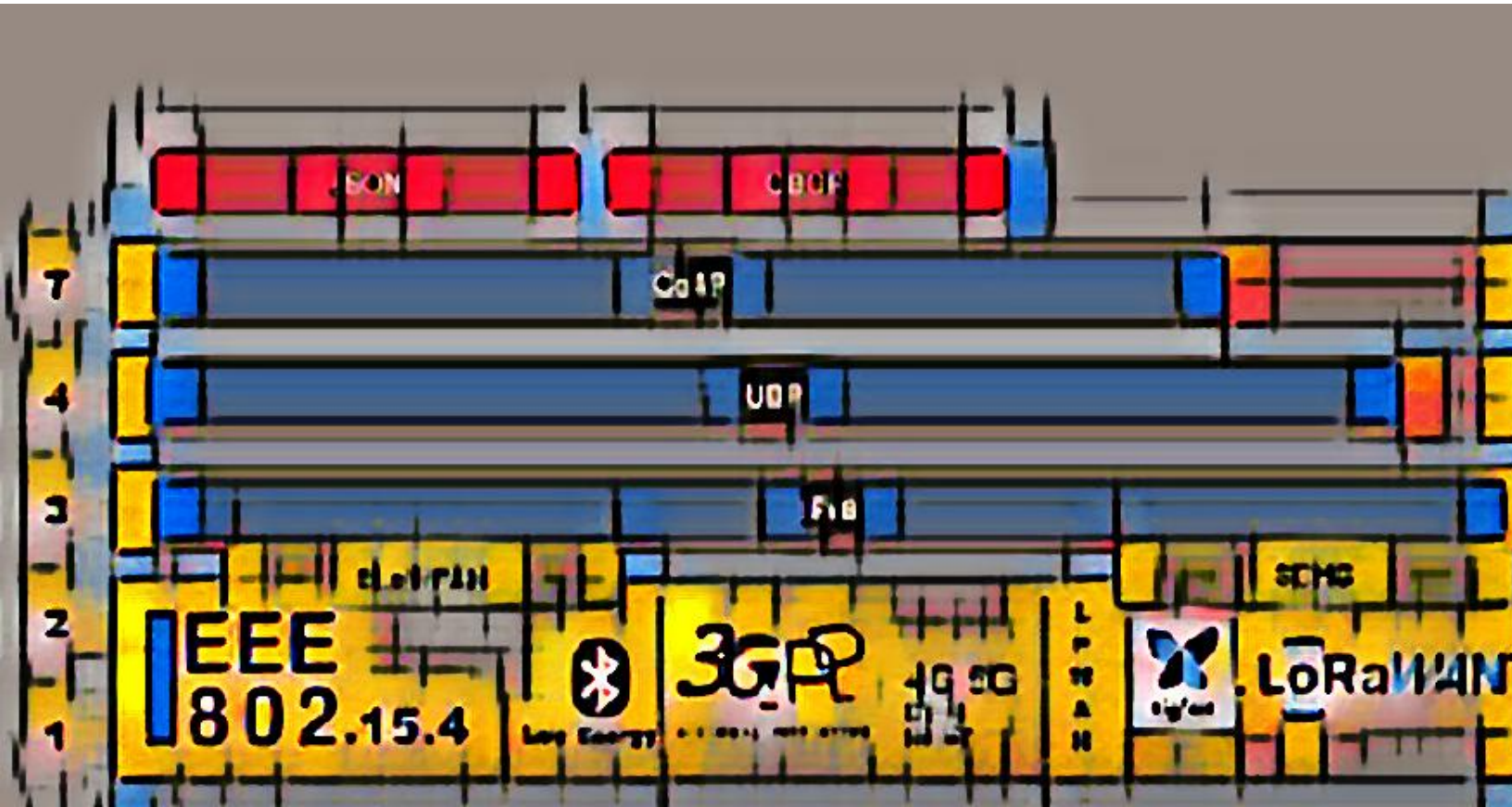
PROGRAMMING THE INTERNET OF THINGS

Laurent TOUTAIN



IMT ATLANTIQUE

Based on the PLIDO MOOC.
Published 18 août 2022

Table of Contents

Acronyms

3GPP 3rd Generation Partnership Project
ABP Authentication By Personalisation
ADSL Asymmetric Digital Subscriber Line
AMQP Advanced Message Queuing Protocol
AS Application Server
ASCII American Standard Code for Information Interchange
BLE Bluetooth Low Energy
CBOR Concise Binaire Object Representation
CoAP Constrained Application Protocol
Cosem Companion Specification for Energy Management
CRC Cyclic Redundancy Check
CSV Comma Separated Values
DLMS Device Language Message Specification
DTT Digital Terrestrial Television
DR Data Rate
GSMA GSM Association
HTML HyperText Markup Language
HTTP HyperText Transport Protocol
HTTPS HyperText Transport Protocol Secure
IANA Internet Assigned Numbers Authority
IBAN International Bank Account Number
IEEE Institute of Electrical and Electronics Engineers
IETF Internet Engineering Task Force
IoT Internet of Things
IP Internet Protocol
IPv4 Internet Protocol version 4
IPv6 Internet Protocol version 6
IPSO IP for Smart Objects
ITU International Telecommunication Union
IRI International Resource Identifier
ISBN International Standard Book Number
ISO International Standardization Organization
JMS Java Messaging Service
JSON JavaScript Object Notation
JSON-LD JavaScript Object Notation for Linked Data
LCIM Levels of Conceptual Interoperability Model
LPWAN Low Power Wide Area Network
LwM2M Lightweight Machine to Machine
LNS LoRaWAN Network Server
MQTT Message Queuing Telemetry Transport
NAT Network Address Translation
NGW Network GateWay
NIDD Non IP Data Delivery
OMA Open Mobile Alliance
OTAA Over The Air Authentication
OVH On Vous Herbèrge
PAC Porting Authorization Code
REST REpresentational State Transfer
RFC Request For Comments

RGW Radio GateWay
RNIPP Répertoire National d'Identification des Personnes Physiques
RSSI Received Signal Strength Indicator
RTT Round Trip Time
SCEF Service Capability Exposure Function
SenML Sensor Measuring List
SCHC Static Context Header Compression
SF Spreading Factor
SNR Signal to Noise Ratio
SSID Service Set IDentifier
STIC Sciences et Technologies de l'Information et de la Communication
TCP Transmission Control Protocol
TLV Type Length Value
TNT Télévision Numérique Terrestre
TTN The Things Network
UDP User Datagram Protocol
UIT Union internationale des télécommunications
UNB Ultra Narrow-Band
URI Universal Resource Identifier
URL Univeral Resource Locator
URN Univeral Resource Name
VPS Virtual Private Server
W3C World Wide Web Consortium
WWW World Wide Web
XML Extensible Markup Language
XMPP Extensible Messaging Protocol et Presence

Introduction

The Internet of Things will not only add a new category of equipment to the network, it will also change the way protocols are implemented. So we need to be different but the same, disruptive but conservative. The Internet of Things is a major evolution of global protocols to meet two fundamental challenges : to be energy efficient and above all to be interoperable ; that is to say, to enable Objects to be easily integrated into existing information systems and ultimately to make the information produced widely available.

The IoT also changes the way we teach. Until now, this teaching was very stratified, with lectures showing how the protocols work and how they are organized. The practical implementation was more concerned with the configuration of interconnection equipment such as routers. The traditional vision consists in separating the functionalities. The protocols are stacked one on top of the other with the functionalities such as the coding of information, on which the routing of information is based, and at the top the applications. Each protocol in this stack is independent of the others and has very strict boundaries.

This vision has to be revised for the Internet of Things, the low memory capacities, the nature of the communication links make it difficult to specialize in only one field. Thus, I hoped to have forgotten forever the signal processing or electronic aspects when I left the university. However, in order to design an object, a multidisciplinary approach is essential, so you have to understand electronic concepts in terms of space as well as power consumption, signal processing because the signals are generally very weak, without forgetting traditional network problems such as routing, auto-configuration or security. It is also necessary to have an eye on the applications : how the data are represented and coded during their transmission and how they can interact with existing services.

The Thing itself is only a small part of the problem, it will send a more or less important flow of information to servers that will be responsible for analyzing, storing and finding trends. Since small streams make big rivers, these servers or the networks that lead to them must be correctly sized. But also that the cost of managing an object is very low, otherwise the cumulative cost of millions of objects can be a barrier to deployment..

This book is the result of the experiences we had with the FabLabs, and of an observation that the network is often the poor relation both of the applications created and of the support found on these platforms ; communications are seen from an application perspective, without taking into account a more global vision of interoperability leading to the concept of the Internet of Things. This is understandable because the protocol stacks of the Internet are relatively large and in a restricted environment, it is easier to get rid of them. Nevertheless, these protocol stacks have an advantage, they promote communication between the network components. Thus, it is possible to control an object from its laptop, data can be sent to servers to be processed,...

The variety of communication possibilities will allow the creation of innovative services. The protocol stacks must also adapt. This is what many organizations are doing, including the IETF, which standardizes the protocols used in the Internet.

This book is an adaptation of the MOOC Programming the Internet of Things on Coursera. It will cover the technologies, architectures and protocols needed for the end-to-end realization of information collection on networks dedicated to the IoT to the structuring of the data and its processing.

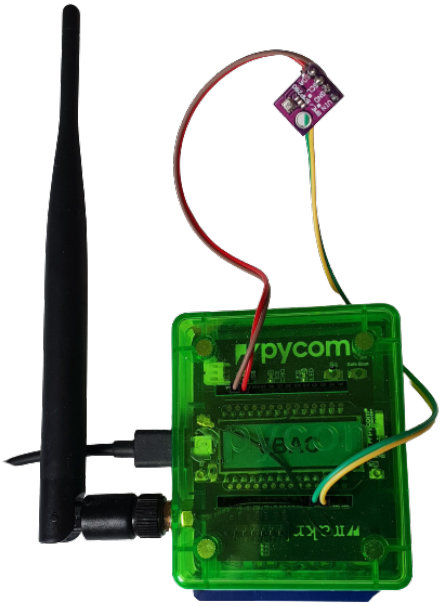
You're going to include :

— discover a new category of networks called LPWAN of which Sigfox and LoRaWAN are the best known representatives ;
— see the evolution of the internet protocol stack from IPv4/TCP/HTTP to IPv6/UDP/CoAP while preserving the REST concept based on resources unambiguously identified by URI ;
— explain how CBOR can be used to structure complex data in addition to JSON ;
— Finally JSON-LD and the MongoDB database will allow us to easily manipulate the collected information. Thus, we will introduce the essential techniques to statistically validate the collected data.

Through this course, you will learn how to program an energy efficient Thing that is interoperable with other Things.

Hardware

You don't need hardware to do most of the exercises and experiment with the concepts. Just about everything can be done with Python scripts, but it's not as much fun as experimenting in real life. Especially when it comes to discovering the magic of long range radio networks. That's why we will use the following material :



- a LoPy4 [1] -*Pycom - Quadruple Bearer MicroPython enabled Dev Board* ; Caution : take the option *With headers*
- A Expansion Board 3.0 [2] - Pycom - *Compatible with Pycom Multi-Network IoT* ;
- une LoRa Antenna [3] (868MHz/915MHz) & Sigfox Antenna Kit - Pycom with its small wire allowing to connect it to LoPy ;
- a box [4],but it is not necessary, if you are not careful with your device *Pycase Clear - Fits Pycom IoT Dev Boards, Expansion Board & Antenna kit* ;
- A sensor BME280 3v3 [5] Capteur de pression temperature humidite BME280 - Boutique Semageek or BME280 3.3 [6] ;
- a USB cable (USB 2.0A to micro B 2.0 1.5 m) ;
- dupont male-femelle [7] cable

LoPy offers a free one-year subscription to the Sigfox network, which is enough to experiment. An annual subscription costs about ten euros. On the other hand, accessing a LoRaWAN network is more problematic. The offers of the operators are not always adapted and the coverage of the community networks is not complete. But you can extend this coverage by installing your own LoRa antenna for less than a hundred euros. We will also use the solution proposed by Pycom.

To do this you need :

- an extension board pygate [8] ;
- a LoPy as before or a slightly cheaper wi-py [9] ;
- a pretty box [10] to look professional ;

1. https://pycom.io/product/lopy4/
2. https://pycom.io/product/expansion-board-3-0/
3. https://pycom.io/product/lora-868mhz-915mhz-sigfox-antenna-kit/
4. https://pycom.io/product/pycase-clear/
5. https://boutique.semageek.com/fr/704-capteur-de-pression-temperature-humidite-bme280-3009052078446.html
6. https://fr.aliexpress.com/item/1005002387867504.html?spm=a2g0o.productlist.0.0.580f7c3elwXr70&algo_pvid=bbd88dd7-92c5-4904-a4e7-6045b186dbd6&algo_expid=bbd88dd7-92c5-4904-a4e7-6045b186dbd6-1&btsid=0b0a182b16193744192742943e5955&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_
7. https://www.amazon.fr/cable-dupont/s?k=cable+dupont
8. https://pycom.io/product/pygate/
9. https://pycom.io/product/wipy-3-0/
10. https://pycom.io/product/pygate-case/

— its antenna with its cable [11] ;
— and this time a USB-C cable.

0.1 Available resources

This Open Source book is available on `http://source.plido.net` in French `http://livre.plido.net`and English `http://book.plido.net`version. Remarks and comments can be sent back using Github tools like *Issue* and *Pull requests*.

The videos referenced in this book are also available on Youtube `http://video.plido.net`. And as the Youtubers say, don't forget to like the videos and subscribe to the channel.

Finally, a more interactive version, in the form of a MOOC, is available here : `http://mooc.plido.net`. The MOOC allows more interactivity with forums to directly ask questions and animations to better configure the system.

0.2 Authors

Laurent Toutain is a professor in the Network Systems, Cybersecurity and Digital Law department at IMT Atlantique, a school of the Mines-Telecom Institute. He is a member of the OCIF team (Communicating Objects - Internet of the Future) which focuses on protocol and architectural evolutions of the Internet related to the design of new services (Smart grid, smart clothes...). After having worked on the IPv6 protocol and transition mechanisms in different environments, he is currently interested in their integration in the Internet of Things. He also contributes to Fab Labs for the adoption of these protocols. He is the author of several reference books on networks.

Kamal Singh est Maitre de Conférences à Télécom Saint-Étienne où il dispense des cours de réseaux informatiques et de réseaux d'opérateurs. Il fait également partie de l'équipe de recherche Data Intelligence du Laboratoire Hubert Curien. Son travail porte sur l'internet des objets, les villes intelligentes, le Big Data, le Web sémantique, la qualité de l'expérience et le software defined networking.

11. `https://pycom.io/product/lora-868mhz-915mhz-sigfox-antenna-kit/`

Marc Girod Genet is an associate professor at Télécom Sud-Paris and a CNRS-SAMOVAR associate researcher (UMR 5157), where he leads the transverse theme on energy. His research interests include personal networks (including sensor networks and measurement architectures), M2M communications and IoT/WoT architectures, semantic data models and ontologies. Marc is also involved in standardization activities within AIOTI (Alliance for Internet of Things Innovation) and ETSI (rapporteur, TC SmartBAN). In 2010, he received the special "Digital Green Growth" jury award for his work on smart grids and energy management (one of his two application areas along with eHealth).

Patrick Maillé is a professor in the Network Systems, Cybersecurity and Digital Law department of IMT Atlantique, a school of the Mines-Telecom Institute. A graduate of the Ecole Polytechnique and Télécom ParisTech, he defended his thesis at Télécom Bretagne (now IMT Atlantique) in 2005. His research work focuses on the economics of telecommunication networks using applied mathematics and economics tools (notably game theory).

Mireille Batton-Hubert is a Professor at the École Nationale Supérieure des Mines de Saint Étienne, at the teaching and research center, the Henri Fayol Institute. She is in charge of the Mathematical and Industrial Engineering team (GMI) and is attached to the UMR Limos. The Mathematical and Industrial Engineering department aims to propose quantitative methods for evaluating the overall performance of companies and their products. The Mathematical and Industrial Engineering department aims to propose quantitative methods for evaluating the overall performance of companies and their products. The Mathematical and Industrial Engineering department aims to propose quantitative methods for evaluating the overall performance of companies and their products. It brings together skills ranging from applied mathematics to industrial engineering.

Vincent Lerouvillois, Teaching Assistant

1. THE BASIS OF THE INTERNET OF THINGS (IOT)

1.1 Introduction

In this first part of the course, we will lay the foundations of what the Internet of Things (IoT) is. What do we mean by "Internet of Things" in the context of the book ? What are the issues that the Internet of Things must address and its evolution today ? What are the underlying technologies, architectures and protocols that will be used in this book ?

To do this, we will draw a parallel between the way the Internet has integrated television and what we are currently experiencing with the Internet of Things.

Youtube

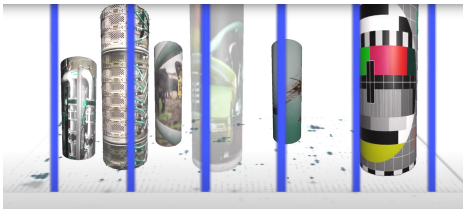
1.1.1 Dedicated networks

In the 1950s, television became very popular and almost every household bought a television set to watch their favorite programs. Dedicated transmission networks were deployed all over the world. In fact each type of communication had its own network one for radio one for telephone one for telex,...

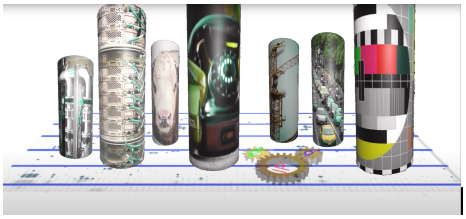
In the 80's, the internet was born, but the transmission speeds were low and the network was limited to file uploads. In the 90s, images and hypermedia with the World Wide Web (WWW) appeared, in connection with the increase of the speeds. Also in the 90s, the increase in power of microprocessors allowed the digitization of the TV signal. Televisions started to include microprocessors and the networks went from analog to digital transmission ; but they remained dedicated to this single use broadcasting television. With the entry in the new millennium, the Internet gained in speed with the Asymmetric Digital Subscriber Line (ADSL) and optical fibers. It was possible to integrate images in web pages

but the quality was poor. At the same time, hundreds of television channels were broadcasting their programs in high resolution via satellite or Digital Terrestrial Television (DTT). Nowadays, Internet communications have gained in speed and quality and some countries have cut off DTT and chosen to transmit their programs only via Internet. In fact the use of the internet is not only a change in the distribution network but also a major change in the uses and applications. You can watch television on your cell phone or even watch your favorite series whenever you want, on demand.

1.1.2 3 technological phases



From this example, we can define three phases in the development of a technology. In the first phase, a specific network is built for a well-defined use. This is called a **vertical** approach ; a technology is dedicated to a single use. It is difficult to exchange information between two verticals. We also refer to **silos** because they are isolated. In a second phase, the verticals start to integrate common technologies but not in a coordinated way. They still cannot communicate easily because they have not made the same choices.



In a final phase, verticals coordinate to converge on the same technologies by defining common rules and uses. This is done in order to reduce costs or to increase their impact in this case. We talk about **horizontal** because it covers several sectors. The Internet has become one of these horizontals for many services. The Internet of Things follows this same movement. Particular solutions have emerged to solve specific needs in agriculture, in the automotive industry, in health, in energy. When Internet networks allowed low cost communications, the architecture of the Internet was taken into account but without compatibility. The change we are currently experiencing is the definition of common functionalities for different domains. The goal is to reduce costs but also to cross information for a better management of the industrial process and a better use of resources.

1.2 The Internet of Things

How do we define the Internet of Things ? Or rather, which Internet of Things are we going to study ? The ambiguity of the two terms "Internet" and "Things" requires a more precise definition ; or at least a classification to better understand what we are referring to.

The internet is now totally integrated in our lives, for work, for education, for entertainment. We use it at home or at work on our computers, and we carry it with us more and more with our smartphones.

Everyone has their own definition of what the internet is. For the general public, it can be very popular applications like Facebook, Tik-Tok, Netflix, Zoom. For some, a little more technophile, the Internet can be confused with the Web which is accessed via Chrome or Firefox. Technicians will talk about protocols like IP, TCP, HTTP, and addresses like IP addresses or URLs.

As this book is technology-oriented, our approach falls more into the latter category. We will see how protocols developed twenty years ago for computers can be applied to other devices that we have yet to define.

The goal of the Internet of Things is to further integrate the Internet to allow something other than computers to exchange data. The main goal is to optimize processes so that they are more efficient to save resources or increase productivity. It is therefore a buried internet, far from the fridge or the connected watch, which will bring back information with an infrastructure or other equipment. We can imagine sensors in a factory to control production, connected cars that will talk to each other to avoid collisions, measuring the filling rate of recycling bins in a city to optimize collection circuits, monitoring the degree of humidity in a field to reduce water consumption...

The Internet of Things can be summarized as follows : using protocols developed for computers and now cell phones (more powerful than the computers used by the Internet in its early days) but in more constrained environments. Indeed, Moore's laws, defining the processing power of processors as well as the continuous decrease of memory costs, have allowed us to provide small objects with resources comparable to those of computers of thirty years ago.

The Internet of Things means solving the following equation : continue to do the same thing because all current information systems use the same principles but do it differently because these principles are too costly in terms of energy, computing time and data exchange.

The Internet of Things (IoT) is a global architecture allowing objects (equipment of the same type or not) to interact autonomously via the Internet. This interaction :

— is realized, by construction, through an Internet network, which generally implies that the objects/things are provided with an IP address ;
— is related to commands (control operations or function calls) or data or information exchanges.

The new IoT paradigm is a convergence of many application domains such as : smart homes or buildings, cities of the future, industry of the future (industry 4.0), energy, transportation systems, agriculture, eHealth, etc., towards a reduced, interoperable and secure protocol suite. The movement is underway and, given the number of players involved, will take several years. But the foundations are already well established and that's what you'll learn in this book.

1.3 The problem

One of the problems with the Internet of Things is that the IoT doesn't start ex-nihilo. Now that the technologies that made the Internet successful are mature, it's not just a matter of applying them to a new domain. Objects were able to communicate long before the Internet existed. Each sector has already developed its solutions, more or less standard, more or less proprietary.

La figure 1.1 on the facing page reprend un certain nombre de travaux et de groupes qui spécifient les protocoles pour l'internet des objets. The figure 1.1 on the next page lists a number of works and groups that specify protocols for the Internet of Things.

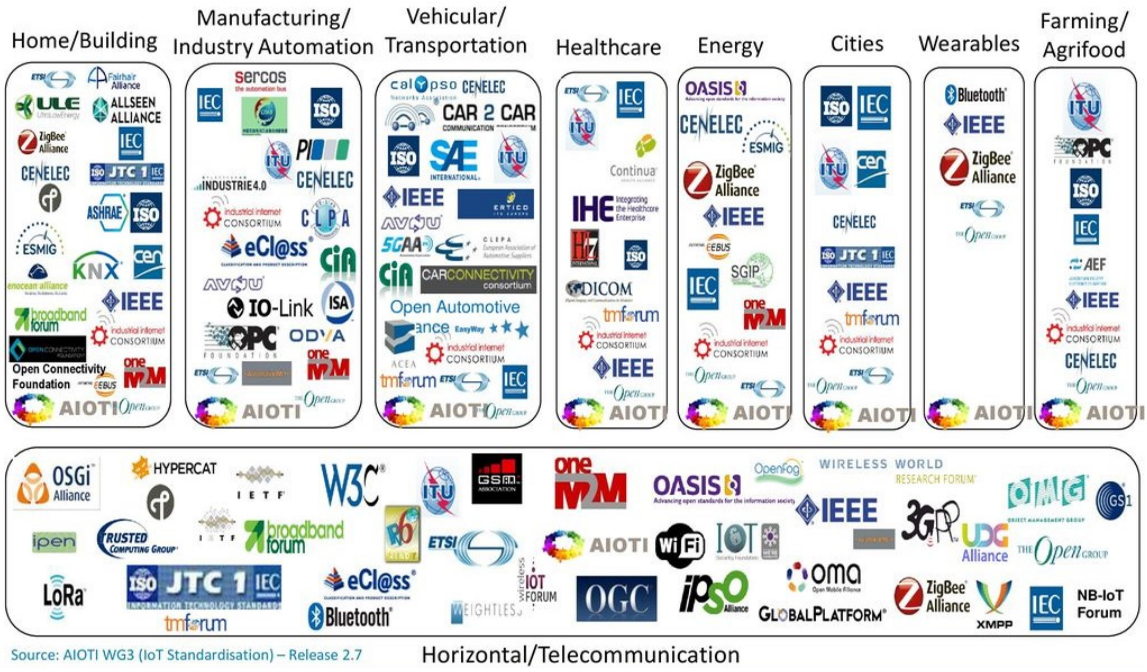


FIGURE 1.1 – Some IoT standards

Without going into detail, we can see that some logos are found in several places, that there is a profusion of solutions for each sector that hinders interoperability and evolution. The Internet of Things, in its broadest sense, is about simplifying this architecture, just as the Internet did a few years ago in the telecoms sector, by simplifying this model and enabling these different players to converge on a common architecture and a smaller set of solutions.

This does not necessarily mean fewer players, but greater consistency in technological choices.

The figure 1.2 on the following page analyzes the IoT by application domain, focusing on the network component. IoT and connected objects are complex systems for which open source solutions, alliances between manufacturers, and standardization bodies are still fragmented ; however, to a lesser extent, showing that structuring is underway.

This fragmentation of the ecosystem is paradoxical. If we summarize, the proposed solutions amount to interrogating a piece of equipment in the field to access a value, process it and send back a command to interact with the environment.

Why are there so many different solutions ? It may come from the needs of reliability, security, data scope, but it also comes from history. Communication with Things is just as old as communication between computers (which are themselves Things). But at the time, each field went its own way, specializing solutions to meet its own needs. The result is solutions optimized for a particular domain. But every time a technology has to be modified, the work has to be adapted for each domain, introducing additional costs and delays.

Also, since each domain has its own data representation, it is relatively difficult to combine them to have a more global vision. We therefore end up with closed systems, expensive, not very scalable, but optimized for the tasks they have to perform.

Over time, the protocols of the Internet could be used, but it is mainly a matter of complementing existing technologies without it being possible to interconnect two domains.

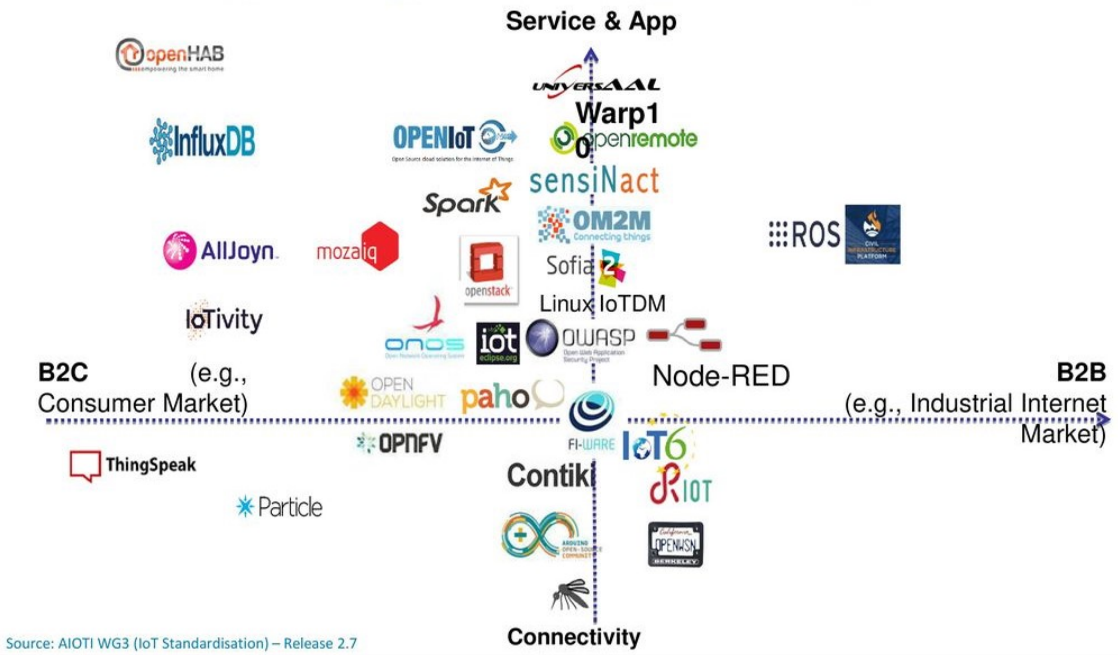


FIGURE 1.2 – Some open source applications for the IoT

1.4 IoT evolution

The example of the evolution of the television network speaks for itself. At the beginning, this network was analog and highly specialized to broadcast programs carried on analog signals on specialized equipment, the television sets.

With the progress of computer processors, it becomes possible to transport data using digital coding. But specialized networks are still needed, as the Internet does not offer the same quality.

The last step consists in integrating these flows into the traditional Internet. This becomes possible by increasing the speed of wired and radio networks (Wi-Fi, 4G...).

This pooling of accesses via the Internet allows not only a reduction of costs, but also the appearance of new uses such as television on cell phones or series on demand.

The Internet of Things follows the same path. In addition to specific technologies, the protocols of the Internet are integrated, but adapted to the contexts of the sector. We are currently experiencing the convergence towards a reduced set of protocols, a standardization of data representation, and its processing on more generic platforms.

The trigger is not the increase in speed as for television, but the possibility of having inexpensive equipment, with reduced capacities compared to traditional computing and energy autonomous, while having a better integration in the current information systems.

1.5 Constrained objects

With the progress of electronics, processors are becoming more and more powerful and the supercomputers of yesterday are now in a watch or a smart-phone.

For the Internet of Things, the logic is a little different. Moore's Law will lead to a reduction in manufacturing costs rather than an increase in processing power. The main criterion for a massive Internet of Things remains energy ; connecting a device to a power source or recharging a battery has a cost. Increasing the speed of the processor or the size of the memory induces a higher energy consumption of the object. We can therefore expect a certain stability in the performance of the objects because they will remain limited in performance.

Objects are generally limited in terms of processing power, memory and energy. According to the IETF standard RFC 7228, devices can be divided into three classes that are also found in the segmentation of processors :

— Class 0, with less than 10 kB of volatile memory to store temporary data and 100 kB of Flash memory to store the object's computer code. It is the equivalent of an **Arduino** UNO (2 kB RAM and 32 kB Flash). It is almost impossible to install both the protocols used to communicate on the Internet (even in a restricted way) and the applications that run on it.
— Class 1 has about 10 KB of RAM and 100 KB of Flash. With an adaptation, it is possible to install an IP stack. For example, equipment like the Pycom Lopy4 that we will use later (and which is in the upper limit) on which the operating system is minimal. Thus, the Pycom uses a simplified version of the Python language (micro-Python) which allows it to be adapted to the system's limitations.
— Class 2 is less restricted with at least 50k RAM and 250k Flash (like a **Raspberry Pi**). The **Linux** operating system can run on these devices. Therefore, there are few limitations on the IP stack and the applications running on it. Class 1 devices have too many restrictions to use the protocols defined for larger objects. The Internet Engineering Task Force (IETF), the organization that standardizes Internet protocols, has proposed a revision of its protocol stack to adapt its protocol stack to a constrained environment.

Figure 1.3 on the next page summarizes the means of interconnection according to the class of the object :

— A class 0 device cannot directly use the internet to exchange information, hence the need to install a gateway to capture the traffic and send it to the internet. It does not have an IP address directly. The LoRaWAN gateways LoRaWAN Network Server (LNS) and 3GPP Service Capability Exposure Function (SCEF) act in this sense (we will come back to this). The data produced is encapsulated by these gateways in protocols such as HyperText Transport Protocol (HTTP) or Message Queuing Telemetry Transport (MQTT), which we will also see later in the course.
— Class 1 devices can also use a gateway to interconnect to the traditional internet, but rather than encapsulating the data produced in other protocols as Class 0 does, gateways for Class 1 devices will translate a constrained protocol into its equivalent in the unconstrained world.Class 2 devices can interact directly with other nodes on the Internet, without going through a gateway.

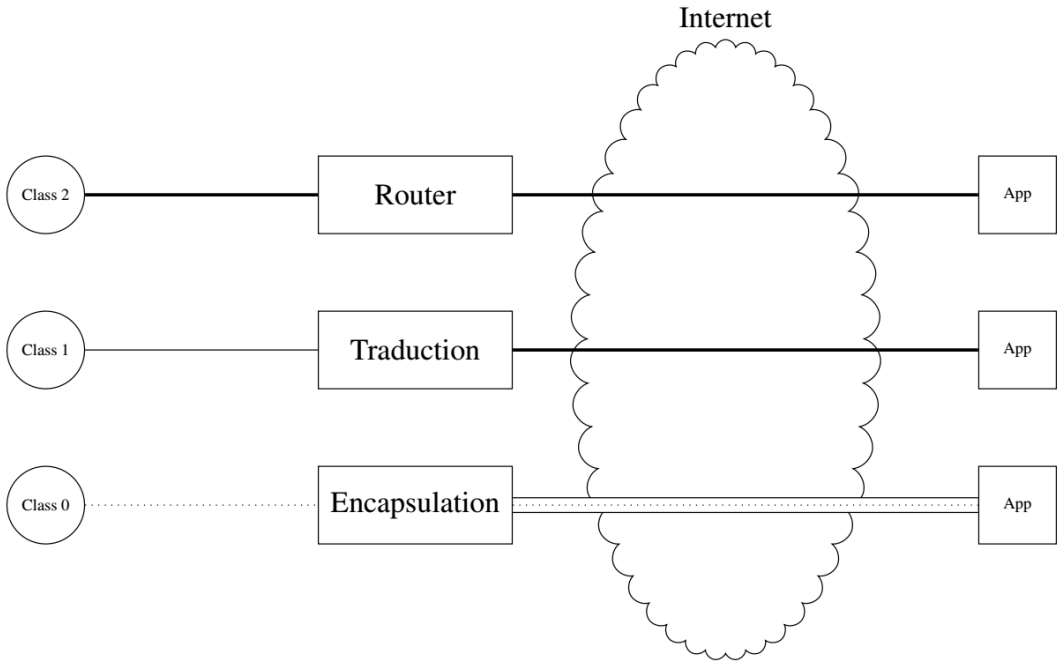


FIGURE 1.3 – Interconnection possibilities

1.6 Interoperability

Another challenge is the number of devices. Some studies predict 500 billion devices by the end of the decade.

With this massive Internet of Things, where almost every piece of equipment will include sensing or actionable elements, integration into an information system will become a real challenge. Because today's Internet of Things is designed for a vertical (i.e., for specific applications), devices are chosen and integrated at the time of system design. Engineers choose their sensors, know exactly what their characteristics are, and write their code based on what they have integrated.

Youtube

The massive Internet of Things is a game changer. Devices or things cannot be integrated into a static information system from the start. The integration has to happen over time and has to handle device evolutions for a long time (device manufacturers may change, products will evolve with new features, etc.).

The Internet we know is a very good illustration of the need for interoperability. It has allowed, thanks to a standardization of the network and a strong decrease of the transmission costs, to develop new uses. You would never have invested in a network dedicated to videoconferencing and teleworking. But by pooling uses, it is possible ! The Internet of Things must follow the same path and also converge with the architecture of the current Internet. The key word is interoperability.

This interoperability issue has been formalized in the model Levels of Conceptual Interoperability Model (LCIM) [TM03] (see figure 1.4 on the facing page) can be represented by a counter to measure

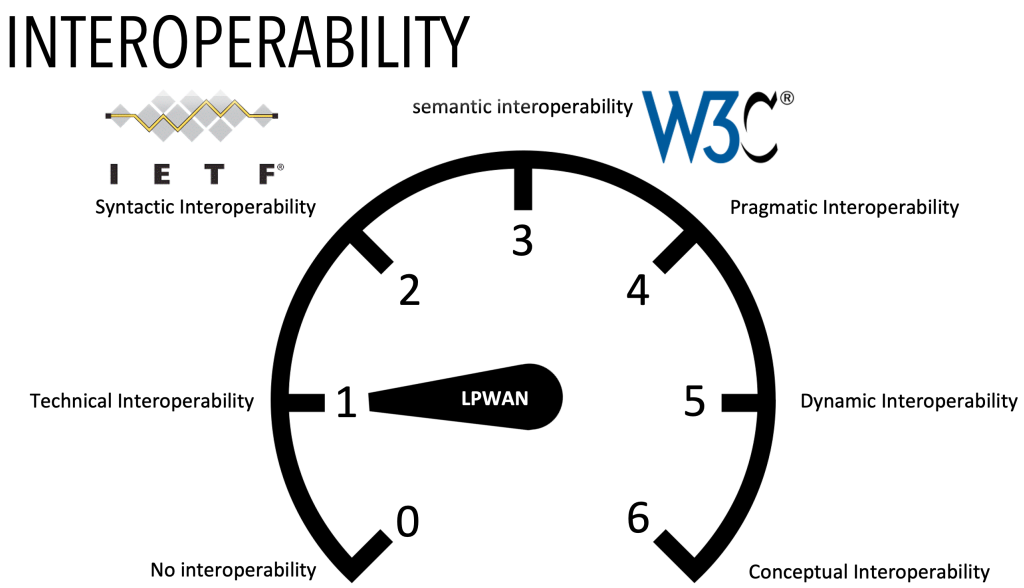


FIGURE 1.4 – Level of interoperability

the degree of interoperability.

It distinguishes six levels of interoperability, among which :

— At level zero, we are not connected ; we don't talk to anyone, so we don't have interoperability problems.

— At level 1, we are able to transmit information, but both sides must know the rules. We have an integrated system ; the applications must know precisely the specifications of the objects with which they communicate, because they define their own data exchange formats. We could take the example of an electrical board where a processor communicates with sensors via a printed circuit. The code running on the processor can be written in advance because there is little chance that a user will loosen the components to replace them with others. This corresponds to encapsulation, if we consider the networked objects in figure 1.3 on the preceding page, the interconnecting element must be configured to encapsulate the data to the right application at the right receiver depending on the sending object.

— Syntactic interoperability (level 2) where two nodes can exchange data without first being configured for this exchange. This is the case of the Internet. By using this suite of protocols, by having a valid address on the network, any application is able to exchange data with another. On the other hand, the data you will exchange is specific to an application. Video conferencing is a great example of Level 2 interoperability. You cannot use Zoom if your correspondent uses Teams because the formats are different. The IETF is the grouping of different actors (industrial, academic,...) who produce the standards related to this network. It is recognized by the acronym Request For Comments (RFC) followed by a number.

— Semantic interoperability (level 3) implies that the receiver is able to interpret the received data. The web is a very good example of level 3 interoperability. Whatever your browser, you can display the pages of a web site and follow the links. The meaning of the information

is understood in the same way on both sides. For the Web, the format HyperText Markup Language (HTML) makes it possible using tags (keywords) to structure a text by adding formatting information or links to other documents. The World Wide Web Consortium (W3C) defines the standards.

— The higher levels of interoperability will be linked to the accuracy of the model that will represent the system.

1.7 The need for standardization

Objects did not wait for the Internet to communicate. They have each evolved in their own vertical, developing solutions that are satisfactory but limited in terms of evolution and interoperability. This report on the dispersion of ecosystems highlights the need to :

— coordination among the alliances ;
— harmonization or alignment of standards ;
— to have reference implementations and reference models as soon as possible.

Otherwise, interoperability will not be properly addressed, new innovative multi-domain services will not be covered, and the development of IoT could be stalled or even aborted.

Aspects related to education and training of IoT actors, as well as those related to acceptance, uses and obviously the socio-economic aspect of IoT, are also essential points that must be considered.

Standards bodies such as the IETF or the W3C have designed protocols or data models capable of handling interoperability. In a way, this is the key to the success of the current Internet. It has solved the problem of interoperability at the syntactic and semantic level but at the cost of large messages.

The challenge for the Internet of Things is to integrate into this giant distributed system. As the Internet of Things is a newcomer, the evolution will have to be done on its side to take into account the existing rules, but adapting them. The next chapters will deal with these changes.

1.8 Questions

Question 1.8.1: New Area

Communicating objects are a brand new field, linked to the progress in miniaturization of electronic components :

- True
- False

Question 1.8.2: Protocoles

Which of these statements is true ?

- There are very few protocols to make objects communicate. As the Internet is a technology that has been very successful, its success will allow objects to communicate.
- There are many solutions to enable objects to communicate, the Internet of Things must enable them to be federated.

Question 1.8.3: Source of data

What is the primary source of data creation in the IoT ?

- ○ sensors
- ○ nanocomputers (Raspberry Pi type)
- ○ Internet
- ○ Web servers

Question 1.8.4: Challenges

What are the main technological challenges for the Internet of Things (3 answers) ?

- ☐ Have a low energy consumption.
- ☐ Have a simplified protocol architecture.
- ☐ Be able to run on open operating systems like Linux.
- ☐ Allow to secure data that may be sensitive.
- ☐ Continuously transmit their status and measured values.

2. ARCHITECTURE OF THE INTERNET

2.1 Protocols

You probably know the principle of protocol stacking in networks. Each protocol provides a service and relies on the lower layer to perform it. The original model defines seven layers to transport the data of an application, anywhere in the world. The network protocols are stacked on top of each other, with those above using the services offered by those below to carry the data. This gave rise to the reference model of the International Standardization Organization (ISO) which has been structuring networks since the 1970s. In theory, there are 7 **layer**s, but the Internet has made this model evolve and the numbers of the layers, associated with functionalities, have remained ; this can lead to a strange numbering.

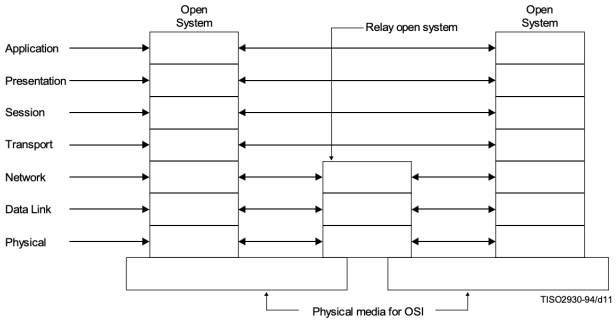


Figure 12 – Communication involving relay open systems

FIGURE 2.1 – Extract from ITU-T Rec. X.200 (1994 E)

The Internet has simplified this architecture (see figure 2.2 on the facing page). That's why there are fewer layers and the numbers are not contiguous.

Youtube

The first two layers from the bottom, grouped under the name of Interface, allow the transmission of binary data on a physical medium. Layer 1 deals with this modulation on a particular physical medium (optical fiber, copper pair, radio wave). Layer 2 groups together the mechanisms that allow this data to be structured in finite size blocks called frames, to define the access methods,

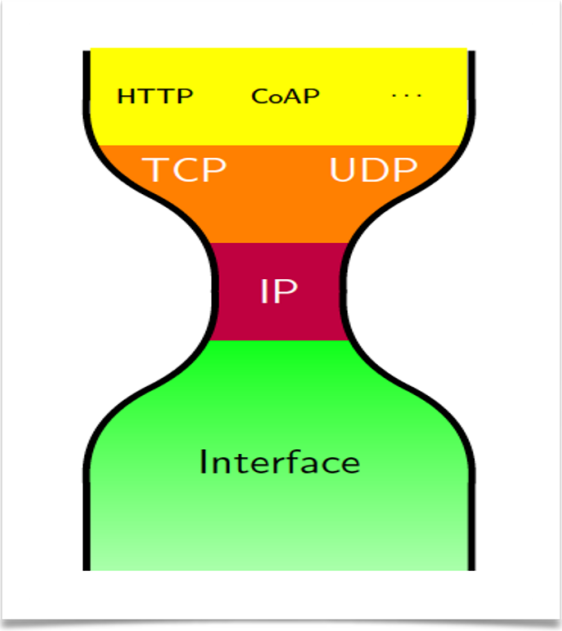


FIGURE 2.2 – Protocol layering of the Internet

i.e. when the equipment can transmit, and the address formats used to identify the equipment.

— the Institute of Electrical and Electronics Engineers (IEEE) which proposes standards like Ethernet for the wired networks or Bluetooth and Wifi for the radio networks,
— the 3rd Generation Partnership Project (3GPP) which operates at the same level and defines the protocols for cellular telephony (4G),
— ...

Above, we have the IETF standardized protocol. The Internet Protocol (IP) protocol adapts simply to any medium of communication. IP thus proposes an abstraction of the means of communication for the application layers, making the access to the network and the addressing universal. The treatment in the **router**s (equipment in charge of routing the information in the network) must be as fast as possible to treat a maximum of packets per second. Moreover, IP does not specialize for one service or another ; it only routes the packets to the right destination. The Internet is a worldwide network built around this protocol, potentially reaching all the equipment connected to it.

Internet experts like this representation in **hourglass** where IP appears in a central position but is smaller compared to the other protocols. By design, IP is very simple ; both to be easily ported to many Layer 2s and to be easily used by higher layers, but also to process data very quickly in the interconnection nodes.

IP is implemented everywhere on the Internet as well in the equipment at the end of the network as in the routers in charge of sending the data to the right destination.

Above that are two protocols that are only implemented in the end devices. If level 3 allows to reach a machine, level 4 will allow to identify the application which must process the data. The

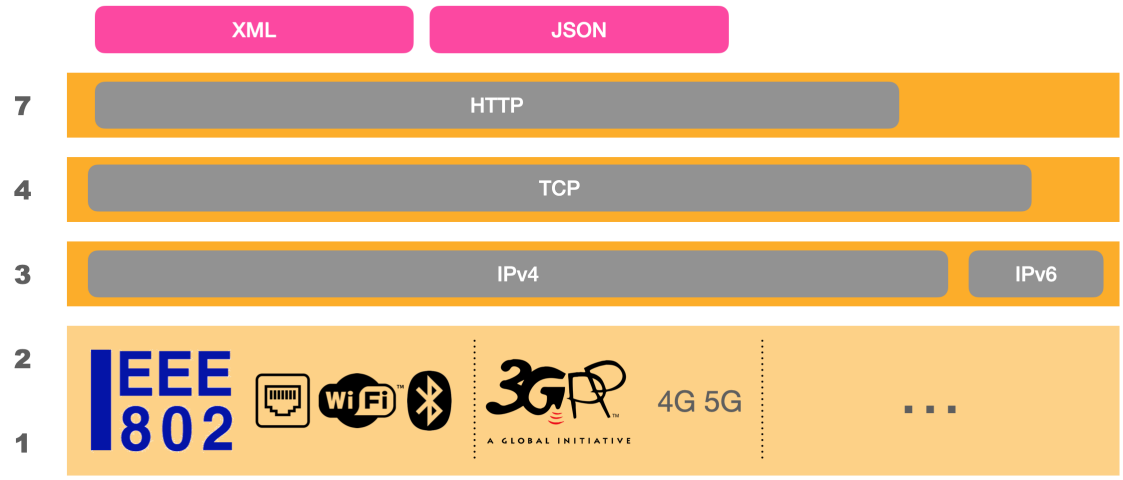


FIGURE 2.3 – Main Internet protocols

"addresses" of these applications are numbers between 1 and 65535 called **port**s. For example, Web servers use port number 80 or 443.

The protocol Transmission Control Protocol (TCP) will monitor the transferred data and will be able to retransmit lost data, slow down or accelerate the data transfer if it detects a network saturation. On the other hand, its implementation is complex and costly in memory. In the simple cases, User Datagram Protocol (UDP) is preferred ; it does not bring additional treatment UDP, it is a minimal protocol which is satisfied to direct the data towards the good application without any other control.

Above, we find the applications that historically are classified in layer 7. The applications are very numerous but the most widespread is HyperText Transport Protocol (HTTP) which is used to transport the Web pages, but also it allows direct communications between computers.

For the general public, the Internet designates above all the totality of this protocol assembly and is often confused with the application that democratized its use : the Web. This is also true for technicians, the traffic produced by the Web is largely present in the Internet. This diagram, figure 2.3 shows the protocol stack that is mostly used in the Internet. We see that at level 3 we have two versions of the protocol ; version 4 is the version historically deployed and it has been so successful that it is more and more difficult to have addresses for machines. To keep the network running, a new version has been developed. Internet Protocol version 6 (IPv6) makes the addressing almost infinite with addresses on 128 bits. IPv6 gains little by little ground in the traditional uses and it is especially an essential brick for the Internet of the objects.

The Web uses mainly the protocol HTTP. And as HTTP relies on TCP, these two protocols are dominant on the network.

Finally this graph adds an additional layer, above layer 7, to indicate how the transported data is structured with formats like Extensible Markup Language (XML) or JavaScript Object Notation (JSON) that we will see in the following.

Question 2.1.1: Protocol Stack

In the internet protocol stack, which protocols are responsible for routing packets to their destination (2 answers)

☐ Ethernet	☐ IPv4	☐ MQTT	☐ JSON
☐ IEEE	☐ IPv6	☐ HTTP	
☐ 802.15.5	☐ UDP	☐ CoAP	
☐ Wi-Fi	☐ TCP	☐ XML	

2.2 Foundations of the Web

One of the most important success stories based on the Internet is the architecture that led to the Web since it is a great source of inspiration for the development of new services. The Web forms large distributed systems and is based on several principles that make it universal and scalable. Surfing with a browser is only the visible part of the traffic ; the principles of the web are also used for video streaming, exchanges between computers.

Youtube

The Web and its extensions are based on a client-server model. Servers own resources and clients can access or modify them through a protocol such as HTTP. The client-server model is something common in computer networks, but the Web follows certain design guidelines known as REpresentational State Transfer (REST).

According to Roy Fielding, who defined this model, REST is a set of principles, properties and constraints. REST uses the client-server communication model and generally uses the HTTP protocol.

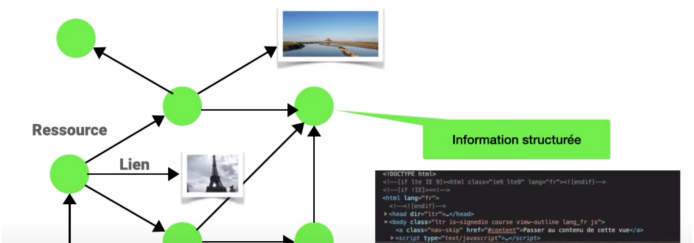
The REST principle allows to design scalable servers. A server must be stateless, which means that it does not retain any information after responding to a client request. This simplifies the processing in the server that has to handle requests from a large number of clients.

This requires that the report be located on the client side. This state is fed from the structured data that the client receives from the server. Thus, when a client requests a Web page, it can contain other Universal Resource Identifier (URI) to complete it, for example images, style sheets, scripts, etc.

The client must therefore understand the data that the server sends him and therefore know the representation format of the resource that he receives in order to find the URI. In addition to the resource itself, the server adds additional information, called metadata. Among other things, this information includes the format of the content (content format). It can be pure text, an image or a structured text format such as HyperText Markup Language (HTML) or JSON.

2.2.1 Resources

The basic element is the resource, which can be defined as a data block of



finite size. Resources can themselves contain references to other resources which in turn will refer to other resources etc. This forms a mesh between resources which is compared to a spider's web. The resource can be, for example, an image in which case it will not refer to anything else. To refer to another resource, its content must be structured and therefore defined in a format where it is easy to understand that part of the content is a reference to another resource. HTML is one such language that allows web pages to reference each other through links.

2.2.2 Identifiers

Each Web resource is identified by a unique value called URI. If the URI contains international characters, (like accented letters, ...) it is called International Resource Identifier (IRI).

For example, to identify an image, we can name it

```
image
```

but there is little chance that this name is unique, other people on Earth have surely had the same idea. On the other hand, if I preface it with my phone number

```
33667789078image
```

will be unique if I name only one resource "image". Another user on the same principle can name his resource :

```
33667239018image
```

without any possible ambiguity. However, because the phone number is unique in the phone number space, other unique numbers could conflict in other numbering spaces.

To avoid conflicts, it is interesting to give the numbering space at the beginning, for example :

```
tel:33667789078image
```

and

```
ss:33667789078image
```

the two identifiers will be unique, even if by chance this phone number and this social security number coincide.

The URI formalize this principle. The RFC 3986 explains how they can be constructed. A URI starts with a scheme indicating the naming authority, followed by an authority value and then a path in the authority space. Characters such as " :" or "/" are used to improve the readability of the URI.

For instance :

```
mailto:mduerstifi.unizh.chssh://utilisateurexample.com
ftp://ftp.is.co.za/rfc/rfc1808.txt
```

FIGURE 2.4 – Structure of a URI

```
http://example.com/ma_ressource
```

No one else in the universe will be able to identify their resources with this string since `example.com` belongs to me. I therefore have an infinite namespace that allows me to designate the infinite set of resources without anyone else being able to take the same names. An URI is an administrative construct that allows you to assign a globally unique identifier to a specific resource.

The URI (cf.figure 2.4) is intended to easily name a resource, to be able to link resources together to form this global spider web. The schema defines both the namespace of the authority and its format. An address or domain name as an authority is both a way to ensure global uniqueness, but also to know how to access the resource.

For example, **spotify** has defined its own schema and then it no longer needs authority but structures the path to reference a playlist.

All the books have a number International Standard Book Number (ISBN) which allows to identify. It can also be integrated in a URI. These two types of identifiers make it possible to refer to a single object but only by reading it one cannot reach the resource. This sub-family of URI identifiers is called Univeral Resource Name (URN).

A subset of URI can be used directly to locate the resource, i.e. find out on which server the resource is located and how to access it. This is a Univeral Resource Locator (URL) that is well known to the general public and used by Web browsers.

The http schema is very convenient because it can also be read as an URL. This scheme gives :

— le protocole à utiliser pour accéder à la ressource (http),
— the authority that indicates the server address (and its port),
— and finally, the access path of what we are going to ask to the server and which can sometimes correspond to a tree of files on a server.

But we must see that the initial goal is to make a unique identifier. The https schema gives the way the suite will be built and in a second time only will be seen as the protocol to use to access the resource. The authority is unique and in a second time will be used to locate the server. And finally the path will indicate how to access the resource on the server. So the resources of our global spider web are present on servers and each resource has a unique identifier. First, the client knows the URI of a resource. If it is a URL, he can contact the server. The server returns the resource to him. The client analyses it and discovers the URLs it contains. It can then question the other server(s) to reconstruct locally a part of the web necessary for the treatment that the client wants to carry out.

FIGURE 2.5 – Content of an HTTP GET request

Question 2.2.1: Unicity

What is unique in the world (6 answers) ?

- ☐ A first name.
- ☐ A family name.
- ☐ a social security number used in France.
- ☐ a passport number.
- ☐ a cell phone number with its international prefix.
- ☐ a full bank account number (International Bank Account Number (IBAN)).
- ☐ the IP address of my machine in my private network.
- ☐ the IP address of a Coursera server (`13.225.34.28`).
- ☐ the domain name `plido.net`.
- ☐ the name of a city.

2.2.3 Interactions

The interactions between clients and servers are very simple. The client will manage the interactions with the resources on a server. It can, for example, retrieve a resource using a **GET** method. It can also write data in an existing resource thanks to a method **PUT**.

The number of interactions is very limited. HTTP or HyperText Transport Protocol Secure (HTTPS) is a way to implement these methods.

HTTP is a protocol which can be used to implement a Web server the principles of REST. (qualified in English of **RESTfull**). HTTP defines different methods for the client to interact with resources on the server :

— **GET** is used to retrieve the representation of a resource (e.g. Web page, temperature value of a sensor, etc.). For example, the figure 2.5 gives the header format HTTP GET to retrieve a Web page ;
— **HEAD** est utilisée pour récupérer uniquement les métadonnées présentes dans les en-têtes de réponse sans le corps de réponse ;
— **POST** is used to inform the server of a new resource ;
— **PUT** is used to store a resource at the location identified by the URI in the request. If the resource already exists, it will be modified ;
— **PATCH** allows the client to modify only a part of the resource ;

— **DELETE** is used to delete the specified resource.

Question 2.2.2: State

The server keeps track of previous requests ?
- ○ True
- ○ False

Question 2.2.3: Wold Wide Web

The World Wide Web is based on this principle of states for :
- ○ work on both computers and smartphones,
- ○ be able to serve a large number of requests,
- ○ encrypt communications.

Question 2.2.4: Presentation of Information

What formats are used to represent structured information (2 answers) :

☐ Ethernet	☐ IPv4	☐ MQTT	☐ JSON
☐ IEEE	☐ IPv6	☐ HTTP	
☐ 802.15.5	☐ UDP	☐ CoAP	
☐ Wi-Fi	☐ TCP	☐ XML	

Question 2.2.5: Scheme

In the URI `https://plido.net/unit/definition.html`, where is the scheme ?

Question 2.2.6: Authority

In the URI `https://plido.net:8080/unit/definition.html`, where is the authority ?

2.3 Publish/Subscribe Model

There are other formalisms than REST. Another formalism, very popular, is broascast oriented by using the "**publish/subscribe**" principle. As we will show in the following, even if the functionalities between these two modes may seem similar to HTTP, the design philosophy is very different : publish/subscribe aims at integrated applications while REST aims at global interoperability.

The publish/subscribe model makes the decoupling between the sender of a message and its recipient. In this paradigm (see figure 2.6 on the following page), there are "Publishers" who produce data or messages and send the message to an entity generally called "Broker". In addition, messages can be classified into "Topics", contents or types, etc. Then, there are subscribers who subscribe to the broker, for example to a given topic, in order to receive the messages that interest them, as shown in the diagram.

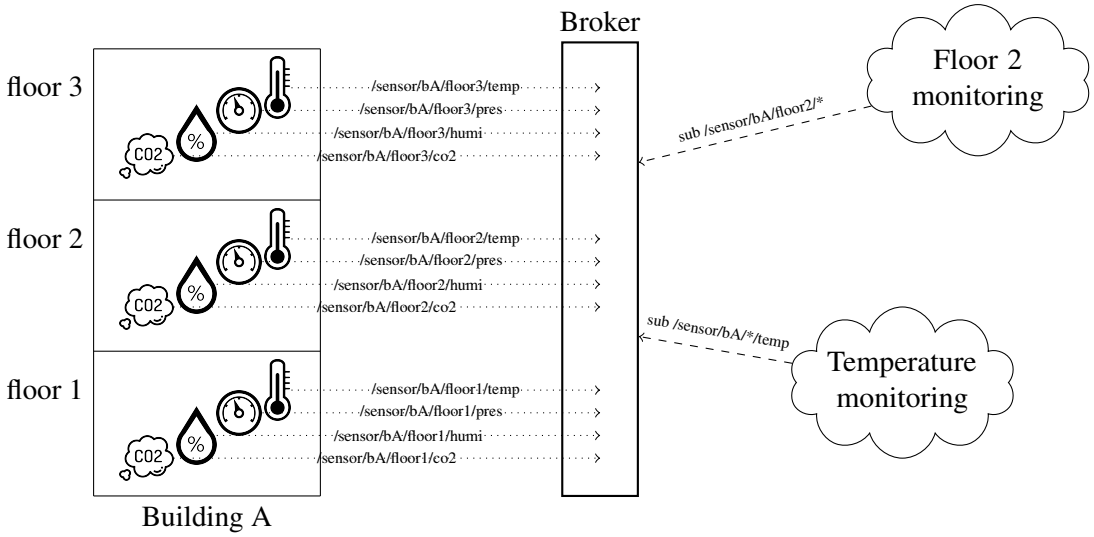


FIGURE 2.6 – Example of MQTT topics.

The broker can then use filters to send only these messages to the subscribers of the concerned topic. There are several Publish-Subscribe protocols such as Message Queuing Telemetry Transport (MQTT), Advanced Message Queuing Protocol (AMQP), Java Messaging Service (JMS) or Extensible Messaging Protocol et Presence (XMPP).

2.3.1 MQTT

MQTT is detailed in the rest of the course because it is very popular for communication between processes, but also in the Internet of Things.

Originally developed by IBM in 1999, for the MQSeries middleware, it became an OASIS standard in 2013, and in 2016 an ISO standard [1].

For example, let's imagine that several sensors are installed, on several floors, in two buildings A and B. Some sensors collect temperature information and others collect humidity information. These sensors can send the data regularly to a central broker.

The data can be classified into different topics which can also be organized hierarchically. For example, the topic /sensor means "all sensor data", `/sensor/buildingA/` means "sensor data only installed in building A". In addition, `/sensor/buildingA/floor3/temperature` could mean "temperature sensor data installed only on the third floor in building A".

Some subscribers may subscribe to messages based on their interest. For example, a subscriber interested only in the humidity data of the whole building B can subscribe to the topic `/sensor/buildingB/*/humidity` and the broker will send only this data to this subscriber.

2.3.2 difference with REST

The main advantages of the publish-subscribe paradigm over the client-server paradigm, as included in REST, are the following :

1. `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.pdf`

— weak coupling between sender and receiver, the broker acts as an intermediary and stores the information ;
— scaling. Data from a source is sent out only once by the source. The broker copies it to all subscribers. In a client/server mode, the data must be sent by the server as many times as the clients request.

The lack of coupling between the sender and the receiver is done in terms of space, time and synchronization. The one who publishes the data has a simplified task. He doesn't have to manage or know who is consuming it, he only has to send it to the broker.

MQTT is very lightweight and designed for low-power devices. It has a very small software footprint and is optimized to work in low-bandwidth environments. This makes MQTT ideal for IoT applications. Even so, the use of TCP and the many, many acknowledgements can be cumbersome for highly constrained devices or networks. A lighter version based on UDP exists for these use cases, but it is not widely used.

Although they are similar, the naming principles of MQTT topics and REST URIs are completely different. Compared to MQTT, the path in the URI has no semantics. It is just meant to be unique. It cannot be used to aggregate multiple information sources. If two sensors publish respectively on the topics `/sensor/buildingA/temperature` and `/sensor/buildingB/temperature`, a subscriber can subscribe to the topic `/sensor/*/temperature` in order to receive all the measurements ; this is impossible with REST : it will be necessary to make as many requests as sensors to get all the measurements.

URIs are simply unique to the world in their construction, whereas MQTT topics are application specific. An MQTT topic can be interpreted differently by two different applications. This does not allow for semantic interoperability. Subscribers must be constructed with knowledge of the topics used by the publishers.

3. Wireshark

Wireshark is going to be our friend in the rest of this book. It will help us to understand the protocols and to analyze the data which will circulate. Unfortunately, in certain cases, we must resort to more rustic tools such as traces in **hexadecimal** [1] (base 16). It is thus necessary to become familiar with these tools. We will do straight away by analyzing simple HTTP requests. If you have access to a computer that can run Wireshark, we recommend that you try to do the manipulations described bellow and answer the questions.

3.1 Installation

The installation of Wireshark is done by going on the eponymous site `https://www.wireshark.org/`, or under Linux by installing the package `wireshark`. This program requires particular rights to access messages coming from the network, you must grant them at the time of the installation.

3.2 Startup

If you launch Wireshark with the right privileges, the welcome window will display the available interfaces, as shown in the figure 3.1 on the next page on Windows. Compared to the reference model of the ISO, these are all the level 2 interfaces present on the computer. It can be a physical card like Ethernet or Wi-Fi or a virtual interface used to communicate internally on the computer.

It is a question of determining which interface to choose. This is not always easy because their names are not always very explicit. The small curves on the left of the name indicate the instantaneous traffic that Wireshark measures. On the diagram, 3 interfaces are active : Ethernet, communication with a virtual machine and an interface called *Indexloopback*. The first one allows to have the communication with the outside and the last one will be very useful during the exchanges between two processes in this machine.

1. `https://en.wikipedia.org/wiki/Hexadecimal`

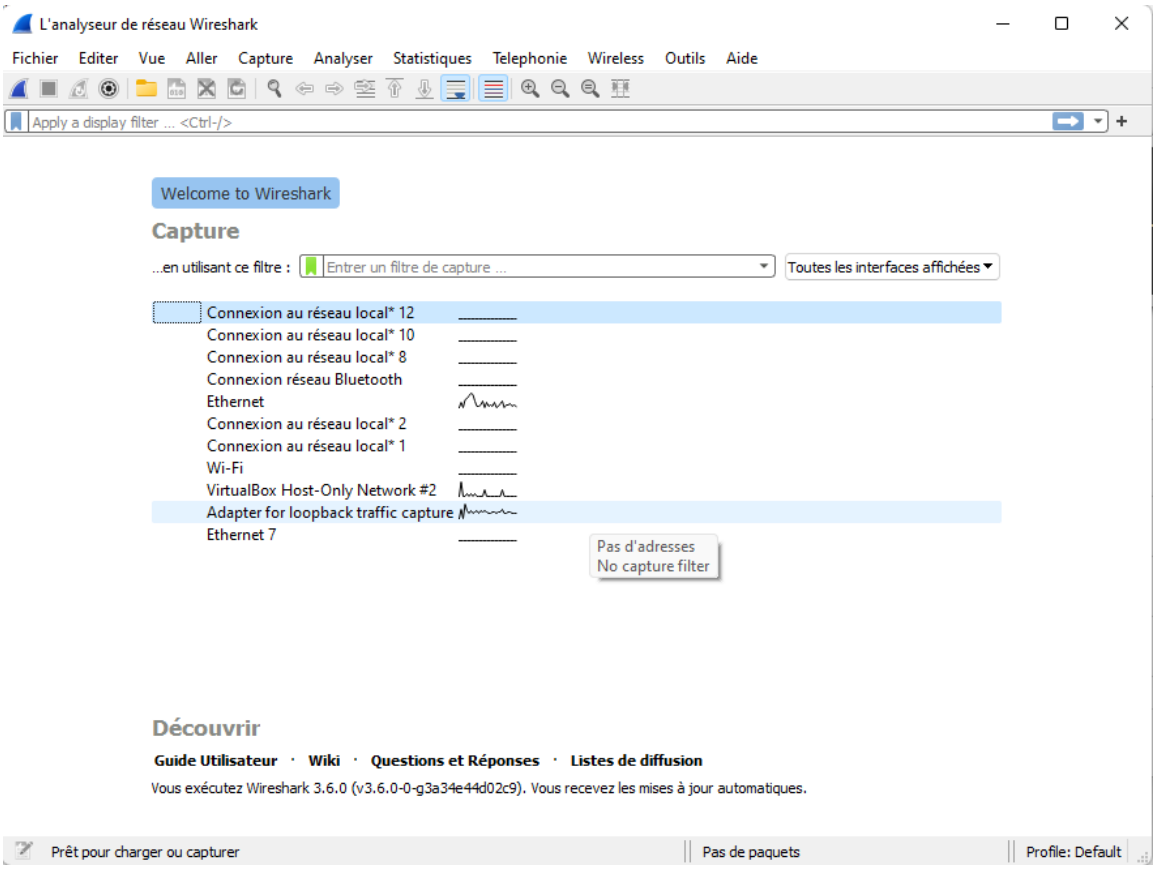


FIGURE 3.1 – Wireshark opening

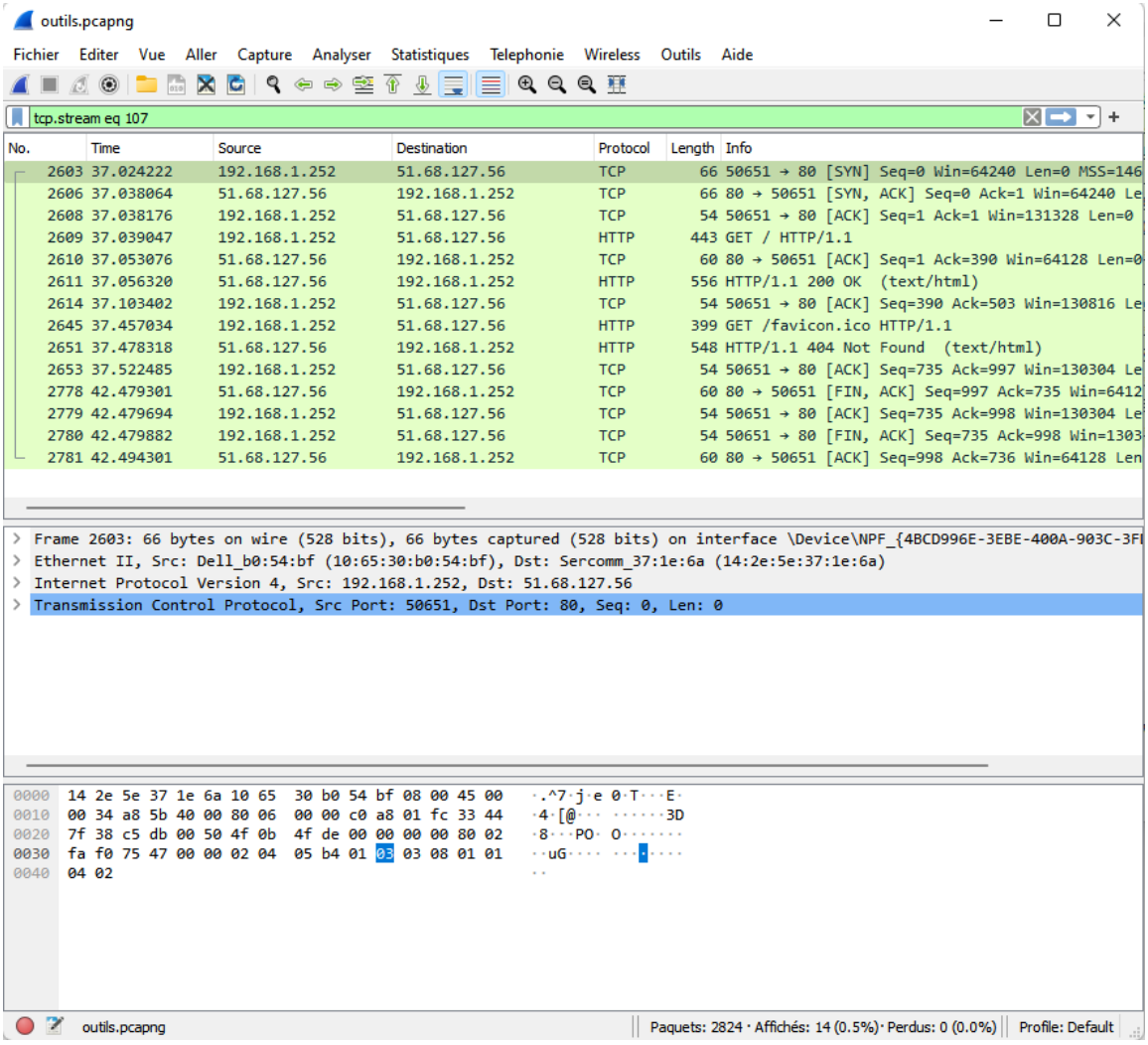


FIGURE 3.2 – Traffic Capture

3.3 Capture

By clicking on the name of the interface giving access to the external network (**Ethernet** in our case), the window splits into 3 parts, as shown in the figure 3.2.

The Wireshark screen is divided into 3 parts :

— at the top, scrolls the frames that are captured on the network, each protocol has a dedicated color for easy identification :
  — the captured frame number, this is an information added by Wireshark,
  — the time of capture of the frame. This information is also added by Wireshark,
  — the IP address (IPv4 or IPv6) of the machine originating the packet,
  — the IP address (IPv4 or IPv6) of the machine receiving the packet,
  — the highest level protocol contained in the frame. In our case, this can be TCP if the TCP message does not contain data, as in the case of connection opening, or certain acknowledgements. We also see the messages HTTP which are of course encapsulated

in TCP,
  - the size in bytes of the frame captured by Wireshark,
  - Finally Wireshark provides a summary of the content of the frame, to understand what is happening on the network. In the screenshot, we can see for the messages, the GET requests or the notifications ;
- if a frame is selected in the list, it appears in the middle area with the protocol stack. The content of each of these protocols can be detailed by clicking on the small triangle on the left ;
- the bottom window gives the equivalent in hexadecimal. The highlighted parts correspond to the fields selected in the middle window. Note that the information is found both in hexadecimal and in the character American Standard Code for Information Interchange (ASCII), which helps in reading when looking for a specific value.

Question 3.3.1: Column one

In the first column :
- ○ The frame number assigned by Wireshark upon reception
- ○ The frame number is read directly in the Ethernet frame

Question 3.3.2: 2nd column

In the second column :
- ○ The time of reception by Wireshark
- ○ The sending time of the frame

Question 3.3.3: The third and fourth columns

In the third and fourth columns :
- ○ The Ethernet addresses of the hosts.
- ○ Only the IPv4 addresses of the machines.
- ○ IPv4 or IPv6 addresses of machines.

Question 3.3.4: The fifth column

In the fifth column :
- ○ The application protocol (level 7).
- ○ The last (higher level) recognized protocol.
- ○ The level 4 protocol (here TCP or UDP).

Question 3.3.5: The sixth column

In the sixth column :
- ○ The size in bits of the frame.
- ○ The size in bytes of the frame.

Hello!

This is the default web page for this server.

It just display a simple page. You will find a more sophisticated page here

Question 3.3.6: The seventh column

In the seventh column :
- A summary of the information carried by the higher-level protocol.
- IPv4 options.
- The ASCII content of the highest level message.

That's a lot of traffic, so we'll limit what is displayed by adding a filter to a particular recipient. The site `tools.plido.net` at IPv4 address `51.68.127.56`. In the window currently showing *Apply a display filter.* type the following instructions :

```
ip.addr==51.68.127.56
```

don't forget the double ==. The window should turn green when everything is typed indicating that the filter syntax is correct. When you press enter, the window should be empty.

3.3.1 Web traffic analysis

In the address bar of your favorite browser, type the following URL :

```
http://outils.plido.net
```

and the Web page indicated figure 3.3.1 must appear.

Wireshark allowed to visualize the traffic exchanged between the computer and the Web server. The traffic should be similar to the one in the figure 3.2 on page 38. The figure is obtained by selecting the menu *Statistics/Flow Graph* and checking *Limit to Display Filter*. It is a little more readable because it represents the exchanges in the form of time diagrams.

Three phases can be distinguished :
- TCP connection opening with the emission of three TCP messages ;
- the data transfer phase :
  - the client sends an HTTP GET request to the server to request the resource from the root (/),
  - the server acknowledges the message at the TCP level to indicate that it has been received.
  - the server sends the answer to the previous request and specifying the status (200 : OK) and the content is formatted in HTML.

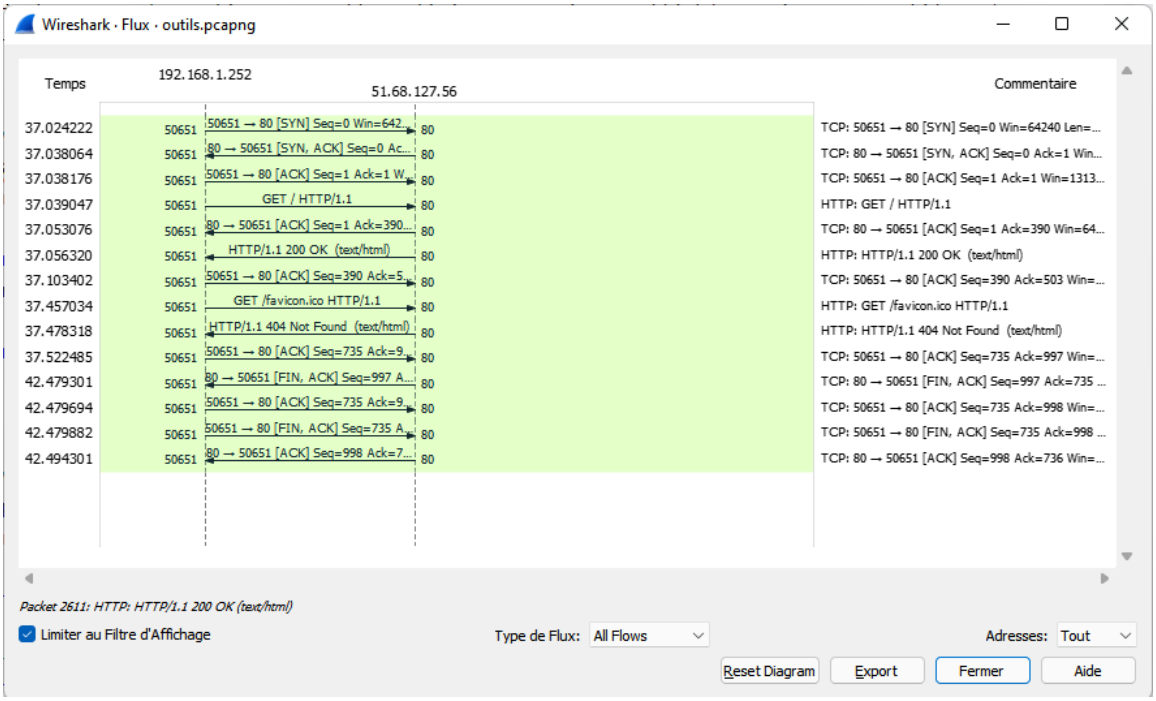


FIGURE 3.3 – Time diagram of exchanges.

  - the client acknowledges this message at the TCP level,
  - the client sends a new HTTP GET request to obtain the resource `/facicon.ico`
  - the server answers that the resource does not exist (`404 : Not Found`). This request implicitly acknowledges the previous message.
  - the client acknowledges the server's response at the TCP level.
- the server ends the connection after 5 seconds of inactivity. The closing is done by exchanging 4 TCP messages.

Question 3.3.7: HTTP notification code

In the following trace, we saw that the server responded to client requests with a 3-digit number. Using the RFC 7231, can you assign the left digit to a category of notifications :

0 ☐	☐ Redirection
1 ☐	☐ Error on the server side
2 ☐	☐ Error on the client side
3 ☐	☐ Unassigned
4 ☐	☐ Success
5 ☐	☐ Information

3.3.2 Analysis of HTTP requests

The version 1.1 of the protocol HTTP is specified by the RFC 7230. We will look at a small description in English of the architecture and the formats of the messages.

```
2.  Architecture

  HTTP was created for the World Wide Web (WWW) architecture and has
  evolved over time to support the scalability needs of a worldwide
  hypertext system.  Much of that architecture is reflected in the
  terminology and syntax productions used to define HTTP.

2.1.  Client/Server Messaging

  HTTP is a stateless request/response protocol that operates by
  exchanging messages  across a reliable transport- or
  session-layer "connection".  An HTTP "client" is a
  program that establishes a connection to a server for the purpose of
  sending one or more HTTP requests.  An HTTP "server" is a program
  that accepts connections in order to service HTTP requests by sending
  HTTP responses.
```

Question 3.3.8: Standardization body.

Which standards organization published this document ?

- Microsoft
- ISO
- IEEE
- IETF

```
  The terms "client" and "server" refer only to the roles that these
  programs perform for a particular connection.  The same program might
  act as a client on some connections and a server on others. [...]

  Most HTTP communication consists of a retrieval request (GET) for a
  representation of some resource identified by a URI.  In the simplest
  case, this might be accomplished via a single bidirectional
  connection (===) between the user agent (UA) and the origin
  server (O).

            request   >
       UA ======================================= O
                                   <   response

  A client sends an HTTP request to a server in the form of a request
  message, beginning with a request-line that includes a method, URI,
  and protocol version, followed by header fields
  containing request modifiers, client information, and representation
  metadata, an empty line to indicate the end of the
  header section, and finally a message body containing the payload
  body.

  A server responds to a client's request by sending one or more HTTP
  response messages, each beginning with a status line that includes
```

```
the protocol version, a success or error code, and textual reason
phrase possibly followed by header fields containing
server information, resource metadata, and representation metadata,
an empty line to indicate the end of the header
section, and finally a message body containing the payload body.

The following example illustrates a typical message exchange for a
GET request on the URI "http://www.example.com/hello.txt":

Client request:

  GET /hello.txt HTTP/1.1
  User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
  Host: www.example.com
  Accept-Language: en, mi

Server response:

  HTTP/1.1 200 OK
  Date: Mon, 27 Jul 2009 12:28:53 GMT
  Server: Apache
  Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
  ETag: "34aa387-d-1568eb00"
  Accept-Ranges: bytes
  Content-Length: 51
  Vary: Accept-Encoding
  Content-Type: text/plain

  Hello World! My payload includes a trailing CRLF.
```

Question 3.3.9: Formatting HTTP messages

Do HTTP headers have a fixed size (you can check the RFC 7231 which gives indications on the protocol) ?

- the header is one line of 80 characters.
- a blank line separates the header from the content. The header can contain as many lines as necessary.

Question 3.3.10: HTTP Header Options

How are the optional lines in the header constructed ?

- keyword : values
- unformatted text
- keyword : data length : values

3.3.3 Protocol stack analysis

The frame containing the HTTP GET request allows to visualize the protocol encapsulation defined by the reference model of the ISO. In Wireshark, by clicking on the frame, we can see it disassembled and in hexadecimal in both windows as shown in the figure 3.4 on the next page.

The second window shows the protocol stack, inverted with respect to the classical representations

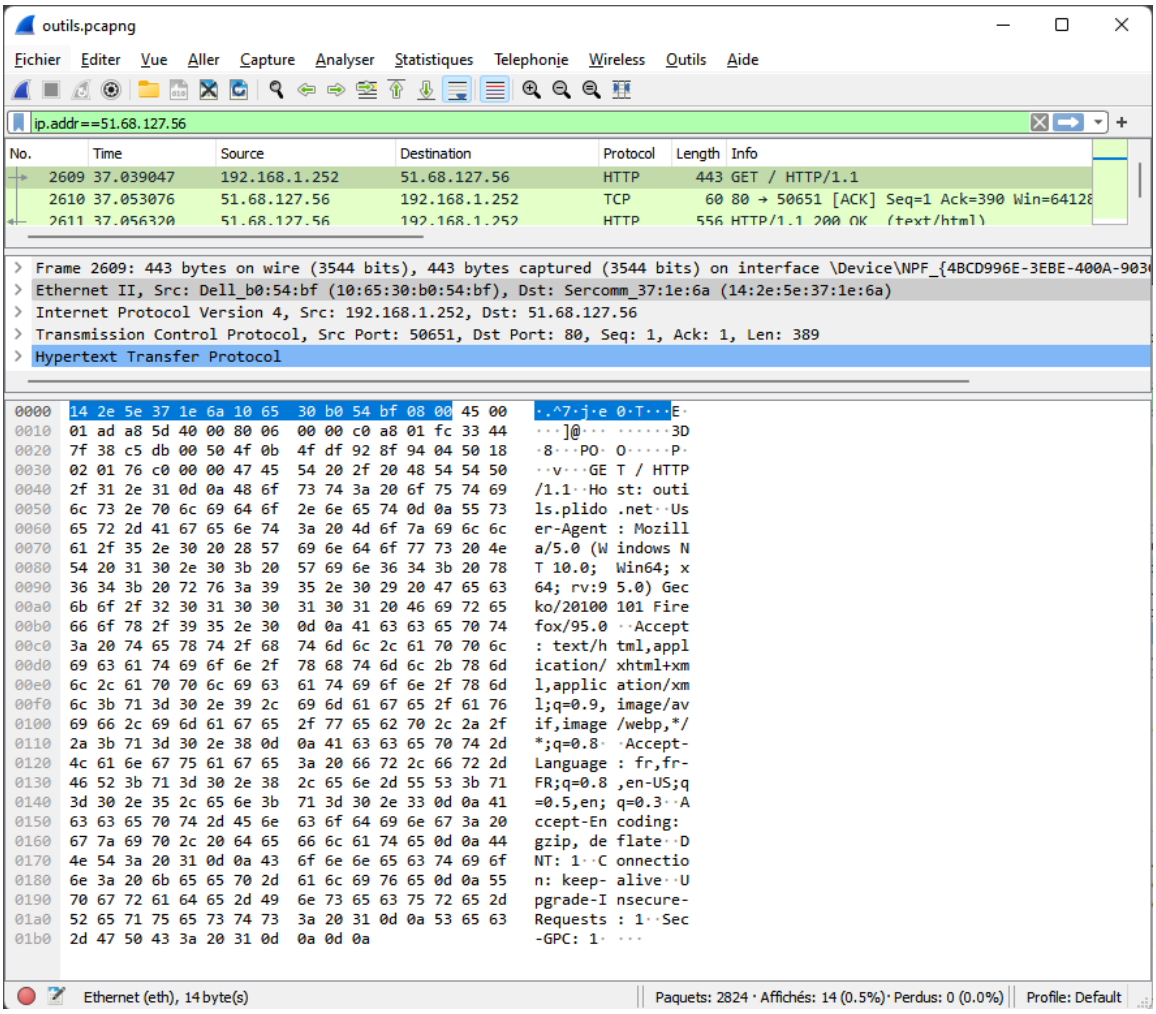


FIGURE 3.4 – Content of the frame carrying the HTTP GET request

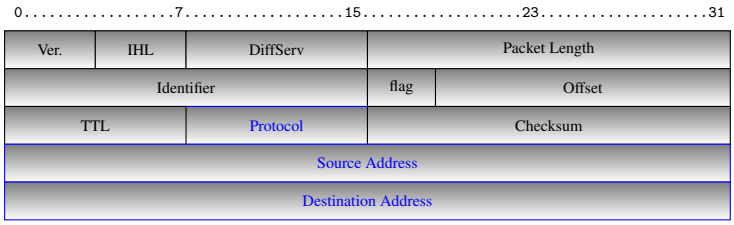


FIGURE 3.5 – Format of an IPv4 header

(cf. figure 2.3 on page 28), but corresponding to the order of encapsulations in the frame. How Wireshark could arrive at such a result.

Ethernet

Wireshark receives a frame from the network **Ethernet** or **Wi-Fi** [2]. The format of an Ethernet frame is defined by the standard **IEEE 802.3**. The header contains three fields :

— 6 bytes for the MAC address of the destination,
— 6 bytes for the source address,
— 2 bytes for the higher level protocol. Thus the value `0x0800` indicates IPv4 and `0x86dd` IPv6.

Following the principle of the ISO reference model, the addresses are those of adjacent nodes, i.e. connected to the same Ethernet or Wi-Fi network.

In our case, the top level protocol is therefore an IPv4 packet and Wireshark can continue to analyze this. Formally it is data from the Ethernet frame, but it can be understood as an IPv4 packet.

Question 3.3.11: My address

In the example, figure 3.4 on the preceding page, what is the Ethernet address of the machine sending the frame ?

IPv4

The IPv4 packet format defined in the RFC 791 has changed very little since its publication in 1981. Figure 3.5 shows this format. Without going into detail, the fields :

— Addresses `source` and `destination` will contain the IPv4 addresses on 32 bits of the end equipment. Intermediary equipments, called routers are in charge of copying the packet to its destination.
— The field `protocol` designates the upper layer, the value 6 corresponds to **TCP** and 17 to **UDP**.

Question 3.3.12: hop by hop

Does the Ethernet address `14:2e:5e:37:1e:6a` found in the packet 3.4 on the preceding page correspond to the Ethernet address of the recipient of the packet ? What does it correspond to ?

2. For the Wi-Fi network, it transforms the format into that of an Ethernet frame for a more compact display

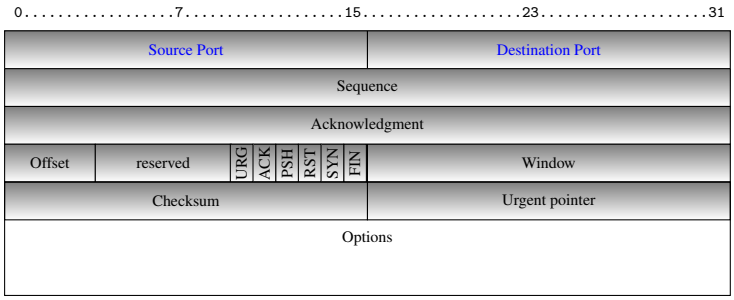


FIGURE 3.6 – Format of a TCP header

TCP

Wireshark, from the protocol field 0x06, determines that the IP data following the header is a TCP message, so it can continue disassembling the frame. The format of the TCP header is given in figure 3.6. The port numbers determine which application is used. If a client is going to use any number (50651 in the figure 3.4 on page 44), the servers will use numbers known by all. Thus, the Web servers will be assigned the value 80. They can choose others, as we saw when building the URL.

Wireshark knows this list of well-known port numbers and can continue to parse the frame as HTTP.

Without going into details, we can also notice a series of binary values which are used for example to open or close a TCP connection. If we go back to the opening phase of the connection (cf. figure 3.3 on page 41), the connection is opened by :

— the transmission by the client of a TCP message with the SYN bit set,
— the transmission by the client of a TCP message with the SYN bit set,
— the client responds by returning a message with the ACK bit set.

These three messages, which do not contain any data, are used to synchronize the initial value of the `sequence` field at each end of the connection.

Question 3.3.13: Closing the connection

Using the figure 3.4 on page 44 or your Wireshark captures, what are the messages involved in closing the connection ?

3.4 Do it yourself

A Web server can be written in Python thanks to the module **Flask**. The program `simple_server.py` allows to create a Web server on its computer.

Listing 3.1 – simple_server.py

```
1 from flask import Flask
  app = Flask("My First Web Server")
3
  @app.route('/', methods=['GET'])
5 def hello_world():
      return "Hello World"
7
```

```
app.run(host="0.0.0.0", port=8080)
```

This script requires some explanation :

— The import line 1 includes the `Flask` object from the flask module.
— On line 2 an instance of a Flask object, i.e. a web server, is created. A name is associated to it for debugging purposes.
— Line 4 contains the most delicate part of the script. `@` is a decorator that is used in python to add properties to a function. Here we associate a URI path and a REST method to the function which is then defined. This way, when the Flash server receives a GET request on this URI path, it will call the function `hello_word`.
— The function `hello_word` simply returns a text that the browser will display.
— the server is launched, line 8, by calling the method `run`. It will wait on all interfaces (wildcard address `0.0.0.0`) and on port 8080.

To launch the server, you must first install the `Flask` module with `Indexpip`.

```
# pip3 install Flask
Collecting Flask
  Downloading Flask-2.0.2-py3-none-any.whl (95 kB)
     |                                        | 95 kB 4.3 MB/s
Collecting Jinja2>=3.0
  Downloading Jinja2-3.0.3-py3-none-any.whl (133 kB)
...
```

Once the package is installed, simply run the program :

```
# python3.9 simple_server.py
 * Serving Flask app 'My First Web Server' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production
   deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production
   deployment.
 * Running on http://192.168.1.53:8080/ (Press CTRL+C to quit)
127.0.0.1 - - [14/Dec/2021 21:06:55] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [14/Dec/2021 21:06:59] "GET /favicon.ico HTTP/1.1" 404 -
```

Question 3.4.1: loopback

What URI should you enter in your browser to access this server locally.

Question 3.4.2: Server name

Using Wireshark, you can determine in the response the values of the HTTP options `IndexContent-Type` and `Server`.

4. Modbus

4.1 Introduction

Modbus appeared in 1979 at a time when the Internet did not exist yet ! It is still very popular in the industry. Originally Modbus was built on a serial bus **RS-485** which connected different equipments called (see figure 4.1) :

— secondary or slaves and

— a primary, also called master, which manages communications.

Each secondary has a unique number or address. The addresses are between 1 and 247. The primary does not need an address since all communication is with it.

Youtube

The primary sends a request to a secondary and the secondary replies to the primary. Direct communication between two secondaries is not possible.

4.1.1 Registers

A Modbus device can take two types of data through registers :

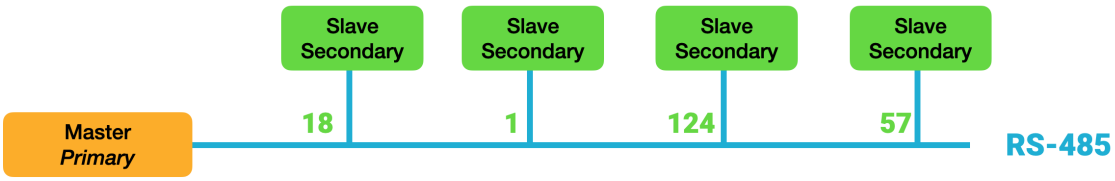


FIGURE 4.1 – Modbus wired architecture

FIGURE 4.2 – Modbus frame

— relays that can take a binary value "on" or "off". If the primary can change the state and, of course, read it, it is called a ***coil***. If the binary value can only be read it is a ***discrete input***.
— 16-bit registers. They are used to represent a value such as an electric current, a temperature, a rotation speed,... Similarly, if one can only read the value at is called an ***input register*** otherwise, if it can also be modified by the primary, it is called an ***holding register***.

A Modbus device can have up to 10 000 registers of these four categories.

4.1.2 Protocol

Modbus is a query/response protocol. The primary sends a request to the address of a device to read or write one of its registers.

A Modbus frame is a sequence of characters starting with a byte with the address of the secondary followed by a command or function code specific to each register category :

— 1 to read a coil,
— 2 to read a discrete input,
— 3 to read a holding register,
— 4 to read an input register,
— 5 to write a coil,
— 6 to write a holding register.

The rest of the frame contains the data and then a Cyclic Redundancy Check (CRC) to validate that there is no transmission error in the frame. The data part can be different in the request and the response. For example, to read a holding register, the request contains the address of the first register to be read and the number of registers to be read and the response contains the number of data transmitted followed by their values. To write to a register, the data in the frame will be the address of the register and the data to write.

4.1.3 Example : XY-MD02

Let's take a closer look at a concrete example. We will use a temperature and humidity sensor, the **XY-MD02** (see figure 4.3 on the next page) whose specifications are easily accessible via an Internet search.

The green part is composed of four connectors whose meaning is indicated on the label. The two left connectors constitute the bus named `A+` and `B-` and the two right connectors allow to supply the equipment with a voltage between 5V and 30V.

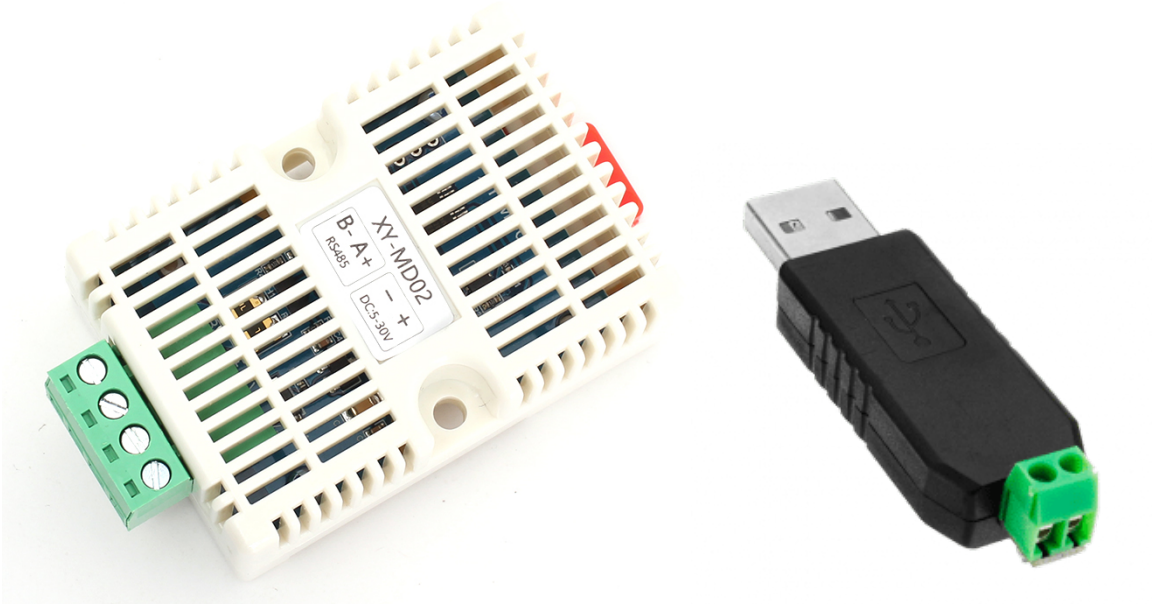


FIGURE 4.3 – XY-MD02 and USB/RS-485 Adapter

Youtube

An adapter **USB/RS-485** (see figure reffig-XYMD02) is connected to a computer. It contains the two connectors A+ and B- of the RS-846 bus. The computer plays the role of primary which will interrogate the temperature sensor.

The program **QModMaster** [1] (see figure 4.4 on the facing page) allows to query or write the registers of the secondaries. In the left window you can access the registers of a secondary. The right window shows the traffic on the RS-485 bus.

In order for the primary to be able to connect to the secondary, in addition to the name of the serial port (here `COM3`), it needs several pieces of information that can be found in its documentation :

— the transmission speed on the bus (here 9 600 bit/s) and the coding of the transmitted characters (here 8 bits without parity bit and one stop bit) [2]
— the address of the secondary on the bus.

The documentation also gives the nature of the registers and their coding. The table 4.1 on the next page takes up the definition of *Input Registers*. These are registers that can only be read. The specification indicates that the temperature is stored in register 1 over a length of 2 bytes, i.e. the entire length of the register.

The documentation indicates that the secondary has the address `0x01` on the RS-485 bus. It only remains to send a Modbus request to read this register. The trace window on the right on the figure 4.4 on the facing page gives the exchanges. We will analyze the last two lines. The first one

1. `https://sourceforge.net/projects/qmodmaster/`
2. `https://en.wikipedia.org/wiki/8-N-1`

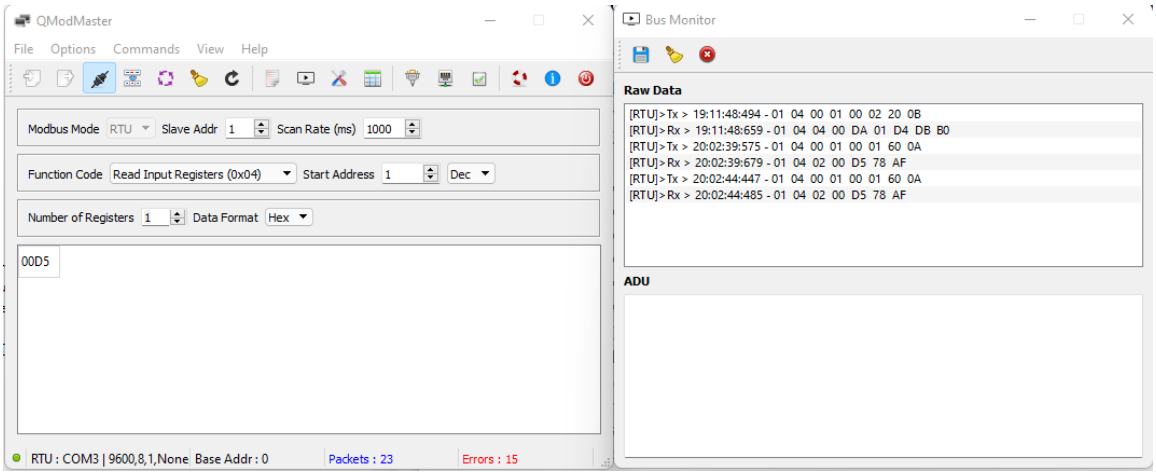


FIGURE 4.4 – QModMaster with message dump

Register Type	Register Address	Register Contents	Bytes
Input register	0x0001	Temperature	2
	0x0002	Humidity	2

TABLE 4.1 – Input Register of an XY-MD02

indicates the content of the request and the last one the answer of the secondary :

```
01 04 00 01 00 01 60 0A
01 04 02 00 D5 78 AF
```

The request starts with the address of the secondary (`01`), then the action (`04`) to read an *Input Register*, followed by the address of the register (`00 01`) and the number of registers to read (`00 01`). The request ends with the CRC validating the integrity of the frame (`60 0A`).

The response also contains the address of the secondary (`01`) and the action, followed by the size of the response in bytes (`02`) and the requested result (`00 D5`).

Question 4.1.1: Humidity

Looking at the exchanges in figure 4.4 what is the measured value for humidity ?

It remains to be able to interpret this value. The documentation indicates that the value is in tenths of degrees and that the unit is Celsius. By converting **00 D5** into decimal, we obtain 213, that is 21.3°C.

Question 4.1.2: Evolution of temperatures

Looking at the exchanges in figure 4.4 what is the evolution of the temperature over time ?

In summary, we can see that it would be very difficult to operate the equipment without documentation to know : the speed of the RS-485 bus, the address of the secondary, the addresses of the

Register Type	Register Address	Register Contents	Bytes
Holding register	0x0101	Device Address	2
	0x1202	Bit rate : • 0 : 9600 • 1 : 14400 • 2 : 19200	2
	0x0103	Temperature correction -10°C - 10°C	2
	0x0104	Humidity correction -10%RH - 10%RH	2

TABLE 4.2 – Holding Register of an XY-MD02

registers used and their contents and the coding of the information in the registers and the units used. Therefore, there is a low degree of interoperability, the two entities must agree on a large number of parameters.

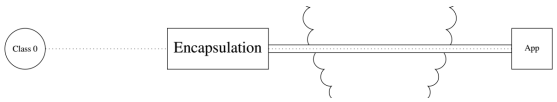
Question 4.1.3:

What is the moisture content at the time of measurement ? The documentation states that it is a percentage with an accuracy of one tenth of a percent.

We could communicate with the object using the default parameters. But to insert it in a bus, it is necessary to be able to modify certain parameters. The baud rate and the byte coding must be the same, the secondaries cannot have the same address on the bus.

The XY-MD02 has also *holding register* allowing to parameterize it as shown in the table 4.2

4.1.4 IP Gateway



It is possible to extend the scope of a Modbus network by adding an IP gateway. This corresponds to the third method of interconnection of the figure 1.3 on page 22. The gateway, connected on the bus where the secondaries remain connected, has an IP address. The primary opens a TCP connection with the gateway and sends its requests. The gateway copies the data on the bus. Conversely, the responses of the objects are returned to the gateway which sends them to the primary using the TCP connection. The figure 4.5 on the next page illustrates the exchanges. We note the opening of TCP connection which is done at the starting of the primary which remains active for all the exchanges. We can also notice that the TCP messages are acknowledged. The figure 4.6 on the facing page shows the fields common to the format on the RS-485 bus and in IP packets.

As in the previous example of the temperature sensor, the specifications of the **electric counter** are necessary to understand the meaning of the usable registers. The counter encodes its values on 32 bits floating numbers whose encoding is specified by the **IEEE 754** standard. These values on 32 bits must be coded on two consecutive registers, hence the incrementation of 2 in 2 that we find on

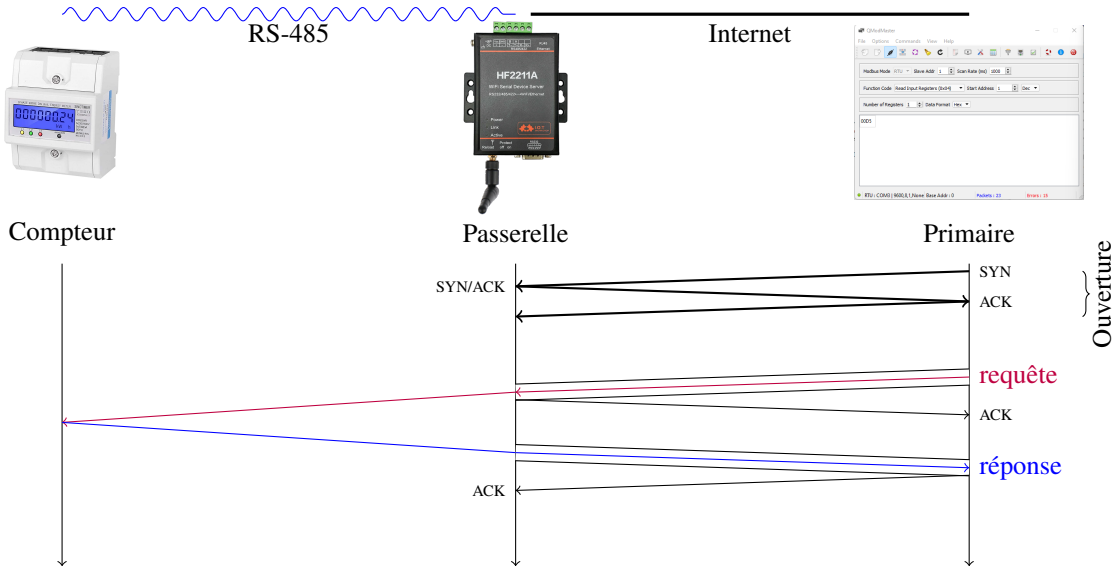


FIGURE 4.5 – Gateway between the Internet and Modbus.

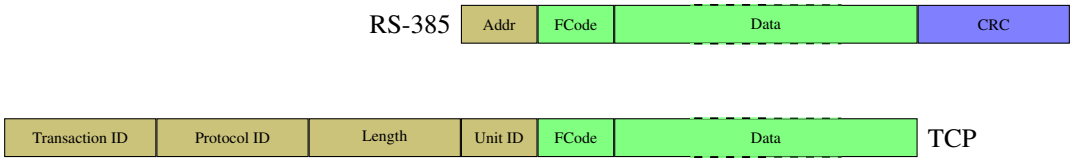


FIGURE 4.6 – Modbus messages on RS-485 bus and on IP/TCP

Register Type	Register Address	Register Contents	Unité	Format
Input register	0x0000	**Voltage** phase A	V	IEEE 754
	0x0002	Voltage phase B	V	IEEE 754
	0x0004	Voltage phase C	V	IEEE 754
	0x0008	**Intensité** phase A	A	IEEE 754
	0x000A	Intensité phase B	A	IEEE 754
	0x000C	Intensité phase C	A	IEEE 754
	0x0010	**Puissance** Totale	KWh	IEEE 754
	0x0012	Puissance phase A	KWh	IEEE 754
	0x0014	Puissance phase B	KWh	IEEE 754
	0x0016	Puissance phase C	KWh	IEEE 754
	0x0036	**Fréquence**	Hz	IEEE 754

TABLE 4.3 – some *Input Register* of the electric meter

the table 4.3, the counter being able to measure three electrical phases.

Wireshark can capture a request having circulated on the Ethernet network, primary side (cf. figure 4.7 on the facing page).

```
0000  98 d8 63 62 29 49 10 65 30 b0 54 bf 08 00 45 00   ..cb)I.e0.T...E.
0010  00 34 db ef 40 00 80 06 00 00 c0 a8 01 fc c0 a8   .4..@...........
0020  01 57 e2 c9 01 f6 16 90 37 98 5d 57 a0 fa 50 18   .W......7.]W..P.
0030  02 01 84 ca 00 00 00 0a 00 00 00 06 1c 04 00 00   ................
0040  00 02 ..
```

We find the encapsulations of the Ethernet, IP and TCP protocols, followed by the TCP data. They consist of three fields which do not exist in the request circulating on the RS-485 bus :

— the transaction number on two bytes incremented at each request,
— the protocol version on two bytes,
— the length in bytes of the transaction.

The following fields are identical to those of the frame on the RS-485 bus :

— the address of the secondary on one byte, here 0x1c or 28,
— the nature of the request : 0x04 to read an *input register*,
— the register to be read on two bytes, here 0x0000 corresponding to the voltage on phase A,
— the number of registers to read, here 2 to obtain the 32 bits of the value.

The primary receives the following response :

```
0000  10 65 30 b0 54 bf 98 d8 63 62 29 49 08 00 45 00   .e0.T...cb)I..E.
0010  00 35 c9 75 00 00 40 06 2c aa c0 a8 01 57 c0 a8   .5.u..@.,....W..
0020  01 fc 01 f6 e2 c9 5d 57 a0 fa 16 90 37 a4 50 18   ......]W....7.P.
0030  44 70 e1 6e 00 00 00 0a 00 00 00 07 1c 04 04 43   Dp.n...........C
0040  69 9e 4a                                         i.J
```

After the protocol encapsulations of Ethernet, IP and TCP, we find the following data :

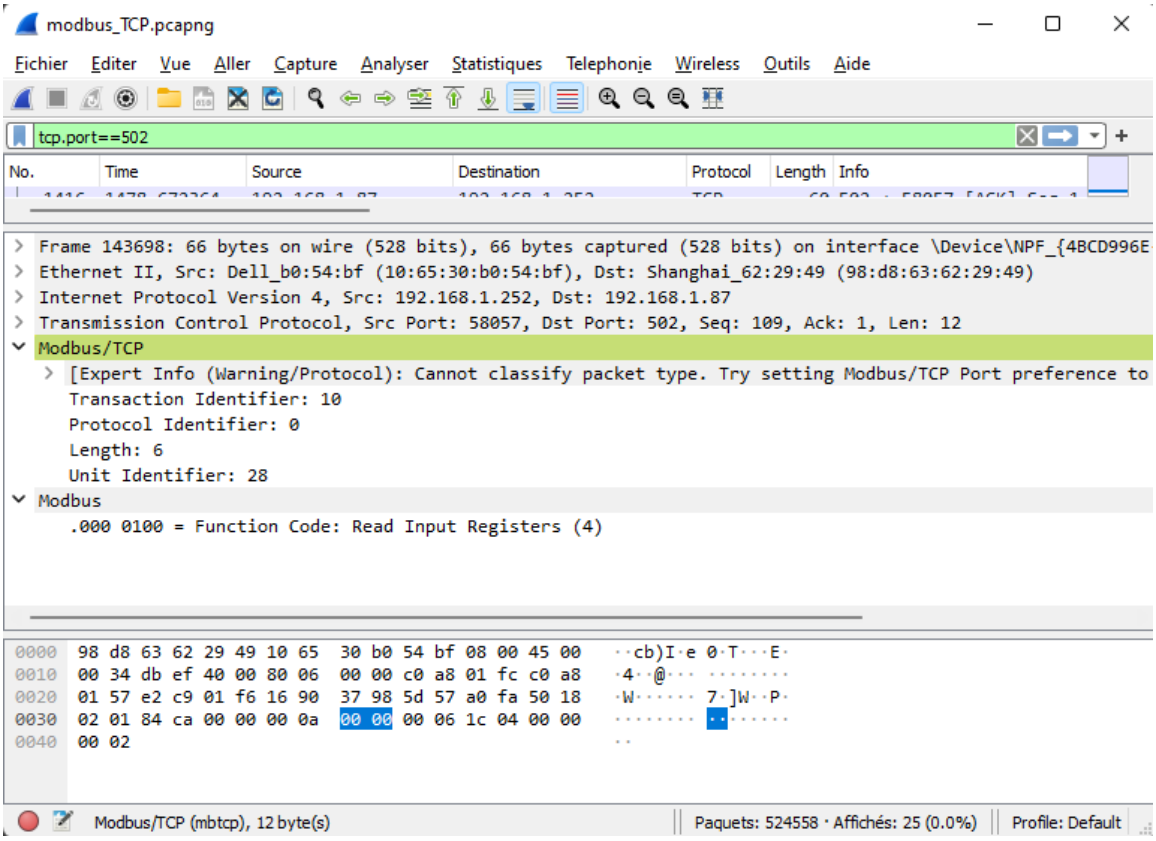


FIGURE 4.7 – Capture with **Wireshark** of a frame containing a Modbus message.

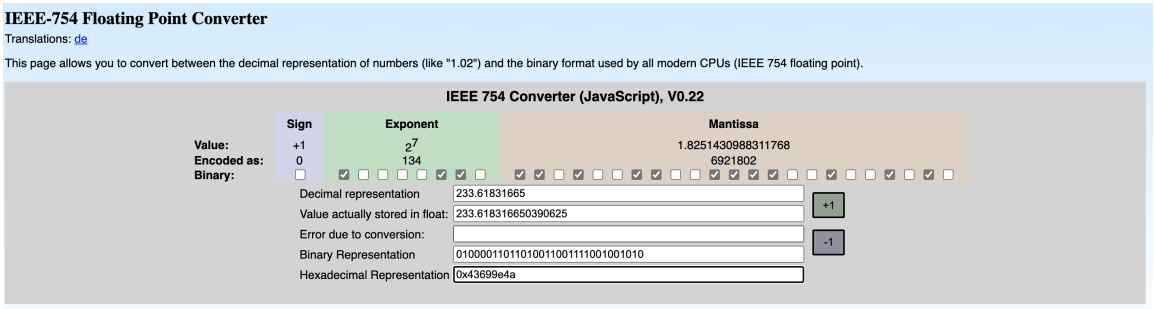


FIGURE 4.8 – Conversion of a floating number

— the transaction number that corresponds to the one used in the previous request. This makes it possible to make the link between the two messages that could have been lost in case of packet loss on the Internet network,
— the protocol version,
— the length of the response, here 7 bytes,
— the address of the secondary school that responded, here 28,
— the nature of the request,
— the number of bytes returned, here 4,
— and the value of the two registers `0x43699e4a` which corresponds to a floating-point number as represented by the IEEE 754 standard. There are many sites on the Internet that allow the conversion of footnoteurl`https://www.h-schmidt.net/FloatConverter/IEEE754.html`. As shown in figure 4.8, we obtain the value 233.61831665which corresponds well to a voltage offered by an electrical network.

Question 4.1.4: Modbus TCP request

Let the given exchange figure 4.9 on the facing page correspond to a Modbus request and a response. What is the port number used by Modbus TCP.

Question 4.1.5: Modbus TCP request

Continuing the traffic analysis, which register value is requested.

Question 4.1.6: Modbus TCP response

By analyzing the following packet, how can we verify that the response can match the previous request.

Question 4.1.7: Modbus TCP response

What value is returned. Is this consistent ?

```
0000  98 d8 63 62 29 49 10 65 30 b0 54 bf 08 00 45 00  ..cb)I.e0.T...E.
0010  00 34 db f3 40 00 80 06 00 00 c0 a8 01 fc c0 a8  .4..@...........
0020  01 57 e2 c9 01 f6 16 90 37 b0 5d 57 a1 14 50 18  .W......7.]W..P.
0030  02 01 84 ca 00 00 00 0c 00 00 00 06 1c 04 00 36  ...............6
0040  00 02                                            ..
```

```
0000  10 65 30 b0 54 bf 98 d8 63 62 29 49 08 00 45 00  .e0.T...cb)I..E.
0010  00 35 8b 15 00 00 40 06 6b 0a c0 a8 01 57 c0 a8  .5....@.k....W..
0020  01 fc 01 f6 e2 c9 5d 57 a1 14 16 90 37 bc 50 18  ......]W....7.P.
0030  44 70 f1 f0 00 00 00 0c 00 00 00 07 1c 04 04 42  Dp.............B
0040  47 e9 5b                                         G.[
```

FIGURE 4.9 – Capture to be studied.

5. ARCHITECTURE FOR IOT

5.1 Introduction

The objects are characterized by a limited processing capacity and by a reduced energy consumption to preserve the autonomy imposed by a battery power supply. However, the most consuming activities for an equipment are the transmission and reception of data. To maximize the autonomy of equipment, it is necessary to review all the protocols, but by modeling them on existing architectures to ensure compatibility.

Youtube

The figure 5.1 on the next page shows a number of protocol adaptations, at different layers of the model, capable of adapting to the characteristics of the constrained objects. In the following chapters, we will come back to these technologies, starting from the data representation and going to the lower layers.

5.2 Topologies

Networks for the Internet of Things can be divided into two categories : **mesh** and **star** topologies.

Mesh Networks

Mesh networks, such as the IEEE 802.15.4 family, are an adaptation of a Wi-Fi access protocol to preserve energy. The transmission range is limited to 50 meters to limit energy consumption, and therefore messages must be relayed by other nodes to reach their destination.

The data rate is a few hundred kilobits/s and the frame size is a few hundred bytes.

These networks are good at carrying IoT data, but the routing protocol, as well as frame relaying, consumes the objects' energy.

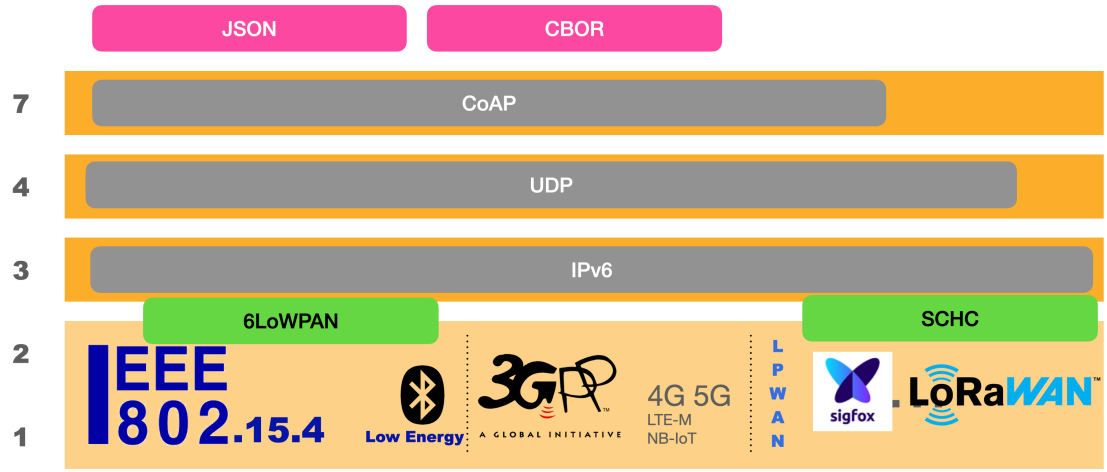


FIGURE 5.1 – IoT protocol stack

Star Network

The topologies in **star** do not require such routing mechanisms. All communications are with a central point that relays information to the destination.

The progress made in signal processing allows to extend the transmission range at low power. This family of networks is called low power wide area networks (LPWAN) like **Sigfox**, **LoRaWAN**, or even on the cellular side with evolutions of the **4G** standard and a more complete integration in **5G**. The RFC 8376 gives, in English, an overview of these techniques.

With a transmission power of 25 mW, it is possible to communicate over a distance of 3 km in an urban environment and 20 km in a clear environment. The LPWANs are compatible with class 0 devices because they do not require the installation of an IP stack. The figure below describes a typical architecture for LPWANs.

The device sends raw data over the radio network. The radio signal is picked up by one or more radio gateways, and the frame is sent to a network gateway (LoRaWAN Network Server (LNS) for networks **LoRaWAN**, and Service Capability Exposure Function (SCEF) for networks 3GPP).

The owner of the device has associated the device with a connector in the LPWAN Network GateWay (NGW) which can be a URL, a Message Queuing Telemetry Transport (MQTT) broker address or a web socket. When the device sends data, it is connected to the application through this tunnel.

Some technologies such as LoRaWAN or Sigfox use unlicensed bands, imposing a duty cycle of 0.1 to 10 percent depending on the channel to ensure fairness between nodes, thus preventing one device from monopolizing the transmission channel. Since this restriction also applies to the provider's antenna, communication between the network and the device is significantly limited.

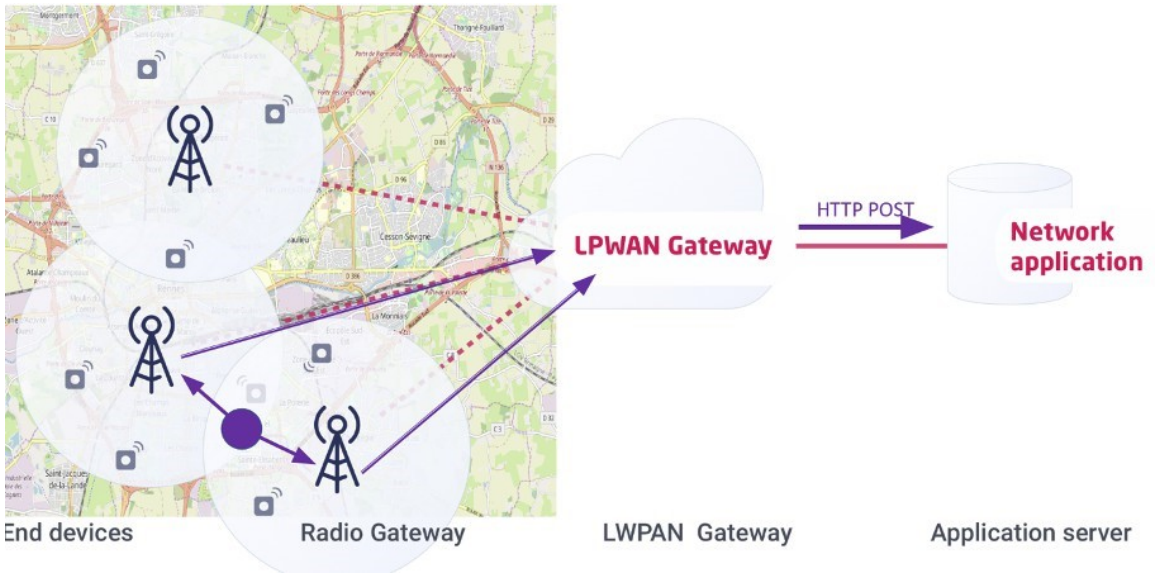


FIGURE 5.2 – Simplified LPWAN architecture

5.3 Layers 1 and 2

Concerning layer 2, the goal is to save energy during transmissions. We can already say goodbye to Ethernet because it would require the use of wired infrastructure and therefore we could not place the objects where we want, especially if they move. The communications by radio waves are privileged.

For the Internet of Things, Wi-Fi is also too power-hungry. It is therefore preferred to an evolution called **IEEE 802.15.4** which uses its operating principle but adapts it to a low speed and to small frames. In particular to save energy, the range is reduced to about ten meters and it is generally necessary to use relays to reach a destination.

Bluetooth has been adapted for objects with a low consumption Bluetooth Low Energy (BLE).

On the cellular side, protocols are evolving to take objects into account. The 4G standard has integrated lower speed communications. The 5G will include a class allowing communications with objects energy efficient and reducing latency.

5.4 IP and adaptation layers

But, as seen in Figure 5.3 on the facing page IPv6 implies larger headers, which is troublesome because Layer 2 networks carry smaller frames. An **adaptation layer** between the IP layer and Layer 2 is needed since Layer 2s designed for the Internet of Things cannot naturally carry large packets. Two actions are implemented : **compression** of header size to reduce their impact, and **fragmentation** to break the packet into smaller frames if the first measure is not sufficient.

There are two main families of adaptive layers :

— **6LoWPAN** RFC 4944, RFC 6282, which will integrate a mechanism of compression of the header IPv6 and of fragmentation to send a large packet divided into small frames. Indeed, in a mesh network, it is not possible to deprive oneself of information provided by the IP layer

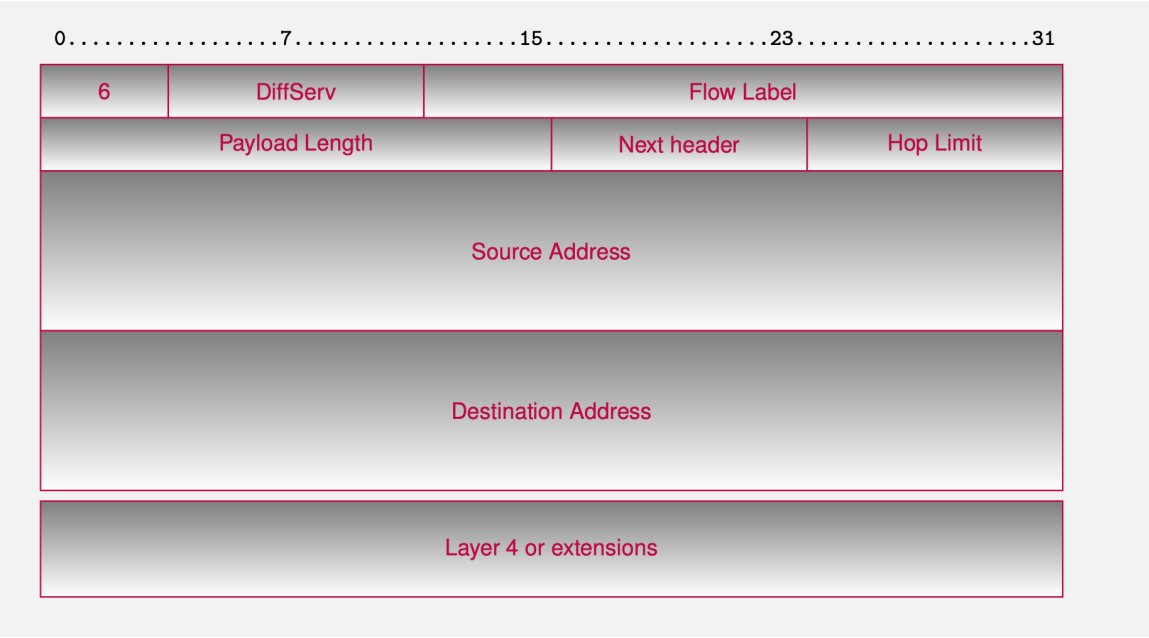


FIGURE 5.3 – IPv6 header format

because the intermediate nodes need it to route the message to the recipient. 6LoWPAN is stateless and compresses all IPv6 headers without configuration.

— Static Context Header Compression (SCHC) (pronounce chic) RFC 8724 will impose rules describing the message header and will send the rule number in place of the header. Compression is much more important and can involve several protocol layers. However, to implement it, you need to have an idea of the flows that will circulate on the network. SCHC is specified for networks in **star** and more particularly for **LPWANs!** (**LPWANs!**).

5.5 Implementation of REST

Above we had seen that as HTTP was the dominant protocol, TCP was also. But for the IoT this is not optimal. Indeed TCP/HTTP are complex protocols that require a lot of memory. To reduce the impact of the protocol stack, the IETF has defined a new protocol called CoAP which requires only a few Kilobytes to work. Constrained Application Protocol (CoAP) is based on UDP which simplifies again the implementation.

To continue in the integration of the objects in the Internet, the protocol CoAP RFC 7252 replaces HTTP. It takes over the naming mechanism, the use of resources, and the handling primitives between a client and a server.

The processing capacity of the sensor and its power supply are often very limited. The great strength of CoAP is to be :

— easy to implement. The implementations of CoAP require little memory ;

—

As a result, CoAP will manipulate resources, identified by URIs. It is thus possible to anchor the data provided by the objects in the current ecosystem of communications between computers,

strongly structured around the REST principles.

Security, especially data encryption, also follows the same paths as the traditional Internet. There is an encryption above UDP which, like HTTPS, encrypts the exchanges.

5.6 Data representation

For data structuring, XML is not used because it is too talkative. JSON is much more effective to transport structured information. There is a binary equivalent that we will see later : CBOR, which is much more efficient, and simple to implement, and compatible with JSON.

5.7 Alternatives to REST

It does not have to implement all the protocols defined by the IETF. It is also possible to integrate protocols specified for a business.

For example, the electric meter **Linky** that all French people know implements only a part of it. Instead of using CoAP, the electricians use their own applications following the standard DLMS/Cosem. This one is based on UDP then IPv6 and **6LoWPAN** and finally on a variant of **IEEE 802.15.4** adapted to transport the information on the electric cables.

6. THE REPRESENTATION OF DATA

6.1 Introduction

Sending data over a network is not as simple as it seems.

There is a difference between the format used to store data in the computer's memory and the one used to send it to another machine. Indeed, each machine has its own representation often linked to the capacities of their processor. This is especially true for numbers. They can be stored on a more or less important number of bits or can be represented in memory in an optimized way to accelerate their treatment.

Youtube

On the other hand, the representation of (unaccented) character strings is relatively uniform because it is based on the ASCII code which is the same for all computers. A basic text is easily understandable by all machines. A solution would therefore be to use only strings of characters.

For example, if we want to send the integer with the value 123, there are several possible representations :

— send a string "123" containing the digits of the number ;
— send the binary value 1111011.

We see that just to transmit a simple value stored in the memory of a computer, there are several options and obviously for this value to be interpreted in the right way, it is necessary that both ends have agreed on a representation.

When you want to transmit several values, i.e. when you have structured data, other problems arise.

For example : what is the size of the blocks we are going to transmit ? How to indicate the end of the transmission ? For a string, how to indicate that it ends ? Another example : if we want to transmit "12" and then "3", how do we make sure that the other end does not include "123" ?

In order for the transmission to take place correctly, the sender and receiver must adopt the same conventions. When it is a question of a set of data, it is necessary to be able to separate them. With spreadsheets, a first method is possible with the notation Comma Separated Values (CSV) As its name indicates, the values are separated by commas. The values are represented by strings. The texts are differentiated from the numerical values by the use of quotation marks. Thus, 123 will be interpreted as a number and "123" as a text.

If this representation is adapted to spreadsheets, it is relatively poor because it only allows to represent values on rows and columns. For Web uses, it was necessary to find a more flexible format allowing to represent complex data structures. Obviously, as nothing is simple, there are several of them and applications exchanging data will have to use the same one.

We can see that sending the string is not enough, it must be formatted so that the receiver can find the type of the transmitted data, so that a number is not interpreted as a string, so that a string remains a string even if it contains only numbers.

6.2 The serialization

Under this barbaric name hides the method used to transmit data from one computer to another. A data can be simple (a number, a text) or more complex (an array, a structure...). It is stored in the computer's memory according to a representation that is specific to it. For example, the size of integers can vary from one processor technology to another, the order of bytes in a number can also be different (little and big endian). For complex structures such as arrays, the elements can be stored in different locations in the memory.

Serialization consists of transforming a data structure into a sequence that can be transmitted over the network, stored in a file or a database. The opposite operation, consisting in locally reconstructing a data structure, is called deserialization.

There are several formats for serializing data. They can be binary but those generally used are based on character strings. Indeed, the representation ASCII defining the basic characters and coded on 7 bits is common to all the computers. The other advantage of the ASCII code is that it is easily readable and simplifies the development of programs.

Wikipedia gives this table (cf. figure reffig-ASCII) of the codes dating from 1972 (an eternity in computing) and recolored by us.

The characters in orange are not printable. They are used to control data communication or to manage the display by returning to the line. They can be recognized because the binary sequence starts with 00X XXXX. One recalls that the ASCII code is on 7 bits ; the additional bit (parity bit) leading to 1 byte was used to detect transmission errors. The values from 0x30 to 0x39 code the digits from 0 to 9.

Hexlify

In Python, there is the module **binascii** very practical which makes it possible to convert a binary sequence into a character string or conversely :

— `hexlify` takes an array of bytes and converts it to a more specialist-readable hexadecimal string. This allows you to view any sequence of data.

— `unhexify` does the opposite. It takes a string and converts it to an array of bytes. This can make programming easier for you because in your code it is easier to manipulate strings.

USASCII code chart

b_7 b_6 b_5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Bits b_4 b_3 b_2 b_1 ↓ / Column → / Row ↓	0	1	2	3	4	5	6	7
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FS	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

FIGURE 6.1 – ASCII character encoding

In the following, we will use these functions to manipulate identifiers. For example, this piece of code illustrates the use of these functions :

```
mac = lora.mac()
print ('devEUI: ',  binascii.hexlify(mac))

# create an OTAA authentication parameters
app_eui = binascii.unhexlify('70 B3 D5 7E D0 03 3A E3'.replace(' ',''))
```

As we will see later, the function `lora.mac()` returns an array of bytes. The function `hexlify` in the follwoing line converts it to a string for a cleaner display.

Conversely, we must assign a binary sequence to the variable `app_eui`. We put this hexadecimal sequence into a string. Spaces offer more readability. They are removed by the replace method and the result is converted into binary thanks to `unhexify`

6.3 Base64

The passage from a binary sequence to an ASCII character string representing the values leads to a doubling of the size. Each block of 4 bits will lead to produce a byte corresponding to the character of a digit or a letter from A to F. The rest of the codes are not used.

The encoding **base64** offers a better performance by using 6 bits to encode the values. A dictionary maps 64 values to an ASCII character. However, if we want to encode 4 bytes, or 32 bits, we will need 5 blocks of 6 bits, and there will be 2 bits left. The symbol = indicates that 2 bits are added at the end of the coding. So, in our case, it will be necessary to add two symbols = as shown in the figure below :

Note that for small sequences, this coding is not better than the transformation of the hexadecimal sequence into a string. Here, 8 characters are needed to encode 4 bytes.

There are many online tools to make conversions between these different representations, such as the site `www.asciitohex.com`.

Python module : base64

In Python3, the base64 module allows to do these conversions. This module is a bit touchy about the data types to use.

```
1 import base64
2
3 val = b"\x01\x0234"
4 ser = base64.b64encode(val)
5 print (ser)
6 print (ser.decode())
7 ori = base64.b64decode(ser)
8 print (ori)
```

which gives as a result of the execution :

```
b'AQIzNA=='
AQIzNA==
b'\x01\x0234'
```

Note that the use of `ser.decode()`, line 6, to transform a string of bytes into a string of characters, i.e. remove the `b` at the beginning, can be used in certain cases.

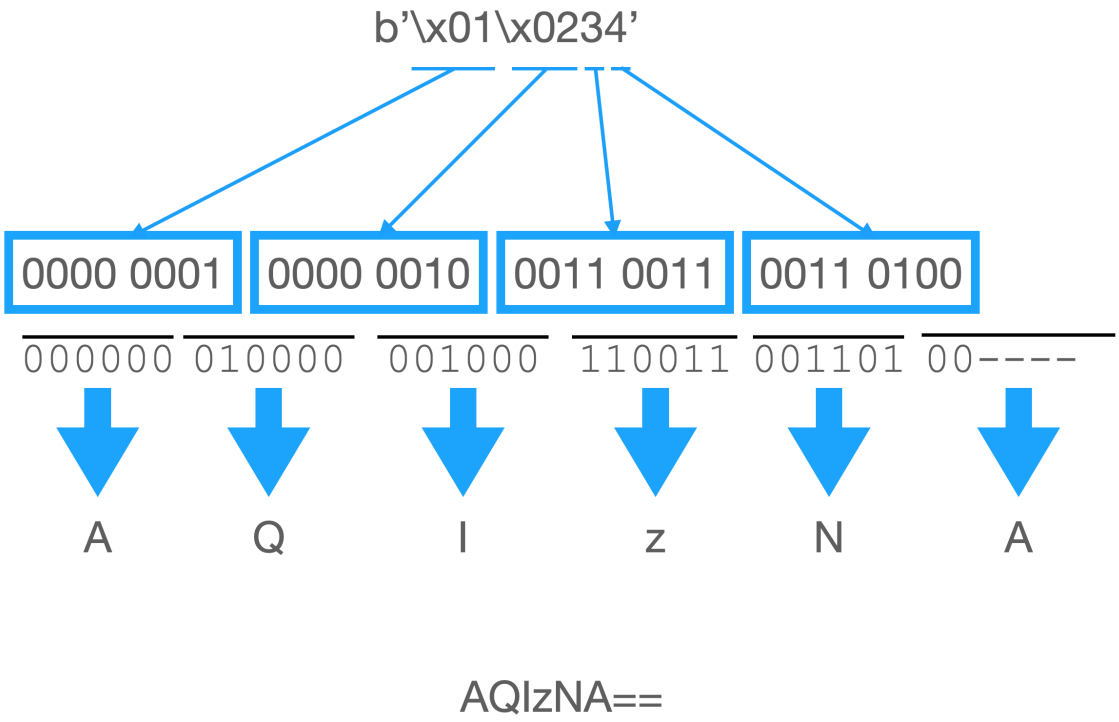


FIGURE 6.2 – Base64 coding of binary data

```
<p>Les s&eacute;rialisations en cha&icirc;nes de caract&egrave;res (par exemple en Python via la
commande <span style="font-family: 'courier new', courier;">hexlify<span style="font-family:
'Open Sans', Verdana, Arial, Helvetica, sans-serif;">)</span></span> ou en base64 concernent
surtout des donn&eacute;es binaires, mais la donn&eacute;e peut &ecirc;tre aussi
structur&eacute;e, par exemple la page d'un tableur. Il faut donc formater le document pour
&eacute;viter fusion entre les diff&eacute;rents champs. Le format CSV (<em>Comma-Separated
Values</em>) comme son nom l'indique s&eacute;pare les donn&eacute;es par des virgules
(<em>comma</em> en anglais). Mais si ce format s'applique bien aux donn&eacute;es d'un tableau,
c'est &agrave; dire un tableau de lignes et de colonnes, il est tr&egrave;s limit&eacute; pour
repr&eacute;senter une information telle que la mise en forme d'une page web.</p>
```

FIGURE 6.3 – HTML coding of a Web page

```
<img src ="/asset-v1:MinesTelecom+04038+session01+type@asset+block/Mooc-internet_objets.jpg" width="300" type="saveimage" target="[object Object]"/>
<br>
  <p><strong>Enseignants </strong><br>

    <img src ="/asset-v1:MinesTelecom+04038+session01+type@asset+block/Laurent-Toutain.jpg" width="100" type="saveimage" target="[object Object]"/>
    <br>Laurent Toutain<br>
    <p>
</p>
<img src ="/asset-v1:MinesTelecom+04038+session01+type@asset+block/Photo_Marc_Girod_Genet.jpg" width="100" type="saveimage" target="[object Object]"/>

    <br>Marc Girod-Genet<br>
    <p>
</p>
<img src ="/asset-v1:MinesTelecom+04038+session01+type@asset+block/Kamal_Singh.png" width="100" type="saveimage" target="[object Object]"/>
    <br>Kamal Singh<br>
    <p>
</p>
<img src ="/asset-v1:MinesTelecom+04038+session01+type@asset+block/BattonHubert.jpg" width="100" type="saveimage" target="[object Object]"/>
    <br>Mireille Batton Hubert<br>

    <p>
</p>
<img src ="/asset-v1:MinesTelecom+04038+session01+type@asset+block/PatrickMaille.jpg" width="100" type="saveimage" target="[object Object]"/>
    <br>Patrick Maille<br>

<br>

    <p><strong>Support pédagogique</strong>
    </p>
<img src ="/asset-v1:MinesTelecom+04038+session01+type@asset+block/DenisMedalSmall.png" width="100" type="saveimage" target="[object Object]"/>
<br>Denis Moalic<br>
```

FIGURE 6.4 – Capture a web page

6.4 HTML

Serialization in strings (for example in Python via the command `hexlify`) or in Base64 concerns mainly binary data. But the data can also be structured, for example the page of a spreadsheet. The document must therefore be formatted to avoid merging the various fields.

HTML, without going into detail, defines a format where fields are marked up with markup. A beginning tag is a keyword between <> and, for an ending **tag**, the keyword is preceded by the character /. For example, the figure 6.3 with the markup, the first paragraph is formatted this way in the MOOC :

Tags can also take arguments, like the **span** tag in the previous example. Thus, if we look at a Web page, as shown in figure 6.4, the browser is able to analyze it to find the URI it contains. With the text tag indicating that it is an image, the client can query the server to display it on the screen. This structured serialization format allows us to implement a feature of REST, i.e. the links between resources.

6.5 XML

If HTML is dedicated to the formatting on screen of textual data and to the navigation on the Web. XML [1] defined by the World Wide Web Consortium (W3C), is a format for exchange between

1. `https://www.w3.org/TR/xml/`

two applications. For example, to exchange student grades between the FUN platform and a course certification authority, one could use the following format :

```
<etudiant>
    <prenom>John</prenom>
    <nom>Deuf</nom>
    <note>18</note>
</etudiant>
```

It is easy by reading the example to find the student's first name, last name and grade. It can be noted that there is no difference between the grade and the student's name. They are characters.

If it is syntactically correct, there is no guarantee that the creator is providing something correct that can be interpreted by another instance. can include a grammar or schema that is used to validate that the information represented in the file is not only syntactically compliant with the language, but also complies with the schema. This schema will describe the expected fields and their type (text, number...). You can access this course if you want to know more about schemas.

From the point of view of the Internet of Things, even if XML could be a good candidate for the exchange of information, it is too heavy a format and therefore energy consuming. We can note that to send a note out of 20 which, in the absolute, would take 6 bits, we transmit `<note>18</note>`, that is to say 15 characters or 120 bits !

6.6 JSON

JSON offers a way to structure information in a more compact way than XML. JSON is emerging as the common language for exchanging information. Originally, JSON was used by **Javascript** to exchange information ; for example, to display in real time the evolution of stock exchange prices or to display dynamic graphs on the user's screen.

Youtube

JSON RFC 8259 is a simple exchange format. It defines 4 data types :

- Number : Numbers are composed of digits and can be positive, negative, integers or floats.
- Text : The text is delimited by single or double quotation marks.
- **Array** : Arrays are lists of elements separated by commas and surrounded by square brackets.
- **Object** : The object is a list of pairs composed of an **key** and a value. The key is a string and the value can be of any type. The key must be unique within an object, and fully references the value that follows it. The couple key - value is separated by the colon character `:`. The elements of the object are separated by commas. The object is delimited by braces.

For example, some JSON structures :

- `[1, -2, 0.3, 4e1]` is an array that contains 4 numbers ;
- `[1, "2", "34"]` is an array containing a number and two strings ;
- `[1, [2, 3 , "4"]]` is an array of two elements whose second element is also an array of 3 elements ;
- `{ "color": [34, 16, 3]}` is an object that contains an element and the value is an array ;

— `{"name" : "bob", "age" : 30}` is an object that contains two elements referenced by the strings (or index) "name" and "age".
The order in which the elements are placed is irrelevant. For example, `{"age": 30, "name": "bob"}` is equivalent to the last example.
This imposes that the index used to access a value must be unique in the object structure : `{"name" : "bob", "name" : "alice"}` is prohibited.

The following listing gives an example of a JSON structure from the RFC 8259. It contains a JSON object with a single key `"Image"`. The value of this key is another structure that contains six elements.

```
{
"Image": {
      "Width": 800,
      "Height": 600,
      "Title": "View from 15th Floor",
      "Thumbnail": {
           "Url": "http://www.example.com/image/481989943",
           "Height": 125,
           "Width": 100
       },
       "Animated" : false,
       "IDs": [11, 943, 234, 38793]
    }
}
```

Key markup is a fundamental element in the structure of data. It is essential to be consistent and to ensure that the sender and receiver agree on the name of the key in order to be able to retrieve the desired information. In the same way, it is necessary to agree on the units of measurement : an interpretation of a measurement in centimeters when it is in pixels can be disastrous ; it is an interoperability problem.

JSON is easily exploitable in other languages. For example in Python, the JSON module can be used to convert a JSON structure that is an ASCII string into a Python internal representation. Arrays are converted into lists and objects into dictionaries.

Listing 6.1 – example_json.py

```
import json
import pprint

struct_python = {
"Image": {
      "Width": 800,
      "Height": 600,
      "Title": "View from 15th Floor",
      "Thumbnail": {
           "Url": "http://www.example.com/image/481989943",
           "Height": 125,
           "Width": 100
```

```
        },
        "Animated" : False,
        "Copyright" : None,
        "IDs": [0x11, 0x943, 234, 38793],
        "Title": "Empty picture"
    }
}

print (struct_python)
pprint.pprint(struct_python)

struct_json = json.dumps(struct_python)

print(struct_json)

struct_python2 = json.loads(struct_json)

pprint.pprint (struct_python2)
```

The program `example_json.py` uses the previous structure. The variable `struct_python` is a Python structure. We can see that the values for `"Animated"` and `"Copyright"` are the Python keywords `False` (with a capital F) and `None`. The program displays this value twice with the standard command `print` then with the module `pprint` to have a more readable display. You can notice that the order of display of the keys is different. As `"Title"` was defined twice, only the last one is kept in the Python structure.

```
{'Image': {'IDs': [17, 2371, 234, 38793], 'Height': 600, 'Animated': False, 'Title': 'Empty picture', 'Thumbnail':
{'Url': 'http://www.example.com/image/481989943', 'Width': 100, 'Height': 125}, 'Width': 800, 'Copyright': None}}
{'Image': {'Animated': False,
           'Copyright': None,
           'Height': 600,
           'IDs': [17, 2371, 234, 38793],
           'Thumbnail': {'Height': 125,
                         'Url': 'http://www.example.com/image/481989943',
                         'Width': 100},
           'Title': 'Empty picture',
           'Width': 800}}
```

Thanks to the function `dumps` of the module `json`, the variable `struct_python` is transformed into JSON. The keywords `False` and `None` are replaced by `false` and `null`. The program displays a string.

```
{"Image": {"IDs": [17, 2371, 234, 38793], "Height": 600, "Animated": false, "Title": "Empty picture", "Thumbnail":
{"Url": "http://www.example.com/image/481989943", "Width": 100, "Height": 125}, "Width": 800, "Copyright": null}}
```

To transform it back, from JSON to Python variable, we use the inverse function `loads` which translates a string into a Python variable.

```
{'Image': {'Animated': False,
           'Copyright': None,
           'Height': 600,
           'IDs': [17, 2371, 234, 38793],
           'Thumbnail': {'Height': 125,
                         'Url': 'http://www.example.com/image/481989943',
                         'Width': 100},
           'Title': 'Empty picture',
           'Width': 800}}
```

Other programming languages also have their own libraries for translation.

Compared to XML, JSON is much more permissive and lacks a formalism to describe the structure. The JavaScript Object Notation for Linked Data (JSON-LD) defined by the W3C reinforces the interoperability of JSON by introducing specific keys describing the data structure, a reference to units, etc. We will see these concepts in the rest of the course.

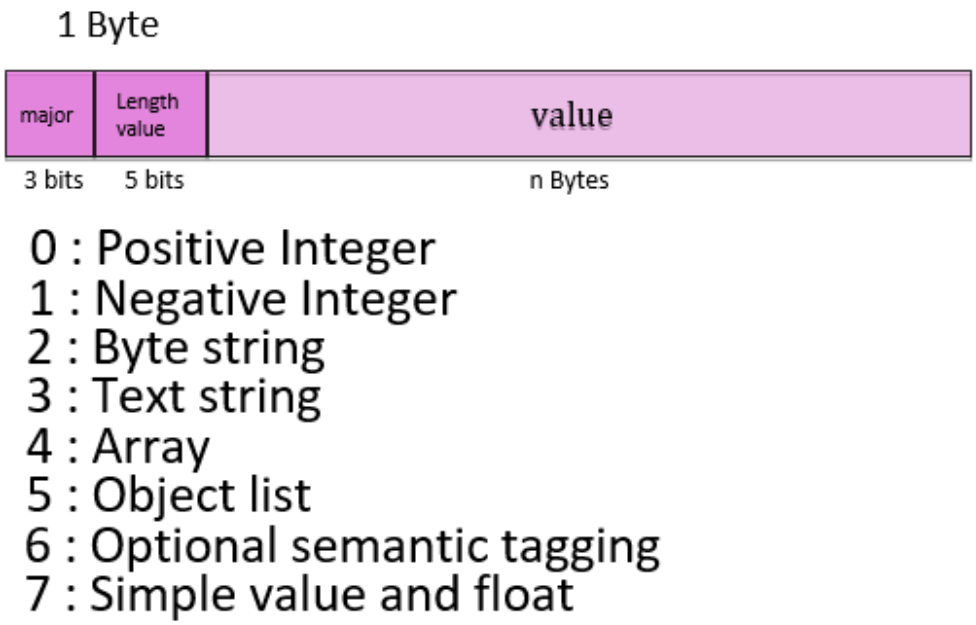


FIGURE 6.5 – Definition of major in CBOR

6.7 CBOR

JSON and Concise Binaire Object Representation (CBOR) are both data encoding modes.

Youtube

JSON introduces a very flexible notation allowing to represent all data structures. The choice of the ASCII makes this format universal and any computer will be able to understand it. But the use of the ASCII does not make it possible to transmit information in an optimal way on a network. When the networks have a reasonable flow, it does not pose a problem. When it comes to the Internet of Things, we must take into account the limited processing capacity of the equipment and the small size of the messages exchanged.

Thus, in ASCII, the value `123` is coded on 3 bytes (one byte by character) while in binary it would occupy only one byte : `0111 1011`.

CBOR, defined in the RFC 8949, allows to represent the structures of JSON but according to a binary representation. As we will see later, if CBOR is completely compatible with JSON, it is possible to represent other types of information very useful in the Internet of Things.

The size of the information is reduced and the processing simplified. You have to know how to juggle with the binary representation but it remains basic.

CBOR defines 8 major types which are represented by the first 3 bits of a CBOR structure (cf. figure **??** on page ??). These major types thus have values ranging from 0 to 7 (`000` to `111` in binary).

The next five bits contain either a value or a length indicating how many bytes are needed to

encode the value. , this type offers optimizations that allow to reduce the total length of the data structure as we will see later when studying the different major types.

6.7.1 CBOR in Python

The following examples can be tested on your computer with Python3. If an error occurs when defining the module `Indexcbor2`, you must install it on your computer by typing the command :

```
# pip3 install cbor2
```

6.7.2 Type Positive Integer

JSON does not make a difference between numbers, integers, decimals, positive or negative. CBOR reintroduces a distinction to optimize the representation.

The first major type corresponds to the positive integers. It is coded by 3 bits at 0 ; the 5 following bits end the byte and, according to their value, will have a different meaning :

— from 0 to 23, it is the value of the integer to be coded ;
— 24 indicates that the integer is coded on 1 byte which will be coded in the following byte ;
— 25 indicates that the integer is coded on 2 bytes which will be coded in the two following bytes ;
— 26 indicates that the integer is coded on 4 bytes which will be coded in the four following bytes ;
— 27 indicates that the integer is coded on 8 bytes which will be coded in the eight following bytes.

One can note that there is no extra cost to code an integer from 0 to 23. Thus, the value 15 will be coded 0x0F (`000-0 1111`) while, for all the other higher values, the overcost will be only of one byte. The value 100 will be coded `000-1 1000` [2] followed by the coding on 1 byte of the value 100 (`0110 0100`).

Listing 6.2 – cbor-integer-ex1.py

```
1 import cbor2 as cbor

3 v = 1

5 for i in range (0, 19):
      c = cbor.dumps(v)
7     print ("{0:3} {1:30} {2}".format(i, v, c.hex()))

9     v *= 10
```

The program `cbor-integer-ex1.py` displays the powers of 10 between 10^0 and 10^{18} :

— Line 1, the program imports the module `Indexcbor2` and renames it for simplicity `cbor`. Line 5, the loop allows to have the multiples of 10 (variable `v`).
— Line 6, the module `cbor` uses as for JSON the method `dumps` to serialize an internal Python structure into the requested representation. Conversely, the `loads` method will be used to import a CBOR structure into an internal representation.

2. `11000` corresponds to 24

— On line 7, the `print` is used to align the data so that the display is clearer ; between the braces, the first number indicates the position in the format arguments ; the second, after the `:`, the number of characters. For example, `{:1:30}` indicates the format argument `v` displayed in 30 characters.

The program gives the following result :

```
# python3 cbor-integer-ex1.py
 0                                  1 01
 1                                 10 0a
 2                                100 1864
 3                               1000 1903e8
 4                              10000 192710
 5                             100000 1a000186a0
 6                            1000000 1a000f4240
 7                           10000000 1a00989680
 8                          100000000 1a05f5e100
 9                         1000000000 1a3b9aca00
10                        10000000000 1b00000002540be400
11                       100000000000 1b000000174876e800
12                      1000000000000 1b000000e8d4a51000
13                     10000000000000 1b000009184e72a000
14                    100000000000000 1b00005af3107a4000
15                   1000000000000000 1b00038d7ea4c68000
16                  10000000000000000 1b002386f26fc10000
17                 100000000000000000 1b016345785d8a0000
18                1000000000000000000 1b0de0b6b3a7640000
```

It is easy to see that the values 1 and 10 are coded on 1 byte ; that 100 is coded on 2 bytes while the values 1 000 and 10 000 are coded on 3 bytes. The values between 100 000 and 1 000 000 000 require 5 bytes and the following values, 9 bytes.

The size of the representation adapts to the value. Thus, it is not necessary to define a fixed size to encode a data.

We can also note that as the major type is on 3 bits, this type can be recognized in a hexadecimal reading of the result because the sequence always starts with the symbol `0` or 1.

6.7.3 Type Negative Integer

The major type negative integer is roughly similar to the positive integer. The major type is `001` and the encoding of the value is done on the absolute value of the number to which we subtract 1. This avoids two different codes for the values 0 and -0.

Thus, to code -15, we will code the value 14, which gives in binary 001-1 1110. Thus, -24 can also be coded on 1 byte while +24 will be coded on 2 bytes.

Listing 6.3 – cbor-integer-ex2.py

```
import cbor2 as cbor

v = -1

```

```
for i in range (0, 19):
    c = cbor.dumps(v)
    print ("{0:3} {1:30} {2}".format(i, v, c.hex()))

    v *= 10
```

The program `cbor-integer-ex2.py` uses the same code as the previous program, but the variable v is initialized with the value -1. This program will process negative powers of 10.

```
# python3.5 cbor-integer-ex2.py
  0                                -1 20
  1                               -10 29
  2                              -100 3863
  3                             -1000 3903e7
  4                            -10000 39270f
  5                           -100000 3a0001869f
  6                          -1000000 3a000f423f
  7                         -10000000 3a0098967f
  8                        -100000000 3a05f5e0ff
  9                       -1000000000 3a3b9ac9ff
 10                      -10000000000 3b00000002540be3ff
 11                     -100000000000 3b000000174876e7ff
 12                    -1000000000000 3b000000e8d4a50fff
 13                   -10000000000000 3b000009184e729fff
 14                  -100000000000000 3b00005af3107a3fff
 15                 -1000000000000000 3b00038d7ea4c67fff
 16                -10000000000000000 3b002386f26fc0ffff
 17               -100000000000000000 3b016345785d89ffff
 18              -1000000000000000000 3b0de0b6b3a763ffff
```

6.7.4 Type Binary sequence or Character string

Binary sequences and strings have the same behavior. The major type is respectively `010` and `011`. It is followed by the length of the sequence or the string. The same type of encoding as for integers is used :

- if the length is lower than 23, it is coded in the continuation of the first byte. One finds then the number of bytes or characters corresponding to this length ;
- if the length can be coded in 1 byte (thus lower than 255), the continuation of the first byte contains 24 then the following byte contains the length followed by the number of bytes or characters corresponding.
- if the length can be coded in 2 bytes (thus lower than 65535), the continuation of the first byte contains 25 then the following byte contains the length followed by the number of bytes or characters corresponding.
- if the length can be coded in 4 bytes, the continuation of the first byte contains 26 then the following byte contains the length followed by the number of bytes or characters corresponding.
- if the length can be encoded in 8 bytes, the continuation of the first byte contains 27 then the following byte contains the length followed by the number of bytes or characters corresponding.

This coding is also quite optimal. It is rare to send more than 23 characters.

Listing 6.4 – cbor-string.py

```
1 import cbor2 as cbor
2
3 for i in range (1, 10):
4     c = cbor.dumps("LoRaWAN"*i)
5
6     print ("{0:3} {1}".format(i, c.hex()))
7
8 bs = cbor.dumps(b"\x01\x02\x03")
9 print (bs.hex())
```

The program `cbor-string.py` shows the representation of strings of increasing length and a binary sequence :

— line 3, the variable `i` takes values from 1 to 9.

— line 6, The multiplication of a string by an integer (line 4) indicates the number of repetitions of it.

— lines 8 and 9 show the encoding of a byte string. The variable bs contains the CBOR representation of a Python byte string (represented by the character `b` before the quotation marks, the values which do not correspond to ASCII characters are preceded by the symbols `\x`). The hexadecimal representation of the CBOR object is then displayed.

The result is as follows :

```
# python3.5 cbor-string.py
  1 674c6f526157414e
  2 6e4c6f526157414e4c6f526157414e
  3 754c6f526157414e4c6f526157414e4c6f526157414e
  4 781c4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e
  5 78234c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e
  6 782a4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e
  7 78314c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e
  8 78384c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e
  9 783f4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157...
43010203
```

Up to 3 repetitions of the string "LoRaWAN", the length coding is optimal (coded on 2 bytes).

6.7.5 Type Array

The array type will group a set of elements. Each of these elements being a CBOR structure, the only information needed to know the beginning and the end of an array is its number of elements. The major type is 100. There are two methods to encode the length of an array :

— if this one is known at the time of coding, it is enough to indicate it with a coding identical to the one used to indicate the length of a character string ;

— if this is not known at the time of coding, there is a special code to indicate the end of the table. We will talk about this later.

Listing 6.5 – cbor-array.py

```
1 import cbor2 as cbor
2
3 c1 = cbor.dumps([1,2,3,4])
4 print (c1.hex())
5 print ()
6
7 c2 = cbor.dumps([1,[2, 3], 4])
8 print (c2.hex())
9 print()
```

```
11 c3 = cbor.dumps([1000, +20, -10, +100, -30, -50, 12])
   print (c3.hex())
```

The program `cbor-array.py` gives some examples of array coding :

— `[1,2,3,4]` defined on line 3, becomes `8401020304`. We can guess the structure of the CBOR message : `0x84` indicates an array of 4 elements (be careful, decoding is not always so simple). The 4 elements are integers lower than 23 ;

— `[1,[2, 3], 4]` defined on line 7 becomes `830182020304`. This is an array of 3 elements, the second of which is an array of two elements ;

— `[1000, +20, -10, +100, -30, -50, 12]` defined on line 11, becomes `871903e814291864 381d38310c`. Note that the coding of the elements is of variable length, but as each element codes its length, it is only necessary to know the number of elements.

The type List of pairs or Map is indicated by the value `101`. It works in the same way as arrays by counting the number of elements. But this time, the value represents a pair, i.e. two consecutive CBOR objects.

Listing 6.6 – cbor-mapped.py

```
   import cbor2 as cbor
2
   c1 = {"type" : "hamster",
4        "taille" : 300,
         2 : "test",
6        0x0F: 0b01110001,
         2 : "program"};
8
   print (cbor.dumps(c1).hex())
```

The program `cbor-mapped.py` gives an example of encoding. Note that the structure to be encoded is not directly compatible with JSONfootnotejson.`dumps` would have converted the numeric keys into strings `'{"type": "hamster", "2": "program", "taille": 300, "15": 113}'`, some keys are not strings.

The result is `a46474797065676861 6d7374657266746 96c6c6519012c026770726f6772 616d0f1871` which is not very easy to read.

cbor.me

The web site `https://cbor.me` allows to do automatically the encoding in a direction or in the other. The left column represents the data in JSON and the right one in CBOR (called "canonical representation" which facilitates the reading). Having entered the above hexadecimal sequence, the site presents it as shown in figure 6.6 on the following page. The CBOR part is indexed and commented to make the CBOR object more readable. It can also be translated into a JSON equivalent, although some keys remain numeric.

Youtube

On these examples, we can see that CBOR is much more permissive and complete than JSON, the first field of the CBOR map can be numeric and does not have to be unique in the whole structure. Nevertheless CBOR defines a strict mode in which these keys must be ASCII encoded and unique to be compatible with JSON. If a key is repeated several times in a CBOR structure, it is necessary to process the information

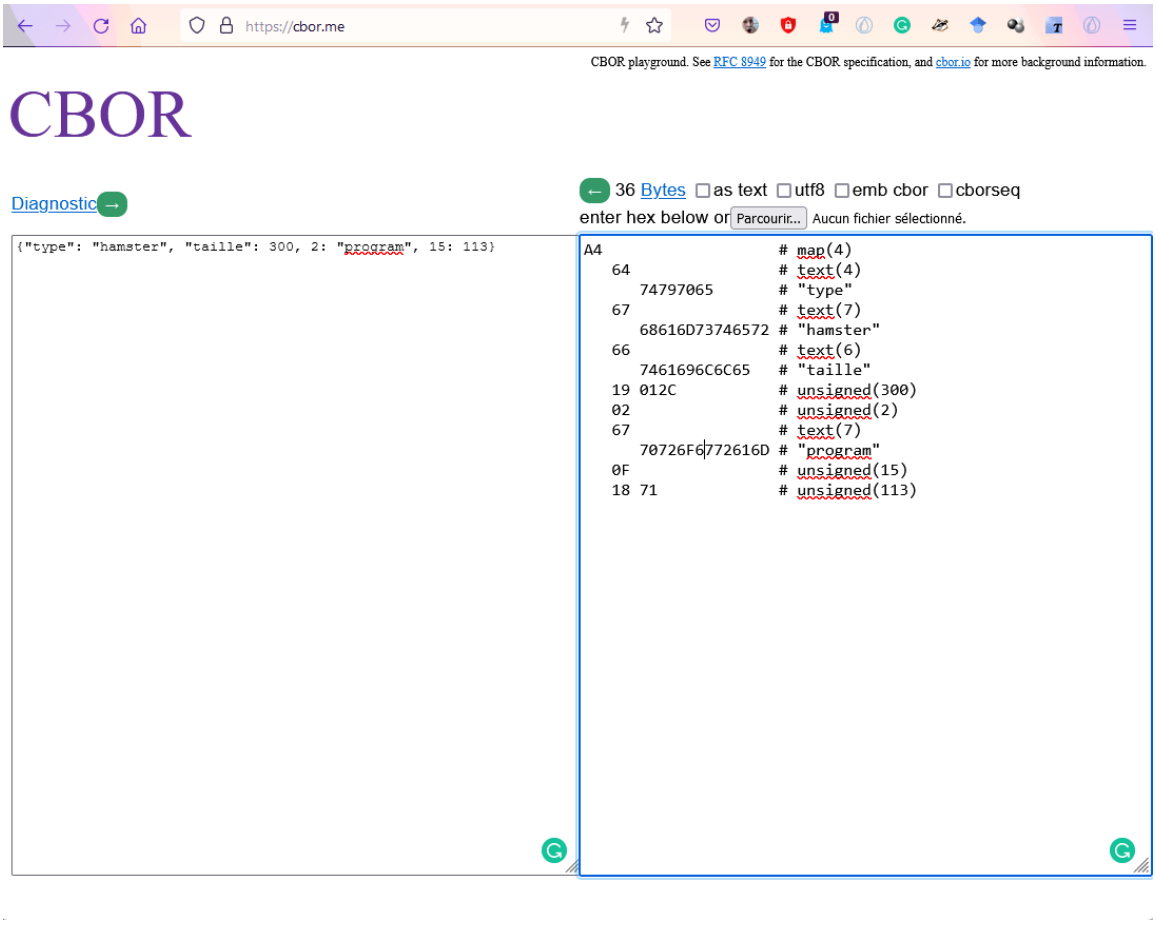


FIGURE 6.6 – Definition of major in CBOR

directly in the CBOR structure and not to try to convert or deserialize it because there is a risk of losing information.

6.7.6 Type Tag

CBOR enriches the typing of data; this makes it easier to manipulate data. For example, a character string can represent a date, a URI, or even a URI encoded in base 64. The type 110 can be followed by a value or **tag** of which an exhaustive list is maintained by the IANAfootnoteurlhttps ://www.iana.org/assignments/cbor-tags/cbor-tags.xhtml.

Listing 6.7 – cbor-tag.py

```
import cbor2 as cbor
from datetime import date, timezone

print (date.today())
c1 = cbor.dumps(date.today(), timezone=timezone.utc, date_as_datetime=True)

print (c1.hex())

print (cbor.loads(c1))
print (type(cbor.loads(c1)))
```

For example, the program `cbor-tag.py` returns the following results :

```
# python3.5 cbor-tag.py
2018-05-22
c074323031382d30352d32325430303a30303a30305a
2018-05-22 00:00:00+00:00
43010203
<class 'datetime.datetime'>
```

The canonical representation shows more easily the tag in the binary sequence :

```
C0                                          # tag(0)
   74                                       # text(20)
      323031382D30352D32325430303A30303A30305A # "2018-05-22T00:00:00Z"
```

The tag 0 implies a normalized format for the date; hence the addition of hours, minutes and seconds, even though they were not initially specified. We can also notice that `loads` returns a type `date` and not a character string.

6.7.7 The floating type and particular values

The last major type (111) allows to encode floating numbers by using the representation defined by the **IEEE 754**. According to the size of the representation, the continuation of the byte contains the values 25 (half precision on 16 bits), 26 (simple precision on 32 bits) or 27 (double precision on 64 bits).

This type also allows to encode the values defined by JSON : True (value 20), False (value 21) or None (value 22).

Finally, this type can indicate the end of an array or a list of pairs when the size is not known at the beginning of the coding.

6.8 Questions about CBOR

Question 6.8.1: Benefits of CBOR

What are the advantages of CBOR over JSON (2 answers) ?

- ☐ It is more compact in its data representation.
- ☐ It allows to represent floating numbers.
- ☐ It compresses strings.
- ☐ It is easier to implement.

Question 6.8.2: Floating

Is a float always more compact in CBOR than in JSON ? - You can help yourself with `https://cbor.me`

- ○ Yes, that is the goal of CBOR.
- ○ Yes, for floats that have the decimal part at 0.
- ○ Yes, for small precision floats (up to a hundredth).
- ○ Yes, for high precision floats (6 digits after the decimal point).

Question 6.8.3: lgfChaîne de caractères

Consider a string in CBOR.

- ○ It is compressed with an entropy algorithm (e.g. Huffman coding).
- ○ It can contain accented characters.
- ○ Each character is coded on 6 bits.

Question 6.8.4: Variable length

In CBOR, the size of an integer varies according to its value !

- ○ True
- ○ False
- ○ It depends on how this integer was declared.

Question 6.8.5: Array

In CBOR, an array can contain objects of different types.

- ○ True
- ○ False
- ○ It depends on how this table was declared.

Question 6.8.6: Fractional

We want to define an array of two elements as a fraction. What tag should precede the structure ? (you can use the RFC 8949).

6.9 SenML

is a specification that leverages JSON or CBOR. It lists a set of names/units/measures and standardizes them into a unique key name. By using this standardization, interoperability is facilitated. The keys and values are therefore regulated and typed to avoid any interoperability conflicts. The format is defined in the RFC8428 and is based on a table structure grouping objects as shown in the following figure taken from the RFC.

```
[
 {"bn" : "urn:dev:ow:10e2073a01080063", "bt":1.320067464e+09,
  "bu" : "\%RH", "v":21.2},
 {"t":10, "v":21.3},
 {"t":20, "v":21.4},
 {"t":30, "v":21.4},
...
```

SenML defines the keys used in the object. To have a compact notation, they are limited to 1 or 2 characters. Among them, "bn" indicates a base name and "n" the name of a device. If several devices send the common part of the device identifier, we can put the "bn" to avoid repeating it each time.

The base time (or "bt") is also a way to compact the time notation. The time ("t") gives the offset and leads to a smaller value as seen in the example.

The base unit ("bu") indicates the default unit if the other objects do not have a keyword indicating the unit ("u").

The RFC 8428 defines a list of units such as kilogram ("kg"), volt ("V"), etc. In the example, "%RH" refers to a percentage of relative humidity. A numeric value uses the letter "v", a string uses the "vs" key.

CBOR uses the same structure but small positive and negative integers are substituted in the keys of CBOR objects : "bn", "bt", "bu" will be represented by -1, -2 and -3 respectively and "n", "t", "u" by +0, +2 and +6.

7. Implementing a Virtual Sensor

The programs related to this section are located in the directory `plido-tp3`. It is recommended to use the virtual machine with one terminal window for the object and another for the server.

7.1 JSON

We've been talking about the Internet of Things for a long time, it's time to put our knowledge into practice. We will start with a simple implementation in Python on your machine. The goal in this part is to test everything we have seen without any other hardware than a computer. We will open two terminal windows. In one we will run a program that will emulate the object with three sensors. This object will communicate with a server that will collect the information. The communication will be done internally with a socket using the interface **loopback** (cf. figure 7.1 on the next page).

Youtube

7.1.1 Minimal Server

Listing 7.1 – minimal_server.py

```
import socket
import binascii

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(('0.0.0.0', 33033))

while True:
    data, addr = s.recvfrom(1500)
    print (data, "=>", binascii.hexlify(data))
```

Let's start by building a summary server (`minimal_server.py`) which will display everything it receives.

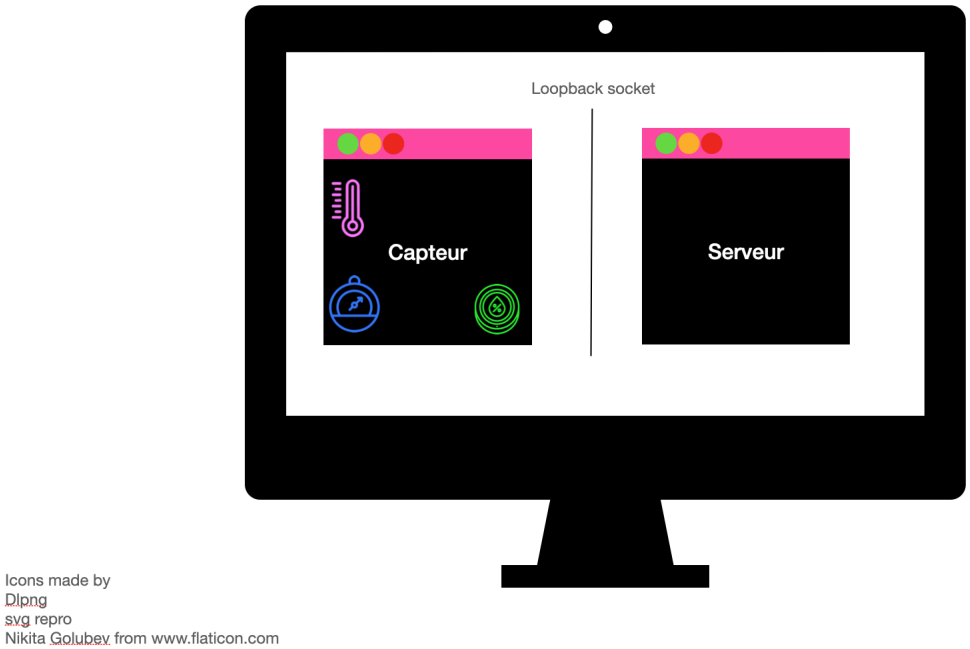


FIGURE 7.1 – Client Server architecture

— lines 1 and 2 the modules `socket` for the communication and `binascii` to transform the binary into string are imported.
— line 4, the socket function creates the socket `s` with parameters for IP (`AF_INET`) and UDP (`SOCK_DGRAM`) communications.
— line 5, the socket is associated with the port number 33033 via the function `bind`. The address `0.0.0.0` indicates that the data can come from any interface (Ethernet, Wi-Fi, loopback,...).
— line 8, in the endless loop, the function `recvfrom` will get stuck waiting for data. It returns the data and the address of the sender.
— line 9, the data are displayed in byte string and in hexadecimal.

We launch the server program. As nobody talks to it, it does not display anything.

7.1.2 Virtual sensor

Listing 7.2 – virtual_sensor.py

```
1 import random
2
3 class virtual_sensor:
4
5     def __init__(self, start=None, variation=None, min=None, max=None):
6         if start:
7             self.value = start
8         else:
9             self.value = 0
```

```
11          self.variation = variation
            self.min = min
13          self.max = max

15      def read_value(self):
            self.value += random.uniform(self.variation*-1, self.variation )
17
            if self.min and self.value < self.min: self.value = self.min
19          if self.max and self.value > self.max: self.value = self.max

21          return self.value

23  if __name__ == "__main__":
        import time
25
        temperature = virtual_sensor(start=20, variation = 0.1)
27      pressure    = virtual_sensor(start=1000, variation = 1)
        humidity    = virtual_sensor(start=30, variation = 3, min=20, max=80)
29
        while True:
31          t = temperature.read_value()
            p = pressure.read_value()
33          h = humidity.read_value()

35          print ("{:7.3f} {:10.3f} {:7.3f}".format(t, p, h))

37          time.sleep(1)
```

The module **virtual_sensor** with the class of the same name, reflects in a more or less realistic way the behavior of a sensor. We can see in the main program (line 23 to 37) that we have created three virtual sensors : one for temperature (line 31), another for pressure (line 32), and the third for humidity (line 34). The argument `start` specifies the starting value, `variation` the range of variation between two measurements and `min` and `max` the values not to exceed. The endless loop that displays the different values every second.

```
> python3 virtual_sensor.py
 54.956    999.850   30.609
 54.963   1000.473   32.505
 55.062   1000.845   31.870
 55.017   1001.619   32.257
 55.083   1000.767   31.757
 55.027   1001.442   31.742
```

Direct sending

Listing 7.3 – minimal_client1.py

```
1  from virtual_sensor import virtual_sensor
   import time
3  import socket

5  temperature = virtual_sensor(start=20, variation = 0.1)
   s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
7
   while True:
```

```
    t = temperature.read_value()

    s.sendto (t, ("127.0.0.1", 33033))
    time.sleep(10)
```

Scripts using this module can be written, as for example the program `minimal_client1.py`.

The variable `t` contains the temperature which is transmitted on port 33033 at the *loopback* address. We thus obtain a communication between two programs in your computer whose local IP address is `127.0.0.1`. But when you launch the program `minimal_client1.py`, you get the following error :

```
 >python3.5 minimal_client1.py
Traceback (most recent call last):
  File "minimal_client1.py", line 11, in <module>
    s.sendto (t, ("127.0.0.1", 33033))
TypeError: a bytes-like object is required, not 'float'
```

Question 7.1.1: Bug ?

Why doesn't the client program work ?
- ○ The IP address is not correct.
- ○ The data serialization process is missing.
- ○ The variable `t` is not defined.
- ○ The variable `t` cannot be read.

Sending a byte string

Listing 7.4 – minimal_client2.py

```
    t = temperature.read_value()

    s.sendto (str(t).encode(), ("127.0.0.1", 33033))
```

In the program `minimal_client2.py`, the floating number contained in the variable `t` is transformed into character string with the function `Indexstr`, then into bytes string with the method `encode` to be compatible with the argument expected by the method `sendto`. On the server side the function `Indexstr` converts the received byte string into a float.

Sending several values

Listing 7.5 – minimal_client3.py

```
while True:
    t = temperature.read_value()
    p = pressure.read_value()
    h = humidity.read_value()

    msg = "{}, {}, {}".format(t, p, h)
    s.sendto (msg.encode(), ("127.0.0.1", 33033))
```

To send simultaneously the values of the three sensors, the representation via a string is a little more complicated to implement. If the program `minimal_client3.py` uses `format` to send data

separated by vigulas. On the server side, you have to decode this string to find the integers. And this is much more complex to do !

JSON

Listing 7.6 – minimal_client4.py

```
12 while True:
       t = temperature.read_value()
14     p = pressure.read_value()
       h = humidity.read_value()
16
       j = [t, p, h]
18     s.sendto (json.dumps(j).encode(), ("127.0.0.1", 33033))
       time.sleep(10)
```

The simplest solution is to put these three values in a python array and transform it into a JSON representation using the `dumps` function of the `json` module.

This JSON string is in turn transformed into a byte string with encoding and sent to the server. In our case, the server only displays the string but you can use the `loads` method of the `json` module to deserialize it and turn it into a Python structure in the server, on which it is now easy to perform operations such as, for example, an average calculation.

```
% python3 minimal_server.py
b'[19.93044784157464, 999.1552628155773, 35.723583473834566]' => b'5b31392e393330343437383431353734363 42c203939392e313535c...
b'[19.940155545405723, 998.7581534530281, 35.820037116376184]' => b'5b31392e393430313535353435343053732332c203939382e3735...
b'[20.003803212269627, 999.3517302791449, 34.33544522779677]' => b'5b32302e303033383033323132323639363237 2c203939392e33353...
```

7.2 CBOR

The passage from JSON to CBOR is very simple. It is enough to change a module `cbor` instead of the module of `json`. The program `minimal_client5.py` :

- line 1 calls the cbor2 module and renames it cbor
- The rest of the program is identical to the previous one, it is only in the serialization, line 18, that the function `json.dumps` is replaced by `cbor.dumps`. The format returned being a string of bytes, it is no longer necessary to call the `encode` method.

Youtube

The program `minimal_server.py` is not modified since it only displays what it receives.

```
> python3 minimal_server.py
b'\x83... => b'83fb40341086f3e8b66bfb408f3b7791c8d61ffb403fac15ba06088e'
b'\x83... => b'83fb40341d4d28495268fb408f33d502185c3dfb403d95a2c4981444'
```

Listing 7.7 – minimal_client5.py

```
1 from virtual_sensor import virtual_sensor
  import time
3 import socket
  import cbor2 as cbor
5
```

```
temperature = virtual_sensor(start=20, variation = 0.1)
pressure    = virtual_sensor(start=1000, variation = 1)
humidity    = virtual_sensor(start=30, variation = 3, min=20, max=80)

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

while True:
    t = temperature.read_value()
    p = pressure.read_value()
    h = humidity.read_value()

    j = [t, p, h]
    s.sendto (cbor.dumps(j), ("127.0.0.1", 33033))
    time.sleep(10)
```

We get the following result. The CBOR sequence is 28 bytes long, the JSON equivalent would have been 60 bytes. Even if this divides by two the size of the data to be transmitted, the result is not compact. This is due to the representation of floats in CBOR, because here floats are coded on 8 bytes.

Question 7.2.1: Decoding

Let us analyze the received sequence : `83fb40341086f3e8b66bfb408f3b7791c8d61ffb403fac15ba06088e`

What does the byte 0x83 that starts the received CBOR structure correspond to ?

- to the coding of the positive integer 131.
- to the coding of the negative integer 132.
- to the definition of an array of 3 elements.
- the definition of a CBOR map of 3 elements.
- to the definition of an array of undefined size.

Question 7.2.2: Floating

In this structure, what is the CBOR marker (in hexadecimal) that indicates that we have a floating number ?

Question 7.2.3: Floating size

What is the size of this floating number in bytes ?

Use of integers

To reduce the size of the transmitted data, we will use integers. We will need a precision to the hundredth (two digits after the decimal point). To do this, on the client side, we just take the integer part of the number multiplied by 100 and on the server side, we divide the received value by 100. The modification of the code is minor.

Listing 7.8 – minimal_client6.py

```
    j = [int(t*100), int(p*100), int(h*100)]
    s.sendto (cbor.dumps(j), ("127.0.0.1", 33033))
```

```
> python3 minimal_server.py
b'\x83\x19\x07\xd7\x1a\x00\x01\x86O\x19\x0c\xa7' => b'831907d71a0001864f190ca7'
b'\x83\x19\x07\xd4\x1a\x00\x01\x86f\x19\x0cJ' => b'831907d41a00018666190c4a'
b'\x83\x19\x07\xd4\x1a\x00\x01\x86\x92\x19\rP' => b'831907d41a00018692190d50'
```

The change is minor and fits in 12 bytes, but there is a decrease in interoperability, as both entities need to know the transformation of the value related to the multiplication by 100.

Question 7.2.4:

What is the minimum and maximum size of the CBOR structure sent, taking into account the possible values.

8. Time series

The virtual sensors that we have programmed so far emit data each time they have a measurement. This allows the server to follow the behavior of the studied system in real time. But in some cases, real time is not necessary and it is preferable to limit the number of emissions, for example to save the energy of the sensor.

To do this, we can use an array that will accumulate the values and send it when the array reaches a certain size.

Youtube

8.1 Sending an array

The program `minimal_humidity1.py` accumulates the data in an array `humidity` when this one reaches 30 elements (line 17), the data are sent to the server.

Listing 8.1 – minimal_humidity1.py

```
from virtual_sensor import virtual_sensor
import time
import socket
import cbor2 as cbor

humidity    = virtual_sensor(start=30, variation = 3, min=20, max=80)

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
NB_ELEMENT = 30
h_history = []

while True:

    h = int(humidity.read_value()*100)

    # No more room to store value, send it.
```

```
    if len(h_history) >= NB_ELEMENT:
18      s.sendto (cbor.dumps(h_history), ("127.0.0.1", 33033))
        h_history = []
20
    h_history.append(h)
22  print (len(h_history), len(cbor.dumps(h_history)), h_history)

24  time.sleep(10)
```

```
1 4 [3241]
2 7 [3241, 2945]
3 10 [3241, 2945, 2762]
4 13 [3241, 2945, 2762, 2625]
5 16 [3241, 2945, 2762, 2625, 2480]
6 19 [3241, 2945, 2762, 2625, 2480, 2769]
```

The first digit of the line indicates the number of elements and the second the size in the CBOR coding. We notice that adding an element increases the size of the array by 3 bytes. The values corresponding to a moisture measurement do not vary greatly. Thus an array of 30 measurements has a size of 92 bytes.

8.2 Differential coding

The amount of data transferred can be optimized by using delta coding (i.e. the variation in humidity). The first value in the table is the measured value while the following values represent the difference between the measured value and the previous one.

Listing 8.2 – minimal_humidity2.py

```
    if len(h_history) == 0:
18      h_history = [h]
    elif len(h_history) >= NB_ELEMENT:
20      print ("send")
        s.sendto (cbor.dumps(h_history), ("127.0.0.1", 33033))
22      h_history = [h]
    else:
24      h_history.append(h-prev)

26  prev = h
```

The program `minimal_humidity2.py` handles the filling of the array differently :

— line 14 and 15, if the table is empty, the table is created with the measured value,
— line 16 to 22, otherwise if the table is full, it is serialized in CBOR and sent to the server, then reset with the measured value,
— line 23 and 24, otherwise the difference between the previous value and the measured value is stored in the table.

The following listing gives an example of execution.

```
1 4 [2521]
2 6 [2521, 79]
3 8 [2521, 79, 224]
4 10 [2521, 79, 224, -40]
```

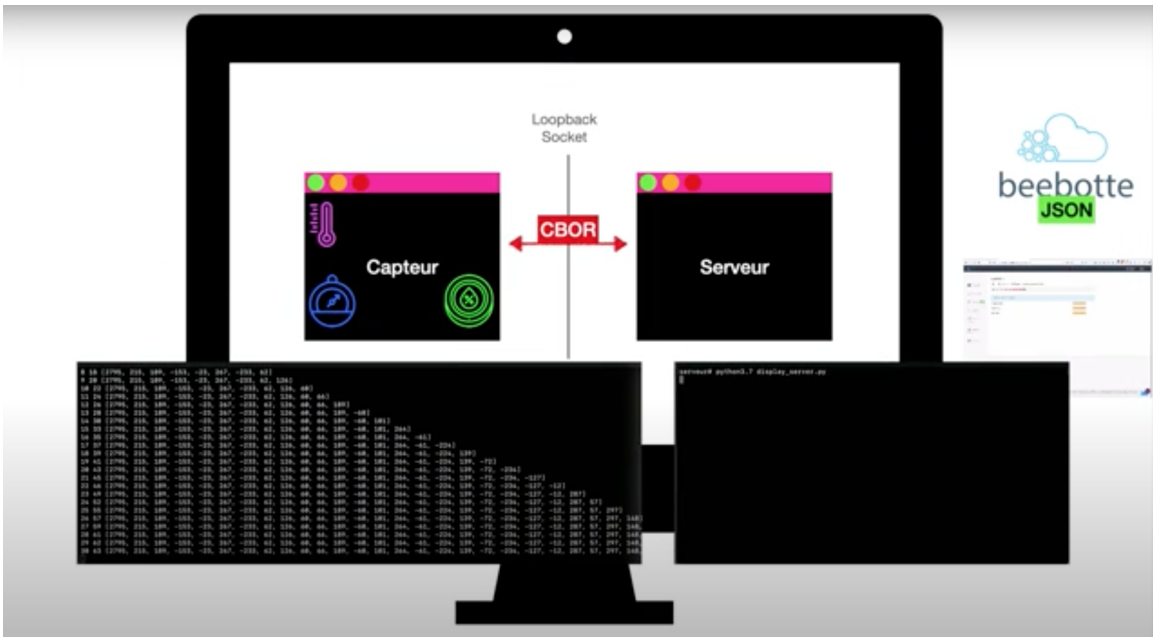


FIGURE 8.1 – Client/Server Architecture

```
5 12 [2521, 79, 224, -40, -112]
6 13 [2521, 79, 224, -40, -112, 1]
7 15 [2521, 79, 224, -40, -112, 1, 130]
8 18 [2521, 79, 224, -40, -112, 1, 130, -288]
9 21 [2521, 79, 224, -40, -112, 1, 130, -288, 299]
```

This highlights two flexibilities of CBOR :

— the size of the table is dynamic. If the number of values to be transmitted is changed, the array indicates this and there is no need to modify the receiver code ;

— the size of the data depends on its value. For variations between -24 and +23, only one byte will be necessary. We can see it on the previous example : the addition of the value '1' in the table increases the size of the CBOR representation from 12 to 135 bytes. The values between 256 and +255 are transmitted on 2 bytes ; it is thus possible in this way to optimize the transmission without adding constraints. If there was a sudden change in humidity, the CBOR representation would adapt to transmit it.

The size is reduced by one third (about 66 bytes) to transmit the same information.

8.3 Architecture

Figure **??** on page ?? represents the general architecture of the system. The program $\texttt{minimal}_h umidity2.py$ provides the time series. It remains to define the server program which will process them and call another service to display them in the form of a graph.

If we follow the information flow, the sensor will produce data in CBOR format to be compacted and the server program will transform this information into a JSON structure respecting the specifications of the display service.

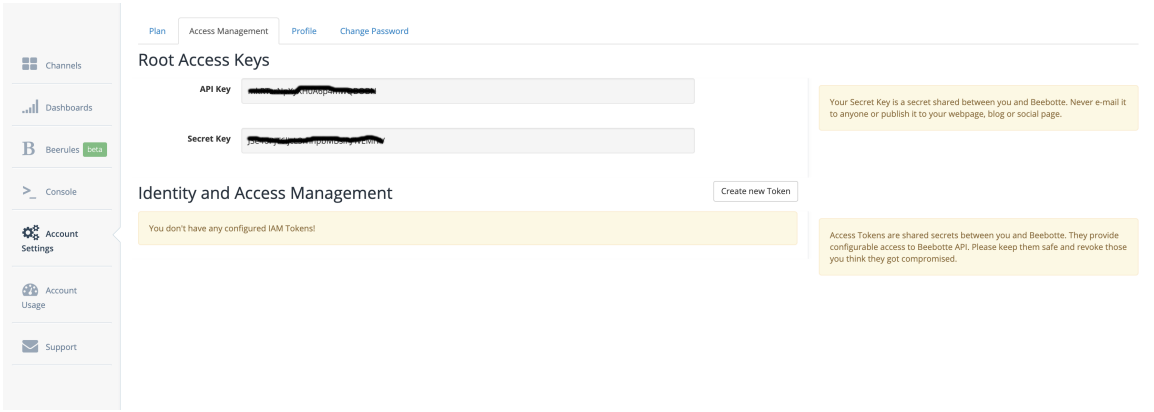


FIGURE 8.2 – Key and secret for authentication

8.4 Beebotte

There are several sites that allow you to do this. We are going to use `https://beebotte.com`, but what we are going to present can very well be applied to other sites.

8.4.1 Configuration

The first step is to create an account by clicking on *Sign Up* on the front page and filling out a standard form with your login, email address and password. Once the account is validated, the service is accessible.

The account allows us to authenticate ourselves to manage the data on the site, but we also need to have authorization to be able to deposit data via the **API REST**. To do this, you must go to the *Account Setting* page and then the *Access Management* tab. This page (see figure 8.2) gives a key and a secret to manage all the data on the site.

Note these values and store them in a file called `config_bbt.py` that looks like this (your values are bound to be different) :

Listing 8.3 – config_bbt.py

```
1 API_KEY    = "GAJ3SFmUZSXmB2zqdcmzcuXc"
  SECRET_KEY = "4NCsrM1cfmFdMZF4E47aTfmCaU3UfyQo"
```

We are now going to create a channel (*channel*) in which we will define the objects corresponding to the sensors. By clicking on *Channels* and then *Create New*, the page shown in figure reffig-new-channel appears.

You have to give a name to the channel (*sensors* in the example), check the *public* box and create three resources for the three values we are interested in (*temperature*, *humidity*, *presure*) and match the units.

8.4.2 Resource registration

The program `display_server.py` allows to correspond with Beebotte via its REST API. It starts by importing the necessary modules :

Listing 8.4 – display_server.py

```
import socket
import binascii
import cbor2 as cbor
import beebotte
import config_bbt #secret keys
import datetime
import time
import pprint
```

— line 4, the Python module `beebotte` is available to simplify the manipulation of data [1].
— line 5, the module `contains` the key and the secret necessary to the connection obtained previously.

```
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(('0.0.0.0', 33033))
```

— line 10 and 11 allow to open the socket to communicate with the sensors.

```
bbt = beebotte.BBT(config_bbt.API_KEY, config_bbt.SECRET_KEY)
```

— line 13 an instance allowing the connection with the Beebotte servers is defined thanks to the function `BBT`. The connection parameters from the module `config_bbt` are taken into account.

```
while True:
    data, addr = s.recvfrom(1500)

    j = cbor.loads(data)
    to_bbt("capteurs", "temperature", j, factor=0.01)
```

In the main program, an endless loop waits for the CBOR coded time series coming from the sensor (line 42), transforms them into a Python array (line 44) and calls the function `to_btt` by specifying :

— the channel and resource that were previously defined on Beebotte ;
— the time series ;
— the precision to transform these integers into floats.

```
def to_bbt(channel, res_name, cbor_msg, factor=1, period=10, epoch=None):
    global bbt

    prev_value = 0
    data_list = []
    if epoch:
        back_time = epoch
    else:
        back_time = time.mktime(datetime.datetime.now().timetuple())

    back_time -= len(cbor_msg)*period

    for e in cbor_msg:
```

1. If it was not present on your computer, you should install it with the command `pip3 install beebotte`.

```
28          prev_value += e

30          back_time += period

32          data_list.append({"resource": res_name,
                              "data" : prev_value*factor,
34                            "ts": back_time*1000} )

36      pprint.pprint (data_list)

38      bbt.writeBulk(channel, data_list)
```

The function `to_bbt` does most of the transformation work. It takes as argument :

— the name of the channel created on Beebotte. In our case, it will be `sensors` ;
— the name of the object in this channel that we have also created on the website. In our case, it will be `humidity` ;
— the Python table of delta-coded measurements ;
— the multiplicative factor, i.e. the precision. Here, you will have to divide by 100 ;
— the period between two measurements ; this will allow us to calculate the time of the measurement. By default, the period is 10 seconds ;
— the time of reception of the message to date the samples. If it is not specified, the current time is taken.

This function transforms the following Python table :

```
[3311, 124, -144, -188, -94, 289, -1, -72, 1 ...
```

into a dictionary table :

```
[{'data': 33.11, 'resource': 'humidity', 'ts': 1596730115000.0},
 {'data': 34.35, 'resource': 'humidity', 'ts': 1596730125000.0},
 {'data': 32.91, 'resource': 'humidity', 'ts': 1596730135000.0},
 {'data': 31.03, 'resource': 'humidity', 'ts': 1596730145000.0},
 ...
```

Each dictionary contains three elements imposed by Beebotte :

— the name of the resource (`resource`) as defined on the interface for the channel ;
— the associated value for this resource (`data`) ;
— the time at which this measurement was made (`ts`). The time is represented according to the format **Epoch** which counts the number of seconds since the first of January 1970 [2].

The calculation of the timestamp (`ts`) is the most complex operation of this function but the modules `time` and `datetime` facilitate the calculation. If the argument `epoch` has been provided at the time of the call, the function takes this value, otherwise it calculates it on line 23. The function `now` returns the current date and time, which is transformed into a tuple by the function `timetuple`. From the latter, the function `maketime` converts it into epoch.

Line 25, the epoch at which the first measurement of the array was made is calculated by taking the current time (this assumes that processing and transmission time are neglected) minus the

2. see `https://www.epochconverter.com/` for the conversions

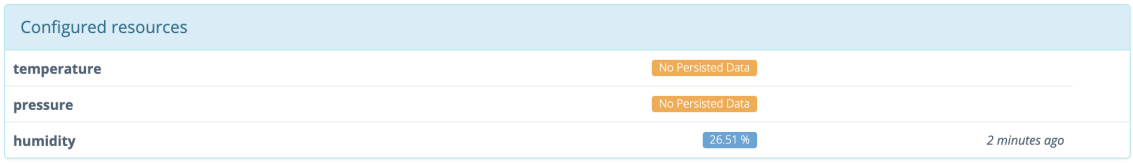


FIGURE 8.3 – Resource status

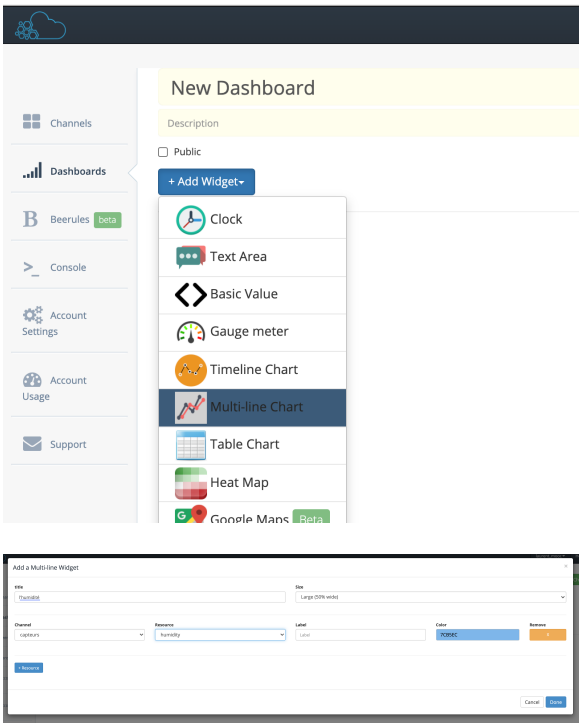


FIGURE 8.4 – Creating a widget

duration of the capture, i.e. as the number of elements in the array multiplied by the interval between each measurement (`period`).

Line 32 to 34 the structure expected by Beebotte is built. The result is sent, line 38, thanks to the function `writeBulk` which allows to send a set of values in an array.

We can verify that Beebotte has received data by viewing the sensor channel on the web interface. We can see on the figure **??** on page ?? that only the resource `humidity` has received data. The interface displays the last value received and the date of reception.

8.4.3 Resource visualization

Now that the resources are stored in the Beebotte servers, it is possible to visualize them graphically, by going to *Dashboard* and then *create Dashboard* and *Add Widget* to select a widget such as *Multi-line chart*.

Then, configure the widget by setting the channel and the resource for that channel as shown in figure 8.4.

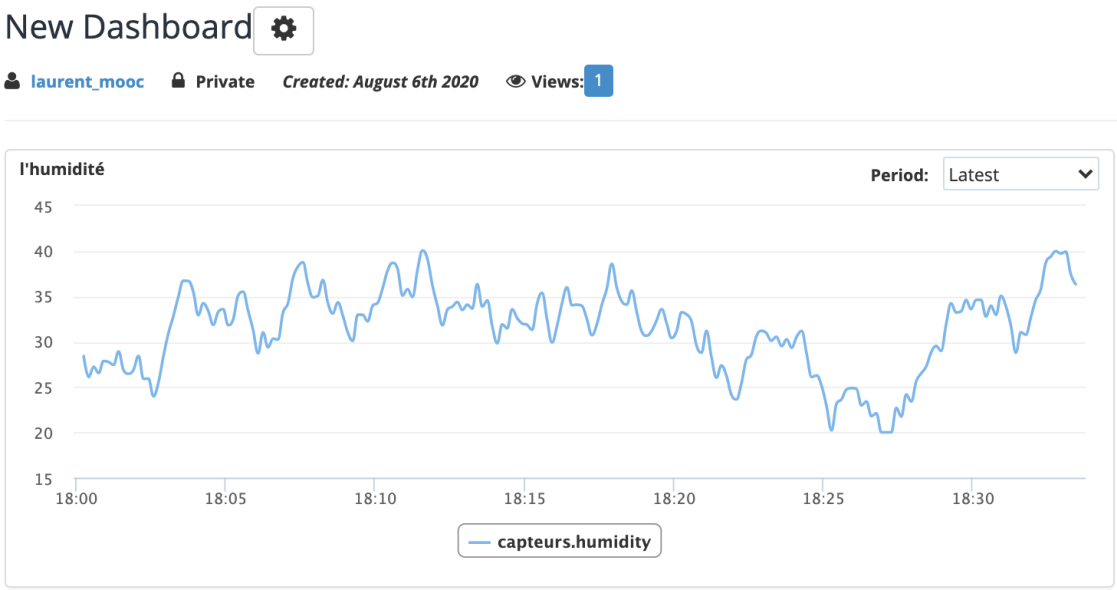


FIGURE 8.5 – Humidity monitoring

By going back to the dashboard, we can see the evolution of the humidity over time (see figure 8.5).

8.5 Interoperability

The chain of information collection that we have just built, from the sensor to the display, is not completely interoperable. Certainly the sensor sends data in CBOR format that can be interpreted by the other end, but the receiver does not know :

—
—
—
—

This information was specified in the program `display_server.py`, as well as the transformation of the table structure of the time series into a dictionary with keywords specific to Beebotte was engraved in the program.

We will see later on how to improve this interoperability.

8.6 and SenML ?

In the communication with Beebotte, the site structures the sending of measurements by defining a JSON dictionary with particular keywords. To use another site, the exchange format must be modified even if the information remains identical.

In addition, when configuring the resources on the Beebotte site, the nature of the measurement had to be specified ; for example, whether it is a temperature, a humidity level... It is also sometimes

necessary to indicate the type of measurement (text, integer, float...) and even the units.

Sensor Measuring List (SenML) defined in the RFC 8428 structures the data provided by the sensor. To reduce the impact of the transmission, the field names have been chosen to be as compact as possible. For example, the letter `v` will indicate a value (to be compared with the key `data` used during the communication with Beebotte). To be even more compact, the representation in CBOR will use short integers instead of characters.

It is also possible to transport the unit of measurement with the keyword `u` .

SenML does not only define units of the international system, but also secondary units to limit the size of the representation. It will be more compact to transmit :

```
{"u": "MHz", "v": 868}
```

than

```
{"u": Hz", "v": 868000000}.
```

The standard also defines base times and base values to which the times and values will refer ; this also makes it possible to reduce the size of the values. Finally, the object(s) can be identified in the transmitted data by defining a base name (`bn` : *base name*), the name of the sensor (`n` : *name*) completes the base name.

Sending

Listing 8.5 – minimal_senml_client.py

```
from virtual_sensor import virtual_sensor
import time
import socket
import json
import kpn_senml as senml
import pprint
import binascii
import datetime
import time
import pprint


NB_ELEMENT = 5


temperature = virtual_sensor(start=20, variation = 0.1)
pressure    = virtual_sensor(start=1000, variation = 1)
humidity    = virtual_sensor(start=30, variation = 3, min=20, max=80)

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

while True:
    pack = senml.SenmlPack("device1")
    pack.base_time = time.mktime( datetime.datetime.now().timetuple())

    for k in range(NB_ELEMENT):
        t = round(temperature.read_value(), 2)
        h = round(humidity.read_value(), 2)
        p = int(pressure.read_value() *100) # unit is Pa not hPa
```

```
31
        rec = senml.SenmlRecord("temperature",
33          unit=senml.SenmlUnits.SENML_UNIT_DEGREES_CELSIUS,
            value=t)
35      rec.time = time.mktime(datetime.datetime.now().timetuple())
        pack.add(rec)
37
        rec = senml.SenmlRecord("humidity",
39          unit=senml.SenmlUnits.SENML_UNIT_RELATIVE_HUMIDITY,
            value=h)
41      rec.time = time.mktime(datetime.datetime.now().timetuple())
        pack.add(rec)
43
        rec = senml.SenmlRecord("pressure",
45          unit=senml.SenmlUnits.SENML_UNIT_PASCAL,
            value=p)
47      rec.time = time.mktime(datetime.datetime.now().timetuple())
        pack.add(rec)
49
        time.sleep(10)
51
        pprint.pprint(json.loads(pack.to_json()))
53      print ("JSON length:", len(pack.to_json()), "bytes")
        print ("CBOR length:", len(pack.to_cbor()), "bytes")
55
    s.sendto(pack.to_cbor(), ("127.0.0.1", 33033))
```

The program `minimal_senml_client.py` illustrates how SenML works. It is based on two objects :

— the object `SenmlPack` includes information common to the object, such as the base name (here `device1` line 23) or the time base, line 24.

— the object `SenmlRecord` contains a measure where we can specify its name, its unit and its value (lines 32, 38 and 44). The time is also specified on lines 35, 41 and 47. These records are added to the `pack` object.

The program retrieves the three values of temperature, humidity and pressure (lines 27 to 29) by rounding them to 2 digits after the decimal point for temperature and humidity and converts the pressure from hecto Pascal to Pascal since this is the unit defined by SenML.

Measurements are made every 10 seconds (delays line 48) and when the number of measurements defined on line 13 is reached, the SenML coding in CBOR is sent to the server.

```
[{'bn': 'device1',
  'bt': 1650463643.0,
  'n': 'temperature',
  't': 0.0,
  'u': 'Cel',
  'v': 20.08},
 {'n': 'humidity', 't': 0.0, 'u': '%RH', 'v': 31.1},
 {'n': 'pressure', 't': 0.0, 'u': 'Pa', 'v': 99920}]
JSON length:  197 bytes
CBOR length:  126 bytes
```

This first listing shows the first record for the three measured quantities. It is a table of 3 elements. The first contains the basic values (here the name and the reference time) followed by the quantity to

be measured, its unit and its value. The second and third elements update the name of the quantity, its unit and its value, the other information previously defined remains valid.

```
[{'bn': 'device1',
 'bt': 1650463643.0,
 'n': 'temperature',
 't': 0.0,
 'u': 'Cel',
 'v': 20.08},
 {'n': 'humidity', 't': 0.0, 'u': '%RH', 'v': 31.1},
 {'n': 'pressure', 't': 0.0, 'u': 'Pa', 'v': 99920,}
 {'n': 'temperature', 't': 10.0, 'u': 'Cel', 'v': 20.04},
 {'n': 'humidity', 't': 10.0, 'u': '%RH', 'v': 31.49},
 {'n': 'pressure', 't': 10.0, 'u': 'Pa', 'v': 99872}]
JSON length:  361 bytes
CBOR length:  232 bytes
```

When new measurements are added 10 seconds later, a relative time of 10 seconds is indicated for the temperature recording and it remains valid for the following recordings.

Question 8.6.1: Encoding

What does the key `{'u' : 'Cel'}` found in the previous structure correspond to ?

Question 8.6.2: Growth

In both JSON and CBOR representations, how much is the size increased by adding the measurements made ? Where do these differences come from ?

Question 8.6.3: A single value

If we were only interested in one quantity, for example humidity. What would the SenML structure look like in JSON ?

Receiving

The processing by the SenML module as implemented is not complete, it does not handle timestamps correctly. But, it is not really necessary to process these messages. Indeed, as we have seen previously, SenML values are composed of a base and a value. By concatenating the different elements of the array, we keep the base keys and we replace the keys which are repeated at each element. This allows to have a very simple implementation of the decoding, for the SenML series which is provided to us.

Listing 8.6 – minimal_senml_server.py

```
1 import socket
  import pprint
3 import binascii
  import pprint
5 import cbor2 as cbor

7 import beebotte
  import config_bbt #secret keys
```

```
9
naming_map = {'bn': -2, 'bt': -3, 'bu': -4, 'bv': -5, 'bs': -16,
11                      'n': 0, 'u': 1, 'v': 2, 'vs': 3, 'vb': 4,
                        'vd': 8, 's': 5, 't': 6, 'ut': 7}
13
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
15 s.bind(('0.0.0.0', 33033))

17 bbt = beebotte.BBT(config_bbt.API_KEY, config_bbt.SECRET_KEY)

19
while True:
21     data, addr = s.recvfrom(1500)

23     sml_data = cbor.loads(data)

25     sml_record = {}
       bbt_record = []
27
       for e in sml_data:
29         sml_record = {**sml_record, **e} # merge dict
           print (sml_record)
31
           ts =  sml_record[naming_map["t"]]
33         if naming_map["bt"] in sml_record:
               ts += sml_record[naming_map["bt"]]
35
           res = sml_record[naming_map["n"]]
37
           data = sml_record[naming_map["v"]]
39         if naming_map["bv"] in sml_record:
               data += sml_record[naming_map["bv"]]
41

43         bbt_record.append({"resource" : res, "data": data, "ts": ts*1000})

45     pprint.pprint (bbt_record)
       channel = sml_record[naming_map["bn"]]
47     bbt.writeBulk(channel, bbt_record)
```

The program `minimal_senml_server.py` will convert the SenML format encoded in CBOR into the format expected by Beebotte. The CBOR version uses numbers rather than tags. The dictionary `naming_map` defined on lines 10 to 12 allows the correspondence used afterwards to make the code more readable.

Lines 14 to 17 initialize the communications coming from the sensor and those going to Beebotte.

The data received on line 21 is transformed into a Python structure on line 23. This correspondence is possible because Python allows numerical keys and these are not repeated several times in a CBOR map.

The loop starting on line 28 allows you to explore all the elements of the SenML array, the new entries are merged with the old ones (line 29) [3]

The information about the time is then searched for. First the time (line 32) and if there is a basic time (line 33) it is added. The same procedure is followed for the value (lines 38 to 40). For the

3. In more recent versions of Python, it is possible to use the | operator.

name, there is no concatenation because the basic name will be used as a Beebotte channel, it is recovered at the end of line 46.

From this information, the structure expected by Beebotte is constructed line 43 by adding the dictionary in the table `bbt_record`.

Line 47, the information is sent to Beebotte. If the authentication keys, channel name and resources are correct, the information is displayed on the site, as before.

```
{0: 'temperature', 2: 19.94, 6: 0.0, 1: 'Cel', -2: 'device1', -3: 1650463732.0}
{0: 'humidity', 2: 27.7, 6: 0.0, 1: '%RH', -2: 'device1', -3: 1650463732.0}
{0: 'pressure', 2: 100109, 6: 0.0, 1: 'Pa', -2: 'device1', -3: 1650463732.0}
{0: 'temperature', 2: 19.88, 6: 10.0, 1: 'Cel', -2: 'device1', -3: 1650463732.0}
{0: 'humidity', 2: 24.82, 6: 10.0, 1: '%RH', -2: 'device1', -3: 1650463732.0}
{0: 'pressure', 2: 100056, 6: 10.0, 1: 'Pa', -2: 'device1', -3: 1650463732.0}
{0: 'temperature', 2: 19.93, 6: 20.0, 1: 'Cel', -2: 'device1', -3: 1650463732.0}
{0: 'humidity', 2: 23.74, 6: 20.0, 1: '%RH', -2: 'device1', -3: 1650463732.0}
{0: 'pressure', 2: 100123, 6: 20.0, 1: 'Pa', -2: 'device1', -3: 1650463732.0}
{0: 'temperature', 2: 19.92, 6: 30.0, 1: 'Cel', -2: 'device1', -3: 1650463732.0}
{0: 'humidity', 2: 25.82, 6: 30.0, 1: '%RH', -2: 'device1', -3: 1650463732.0}
{0: 'pressure', 2: 100220, 6: 30.0, 1: 'Pa', -2: 'device1', -3: 1650463732.0}
{0: 'temperature', 2: 19.9, 6: 40.0, 1: 'Cel', -2: 'device1', -3: 1650463732.0}
{0: 'humidity', 2: 24.47, 6: 40.0, 1: '%RH', -2: 'device1', -3: 1650463732.0}
{0: 'pressure', 2: 100173, 6: 40.0, 1: 'Pa', -2: 'device1', -3: 1650463732.0}
[{'data': 19.94, 'resource': 'temperature', 'ts': 1650463732000.0},
 {'data': 27.7, 'resource': 'humidity', 'ts': 1650463732000.0},
 {'data': 100109, 'resource': 'pressure', 'ts': 1650463732000.0},
 {'data': 19.88, 'resource': 'temperature', 'ts': 1650463742000.0},
 {'data': 24.82, 'resource': 'humidity', 'ts': 1650463742000.0},
 {'data': 100056, 'resource': 'pressure', 'ts': 1650463742000.0},
 {'data': 19.93, 'resource': 'temperature', 'ts': 1650463752000.0},
 {'data': 23.74, 'resource': 'humidity', 'ts': 1650463752000.0},
 {'data': 100123, 'resource': 'pressure', 'ts': 1650463752000.0},
 {'data': 19.92, 'resource': 'temperature', 'ts': 1650463762000.0},
 {'data': 25.82, 'resource': 'humidity', 'ts': 1650463762000.0},
 {'data': 100220, 'resource': 'pressure', 'ts': 1650463762000.0},
 {'data': 19.9, 'resource': 'temperature', 'ts': 1650463772000.0},
 {'data': 24.47, 'resource': 'humidity', 'ts': 1650463772000.0},
 {'data': 100173, 'resource': 'pressure', 'ts': 1650463772000.0}]
```

The previous listing shows this transformation. The first rows are the merged records and the final table is what was sent to Beebotte.

Question 8.6.4: base value

Could we use the SenML field *base value* to decrease the size of the air pressure data ?

9. Discover the LoPy

The programs related to this section are located in the `plido-tp3` directory for the server and `pycom` for the LoPy.

9.1 Introduction

Using the sensor emulators described in the previous chapter, you have been able to apply the essential IoT concepts to your computer.

However, if you can, we invite you to do it on real connected objects using **LoPy4** (IoT prototyping platform) from the company **Pycom** and temperature, humidity and pressure sensors **BME280** (see figure 9.1).

A LoPY4 is programmed in Python (or rather **micro-python** which is the version of the language for embedded systems) to process data. At first, we will use Wi-Fi to communicate with your computer but, later, we will set up a communication via **LoRaWAN** or **Sigfox** which may require more configuration but will allow you to better understand these protocols.

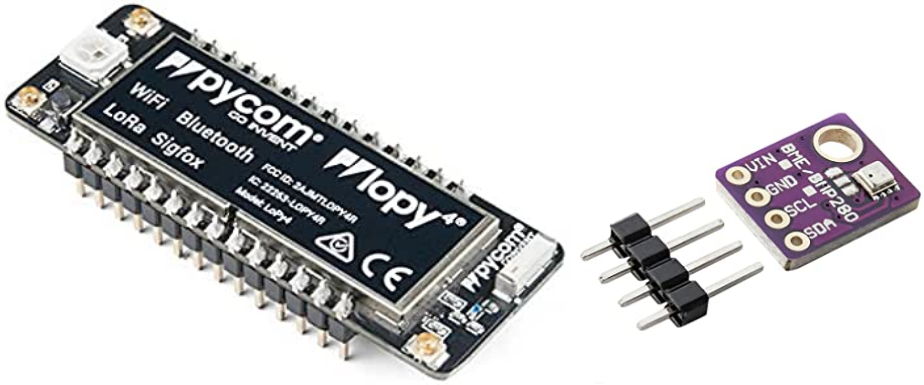


FIGURE 9.1 – LoPY4 and BME280 sensor

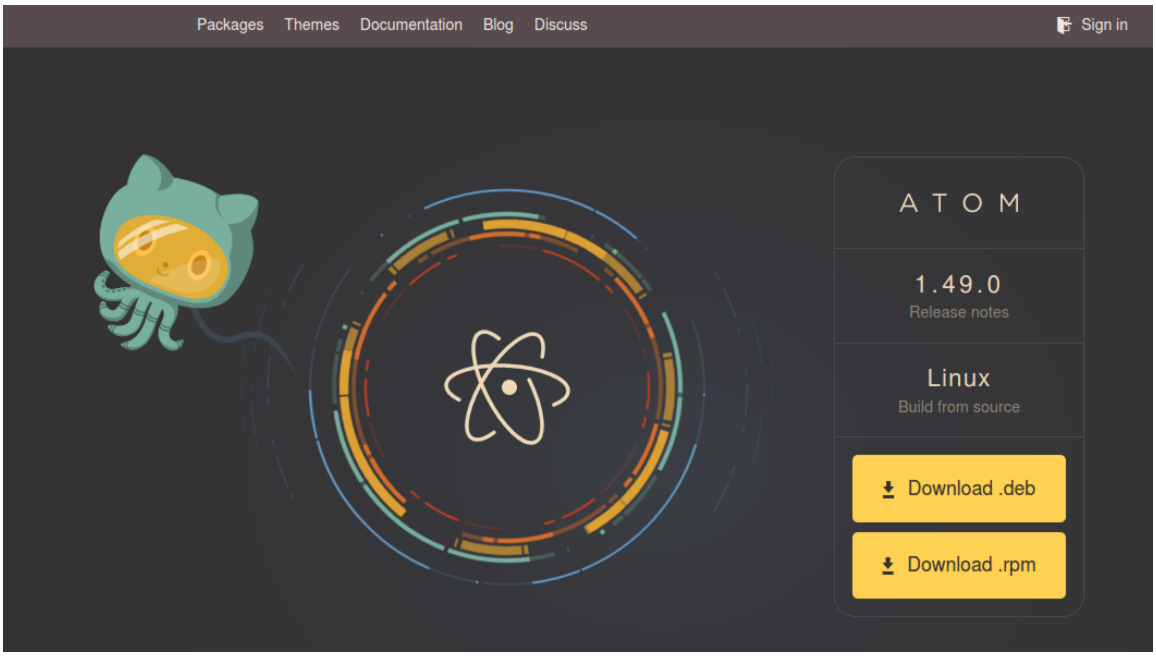


FIGURE 9.2 – Atom homepage

Even if you don't have LoPy4, you can browse this section to see additional constraints related to connected objects.

9.2 Installing Atom

Atom is a powerful text editor, specifically designed for coding in different languages. Atom will help us to program in Python and will also manage the communication with our LoPy via the **USB** (thanks to the plugin **pymakr**). Atom works in much the same way on **Mac OS**, **Windows** and **Linux**, but it adapts to the particularities of the operating system (place in the menus, name of the serial links...). So, you may have some differences between what you have in this book and the screen of your computer. The menus and websites shown may also change over time, although we try to update the course regularly.

To start programming with your LoPy, you must install on your computer the Atom software (see `http://atom.io`). You can download the package corresponding to your operating system (see next figure).

— For Mac and Windows, click on the "download" icon to install it. There is a risk of incompatibility between the latest versions of Atom and the LoPy management package (pymakr). We recommend you to use an older version, like the 1.43 available in the Atom Release archives (`https://github.com/atom/atom/releases/tag/v1.43.0`(`AtomSetup-x64.exe` for Windows or `atom-mac.zip` for Mac OS).

— For Linux, download letexttt.deb and type `sudo dpkg -i atom-amd64.deb` [1]. Launch Atom by clicking on the icon or, under Linux, by typing atom in a terminal.

1. It is possible that a message tells you that git is not installed. In this case, type `sudo apt-get install git` and follow the instructions

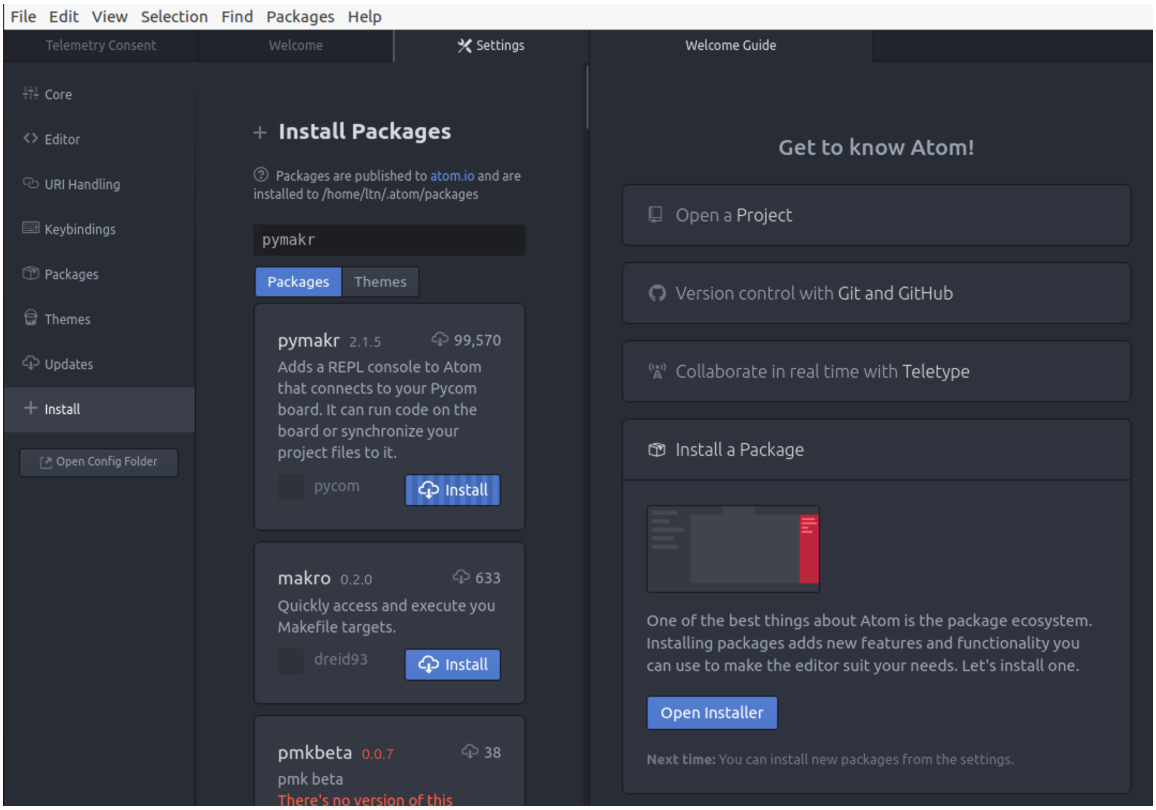


FIGURE 9.3 – Package Installation

Launch Atom by clicking on the icon or, under Linux, by typing atom in a terminal. The welcome screen appears.

9.2.1 Communicate with your Pycom

To communicate with the Pycom through Atom, you must install the `pymakr` package.

Click on *Install a Package* then *Open Installer*. Another window will open (see figure 9.3. Type `pymakr` in the menu. A package with this name appears. Click on *Install*. The installation may take several minutes. You have time to have a coffee.

Once the coffee is drunk and the installation is finished, a new window (terminal) opens at the bottom of Atom.

This terminal (cf.figure 9.4 on the next page) will allow you to dialogue with the LoPy. Connect the LoPy to your computer. You should see the typical Python interpreter prompt >>> [2].

All the commands you type in this window will run on your LoPy. For example, if you type [3] :

2. Under Linux, you must be a member of the dialout group to be able to manage the communication on the USB port. If you do not see the prompt, type `sudo adduser` *`login`* `dialout` replacing login with the name of your Linux account. Reconnect under your account

3. The windows with gray background show the micropython code and their result.

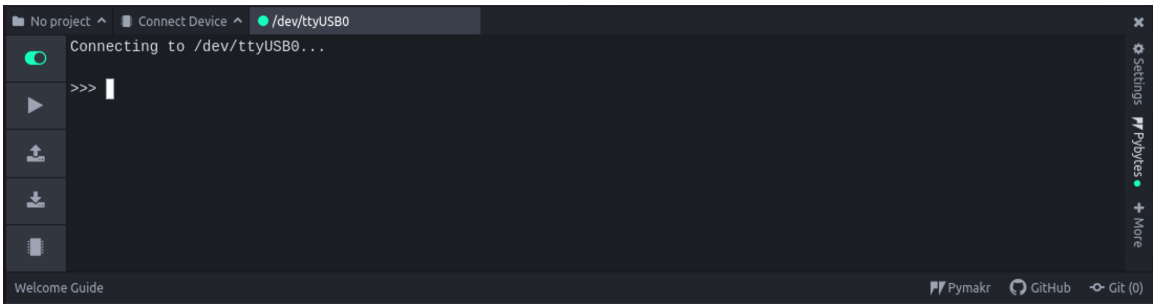


FIGURE 9.4 – Pymakr window

```
Connecting to /dev/ttyUSB0...
>>> 1+1
2
>>>
```

the addition is done on the LoPy.

On the left side of the pymakr window, several icons are present :

— the switch allows to activate or deactivate the connection with the LoPy ;
— the triangle allows you to execute the program displayed in the Atom window on the LoPy ;
— the up arrow, allows you to copy the active directory into the LoPy memory. This will be useful to install new modules on the LoPy ;
— Conversely, the down arrow allows you to copy the memory of the LoPy to the computer ;
— the processor allows to have information about the loPy.

On the right side, the vertical tab *Setting* allows to modify the connection parameters with the LoPy.

9.2.2 Set up your work environment

To program the LoPy, you have to get the micropython modules. The repository can be downloaded in the directory of your choice :

```
> git clone https://github.com/ltn22/PLIDObis.git
```

In Atom's *Files>Open Folder* menu, select the `pycom` directory of the downloaded repository, and validate. On the left side of the screen, all the files composing this directory appear. There are a lot of them, because they will be used later.

By clicking in the window *Indexpymakr* on the button *Upload project to device*, the files of this directory will be copied in the memory of LoPy. Afterwards, if a module is modified, it will have to be resynchronized in the memory of LoPy.

9.3 Connection to the Wi-Fi network

To attach LoPY to a **Wi-Fi** network, it must first be configured via the USB link of the computer to give it the necessary parameters for the connection.

Youtube

The file `Indexboot.py` has been copied when uploading the files in the LoPy memory. When LoPy starts, this program will try to connect to a Wi-Fi network. As the network name and the secret key have not been provided, it does not succeed. The LoPy turns into an access point. At startup, the LoPy should have displayed the following message, indicating that the LoPy becomes a Wi-Fi access point and will deploy its own network on which your computer can connect. The name of this network has the form `PLIDO_XXXX` where `XXXX` is a hexadecimal sequence specific to the equipment. The key is `www.pycom.io`. The LoPy at address `192.168.4.1` on this network.

```
Failed to connect to any known network, going into AP mode
To connect look for 'PLIDO_5bac' access point, key = 'www.pycom.io'
```

But this is not very interesting because your computer will lose its connection to the internet. Not very practical to follow the MOOC. Before displaying this message, the LopY showed the list of Wi-Fi networks it detected. You can connect it to one of these networks by filling in the file `wifi_conf.py` which is in the directory `pycom`.

Listing 9.1 – wifi_conf.py

```
known_nets = {
    'MOM_SSID': {'pwd': 'MON_MOT_DE_PASSE'}
}
```

`MON_SSID` must be replaced by the name of the Wi-Fi network or Service Set IDentifier (SSID) and `MON_MOT_DE_PASSE` by the key associated with it. Note that several Wi-Fi networks can be added, since `MON_SSID` is seen as a key of the JSON object. The file edition was done on the computer, it must be copied in the memory of the LoPy by clicking on the up arrow.

The Pycom restarts and should now display a message like :

```
net to use ['MONWIFI']
Connected to MONWIFI with IP address: 192.168.1.76
Pycom MicroPython 1.20.2.r1 [v1.11-a5aa0b8] on 2020-09-09; LoPy4 with ESP32
Type "help()" for more information.
>>>
```

It is possible to ping or connect with **FTP** or **telnet** using this IP address.

```
# telnet 192.168.1.76
Trying 192.168.1.86...
Connected to 192.168.1.86.
Escape character is '^]'.
MicroPython v1.8.6-760-g90b72952 on 2017-09-01; LoPy with ESP32
Login as: micro
Password: python
Login succeeded!
Type "help()" for more information.
>>>
```

It can be useful to follow the behavior of your object without launching atom if the object is not connected via the USB link to the computer.

Atom can also be configured to use this IP address. The configuration is done in the menu *setting*, and by entering the ip address of the LoPy and disabling *auto connect*. The next time Atom is launched, it will be possible to join the LoPy in Wi-Fi.

9.4 Setting up a client

A relatively simple program allows to check the communication between the LoPy and the server. The command **ifconfig** [4] gives the server IP address. The address must be different from the one we had obtained on the LoPy.

If the server is running in a local environment, the address should start with `192.168` or `10`. If the LoPy and the computer are connected to the same Wi-Fi network, the first digits must be identical.

If the server is running on an external server (i.e. the *cloud*), the address is random.

Youtube

The following command is typed from the terminal of the computer, we notice the two available interfaces one for the Ethernet or Wi-Fi network and one for the ***loopback***, the names of the interfaces can change from one configuration to another :

```
> ifconfig
eth1      Link encap:Ethernet  HWaddr 10:65:30:b0:54:bf
          inet addr:192.168.1.237  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::d8ac:86e7:8bdb:e333/64 Scope:Unknown
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Unknown
          UP LOOPBACK RUNNING  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

The program `minimal_server.py` presented in chapter 7.1.1 on page 82 has not been modified and waits for data on port 33033.

Listing 9.2 – sending_client.py

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto("message", ("192.168.1.237", 33033))
```

The program `sending_client.py` must be modified (line 4) to take into account the IP address of the server obtained with the command `ifconfig`. If at each execution on the LoPy of this program, the server receives the value, the communication is established between the two devices.

```
> python3 minimal_server.py
b'message' => b'6d657373616765'
b'message' => b'6d657373616765'
```

4. Under Linux, it is necessary to add the package **net-tools** `sudo apt install net-tools`.

9.5 BME 280

Instead of generating false data, we will use in this section a real temperature humidity pressure sensor : the BME 280 from Bosch.

Youtube

9.5.1 I2C bus

The I2C bus is standardized by the electronic components manufacturer **NXP** [5] which allows a better interoperability between the components.

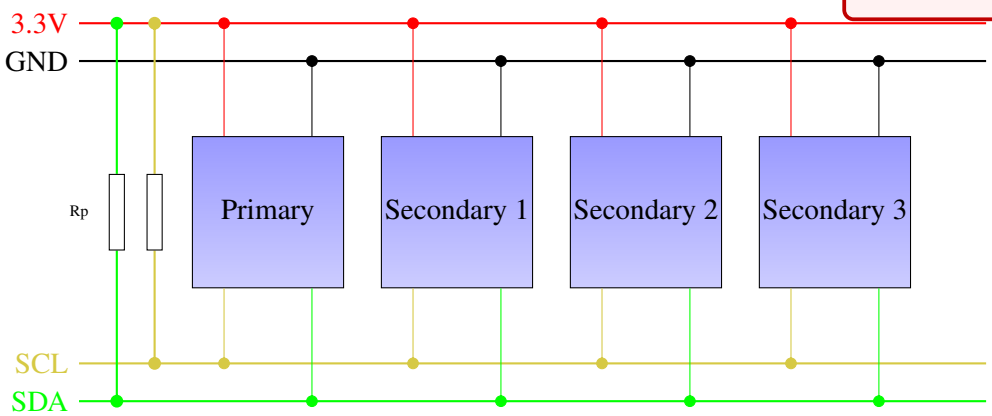


FIGURE 9.5 – I2C bus

On one of the wires the clock signal will be transmitted by the primary. On the other wire, the data will be coded either in the primary/secondary direction, or in the other. As with **Modbus**, the communications between a secondary and the primary will be managed by the primary. Each secondary is configured with a unique address on the bus. Either the master sends data to this address, or the primary interrogates the slave to obtain its data. The wire will thus be exploited in both directions.

The reading of the binary information is done when the clock signal is in the high state (see figure 9.6 on the next page). When the SDA signal is in the high state, a 1 bit is transmitted and in the low state a 0 bit is transmitted. The changes of state of the SDA signal are thus done when the clock signal is in the low state.

However, there are two exceptions : if the DID signal goes from high to low while the clock signal is high, this indicates the start of data transmission. If the DID signal goes from low to high under the same conditions, this indicates the end of data transmission. Between the two the binary data form a frame (or PDU in ISO vocabulary) which is structured as shown in figure 9.6 on the facing page.

Figure **??** on page ?? gives the most used formats.

The first bit stream illustrates the transmission of data from the primary to a secondary. The primary starts by emitting a non-binary signal (S) indicating a start of transmission. The next 7 bits give the address of the secondary and the next bit indicates whether the primary wants to send data (value at 0) or receive information from the secondary (value at 1).

5. urlhttps ://www.nxp.com/docs/en/user-guide/UM10204.pdf

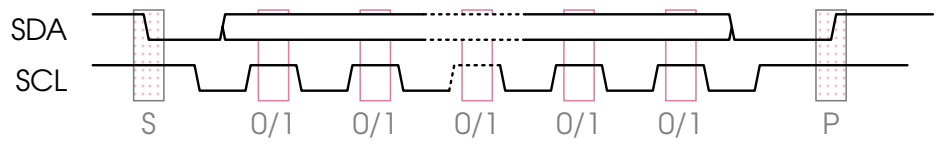


FIGURE 9.6 – Example of communication with the I2C bus

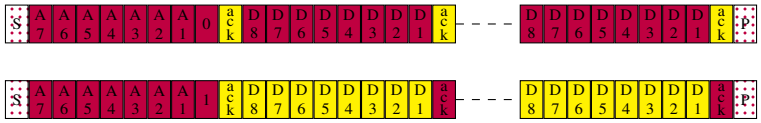


FIGURE 9.7 – Example of communication with the I2C bus

If a secondary recognizes its address on the bus, then it writes the next bit in the bitstream. The primary is thus informed by reading this value that the secondary is indeed present on the bus and that it can receive data. The primary will then send them byte by byte. Each byte is acknowledged in the same way by the secondary. The binary train ends with the non-binary signal (`P`).

In the case where the primary wishes to receive, once the start bit and the address bits have been transmitted, the eighth bit is set to 1. The secondary acknowledges and then transmits its bytes which the primary acknowledges.

Question 9.5.1: scan

The I2C module of LoPy has a function `scan` which displays the addresses of the connected secondaries. How is this detection possible ?

Question 9.5.2: Broadcast

Is the standard an address that allows you to talk to all the secondaries at the same time ?

9.5.2 Temperature measurement

The communication between the LoPy and the component is done via the bus **I2C**. So we need four wires to connect it (see figure 9.8 on the following page :

— the ground (`GND`),
— a 3.3v power supply (`VIN`),
— a wire for the clock (`SCL`) et
— another one finally for the data (`SDA`).

To do this, you must connect :

— the GND of the LoPy to the GND pin of the component with a black wire,
— 3.3V power supply of LoPy (3V3) on VIN of the component with a red wire (this port is also called **DCC** on some boards),
— The clock signal from the G18/P10 port of LoPy (or G17 on LoPy 1 versions) on the SCL port of the component with a yellow wire (this port is also called **CLC** on some boards),
— The data wire from the Pycom G16/P9 port to the SDA port of the component with a green

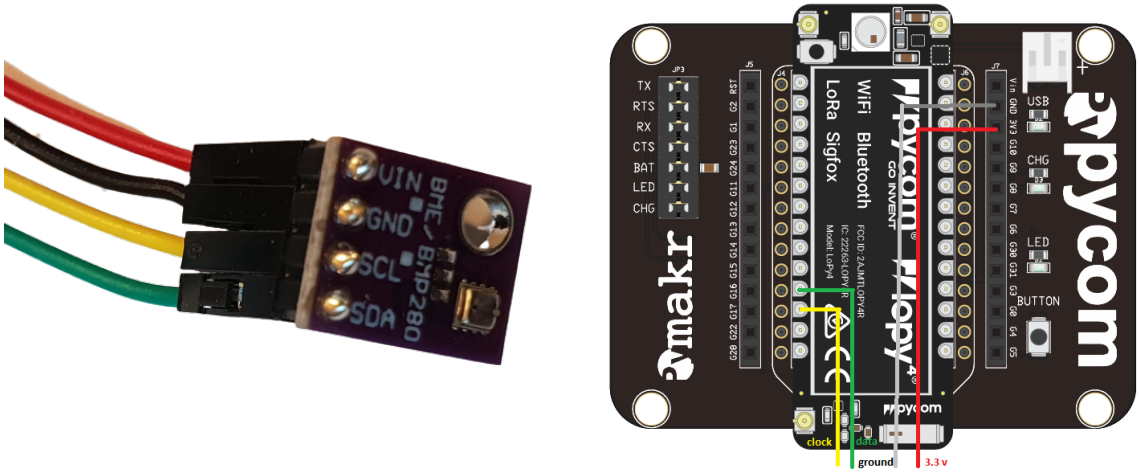


FIGURE 9.8 – BME280 sensor and connectors

wire.

If the BME280 is connected correctly to the LoPy connectors, you should get the following result :

```
>>> Running BME280.py

>>>
>>>
[118]
temp 550576 27.98  - hum 26626 48.784 % - pres 389338 pres 1004.494 hPa [delta -389338 ]
temp 551952 28.42  - hum 26587 48.557 % - pres 389712 pres 1004.527 hPa [delta -374 ]
temp 551792 28.37  - hum 26577 48.491 % - pres 389664 pres 1004.621 hPa [delta 48 ]
temp 551712 28.34  - hum 26594 48.596 % - pres 389648 pres 1004.654 hPa [delta 16 ]
temp 551632 28.32  - hum 26587 48.546 % - pres 389616 pres 1004.713 hPa [delta 32 ]
```

You can either touch the sensor or blow on it to increase the temperature or pressure.

Listing 9.3 – BME280.py

```
282 if __name__ == "__main__":
        from machine import I2C
284     import time

286     i2c = I2C(0, I2C.MASTER, baudrate=400000)
        print (i2c.scan())
288
        bme = BME280(i2c=i2c)
290
        ob = 0
292
        while True:
294         ar = bme.read_raw_temp()
            a = bme.read_temperature()
296
            br = bme.read_raw_pressure()
298         b = bme.read_pressure()

300         cr = bme.read_raw_humidity()
            c = bme.read_humidity()
```

```
302
        print ("temp", ar,  a,
304             "␣-␣hum", cr,  c,
                "%␣-␣pres", br,
306             "pres", b,
                "hPa␣[delta", ob - br, "]" )
308     ob =br

310     time.sleep(5)
```

The program `BME280.py` can work as a module but the last part gives an example of exploitation of the results :

- The main program starts by importing (line 283) the module managing the I2C bus. It is invoked at line 286. The LoPy will manage the communication with the BME280, so the address is `0` and its status is `MASTER`. The communication speed is then specified.
- The next line scans the bus for components. If all goes well, it should find one at address 118, which is the default address of the BME280. [6]. Otherwise, review your wiring.
- The BME280 module is initialized by passing the previously defined I2C bus reference as a parameter.
- The program will then display the values captured by the component. There are two types of values : raw (*raw*) and calibrated. The first ones react more quickly to changes but are much noisier than the second ones which undergo a mathematical processing.
  Thus, the first and second columns give the raw and calibrated temperatures. They are accessed by the methods `read_raw_temp` and `read_temperature`. It is the same for the two other quantities, humidity and pressure.
- The program also displays the raw pressure difference between two measurements. This makes it easier to highlight the fact that the sensor is being blown.

9.6 Wi-Fi thermometer

You now have all the tools to retrieve the temperature of your home, transmit it to your computer via Wi-Fi, and transmit it to Beebotte for display. There is very little change from the fully computerized version. The program to transform the CBOR structure of the time series representation into JSON understandable by Beebottte remains the same. It will just be necessary to modify the name of the humidity to temperature sensor.

Youtube

The program you built in the previous tutorial used the module `virtual_sensor` to produce random time series symbolizing the behavior of a sensor. Now that we have an **BME280**, we will be able to process real values.

The program `wifi_temperature.py` shows this adaptation.

Listing 9.4 – wifi_temperature.py

```
import BME280
2 import time
```

6. If another value is indicated, you can add the parameter `addr=value` to the instantiation of the BME280 module, line 289.

```
import socket
import kpn_senml.cbor_encoder as cbor
from machine import I2C


s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
NB_ELEMENT = 30
t_history = []

i2c = I2C(0, I2C.MASTER, baudrate=400000)
print (i2c.scan())

bme = BME280.BME280(i2c=i2c)

while True:

    t = int(bme.read_temperature()*100)

    # No more room to store value, send it.
    if len(t_history) == 0:
        t_history = [t]
    elif len(t_history) >= NB_ELEMENT:
        print ("send")
        s.sendto (cbor.dumps(t_history), ("192.168.1.47", 33033))
        t_history = [t]
    else:
        t_history.append(t-prev)

    prev = t

    print (len(t_history), len(cbor.dumps(t_history)), t_history)

    time.sleep(10)
```

At the module import level, `BME280` and `indexI2C` replace `virtual_sensor`, and the module `CBOR` is that of `Indexkpn_senml`.

But its behavior remains the same, especially the function `dumps` which converts a Python structure into CBOR. You just have to adapt line 26 to put the IP address of your computer.

On the computer side, you must restart the program `display_server.py`, but changing the name of the sensor from "humidity" to "temperature".

On your Beebotte account, after 300 seconds, you should see data associated with the "temperature" sensor.

Question 9.6.1: changing measurement interval

What happens if in the program `wifi_temperature.py` you modify the measurement step line 36, to put it for example at 60 seconds.

10. Sigfox

Sigfox is one of the very first networks entirely dedicated to the Internet of Things. As we have seen before, it belongs to the **LPWAN** family, which favors range and energy consumption over throughput. The communications are also highly asymmetrical, which means that the exchanges are not like on a Wi-Fi network.

Youtube

LoPy can use the network and benefits from one year of free connectivity on the Sigfox network. After that the subscription costs are relatively limited.

10.1 Retrieving identifiers

First, you must register your sensor on the Sigfox website. You need two elements : its identifier and its password called Porting Authorization Code (PAC). The latter must remain secret because it allows anyone who has it to register an object or change its owner.

The program `sigfox_id.py` allows to display these two values and to send a message on the Sigfox network. Before running it, check that you have connected an antenna on the right connector, the one opposite to the Reset button on the Pycom, LED side.

Listing 10.1 – sigfox_id.py

```
from network import Sigfox
import binascii
import socket

# initalise Sigfox for RCZ1 (You may need a different RCZ Region)
# RCZ1: Europe, Oman, South Africa
# RCZ2: USA, Mexico, Brazil
# RCZ3: Japan
# RCZ4: Australia, New Zealand, Singapore, Taiwan, Hong Kong, Columbia, Argentina
sigfox = Sigfox(mode=Sigfox.SIGFOX, rcz=Sigfox.RCZ1)
```

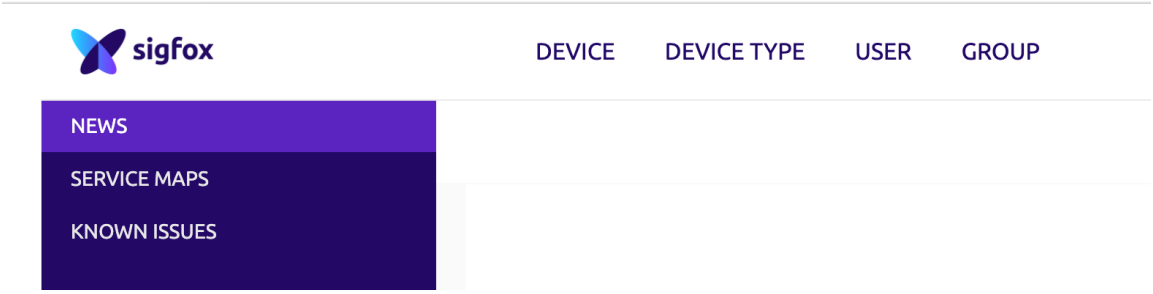


FIGURE 10.1 – Home page

```
11
# print Sigfox Device ID
13 print("Sigfox ID:", binascii.hexlify(sigfox.id()))

15 # print Sigfox PAC number
print("PAC Number:", binascii.hexlify(sigfox.pac()))
17
s = socket.socket(socket.AF_SIGFOX, socket.SOCK_RAW)
19 s.send("Hi! Sigfox")
```

This program imports the `Sigfox` object from the `network` module (line 1) and creates an instance of `sigfox` (line 10). It is important to specify the right region of use because frequency bands can differ from one continent to another. In addition to transmitting illegally, the Sigfox network will not receive the messages.

Lines 13 and 16 display the Sigfox identifiers of your LoPy. Note them, they will be used to register the object on the Sigfox network.

Line 18 creates a functionsocketsocket, like what was done with UDP in the previous chapter. So we can use the same primitives in Sigfox as in UDP. The line 19 allows to send a message that you can customize within the limit of 12 characters ; maximum size of Sigfox frames.

10.2 Object registration

Now that you have the precious Sigfox identifiers, connect with a browser to the site `https://backend.sigfox.com/activate`. The process is very simple. You just have to fill the fields of the different forms :

— indicate your country ;
— fill in the form with the object identifier and the **PAC** that you obtained with the program `sigfox_id.py` ;
— indicate for statistical purposes what brings you here ;
— create your Sigfox account.

This will lead to register the object and create an account on the Sigfox backend.

10.3 Visualization of the data issued by the Pycom

Go to the Sigfox website (`https://backend.sigfox.com/`) and identify yourself with the account you just created.

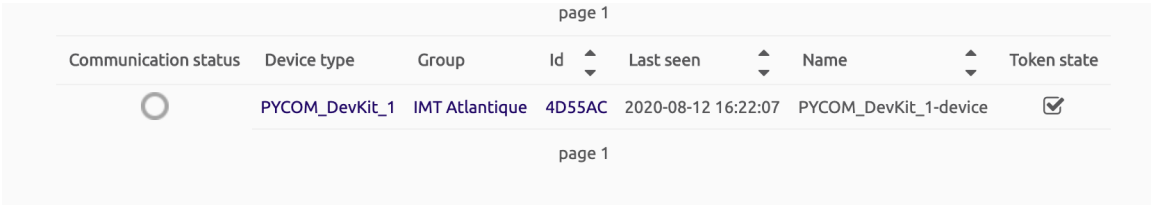


FIGURE 10.2 – Registered objects

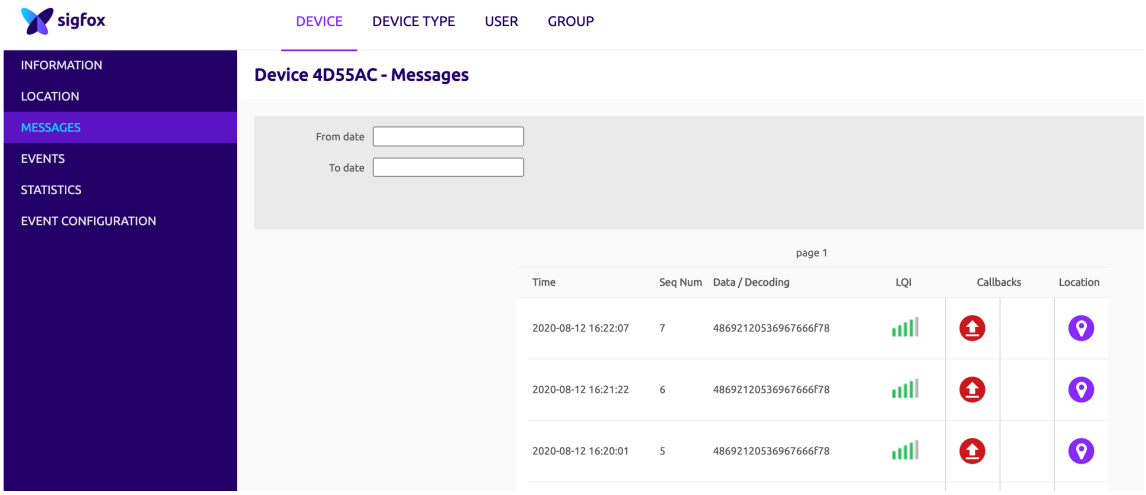


FIGURE 10.3 – List of received messages

The tabs at the top of the page (see figure vreffig-sigfox-home) will allow you to navigate through different types of information. In this part, we will only use the tab *DEVICE*. It gives access to the registered objects belonging to you (see figure 10.2). It contains :

— a name created by Sigfox according to the type of object ;
— the owner ;
— the ID you gave when you registered ;
— the date of the last message received by Sigfox for this object.

You must click :

— on the object name to configure it ;
— on the group name to access object administration settings ;
— on the object ID to get information about the received messages then *MESSAGES* in the left menu, to get the list of messages received by Sigfox, as shown in figure 10.3 [1]

10.4 What happened on the radio side

The figure 10.4 on the next page shows on a **spectrum analyzer**, the emission of 4 frames of data, including one in progress, by an object. The small vertical lines correspond to an emission. These lines are very fine ; the bandwidth used is very small, hence the term Ultra Narrow-Band (UNB). This

1. The sequence `48 69 21 20 53 69 67 66 6f 78` corresponding to the string `Hi! Sigfox`.

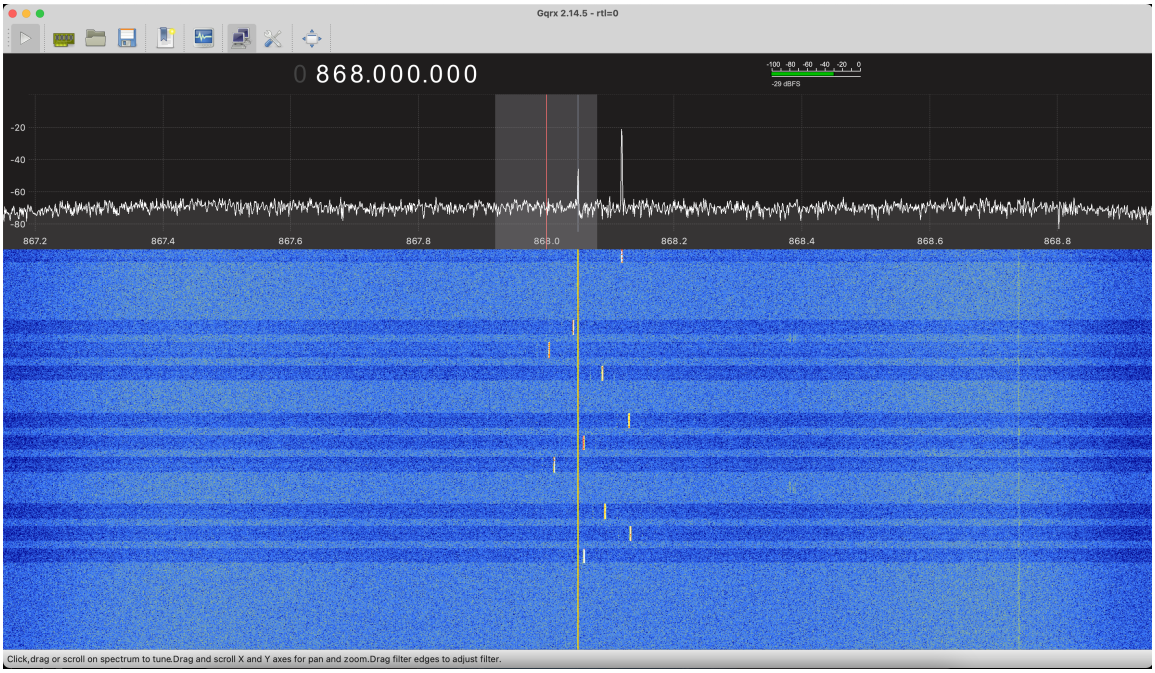


FIGURE 10.4 – Radio emissions related to the sending of Sigfox frames.

limits the risk of **collision** which would make the data incomprehensible linked to the simultaneous emission of another equipment on the same frequency.

In fact, the same message is transmitted 3 times on different and random frequencies, increasing its chances of being received.

10.5 Data retrieval

Ideally, we would like to have direct access to this data in order to manipulate it in a program. To do this, we can use the REST API developed by Sigfox. In the tab *DEVICE* (figure 10.2 on the preceding page), we have to click this time on :

— the name of your group ;
— from the left-hand menu, select API ACCESS ;
— at the top right, on the very small button *New*.

A page similar to figure reffig-sigfox-api is displayed. Give a name to this access and choose, in the menu *Profiles* the choice *DEVICE MESSAGE [R]* to have the right to read the messages. Then, of course, click on Ok.

Youtube

You see a new page with two fields in hexadecimal : *login* and *password*, that you must, as for the Beebotte API, write down somewhere or learn by heart for the following.

The best is to fill a configuration file with these values as shown in the program `config_sigfox.py`.

Listing 10.2 – config_sigfox.py

```
1 API_USER = "603f575525643207b6322e9b"
  API_PASSWORD = "93fa105063c2d0aee2bfdbadccfd2460"
```

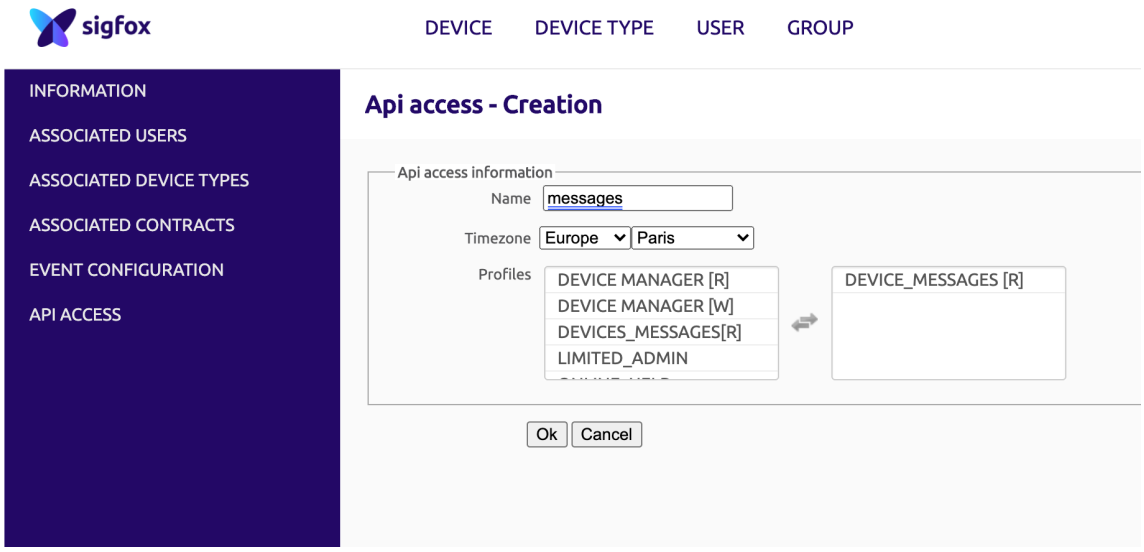


FIGURE 10.5 – REST API configuration.

```
3 DEVICEID =  "1B28CF4"
```

All the elements are now in place to transmit a temperature reading using the Sigfox network.

10.5.1 On the server

We have the access keys to the API, now we just need to write a small Python script.

For security reasons, we invite you to make a habit of putting sensitive information in a separate file.

The program `device_messages.py` allows to list the messages received by Sigfox for a particular object on your computer.

Listing 10.3 – device_messages.py

```
1  import requests
   import os
3  from requests.auth import HTTPBasicAuth
   import pprint
5  import json
   from config_sigfox import API_USER, API_PASSWORD, DEVICEID
7  import binascii

9  url = 'https://backend.sigfox.com/api/v2/devices/'+DEVICEID+'/messages'

11 print(url)

13 r = requests.get(url, auth=HTTPBasicAuth(API_USER, API_PASSWORD))
   print (r.status_code)
15
   if r.status_code != 200:
17     exit
```

```
19 resp = json.loads(r.text)
pprint.pprint(resp)
21 for v in resp["data"]:
    print ("{:10d}: {:2d} {:25} {:20} received {}".format(
23         v["time"], v["seqNumber"],
         v["data"], "["+str(binascii.unhexlify(v["data"]))+"]",
25         len(v["rinfos"])
         ))
```

Le programme :

- Line 1, import the module `requests` to be able to send http requests to a server.
- Line 3, the module `HTTPBasicAuth` is used to identify itself in a simple way using a login and a password.
- Line 6 this login and password are extracted from the file filled in the previous chapter when the REST API was created.
- Line 9, the URL with the object ID is built and
- line 13 the HTTP request is sent with the authentication method based on the password. the variable `r` is a structure containing several information.
- Lines 14 to 17, If the returned code is 200, everything went well and `r.text` contains the answer.
-
- Line 20, the answer is displayed and then some elements are given. All the messages that have been sent by the LoPy are shown.

```
>python3 device_messages.py
https://backend.sigfox.com/api/v2/devices/1B28CF4/messages
200
{'data': [{'country': 'FRA',
           'data': '486921205369676666f78',
           'device': {'id': '1B28CF4'},
           'lqi': 3,
           'nbFrames': 3,
           'operator': 'SIGFOX_France',
           'rinfos': [],
           'rolloverCounter': 0,
           'satInfos': [],
           'seqNumber': 13,
           'time': 1640279367000},
          {'country': 'FRA',
           'data': '486921205369676666f78',
  ...
           'rolloverCounter': 0,
           'satInfos': [],
           'seqNumber': 11,
           'time': 1640279155000}],
 'paging': {}}
1640279367000: 13 486921205369676666f78  [b'Hi! Sigfox']  received 0
1640279338000: 12 486921205369676666f78  [b'Hi! Sigfox']  received 0
1640279155000: 11 486921205369676666f78   [b'Hi! Sigfox']  received 0
```

The end of the trace displays a more readable summary of the information received :

— `time` gives the reception time coded according to the format **Epoch**, mentioned during the communication with Beebotte in the previous chapter;
— `seqNumber` contains the frame number and is reset to 0 when the object is reflashed. It allows to detect data loss if the numbers are not contiguous;
— "data" contains the data encoded in a hexadecimal string. The program uses the function `unhexifily` to convert it back into a sequence of bytes that can be displayed if they are ACSII characters;
— `rinfos` gives the information about the different radio gateways of the operator that received the message.

10.5.2 On the LoPy

The program `sigfox_temperature.py` is an adaptation of `wifi_temperature.py`, listing 9.6 on page 112 dedicated to Wi-Fi for transmission on the Sigfox network.

Listing 10.4 – sigfox_temperature.py

```
import BME280
import time
import socket
import kpn_senml.cbor_encoder as cbor
from machine import I2C
from network import Sigfox
import binascii
import socket

# initalise Sigfox for RCZ1 (You may need a different RCZ Region)
sigfox = Sigfox(mode=Sigfox.SIGFOX, rcz=Sigfox.RCZ1)
s = socket.socket(socket.AF_SIGFOX, socket.SOCK_RAW)

FRAME_MAX = 12
t_history = []

i2c = I2C(0, I2C.MASTER, baudrate=400000)
print (i2c.scan())
bme = BME280.BME280(i2c=i2c)

while True:

    t = int(bme.read_temperature()*100)

    # No more room to store value, send it.
    if len(t_history) == 0:
        t_history = [t]
    else:
        t_history.append(t-prev)

    print (t_history, len(cbor.dumps(t_history)))


    if len(cbor.dumps(t_history)) > FRAME_MAX:
        # oops too big for Sigfox
        t_history = t_history[:-1] # remove last item
        s.send (cbor.dumps(t_history))
        t_history = [t]
```

```
39
    prev = t
41
    time.sleep(10)
```

— line 5, the `Sigfox` class of the `network` module is set up ;
— line 11, an object `texttt` is instantiated with, here, the parameters for Europe ;
— line 12, instead of using `AF_INET` to use the TCP/IP protocol stack, the value `AF_SIGFOX` is used.
— line 14, the size of the frame is fixed at 12 bytes to be compatible with the network.

The program runs on the LoPy.

```
>>> Running sigfox_temperature.py

>>>
>>>
[118]
[2192] 4
[2192, -89] 6
[2192, -89, -16] 7
[2192, -89, -16, -12] 8
[2192, -89, -16, -12, -15] 9
[2192, -89, -16, -12, -15, -23] 10
[2192, -89, -16, -12, -15, -23, -11] 11
[2192, -89, -16, -12, -15, -23, -11, -14] 12
[2192, -89, -16, -12, -15, -23, -11, -14, -13] 13
[1999, -12] 5
[1999, -12, -5] 6
[1999, -12, -5, -8] 7
[1999, -12, -5, -8, -9] 8
[1999, -12, -5, -8, -9, -6] 9
[1999, -12, -5, -8, -9, -6, 2] 10
[1999, -12, -5, -8, -9, -6, 2, 0] 11
[1999, -12, -5, -8, -9, -6, 2, 0, -5] 12
[1999, -12, -5, -8, -9, -6, 2, 0, -5, -2] 13
[1954, -4] 5
[1954, -4, -1] 6
```

The program `device_messages.py` also retrieves these values from the computer.

```
1640281923000: 15 891907cf2b24272825020024  [b"\x89\x19\x07\xcf+$'(%\x02\x00$"] received 0
1640281823000: 14 8819089038582f2b2e362a2d  [b'\x88\x19\x08\x908X/+.6*-'] received 0
1640279367000: 13 48692120536967666f78      [b'Hi! Sigfox']      received 0
1640279338000: 12 48692120536967666f78      [b'Hi! Sigfox']      received 0
1640279155000: 11 48692120536967666f78      [b'Hi! Sigfox']      received 0
```

Be careful, depending on the temperature variations, the CBOR message grows more or less quickly. In the previous case, there was an emission every 90 seconds, but the subscription to the Sigfox network limits the number of emission to 140 messages per day. At the end of 3 hours the quota of messages will be exhausted. This small period of emission allows to test more quickly the solutions, but it is advisable to increase the period for a regular use.

10.5.3 GET request from the server

On the computer side, the simplest strategy to implement consists in extending the program `device_message.py` seen previously and to periodically query the Sigfox *backend* to see if new data arrived. This gives the following program `display_sigfox.py` :

Listing 10.5 – display_sigfox.py

```
import requests
2 import os
from requests.auth import HTTPBasicAuth
4 import pprint
import json
6 from config_sigfox import API_USER, API_PASSWORD, DEVICEID
import binascii
```

```
8  import time
   import datetime
10 import cbor2 as cbor
   import beebotte
12 import config_bbt

14 bbt = beebotte.BBT(config_bbt.API_KEY, config_bbt.SECRET_KEY)
```

The imports include the modules to send data to Beebotte and to query Sigfox. Line 14, the communication with the Beebotte server is established using the secrets of the module `config_btt`.

```
   url = 'https://backend.sigfox.com/api/v2/devices/'+DEVICEID+'/messages'
42 last_epoch = 0

44 # get the last message to find the starting epoch
   parameters = {'limit': 1 }
46 r = requests.get(url, auth=HTTPBasicAuth(API_USER, API_PASSWORD),
                     params=parameters)
48
   print (r.status_code)
50 if r.status_code != 200:
       exit
52
   j = json.loads(r.text)
54 last_epoch = j["data"][0]["time"]
```

the function `to_bbt` was not modified, one thus passes to the recovery of the data on the servers of Sigfox. The program will regularly query the Sigfox server to get the new messages. As Sigfox stores all the messages received, this can lead to a significant traffic. To limit the traffic, the program will display only the new data that arrive during its execution.

The program :

— line 41, builds the URL for the query by including the LoPy identifier ;

— line 42, initializes the variable `last`$_{epoch}$. *It contains the epoch of the last message received by Sigfox for this object ; l*

— lines 46 to 51, the request is sent with the previously defined parameter. If the status is not 200, the program stops.

— lines 53 and 54, the variable `last_epoch` receives the instant of arrival of the last message.

```
56 # delay next request to avoid 429 status error
   time.sleep(10)
```

A delay of 10 seconds is introduced between two requests, because Sigfox limits the number of requests to avoid saturation of its servers.

```
   # look periodically if new data arrived.
60 while True:
       if last_epoch > 0:
62         parameters = {"since": last_epoch+1}
       else:
64         parameters = None

66     print (parameters)

68     r = requests.get(url, auth=HTTPBasicAuth(API_USER, API_PASSWORD),
                        params=parameters)
70
```

```
    print (r.status_code)
72  if r.status_code != 200:
        break
```

The program enters an endless loop where it will regularly query the Sigfox server. The queries are identical to the previous one, but the parameter contains a JSON object with the key `since` and the epoch. If the answer to the request is not `200: OK`, the program stops.

```
    resp = json.loads(r.text)
76
    for v in resp["data"]:
78      if last_epoch < v["time"]:
            last_epoch = v["time"]
80
        print ("{:10d}: {:2d} {:25} {:20} received {}".format(
82          v["time"], v["seqNumber"],
            v["data"], "["+str(binascii.unhexlify(v["data"]))+"]",
84          len(v["rinfos"])
            ))
86
        measure = cbor.loads(binascii.unhexlify(v["data"]))
88      sending_time = v["time"]
        print (sending_time, measure)
90      to_bbt("capteurs", "temperature", measure, factor=0.01,
                    period=10, epoch=sending_time)
```

The received JSON structure is deserialized line 75 to make an array of messages enriched with parameters added by Sigfox. For each of these elements which correspond to the new received messages, the program will advance the variable last_epoch on the greatest value (lines 78 and 79), then line 87 will take the data indicated by the key `data` in the dictionary. These data, correspond to the CBOR array sent by the object, coded in a hexadecimal character string. The function `unhexilify` converts it into a binary sequence in a byte string then `loads` transforms the CBOR array into a Python array.

Line 91 and 92, the function `to_btt` (line 16 to 39 not represented in this listing) is called. It is identical to the one presented in the previous chapter. But as the procedure is asynchronous, the program treats the data when it makes the request, not when the sensor emits data, the timestamp must be that indicated by Sigfox, it is passed in the parameter `epoch`.

```
    time.sleep(60)
```

The program waits 60 seconds before making a new request.

10.5.4 POST request to the server

We had to change the logic of the program `display_sigfox.py`. Instead of waiting for data as `display_server.py` did in Wi-Fi, it will actively look for data on the *backend*. This is imposed by the limitations of the private IPv4 addressing that you probably have on your local network. Indeed, it is not possible to be reached by the outside, the connections are only made at the initiative of the equipment which has a private address. To return to the previous case where the *backend* could push data, one can either run the program on a virtual machine in the Cloud (e.g. : Virtual Private Server (VPS)

Youtube

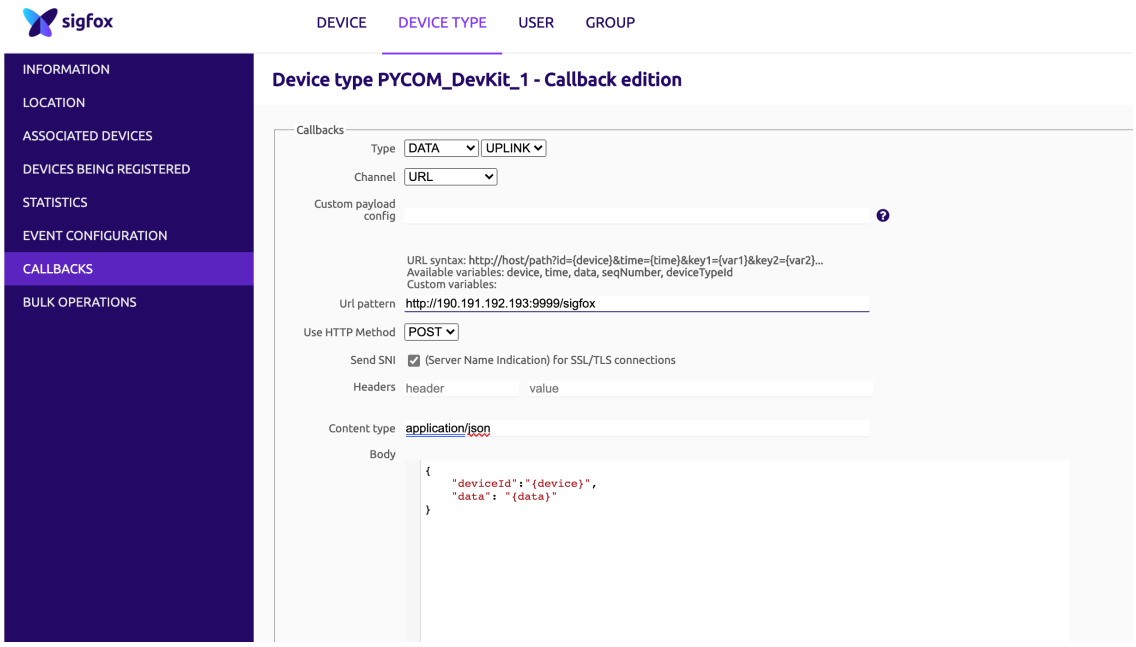


FIGURE 10.6 – *callback* configuration.

of OVH) or configure the Network Address Translation (NAT) of your Internet box to authorize incoming connections on a particular port. It is this last option that we will detail. Of course, the characteristics of the NATs change from one operator to another ; we will not be able to detail its configuration.

callback Configuration

First, go to `https://backend.sigfox.com/device/list`, click on the name of the object and then, in the left-hand menu, on *Callback*, then *New*. A page appears with the possibility of connecting the object to different platforms. We choose *Custom callback* because we want to keep our independence. The figure 10.6 shows the form.

Next, you need to determine the public IP address behind your [2]. The URI of your server will be something like shown below for the *Url Pattern*, where *aa.bbb.ccc.ddd* represents the public IP addressAttention, some operators regularly change the IP address allocated to NAT. It is therefore preferable to use a dynamic DNS if you want to use it in the long term.

```
http://aaa.bbb.ccc.ddd:9999/sigfox
```

Change the method of **GET** by an **POST**, it is still cleaner. The GET is a way to retrieve information, not to transmit it, if you want to remain compatible with REST.

It's not over yet. We now need to format the content. In Content type, replace the value by `application/json` because that's what we know best. In the window below, we will define our JSON format with two keys : *deviceId* and `data`, followed by two variables in braces that Sigfox will replace with the real values.

2. A site like `https://wtfismyip.com/` may help.

Finally, click on Ok. Now, when Sigfox receives data from the object, it will send a POST request to the indicated URI.

NAT configuration

You still need to configure the NAT on your access router so that packets addressed to port 9999 (the value chosen in the URI) TCP are sent to the private address of your computer. The process is generally the same : configure DHCP so that your computer always has the same address in the house and then configure NAT so that packets addressed to a port are sent to that computer. [3]

Wireshark can verify that requests are going through NAT and reaching your computer by looking at TCP traffic on port 9999.

POST request processing

The data contained in the POST request coming from the Sigfox *backend* will be transferred on the *Indexloopback* link of the computer to the UDP port 33033. The program `display_server.py` waiting for the data on this port will be able to be used to transform the time series coded in CBOR into JSON structure expected by Beebotte. In this way, we trivialize the program that will be able to process these three sources of information :

— object emulated by a Python program,
— LoPy with Wi-Fi,
— LoPy with Sigfox,
— and soon LoPy with LoRaWAN.

The program `generic_relay.py` does it for Sigfox and various LoRaWAN servers. We will reveal here only the lines related to the treatment of Sigfox.

Listing 10.6 – generic_relay.py

```
54 def get_from_sigfox():
56     fromGW = request.get_json(force=True)
       print ("SIGFOX␣POST␣RECEIVED")
58     pprint.pprint(fromGW)
60     downlink = None
       if "data" in fromGW:
62         payload = binascii.unhexlify(fromGW["data"])
           downlink = forward_data(payload)
64
       resp = Response(status=200)
66     print (resp)
       return resp
```

We remember the server **Flask** that we had seen in chapter 3.4 on page 46, in this program we find the same principles :

— line 54, the decorator allows to link the function `get_from_sigfox` to the URI path `/sigfox` for the POST method.
— line 57 deserializes the content of the POST which is contained in the variable `request`. The `force` parameter is there in case you forgot to put on the *backend* the format of the content to `application/json`. Everything it receives is considered as JSON.

3. If you can't do that, you can break your piggy bank to buy a public IP address for a few per month. In this case, you will have to install Python3 and the necessary modules (Beebotte, cbor2...)

— lines 62 to 64, if the POST contains data (key `data` in the JSON object, then it is sent on the link *loopback* via the function `forward_data`.
— lines 66 to 68, the POST is acknowledged with status 200 to indicate that the request has been processed successfully.

Note that the function `forward_data` can return a response that will be sent to the HTTP client. For Sigfox, this possibility is not taken into account because Sigfox authorizes only 4 downlink messages per day. We will detail its operation when we will study LoRaWAN networks. For more details on the sending of downlink messages see page **??**.

```
192 parser = argparse.ArgumentParser()
    parser.add_argument("-v", "--verbose",
194                     action="store_true",
                        help="show␣uplink␣and␣downlink␣messages")
196 parser.add_argument('--http_port', default=9999,
                        help="set␣http␣port␣for␣POST␣requests")
198 parser.add_argument('--forward_port', default=33033,
                        help="port␣to␣forward␣packets")
200 parser.add_argument('--forward_address', default='127.0.0.1',
                        help="IP␣address␣to␣forward␣packets")
202
    args = parser.parse_args()
204 verbose = args.verbose
    defPort = args.http_port
206 forward_port = args.forward_port
    forward_address = args.forward_address
208
    app.run(host="0.0.0.0", port=defPort)
```

The main part of the program analyzes the arguments used during its call. If the option `-v` is used, the program will display the relayed messages. Other options are defined to change the port numbers. These are useful if these programs are run on the same machine in the cloud.

Example

The following example traces the path of a time series from a LoPy to its sending to the Beebotte server for viewing.

```
>>> Running sigfox_temperature.py

>>>
>>>
[118]
[1895] 4
[1895, -4] 5
[1895, -4, -2] 6
[1895, -4, -2, 1] 7
[1895, -4, -2, 1, -3] 8
[1895, -4, -2, 1, -3, -1] 9
[1895, -4, -2, 1, -3, -1, 0] 10
[1895, -4, -2, 1, -3, -1, 0, 0] 11
[1895, -4, -2, 1, -3, -1, 0, 0, 6] 12
[1895, -4, -2, 1, -3, -1, 0, 0, 6, 2] 13
```

The LoPy collects the measurements and builds its time series which is sent when the capacity of the frame is reached. In the case of Sigfox, it is 12 bytes, the last line shows that the capacity is exceeded, so the last element is removed.

```
 >python3.5 generic_relay.py -v
SIGFOX POST RECEIVED
{'data': '891907672321012220000006', 'deviceId': '4D3D0E'}
--UP-> b'891907672321012220000006'
no DW
<Response 0 bytes [200 OK]>
185.110.98.2 - - [24/Dec/2021 10:14:26] "POST /sigfox HTTP/1.1" 200 -
```

The program `generic_relay.py` receives the POST request from the Sigfox network on the URI path `/Sigfox`. It contains the CBOR element sent by the LoPy. This data is sent on the port 33033. No response is received, the POST request is simply acknowledged.

```
 >python3 display_server.py
[{'data': 18.95, 'resource': 'temperature', 'ts': 1640337186000.0},
 {'data': 18.91, 'resource': 'temperature', 'ts': 1640337196000.0},
 {'data': 18.89, 'resource': 'temperature', 'ts': 1640337206000.0},
 {'data': 18.90, 'resource': 'temperature', 'ts': 1640337216000.0},
 {'data': 18.87, 'resource': 'temperature', 'ts': 1640337226000.0},
 {'data': 18.86, 'resource': 'temperature', 'ts': 1640337236000.0},
 {'data': 18.86, 'resource': 'temperature', 'ts': 1640337246000.0},
 {'data': 18.86, 'resource': 'temperature', 'ts': 1640337256000.0},
 {'data': 18.92, 'resource': 'temperature', 'ts': 1640337266000.0}]
```

The program `display_server.py` processes the CBOR data received on port 33033 and converts it into the JSON format expected by Beebotte.

10.6 Conclusion

We have built a prototype sensor that works on Sigfox. It is up to you to improve it. In particular, you have to increase the interval between two measurements which has been fixed at 10 seconds to avoid having to make too long tests. As the size of a message is 12 bytes, CBOR will take 1 byte to encode the array and the reference value is encoded on 3 bytes. If the deltas are small, they will fit on one byte. There is still room for 8 deltas. So the transmission period is 90 seconds if the intervals between two measurements remain at 10 seconds.

As Sigfox allows only 140 messages per day, the measurement time will be only 210 minutes per day, or 3 hours and a half. It is therefore preferable to take larger intervals. You can calibrate your LoPy and your reception program to be able to track the temperature over a day. Do not forget to change also this period in the program `display_server.py`.

The program `display_server.py` allows to collect information from several sources :

— a local program that emulates the measures,
— of the Wi-Fi network,
— via `generic_relay.py` from an LPWAN network.

We have our convention of representing information based on CBOR to define a time series. On the other hand, we had to modify the program `display_server.py` when we changed from a

time series representing the evolution of humidity to another one dealing with temperature and if the period changes.

We have seen the difference between GET and POST requests which induce different behaviors. If GET is more universal and can work behind a NAT, it requires regular queries to know if the resource has changed or not. POST requires the server to be accessible and thus exposes it, but the results are received "instantaneously".

The notion of client and server is vague, it cannot be attributed to a program or to an equipment, for example, the program `display_server.py` is server for Sigfox and client for Beebotte. The Sigfox site is server, if we use a GET method to get the data and client if it makes a POST.

To return to the REST paradigm, with CBOR we have defined the content of the resource but we have not named it. The receiver needs to know the format (of the CBOR containing a differentially encoded time series) and what it is. This is what we will see in the next session with the **CoAP** protocol where we will be able to define, as in HTTP, the name of the resource and its content.

11. LoRaWAN

With **Sigfox**, it was only happiness ! We had a unique environment developed by a single actor. Attaching an object to the network, retrieving data was greatly simplified.

With **LoRaWAN**, the ecosystem is more complex. As a reminder, **LoRa** is a long range modulation and LoRaWAN is a level 2 protocol which aims to federate the actors (object manufacturers, object users, network operators) around the radio communication between the object and the network core called here LNS . But the registration of objects, the link between applications and the LNS differ from one network to another.

Even if you do not intend to implement the Sigfox network, we recommend you to read the previous chapter (or at least to skim it) because we will refer to it in the following text.

LoRaWAN, like Sigfox, operates in the same unlicensed frequency band. There are several LoRaWAN network operators :

— The Things Network (TTN) [1] proposes a community approach. Each person can provide radio gateways for the community and register objects. TTN manages the core network. If this approach is nice, the coverage will be very patchy and will depend on the density of geeks in a region. Furthermore the coverage of the network will depend on the position of the antennas which must be placed on a high point. Having an antenna on your balcony does not imply great coverage of the area. If you are lucky, you may be able to get access via TTN. We will see how to connect to it.

— national operators such as, in France, **Orange** or **Bouygues Télécom**. But it is generally necessary to untie the purse to connect the objects and to limit the use of the downlink ; the sending of messages to the sensors is charged extra. On the other hand, the coverage is

1. https://www.thethingsnetwork.org/

superior.

— private networks. It is possible to run your own LNS. It implies to buy components to make a radio gateway. But there are open implementations of LNS like **chirpstack** [2].

11.1 Information about LoPy

Each object will have a unique identifier coded on 64 bits called *IndexdevEUI*. This information is necessary to enter a device on the network. The program `lorawan_devEUI.py` gives access to this value which is unique for each LoPy.

Listing 11.1 – lorawan_devEUI.py

```
from network import LoRa
import pycom
import binascii
#
lora = LoRa(mode=LoRa.LORAWAN)
mac = lora.mac()
#
print ('devEUI: ', binascii.hexlify(mac))
#
```

The Python object `LoRa` is imported from the module `network` and an instance is created line 5. The variable `mac` will store the MAC address (another name for the *devEUI*) returned by the `lora` object and display it in hexadecimal, as shown in the following example :

```
>>> Running lorawan_devEUI.py

>>>
>>>
devEUI:  b'70b3d54994c61237'
```

We have the *devEUI*. There are still 3 pieces of information missing :

— ***appEUI*** : it is an identifier on 64 bits like the *devEUI* which will be used to identify the application. It can be set to 0 ;

— ***AppKey*** : it is a value on 128 bits which is known only by the object and the LNS. It allows to derive the encryption keys used afterwards ;

And on the LNS, it will also be necessary to configure the connector : it is a question of indicating to the LNS how to send the data towards an application. It can be POST HTTP as we have seen with the configuration of the *Indexbackend* with Sigfox. Of course, once these main principles are established, it would be too simple if all networks worked the same way. So we will see how to do with the TTN network.

11.2 The Things Network

The Things Network, or TTN for short, is a community network that allows anyone to connect either a radio gateway for the benefit of the community, or an object. TTN takes care of running an LNS and sending the collected information back to its owner. We hope that a good samaritan has deployed

Youtube

2. urlhttps ://www.chirpstack.io/

an antenna near you to pick up your traffic. We have no way of knowing, if we don't try.

The first step is to create an account, free as it should be, on the TTN website : `https://www.thethingsnetwork.org/`.

Once the account creation form has been completed and validated, you will have access to the TTN network. Your login appears in the top right corner of the web page.

— Click on your name,
— choose Console,
— then your geographical area (this has nothing to do with the frequency plan. It is better to choose the nearest server to optimize communications).

You can either define an application and associate objects with it, or connect a radio gateway (*Go To Gateways*). Later on, we will explore this option if you want to install your own radio gateway. We will focus on application creation by choosing *Go To Applications*. TTN invites you to create our first application ; click on the *+Add Application* link.

Define an application

An application for TTN will integrate several LoRaWAN objects which will send their information to an application running on an AS. The form (see figure 11.1 on the facing page) must be filled in by indicating :

— the application identifier. It is a name composed only of letters and numbers and a dash. It must be unique for TTN. So be creative because it must be unique for TTN. This name will be found in the URIs allowing to drive the application ;
— the name of the application that will appear in the menus. There are no constraints ; it is better to choose something explicit ;
— a description of the application if necessary.

Click on Create Application to register this application. You will be taken to a control page for your application.

Adding an object

We need to register one or more objects in our application :

— Click on *end devices* in the left menu ;
— puis + *Add end device* ;
— and *try manual registration* below the drop-down menu. TTN offers configuration aids for some objects, but since we are experts, we don't need them.

The menu, described in figure 11.2 on page 132 appears :

— Enter the frequency plan compatible with your region. In Europe, you can choose to have the downlink in SF9 [3] or in SF12. The first one is a good compromise between the range and the transmission time ; the second one is to be chosen if your object is difficult to reach (deeply buried or far from a radio gateway) ;

3. The SF9 footnote defines the transmission speed of the information, it decreases by half at each increment doc the SF12 is 16 times slower than the SF9, but much more robust

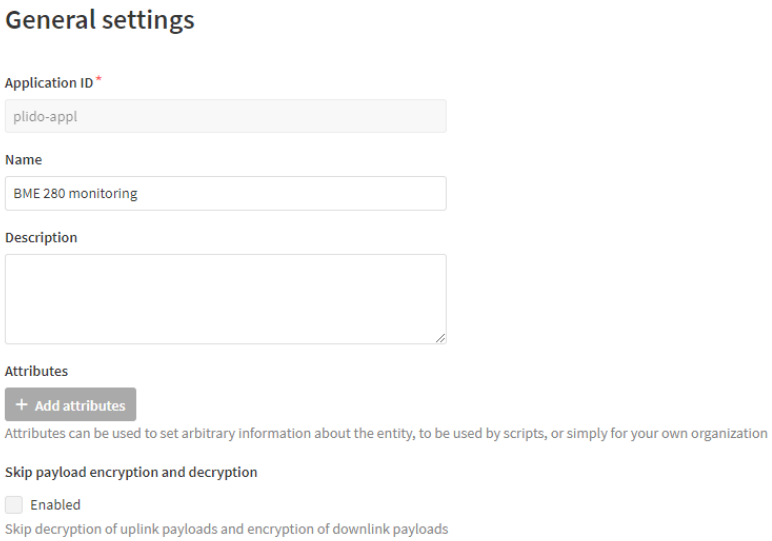


FIGURE 11.1 – Configuration of an application.

— the LoRaWAN protocol version compatible with LoPy :`MAC V1.0.2`;
— the parameters of the physical layer of LoPy : `PHY V1.0.2 REV A`;
— You must recover the *devEUI* of your LoPY by running on Atom the program `lorawan_devEUI.py` (cf. Listing 11.1 on page 129);
— set the *IndexAppEUI* to 0,
— and generate an encryption key *IndexAppKey* and do not forget to copy its value somewhere, we will need it to configure our object.

The TTN interface gives you an identifier for the object based on its *devEUI*. You can keep it, unless you want something more explicit.

The interface displays a summary page. If you forgot to do it in the previous step, copy the *AppKey* to put it in the program `lorawan_send_and_receive.py`.

Connecting the object

Listing 11.2 – lorawan_send_and_receive.py

```
from network import LoRa
import socket
import time
import pycom
import binascii

lora = LoRa(mode=LoRa.LORAWAN, region=LoRa.EU868)
#
mac = lora.mac()
print ('devEUI: ', binascii.hexlify(mac))
```

The program `lorawan_send_and_receive.py` starts, like `lorawan_devEUI.py`, by creating

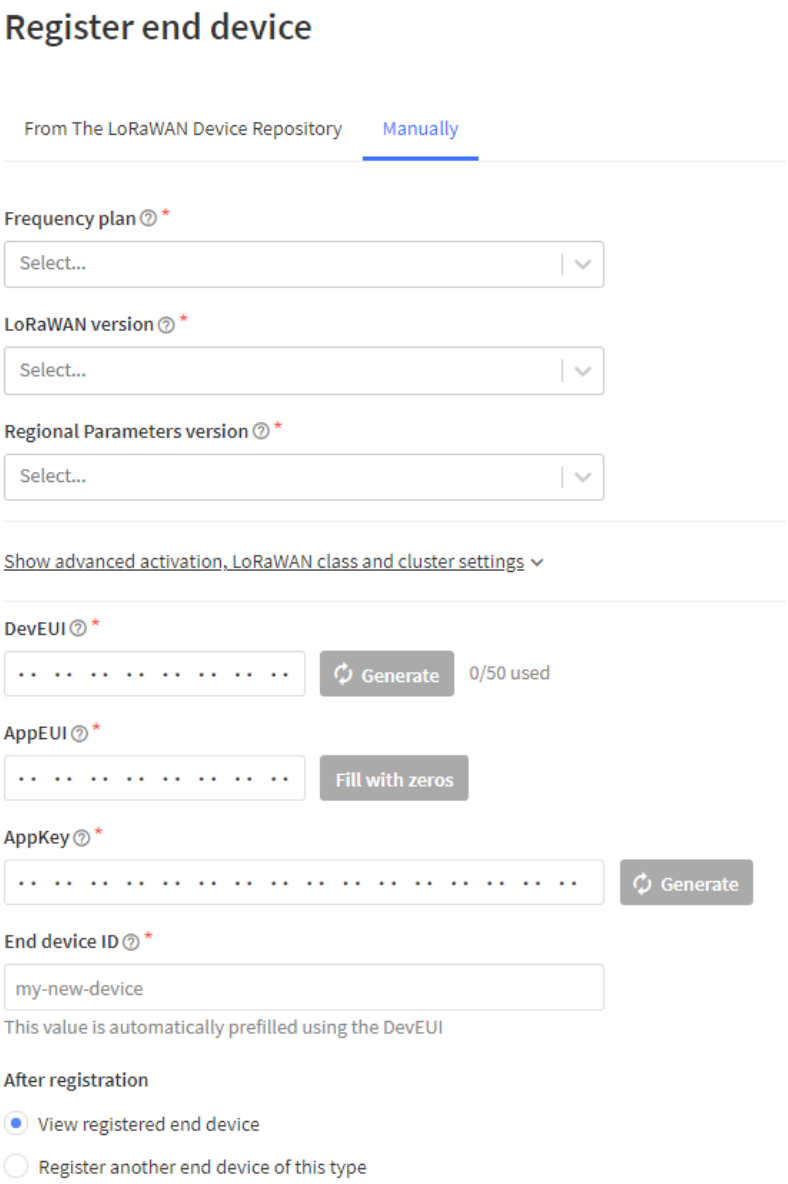


FIGURE 11.2 – Adding an object.

a `lora` object and displaying the *devEUI* of the object, it is always partique for debugging.

```
12 # create an OTAA authentication parameters
   app_eui = binascii.unhexlify(
14                     '00 00 00 00 00 00 00 00'.replace(' ','') )

16 app_key = binascii.unhexlify(
              '4EAE56D0689F6F8B02C2AFA7E08DADBA'.replace(' ','') )
```

We put in two variables, the identifiers present on the LNS of TTN, namely the *app_eui* that we had put to zero and of *app_key* that we generated randomly on the site of TTN and that we had asked you to copy somewhere.

To avoid manipulating binary sequences, these values are seen as strings (spaces can be inserted for more readability, the function `replace` allows to eliminate them). The binary sequence is obtained with the function `unhexlify`.

```
   pycom.heartbeat(False)
20 pycom.rgbled(0x101010) # white
```

We stop the periodic blue flashing of the LoPy LED by calling the pfunctionpycomheartbeat with the argument `False`. Then the LED is lit in white with the function pfunctionpycomrgbled. The argument gives the intensity of the three components Red, Green and Blue.

The LED will allow us to follow the status of the LoPy step by step.

```
22 # join a network using OTAA (Over the Air Activation)
   lora.join(activation=LoRa.OTAA, auth=(app_eui, app_key),  timeout=0)
24
   # wait until the module has joined the network
26 while not lora.has_joined():
       time.sleep(2.5)
28     print('Not yet joined...')

30 pycom.rgbled(0x000000) # black
```

The LoPy makes the connection to the network, this procedure is called *IndexJoin*. It corresponds to the emission of a message towards the LNS. If this one recognizes and accepts the object, it will return a few seconds later a message `Accept`. This phase allows the Object to obtain the encryption keys of the messages and a shorter address called *IndexdevAddr*.

To do this, the program calls the function `join` with the following parameters :

— the type of connection, very often Over The Air Authentication (OTAA) to recover dynamically the parameters [4],

— the necessary parameters, namely the *AppEUI* and the *AppKey* that we have previously configured.

— a timer (*timeout*). The value of 0 indicates that the object will try to join the network until the network accepts it.

As the function `join` is not blocking, the procedure of *join* will run in the background. The function `has_joined` allows to follow the state of the LoPy. Hence the loop (lines 26 to 28) from which we will only exit when the LoPy is accepted on the network.

4. There is also a method called Authentication By Personalisation (ABP) which consists in configuring the object with all its parameters (like for Sigfox), but it is less flexible and less adapted to the LoRaWAN environment where several operators coexist on the same frequencies.

Every 2.5 seconds a message will be displayed to say that it is still waiting for a response. These messages are absolutely not synchronized with the sending of the frames `Join` which take place every 15 seconds.

The extinction of the LED (line 30) indicates that the LoPy has joined the LoRaWAN network.

```
32 s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)
   s.setsockopt(socket.SOL_LORA, socket.SO_DR, 5)
34 s.setsockopt(socket.SOL_LORA,  socket.SO_CONFIRMED,  False)
```

The program opens the socket, the first parameter is `IndexAF_LORA` to indicate that the LoRaWAN protocol is used. The next two lines, with the functions `setsockopt`, allow to configure LoRaWAN parameters, like :

— le Data Rate (DR). Under this term, several parameters of the physical layer are gathered. The higher the DR, the faster the transmission. In counterpart, it is less robust and can carry less lion. The DR varies between 0 and 6.

— the capacity of LoRaWAN to acknowledge frames. It is not recommended to activate it because it induces emissions which are taken into account in the calculation of the duty cycle (see chapter 5.2 on page 59).

```
36 while True:
       pycom.rgbled(0x100000) # red
38     s.setblocking(True)
       s.settimeout(10)
40
       try:
42         s.send('Hello␣LoRa')
       except:
44         print ('timeout␣in␣sending')
46     pycom.rgbled(0x000010) # blue
```

We enter an endless loop to send and receive messages periodically :

— ligne 37, la LED éclaire en rouge pour indiquer une transmission.

— line 38, the calls to the sockets are made blocking, i.e. they will finish only when the action is performed by the lower layer.

— line 39, but after 10 seconds of waiting an error will be triggered.

— line 41, we recover this error thanks to the instructions `try` and `except` of Python. If the call to `send` remains blocked for more than 10 seconds, the exception processing displays `timeout in sending`. This event is rare and is caused by a local failure at layer 2 [5].

— line 46 in all cases, the LED turns blue.

```
           data = s.recv(64)
50
           print(data)
52         pycom.rgbled(0x001000) # green
       except:
54         print ('timeout␣in␣receive')
           pycom.rgbled(0x000000) # black
56
```

5. You have to be careful with this error, for example if you try to send an integer without serialization, it will lead to an error which will be taken into account in this way.

```
58      s.setblocking(False)
        time.sleep (29)
```

Finally, data is expected. In the most common mode of LoRaWAN, a device can only receive data within a short window after a transmission. If data arrives outside this window, the LNS stores the information and will transmit it when it receives a frame from the object. This saves a lot of energy because the sensor can sleep soundly, it knows that it will not miss any information.

Downlink is a scarce resource, and should generally not be overused [6], so there shouldn't be many return messages. Except here, to test the whole transmission chain :

— line 49, the call to `recv` is also blocking until data has arrived.
— lines 51 and 52, if data arrives, it is displayed and the LED turns green.
— lines 53 to 55, if no data arrive, the timer is triggered after 10 seconds generating the exception. A message warns the user of the absence of data and the LED is turned off.

If all is well, and a radio gateway is in range of your object, the program `lorawan_send_and_receive.py` will connect to the network after displaying `Not yet joined...` 3 times :

```
>>> Running lorawan_send_and_receive.py

>>>
>>>
devEUI:  b'70b3d54994c61237'
Not yet joined...
Not yet joined...
Not yet joined...
timeout in receive
timeout in receive
```

corresponding to the 5 seconds provided by the standard before sending the message of acceptance of the object.

The LED changes from white to red indicating that a message has been sent and then should go out and the message `timeout in receive` is displayed indicating that the Object has not received a message in response to its transmission.

Trace analysis

On the LNS side, it is possible to see the traffic on the page summarizing the characteristics of the object in the *live data* window (see figure 11.3 on the following page).

The events on the figure are read from bottom to top as confirmed by the timestamp, by clicking on the corresponding line, TTN displays the JSON messages used by the LNS to process the information. The data exchanged is shown :

— The event *Accept **Join-request*** indicates that the LNS received the Join message from the object, indicating that it wanted to join the network. The LNS treated it, verified that there was an object declared with the good *IndexdevEUI* and that the *IndexAppKey* used by the object to sign its request was identical to the configured value. In the following trace, we can find the ***devEUI*** of the object and the ***devAddr***, which was temporarily assigned to it. The latter will be used later.

6. TTN does not charge for downstream data, but other operators do, so be careful if you adapt these examples to other environments, because in the first tests we will use the return path.

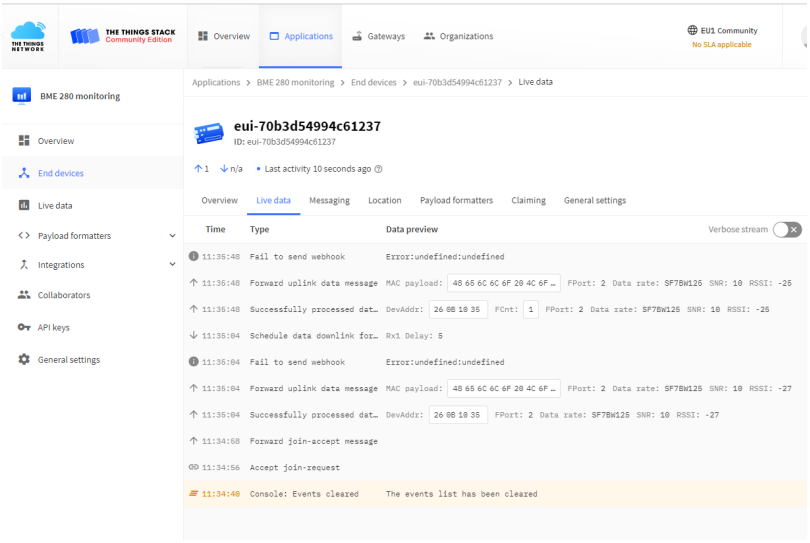


FIGURE 11.3 – Trace of an object.

```
    "device_ids": {
        "device_id": "eui-70b3d54994c61237",
        "application_ids": {
          "application_id": "plido-appl"
        },
        "dev_eui": "70B3D54994C61237",
        "join_eui": "0000000000000000",
        "dev_addr": "260B1035"
      }
```

— The event *Forward **Join-accept** message* indicates that the LNS sends an acceptance message to the Object. This message is delayed by 5 seconds during which the Object sleeps.
— The event *Successfully processed data message* shows that the LNS received correctly the data message emitted by the Object. The trace gives indications on certain physical parameters, such as the SF (here SF7 is the fastest and least robust modulation), the SNR which indicates the ratio between the power of the received signal compared to the noise and the RSSI giving the strength of the signal.
— The event *Forward uplink data message* gives more information about the message and its content. The field `frm_payload` gives the contents decrypted by the LNS but encoded in **base64**. One finds there the original message `Hello LoRa`.

```
    "uplink_message": {
      "session_key_id": "AX4VNOuMbClcPKF63sKy6A==",
      "f_port": 2,
      "frm_payload": "SGVsbG8gTG9SYQ==",
      "rx_metadata": [
        {
          "gateway_ids": {
            "gateway_id": "gateway-lo",
            "eui": "B827EBFFFE08F8A8"
          },
```

```
        "time": "2022-01-01T10:35:04.247807Z",
        "timestamp": 2661650827,
        "rssi": -27,
        "channel_rssi": -27,
        "snr": 10,
        "uplink_token": "ChgKFgoKZ2F0ZX....",
        "channel_index": 1
      }
    }
```


Question 11.2.1: Bad appKey

What trace message do you get from the LNS, if the Object is not configured with the same IndexAppKey as the LNS ?

Question 11.2.2: Autonomy

What impact on batteries ?

The indexfPort found in the LoraWAN data frames indicates the program that will process the information. The value 0 is special and indicates a configuration frame of the LoRaWAN layer, the other values up to 223 indicate specific applications. Although it is little used, it is sometimes useful to change it.

Question 11.2.3: fPort

In the previous traces, what is the value of the default fPort used by the LoPy ?

Question 11.2.4: Changing the fPort

The instruction `bind` allows to modify the **fPort** for a transmission. Modify the program to use fPort 10 and check the effect in the TTN traces.

Connector configuration

The LNS has not been configured to send the message to an Application Server (Application Server). As for **Sigfox**, a URI must be defined to tell the LNS where to post the data.

To configure a connector, go to the left menu *Integrations*. TTN offers direct integrations with existing cloud services, but it is also possible to configure your own connector :

— through **MQTT**, TTN plays the role of **textitbroker** and provides the name of the **topic** and the secret.

— by storing data locally by the choice *IndexStorage integration* and by accessing it by a GET request, as Sigfox proposes (cf. chapter 10.5.3 on page 120). That makes it possible to receive the data if one is located behind a NAT, but makes the reception asynchronous since the application must regularly question the TTN server.

— by configuring a URI on the server so that it produces a POST request when a data frame is received (choice ***Webhooks***).

We will also favour this last mode of operation for TTN, because if it requires a configuration of the LNS, it makes it possible to receive the data in a synchronous way when they are processed by the LNS.

You have to click on **Webhooks**. A certain number of services are proposed, but we will define our own following the example of what we did with Sigfox (cf. chapter 10.5.4 on page 122). To prepare the connectivity, it is necessary :

— If you are in the cloud, the address is obtained by typing the command `ifconfig`.
— If you are behind a box or your ISP's router, you have a private address. A web site like `myip.wtf` will return the public IPv4 address. You must then configure your access router to send TCP packets with port number 9999 to your computer.

This will form the next URI :

`http://aaa.bbb.ccc.ddd:9999/ttn`

where *aaa.bbb.ccc.ddd* represents the public IP address of your equipment. As this address can change over time, it is preferable to use a dynamic DNS service to obtain a more stable domain name over time.

The page listing the *IndexWebhooks* associated with the application is empty. You must click on to add a new one, it is advisable to choose *Custom webhook*, the others define the connectors to data management services (cf. figure 11.4 on the next page).

In this menu, you should indicate :

— the webhook identifier :
— the JSON format :
— the URI that we have just built :
— it is not necessary for the moment to fill the *Download API Key* which will be used to send data to the Object. We will come back to this later.
— you have to choose the events that will trigger a POST request. In our case, we will only choose the reception of an upload message (*Uplink message*). We can complete the path of the previously defined URI with elements that will allow the REST server to better identify the type of request. In our case, this is not necessary. `/ttn` therefore identifies the messages received by the Objects LNS.

Once the *webhook* validated, the program `generic_relay.py` must be launched. It will create a Web server which will wait for the POST requests of the LNS of TTN.

```
@app.route('/ttn', methods=['POST']) # API V3 current
def get_from_ttn():
    fromGW = request.get_json(force=True)

    downlink = None
    if "uplink_message" in fromGW:

        payload = base64.b64decode(fromGW["uplink_message"]["frm_payload"])
        downlink = forward_data(payload)

        if downlink != None:
            from ttn_config import TTN_Downlink_Key

            downlink_msg = {
```

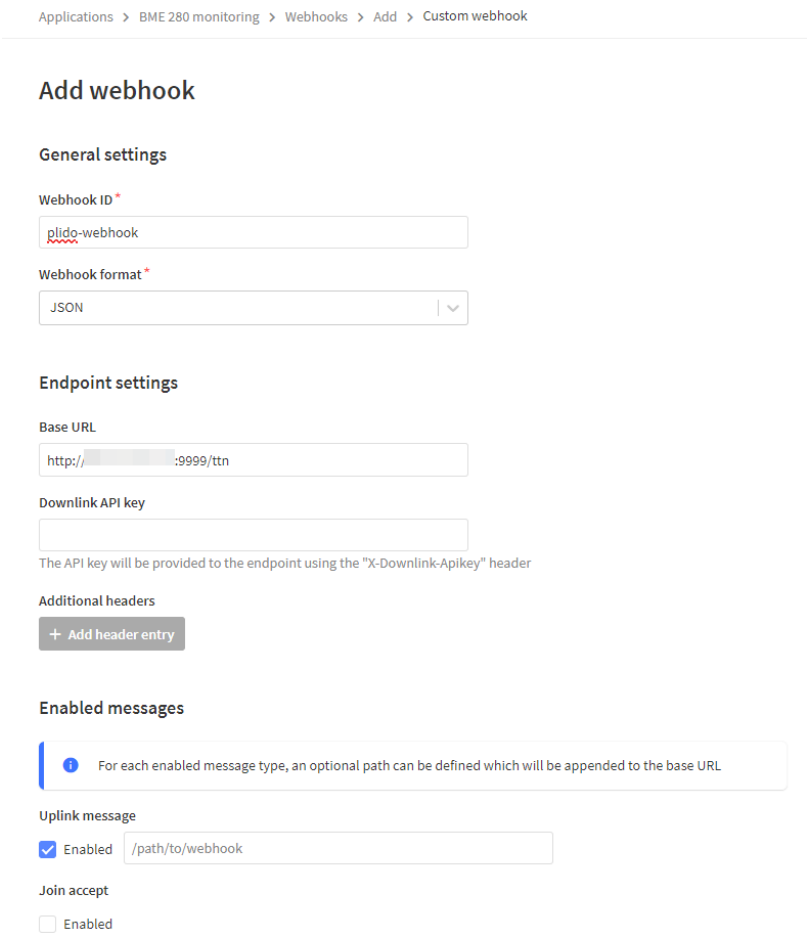


FIGURE 11.4 – Webhook configuration.

```
                "downlinks": [{
                    "f_port":   fromGW["uplink_message"]["f_port"],
                    "frm_payload": base64.b64encode(downlink).decode()
                }]}
            downlink_url = \
            "https://eu1.cloud.thethings.network/api/v3/as/applications/" + \
            fromGW["end_device_ids"]["application_ids"]["application_id"] + \
                            "/devices/" + \
                            fromGW["end_device_ids"]["device_id"] + \
                            "/down/push"

            headers = {
                'Content-Type': 'application/json',
                'Authorization' : 'Bearer ' + TTN_Downlink_Key
            }

            x = requests.post(downlink_url,
                              data = json.dumps(downlink_msg),
                              headers=headers)

    resp = Response(status=200)
    return resp
```

This program uses the module **Flask** to create a Web server. It associates the path `/ttn` with the function `get_from_ttn` for requests of type POST (lines 93 and 94). When a request of this type arrives, the variable `fromGW` variable contains the JSON object [7] (line 95). If it contains the key `uplink_message`, the data are extracted (line 100) to be sent through the interface ***loopback*** thanks to the function `forward_data` to a program which processes them. If this function returns the value `None`, the program does not wish to answer the Object, the Web server acknowledges positively (code 200) the request coming from the LNS (line 130 and 131).

```
def forward_data(payload):
    global verbose, forward_port, forward_address

    inputs = [sock]
    outputs = []

    if verbose:
        print ("--UP->", binascii.hexlify(payload))
    sock.sendto(payload, (forward_address, forward_port))

    readable, writable, exceptional = select.select(inputs, outputs, inputs, 0.1)

    if readable == []:
        if verbose:
            print ("no DW")
        return None

    for s in readable:
            replyStr = s.recv(1000)
            if verbose:
                print ("<-DW--", binascii.hexlify(replyStr))
            return replyStr
```

7. converted implicitly into a Python dictionary.

```
50      return None
```

If the `verbose` mode is activated, the message to be sent to the data processing program is displayed in hexadecimal thanks to the function `hexlify` (lines 33 and 34). Then the message is actually sent (line 35) [8]

To wait for a response that is not systematic, we cannot use the `recvfrom` call because it is blocking. Instead, we use the `select` call of the module of the same name. It allows to wait on several events at the same time coming from different sockets (line 37) where :

- `inputs` is an array of sockets that may have received data. In our case (line 30) it is initialized with the socket talking to the program processing the data ;
- `outputs` is an array which contains the sockets in which it would be possible to write. In our case, this array is empty (line 31).
- the third one concerns the exceptions. In our case, we repeat the table of sockets `inputs`
- finally the fourth parameter indicates the waiting time in the function `select`. In our case, it is set to 0.1 second. This time is both compatible with a response from the data processing program, and the response that must be sent to the LNS so that the response is associated with the upstream message.

`select` returns in a tuple of three elements, the socket(s) that triggered its return. In our case, these are :

- either the expiration of the timer, in this case, the variable `readable` which contains the list of sockets having received data is empty (line 39) and the value `None` is returned.
- or data which arrived in one or more sockets. The variable `readable` contains the array of these sockets. The variable `s` contains the first socket of the array, the data are read and sent back.

The program `generic_relay.py`, launched with the option `-v` shows in a synthetic way the traffic.

```
>python3 generic_relay.py -v
--UP-> b'48656c6c6f204c6f5261'
no DW
63.34.43.96 - - [01/Jan/2022 18:27:45] "POST /ttn HTTP/1.1" 200 -
--UP-> b'48656c6c6f204c6f5261'
no DW
34.255.49.188 - - [01/Jan/2022 18:28:31] "POST /ttn HTTP/1.1" 200 -
```

The content of the rising message is displayed in hexadecimal and corresponds to the famous content `Hello LoRa`. As there is no answer, `no DW` is then displayed.

Data processing

The function `forward_data` of the program `generic_relay.py` sends the data to the address of `127.0.0.1` [9].

The program `display_receive_and_send.py` processes the data of the Object.

Listing 11.3 – display_receive_and_send.py

```
1 import socket
  import binascii
```

8. the address of the recipient and the port are configured during the analysis of the parameters.
9. These two values can be respectively changed by using the arguments `--forward_address` and `--forward_por`.

```
3
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
5 s.bind(('0.0.0.0', 33033))

7 while True:
    data, addr = s.recvfrom(1500)
9     print (data)
    s.sendto("Pleased to meet you!".encode(), addr)
```

It waits for the data on port 33033. The function `recvfom` returns the data stored in the variable `data` and the address of the one who sent the data stored in the variable `addr`. The data is displayed on line 9 and the message `Pleased to meet you` is returned to the sender, on line 10 thanks to the function `sendto`.

By launching the program in a new window, the LoPy message is correctly received.

```
>python3 display_receive_and_send.py
b'Hello LoRa'
```

On the other hand, the program `generic_relay.py` produces an error.

```
--UP-> b'48656c6c6f204c6f5261'
<-DW-- b'506c656173656420746f206d65657420796f7521'
63.34.43.96 - - [01/Jan/2022 19:06:51] "POST /ttn HTTP/1.1" 500 -
Traceback (most recent call last):
...
```

The return message was received by the function `forward_data`, but the downstream path has not yet been configured. The error is linked to the absence of the file `ttn_config.py`, line 104, which forces Flask to return the status 500 and to display the functions which led to the error.

Return path

There is no security problem when the LNS sends data received from sensors to the application server, because the place where it transmits them has been configured by their owner. However, in the other direction, the LNS must ensure that the data to be transmitted to the Object comes from an authorized application server.

As for Beebotte, a secret key will be used. To this key, the LNS can associate rights to more or less limit access. In the left menu, click on *API key* then on + *Add API Key*. In the next screen, give your key a name and choose *Grant Individual Right* to limit the use of the key. Select *Write downlink application traffic* and validate.

Copy the key and confirm that you have done so by validating *I have copied the key*. Edit the file `ttn_config.py` by inserting the key obtained.

Listing 11.4 – ttn_config.py

```
1 TTN_Downlink_Key = "ENTER YOUR KEY"
```

Relaunch the program `generic_relay.py`, the LoPy displays the answer.

The following listing shows the construction of the POST request to the LNS to transport the descending message :

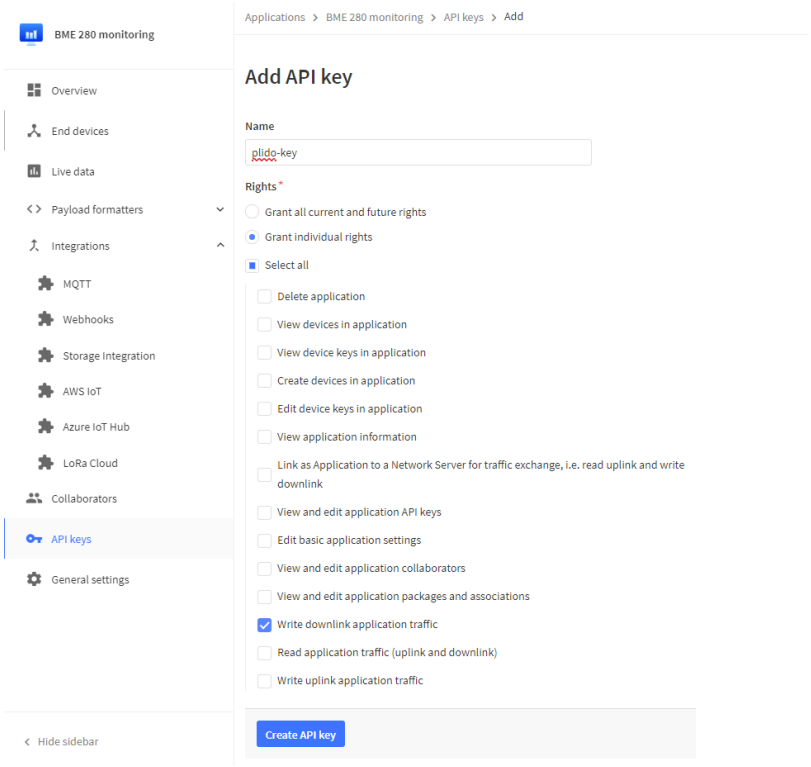


FIGURE 11.5 – Creating a key

```
        if downlink != None:
104         from ttn_config import TTN_Downlink_Key

106         downlink_msg = {
                "downlinks": [{
108                 "f_port":   fromGW["uplink_message"]["f_port"],
                    "frm_payload": base64.b64encode(downlink).decode()
110             }]}
            downlink_url = \
112         "https://eu1.cloud.thethings.network/api/v3/as/applications/" + \
            fromGW["end_device_ids"]["application_ids"]["application_id"] + \
114                     "/devices/" + \
                        fromGW["end_device_ids"]["device_id"] + \
116                     "/down/push"

118         headers = {
                'Content-Type': 'application/json',
120             'Authorization' : 'Bearer ' + TTN_Downlink_Key
            }
122
            x = requests.post(downlink_url,
124                           data = json.dumps(downlink_msg),
                              headers=headers)
```

— line 104, the key authorizing the sending of downlinked messages is copied into the variable `TTN_Downlink_Key`.

— lines 106 to 110, the JSON format expected by the LNS to send a downlink message is built, several messages can be sent at the same time, hence the use of an array. Each of the elements contains a `fPort` which is copied from the upstream message and the data coming from the socket (variable `downlink` encoded in base64.

```
{'downlinks': [{'frm_payload': 'UGxlYXNlZCBObyBtZWV0IH1vdSE=', 'f_port': 10}]}
```

— lines 111 to 116, the URI to access the service is constructed. It contains two variable elements which identify the application and the Object. These are also extracted from the upstream message.

```
https://eu1.cloud.thethings.network/api/v3/as/applications/plido-appl/devices/eui-70b3d54994c61237/down/push
```

— lines 118 to 121, the options that will be added to the HTTP header are indicated. They contain the authentication key.

— finally, line 123 to 125, these different elements are passed to the function `post` of the module `requests`.

11.3 Adding a radio gateway

You want to install a LoRaWAN radio gateway at home to benefit from a long range network. We chose to use a Pygate which will allow us to stay in the Pycom universe that we know well now and especially which is one of the cheapest solutions.

For this you need a specific card that acts as an expansion card. We connect a processor to it. The latter is not necessarily a LoPy because we will not use its LoRa radio part. Indeed, the Pygate has another Semtech radio component

Youtube

that allows the simultaneous listening of eight LoRa channels. A LoPy can only listen on one channel at a time. When sending data, the object randomly chooses a radio channel for the message to be received. The radio gateway must listen on all possible channels.

We insert the Pycom on the Pygate and connect the antenna to the Pygate connector. In a second step we need to adapt the Pycom to its new functions, by downloading the right firmware. This is done by launching the Pycom Firmware Update program and choosing Pygate.

In a third time, we open the Pygate directory with Atom. We find there the file `wifi_conf.py` which is going to allow the Gateway to have access to your wifi network, it can contain the same identifiers as when you had connected your LoPY to your wifi network. The program `main.py` is preconfigured to use the European LNS of The Things Network.

The file `config.json` contains the description of the frequencies that can be used in this area. If you take a look at it, you can see eight frequencies that are listened to by the radio gateway. You can find the other possible configurations on The Things Network website.

We download the program in the memory of the Pycom of the radio gateway and as it is called `main.py` it will launch itself at reboot. It connects to the wifi, then displays the gateway ID. Copy it because we will need it later. Then we have a lot of text indicating that the gateway is configuring the LoRaWAN network. When the LED turns green, the gateway is active.

11.3.1 The Things Network configuration

Now we need to inform The Things Networks about the existence of our radio gateway. If not you will have to create an account on The Things Network. Go to console choose *Europe 1* if you are in Europe or the corresponding geographical area. Add a gateway by giving :

— a textual identifier which is a non-readable without spaces ;
— the gateway identifier which corresponds to the hexadecimal value displayed at startup ;
— a text to better describe the gateway ;
— finally the frequencies planif the objects are easily accessible you can choose the Spreading Factor 9 for the descending path otherwise if you can't join your objects you can fall back on the Spreading Factor 12.

We validate the gateway and it is added to the LNS. You will be able to see the frames it receives.

11.4 General view of the exchanges

The figure 11.6 on the following page illustrates the overall operation. The first transmission of the LoPy is picked up by one or more radio gateways, which relay the message via a specific JSON format called *IndexPacket Forwarder*, this can be directly in UDP or using MQTT. The LNS after identifying the Object, sends a POST request containing the data to a destination configured by a

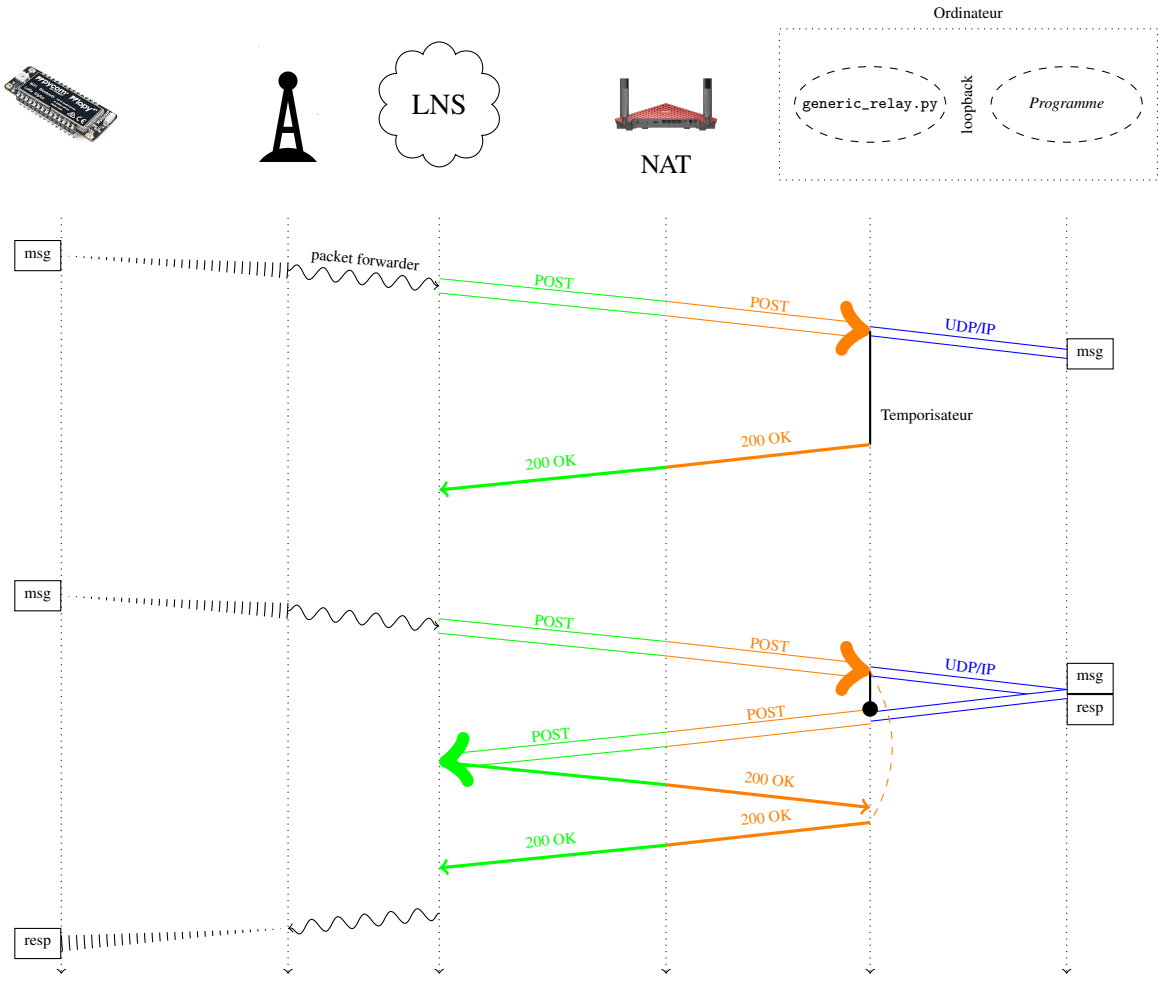


FIGURE 11.6 – Time diagram of the exchanges.

URI. This request arrives at the home router, which sends it to a device in the home depending on the NAT configuration. The program `generic_relay.py` processes the request, extracts its contents and sends it via a socket to the program. If the latter does not answer, the timer is triggered and the request is acknowledged.

Otherwise, if the program `generic_relay.py` receives data in response before the timer is triggered, it initiates a new POST request to the LNS. In our example, when the LNS acknowledges it, the initial request is in turn acknowledged.

11.5 LoRaWAN Thermometer

The program `lorawan_temperature.py` combines the way of joining the LoRaWAN network that we have seen previously and the part that sends differential time series of temperature measurements. The only difficulty lies in the size of this time series. Indeed, according to the **Data Rate** the volume of transported data differs. The table 12.2 on page 153 gives the correspondence between the *Data Rate* and the other physical parameters. It also indicates the maximum size of the transported

Data Rate	Spreading Factor	Largeur de bande	Taille Maximum (octets)
DR0	SF12	125 KHz	51
DR1	SF11	125 KHz	51
DR2	SF10	125 KHz	51
DR3	SF9	125 KHz	115
DR4	SF8	125 KHz	242
DR5	SF7	125 KHz	242
DR6	SF7	250 KHz	242

TABLE 11.1 – Data Rate en Europe

data.

This change of size can pose problems of compatibility, if one chooses a size too large to be transmitted. In theory, one knows the *Data Rate* chosen and one can adapt the size consequently, but the operator can also modify the *Data Rate* of the Object according to the conditions of transmission, and there is no instruction in micro-python to recover the *Data Rate* really used. To simplify the implementation, the maximum size of 50 bytes was chosen.

Just replace `display_receive_and_send.py` by `display_server.py` for the time series to be processed and the result sent to Beebotte.

Question 11.5.1: US902

With the help of the following link `https://www.thethingsnetwork.org/docs/lorawan/regional-parameters/` indicating the regional parameters. Would the program `lorawan_temperature.py` work on a LoRaWAN network located in North America ?

12. CoAP

The REST principles with their representation of information by resources pointed by globally unique identifiers is one of the keys to the success of the Internet and the composition of distributed services. Even if it is not the only solution, it is essential that information from constrained objects can be integrated into this global web. CoAP, for Constrained Application Protocol, allows this integration at a better cost in terms of memory or protocol footprint than the HTTP protocol would allow.

12.1 Introduction

In the previous chapters, we saw that CBOR allowed to send structured data in an efficient way, that the receiver could make the difference between an integer or a string and also know how many elements composed a dictionary or an array. In addition to saving space compared to JSON, the complexity of serializing or deserializing was limited, leading to memory-efficient implementations.

But CBOR is not enough for good interoperability. When a receiver receives the data, it must know that it is a CBOR encoding and not some other structure. In addition, the receiver must know what to do with the data. In the exercises, we transmitted time series corresponding to temperature readings. But if we also wanted to transmit humidity and pressure, how would the receiver know the difference ?

We have a solution to answer these questions : the use of resources.

We have seen that with the REST paradigm, resources are named. So, to distinguish between different time series, it is sufficient to use a different name (or URI). The resources also contain meta-information and it is therefore possible to transport the encoding format to indicate that they are CBOR, JSON, CSV...

Despite its universality, this model poses a problem for constrained objects :
HTTP uses TCP to ensure reliable communications between the client and the server. However,

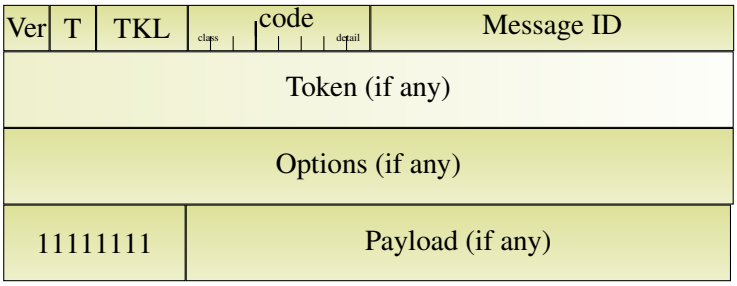


FIGURE 12.1 – Format of a CoAP message

TCP is resource-intensive. It requires memory to store unacknowledged or out-of-sequence packets, and a large number of heuristics must be implemented to improve its performance ; the flexibility to create headers can be a disadvantage for a constrained object. Thus, the request HTTP to the server `www.arduino.cc` may have this form :

```
GET / HTTP/1.1\r \n
Host: www.arduino.cc\r\n
U s e r Agent : Mozilla/5.0 (X11;Ubuntu;Linuxx8664;rv:25.0) Gecko /20100101Firefox/25.0\ r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,   /   ;q=0.8\r\n
A c c e p t Language :  e n US ,en;q=0.5\r\n
A c c e p t Encoding :gzip,deflate\r\n
Connection: keep-alive\r\n
\r \n
```

Below the first line that requests the resource from the root (usually `index.html`), there are a number of lines in the format :

```
Field name: Field value*
```

which will tell the server the name of the server you want to reach, the description of the browser and the formats it can accept.

The goal of CoAP is to define a much stricter protocol that will be easier to implement but that will be able to interoperate with HTTP in order to preserve the principles defined by the REST architecture and to take advantage of the naming of resources so that the constrained resources participate in the great global spider web.

12.2 Format of a CoAP header

The header of CoAP messages is variable in size but highly structured, as shown in Figure 12.1.

Youtube

The first 32-bit word is present in all CoAP messages :

— the field `Ver`, on 2 bits, contains the version number of the protocol, which is `01` in the current version ;

— the field `T` for ***Type***, also on 2 bits, indicates the nature of the message (`00` : **CON**firmable, `01` : **NON** confirmable, `10` : Acknowledgementindexindex, `11` : ResetindexRST) ;

— the field `IndexTKL`, on 4 bits, gives the length in bytes of the token field starting at the second word of 32 bits. If the value is 0, this field is absent. The values from 1 to 8 indicate the length. Values from 9 to 15 are not allowed ;

— the field `IndexCode`, on 1 byte, allows a rather subtle coding of the nature of the request or the answer (cf. next chapter) ;

—

12.2.1 Coding of `code`

In many application protocols such as FTP or HTTP, the server returns a 3-character code indicating whether the request was executed correctly or not. Codes starting with the number :

— 1 informs that the request is being processed normally. This type of notification is not taken into account with CoAP ;
— 2 indicates that the request has been accepted and processed correctly ;
— 3 is used to indicate an indirection ;
— 4 refers to an error on the client side, due to bad syntax or a request that cannot be processed. Thus the famous 404 error indicates that the client has requested a page that does not exist on the server ;
— 5 indicates an error on the server's side.

The web site of the [1] gives the errors found in the HTTP protocol. As previously mentioned, the left-hand number varies between 1 and 5 while the two right-hand numbers, specifying the reason for the notification, generally vary between 0 and 31.

To allow a more compact representation, CoAP will encode this string in a byte. The three bits on the left indicate the nature of the code and the 5 on the right will give the reason.

Thus the HTTP error code `415` (*Unsupported Media Type*) is noted in CoAP `4.15`, written in binary `100.01111` and in decimal 143. This notation concerns the answers to the requests but it leaves room to code the requests as well. Indeed, the code with the first three bits at 0 is not used to code the notifications.

Several requests compatible with the REST architecture can be coded :
— **GET**, coded `0x01`, retrieves the content of a resource present on the server and designated by a URI ;
— **POST**, coded `0x02`, stores a value on an existing resource present on the server ;
— **PUT**, coded `0x03`, creates a resource on the server and assigns it a value ;
— **DELETE**, coded `0x04`, deletes a resource on the server.
Note that the value 0x00 can be used in some cases.

12.2.2 Use of the field `Message ID`

The field `IndexMessage ID` on 2 bytes is used to identify the CoAP messages in order to detect the duplicates. This value is recopied in the acknowledgements to make it possible to know which message is acknowledged. They must not be reused during a fixed period.

Youtube

The CoAP protocol is based on the UDP layer for reasons of simplicity of implementation. It may sometimes be necessary to make data transfer more reliable. To do this, CoAP has a sub-layer implementing a very simple protocol. Each message contains a text field and three types of frames are available :

1. `http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml#http-status-codes-1`

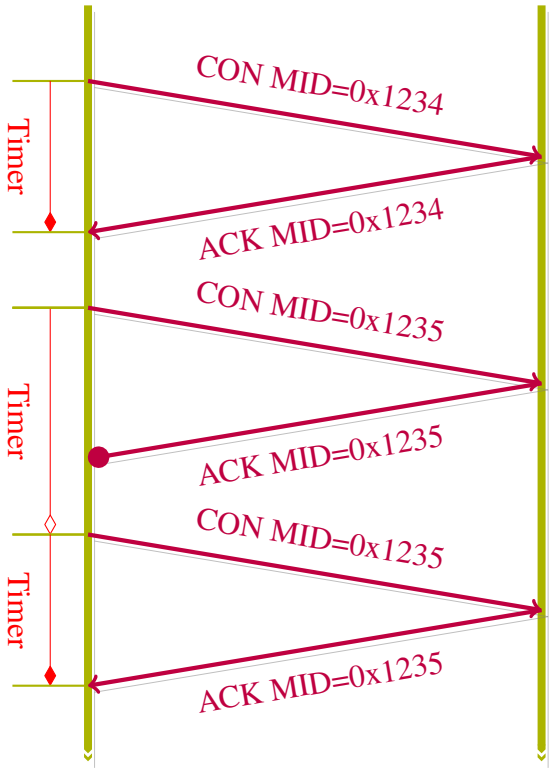


FIGURE 12.2 – Échanges fiabilisés avec CoAP

— messages of type `CON` for *confirmable* indicate that they must be acknowledged by the receiver.
— messages of type `ACK` contain this acknowledgement, the field `message ID` of the message to be acknowledged is copied in this message. This message can also contain data.
— messages of type `NON` for *non confirmable* are pure datagrams, they will not be acknowledged by the receiver, their loss will not be detected by the CoAP protocol. On the other hand, the field `message ID` makes it possible to detect duplicated messages.

The notion of sender/receiver is dissociated from the roles of client or server defined by REST. A REST client can be a sender of **CON**, **NON** or **ACK** frames, as can a server.

A message of type `RST` is added to the three preceding types, it can be emitted for example when one of the nodes lost its context following a restart and does not know how to treat the answers which it receives.

Figure 12.2 shows reliable exchanges with the CoAP protocol. Messages of type `CON` imply an acknowledgement in return. The sender arms a timer and at its expiration re-transmits the message. If it receives an acknowledgement message containing the same value in the `Message ID` field, the message is acknowledged. This case is illustrated with the `Message ID` being worth `0x1234`.

If on the other hand, at the expiration of the timer, the acknowledgement has not arrived, the

Paramètre	Valeur par défaut
ACK_TIMEOUT	2s
ACK_RANDOM_FACTOR	1.5
MAX_RETRANSMIT	4
MAX_LATENCY	100s
PROCESSING_DELAY	2s

TABLE 12.1 – Default values proposed by the RFC 7252

initial message, kept in memory, is retransmitted. The RFC 7252 suggests an initial timer of 2 seconds whose value doubles at each retransmission, and 4 transmissions of the same message are possible. These values can be changed to fit the context. The timer durations include a randomness, to avoid synchronization between several transmitters that could lead to frame collisions.

The value of the field `Message ID` only has meaning for an exchange, if for example a receiver receives the values `0x1234` and `0x1236`, it should not deduce that the message `0x1235` has been lost. Several transmissions can also take place in parallel ; a sender does not have to wait for an acknowledgement before sending the next frame.

Unconfirmed messages (type `IndexNON`) also use a different `message ID` field for each new message. It allows to detect duplications which could occur in the lower protocol layers during the transport of the message.

To reject the duplicates due to duplications of the lower layers, to the losses of the acknowledgment messages forcing a re-transmission (case of the ID messages `0x1235` of the figurer reffig-CoAP-CON) or to badly dimensioned timers triggering a re-transmission before the reception of the acknowledgment), the receiver must keep a copy of the ID messages emitted by a source. A simple calculation allows to define the retention period of the values.

A little elementary algebra allows to calculate this duration. The figure 12.3 on the facing page shows the worst case calculation. It is obtained when all CON messages are lost and retransmitted and only the last one is acknowledged. As the timer duration is doubled at each attempt, the time spent in this step is

$$(1+2+4+...) \times ACK\_TIMEOUT$$

or

$$2^{MAX\_RETRANSMIT} - 1 \times ACK\_TIMEOUT$$

To avoid synchronization between the nodes, the value of the timer is multiplied by a factor *ACK_RANDOM_FACTOR* between 1 and 1.5. As we are in the worst case, we take the maximum value.

We deduce the maximum waiting value before a correct transmission *MAX_TRANSMIT_SPAN* which is 45 seconds.

Once the message is transmitted, it must arrive at its destination. The RFC 7252 takes a very important value of 100s for the latency between the object and the serverfootnoteThe figure vreffig-

Paramètre	Valeurs déduites
MAX_TRANSMIT_SPAN	45s
MAX_RTT	202s
EXCHANGE_LIFETIME	247s
NON_LIFETIME	145s

TABLE 12.2 – Values deduced from the default parameters proposed by the RFC 7252

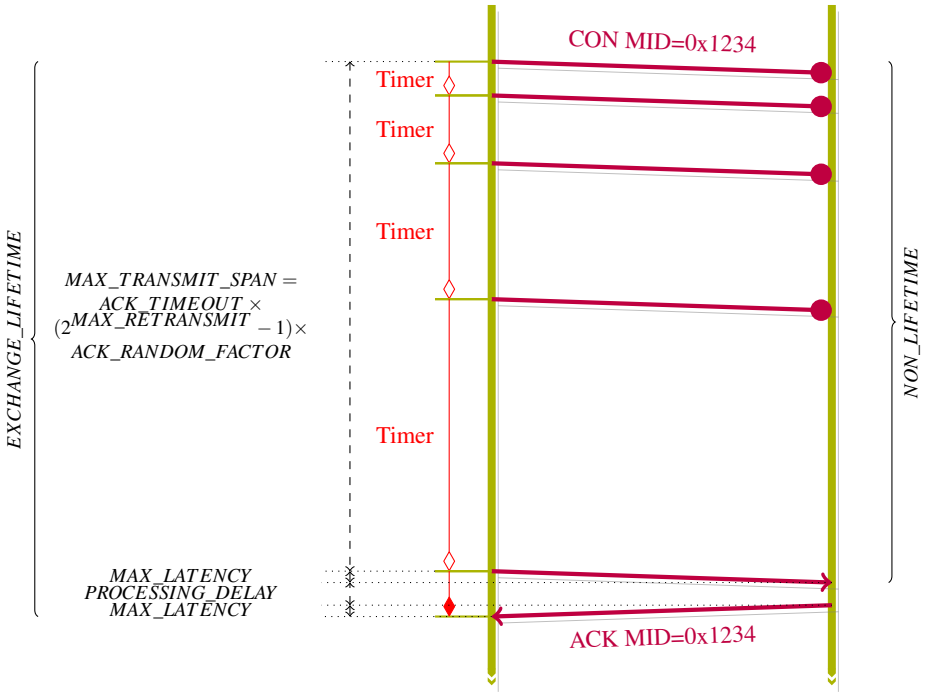


FIGURE 12.3 – Reliable exchanges with CoAP

duree-max is not to scale and 2s for the processing time. The maximum round trip time or RTTT is 202s.

For confirmed **CON** messages, the maximum $EXCHANGE_LIFETIME$ is therefore 245s. For the confirmed **NON** messages, we can suppose that a message will be transmitted several times to make sure that it was correctly received by the recipient. We find the same calculation without the acknowledgement part, hence a duration of 145s.

The standard is that, by default, the activity time of a Message ID is about 5 minutes (247 s) for confirmed messages, and 2.5 minutes (145 s) for unconfirmed messages. Keeping this in mind can save you hours of debugging. Let's assume that a client always starts by numbering its messages at 1. The receiver will then keep the message ID values for 5 minutes. If you restart the object, it will send new messages, but with the same Message ID values that will be acknowledged, but not processed ignored by the receiver.

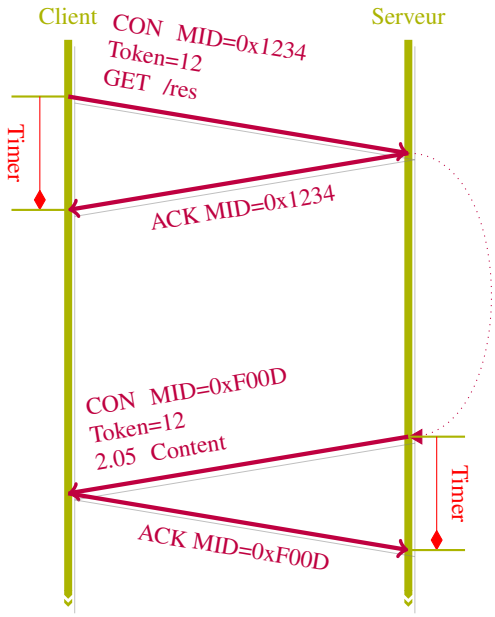


FIGURE 12.4 – Utilisation du Token

12.2.3 Token

CoAP uses the UDP protocol to communicate. Unlike TCP, there is no notion of connection establishment. It is therefore difficult to make the link between establishments and responses, especially if they are not immediate. The following figure illustrates this phenomenon. A GET request is sent by a client to a server.

Youtube

The response cannot be immediate (for example, a value must be read from a sensor that requires activation). The acknowledgement message cannot be delayed, otherwise the client, not seeing its request acknowledged, would retransmit it. The server acknowledges with an empty Ack message (see figure 12.4). When the server can send the resource, it does so in turn in a CON message, which will in turn be acknowledged. You may notice that the values of the Message ID field are completely uncorrelated. To make the link between the request and the response, a token provided by the client is copied by the server. This is why we can consider a token value as a "connection" between the client and the server.

The Token is an optional binary sequence whose size is between 0 (no token) and 8 bytes. The length is specified at the beginning of the header in the Token Length field (`TKL`) and the value immediately follows the mandatory header before the options.

We can see on this example, figure reffig-CoAP-Token the decorrelation between the low level protocol machine based on the `Message ID` and the REST protocol machine. The client sends a CON message containing its request and receives the response in an ACK message. In this example, there are also two levels of acknowledgement at the message level and with REST notifications.

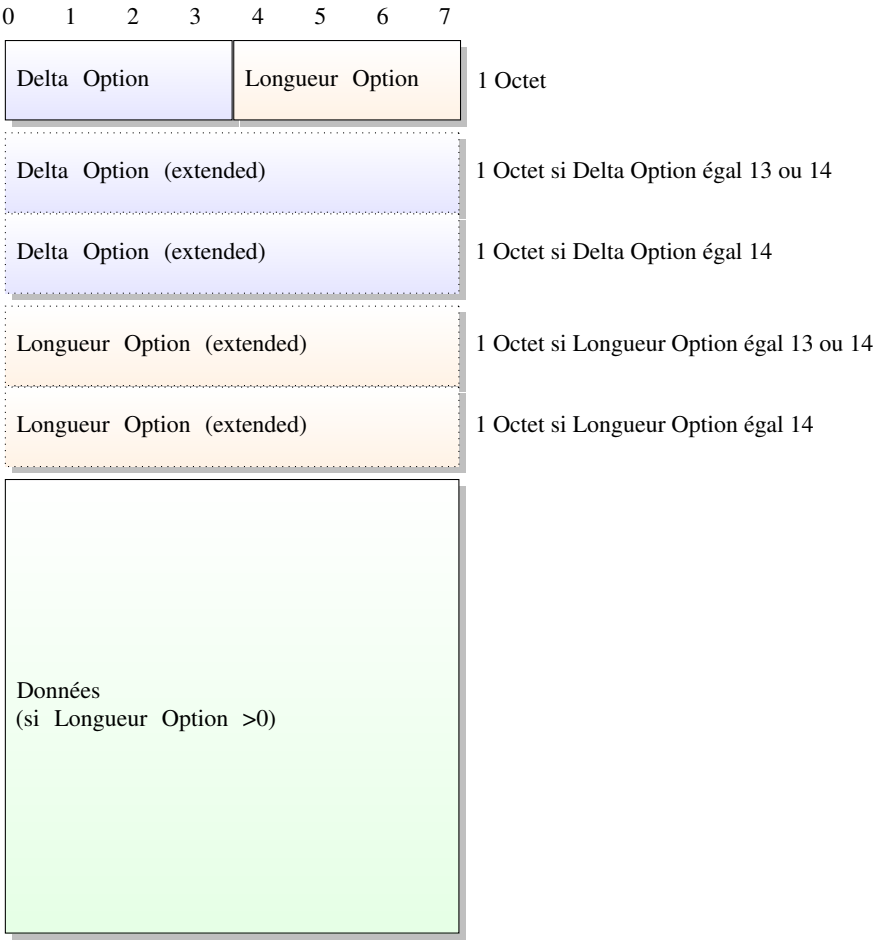


FIGURE 12.5 – Format des options

12.2.4 The CoAP options

The field `Option` will contain **option**s that will either be used to improve the data transfer protocol between the client and the server, or will be used to encode the request and response headers by guaranteeing a certain compatibility with HTTP headers.

The structure used (see figure 12.5) is called Type Length Value (TLV) or Type Length Value. Each field contains at least these two pieces of information :

— Type indicating the nature of the option ;
— Length indicating the size of the data in bytes. If the latter is not null, the data will be after.

CoAP complicates things a little by opting for a differential coding of the option value. Thus, if one must send an option of type 5 then two of type 6, the coding will contain $\Delta T = 5$, $\Delta T = 1$, $\Delta T = 0$.

But as the ΔT field is only 4 bits long, we can't go very far to encode these values. An escape mechanism is implemented for differences greater than 13. In this case, the value 13 is put in the T field and the next byte encodes the difference minus 13.

PFor example, if you have to code two options with values of 5 and 20, the difference is 15. The

first option is coded normally with the ΔT at 5. For the second option, the ΔT is set to 13 and the next byte will take the value 2.

Note that the value 14 put in the ΔT field indicates that the difference requires two bytes to be encoded. The main options used in CoAP are listed in the table 12.3 on the next page. Those with a blue background will be discussed in more detail in this book.

For the length one finds the same principle : the lengths lower than 13 are coded directly ; if they are higher or equal to 13, the value minus 13 is coded in an additional byte. A value of 14 indicates that two bytes are used to code the length minus 269.

Figure 12.5 on the preceding page illustrates the coding of an option in the header of a CoAP message.

It is possible that there is data after the options. In this case, a separator with the value `0xFF` is inserted. It cannot be confused with the encoding of an option since the ΔT and Length fields only change between 0 and 14.

If there is no data to be transmitted (for example in the case of a GET request), the CoAP message ends after the options.

12.2.5 CoAP options

The first bit used to encode the type describes, when set to 1, whether this type must be known by the receiver (critical). In this case, if a receiver receives an option of this type and does not know it, it must produce an error message. Otherwise, this option is ignored by the receiver who continues to process the following options. Thus, even-numbered options are optional and odd-numbered options are critical.

12.2.6 URI representation

The meaning of some of the options given in the previous table is better understood when the syntax of a URI is known (see RFC 2396). We have already seen that the general syntax is :

Youtube

```
<scheme>:<scheme-specific-part>
```

where *scheme* will define the notation scheme. In general, the schema will indicate how the rest of the URI is structured. When the URI is also a locator (thus a URL), the schema refers to the protocol that can be used to find the resource, such as http or https or even coap. After the schema, there are two zones, the authority that will indicate who is responsible for naming the resources. In the case of a URL, it can be written as follows :

```
<scheme>://userinfo@host@:port@/path?query
```

where the italicized fields *userinfo@* and *:port* are optional. They contain respectively the name of the user and the port number on which the service runs.

URI will be composed of a series of segments separated by characters that identify the resource on the server. The URI can end with questions, i.e. a string of characters that will be interpreted by the previously designated server. A question, i.e. parameters provided to build the resource, can contain several parts separated by the character .

Value	Name	Type	Nature	repeate	Comment
0	reserved				
1	**If-Match**	opaque	critical	yes	Used to indicate to the server to make the request only under certain conditions.
3	**Uri-Host**	string	critical		Contains the server name of a URI (name, IPv4 or IPv6 address). Generally, it is not necessary to specify this since CoAP messages are sent to this address.
4	**ETag**	opaque	optional	yes	Used to manage resource caching
5	**If-None-Match**	empty	critical		Used to tell a server to make the request only under certain conditions.
6	**Observe**	integer	optional		Allows a server to send a request to the state changes of a resource. In the response the value must always increase.
7	**Uri-Port**	integer	critical		Contains the number of the UDP port on which CoAP is launched. Usually this field is not needed as the CoAP server is already expecting messages on this port.
8	**Location-Path**	string	optional	yes	Used in response to a POST request to specify a segment of the resource path.
11	**Uri-Path**	string	critical	yes	Contains one of the segments of the URI
12	**Content-Format**	integer	optional		Defines the format in which data is encoded
14	**Max-Age**	integer	optional		The length of time the resource can be cached.
15	**Uri-Query**	string	critical	yes	
17	**Accept**	integer	critical		Indicates the formats that the client can accept.
20	**Location-Query**	string	optional	yes	Used in response to a POST request to indicate the path to the resource.
35	**Proxy-Uri**	string	critical		Contains a URI that must be taken into account by the proxy.
39	**Proxy-Scheme**	string	critical		Indicates the encoding scheme.
60	**Size1**	integer	optional		Indicates the size of the resource.
258	**No-Response**	integer	optional		Limite les notifications REST

TABLE 12.3 – Some options of the CoAP protocol

This can be verified by the disassembly program. THE URI :

```
coap://192.168.1.52/capteur1/temperature?max value&date=20131206
```

uses the coap naming scheme. The server is `192.168.1.52`, the path (*path*) is composed of two segments, followed by two questions. The server receives the following request :

```
Received packet of size 50
40 01 BE BF|B8 63 61 70 74  - 65 75 72 31|0B 74 65 @....capt eur1.te
6D 70 65 72 61 74 75 72 65  -|49 6D 61 78 5F 76 61 mperature Imax_va
6C 75 65|0D 00 64 61 74 65  - 3D 32 30 31 33 31 32 lue..date =201312
30 36                                              06
ver:1 Type = 0 (CON) Token Length = 0 code 1 (GET) Msg id = BEBF
```

```
Option = 11 (+11) length = 8
Uri-Path capteur1
Option = 11 (+0) length = 11
Uri-Path temperature
Option = 15 (+4) length = 9
Uri-Query max_value
Option = 15 (+0) length = 13
Uri-Query date=20131206
```

The previous listing shows what the server receives. The URI is not complete, because the part that was used to locate it is not necessary. Only the parts "path" and "question" are indicated. The coap : scheme is not specified ; neither is Uri-Host and Uri-Port because the server knows its IP address and the port number on which the CoAP server is running.

They could be useful in case of server virtualization, i.e. if several CoAP instances were running, either under different names or on different port numbers. While this possibility exists, for the time being the low capacity of the resources does not push towards virtualization.

If we go back to the optional part, the first option that starts with the byte `0xB8`. `0xb` (=11) indicates that it is an option **Uri-path** (as it is the first option, the delta is confused with the value of the option) and of length 8 bytes which correspond to the value `sensor1`. The second option begins with `0x0B`. The delta is null. One remains on a Uri-path option of length 11 bytes. The third option opens with the byte `0x49`. The increment being of 4, the number of the option passes to 15 that is **Uri-query** with a value on 9 bytes. The next option starts with `0x0D`. One remains on a Uri-query option but the length `0xD` informs that the following byte contains the length. The value is strangely worth `0x00` because it is decreased by 13 to respect the coding defined by CoAP. the length is thus 13 bytes.

Questions on URIs

Let the following CoAP message be :

```
40020001b474656d700573656e7331436d6178ff32332e30
```

Question 12.2.1: Code ?

What does this message represent ?

- ○ A GET request
- ○ A POST request
- ○ A PUT request
- ○ A DELETE request
- ○ A positive notification

Question 12.2.2: Token or not Token ?

What is the value of the token field ?
- Empty
- `0xb4`
- `0xb474`
- `0xb47465`
- `0xb474656d`

Question 12.2.3: URI path

What elements of the URI does this message contain ?
- Aucun
- `/temp`
- `/temp/sens1` et `/max`
- `/temp/sens1/max`
- `/temp/sens1?max`

In order for the client to interpret the data, they must be able to understand how it is represented. This may depend on the font. For example, an accented letter will not be represented in the same way depending on the type of code. The same goes for the representation. The simplest way is to send the requested value in ASCII, for example the string `18` indicates 18 degrees. It is therefore necessary to indicate the type of encoding/serialization used to describe the content of the resource. Where HTTP would use a name, i.e. a character string, CoAP will use a numerical value.

The previous table gives an extract of the values used to represent the formatsfootnoteThe complete list can be found on the IANA site `https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#content-formats`..

Two CoAP options use these codes :
— **Content-format** (12) indicating how the resource is encoded ;
— **Accept** (17) indicating in a request the format in which the response should be encoded.

There are many values for the content-format, they allow to specify in a very economical way the type of resource and consequently the processing to be done.

Each protocol using CoAP will tend to define new codes. The table 12.4 on the next page illustrates this phenomenon for **SenML**. The value will be used to indicate both the data structure and the encoding format.

Question 12.2.4: ASCII

You have the following resource :
`temperature = 20C`
What value should the CoAP Content-type option use for a CoAP response ?
- text/plain
- 0
- 50

Valeur	Type
0	text/plain ; charset=utf-8
40	application/link-format
41	application/xml
42	application/octet-stream
47	application/exi
50	application/json
60	application/cbor
110	application/senml+json
112	application/senml+cbor
11542	application/vnd.oma.lwm2m+tlv
11543	application/vnd.oma.lwm2m+json

TABLE 12.4 – Type des donnŽes

Question 12.2.5: SenML

You want to receive a resource in the SenML ; CBOR format. What value should the Accept option in the request carry ?

Question 12.2.6: Erreur

What error code does the server return if it cannot send a response in this format ? Help you with the RFC 7252.

- 4.04 (Not Found)
- 4.02 (Bad Option)
- 4.06 (Not Acceptable)
- 5.01 (Not Implemented))

12.3 Observe

With the REST architecture, the server always responds to a client's request. If we want to follow the evolution of a resource, the client must periodically ask for the value ; the server does not keep any status on past requests. This is not always compatible with the energy constraints of sensors. Suppose we have a fire alarm that must inform when the smoke level reaches a certain threshold. There are two possibilities :

— the alarm is a REST client and sends a POST to a server when the alert is triggered. For this to work, the alarm must be configured with the address of the server to which to send its POST requests ;

— the alarm is a REST server that has a resource giving the smoke rate. It does not need to be configured. Clients query it and it responds to their address. However, if you want to determine when a threshold is reached, you have to continuously query the resource at a high frequency if you don't want to miss any information.

The **Observe** option, defined in the RFC 7641, allows a server to periodically send the value of a

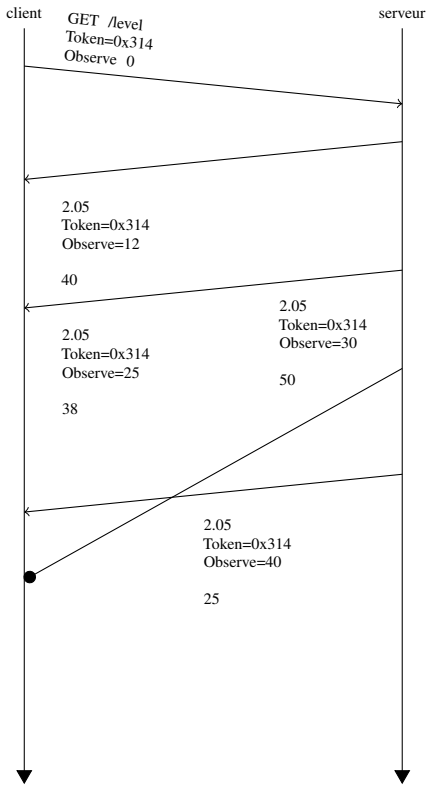


FIGURE 12.6 – Periodic sending with the option `Observe`

resource to the client that requested it. The sending period (or the sending rules such as send when a threshold is reached) is defined by the behavior of the server.

Figure 12.6 illustrates this behavior. The client sends a GET request with the Observe option set to 0, but in value. If the client accepts this option, it will respond by setting it in its responses. In this case it must have a value that can only increase from response to response. This is useful to allow the client to detect a desequencing of the responses. Thus in the example, the Observe stamped 30 arrives after the one stamped 40 and will be rejected by the client.

Note also that the **Token** field must be present to make the link between the request and the answers.

It must also be possible to stop an Observe. There are several cases :

— lhe client wants to stop an Observe in progress. He redoes the same request but sets the Observe option to 1.

— the client restarts, it may lose its context about the Observe, but continue to receive periodic requests from the server. The client not recognizing the Token, sends a message **ReSeT**. The server cancels the periodic transmission to the client.

— the client is unreachable, it will not be able to cancel the transmission. As a rule, responses with the Observe option are carried in confirmable **CON** messages. The server may occasionally send the response in a confirmable **CON** message. If it does not receive an acknowledgement

from the client, it deduces that it has disappeared and stops sending periodic responses.

13. Let's experiment CoAP

The programs are located in the directory `pycom` for the Object and `plido-tp4` for the server.

The programs for the Object can run either in a window on your computer or on your Pycom. In the following, we will assume that it is a second terminal window on your computer but feel free to run it on your Pycom in Wi-Fi (or even in LoRaWAN) for more realism. The use of the Sigfox network is more problematic since the requests from your Pycom are limited to 12 characters and the answers are only 4 per day and limited to 8 characters.

To concentrate on the functioning of CoAP, we will first experiment locally on our computer. We will then see how to use the program `generic_relay.py` to benefit from LPWAN networks.

Be careful, the server does not work directly under Windows. You must run it in the virtual machine.

13.1 Client/server implementation

We will implement the CoAP protocol between two processes on your machine and then, if you want, you can test it on a LoPy. On the Object side, we will use a simple but compact implementation of the protocol to understand how it works. At the other end, we will use the **aiocoap** implementation which is very complete but much more complex and resource intensive.

Youtube

13.1.1 aiocoap

As its name indicates, aiocoap implements CoAP, with asynchronous Input/Output modules, allowing a high degree of parallelism. The following program (`coap_basic_server1.py`) gives an example of a simple server managing a single time resource :

Listing 13.1 – coap_basic_server1.py

```
1 #!/usr/bin/env python3
```

```
# This file is part of the Python aiocoap library project.
#
# Copyright (c) 2012-2014 Maciej Wasilak <http://sixpinetrees.blogspot.com/>,
#               2013-2014 Christian Ams ss <c.amsuess@energyharvesting.at>
#
# aiocoap is free software, this file is published under the MIT license as
# described in the accompanying LICENSE file.

"""This is a usage example of aiocoap that demonstrates how to implement a
simple server. See the "Usage Examples" section in the aiocoap documentation
for some more information."""

import datetime
import logging
import socket

import asyncio

import aiocoap.resource as resource
import aiocoap

class TimeResource(resource.Resource):

    async def render_get(self, request):
        await asyncio.sleep(5)

        payload = datetime.datetime.now().\
                strftime("%Y-%m-%d %H:%M").encode('ascii')
        return aiocoap.Message(payload=payload)

# logging setup

logging.basicConfig(level=logging.INFO)
logging.getLogger("coap-server").setLevel(logging.DEBUG)

def main():
    # Resource tree creation
    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
        s.connect(("8.8.8.8", 80)) # connect outside to get local IP address
        ip_addr = s.getsockname()[0]

    port = 5683
    print ("server running on ", ip_addr, "at port", port)

    root = resource.Site()

    root.add_resource(['time'], TimeResource())

    asyncio.Task(
        aiocoap.Context.create_server_context(root,
                                    bind=(ip_addr, port)))

    asyncio.get_event_loop().run_forever()

if __name__ == "__main__":
    main()
```

— Lines 21 and 22, the use of *aicoap* results in the import of the modules which are in the `aoicoap` directory.
— In the function `main` the program :
  — looks for its fixed IP address (lines 40 to 42) ;
  — sets the port number to the default value assigned to the CoAP server (line 44) [1] ;
  — line 47, the variable `root` contains the tree of resources. On the next line, the resource `time` is associated with a class `TimeResource`.
  — The method :

```
asyncio.Task(aiocoap.Context.create_server_context(root, bind=(ip_addr, port)))
```

  allows to link this resource tree to the IP address and port number previously defined.
  — Line 55, the server is then launched in an endless loop.

It is more interesting to see the processing done when the resource is called by the server. The class `TimeResource` derived from the generic class `aiocoap` `Resource` is used (line 24) :

```
class TimeResource(resource.Resource):
```

For each CoAP method, a method can be defined in this class. In the example, the method `render_get` allows to treat the GET requests.

To simulate a processing time, the program starts by waiting for 5 seconds (line 27) and then constructs the string containing the date that it will return in a `aoicoap` object `Message`.

Thus, if everything goes well, the answer to a request (Code = 0x01) will be 2.05 (Content). The figure 13.1 on the next page summarizes this exchange that we had seen theoretically in the chapter 12.2.3 on page 154 devoted to tokens.

13.1.2 Object side

On the client side, we will use a more compact implementation that will allow us to experiment the operation of CoAP by changing the values of the protocol fields.

Listing 13.2 – coap_empty_msg.py

```
1  import CoAP
   import socket
3
   SERVER = "192.168.1.XX" # change to your server's IP address
5  PORT   = 5683

7  s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

9  coap = CoAP.Message()
   coap.new_header()
11 coap.dump(hexa=True)

13 s.settimeout(10)
   s.sendto (coap.to_byte(), (SERVER, 5683))
15
   resp,addr = s.recvfrom(2000)
```

1. The use of an IP address and not the wildcard 0.0.0.0 allows the server to run in a Windows and MAC environment

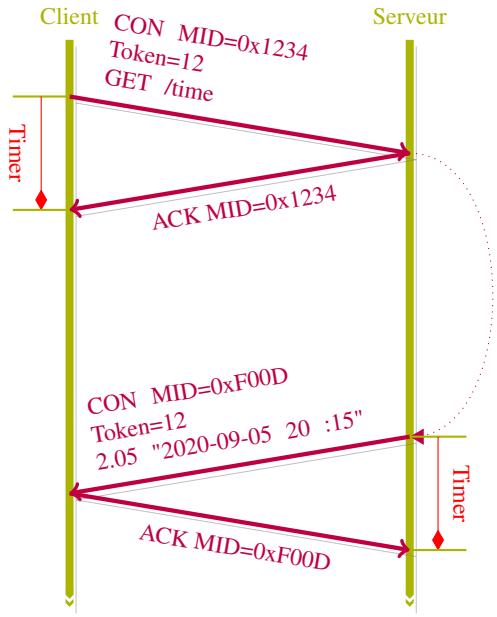


FIGURE 13.1 – Date retrieval

```
17 answer = CoAP.Message(resp)
   answer.dump(hexa=True)
```

The program `coap_empty_msg.py` is much simpler. First, you must replace the IP address with the one provided by your CoAP server. The program creates a UDP socket through which the exchange with the server will be made. The first action consists in creating a message CoAPCoAP (line 9) and on the next line create a mandatory header with the default parameters with the function `CoAP`.

The program displays the message with the function CoAPdump(line 11) and sends it to the socket. Line 13 allows to limit the waiting time of the answer to 10 seconds. This answer is expected on line 16, transformed into a CoAP message on line 17 and displayed.

Run a Wireshark capture to see the traffic passing through the CoAP port (`udp.port==5683` in the filter window).

Now start the client program.

```
b      40000001
CON  0  x0001  EMPTY
b      70000001
RST  0  x0001  EMPTY
```

The following messages were circulated on the network.

```
18:38:30.381449 IP 192.168.1.26.50883 > 192.168.1.26.5683: UDP, length 4
        0x0000:  4500 0020 221f 0000 4011 0000 c0a8 011a  E..."...@.......
        0x0010:  c0a8 011a c6c3 1633 000c 83a2 4000 0001  .......3....@...
18:38:30.382107 IP 192.168.1.26.5683 > 192.168.1.26.50883: UDP, length 4
        0x0000:  4500 0020 efbb 0000 4011 0000 c0a8 011a  E.......@.......
        0x0010:  c0a8 011a 1633 c6c3 000c 83a2 7000 0001  .....3......p...
```

One finds in the contents of the UDP messages, the CoAP message given by the application. If we take the second message, it starts with 0x70; which corresponds in binary to 0b01_11_0000, that is to say version = 1, type = 3 and length of the token = 0. The following byte gives the code 0 (**Empty**) and the last two bytes contain the ID message.

The server doesn't know what to do with the request, so it rejects it by sending a ReSeT message to try to stop the code on the client that sends this kind of request.

13.2 GET /time

We let the CoAP server run and we are going to build the CoAP request from the client to ask for the resource /time [2].

Listing 13.3 – coap_get_time.py

```
import CoAP
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)


coap = CoAP.Message()
coap.new_header()
coap.new_header(code=CoAP.GET, mid=0xF001)
coap.add_option(CoAP.Uri_path, "time")
coap.dump()

s.settimeout(10)
s.sendto (coap.to_byte(), ("192.168.1.77", 5683))

resp,addr = s.recvfrom(2000)
answer = CoAP.Message(resp)
answer.dump()

s.settimeout(10)
resp,addr = s.recvfrom(2000)
answer = CoAP.Message(resp)
answer.dump()
```

the method new_header specifies the code, here GET and and we add the URI time element in option (line 10) thanks to the method add_option. On the client side, we obtain the following result :

```
> python3 coap_get_time1.py
False
b'40010001b474696d65'
CON  0x0001 GET
> Uri-path : b'time'
b'60000001'
ACK  0x0001 EMPTY
```

The CoAP request starts with the word 40010001, indicating a **CON**firmable message, with no Token, a GET code and a MID of 0x0001, followed by the **Uri-Path** option.

2. Don't forget to put the right IP address of the SERVER in this program and in the following ones

FIGURE 13.2 – Échange incomplet

The response is an **ACK** with the same value of *Message* ID and the code is empty (0.00).

We don't get the response to the GET, just an acknowledgement. However, the server logs and the network analysis show that the server has responded.

```
DEBUG:coap-server:Incoming message <aiocoap.Message at 0x7f3d997ace80: Type.CON GET (MID 1, empty token) remote
<UDP6EndpointAddress 192.168.1.79:52495 (locally 192.168.1.79%lo)>, 1 option(s)>
DEBUG:coap-server:New unique message received
DEBUG:coap-server:Sending empty ACK: Response took too long to prepare
DEBUG:coap-server:Sending message <aiocoap.Message at 0x7f3d99742748: Type.ACK EMPTY (MID 1, empty token) remote
<UDP6EndpointAddress 192.168.1.79:52495 (locally 192.168.1.79%lo)>>
```

The frames that have been circulating on the network are shown in figure reffig-get-time1. As we added a delay of 5 seconds before answering on the CoAP server in the method `render_get`, the server acknowledges the request and tries 5 seconds later to send a new confirmed request. But the client, having closed its socket, cannot receive it anymore and returns an ICMP error.

The solution is to wait for a second message and to decode it as shown in the following listing which gives the code of the program `coap_get_time2.py` in which you will not have forgotten to change the server IP address.

Listing 13.4 – coap_get_time2.py

```
import CoAP
import socket

SERVER = "192.168.1.XX" # change to your server's IP address
PORT   = 5683

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

coap = CoAP.Message()
coap.new_header(code=CoAP.GET)
coap.add_option (CoAP.Uri_path, "time")
coap.dump()

s.sendto (coap.to_byte(), (SERVER, PORT))

s.settimeout(10)
resp,addr = s.recvfrom(2000)
answer = CoAP.Message(resp)
answer.dump()

s.settimeout(10)
resp,addr = s.recvfrom(2000)
answer = CoAP.Message(resp)
answer.dump()
```

We can let our joy explode because we have the answer :

```
> python3 coap_get_time2.py
```

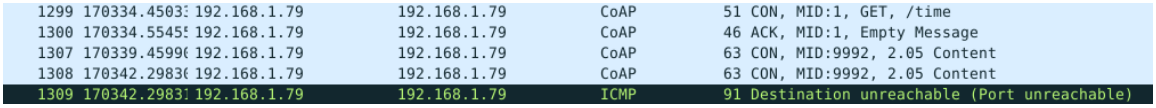


FIGURE 13.3 – Échange encore incomplet

```
False
CON  0x0001 GET
> Uri-path : b'time'
ACK  0x0001 EMPTY
CON  0x2708 2.05
---CONTENT
hex: b'323032312d30332d33302031343a3537'
txt: b'2021-03-30 14:57'
```

But our joy is short-lived because if we look more closely at the figure 13.3 the traffic on the network, we see that the response was sent twice and that we then find an ICMP error. This is due to the fact that the message from the server is not acknowledged. Believing it to be lost, it retransmits it and finds a non-existent socket.

To do well, we must provide an acknowledgement to the server with this client code with the program `coap_get_time3.py`.

Listing 13.5 – coap_get_time3.py

```
import CoAP
import socket

SERVER = "192.168.1.XX" # change to your server's IP address
PORT   = 5683

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

coap = CoAP.Message()
coap.new_header(code=CoAP.GET)
coap.add_option (CoAP.Uri_path, "time")
coap.dump()

s.sendto (coap.to_byte(), (SERVER, PORT))

s.settimeout(10)
resp,addr = s.recvfrom(2000)
answer = CoAP.Message(resp)
answer.dump()

s.settimeout(10)
resp,addr = s.recvfrom(2000)
answer = CoAP.Message(resp)
answer.dump()

mid = answer.get_mid()
ack = CoAP.Message()
ack.new_header(mid=mid, type=CoAP.ACK)
ack.dump()
```

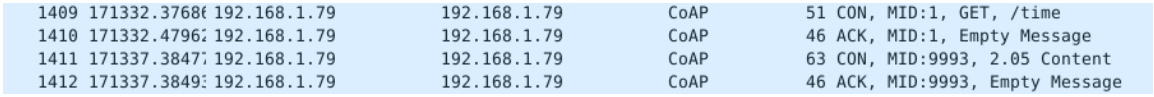


FIGURE 13.4 – State of the art exchange

```
s.sendto (ack.to_byte(), (SERVER, PORT))
```

Here, the execution is perfect :

```
> python3 coap_get_time3.py
False
CON  0x0001 GET
> Uri-path : b'time'
ACK  0x0001 EMPTY
CON  0x2709 2.05
---CONTENT
hex: b'323032312d30332d33302031353a3133'
txt: b'2021-03-30 15:13'
ACK  0x2709 EMPTY
```

if you did not forget to change the IP address of the server, and the use of the network is optimal (cf figure 13.4).

Finally, we can add a token to link the two transactions. Here, there is no ambiguity because we only request one resource. But if we were to request several resources simultaneously, we would have to be able to associate the response with the request. The program `coap_get_time4.py` adds a token field when creating the header.

Listing 13.6 – coap_get_time4.py

```
coap = CoAP.Message()
coap.new_header(code=CoAP.GET, token=0x12345)
coap.add_option (CoAP.Uri_path, "time")
coap.dump()
```

The exchange is the same but the token is repeated in the response.

```
> python3 coap_get_time4.py
False
CON  0x0001 GET   T=012345
> Uri-path : b'time'
ACK  0x0001 EMPTY
CON  0x270A 2.05  T=012345
---CONTENT
hex: b'323032312d30332d33302031353a3232'
txt: b'2021-03-30 15:22'
ACK  0x270A EMPTY
```

Question 13.2.1: NON

What happens if you use a non-confirmable request to ask for the /time resource (put the argument `type=CoAP.NON` in the mandatory header construction).

- The server crashes.
- The server responds with a Confirmable request.
- The server returns an RST because we have not programmed this case.
- The server responds with an Unconfirmable request.
- We switch to winter time.

Question 13.2.2:

Modify the server program to remove the 5 second delay before a response.
What happens when the client sends a CONfirmable request ?

- The server returns the answer in the acknowledgement.
- The server returns a NON confirmable request.
- The server still waits a few seconds to avoid saturating the network.
- The server removes the token from the response.

Question 13.2.3: Immediate response and NON

Modify the server program to remove the 5 second delay before a response.
What happens when the client sends a NON confirmable request ?

- The server returns the answer in the acknowledgement.
- The server returns a NON confirmable request.
- The server still waits a few seconds to avoid saturating the network.
- The server removes the token from the response.

13.3 POST

13.3.1 ASCII encoded resource

We will see the processing of a POST request with *aiocoap*. We could have added this processing to the `TimeResource` class seen earlier, but we preferred to add a new resource for temperature processing.

The following lines have been added to handle the POST. Note the method name `render_post`.

Listing 13.7 – coap_basic_server2.py

```
34 class TemperatureResource(resource.Resource):
       async def render_post(self, request):
36         print ("-"*20)
           print ("payload:", binascii.hexlify(request.payload))
38         resp = aiocoap.Message(code=aiocoap.CHANGED)
           return resp
```

The response to the request is code 2.04 (`aiocoap.CHANGED`) which indicates that the resource was modified.

It is associated with a new class in the resource tree :

```
    root.add_resource(['temp'], TemperatureResource())
```

The whole forms the program coap_basic$_s$*erver*2*.py*.

On the client side, the principle is the same as for the GET. In the program coap_post_temp1.py, the method add_payload allows to add content to the POST request. The program sends its request and displays the result. We are also going to simplify the management of the responses by using the function send_ack of the module CoAP. This function takes as argument a socket, a destination tuple and the CoAP message to send. It will retransmit it at most 4 times until it has received the corresponding acknowledgement.

Listing 13.8 – coap_post_temp1.py

```
import CoAP
import socket
import time

SERVER = "192.168.1.XX" # change to your server's IP address
PORT   = 5683
destination = (SERVER, PORT)

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

coap = CoAP.Message()
coap.new_header(code=CoAP.POST)
coap.add_option (CoAP.Uri_path, "temp")
coap.add_payload("23.5")
coap.dump()

answer = CoAP.send_ack(s, destination, coap)
answer.dump()
```

Figure 12.3 on page 153 showed a lossy exchange. They can be simulated by stopping the server program on your computer.

Launch the server coap_basic$_s$*erver*2*.py* and type ctrl-Z to stop its execution, while leaving the socket open.

```
> python3 ./coap_basic_server2.py
server running on  192.168.1.79 at port 5683 ^Z
Job 2,   python3   ./coap_basic_server2. p y   has stopped
>
```

Launch the client coap_port_temp1.py (don't forget to change the server IP address).

```
> python3.9 coap_post_temp1.py
False
CON  0x0001 POST
> Uri-path : b'temp'
---CONTENT
hex: b'32332e35'
txt: b'23.5'
timeout 2 1
timeout 4 2
```

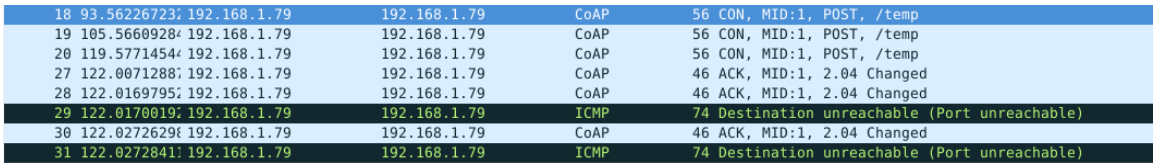


FIGURE 13.5 – POST related traffic

We see that the client does not receive the acknowledgement, triggers a timer and retransmits the message. The duration of the timer doubles with each attempt [3].

Reactivate the coap server by typing `fg` (foregroung)

```
> fg
DEBUG:coap-server:Incoming message <aiocoap.Message at 0x7f05d621fcc0: Type.CON POST (MID 1, empty token) remote
<UDP6EndpointAddress 192.168.1.79:37286 (locally 192.168.1.79%lo)>, 1 option(s), 4 byte(s) payload>
DEBUG:coap-server:New unique message received
--------------------
payload: b'32332e35'
DEBUG:coap-server:Sending message <aiocoap.Message at 0x7f05d622df98: Type.ACK 2.04 Changed (MID 1, empty token)
remote <UDP6EndpointAddress 192.168.1.79:37286 (locally 192.168.1.79%lo)>>
DEBUG:coap-server:Incoming message <aiocoap.Message at 0x7f05d621fcc0: Type.CON POST (MID 1, empty token) remote
<UDP6EndpointAddress 192.168.1.79:37286 (locally 192.168.1.79%lo)>, 1 option(s), 4 byte(s) payload>
INFO:coap-server:Duplicate CON received, sending old response again
DEBUG:coap-server:Sending message <aiocoap.Message at 0x7f05d622df98: Type.ACK 2.04 Changed (MID 1, empty token)
remote <UDP6EndpointAddress 192.168.1.79:37286 (locally 192.168.1.79%lo)>>
DEBUG:coap-server:Socket error recevied, details: SockExtendedErr(ee_errno=111, ee_origin=2, ee_type=3, ee_code=3,
ee_pad=0, ee_info=0, ee_data=0)
DEBUG:coap-server:Incoming error 111 from <UDP6EndpointAddress 192.168.1.79:37286 (locally 192.168.1.79%lo)>
DEBUG:coap-server:Incoming message <aiocoap.Message at 0x7f05d622deb8: Type.CON POST (MID 1, empty token) remote
<UDP6EndpointAddress 192.168.1.79:37286 (locally 192.168.1.79%lo)>, 1 option(s), 4 byte(s) payload>
INFO:coap-server:Duplicate CON received, sending old response again
DEBUG:coap-server:Sending message <aiocoap.Message at 0x7f05d622df98: Type.ACK 2.04 Changed (MID 1, empty token)
remote <UDP6EndpointAddress 192.168.1.79:37286 (locally 192.168.1.79%lo)>>
DEBUG:coap-server:Socket error recevied, details: SockExtendedErr(ee_errno=111, ee_origin=2, ee_type=3, ee_code=3,
ee_pad=0, ee_info=0, ee_data=0)
DEBUG:coap-server:Incoming error 111 from <UDP6EndpointAddress 192.168.1.79:37286 (locally 192.168.1.79%lo)>
```

Several phenomena may be observed :

— the server had kept in memory the previous requests but had not processed them because the program had been stopped. By restarting it, it will be able to process them. It then replies to the client, which can display the notification 2.04 ;

— the other requests are seen as repeats of the first one since the Message ID field is identical. The server simply returns a copy ;

— the client having finished its transaction has closed the socket leading to the emission of an ICMP message which leads to the display of the error log message (Socket error received).

This can be verified on the Wireshark capture (see figure 13.5).

13.3.2 Resource coded in CBOR

The previous program is correct but we used the default ASCII transfer format (***Content-format*** = 0). It would be better to return to the JSON or **CBOR** format that we had seen previously. To do this, we must add the Content-format option to the request header. As its code is 12, it must be placed after the Uri-path option (see table 12.3 on page 157).

Listing 13.9 – coap_post_temp2.py

```
import CoAP
import socket
```

3. The standard recommends adding a randomness during the transmission to avoid that two equipments synchronize and disturb each other always at the same time. It is not implemented in this example

```
import time
try:
    import kpn_senml.cbor_encoder as cbor #pycom
except:
    import cbor2 as cbor  # terminal on computer

SERVER = "192.168.1.26" # change to your server's IP address
PORT   = 5683
destination = (SERVER, PORT)

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

coap = CoAP.Message()
coap.new_header(code=CoAP.POST)
coap.add_option (CoAP.Uri_path, "temp")
coap.add_option (CoAP.Content_format, CoAP.Content_format_CBOR)
coap.add_payload(cbor.dumps(23.5))
coap.dump()

answer = CoAP.send_ack(s, destination, coap)
answer.dump()
```

Note that this program, pprogcoap_post_temp2.pypycom, can be run on your computer or on your LoPy. In the first case, the module `cbor2` will be used. On the LoPy, we will use the module `kpn_senml`. In both cases, the content is transformed into CBOR thanks to the function `dumps`.

On the server side, the methodtextttrender_post must know the format of the resource. To do this, it must access the CoAP options.

Listing 13.10 – coap_basic_server3.py

```
        return aiocoap.Message(payload=payload)

class TemperatureResource(resource.Resource):
    async def render_post(self, request):
        print ("-"*20)

        ct = request.opt.content_format or \
                aiocoap.numbers.media_types_rev['text/plain']

        if ct == aiocoap.numbers.media_types_rev['text/plain']:
            print ("text:", request.payload)
        elif ct == aiocoap.numbers.media_types_rev['application/cbor']:
            print ("cbor:", cbor.loads(request.payload))
        else:
            print ("Unknown format")
            return aiocoap.Message(code=aiocoap.UNSUPPORTED_MEDIA_TYPE)
        return aiocoap.Message(code=aiocoap.CHANGED)
```

In the program `coap_basic_server3.py`, the variable `ct` contains the value of the option **Content-format** if it exists. In general, *aiocoap* allows access to the values of all CoAP options contained in the request. The value `None` indicates that the option is not present in the header. In this case, the variable `ct` will contain the default value indicating ASCII text.

Depending on the format of the data, the program will display the text or the CBOR transformed into a character string with the function `loads`. If the format is not known to the program, an error

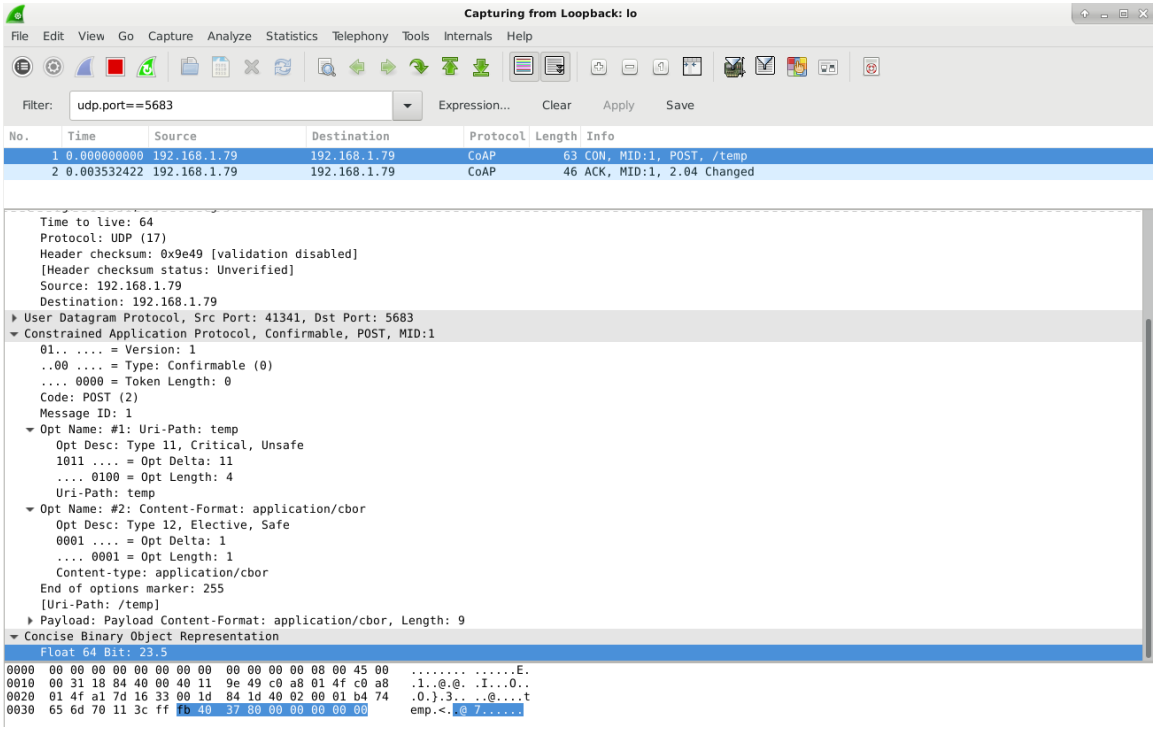


FIGURE 13.6 – Full POST traffic

code is returned to the client.

The figure 13.6 shows the exchange related to the POST and details the content of the request.

Question 13.3.1: Response

What do we receive in response to the POST request ?

- an ACK message
- status 2.00 OK.
- status 2.04 CHANGED.
- nothing.

Question 13.3.2: JSON

Modify the client program to specify a JSON content-format. What notification do you get ?

- an RST message
- le statut 4.04 NOT FOUND.
- le statut 2.15.
- statut 5.00.

13.3.3 No Response

Whether you use the CONfirmable or NON-confirmable type, the server will send a notification :

— 2.xx when everything is going well, and

— 4.xx or 5.xx when the client or server has made an error.

If a sensor wants to use CoAP on an LPWAN network, for each POST it sends on the uplink, it will get an acknowledgement in the downlink. However, we have seen that we must be careful with the downlink channel, which could be subject to saturation or billing.

The **No response** option defined in RFC 7967 allows a client to inform the server that it does not wish to receive notifications during a POST or PUT. The value of the option is 258 and it is followed by a byte containing a bitmap that will indicate what type of notification will be sent :

— if the second bit from the right is set to 1, the client does not want to receive notifications of type 2.xx ;
— if the fourth bit from the right is set to 1, the client does not want to receive notifications of type 4.xx ;
— if the fifth bit from the right is set to 1, the client does not want to receive notifications of type 5.xx.

For example, by setting the value `0x02` in this option, the client does not receive positive acknowledgements but only errors 4.xx and 5.xx. By setting the value `0b00011010` (0x1a, 26), the server will be completely silent.

The program `coap_post_temp4.py` adds this option and the type of the CoAP message has been set to NOT confirmable. However, the URI path is not known to the server. We see that the server returns an error message. If, on the other hand, the POST is successful, only the client sends data and the server remains silent.

Listing 13.11 – coap_post_temp4.py

```
coap = CoAP.Message()
coap.new_header(type=CoAP.NON, code=CoAP.POST)
coap.add_option (CoAP.Uri_path, "temp")
coap.add_option (CoAP.Content_format, CoAP.Content_format_CBOR)
coap.add_option (CoAP.No_Response, 0b00000010)
coap.add_payload(cbor.dumps(23.5))
coap.dump()
```

13.4 Complete chain of measurements

Voilà ! Now we can put the different concepts together to build a complete feedback chain for temperature, humidity and pressure.

Youtube

On the sensor side, we will :

— if the BME280 is present, retrieve the data directly. Otherwise, we will use virtual measurements. We will also define the network protocol : WiFi, Sigfox, LoRaWAN ;
— if we use a LoRaWAN network, we have to set up a relay program to transmit the data to the CoAP server ;

The CoAP server must be able to send the data to the Beebotte server for display.

The CoAP message will be configured as follows :

—
—

—
—

There is still a point to be dealt with because with the 12 bytes limitation of the frames **Sigfox**, it is impossible to transmit data if one uses CoAP. Indeed, the CoAP header will contain :

— 4 bytes for the mandatory header ;
— 0 Token bytes ;
— 2 bytes minimum for the URI-path option if the path is reduced to one character ; for example `T`, `H`, `P` to represent temperature, humidity and pressure ;
— 2 bytes for the option **Content-format** necessary to indicate that one carries CBOR ;
— 3 bytes for the option **No-response**, also essential to indicate that we do not want any acknowledgement (Sigfox limiting them to 4 per day).

That is, a total of 11 bytes. There is only one left. If the temperature exceeds 23°C, the information will not be transportable.

13.5 SCHC

Static Context Header Compression, defined in the RFC 8724, gives the acronym SCHC which we pronounce Chic. SCHC proposes a generic compression mechanism for headers. SCHC is based on a context common to the sender and the receiver which will make it possible to eliminate the known information in the message and to transmit only the data which cannot be predetermined.

We will implement a simplified version. In the CoAP message, we need the Message ID field that will be used to eliminate duplicates that might appear in the network. The CoAP servers keep the information about the Message IDs in memory for 5 minutes. Therefore, the message numbering must allow for longer periods before reusing the same values. We will also carry 3 types of resources : `/temperature`, `/humidity` and `/pressure`.

We can therefore manually build a SCHC header with the following fields :

— 2 bits to number the compression rules. In our case, we will only use rule 00 ;
— 4 bits to number the ID messages, which gives 15 possible values from 1 to 15. In the worst case, more than 3 frames per minute would have to be transmitted for the CoAP server to treat two different frames as duplicates ;
— 2 bits to designate URI paths ; 3 will be used.

This involves several compression techniques defined by SCHC :

— ***not_sent*** : the value of a field is not sent on the network because it is in the rules. This will apply to most fields ;
— ***Least Significant Bit*** : only the least significant bits are sent. This applies to the Message ID field, for which only the 4 least significant bits will be transmitted ;
— ***Matching_sent*** : instead of sending the value, we will send an index on a common array. This applies to Uri-path where we will send 00 for the temperature element, 01 for the pressure element, and 10 for the humidity element.

13.5.1 Client-side transmission

The very simplified compression that we will carry out will be done at the time of the sending of the data for Sigfox in the program `coap_full_sensor.py`.

Listing 13.12 – coap_full_sensor.py

```
174 def send_coap_message(sock, destination, uri_path, message):
    if destination[0] == "SIGFOX": # do SCHC compression
176     global sigfox_MID

178     """ SCHC compression for Sigfox, use rule ID 0 stored on 2 bits,
        followed by MID on 4 bits and 2 bits for an index on Uri-path.
180
        the SCHC header is RRMMMMUU
182     """
        uri_idx = ['temperature', 'pressure', 'humidity', 'memory'].index(uri_path)
184
        schc_residue = 0x00 # ruleID in 2 bits RR
186     schc_residue |= (sigfox_MID << 2) | uri_idx # MMMM and UU

188     sigfox_MID += 1
        sigfox_MID &= 0x0F # on 4 bits
190     if sigfox_MID == 0: sigfox_MID = 1 # never use MID = 0

192     msg = struct.pack("!B", schc_residue) # add SCHC header to the message
        msg += cbor.dumps(message)
194
        print ("length", len(msg), binascii.hexlify(msg))
196
        s.send(msg)
198     return None # don't use downlink
```

In the case of Sigfox, one takes the index by seeking the element in the table (line 183). One builds then the byte SCHC by adding a number of rule (line 185) and the 4 bits of low weight of the field Message ID which will thus vary from 1 to 15 (line 186). Then, one concatenates the CBOR data to be sent (line 193).

For other transmission technologies, the CoAP header is built with the functions of the CoAP.py module.

```
200     # for other technologies we wend a regular CoAP message
        coap = CoAP.Message()
202     coap.new_header(type=CoAP.NON, code=CoAP.POST)
        coap.add_option (CoAP.Uri_path, uri_path)
204     coap.add_option (CoAP.Content_format, CoAP.Content_format_CBOR)
        coap.add_option (CoAP.No_Response, 0b00000010) # block 2.xx notification
206     coap.add_payload(cbor.dumps(message))
        coap.dump(hexa=True)
208     answer = CoAP.send_ack(s, destination, coap)
```

The sending of the frame is done with the function `send_ack`. As the message is of type NO, this function does not wait for a response after the data is sent.

13.5.2 Server side reception

If the data sent by `coap_full_sensor.py` passes through an LPWAN network, the program `generic_coap_relay.py` will act as an intermediary to send it to the CoAP server. The program is almost identical to the one we used to transmit in the previous session. The only differences are :

— the port numbers used : 5683 instead of 33033 ;
— the addition of SCHC decompression for data coming from Sigfox.

Listing 13.13 – generic_coap_relay.py

```
        # Sigfox use SCHC compression, first byte is CoAP compressed header
70      SCHC_byte = payload[0]

72      ruleID = SCHC_byte >> 6
        mID = (SCHC_byte & 0b00111100) >> 2
74      uri_idx = SCHC_byte & 0b0000_0011

76      m = aiocoap.message.Message(
            mtype=NON,
78          code=POST,
            mid=mID,
80          payload=payload[1:]
            )
82      m.opt.uri_path = (
            ["temperature", "pressure", "humidity", "memory"][uri_idx],
84      )

86      m.opt.content_format = 60
        m.opt.no_response = 0b0000_0010
88
        downlink = forward_data(m.encode())
```

To reconstruct the CoAP header, SCHC relies on rules to make the processing independent of the fields used by the protocol. But here, to make it simpler, we use the Message module of **aoicoap** to reconstruct the CoAP message.

Initially, one extracts from the first byte the value of the field message ID and the index of the URI (line 72 to 74) then, starting from these values and those known in advance, the CoAP message is reconstituted.

In all cases, the function `forward_data` is used. It returns the answer of the CoAP server which is sent back to the LPWAN network according to the principles we had seen during the previous session. We did not implement it for Sigfox to avoid an error since 4 messages per day are allowed.

13.5.3 CoAP server

Finally, the program `coap_server.py` has been extended to different URI paths to process the different resources that the sensors will send us. And in the treatment of the resource, the call to the function `to_bbt` was added.

13.6 Ideas for improvement

You can extend this set of programs. Here are some ways to improve :

— sensors also send their available memory. This can be useful to detect a memory leak ; for example when a structure is never released. The CBOR structure is sent but neither `coap_server.py` nor Beebotte have been configured to display these values. You can therefore integrate it into the processing chain ;
— the POST from memory leads to a downlink message to indicate error 4.04. This is a consequence of using the CoAP option **no_response** which only blocks notifications of type 2.xx. You can modify the CoAP.py module so that the response is processed. For example, by limiting this resource to one submission per day ;
— In addition to the memory, it may be interesting to send the battery level. The page Correct formula for BATT monitoring on expansion board | Pycom user forum [4] gives indications to recover the battery level ;
— Finally, we did not solve all the interoperability problems. If, for example, you want to send every hour a reading of the free memory and every minute the temperature, you must also modify the program `coap_server.py`. If these parameters are transmitted optimally by the server, the program `coap_server` can become completely independent of the measured value.

4. https ://forum.pycom.io/topic/1690/correct-formula-for-batt-monitoring-on-expansion-board

14. LwM2M

It's time to see a real IoT platform that implements what we just learned about CoAP and REST. We need to install two programs **java** [1] :

```
wget https://ci.eclipse.org/leshan/job/leshan/lastSuccessfulBuild/artifact/leshan-client-demo.jar
```

and

```
wget https://ci.eclipse.org/leshan/job/leshan/lastSuccessfulBuild/artifact/leshan-server-demo.jar
```

14.1 Introduction

Lightweight Machine to Machine (LwM2M) is an architecture defined by the Open Mobile Alliance (OMA) at the origin to allow the operators to manage certain resources on the cell phones. But this architecture can be extended to other environments.

The specifications are available on the OMA site [2]. We will use a java implementation called . Don't panic we are not going to program we will just need a web browser and **Wireshark**.

As its name indicates, LwM2M is meant to be light, meaning that the implementations should not be too complex and that the traffic generated should not be too important either. LwM2M is a platform and therefore it will do more than just REST traffic. In particular, one of the roles is to allow objects to register and describe their characteristics. LmM2M will also strongly structure the resources by imposing relatively constrained naming rules and resource format.

14.2 Architecture

Like any self-respecting system, LwM2M works in client/server mode. The server is the object management platform and the clients are the objects connected to the network.

1. This is a fairly old implementation, new specifications exist and are somewhat different, but the concepts have not changed
2. `https://omaspecworks.org/what-is-oma-specworks/iot/lightweight-m2m-lwm2m/`

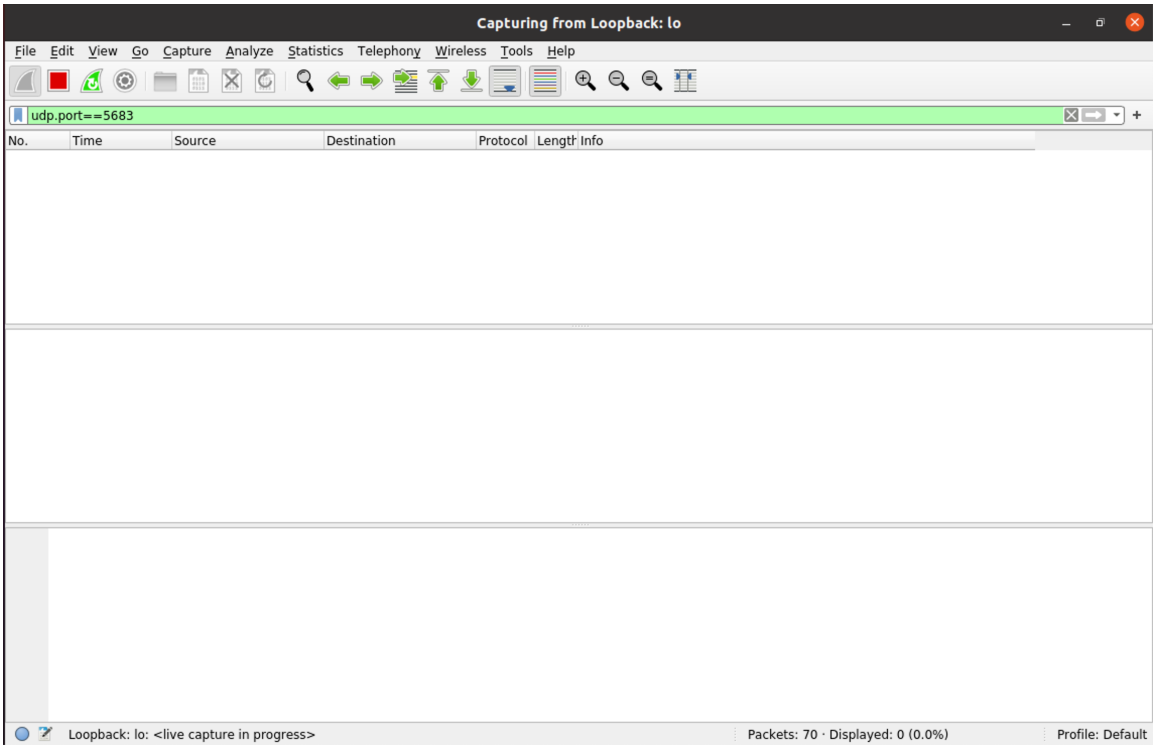


FIGURE 14.1 – Initializing Wireshark to capture CoAP traffic

First, launch Wireshark on the interface and filter the traffic by limiting it to port 5683 for UDP (the one of CoAP) by typing `udp.port==5683`. As shown in figure 14.1.

We are now going to launch the LwM2M server, type [3] :

```
> java -jar ./leshan-server-demo.jar
2020-10-22 01:34:09,365 INFO LeshanServer - LWM2M server started at
coap://0.0.0.0/0.0.0.0:5683 coaps://0.0.0.0/0.0.0.0:5684
2020-10-22 01:34:09,553 INFO LeshanServerDemo - Web server started at
http://0.0.0.0:8080/.
```

It tells us that it uses port 5683 for CoAP and that we can monitor the server with a browser on port 8080.

Launch the browser on the URI `http://127.0.0.1:8080`, the page indicated figure reffig-lwm2m-server1 appears.

You may notice that opening the LwM2M server did not cause any CoAP traffic on the network analyzer.

Now launch the LwM2M client in another window :

```
> java -jar ./leshan-client-demo.jar
2020-10-22 01:49:51,063 INFO LeshanClientDemo - Commands available :

  - create <objectId> : to enable a new object.
```

3. Under Linux, type `apt install -y default-jre` to install Java.

FIGURE 14.2 – LwM2M server home page

```
  - delete <objectId> : to disable a new object.
  - update : to trigger a registration update.
  - w : to move to North.
  - a : to move to East.
  - s : to move to South.
  - d : to move to West.

2020-10-22 01:49:51,064 INFO LeshanClient - Starting Leshan client ...
2020-10-22 01:49:51,158 INFO CaliforniumEndpointsManager - New
endpoint created for server coap://localhost:5683 at
coap://0.0.0.0:48274
2020-10-22 01:49:51,159 INFO LeshanClient - Leshan client
[endpoint:ubuntu] started.
2020-10-22 01:49:51,160 INFO DefaultRegistrationEngine - Trying to
register to coap://localhost:5683 ...
2020-10-22 01:49:51,252 INFO DefaultRegistrationEngine - Registered
with location '/rd/BOr5Pg7yW8'.
2020-10-22 01:49:51,252 INFO DefaultRegistrationEngine - Next
registration update to coap://localhost:5683 in 53s...
```

14.3 Registration of an Object

Since we are using an address, it is a bit more difficult to identify in the recovered Wireshark traffic (see figure 14.3 on the following page) who is the client and who is the server. But since the server is waiting on port 5683, by looking more closely at a packet, we can determine whether it was sent by the client or the server.

14.3.1 Analysis of the CoAP header

We can see on the trace, that the client contacts the server to indicate its properties.

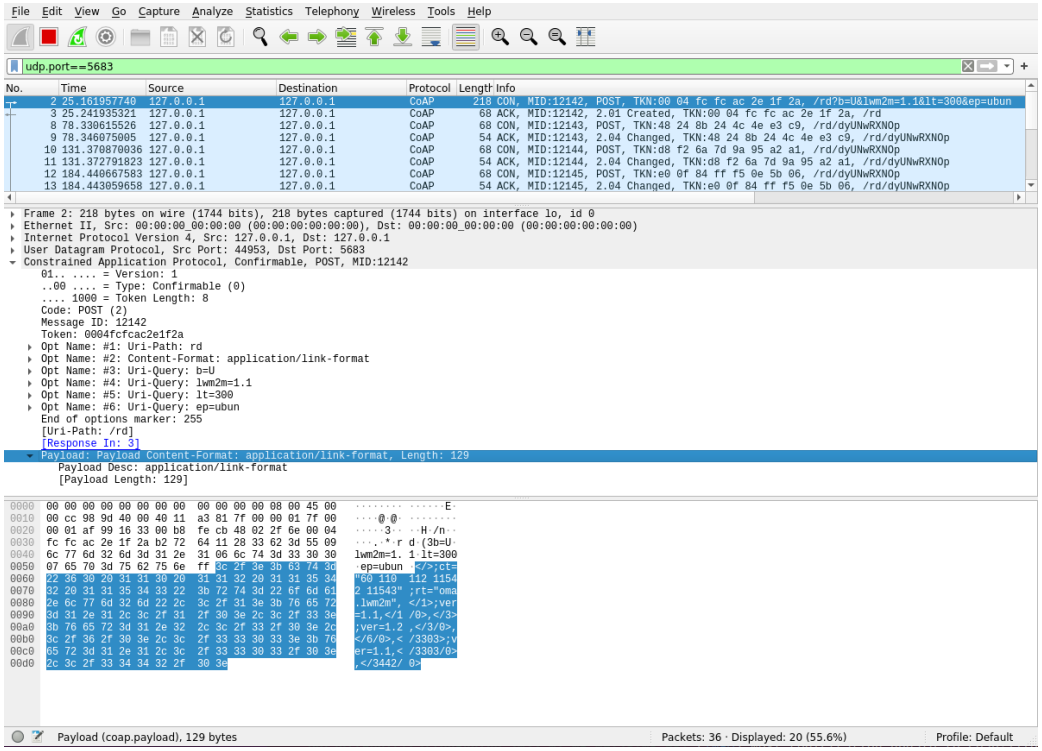


FIGURE 14.3 – First LwM2M captures

Question 14.3.1: Method

What is the nature of the request issued by the client :

- GET
- POST
- PUT
- DELETE

Question 14.3.2: URI

What is the path of the URI :

- empty
- /rd
- /lwm2m

Question 14.3.3: Content

What is the format of the content (content-format) :
- text
- XML
- JSON
- link-format
- CBOR

Question 14.3.4: Period

What is the period in which you see CoAP messages ?

To decode the Uri-Query of the CoAP request it is necessary to help the document LwM2M Core Specification [4] and its table 6.2.1 page 40.

Question 14.3.5: b=U

What does b=U mean ?
- The data will be sent with the units
- The reference of the units is the Universal representation
- The underlying protocol is in datagram mode (UDP)
- The client is not referenced (Unreferenced)

Question 14.3.6: lwm2m=1.1

What does lwm2m=1.1 mean ? It is a question of...
- the lwm2m client version
- the memory size of the implementation (1.1 kb)
- the lwm2m server version

Question 14.3.7: lt=300

What does lt=300 mean ?
- This is the maximum number of objects (less than 300)
- This is the maximum size of an exchange (300 bytes)
- This is the lifetime of an object's registration

14.3.2 POST Content Analysis

Let's go back to the message content of the POST request initially sent by the client. What does **link-format** mean ? We haven't seen this format yet. Fortunately, the Internet Assigned Numbers Authority (IANA) is our friend and by going to look for what this value corresponds to, on this page [5], we find that the RFC 6690 defines this content.

4. http://www.openmobilealliance.org/release/LightweightM2M/V1_1_1-20190617-A/OMA-TS-LightweightM2M_Core-V1_1_1-20190617-A.pdf
5. https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#content-formats

It uses a particular format which is used initially by the Web to define relations between pages initially defined in the RFC 5988. It is important to notice, and after the reading becomes clearer, that the URI are noted between <>. Then, we find attributes linked to this URI separated by semicolons. The commas separate the definitions.

En suivant ces règles de notation, les données émises par le client peuvent être formatés de la manière suivante :

```
</>;ct="60 110 112 11542 11543";rt="oma.lwm2m",
</1>;ver=1.1,
</1/0>,
</3>;ver=1.2,
</3/0>,
</6/0>,
</3303>;ver=1.1,
</3303/0>,
</3442/0>
```

The client uses this format to describe the 9 resources it owns, which are identified by these 9 URI paths. The naming of the resources may seem strange ; we will see later what it corresponds to :

- The first `ct` [6] defines the formats of the possible representations of the objects. The part between quotation marks refers to the Content-format values of CoAP listed in table 12.4 on page 160. We find there respectively the CBOR, SenML types and for the last two a TLV oriented format specific to LwM2M.
- The second `rt` [7] indicates the type of resource (*resource-type*), i.e. how they will be represented. Here, it indicates that the resources will follow the LwM2M specifications of the OMA.
- The following lines describe, again in a hierarchical manner, the URI paths and associated attributes. The attribute `Indexver` indicates the version of the standard used to define the resource.

Question 14.3.8: rt

What is the purpose of the rt attribute ?

- to give the semantics (i.e. how to interpret) of the resource.
- to indicate the size of the resource with reference to lwm2m.
- to indicate the version of lwm2m used.
- to help debug the exchanges.

LwM2M represents resources in a way that is original enough to be both compact, universal (which is sometimes oxymoronic) and flexible. To do this, everything is designated by numbers. Chapter 7 page 63 of the document. principalfootnote`http://www.openmobilealliance.org/release/LightweightM2M/V1_1_1-20190617-A/OMA-TS-LightweightM2M_Core-V1_1_1-20190617-A.pdf` contains the figure 14.4 on the facing page diagram illustrating this hierarchy :

- A physical Object (i.e. our client) contains a list of numerical objects. LwM2M calls this information *object* which makes the definition ambiguous since Objet is also the translation of *thing* identified by a number. This number is assigned by the OMA. For example, a temperature

6. See RFC 7252 chapter 7.2.
7. Voir RFC 6690 chapter 3.1.

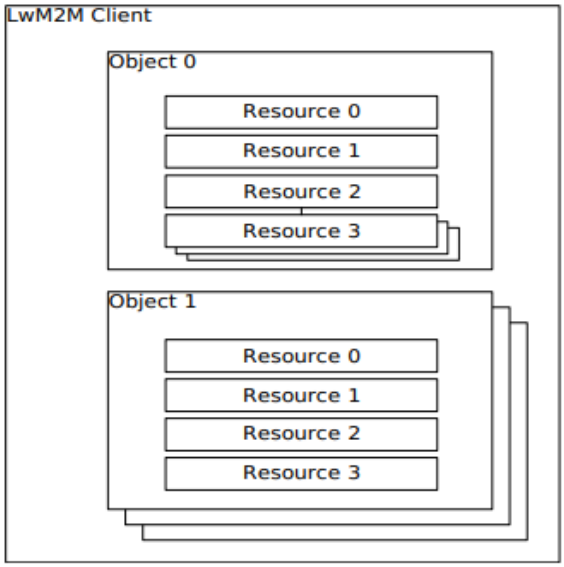


Figure: 7.1.-1 Relationship between LwM2M Client, Object, and Resources

FIGURE 14.4 – Hierarchie de nommage

sensor will have the value *texttt3303*. A list of previously assigned values can be found at this address [8].

— Un client peut évidemment posséder plusieurs instance d'un objet numérique, par exemple, une station météo pourrait avoir un capteurs de température intérieur et extérieur. Le deuxième chiffre du chemin de l'URI indique cette instance. S'il n'y en a qu'un, il prend la valeur *0*.

— Finally, the object can be complex and contain several information called resources. By clicking on the number *textttt3303* in the previous web page, we obtain the description of the object. We can see that :
  — *5700* represents the value measured by the sensor
  — *5601* the minimum value
  — *5602* the maximum value
  — *5701* indicates the units
  — *5605* allows you to reset the calculation of minimum and maximum values

  These values can be found in several numerical objects.

— version 1.2 of the standard also introduces the possibility to have several instances of a resource.

Digital objects can be defined by LwM2M, they concern those dedicated to the management of the protocol. For example, the digital object :

— *0* defines the security environment of the client. It contains the server address, the encryption keys,...

— *1* defines the parameters for the communication with the server, such as the lifetime of the information in seconds,...

— *3* describes the characteristics of the equipment such as its software version,...

8. `http://www.openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html#resources`

— ...

The values between 2 048 and 10 240 are defined by partner standardization organizations of OMA, such as :

— the IPSO Alliance which will define typical digital objects, such as a temperature sensor with reference *textttt3303*,

—

— ...

Higher values are reserved for manufacturers who can register their own specifications.

Values associated with resources

The figure 14.5 on the next page, copied from the OMA website, gives an example of the definition of a digital object. To the object 3303 are associated a certain number of resources, generic that can also be found in other digital objects. Thus :

— *5700* fait référence à la valeur mesurée représenté comme un nombre flottant. Ce nombre ne peut que être lu (R)

— *5700* is a string that specifies the unit of measurement.

— *5601* and *5602* keep the minimum and maximum values. They can be reset via the resource *5605* which leads to a program execution (E).

— *5603* and *5604* are within the operating range of the sensor and cannot be changed.

Thus *3303/3/5601* represents the minimum value (*5601*) of the fourth sensor (*3* because we start at 0) of the numerical object temperature (*3303*).

Compared to **Modbus**, where one had to download the documentation of the Object (cf table 4.3 on page 54), LwM2M offers a standard way to describe the information produced or consumed by an Object. Moreover, the definition of the digital object, in addition to being visualized in the form of table on the Web site, also exists in indexXML to be treated informatically and to allow interoperability.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- BSD-3 Clause License ... -->

<LWM2M  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=
"http://openmobilealliance.org/tech/profiles/LWM2M.xsd">
        <Object ObjectType="MODefinition">
                <Name>Temperature</Name>
                <Description1>
                    This IPSO object should be used with a temperature sensor to report a temperature
                    measurement.  It also provides resources for minimum/maximum measured values and the
                    minimum/maximum range that can be measured
                    by the temperature sensor. An example measurement unit is degrees Celsius.
                </Description1>
                <ObjectID>3303</ObjectID>
                <ObjectURN>urn:oma:lwm2m:ext:3303:1.1</ObjectURN>
                <LWM2MVersion>1.0</LWM2MVersion>
                <ObjectVersion>1.1</ObjectVersion>
                <MultipleInstances>Multiple</MultipleInstances>
                <Mandatory>Optional</Mandatory>
                <Resources>
                        <Item ID="5700">
                                <Name>Sensor Value</Name>
                                <Operations>R</Operations>
                                <MultipleInstances>Single</MultipleInstances>
                                <Mandatory>Mandatory</Mandatory>
                                <Type>Float</Type>
                                <RangeEnumeration></RangeEnumeration>
                                <Units></Units>
                                <Description>Last or Current Measured Value from the Sensor.</Description>
                        </Item>
                        <Item ID="5601">
                                <Name>Min Measured Value</Name>
                                <Operations>R</Operations>
                                <MultipleInstances>Single</MultipleInstances>
```

Temperature

Description

This IPSO object should be used with a temperature sensor to report a temperature measurement. It also provides resources for minimum/maximum measured values and the minimum/maximum range that can be measured by the temperature sensor. An example measurement unit is degrees Celsius.

Object definition

Name	Object ID	Object Version	LWM2M Version
Temperature	3303	1.1	1.0
Object URN		Instances	Mandatory
urn:oma:lwm2m:ext:3303:1.1		Multiple	Optional

Resource Definitions

ID	Name	Operations	Instances	Mandatory	Type	Range or Enumeration	Units	Description
5700	Sensor Value	R	Single	Mandatory	Float			Last or Current Measured Value from the Sensor.
5601	Min Measured Value	R	Single	Optional	Float			The minimum value measured by the sensor since power ON or reset.
5602	Max Measured Value	R	Single	Optional	Float			The maximum value measured by the sensor since power ON or reset.
5603	Min Range Value	R	Single	Optional	Float			The minimum value that can be measured by the sensor.
5604	Max Range Value	R	Single	Optional	Float			The maximum value that can be measured by the sensor.
5701	Sensor Units	R	Single	Optional	String			Measurement Units Definition.
5605	Reset Min and Max Measured Values	E	Single	Optional				Reset the Min and Max Measured Values to Current Value.
5750	Application Type	RW	Single	Optional	String			The application type of the sensor or actuator as a string depending on the use case.
5518	Timestamp	R	Single	Optional	Time			The timestamp of when the measurement was performed.
6050	Fractional Timestamp	R	Single	Optional	Float	0..1	s	Fractional part of the timestamp when sub-second precision is used (e.g., 0.23 for 230 ms).
6042	Measurement Quality Indicator	R	Single	Optional	Integer	0..23		Measurement quality indicator reported by a smart sensor. 0: UNCHECKED No quality checks were done because they do not exist or can not be applied. 1: REJECTED WITH CERTAINTY The measured value is invalid. 2: REJECTED WITH PROBABILITY The measured value is likely invalid. 3: ACCEPTED BUT SUSPICIOUS The measured value is likely OK. 4: ACCEPTED The measured value is OK. 5-15: Reserved for future extensions. 16-23: Vendor specific measurement quality.
6049	Measurement Quality Level	R	Single	Optional	Integer	0..100		Measurement quality level reported by a smart sensor. Quality level 100 means that the

FIGURE 14.5 – Definition of the numerical object 3303 for the temperature

```
                    <Mandatory>Optional</Mandatory>
                    <Type>Float</Type>
                    <RangeEnumeration></RangeEnumeration>
                    <Units></Units>
                    <Description>
                      The minimum value measured by the sensor since power ON or reset.
                      </Description>
            </Item>
            ...
```

Question 14.3.9: 3301/0/5602

Using the digital object and resource description page [a], what does the URI *3301/0/5602* represent ?

- the maximum temperature of the first sensor
- the minimum humidity of the sensor 0
- the maximum brightness of the sensor 0

a. http://www.openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html#resources

Question 14.3.10: 10340/0/2

What does the URI *texttt10340/0/2* represent ?

- lhe temperature in Fahrenheit of the object 10340
- longitude in GPS coordinates
- the active or not status of a camera

Question 14.3.11: Dimmer

Which URI allows access to the element controlling the light variation (*Dimmer*) of a lighting (*Light Control*) ? What is the maximum value this resource can take ?

Question 14.3.12: Hz

What would be the URI path to obtain the frequency of an electric current on the second phase ?

Question 14.3.13: Who is who ?

In the first captured exchange :

```
</>;ct="60 110 112 11542 11543";rt="oma.lwm2m",
</1>;ver=1.1,
</1/0>,
</3>;ver=1.2,
</3/0>,
</6/0>,
</3303>;ver=1.1,
</3303/0>,
</3442/0>
```

1 ☐	☐ Location
3 ☐	☐ Device
6 ☐	☐ LwM2M v1.1 Test Object
3303 ☐	☐ LwM2M Server
3442 ☐	☐ Temperature

14.4 *Resource Directory*

We will focus on the first exchange that we started to analyze by looking at the message sent by the client to the LwM2M server. As we have seen, this first message contains the description of the resources present on the client. The values of the Object ID and Resource ID allow the server to know what the client can do, since they are normalized. The LwM2M client sends this information on a well-known URI path `/rd` (for *resource description*).

Question 14.4.1: LwM2M server response

What does the server answer (see figure 14.6 on the next page) ?

- It just acknowledges the message.
-
-
-

Question 14.4.2: Wait and See

If we let the platform run without intervention, what do we see on the network analyzer ?

- Nothing.
- The sensor sends information indicating a change in the measured value.
- The customer sends the measured values even if they have not changed.
- The client sends an empty message to the server to indicate that it is still present.
- The server sends an empty message to its clients to see if they are still present.

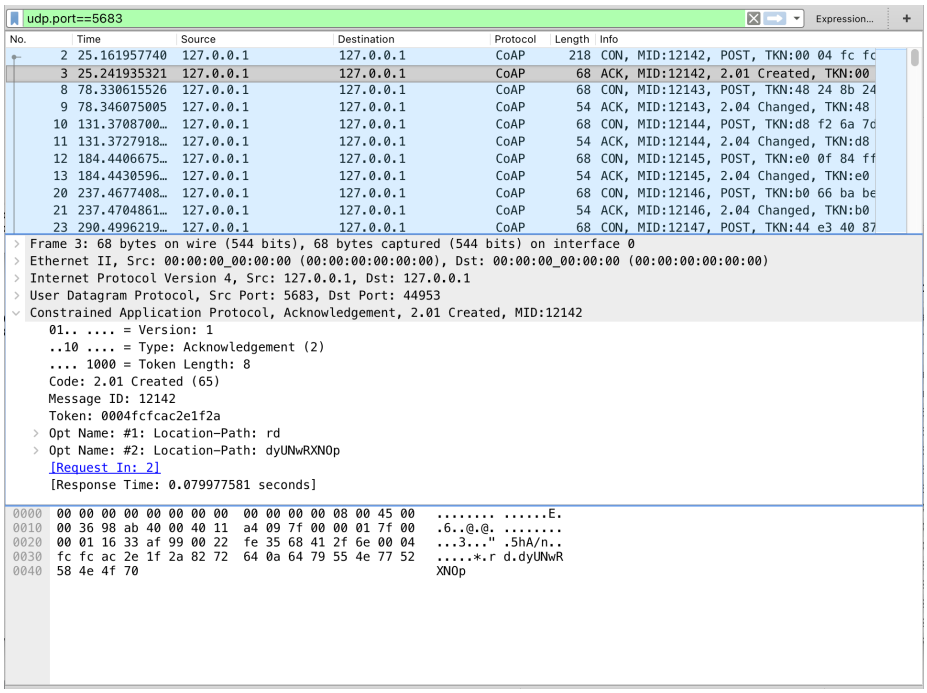


FIGURE 14.6 – LwM2M server response to client POST

14.5 interrogation of the LwM2M client

It is time to return to the platform interface by opening the `http://127.0.0.1:8080` page from the browser. Make sure Wireshark is still capturing traffic on the *loopback* interface. The object we have listed appears. Click on it. You should have something like what is shown in figure 14.7 on the facing page.

In the left menu, we find the LwM2M digital objects described in the first POST of the Object.

By clicking on *Temperature*, the LwM2M resources are displayed. The small logos allow to perform actions :

— *R* to read a resource, *W* to write it and *EXE* to execute a code (the gear allows to define the parameters) ;
— *OBS* allows to launch an **Observe** on a resource, the eye with the cross allows to cancel an Observe.

These actions can be applied individually to each resource, or more globally to an instance of a digital object.

14.5.1 Simple reading

In the left-hand menu, select SENML from the *Single Value* drop-down menu. This will use the **SenML** format rather than the one specified by LwM2M based on TLV.

For the numerical object *Temperature* click on the button `textitR` of *Sensor Value*. Figure 14.8 on page 194 mounts this exchange and details the response. The LwM2M Server acts as a REST

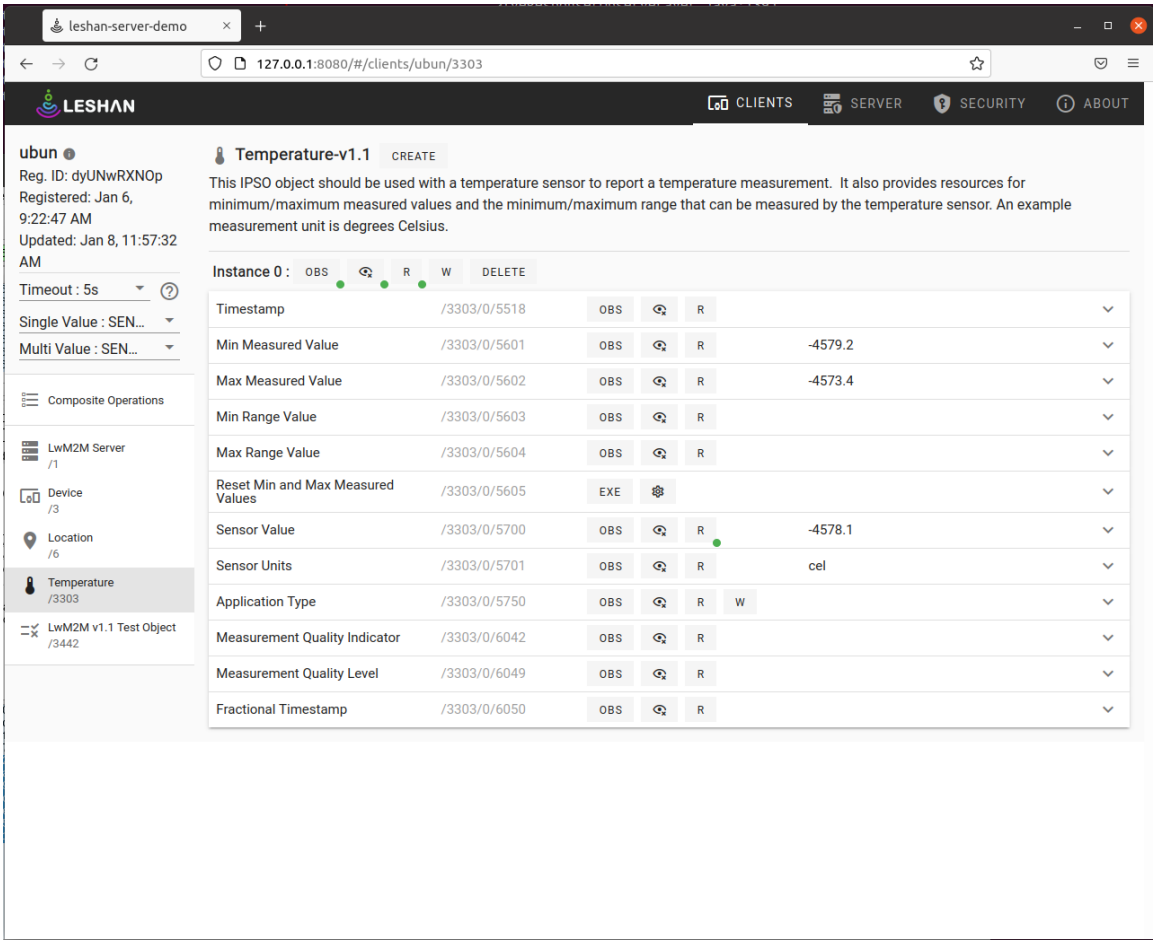


FIGURE 14.7 – Visualization of the LwM2M server

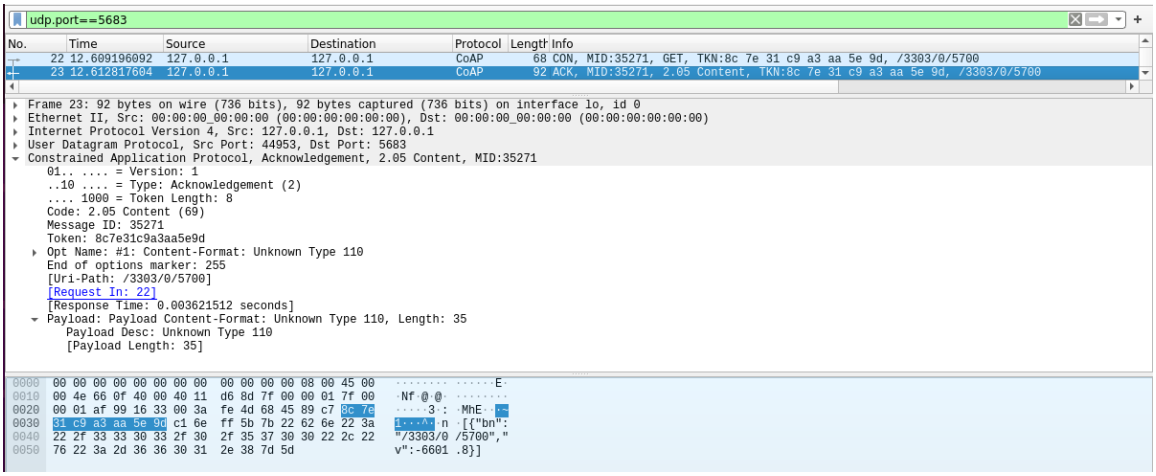


FIGURE 14.8 – Query a resource

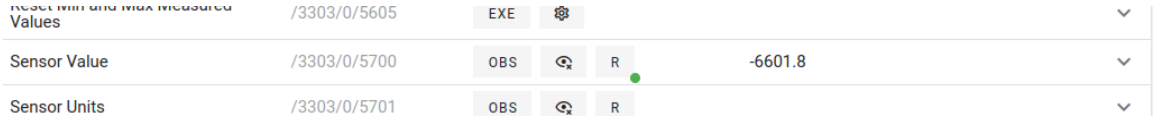


FIGURE 14.9 – Temperature reading

client and sends the LwM2M client's REST server the request :

```
GET /3303/0/5700
```

by indicating the type 110 in the **Accept** option (cf. table 12.4 on page 160) which corresponds to the **SenML** format encoded in JSON .

The response doubly associated by the same **Token** and a message of type **ACK** of format SenML in JSON :

```
[{"bn":"/3303/0/5700","v":-6601.8}]
```

We find the basic name (bn) corresponding to the name of the resource and the value (v) [9].

This value is displayed in the LwM2M server's web page, with the small green dot indicating that the exchange was successful (see figure 14.9).

14.5.2 Reading an instance

By selecting SENML_JSON from the *Multi Value* drop-down menu and clicking on the `R` button to the right of ***Instance 0***, various fields of the instance will be filled in.

The protocol exchange is similar to the previous one. The LwM2M server issued the following request, where the resource value is omitted :

```
GET /3303/0
```

and the answer contains :

9. It is obvious that the returned result does not make any physical sense, the LwM2M client emulating poorly the evolution of a temperature on the long term

```
117.992509491 127.0.0.1    127.0.0.1    CoAP    96 NON, MID:17275, 2.05 Content, TKN:ff 6a 04 4d 3a 24 60 6d, /3303/0/5700
119.992474211 127.0.0.1    127.0.0.1    CoAP    96 NON, MID:17276, 2.05 Content, TKN:ff 6a 04 4d 3a 24 60 6d, /3303/0/5700
121.992141291 127.0.0.1    127.0.0.1    CoAP    96 NON, MID:17277, 2.05 Content, TKN:ff 6a 04 4d 3a 24 60 6d, /3303/0/5700
123.995776262 127.0.0.1    127.0.0.1    CoAP    96 CON, MID:17278, 2.05 Content, TKN:ff 6a 04 4d 3a 24 60 6d, /3303/0/5700
123.998316105 127.0.0.1    127.0.0.1    CoAP    46 ACK, MID:17278, Empty Message
125.992261142 127.0.0.1    127.0.0.1    CoAP    96 NON, MID:17279, 2.05 Content, TKN:ff 6a 04 4d 3a 24 60 6d, /3303/0/5700
127.991978003 127.0.0.1    127.0.0.1    CoAP    96 NON, MID:17280, 2.05 Content, TKN:ff 6a 04 4d 3a 24 60 6d, /3303/0/5700
129.991792102 127.0.0.1    127.0.0.1    CoAP    96 NON, MID:17281, 2.05 Content, TKN:ff 6a 04 4d 3a 24 60 6d, /3303/0/5700
131.992134965 127.0.0.1    127.0.0.1    CoAP    96 NON, MID:17282, 2.05 Content, TKN:ff 6a 04 4d 3a 24 60 6d, /3303/0/5700
```

```
[{"bn":"/3303/0/","n":"5601","v":-6672.6},
{"n":"5602","v":-4573.4},
{"n":"5700","v":-6671.6},
{"n":"5701","vs":"cel"}]
```

We notice that the base name (bn)corresponds to the URI of the instance and that each element contains a name field (n) containing the numerical identifier of the resource [10].

14.5.3 Observe

It is also possible to follow the state of a resource over time. We will do this for the resource *Sensor Value* of the digital object *Temperature*. Click on the button *OBS*.

The LwM2M server sends the GET request as before, having inserted the option ***Observe*** with the value 0 to indicate that it wishes to receive information periodically. The responses also include a ***Observe*** option with an increasing value.

On Wireshark, you will be able to verify that periodically the response is sent by using a message of type ***CON****firmable* rather than ***NON*** *confirmable* to check that the REST client on the LwM2M server is still active (see figure 14.5.3)

Question 14.5.1: End of observe

A click on the eye with the cross allows the server to cancel the Observe. What message is sent ?

Question 14.5.2: EXE

What message is emitted, if an EXE resource is clicked, as resetting the min and max values ?

10. The field (vs) indicates that the content must be interpreted as a character string.

15. Answers to the questions

Question 1.8.1 page 24 Communicating objects are a brand new field, linked to the progress in miniaturization of electronic components :

- True
- **False**

Communicating objects have always existed, even before the deployment of Internet protocols. What is new is their integration into current Internet architectures to better process the information they produce.

Question 1.8.2 page 24 Which of these statements is true ?

- There are very few protocols to make objects communicate. As the Internet is a technology that has been very successful, its success will allow objects to communicate.
- **There are many solutions to enable objects to communicate, the Internet of Things must enable them to be federated.**

Each business has created its own protocols.

Question 1.8.3 page 24 What is the primary source of data creation in the IoT ?

- **sensors**
- nanocomputers (Raspberry Pi type)
- Internet
- Web servers

Measurements made by sensors are the primary source of IoT data creation.

Question 1.8.4 page 25 What are the main technological challenges for the Internet of Things (3 answers) ?

- ☐ **Have a low energy consumption.**

- ☐ **Have a simplified protocol architecture.**
- ☐ Be able to run on open operating systems like Linux.
- ☐ **Allow to secure data that may be sensitive.**
- ☐ Continuously transmit their status and measured values.

Objects are generally limited in energy and capacity. In order to limit energy consumption, it is not necessary to transmit or receive all the time, so the permanent transmission of data does not go in this direction. Similarly, an operating system such as Linux appears to be oversized.

Question 2.1.1 page 28 In the internet protocol stack, which protocols are responsible for routing packets to their destination (2 answers)

☐ Ethernet	☐ **IPv4**	☐ MQTT	☐ JSON
☐ IEEE	☐ **IPv6**	☐ HTTP	
☐ 802.15.5	☐ UDP	☐ CoAP	
☐ Wi-Fi	☐ TCP	☐ XML	

The protocol IP makes it possible to transport information from one end of the network to the other by using the addresses IP contained in the packets. There are two versions of this protocol : Internet Protocol version 4 (IPv4) initially deployed and IPv6 which offers much more addressing capacity.

Question 2.2.1 page 31 What is unique in the world (6 answers) ?

- ☐ A first name.
- ☐ A family name.
- ☐ **a social security number used in France.**
- ☐ **a passport number.**
- ☐ **a cell phone number with its international prefix.**
- ☐ **a full bank account number (IBAN).**
- ☐ the IP address of my machine in my private network.
- ☐ **the IP address of a Coursera server** (`13.225.34.28`)**.**
- ☐ **the domain name** `plido.net`**.**
- ☐ the name of a city.

Neither the first name, nor the last name, nor their combination form unique sequences as John Smith would say. Passport numbers, social security numbers, telephone numbers, bank account numbers are by construction unique in their space, but there could be overlaps. This is why it is necessary to indicate the source or the authority that allocated it to guarantee uniqueness. For the passport, the authority is the country that issued it. The social security number corresponds to the registration number in the Répertoire National d'Identification des Personnes Physiques (RNIPP) (see `https://www.service-public.fr/particuliers/vosdroits/F33078` for France). The country code for the telephone number is assigned by the International Telecommunication Union (ITU) and each operator has its own numbering zone. For the bank account, it is of course the bank ; each bank having its own code contained at the beginning of the IBAN. The IP address in a private network is not unique. The RFC 1918 defines address ranges that everyone can use locally. To go out on the Internet, you need a special device, called a NAT, which will convert the private address into a public address which is unique in the world. The servers must be in this public space and

therefore have a unique address in the world. Domain names are unique in the world by construction but they can be shared by several users. One can thus extend them like `machine1.plido.net`, `machine2.plido.net`... or, if it is the same machine, add a port number after to indicate the service : `www.plido.net:80`, `www.plido.net:8080`. As for the name of a city, it is not unique. It is also necessary to specify the country or even the region.

Question 2.2.2 page 33 The server keeps track of previous requests ?

- True
- **False**

The server responds to one request and then moves on to the next. It does not keep any state. On the other hand, a request can be used to modify the content of a resource, and the result of the modification will be kept.

Question 2.2.3 page 33 The World Wide Web is based on this principle of states for :

- work on both computers and smartphones,
- **be able to serve a large number of requests,**
- encrypt communications.

See previous comment

Question 2.2.4 page 33 What formats are used to represent structured information (2 answers) :

☐ Ethernet	☐ IPv4	☐ MQTT	☐ **JSON**
☐ IEEE	☐ IPv6	☐ HTTP	
☐ 802.15.5	☐ UDP	☐ CoAP	
☐ Wi-Fi	☐ TCP	☐ **XML**	

Il s'agit de XML et JSON qui permettent d'envoyer des données stucturées. Les autres propositions indiquent des protocoles de transport de l'information de niveau 2, 3, 4 et 7.

Question 2.2.5 page 33 In the URI `https://plido.net/unit/definition.html`, where is the scheme ?

`http` : the Scheme indicates how the URI will be constructed.

Question 2.2.6 page 33 In the URI `https://plido.net:8080/unit/definition.html`, where is the authority ?

`plido.net:8080` : this is the globally unique sequence.

Question 3.3.1 page 39 In the first column :

- **The frame number assigned by Wireshark upon reception**
- The frame number is read directly in the Ethernet frame

These numbers are sequential, so they are assigned locally by Wireshark. Moreover there is no such field in Ethernet.

Question 3.3.2 page 39 In the second column :

- **The time of reception by Wireshark**
- The sending time of the frame

As in the previous case, this number is added by Wireshark, there is no protocol field indicating the transmission time.

Question 3.3.3 page 39 In the third and fourth columns :

- The Ethernet addresses of the hosts.
- Only the IPv4 addresses of the machines.
- **IPv4 or IPv6 addresses of machines.**

Wireshark treats IPv4 or IPv6 addresses in the same way, so they are displayed in these columns. The 48-bit Ethernet (or MAC) address is not displayed by default in this screen.

Question 3.3.4 page 39 In the fifth column :

- The application protocol (level 7).
- **The last (higher level) recognized protocol.**
- The level 4 protocol (here TCP or UDP).

Wireshark provides the higher level information. In the figure 3.2 on page 38 some frames are indicated as carrying the HTTP protocol, while others, usually acknowledgements, are indicated as TCP because they do not carry data from the higher layers.

Question 3.3.5 page 39 In the sixth column :

- The size in bits of the frame.
- **The size in bytes of the frame.**

The unit is the byte.

Question 3.3.6 page 39 In the seventh column :

- **A summary of the information carried by the higher-level protocol.**
- IPv4 options.
- The ASCII content of the highest level message.

Wireshark seeks to interpret the higher level protocol fields to provide a synthetic display of the information.

Question 3.3.7 page 41 In the following trace, we saw that the server responded to client requests with a 3-digit number. Using the RFC 7231, can you assign the left digit to a category of notifications :

0 ☐ ☐ Redirection

1 ☐
2 ☐
3 ☐
4 ☐
5 ☐

☐ Error on the server side
☐ Error on the client side
☐ Unassigned
☐ Success
☐ Information

0 : Unassigned
1 : Information
2 : Success
3 : Redirection
4 : Error on the client side
5 : Error on the server side

Question 3.3.8 page 42 Which standards organization published this document ?

- Microsoft
- ISO
- IEEE
- **IETF**

The prefix Request For Comments is characteristic of the IETF.

Question 3.3.9 page 43 Do HTTP headers have a fixed size (you can check the RFC 7231 which gives indications on the protocol) ?

- the header is one line of 80 characters.
- **a blank line separates the header from the content. The header can contain as many lines as necessary.**

As we can see on the example given in the RFC for the response message, the HTTP header contains a mandatory line, followed by options. Their number is not fixed by the standard. To separate them from the data, a blank line acts as a separation.

Question 3.3.10 page 43 How are the optional lines in the header constructed ?

- **keyword : values**
- unformatted text
- keyword : data length : values

The header is of variable size. It has a mandatory first line giving the nature of the request or response, followed by optional information built on the "keyword : value" format. A blank line separates the header from the content.

Question 3.3.11 page 45 In the example, figure 3.4 on page 44, what is the Ethernet address of the machine sending the frame ?

`10:65:30:b0:54:bf`. Be careful, the frame starts with the destination address, followed by the source address.

Question 3.3.12 page 45 Does the Ethernet address `14:2e:5e:37:1e:6a` found in the packet 3.4 on page 44 correspond to the Ethernet address of the recipient of the packet ? What does it correspond

to ?

No, the Ethernet address is only valid on this Ethernet network. To reach the recipient, the packet must cross several routers. As the frame was captured on the sending machine, this address is therefore that of the first router crossed.

Question 3.3.13 page 46 Using the figure 3.4 on page 44 or your Wireshark captures, what are the messages involved in closing the connection ?

The connection closing requires the sending of four messages. One of the ends, necessarily the one that opened the connection, sends a message with the END bit set. It is acknowledged by the other entity which in turn sends a message with the FIN bit set which will be acknowledged to finally close the connection.

Question 3.4.1 page 47 What URI should you enter in your browser to access this server locally.

The loopback address is `127.0.0.1` and the port is 8080, the URI path is `/`. The URI is therefore `http://127.0.0.1:8080/`.

Question 3.4.2 page 47 Using Wireshark, you can determine in the response the values of the HTTP options `IndexContent-Type` and `Server`.

We find the following values :

— `Server: Werkzeug/2.0.2 Python/3.9.6`

— `Content-Type: text/html; charset=utf-8`. The resource is encoded in HTML using 8-bit ASCII encoding.

Question 4.1.1 page 51 Looking at the exchanges in figure 4.4 on page 51 what is the measured value for humidity ?

Only the first exchange asks for the reading of 2 registers from the register with the address 0x0001. In the answer we obtain the value of the two consecutive registers (`00 DA` and `01 D4`). The table 4.1 on page 51 indicates that humidity is the second register, so we have `01 D4`.

Question 4.1.2 page 51 Looking at the exchanges in figure 4.4 on page 51 what is the evolution of the temperature over time ?

We find 3 values measured at 19 :11 :48, 20 :02 :39 and 20 :02 :44 : `00 DA`, `00 D5` and `00 D5`. That is, 21.8°C, 21.3°C and 21.3°C.

Question 4.1.3 page 52 What is the moisture content at the time of measurement ? The documentation states that it is a percentage with an accuracy of one tenth of a percent.

We had read the value `01 D4` in the register 0x02, that is 468 in decimal. Hence a humidity rate of 46.8%

Question 4.1.4 page 56 Let the given exchange figure 4.9 on page 57 correspond to a Modbus request and a response. What is the port number used by Modbus TCP.

There are two ports in the TCP header, but as it is a request, it is necessary to take the destination port :0x1f6, which is 502 in decimal

Question 4.1.5 page 56 Continuing the traffic analysis, which register value is requested.

Register 0x36 is requested, by looking in the table tab-meter-IR, it is the frequency in Hz.

Question 4.1.6 page 56 By analyzing the following packet, how can we verify that the response can match the previous request.

The sequence number 0x000c is the same in both frames.

Question 4.1.7 page 56 What value is returned. Is this consistent ?

The value of the register is 0x4247e95b corresponding to a floating number IEEE 754, converting it we obtain the value 49.9778862 Hz which is very close to the frequency of the European electrical network.

Question 6.8.1 page 79 What are the advantages of CBOR over JSON (2 answers) ?

- ☐ **It is more compact in its data representation.**
- ☐ It allows to represent floating numbers.
- ☐ It compresses strings.
- ☐ **It is easier to implement.**

CBOR does not compress strings. It just adds their length. Both CBOR and JSON encode floating-point numbers, so this is not an advantage of CBOR. On the other hand, using binary values instead of ASCII to represent numbers, not having brackets or braces, makes for a much more compact representation. Small numerical values are represented without any additional cost. Creating a CBOR encoder or decoder is much simpler than in JSON because the data representation is much stricter (no spaces, no line breaks, etc.). Both representations allow to use floating numbers so neither has an advantage on this point.

Question 6.8.2 page 80 Is a float always more compact in CBOR than in JSON ? - You can help yourself with `https://cbor.me`

- ○ Yes, that is the goal of CBOR.
- ○ Yes, for floats that have the decimal part at 0.
- ○ Yes, for small precision floats (up to a hundredth).
- ○ **Yes, for high precision floats (6 digits after the decimal point).**

A floating number, whatever its value, is represented by 8 bytes in CBOR. In JSON, a floating number is represented by a string. So "3.0" needs 3 characters, so it is more compact than CBOR. But "3.1415926" is coded on 9 characters so less compact than CBOR.

Question 6.8.3 page 80 Consider a string in CBOR.

- ○ It is compressed with an entropy algorithm (e.g. Huffman coding).
- ○ **It can contain accented characters.**
- ○ Each character is coded on 6 bits.

Question 6.8.4 page 80 In CBOR, the size of an integer varies according to its value !

- **True**
- False
- It depends on how this integer was declared.

Yes, an integer less than 23 will be encoded on a single byte. For larger values, the length must be added.

Question 6.8.5 page 80 In CBOR, an array can contain objects of different types.

- **True**
- False
- It depends on how this table was declared.

True, we have the same flexibility as in JSON by nesting any data type in an array.

Question 6.8.6 page 80 We want to define an array of two elements as a fraction. What tag should precede the structure ? (you can use the RFC 8949).

This is tag 4, see the chapter **3.4.4. Decimal Fractions and Bigfloats** of the RFC 8949.

Question 7.1.1 page 85 Why doesn't the client program work ?

- The IP address is not correct.
- **The data serialization process is missing.**
- The variable `t` is not defined.
- The variable `t` cannot be read.

Question 7.2.1 page 87 Let us analyze the received sequence : `83fb40341086f3e8b66bfb408f3b7791c8d61ffb403fa15ba06088e`

What does the byte 0x83 that starts the received CBOR structure correspond to ?

- to the coding of the positive integer 131.
- to the coding of the negative integer 132.
- **to the definition of an array of 3 elements.**
- the definition of a CBOR map of 3 elements.
- to the definition of an array of undefined size.

0x83 is written in binary `100-0 0111`. `100` is the major type for an array. The value `00111` is lower than 24. It is thus the number of elements of the array, thus an array of 3 elements.

Question 7.2.2 page 87 In this structure, what is the CBOR marker (in hexadecimal) that indicates that we have a floating number ?

0xFB, if we write it in binary we obtain `111-1 1100`. The major corresponds to the category of floats and special values.

Question 7.2.3 page 87 What is the size of this floating number in bytes ?

8 bytes ; the minor part `1 1100` indicates that it is a float coded on 8 bytes

Question 7.2.4 page 88 What is the minimum and maximum size of the CBOR structure sent,

taking into account the possible values.

— The temperature can reasonably be between -30 and +50, that is to say -3000 and +5000 after the transformation into an integer. The minimum size, if we send 0, the size will be one byte. The maximum size is 3 bytes (1 for the type/length and 2 for the values)
— The pressure evolves around 1000, that is 100 000 after the transformation to integer. The size will always be 5 bytes (1 for the type/length, 4 for the values)
— The moisture content evolves between 0 and 100 , or 0 and 10 000 after the transformation in integer. The size is between 1 byte and 3 bytes.

If we add the type/length 0x83 to indicate an array of three elements, we obtain a minimum size of 1+1+5+1= 8 bytes and a maximum size of 1+3+5+3 = 12 bytes.

Question 8.6.1 page 99 What does the key `{'u' : 'Cel'}` found in the previous structure correspond to ?

Unit = degrees Celcius

Question 8.6.2 page 99 In both JSON and CBOR representations, how much is the size increased by adding the measurements made ? Where do these differences come from ?

If we look at the previous listing, adding the three measures increases the size by 164 bytes for JSON and 106 for CBOR. The difference comes from the use of numbers rather than strings for the keys. Thus 't' requires 3 characters in JSON with the quotes, encoded in CBOR, it would take 2 bytes, a number less than 23 is encoded on a single byte. There are also commas, spaces and closing brackets which are not present in CBOR. The floating numbers like `19.98` have a more compact representation in JSON (5 bytes) than in CBOR where they consume 9 bytes. In all cases the increase is strongly dependent on the name of the elements. Here, it is necessary to repeat each time `temperature`, `humidity` and `pressure`, that is 26 characters.

Question 8.6.3 page 99 If we were only interested in one quantity, for example humidity. What would the SenML structure look like in JSON ?

Each new entry adds 35 bytes to the structure :
[{'bn' : 'device1', 'bt' : 1640110457.0, 'n' : 'humidty', 'u' : '%RH', 'v' : 28.46},
{'t' : 10.0, 'v' : 26.86},
{'t' : 20.0, 'v' : 26.96},
{'t' : 30.0, 'v' : 27.01}]

Question 8.6.4 page 101 Could we use the SenML field *base value* to decrease the size of the air pressure data ?

This would be possible, if this resource was sent alone.

Question 9.5.1 page 109 The I2C module of LoPy has a function `scan` which displays the addresses of the connected secondaries. How is this detection possible ?

The primary will test all possible addresses and send a frame, if the bit following the address in the

frame is not set by the secondary, there is no equipment connected to that address.

Question 9.5.2 page 109 Is the standard an address that allows you to talk to all the secondaries at the same time ?

The standard provides for a General Call to reach all the secondaries using address 0 (see chapter 3.1.12 of the I2C standard).

Question 9.6.1 page 112 What happens if in the program `wifi_temperature.py` you modify the measurement step line 36, to put it for example at 60 seconds.

The program `display`*`server.py`* must also be modified by adding the argument `period=60` to the call of `to_bbt`.

Question 11.2.1 page 137 What trace message do you get from the LNS, if the Object is not configured with the same IndexAppKey as the LNS ?

```
   12:52:11 Join-request to cluster-local Join Server failed MIC mismatch
12:52:01 Join-request to cluster-local Join Server failed MIC mismatch
12:51:51 Join-request to cluster-local Join Server failed MIC mismatch
```

Question 11.2.2 page 137 What impact on batteries ?

The LoPy will emit periodically every 10 seconds a message of type **Join-request**. The goal is to be able to connect, even if the transmission is not optimal. But in case of bad configuration, it will result in a higher occupation of the network, but especially an impact on the autonomy of the Object.

Question 11.2.3 page 137 In the previous traces, what is the value of the default fPort used by the LoPy ?

```
"f_port": 2,
```

Question 11.2.4 page 137 The instruction `bind` allows to modify the **fPort** for a transmission. Modify the program to use fPort 10 and check the effect in the TTN traces.

The program must be modified, for example, after the instructions `setsockopt` by adding the line : `s.bind(10)`

Question 11.5.1 page 147 With the help of the following link `https://www.thethingsnetwork.org/docs/lorawan/regional-parameters/` indicating the regional parameters. Would the program `lorawan_temperature.py` work on a LoRaWAN network located in North America ?

No, the maximum data size for DR0 is 11 bytes.

Question 12.2.1 page 158 What does this message represent ?

- A GET request

- **A POST request**
- A PUT request
- A DELETE request
- A positive notification

The code field (second byte of the CoAP header) is 0x02, or 0.02 so it is a request and a POST.

Question 12.2.2 page 158 What is the value of the token field ?

- **Empty**
- `0xb4`
- `0xb474`
- `0xb47465`
- `0xb474656d`

The first byte `0x40` is written in binary `0b01_00_0000`, that is to say the version (1), the type (CON) and the size of the Token field (0), so there is no Token after the obligatory header, there will be directly the options or the separator `0xFF` to indicate the data

Question 12.2.3 page 159 What elements of the URI does this message contain ?

- Aucun
- `/temp`
- `/temp/sens1` et `/max`
- `/temp/sens1/max`
- `/temp/sens1?max`

The sequence of Type/Length is `0xb4` Uri-path with 4 bytes of data (temp), `0x05` always Uri-path with 5 bytes of data (sense1) and `0x43` thus a type 11+4=15 that is to say Uri-Query with 3 bytes of data (max). `0xff` indicates the end of the options.

Question 12.2.4 page 159 You have the following resource :

`temperature = 20C`

What value should the CoAP Content-type option use for a CoAP response ?

- text/plain
- **0**
- 50

The integer 0 which corresponds to an encoding using the ASCII characters

Question 12.2.5 page 159 You want to receive a resource in the SenML ; CBOR format. What value should the Accept option in the request carry ?

`112`, as shown in the table reftab-CoAP-MIME

Question 12.2.6 page 160 What error code does the server return if it cannot send a response in this format ? Help you with the RFC 7252.

- 4.04 (Not Found)
- 4.02 (Bad Option)

- **4.06 (Not Acceptable)**
- 5.01 (Not Implemented))

See Chapter 5.10.4. of the RFC. 4.06 is not easy to find. You have to go to the IANA site, go to the CoAP RFC which points to the HTTP RFC for the definition of this code which is rarely used in HTTP. 4.04 is not possible because the resource exists but not in the right format. 4.02 is also not possible, the Accept option is critical and must be known by the server. Finally, this is a client error and not a server error, so error 5.01 is not possible either.

Question 13.2.1 page 170 What happens if you use a non-confirmable request to ask for the /time resource (put the argument `type=CoAP.NON` in the mandatory header construction).

- The server crashes.
- The server responds with a Confirmable request.
- The server returns an RST because we have not programmed this case.
- **The server responds with an Unconfirmable request.**
- We switch to winter time.

Question 13.2.2 page 171 Modify the server program to remove the 5 second delay before a response.

What happens when the client sends a CONfirmable request ?

- **The server returns the answer in the acknowledgement.**
- The server returns a NON confirmable request.
- The server still waits a few seconds to avoid saturating the network.
- The server removes the token from the response.

Question 13.2.3 page 171 Modify the server program to remove the 5 second delay before a response.

What happens when the client sends a NON confirmable request ?

- The server returns the answer in the acknowledgement.
- **The server returns a NON confirmable request.**
- The server still waits a few seconds to avoid saturating the network.
- The server removes the token from the response.

Question 13.3.1 page 175 What do we receive in response to the POST request ?

- an ACK message
- status 2.00 OK.
- **status 2.04 CHANGED.**
- nothing.

At each request we receive a REST notification indicating that the resource has been modified.

Question 13.3.2 page 175 Modify the client program to specify a JSON content-format. What notification do you get ?

- an RST message
- le statut 4.04 NOT FOUND.
- **le statut 2.15.**
- statut 5.00.

Question 14.3.1 page 183 What is the nature of the request issued by the client :

- GET
- **POST**
- PUT
- DELETE

The LwM2M client acts as a REST client and sends this POST request :

```
Constrained Application Protocol, Confirmable, POST, MID:12142
01.. .... = Version: 1
..00 .... = Type: Confirmable (0)
.... 1000 = Token Length: 8
Code: POST (2)
Message ID: 12142
```

Question 14.3.2 page 184 What is the path of the URI :

- empty
- **/rd**
- /lwm2m

The URI path is composed of a single element :

```
Opt Name: 1: Uri-Path: rd
```

Question 14.3.3 page 184 What is the format of the content (content-format) :

- text
- XML
- JSON
- **link-format**
- CBOR

After the URI path we find the Content-Format option which contains the value 40 or **link-format** :

```
Opt Name: 2: Content-Format: application/link-format
```

Question 14.3.4 page 185 What is the period in which you see CoAP messages ?

Figure 14.3 on page 184 shows several exchanges. The first one at 25.16 seconds is a POST to /rd. The second at 78.33 seconds is a POST to /rd/dyUNwRXNOp. The third at 131.37 seconds is identical and the following ones are identical. The sending period is 53 seconds.

Question 14.3.5 page 185 What does b=U mean ?

- The data will be sent with the units
- The reference of the units is the Universal representation
- **The underlying protocol is in datagram mode (UDP)**
- The client is not referenced (Unreferenced)

This parameter designates the binding, U indicates that CoAP will be transported over UDP. The standard also offers other modes, such as :

— T : TCP. CoAP also provides a mode of operation over **TCP** (RFC 8323). The mode allows for example to cross networks which restrict the use of IndexUDP for so-called security reasons ;
— S : SMS. LwM2M is defined by the Open Mobile Alliance, it is logical to include this means of communication to manage an equipment connected to the cellular network ;
— N : **NON-IP**. In this mode, CoAP messages are sent directly on level 2. It is also known as Non IP Data Delivery (NIDD) in 4G networks.
— We are based on version 1.1 of the standard, revision 1.2 of LwM2M also integrates LoRaWAN, MQTT (M), HTTP (H),...

Question 14.3.6 page 185 What does lwm2m=1.1 mean ? It is a question of...

- **the lwm2m client version**
- the memory size of the implementation (1.1 kb)
- the lwm2m server version

This is the client's version.

Question 14.3.7 page 185 What does lt=300 mean ?

- This is the maximum number of objects (less than 300)
- This is the maximum size of an exchange (300 bytes)
- **This is the lifetime of an object's registration**

This is the lifetime of a value, if it is not refreshed before this time by the client, it will disappear from the server.

Question 14.3.8 page 186 What is the purpose of the rt attribute ?

- **to give the semantics (i.e. how to interpret) of the resource.**
- to indicate the size of the resource with reference to lwm2m.
- to indicate the version of lwm2m used.
- to help debug the exchanges.

When the server receives the POST from the client, it is able to associate a representation with the described URI path. Both the client and the server must have this knowledge a priori.

Question 14.3.9 page 190 Using the digital object and resource description page [1], what does the URI *3301/0/5602* represent ?

- the maximum temperature of the first sensor
- the minimum humidity of the sensor 0
- **the maximum brightness of the sensor 0**

Going to the web page, we find that *texttt3301* corresponds to the luminosity measurement *Illuminance*). The description of this numerical object is identical to that of the numerical object *temperature* ; *texttt3301/3/5602* indicates the maximum value.

Question 14.3.10 page 190 What does the URI *texttt10340/0/2* represent ?

- lhe temperature in Fahrenheit of the object 10340
- longitude in GPS coordinates

1. `http://www.openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html#resources`

- **the active or not status of a camera**

By going on the web page, we find that *10340* corresponds to a digital object *Camera* defined by the company CloudMinds. The description of this digital object shows that the resource *2* corresponds to the status (*1 :Enabled, 0 :Disabled.*)

Question 14.3.11 page 190 Which URI allows access to the element controlling the light variation (*Dimmer*) of a lighting (*Light Control*) ? What is the maximum value this resource can take ?

3311/0/5851 The maximum value is 100.

Question 14.3.12 page 190 What would be the URI path to obtain the frequency of an electric current on the second phase ?

3318/1/5700

Question 14.3.13 page 190 In the first captured exchange :

```
</>;ct="60 110 112 11542 11543";rt="oma.lwm2m",
</1>;ver=1.1,
</1/0>,
</3>;ver=1.2,
</3/0>,
</6/0>,
</3303>;ver=1.1,
</3303/0>,
</3442/0>
```

1 : Device
3 : LwM2M Server
6 : Location
3303 : Temperature
3442 : LwM2M v1.1 Test Object

Question 14.4.1 page 191 What does the server answer (see figure 14.6 on page 192) ?

- It just acknowledges the message.
-
-
-

The response contains the option ***Location-Path*** which makes it possible to specify where the resource provided by the client was referenced on the server. As for ***URI-Path***, it is an option which can be repeated. In the example, figure 14.6 on page 192, the URI provided is therefore `/rd/dyUNwRXNOp`.

This URI will then be used as an internal identifier for the client.

Question 14.4.2 page 191 If we let the platform run without intervention, what do we see on

the network analyzer ?

- Nothing.
- The sensor sends information indicating a change in the measured value.
- The customer sends the measured values even if they have not changed.
- **The client sends an empty message to the server to indicate that it is still present.**
- The server sends an empty message to its clients to see if they are still present.

Every 53 seconds, the client sends a POST to the resource created at registration. This POST is empty. It is used to tell the server that the server is still present. The server acknowledges this message indicating to the client that it is also accessible.

Question 14.5.1 page 195 A click on the eye with the cross allows the server to cancel the Observe. What message is sent ?

When the LwM2M server receives an Observe for a session that has been cancelled, it responds with a message of type *ReSeT*, the value of the *Message ID* field lets the LwM2M client know which Observe session to cancel.

Question 14.5.2 page 195 What message is emitted, if an EXE resource is clicked, as resetting the min and max values ?

This does not change the protocol, a POST on the URI of the resource is issued by the LwM2M server.

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Bibliography

[TM03] Andreas TOLK et James A MUGUIRA. “The levels of conceptual interoperability model”. In : *Proceedings of the 2003 fall simulation interoperability workshop*. Tome 7. Citeseer. 2003, pages 1-11 (cf. page 22).