Internet of Things

Niccoló Didoni

March 2022

Contents

Part I

Introduction

Chapter 1

Basic concepts

1.1 Definitions

The Internet of Things (IOT) is getting more and more visibility nowadays, thus it's important to understand how it's developed and what are the basic protocol and paradigms it uses. Before diving into the details, let us introduce some definitions.

Definition 1 (Internet of Things)**.** *A global network infrastructure, linking physical and virtual objects through the exploitation of data capture and communication capabilities. This infrastructure includes existing and evolving Internet and network developments.*

Definition 2 (Internet of Things)**.** *A world where physical objects are seamlessly integrated into the information network, and where the physical objects can become active participants in business processes..*

Definition 3 (Internet of Things)**.** *A world-wide network of uniquely addressable interconnected objects, based on standard communication*

All these definitions focus on

- **Smart objects**, that is devices that are able to send, sense, store and process information. Smart objects are the very foundation of Internet of Things.
- **Data**. Data is the real value of IOT systems, in fact these systems are made to exchange and process large volumes of data.
- **Pervasiveness**. IOT systems are made of a big number of smart objects.
- **Seamless communication**. This characteristic focuses on the service that a system provides but not on the technology with which it's implemented. In other words the implementation of the system has to be transparent to the user.

1.1.1 Environment and heterogeneity

IOT systems can be applied to several environments and applications that have different types of requirements, thus we need different types of software technologies that can be applied to each application.

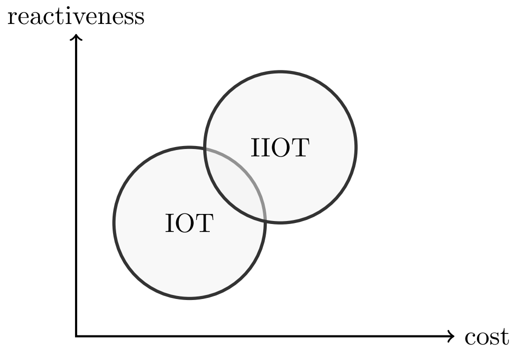


Figure 1.1: A representation of IOT types plotted on a reactiveness-cost graph.

1.1.2 Classification of IOT systems

IOT systems can be divided in

- **Consumer IOT** (or simply IOT). Consumer IOT systems are related to to end users and must have a limited cost (because users have to be able to buy them). These systems have lower requirements with respect to delay (i.e. they have to be less reactive).
- **Industrial IOT** (IIOT). Industrial IOT systems are used in industrial environment, thus they can be expensive (companies usually can invest more). On the other hand such systems have to ensure an high reactiveness (i.e. small delay). These systems are usually in a control loop (data is used to provide feedback and adjust a parameter, like a thermostat that manages a room's temperature).

Some systems can have some characteristics of both types. A representation of the types of IOT systems is shown Figure 1.1.

1.2 Components

An IOT system can be divided in four vertical components

- **Things**.
- **Connectivity**.
- **Data**.
- **Analysis**.

Such components must be present in whatever IOT system, independently from the application or the environment. In other words, these components define an IOT system. Things, connectivity, data and analysis are grouped in a software component called **platform**. A platform hides the complexity of the vertical components and represents a control software. Basically, platforms have the same job of operating systems that hide the hardware complexity of a computer. Two extra layers are applied on the four main components

- **Security**. Security is very important in IOT because

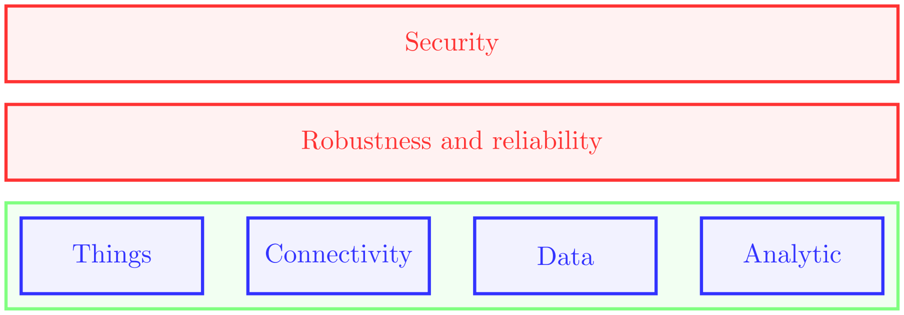


Figure 1.2: Components and requirements of an IOT system.

  - IOT systems generate a lot of data that has to be protected (i.e. we have to ensure privacy). In particular, the data has to be seen only by those authorised.
  - Data has to be transmitted and stored, thus we have to consider classical security methods like encryption.

- **Robustness and reliability**. Robustness and reliability are also fundamental in IOT systems, in particular we have to consider
  - The hardware level, in fact sensors can be in harsh environments, thus they have to cope with an high amount of stress.
  - The software level, in fact we have to ensure that the protocols works even if some failures occur (e.g. we have to handle re-routing if a link goes down).

These layers are requirements, not components of the system. Figure 1.2 shows the components and the requirement layers of an IOT system.

Another important requirement that should be taken into consideration is **network integration**, i.e. the capability of a new smart object to integrate with the preexisting network.

1.2.1 Components' challenges

Each of the four components has different challenges to face.

Things

Things (i.e. field devices, sensors, edge nodes) have to face many challenges,

- **Cost reduction**. IOT systems are made of a large number of edge nodes, thus it's important to keep the cost of a single component low, to decrease (by a lot) the cost of the whole system.
- **Miniaturisation**. Edge nodes should be small to be used in all types of environment and conditions.
- **Energy efficiency**. Usually smart nodes run on batteries, thus we have to ensure that they live as long as possible.
- **Resilience and robustness** as it is one of the requirements of an IOT system (Figure 1.2).

- **Shape and design**. The shape and the design of a smart thing is very important if it has to interact directly with a human. A badly designed thing may not work even if the system itself works well, because the user can't correctly interact with it. When we talk about human interaction we refer to the **Human and Machine Interaction**, thus we have to design a good Human and Machine Interface.

Connectivity

Connectivity poses us in front of two main challenges,

- Classical protocols for network communication can't be used in IOT because smart things are not routers. This means that we have to develop and design new protocols for IOT.
- The technology used to communicate has to be taken into consideration because smart things have to be energy efficient, thus we have to use technologies that don't use too much power.

Data and analytic

The main challenge of data and analytic is the fact that **the system should scale up easily**. In particular, IOT systems collect huge amounts of data, thus we have to find efficient ways to store (how and where) and process it.

Part II

Things

Chapter 2

Sensor nodes

Let's start our analysis of IOT systems from the edge. In particular, we will consider all those things that do some sensing and that operate on the environment. Such elements are called **smart objects**, **sensor nodes** or **motes**.

Definition 4 (Mote)**.** *A sensor node, or mote, is an device that senses, process locally, stores locally and communicates with other nodes.*

Some motes can also act on the environment they are in. In this case we talk about **actuators**. Sensors and actuators aren't exclusive, in fact a node can be

- Only a sensor.
- Only an actuator.
- Both an actuator and a sensor.

2.1 Sensing

Let's start by analysing the sensing part of a sensor. Choosing the right sensor for a problem is very important, thus let us define what we need to know when picking a sensor.

- **What we need to sense**. We should be able to list all the phenomena that the sensor has to perceive to design (or buy) appropriate components. For instance, if we want to measure the temperature of a room we have to find the sensor that fits best with the temperature measured in the room.
- The **constraints of the system**. In particular we have to evaluate the constraints with respect to
  - Connectivity. In particular choosing between wireless and wired connectivity is very important because it the latter case we have to care less about energy efficiency because the same cable that brings connectivity can also bring power. In case of wireless connectivity, a device usually runs on batteries, thus it's important to choose energy efficient technologies.
  - Flexibility and programmability.

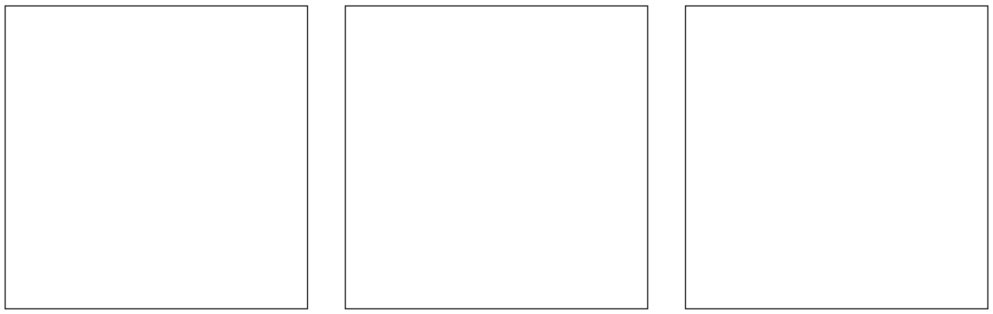


Figure 2.1: A mote

- Cost.
- Human and Machine Interface.
- Integration, size and design.

2.2 Structure of a mote

A sensor node is divided in three subsystems (or modules)

- A **processor** subsystem.
- A **sensor/actuator** subsystem.
- A **communication** subsystem.

These modules can communicate with serial interfaces (e.g. SPI and I^2C). An schematic representation of a mote is shown in Figure 2.1.

2.2.1 Processor

The processor subsystem handles processing and, optionally, data storage in a flash memory. A processor is usually designed according to the SHARC (Super Harvard Architecture Single-Chip Computer) architecture. According to the SHARC architecture, which is an evolution of the Von-Neumann architecture, a processor should be made of

- A **cache**.
- A **program memory** to store the software that has to be executed.
- A **data memory** to store the volatile data generated and used by the program.
- An **input/output controller**.
- A **processor**.

Each component can be designed in different ways and there exist many different solutions that implement it. We are going to analyse only some solutions, in particular

- **Microcontrollers**.

- **Digital Signal Processors** (DSPs).
- **Application Specific Integrated Circuits** (ASICs).
- **Field Programmable Gate Arrays** (FPGAs).

Microcontrollers

A microcontroller is a single Integrated Circuit (IC) used for a specific application. In particular a microcontroller has a CPU that is characterised by the size of the registers it uses (usually CPUs go from 4-bit to 64-bit architectures). Together with a CPU, a microcontroller has a clock generator to synchronise operations.

Memory and storage Microcontrollers have

- A **RAM memory** to store volatile data.
- A **flash memory** to store non-volatile data. Usually this memory has block access only (i.e. data can't be access in bytes but only in larger sizes).
- **EEPROM** (Electronic Erasable Programmable Read-Only Memory) memory that allows read-only accesses. In this case we have byte-access.

Module communication Microcontrollers have several busses for communicating with other modules. Usually serial interfaces like SPI or I^2C are used. Microcontrollers can also include ADCs and DACs to transform data from and to the other modules.

Advantages and disadvantages The main advantages of microcontrollers are

- **Cheapness**.
- **Flexibility**, in fact the CPU can run different programs without being modified.

The most important disadvantage is **performance**, in fact microcontrollers lack the speed of other devices. This is the price we have to pay for flexibility.

Digital signal processors

A Digital Signal Processor (DSP) is a microprocessor specialised in discrete signal processing. For this reason, DSP can carry out data intensive operations in almost real time (DSP can process hundreds of millions of samples per second).

Advantages and disadvantages The main advantage of DSPs is related to **performance**, in fact these motes can perform data intensive task at a staggering speed. As always, the price we have to pay for such speed is **low flexibility**, which is in fact the main disadvantage of DSPs.

Application Specific Integrated Circuits

An Application Specific Integrated Circuit (ASIC) is an integrated circuit designed to run application specific instructions.

Advantages and disadvantages Running only specific (i.e. custom made for an application) instructions boosts performance, allowing ASICs to reach good speed. Having specific instructions also means that ASICs are not flexible. In particular, it's possible to reprogram an ASIC but only after switching off the system. Furthermore, ASICs are also very expensive, thus they are rarely used in common-purpose systems.

Field Programmable Gate Arrays

A Field Programmable Gate Array (FPGA) is very similar to an ASIC. FPGAs are more high-level.

Advantages and disadvantages Since FPGAs are similar to ASICs, they share the same advantages and disadvantages. Namely, FPGAs have very high performance but lack in flexibility, even though it's easier to reprogram an FPGA that an ASIC. That being said, FPGAs are more expensive than ASICs.

2.2.2 Sensor and actuator

The sensor/actuator subsystem contains the sensing peripherals that produce data. The components in this subsystem (i.e. the sensors) depend on the phenomenon that has to be sensed. Signal conversion is an important problem for this subsystem. In particular, sensors usually produce an analogical output that has to be converted in bits (i.e. digital world). This task is executed by a Analog-Digital Converter (ADC). The ADC can be

- Shared among all sensors.
- Dedicated for each sensor.

If the output data isn't converted by the sensor subsystem, then the processor module should have ADCs and DACs (Digital-Analog Converters) to handle signal conversion.

Connection with the processor The sensing subsystem is connected to the processor via some interfaces, usually realised as serial busses (i.e. busses in which information is pushed in a serial way). The two most common interfaces are SPI and I^{2}C.

Analog-Digital Converters

ADCs allows the mote to move from the time (i.e. continuous domain) to the digital (i.e. discrete) domain. An ADC is essentially made of two blocks

- A **sampler**. A sampler is an integrated circuit responsible for discretising the incoming signal working on the time axis.
- A **quantiser**. After the input, time-continuous signal, passes through the sampler, we obtain a discrete signal with respect to the time axes (i.e. we obtain some samples at specific time instances instead of all samples at all time instances). However we still have a continuous amplitude (y axis). The quantiser discretises also the amplitude of the signal and outputs a sequence of bits (i.e. a time discrete, amplitude discrete signal).

Usually ADCs also contain filters before the sampler to prepare the incoming signal and after the quantisers.

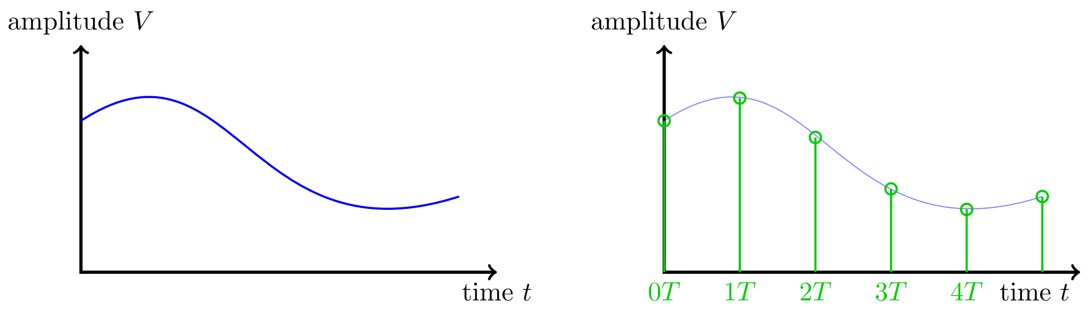
Sampler

The goal of a sampler is to **take a time continuous signal (commonly a voltage signal) and read only some samples (i.e. values of the signal) at discrete time instances**. The samples are commonly taken at a specific frequency (i.e. at specific time instances separated by time interval). If the time interval is T then we sample the input signal at T, $2T$, ..., nT. The inverse of the sampling interval T is called sampling rate $f_s = \frac{1}{T}$ and is the frequency at which we take the samples. An example is shown in Figure 2.2.

When we sample the input signal, we are losing some information because we are reading only some of the input values, however the Nyquist theorem says that

Theorem 1 (Nyquist)**.** *If the sampling frequency is at least double the signal bandwidth B, then the sampling process is error free and the output signal contains all the information of the input signal (i.e. we are able to go back to the original signal from the samples).*

In some cases, ADCs use a frequency that is higher than the Nyquist frequency for redundancy reasons (i.e. to make samples more robust in case of errors).



(a) Input signal of the sampler. (b) Output signal of the sampler.

Figure 2.2: Signal before and after passing through a sampler.

Quantisers

After passing through a sampler, the values of each samples can take any continuous value. However bits can't represent continuous values, hence the quantiser has to **take as input a continuous voltage value and associate a digital output (i.e. a string of bits) to it**. A quantiser also takes as input the scale of the input value, i.e. the values allowed for the input voltage (e.g. from -1 V to 1 V). In other words, the output of the quantiser is a digital representation of the input voltage.

In particular, a quantiser

1. Takes the possible values of the voltage that can be put in input (i.e. the input scale V_{fs}).
2. Divides the input scale in discrete intervals.
3. Assigns a codeword (i.e. a string of bits) to each interval.
4. Assigns a codeword c_k to a sample if the value of the sample is in the interval i_k.

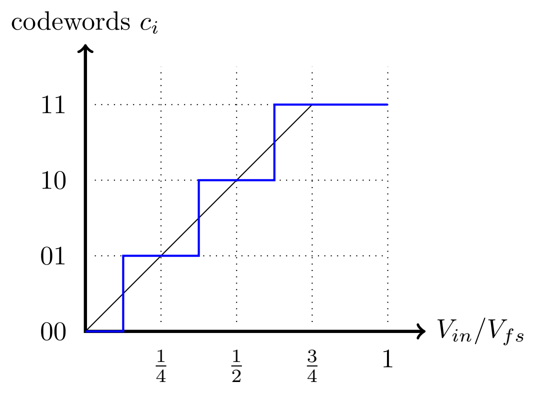


Figure 2.3: Quantisation.

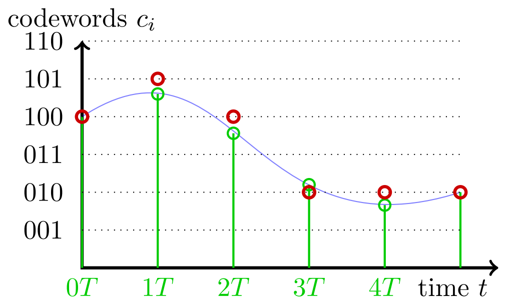


Figure 2.4: Quantisation of signal 2.2b (output of the sampler).

A visual representation of this process is shown in Figure 2.3.

The number of discrete intervals depends on how many bits we want to use for the codeword (i.e. how many bits we use to encode a sample). If we use n bits than we have 2^n intervals. Knowing the number of intervals we can define the resolution LSB of the quantiser as the ratio between the input scale and the number of intervals.

$$LSB = \frac{V_{fs}}{2^n}$$

Basically, the resolution says how wide an discrete interval is. A low resolution means that we are using a lot of intervals, hence each interval is thin.

Thanks to the Theorem 1 we can have no errors during sampling, however we can still have errors in the quantisation phase. This phenomenon is visible in Figure 2.4 where we can see that the red dots only approximate the green ones. This error comes from the behaviour of the quantiser, in fact if we check Figure 2.5 we can see that voltage is green is overestimated because we are assigning a voltage level higher than the actual level. On the other hand the voltage in blue is underestimated. The error depends on the size of the interval (the wider, the biggest the error). Since the resolution depends on the number of levels, hence on the number of bits n of the codewords, then the quantisation error also depends on n. In particular the errors is a monotonic decreasing function of n (the biggest n the lower the error). However, increasing too much the number of quantisation levels (i.e. of number of bits) has some disadvantages,

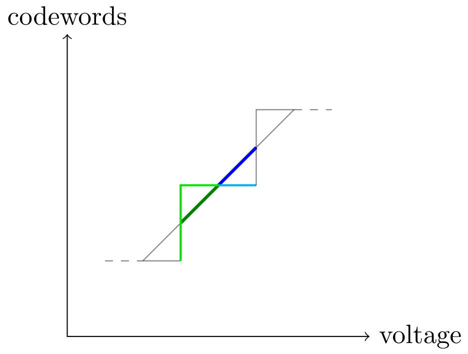


Figure 2.5: Quantisation error.

- With more levels the sensor is more sensitive to noise.
- We generate a lot of data (i.e. the ADC has an higher throughput).

Variable interval quantisers The quantiser we have described is the most basic one. In practice, ADCs use quantisers whose voltage ranges have different width because some values of voltage are more probable to be seen in input. Basically, the intervals are smaller for more probable voltage levels so that we obtain smaller errors for more probables voltage values.

2.2.3 Communication

The communication module sends and receives bits. This modules can include different interfaces that allow to use different communication technologies (e.g. Bluetooth and Wi-Fi). Usually each interface has it's own co-processor that allows to decode and encode data.

2.3 Hardware choice

In the IoT field there exists multiple type of motes, with different characteristics. To evaluate the best device for a problem or application we can evaluate three parameters

- **Cost**.
- **Power**.
- **Flexibility**. Flexibility evaluates how easy it is to support different types of services.

In recent years, OS-based boards are becoming more popular. These boards have an operating system that runs on top of the hardware. The services and applications of the mote run on top of the OS. This trend is in contrast with non-OS boards in which the application is directly run on top of the hardware. OS-based boards offer a lot of flexibility because the operating system abstracts the hardware interface and allows to run different applications and services. However, having the OS reduces performance in terms of throughput, latency and responsiveness because the hardware has to run also the OS.

Chapter 3

Energy consumption

Sensor nodes usually run on batteries, thus increasing energy efficiency and power consumption is vital to reduce the number of times the battery has to be changed. Changing batter is, in fact, a very expensive operation and in some case it might also be impossible. Batteries can be divided in

- **Primary batteries**, which are not rechargeable.
- **Secondary batteries**, which are rechargeable thus should be used in combination with some form of energy harvesting (e.g. solar panels, vibration harvesting).

An important parameter to consider when talking about energy consumption in a mote is the lifetime. This parameter tells how much time the sensor can run without changing the battery assuming average use.

3.1 Energy consumption for communication

If we consider the sources of power consumption of a wireless mote (sensors, CPU, radio transmission, radio receiving, radio idling and sleep time) the communication part (transmission, receiving and idling) is the most power hungry. Moreover, for short range radio communication, the power consumption is almost identical whether the radio is receiving, transmitting or simply listening for incoming messages. Since the mote doesn't know when a message might arrive, it has to keep listening for incoming messages which aren't very frequent. This means that a mote passes the most of the time listening for incoming messages (which consumes a lot of energy). Another important thing to remember is that, on average, the communication subsystem has comparable power consumption whether its receiving, transmitting or simply idling. This means that, even when the communication system isn't doing anything, it still consumes the same power as when it's sending or receiving messages. Everything we've said lead us to say that the most important thing in IoT communication is to reduce the time the communication subsystem listens to incoming messages.

3.1.1 Duty cycles

To solve the idle listening problem, mote usually apply the duty cycle model. In particular the radio component is not active for most of the time and at regular intervals it's waken up to allow radio communication. In other words, a mote continuously repeats the following instructions

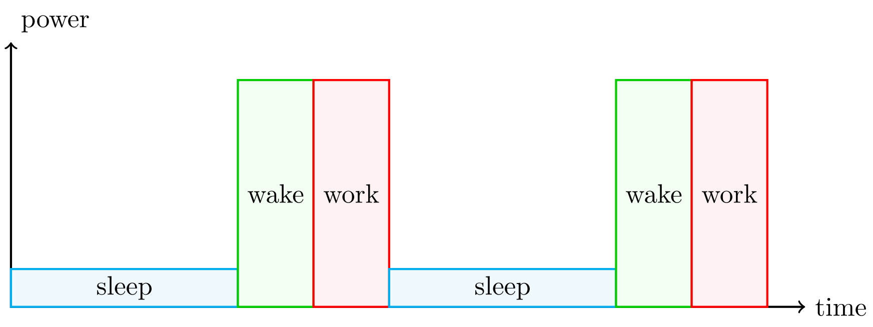


Figure 3.1: Duty cycles of a mote.

1. Wake up the radio component.
2. Send and receive messages for a small period of time.
3. Turn off the radio component.

This allows a mote's communication component to sleep most of the time and being active only for a limited amount of time. Since the communication component is the most power hungry, this technique can reduce power consumption and the longer the sleep time is, the more energy is saved. A representation of some duty cycles is shown in Figure 3.1.

To support duty cycles, most motes are equipped with wake-up radios, i.e. radios that can be dynamically switched on and off by the processor.

Now that we have understood how duty cycles work, we can compute the power consumed by the communication component of a mote. In particular we can distinguish between

- **Peak power** P_{active}. Peak power is the power used when the mote is using the radio. In motes, this power is in the order of milliwatts.
- **Sleep power** P_{sleep}. Sleep power is the power consumed when the communication component is sleeping. This power is very limited (microwatts in motes) and basically comes from minimal running components and leakages.

Knowing the frequency f_{active} of the active part the duty cycle (i.e. what fraction of the cycle is used for listening and receiving messages) we can compute the average power consumed as

$$P_{avg} = (1 - f_{active}) \cdot P_{sleep} + f_{active} \cdot P_{active}$$

More precisely, if we know the fraction of cycle used for each activity (sleep, wake up and work) we can compute P_{avg} as

$$P_{avg} = f_{sleep} \cdot P_{sleep} + f_{wakeup} \cdot P_{wakeup} + f_{work} \cdot P_{work}$$

A common value for f_{active} (the node is active during wake-up, transmission and receiving) is 1 % of the total duty cycle time. The power consumed can be used to compute the lifetime of a mote

$$t = \frac{E_{battery}}{P_{avg} - P_{generated}}$$

where

- $E_{battery}$ is the initial energy provided by the battery.
- P_{avg} is the average power consumed by the mote.
- $P_{generated}$ is the power obtained from energy harvesting.

As for now we have understood that the active part of a duty cycle is the most power hungry, thus we should focus on it. Typically, during an active stage the mote can receive or transmit data, thus it's useful to compute the energy consumed during transmission and reception.

Transmission

The energy consumed during transmission is the sum of the energy used to keep the radio circuit active and the energy used to power the antenna and send a radio signal

$$\begin{aligned} E_{tx} &= E_{circuit} + E_{send} \\ &= P_{tx}(T_{wakeup} + T_{tx}) + P_o \cdot T_{tx} \end{aligned}$$

where

- P_{tx} is the power consumed by transmitter .
- P_o is the output power of transmitter.
- T_{tx} is the time to transmit a packet.
- T_{wakeup} is the start-up time for the transmitter.

Receiving

When the mote receives some data, it doesn't have to power the antenna up, thus the energy consumption is only given by the energy needed to power up the circuit

$$E_{rx} = P_{rx}(T_{wakeup} + T_{rx})$$

where

- P_{rx} is the power consumed by receiver.
- T_{rx} is the time to receive a packet.
- T_{wakeup} is the start-up time for the receiver.

Notice that the duration for which the circuit has to be active are hardware-dependent parameters and their value is a design choice. For instance the duration of the transmission depends on the length of the packet to send, thus different protocols might require different uptimes.

Wake up and sleep time

Both in transmitting and receiving, we have to pay a price for turning on the device. This is however an overhead, in fact during T_{wakeup} the circuit isn't used (i.e. it doesn't send nor receive messages). Also notice that, even if it hasn't been presented, we should consider the power needed to switch off the circuit, too. The energy consumed in this phase is however negligible with respect to wake up and transmission or receiving, thus it can practically be omitted.

3.1.2 Energy consumption per bit

Knowing how much energy is consumed in transmission and reception is useful, but sometimes we would like to know how much energy has been consumed, with respect to the message sent or received. Basically, we want to know if we have consumed a lot of energy because we sent a lot of data or because the communication component is not efficient. We can use the energy per bit (sent or received) to check if we are in the former or latter scenario. In particular the energy per bit E_b is the ratio between the energy consumed while sending and the total number of bits L sent

$$\begin{aligned} E_b &= \frac{E_{tx,tot}}{L} \\ &= \frac{E_o + E_{wakeup} + E_{tx}}{L} \\ &= \frac{P_o \cdot T_{tx} + P_{tx}(T_{wakeup} + T_{tx})}{L} \end{aligned}$$

The same applies for receiving

$$\begin{aligned} E_b &= \frac{E_{rx,tot}}{L} \\ &= \frac{E_{wakeup} + E_{rx}}{L} \\ &= \frac{P_{rx}(T_{wakeup} + T_{rx})}{L} \end{aligned}$$

After obtaining the energy consumption per bit we can try to analyse what is the length that minimises power waste, in particular if we send just a few bits we pay a big overhead for waking up with respect to the power consumed for sending the packet. This means that we should increase the dimensions of a packet to reduce the energy wasted for waking up with respect to the sending part. The packet can't however be too big, in fact if we try to send a lot of bits

- The mote needs a big memory to buffer the packet.
- If a packet gets lost we have to resend a lot of data.
- We introduce a lot of delay because, before sending a packet, we have to wait the data generated by the sensor.

3.1.3 Tuning emitted power

Let us continue our travel through consumption optimisation by analysing how we can tune the emitted power P_o to reduce energy consumption without compromising transmission and reception. In particular we can decide to set P_o to the lowest value for good reception. Now the problem is: how do we measure good reception? We can use two different metrics

- The **Bit Error Rate** (BER) that represents the fraction of bits not correctly received.
- The **Packet Error Rate** (PER) that represents the fraction of packets not correctly received. Basically the PER works just like BER but considering packets instead of bits.

The relation between these performance measures depends on the channel and, in general we can say that

$$PER = 1 - (1 - BEP)^l$$

where l is the length of the packet. Basically,

- $(1 - BEP)$ is the probability that a bit is sent correctly.
- $(1 - BEP)^l$ is the probability that l bits (i.e. a packet) are sent correctly.
- $1 - (1 - BEP)^l$ is the probability that at least one bit is not correctly sent, that is that the packet is not correctly sent.

Notice that this dependency is valid if the probability of a bit being wrong is not related with the probability of the other bits of being wrong. This is however a very strong assumption in fact usually if a bit is wrong, the probability of the next bits being wrong is much higher.

Evaluating BER and PER Bit Error Rate and Packet Error Rate depend on many factors

- The context.
- The noise between sender and receiver. The noise can be divided in background noise (that can't be eliminated) and external noise of the link.
- The SINR, i.e. the ratio between the useful power that is arriving at the receiver's front-end and the noise.

  $$SINR = 10 \log_{10} \left(\frac{P_{rcv}}{N_0 + \sum_i I_i} \right)$$

  where

  - P_{rcv} is the power received.
  - N_0 is the thermal noise (i.e. the KTB).

Knowing the SINR and the characteristics of the channel we can compute BER and PER using theoretical or empirical models.

Sensitivity To understand if a communicating component is able to receive a signal we need to know the sensitivity P_{min} of the receiver. In particular the sensitivity is

Definition 5 (Sensitivity)**.** *The minimum input signal power needed at receiver input to provide an adequate SNR (Signal-to-Noise Ratio) at receiver output to do data demodulation.*

The sensitivity of a communication module is usually specified in the data-sheet that usually says what is the minimum power, at a certain frequency, to get a certain PER. For instance one could read

$$P_{min} > -85 \text{ dBm @ } 2.4 \text{ GHz for } PER < 1\%$$

Minimum emitting power Now that we know the minimum power that has to reach the receiver, we can compute the minimum power that has to be emitted by the sender in order to have a specific PER. To compute P_o we can characterise the communication channel with the following model

$$P_{rcv} = P_o \cdot g_t \cdot g_r \left(\frac{\lambda}{4\pi d} \right)^2$$

where

- P_{rcv} is the power received.
- P_o is the power emitted by the sender.
- d is the distance between sender and receiver.
- λ is the wavelength (which can be obtained from the frequency $\lambda = \frac{c}{f}$, with c speed of light).
- g_t and g_r are the gains of the sender and receiver, respectively.

If we assume unitary gains, the model becomes

$$P_{rcv} = P_o \left(\frac{\lambda}{4\pi d} \right)^2$$

Now we can obtain the minimum power that the sender has to emit

$$P_o = P_{min} \left(\frac{4\pi d}{\lambda} \right)^2$$

Notice that this value should be continuously computed because it might change. Consider for instance a mote that moves; in this situation we have to compute the P_o every time the distance d changes (intuitively when the sender gets further we have to increase the emitted power).

To sum things up, when we want to compute the emitted power we have to

1. Check the sensor's characteristics.
2. Check the link parameters.
3. Take the proper law to describe the communication channel.
4. Invert the law and obtain P_o.

3.2 Energy consumption for processing

Processing task aren't the most power hungry in a mote, especially if we compare them to communication, however it's useful to analyse the power emitted while computing. We can compute the power consumed by the CPU as the sum of

- The power P_{dyn} emitted to actually do some computation.
- The power P_{sc} dissipated by short circuits.
- The power P_{leak} caused by circuit leaks.

$$P_{cpu} = P_{dyn} + P_{sc} + P_{leak}$$

Basically, we are separating the power needed for the computation from the power dissipated by losses

$$P_{cpu} = P_{dyn} + P_{losses}$$

where

- P_{dyn} can be computed as
  $$P_{dyn} = CfV^2$$
  where
  - C is the **capacitance** of the CPU (usually around 0.67 nF).
  - f is the **frequency** (i.e. the clock rate) at which the CPU operates.
  - V is the **voltage** provided to the CPU.
- P_{losses} is a fixed value that depends on many factors like the architecture of the CPU, the subsystems mounted on the mote and the bus. Usually P_{losses} is much lower than P_{dyn}.

Notice that we can control only the power needed for computation, while the power due to losses is just a number that is fixed once we choose a specific architecture for the CPU. In particular, to reduce power consumption, we can

- Reduce the voltage.
- Reduce the frequency.

Even if these actions reduce the computing power and the performance (i.e. increase the processing time) of the processor, it's very common to downclock or reduce input voltage to improve power efficiency. This means that we have to find a trade-off between processing power and energy consumption.

Local filtering Radio communication consumes way more energy than processing, thus we can process data locally (sacrificing processing efficiency) to filter out useless data and create packets with compressed information in order to reduce radio usage. To put it into perspective, sending 1 kB of data at 100 meters consumes the same energy needed by a 100 MHz processor to execute 3 million instructions.

3.3 Energy consumption for memory and sensing

3.3.1 Memory

In general, the energy used by memory is negligible, especially if we compare it with one used by the communication modules. Usually, the closer the memory is to the CPU, the fewer energy it consumes. In motes, flash memories are the ones that consume more energy,

- 1 nAh per byte while reading.
- 83.3 nAh per byte while writing.

3.3.2 Sensing

Sensing usually doesn't consumes a lot of energy, if we exclude from our analysis multimedia sensors (e.g. cameras and microphones), which are very power hungry. In particular the energy consumption of a sensor depends

- Linearly, on the **sensing frequency** f_s.
- Exponentially, on the **resolution** (or quantiser) n, i.e. the number of bits used to encode each sample.

$$P_{sensor} \sim f_s \cdot 2^n$$

3.4 Design guidelines

As we have stressed out, energy consumption is the issue of motes, thus we have to do whatever possible to reduce it. The easiest ways to improve energy efficiency are

- Put to sleep whatever component isn't needed.
- Downscale the components (the bigger the components, the more the power emitted).

These actions shouldn't be taken statically, in fact a mote should be able to dynamically allocate resources and change the duration of duty cycles to ensure it's functionalities, without consuming too much energy. Notice that the intelligence needed to control the mote depends on the technology (i.e. on the architecture) used.

Part III

Application level protocols

Chapter 4

Introduction

An important part of IOT systems is the communication between the devices in the system. In particular, the main devices in an IOT systems are

- **Wearable devices** like smart watches.
- **Environmental sensors**.
- **Domotics sensors**.
- **Vehicles**.

If we consider Industrial IOT, we can connect

- Higher Layer components among themselves.
- Controllers with higher layer components (MES, ERP, Data Bases, Data Lakes).
- Controllers with controllers (PLCs, SCADA).
- Sensors with controllers (PLCs, DCS).

Given large number of different devices, we need different technologies suited for different environments and constraints. In particular, to choose the best technology, we should consider

- The **environment**. In particular we should consider interference, the presence of moving objects, harsh conditions and mechanical stress.
- The **performance requirements**. Some examples are timeliness, determinism and reliability.

Moreover, we can't use the classical internet technologies because smart objects have lower capabilities and require a low power consumption.

Traffic An important part of communication is the traffic generated by the system's devices. In particular, we can identify four types of traffic

- **Cyclic traffic**. Cyclic traffic is a periodic traffic because motes collect data at periodic intervals. This means that devices indefinitely generate a small amount of data according to a period. The main requirement for cyclic traffic is determinism because we have to ensure that the periodicity is respected (e.g. if we have to ensure that 1 sample every second is sent, then we should provide such period).

- **Acyclic traffic**. Acyclic traffic is generated by a system that implements a conditional behaviour (e.g. if data has a property then send the data). Such systems generate (i.e. send) data in an unpredictable way (i.e. without a period). The main requirement for acyclic traffic is that data should be send as soon as it's generated.
- **Multimedia traffic**. Multimedia traffic is generated by multimedia (e.g. audio, video) sensors. The main requirement for multimedia traffic is bandwidth because the volume of data exchanged is very large.
- **Back-end traffic**. Back-end traffic is made of the aggregate traffic from different sources that send data to the same controller. The main requirement for back-end traffic is bandwidth because the volume of data exchanged is very large.

4.1 Networking introduction

A network is a set of hardware and software components that are used to move data from a source to a destination. A network is made of

- A **physical infrastructure**. The physical infrastructure is the set of hardware components that interconnect the devices of the system.
- A **network architecture**. The network architecture defines how hardware is interconnected (i.e. what is the topological structure of the devices).
- Some **communication protocols**. Communication protocols are software components that define how devices exchange data.

Whenever we want to define a network, we have to define these three components. Notice that, in IOT, we use the package-switch model, namely data is moved in packets.

4.1.1 Evaluation parameters

When we describe a network we should use

- The **throughput** (also, for our purposes, rate, bandwidth and capacity). The throughput is the number of bits (bytes) per second that can be sent across a link.
- The **delay**. The delay is the time that elapse between when the first bit is sent and the last bit is received. Namely, the delay can be decomposed in two parts
  - Transmission time $\frac{L}{R}$ (where L is the length of the data to send and R is the throughput), i.e. the time needed to send all bits.
  - Propagation delay $\frac{d}{\nu}$ (where d is the distance between sender and receiver and ν is the wave propagation speed), i.e. the time that elapse between when a bit is sent and received.

  We should also consider the time needed by the protocol to elaborate the data.
- The **minimum cycle time**. The minimum cycle time is the time for a specific system to do a specific duty cycle in a control loop.

- The **jitter**. The jitter J is the precision in performing a periodic operation. In formulas

$$J = \frac{|P - \hat{P}|}{P}$$

where P is the time for a generic control operation and $\hat{P}$ is the time for a generic operation. Notice that if J is 0, then the system is fully reliable.

4.1.2 Physical components

The physical infrastructure of a network is made of

- **Communication endpoints**. The communication endpoints are the hosts that communicate through the network. Some examples are user terminals, personal computers, motes, smartphones and servers.
- **Links**. Links are the hardware components that physically interconnect the devices of the network (not only the endpoints). Some examples are copper wires, optical fibers and wireless.
- **Network devices**. Network devices are devices that relay and handle traffic and create the physical interconnections. Some examples are access points, routers and switches. Notice that network devices are different from communication endpoints, in fact the former are not the source or destination of data.

Multi-link connectivity

An important thing to highlight is that connectivity is not always obtained with one single communication technology. For instance a smartwatch communicates with the server using two different technologies, in fact it uses BLE to communicate with a smartphone, that uses Wi-Fi or GSM to communicate with the server. Hence we have leveraged two different links.

Choosing the best technology

Choosing the best communication technology is fundamental to obtain the best performance at the lowest cost. In particular, we should consider

- **Throughput**.
- **Latency**.
- **Reliability**.
- **Energy consumption**.
- **Range**. Regarding range, we can distinguish different types of networks, with increasing range
  - Personal Area Networks (PANs) have a very limited rage (around 1 or 2 meters).
  - Local Area Networks have a medium sized range. An example is Wi-Fi.
  - Metropolitan Area Networks (MANs) have the size of a city.
  - Wide Area Networks (WANs) span huge distances. An example is fiber optics cables that can connect America with Europe.

4.1.3 Network's shape

The nodes (i.e. devices and endpoints) of a network can be physically connected through links in various ways. A configuration of nodes and links is called **topology** or **architecture**. Some of the most common architectures are

- **Linear topology**. In the linear topology each node is connected to another node. This topology requires multiple hops to transmit a packet from the source to the destination. An example is shown in Figure 4.1.

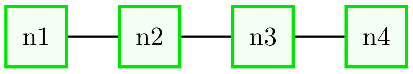


Figure 4.1: A linear topology.

- **Star topology**. In the star topology each node is connected only to a centre device. This means that this architecture doesn't allow cross communication between nodes and all the communication has to go through the centre of the star. Also notice that the centre is the bottleneck and the single point of failure of the network. However, an advantage of this topology is that the end devices (i.e. those that are not the centre device) can be very simple because all the communication overhead is handled by the centre device. An example of star topology is shown in Figure 4.2.

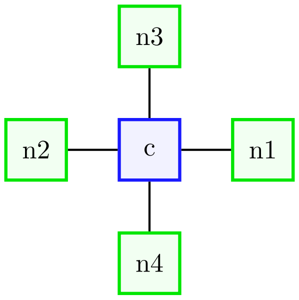


Figure 4.2: A star topology.

- **Tree topology**. In the tree topology nodes are disposed like in a tree. In particular we have a root node that has n children and each child can have children. In other words this architecture can be seen as a nested star of stars. An example of tree topology is shown in Figure 4.3.

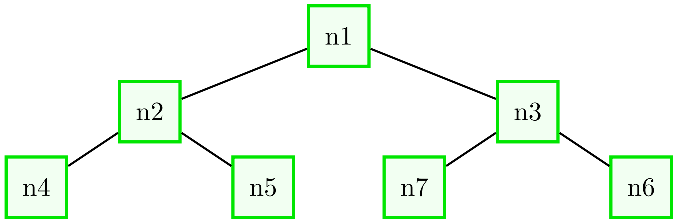


Figure 4.3: A tree topology.

- **Ring topology**. In the ring topology every node is connected to two different nodes to form a ring. An example of ring topology is shown in Figure 4.4.

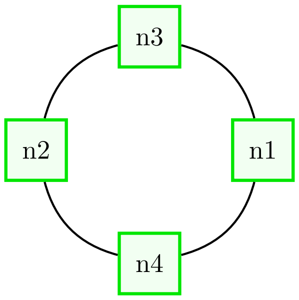


Figure 4.4: A ring topology.

- **Mesh topology**. In the mesh topology there is no single point of data collection and every node communicates with every other node. This means that this topology allows direct cross traffic (whilst the other didn't). An example of mesh topology is shown in Figure 4.5.

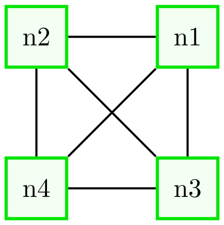


Figure 4.5: A mesh topology.

4.1.4 Protocols

Protocols are the software components that allow devices to communicate. More specifically, a protocol is a set of rules used from two devices to move data.

Packets IOT protocols move data in packets (remember we are considering the packet switch model) each of which has not only to contain the data but also some metadata and signaling data. In particular a packet is made of

- An **header**. The header contains signaling information and data about the packet itself (e.g. the address of the sender and of the destination, the length). The header data is an overhead because it's not the data the client wanted to send but it's needed only by the protocol to implement the rules.

- A **payload**. The payload is the actual data the source wants to send.

Components of a protocol When we describe a protocol we have to define

- The **syntax**. The syntax of a protocol is the format of the packets, i.e. what a well-formed packet should and can contain.
- The **operations** allowed. The operations define the logic of the communication. Operations are used to open and close a communication channel or to move data around.
- The **semantics**. The semantics of a protocol is the semantic of the packets, i.e. what every field of a packet means.

Layered structure

Protocols usually adopt a layered structure in which each layer adds an header. More precisely, When an application wants to send a packet it creates it (i.e. creates payload and header). The packet is passed to the lower layers, each of which adds some header needed to communicate with the respective layer of the receiver. When a packet reaches the physical layer (layers usually go up to down, from abstract to physical) it's sent to the receiver that starts removing the headers in reverse order while passing the packet to upper layers. When the packet reaches the application layer, the receiver can read the payload. An example of layered structure and the journey of a packet are shown in Figure 4.6. An important thing to remember is that a layer uses the services of the lower layer, but never the opposite. If the concept of using services isn't clear remember that a layer handles some functionality (e.g. the physical level handles the physical communication), thus a layer asks to the layer below to use its functionality (e.g. to send the packet).

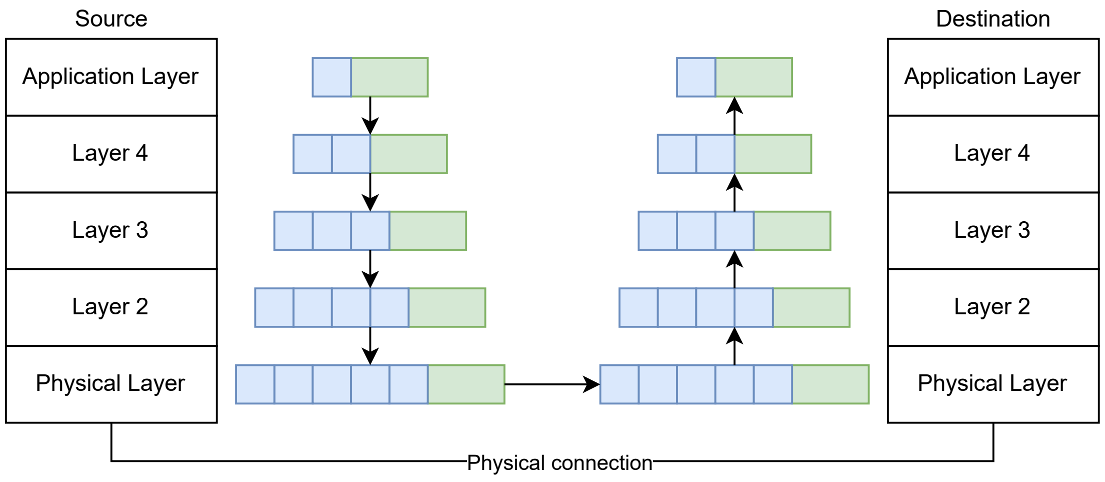


Figure 4.6: Layered protocols and the journey of a packet (headers in blue, payload in green).

Advantages and disadvantages The main advantages or the layered structure are

- It's very **flexible**, in fact we can simply add a layer to add some functionality.
- It allows **intraoperability**, in fact different devices can work independently on different levels.
- It allows for modular programming, in particular, once a layer has been programmed, the upper and lower layer can use it's interface.

The main disadvantage of the layered structure is that every layer adds some information and increases the overhead of the packet. This is critical in IOT applications because more data means more power consumption to send it. Practically, Internet devices use the OSI layered model that has 7 levels, however IOT applications use architectures with fewer levels.

4.1.5 Application layer protocols

Network devices exchange messages, however in IOT we use multiple technologies to get from the source to the destination. This means that each intermediate hop may use a different communication technology, hence a different layer stack. To solve this problem we need a common mediation protocol that runs on top of the stack (i.e. on top of all the layers). This layer is called **application layer** and typically a network uses a common application layer and different lower layer components. Examples of types of application layer protocols are

- **Client-server protocols**. In client-server protocols a device, called client, works as an active agent that issues requests to a passive device, called server. The server receives a request, executes it and replies to the client. Some examples of client-server protocols are *HTTP* for standard internet, *COAP* for IOT and *OPC-UA* (in client-server mode) for industrial environments.
- **Publish-subscribe protocols**. In publish-subscribe protocols devices can have two roles: publishers and subscribers (even both). A subscriber requests to the network (i.e. to all the other devices) that when a certain type of data (i.e. the measure of a temperature in a specific room) is ready, it's sent to the subscriber. A publisher publishes data, that is sends it to all nodes that are interested in such data. Some examples of publish-subscribe protocols are *MQTT* for IOT and *OPC-UA* (in publish-subscribe mode) for industrial environments.

Chapter 5

Client-server protocols

5.1 HTTP

HTTP (and its secure version HTTPS) is the most used protocol on the internet and is used to move web pages from a server to a client. Each resource is identified by an Uniform Resource Locator (URL) or an Uniform Resource Identifier. Say for instance we want to get page `home.html` from `www.coolwebsite.com`; the identifier of such resource is

```
http://www.coolwebsite.com:80/home.html
```

where

- `http` is the protocol used to get the resource.
- `www.coolwebsite.com` is the human-friendly address of the server where the resource is (the actual address is obtained from a DNS).
- `80` is the socket's port.
- `home.html` is the name of the resource.

Notice that each of this components is used by a different layer in the Internet protocol stack (in order, form top to bottom, application layer, HTTP layer, TCP layer, IP layer and physical layer). This approach is called **REST** and the main characteristic is that resources are uniquely identified and when a client requests a resource it's handled with a representation of such request. For instance `home.html` is the resource, uniquely identified by the URL `http://www.coolwebsite.com:80/home.html` and when a client requests it, the server returns synchronously (i.e. the server replies immediately) an HTML page (i.e. the representation of the request). Also notice that data transfer is stateless, i.e. the server doesn't keep memory of the requests previously issued. In other words, every request is treated as the first one issued by the sender.

5.1.1 IOT

The same concept can be applied to IOT but in this case the resources aren't entire web-pages but small files or scalar data.

Notice that in this architecture, motes should play the role of the servers while the devices that collect and analyse data should behave like clients. In particular a device should query one or more motes to get some data and then analyse them. This is, however, problematic because

- The idea of a server is that it should always be ready to serve a request. On the other hand motes can't be always on because it would be **too power consuming** (remember that motes use duty cycles).
- A device should contact all sensors, hence we have a **scalability problem**.
- **Motes might move** (i.e. change IP), however, as servers, they should always be reachable by clients.
- **Server applications and processes are too power hungry** for motes.

This means that, even if theoretically possible, HTTP should not be used for IOT applications.

Names Notice that, as for web pages, we have to define a name for each resource of the system In IoT, resources are usually sensor readings and we should specify the machine whose reading we want to obtain and the the specific resource. For instance, if we want to access a temperature reading of a mote `www.room_sensor.com` that has multiple sensors, a reasonable syntax would be

```
http://www.room_sensor.com/sensors?type=temperature
```

Dictionary of names Every client (i.e. entity that wants to access a resource) should have a dictionary that contains, for each mote, what resources it has. This is because a client should know to what mote a request should be sent. Basically, each node should know what resources the network offers and where are they.

Requests and responses format HTTP requests and responses in IoT have the same format of normal Internet HTTP, hence they are completely encoded in ASCII (i.e. completely textual). This is important because ASCII isn't an efficient way to encode data (in terms of number of bits sent)

5.2 COAP

COnstrained Application Protocol (COAP) is a client-server protocol that solves the problems of HTTP (i.e. the features that make HTTP not suited for IoT), in particular

- HTTP being **stateless**.
- HTTP being **synchronous**.

In particular COAP was designed to handle HTTP-like transactions over constrained networks (i.e. IoT networks). This means that we want to keep the REST transaction model and reduce the overhead of the messages.

Features The main features of COAP are

- It's a **REST protocol**, hence it's based on the client-server paradigm where a server is a sensor and a client is a device that wants to read some data from a sensor.
- Transfer can be **asynchronous**. This means that responses might come back to the client later in time that when they were issued.

- COAP uses **UPD** instead of TCP. UDP is not reliable (i.e. it's a best effort protocol) but its header is smaller (i.e. it adds less overhead) and it's faster (e.g. no need to set up communication) and more efficient. Notice that, since UPD is not reliable, we have to add some mechanisms at the application layer to reduce the number of packet lost.
- COAP supports four commands (i.e. methods): `GET` (to read information in the server), `POST` (to put data in the server), `DELETE` (to delete some data from the server) and `PUT` (to put data in the server).
- All resources are identified by a **URI**.
- COAP uses **binary encoding** (instead of textual encoding), hence it reduces the space occupied by the header. In particular the header only occupied 4 bytes.
- COAP has **resource discovery**. In other words, it has procedures to allow clients to know the resources of each server.
- COAP has **block transfer**.

5.2.1 Architecture

An IoT network we divided in different types of sub-networks

- **Constrained networks**. Constrained networks are made of constrained machines, like motes, that communicate using the COAP protocol. This network contains both clients and servers.
- **Internet networks**. Internet networks use HTTP and Internet to connect non-constrained devices.

These sub-networks can communicate thanks to **proxies**, i.e. devices that translate protocols. Say we, from our computer, want to issue a COAP request from the web-browser (that issues HTTP requests). We have to send the HTTP request to a proxy that translates it to COAP and sends it to the sensor (and vice versa for the response).

Proxies Proxies are also used for caching to improve performance. For instance, if we do multiple requests in a short period of time, the first request is sent to the sensor and the result is stored in the proxy. The following requests aren't sent to the proxy and the stored response is sent back instead. Caching in IoT systems is very important because it can increase speed and reduce the network traffic and the number of accesses to sensors, hence their energy consumption.

Proxies however are single points of failures and can create problems with the freshness of information, in fact the data cached on a proxy might not be the newest produced by the sensor.

The COAP standard doesn't define the policies that should be used for proxies, hence it's up to the designer to define how and when information should be stored and refreshed.

5.2.2 Messages

Since COAP uses UDP, which is not reliable, we have to add reliability functionalities to the application layer. In other words, it's COAP itself that has to handle reliability. In particular COAP defines two type of messages

- **Confirmable messages**. Confirmable messages have to be reliable. Reliability is obtained thanks to a stop-and-wait mechanism that forces the sender to wait the acknowledgement from the receiver for an interval of time.
- **Non-confirmable messages**. Non-confirmable message aren't required to be reliable, hence are delivered in a best-effort way.

Both types of messages can detect duplicate messages using an identifier for each message, hence if a message is sent twice, a device can understand it because both messages have the same identifier. Notice that this functionality wasn't implemented in HTTP.

5.2.3 Semantics

A message, like in HTTP, is either a **request** message or a **response** message. Formally, COAP defines request messages or response messages that can be embedded in confirmable or non-confirmable messages. This means that the application layer can handle four different types of message (request confirmable, request non-confirmable, response confirmable, response non-confirmable) to the UDP layer. Each type of message is identified by a different id, hence a message is identified by a couple of ids, in particular

- The request or response message is identified by a **token**.
- The confirmable or non-confirmable message is identified by an **id**.

Say for instance we want to send a `GET` request for the `\temperature` resource using a confirmable message. First we create a request message with a token that identifies it and then we encapsulate it in a confirmable message with an id. As we can see, the full message is identified by the couple token-id.

Confirmable messages

Confirmable messages are identified by a `con` tag (i.e. a string of bits) in the header of the message. A confirmable message should be immediately (i.e. synchronously) acknowledged by the receiver with a message with the `ack` tag. An acknowledgement message contains the id of the request message that it's acknowledging.

Synchronous and asynchronous communication Confirmable messages can support synchronous and asynchronous communication, in particular

- In **synchronous communication**, when a client sends a request, the response is embedded in the acknowledgement, hence it's delivered immediately. Notice that in this case the id and token of the acknowledgement (i.e. of the response) are the same of the confirmable message (i.e. of the request). An example is shown in Figure 5.1.
- In **asynchronous communication**, when a client sends a request, the acknowledgement doesn't contain the response. Instead, the response is sent asynchronously (i.e. later in time) using a confirmable message (that should be acknowledged by the client) that contains the response message. This means that the confirmable message in response has a different id but the response message inside it has the same token of the respective request. An example is shown in Figure 5.2.

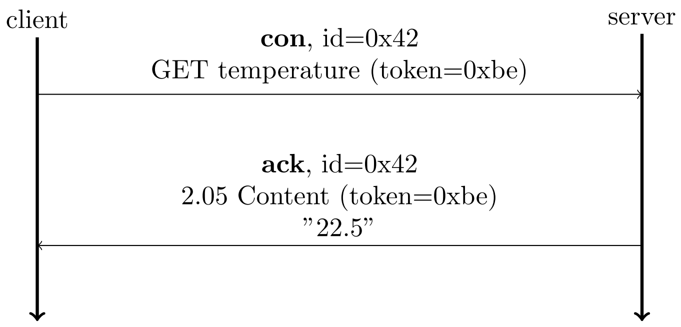


Figure 5.1: Synchronous confirmable interaction.

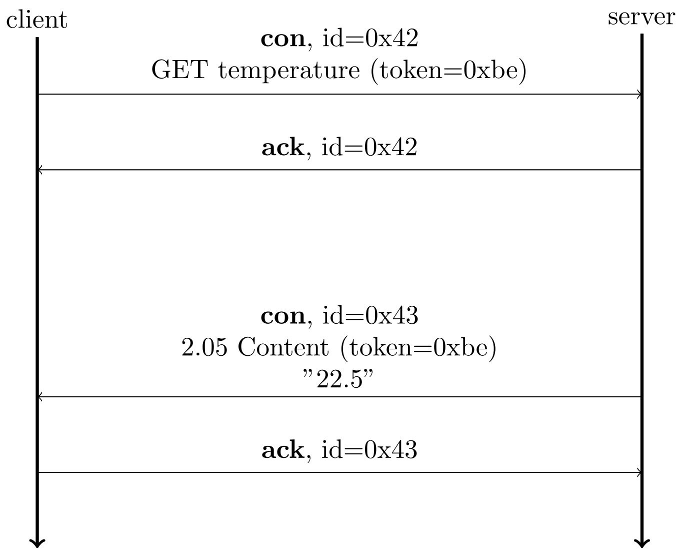


Figure 5.2: Asynchronous confirmable interaction.

Non-confirmable messages

Non-confirmable messages are identified by a `non` tag in the header of the message. A non-confirmable message isn't acknowledged by the receiver. A request-response transaction is shown in Figure 5.3.

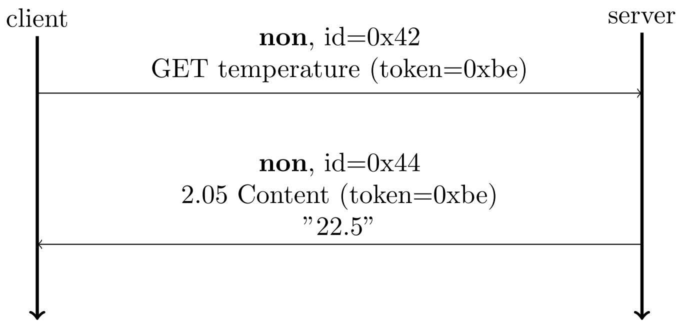


Figure 5.3: Synchronous confirmable interaction.

Header

The header of a COAP message is made of the following fields

- **Ver** (2 bits). The version of the protocol.
- **T** (2 bits). The message type (Confirmable, Non-Confirmable, Acknowledgement, Reset).
- **TKL** (4 bits). The token length, i.e., if any, the number of token bytes after this header.
- **Code** (8 bits). The request method (1-10) or response code (40-255).
- **Message ID** (16 bits). The identifier of the message.

The header can also contain some optional fields that occupy 8 bytes,

- The **token** of the message is added after the Message Id field and is 4 bytes long.
- Other optional fields are located after the token and occupy 4 bytes in total.

5.2.4 Reliability

COAP enforces reliability using a stop-and-wait mechanism. In particular when a message is sent it's stored in a buffer until an acknowledgement. In the meanwhile, a timer is started and if the message isn't acknowledged when it expires, the message is sent again. An example is shown in Figure 5.4. The COAP standard defines how the timeout should be set. In particular, initially the timeout should be set to something around 2.5 seconds and it should be doubled every time it expires. The timeout should never exceed 93 seconds (or exceed a number of retransmissions) and if this threshold is reached, the stop-and-wait procedure should stop.

Notice that the timeout is doubled and not kept equal because if a packet is lost, the probability that a packet is lost again is very high. This happens because the first cause of packet loss in a network is buffer overflow in routers (or intermediate devices). Basically, if the network is congested the routers' buffers might not be able to accept packets hence dropping them.

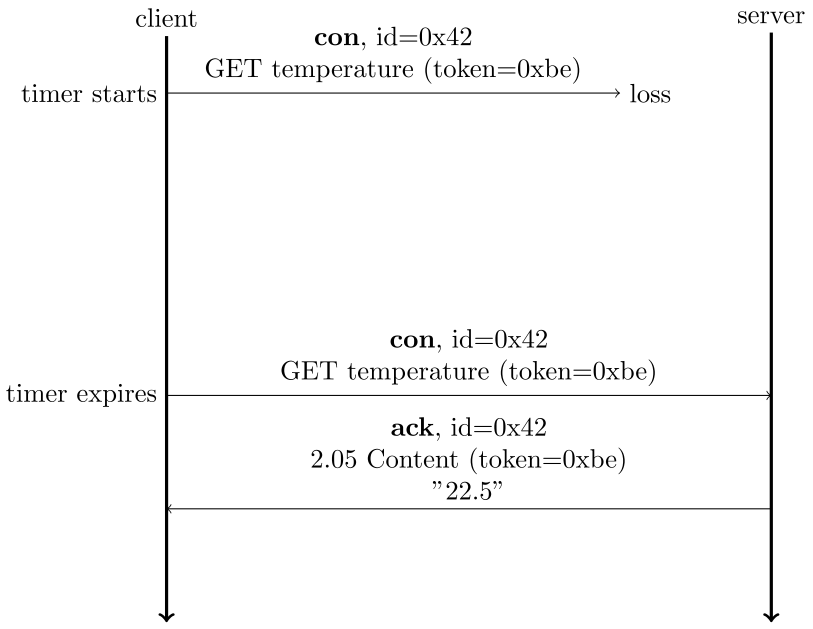


Figure 5.4: Stop-and-wait mechanism for COAP's confirmable interaction.

5.2.5 Observation

IoT systems have a **converge-cast** traffic pattern. Namely, data goes from many field sensors to a single point. However, client-server protocols and the REST paradigm define a pull traffic pattern. Moreover, in IoT the traffic is usually periodic, especially for monitoring systems (i.e. systems that monitor an event, e.g. the temperature of a room). However, the only way to implement periodic traffic with client-server protocols is to periodically issue requests, which generates a lot of traffic (for every read we have to send a request and a response). COAP solves this problem using observation, in particular a client can request a server to keep sending responses periodically without waiting for a request. The client can activate the observation mode using the `Observe` field in a request message specifying if the update should be send at specific time intervals or when the read changes. When the server receives a request for a resource with the `Observe` field, it stores the client that requested to activate the observe-mode and sends to it data without waiting for requests.

Observation requires the server side to be more complex, because it need to store some information and requires a more complex logic.

5.2.6 Block transfer

COAP is based on UDP and IP, which provides segmentation. Namely, if an UDP frame is too long for an IP packet, it's fragmented to fit in a packet. However, COAP doesn't use normal IP, hence we have to fragment data at the application layer so that it can fit in an IP packet. This functionality, called **Block Transfer Mode**, is activated using an option in the header that specifies

- The **incremental block number** `nr` within original data.
- The **more blocks flag** `m`. This field says if more blocks of data are available.

- The **block size** `sz`.

5.2.7 Resource discovery

An important functionality offered by COAP is resource discovery. It allows clients to query a server and ask what resources can be accessed. In particular a client can obtain the resources exposed by a server accessing the `.well-known/core` resource. Basically resource discovery works as follows

1. The client sends a `GET` request for the resource `.well-known/core`.
2. The server replies with a link-header style format that specifies, for each resource, its characteristics like URL, type, interface and content-type.

Chapter 6

Publish-subscribe protocols

6.1 MQTT

MQTT is a publish-subscribe protocol suited for IoT thanks to its ease of implementation (especially on the mote's side) and light-weightiness. The main characteristics of MQTT are

- It's **simple to implement**. This is the main advantage with respect to COAP, in fact the client-server application forces us to implement and run the server side on the mote. In this case the program on the sensor is way less complex.
- It has **Quality of Service** (QoS) support. This means that MQTT provides different services depending on the type of traffic that we want to generate. For instance we can decide how reliable or fast a communication can be.
- It's **lightweight** and **bandwidth efficient**. This is particularly good if we remember that most of the power in motes is used in the communication module for sending and receiving messages.
- It's **data agnostic**. This means that it can transfer any type of data.
- It has **session awareness**. This means that when a device disconnects from the network, it's not forgotten and when it comes back it's like it never left because some information is kept by the network. This is a big advantage because, say a device subscribed for many topics, when it disconnects and reconnects it doesn't have to subscribe again (hence sending messages and consuming power) because the network remembered its subscriptions.
- It **works on top of TPC**, hence it's reliable by construction.

6.1.1 Communication pattern

In a network that uses MQTT, every device has a MQTT client that can interact with the network **publishing** messaging or **declaring interest** in a message. When the network receives a message, it sends it to all devices that are subscribed to (i.e. that have declared interest in) that message. In other words the network is a **message broker** that mediates all transactions. This means that

- There is **no direct interaction** between devices.

- **Devices don't know each others**.

Another thing to highlight is that MQTT uses a push traffic pattern, in fact the devices push messages (publish or subscribe messages) to the network. This pattern generates one-to-many (a message is sent from the broker to all subscribers) multicast communication.

Network

As we have seen, the network acts like a message broker and it's fundamental for the protocol to work, hence it's worth describing it in further detail. The network is a distributed system connected to publishers and subscribers, however the devices perceive it like a single device to which send subscribe request and publish data.

Subscriptions

The network has to store (say in a database) all the subscriptions so that when a device publishes a message, it knows where to relay it. Subscriptions are based on topics, in particular every message has one or more topics (specified by the publisher) and each topic is a structured hierarchical name (i.e. a sequence of names divided by backslashes, each of which refers to a level in the hierarchy). Consider for instance an house that has many rooms and appliances with different sensors and say we want to identify the temperature of the fireplace in the living room. That topic can be identified as

```
/home/living_room/fireplace/temperature
```

When the temperature sensor in the fireplace publishes a message it adds the topic above so that all those who have subscribed to it can receive the temperature update. The topic can also contain wildcards that allow to specify multiple topics with a single name,

- The `+` wildcard represents all topics of a certain level in the hierarchy. For instance we can use `/home/living_room/+/temperature` to represent all temperature motes in the living room. Remember that, a `+` represents a single level, hence a topic `/home/garden/flowers/humidity` won't match the topic `/home/garden/+`
- The `#` wildcard (that can be used only at the end of a topic) represents all subtopics of a certain topic. For instance we can use `/home/living_room/#` to represent all resources in the living room.

The broker (i.e. the network) stores topics and subscribers in a key-value structure where

- The key is the topic.
- The value is the subscriber.

and

- When the network receives a subscription from a subscriber it adds an entry that maps the subscription topic with the subscription sender (i.e., `<msg.topic:msg.sender>`).
- When the network receives a message from a publisher it relays it to the devices that have subscribed to that topic (i.e. to `database[msg.topic]`).

6.1.2 Communication protocol

The MQTT protocol offers operations to

- **Establish the connection** with a broker.
- **Publish** a message.
- **Subscribe** for a topic.
- **Unsubscribe** for a topic.

Establishing connection

The first step of the protocol is establishing a connection with the broker. In particular

1. A device willing to open a connection sends a **connect** message to the broker. The connect message contains
   - The `clientId` field that contains the identifier of the client.
   - The `cleanSession` field that specifies if the information saved by the broker should be deleted (`=true`) or not (`=false`).
   - The `username` and `password` fields used for authentication.
   - The `lastWillTopic` field (more on this later).
   - The `lastWillQoS` field (more on this later).
   - The `lastWillMessage` field (more on this later).
   - The `keepAlive` field that specifies how much time should pass between two messages before the broker disconnects the device.
2. The broker acknowledges the message with a **connack** message. The connack message contains
   - The `sessionPresent` field.
   - The `returnCode` field where 0=OK, 1=unacceptable version, 2=id rejected, 3=server unavailable, 4=bad username and pwd, 5=unauthorized.

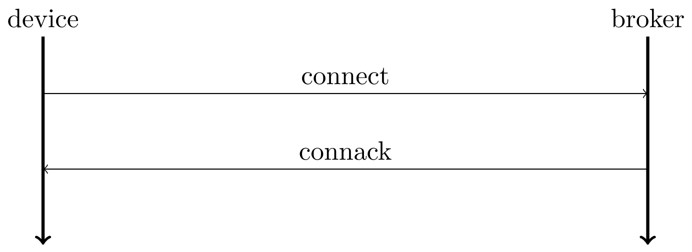


Figure 6.1: Connection procedure.

Publish

The publish procedure works as follows

1. A device that wants to publish a message sends a **publish** message to the broker. The publish message contains
   - The `packetId` field that specifies the id of the packet. The id is used for QoS.
   - The `topicName` field.
   - The `QoS` field that specifies the Quality of Service requested by the device. The value of this field is an integer between 0 and 2 which represents a QoS class.
   - The `retainFlag` field.
   - The `payload`, i.e. the the message that the device wants to send.
   - The `dupFlag` field. The dup flag indicates that the message is a duplicate and was resent because the intended recipient (client or broker) did not acknowledge the original message.
2. The broker relays the message to all devices interested.

Quality of Service The Quality of Service field specifies the requirements that the devices wants when it sends the publish message. In particular

- `QoS=0` means that the broker (or in general the receiver) should receive the publish message **at most once** (i.e. best effort transfer). In practice, the broker doesn't acknowledge the device, hence the broker might have received the message or not. This QoS ensures the lowest latency and power consumption (only one message is sent) but also the lowest reliability.

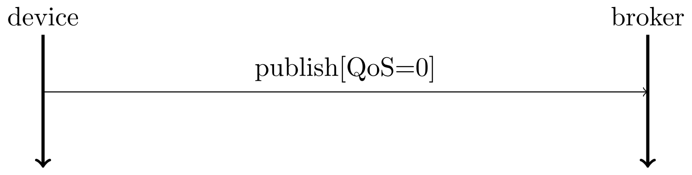


Figure 6.2: Publish protocol with QoS 0.

- `QoS=1` means that the broker (or in general, the receiver) should receive the publish message **at least once**. In practice, the device keeps sending a publish message (each with a different id) until it receives a **puback** acknowledgement message from the broker. The puback message contains the `packetId` field with the id of the message that has been acknowledged. Notice that, this QoS ensures that a message is received at least once, but the broker could receive the same message multiple times.
- `QoS=2` means that the broker (or in general, the receiver) should receive the publish message **exactly once**. In practice,
  1. The device sends the **publish** message.
  2. The broker stores the publish message so that no duplicate package is stored and acknowledges the publish message with a **pubrec** message.
  3. The device acknowledges the pubrec message with a **pubrel** message.

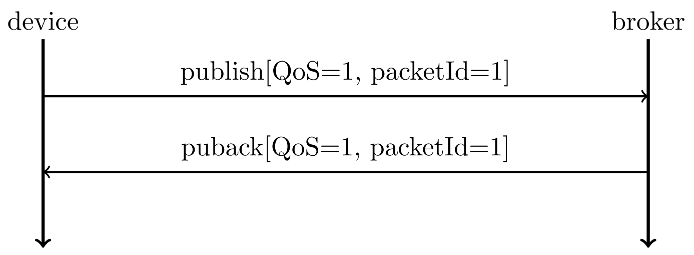


Figure 6.3: Publish protocol with QoS 1.

4. The broker acknowledges the pubrel message with a **pubcomp** message.

All acknowledgment messages (i.e. pubrec, pubrel and pubcomp) contain the `packetId` of the publish message. This QoS is the slowest because it requires, in the best case 4 messages but it's the most reliable.

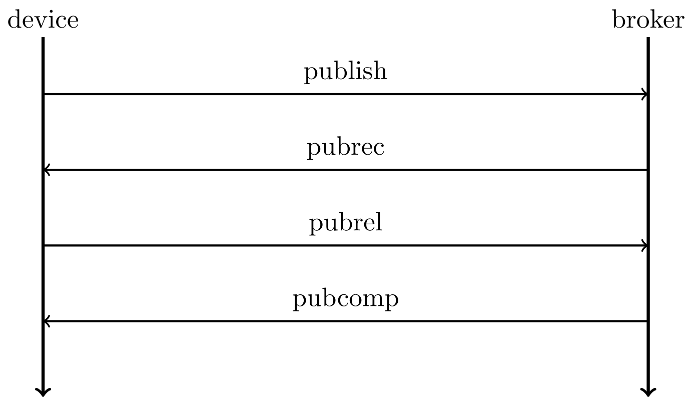


Figure 6.4: Publish protocol with QoS 2.

Subscribe

The subscribe procedure works as follows

1. The device sends a **subscribe** message to specify the topics to which it wants to subscribe. This message contains
   - The `packetId` field that specifies the identifier of the message.
   - Up to eight couples `QoSn, topicn` that specify, for each couple, a topic to subscribe to and the Quality of Service requested when the published message is relayed to the device.
2. The broker acknowledges the subscribe message with a **suback** message. The suback message contains
   - The `packetId` field that specifies the identifier of the message acknowledged.
   - Eight fields `returnCode n` that specify the success code for each topic specified in the subscription message.

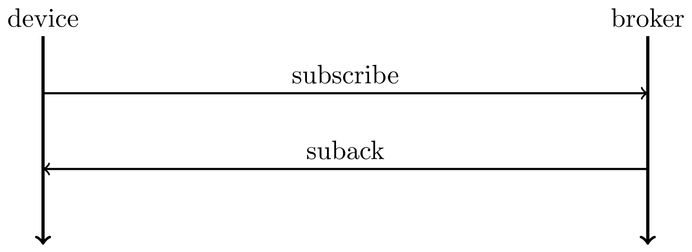


Figure 6.5: Subscribe procedure.

Unsubscribe

The unsubscribe procedure works as follows

1. The device sends an **unsubscribe** message to specify the topics to which it wants to unsubscribe. This message contains
   - The `packetId` field that specifies the identifier of the message.
   - Up to eight fields `topic`, each of which specifies a topic to unsubscribe from.
2. The broker acknowledges the unsubscribe message with a **unsuback** message. The unsuback message contains the `packetId` field that specifies the identifier of the message acknowledged.

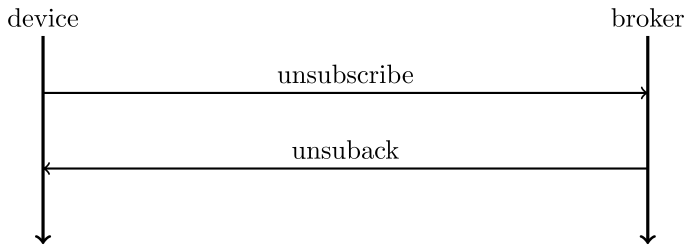


Figure 6.6: Unsubscribe procedure.

6.1.3 Persistent sessions

An important feature of MQTT is that, when a device disconnects from the network, its data is not lost. In particular the broker stores

- The **identifier of the client**.
- All the **subscriptions**.
- All **pending messages** that haven't been acknowledged by the client.
- All **pending messages** that haven't been acknowledged to the client.
- All **messages that the client was subscribed to** but that it hasn't received because it was disconnected.

On the other hand, the device stores

- All messages which are not confirmed by the broker.
- All received QoS 2 messages, which are not yet confirmed to the broker

Notice that this feature requires the broker to have a lot of storage.

Retained messages

In MQTT, being a publish-subscribe protocol, publish and subscribe messages are decoupled and asynchronous. This means that when a device subscribes to a topic, all previous messages for that topic aren't sent to the device. To solve this problem, publishers can set the `retainFlag` in the publish message to true to specify that the message should be saved by the broker and sent to all new subscribers to the message's topic.

6.1.4 Last will

Another important feature of MQTT is that a device can specify a message that the broker has to send when the device unexpectedly fails. In particular the device specifies, in the connection message,

- The text of the **message** in the `lastWillMessage` field.
- The **topic** of the message in the `lastWillTopic` field.
- The **QoS** required for the transmission of the message in the `lastWillQoS` field.

The last will message and the related data is stored locally by the broker and is sent to all devices subscribed to the `lastWillTopic`.

6.1.5 Keep alive

The broker can check if a device is still active using `ping` messages. In particular a device sends a `ping` message every n seconds and the broker replies with a `pong` message to notify the client that it's active and has received the `ping`. The parameter n is communicated by the device in the `keepAlive` field in the **connect** message. The broker assumes that a device has failed if after n seconds with no communication, and starts the last will procedure.

6.2 MQTT-SN

MQTT-SensorNetwork is a lightweight version of MQTT designed for IOT systems that have very strict constraints. The main differences with MQTT are

- It uses **UPD instead of TCP**. This change allows to remove some overhead, in fact TCP adds around 40 bytes of overhead, whilst UPD only 8. The main drawback is that UDP is not reliable.
- **Removes some overhead** of MQTT. In particular it removes some unnecessarily packets and some parameters in the messages.

In a nutshell, MQTT-SN is a compressed version of MQTT that runs on UDP. This allows the protocol to be more lightweight hence less power consuming.

Part IV

Communication technologies

Chapter 7

Wired communication technologies

Wired communication technologies are widely used in Industrial IoT systems. In particular, these technologies can be separated in

- **Fieldbus**. Fieldbusses are physical busses used to move data within devices of IIoT systems.
- **Industrial Ethernet**. These technologies are based on the Ethernet standard and are custom made for industrial environments.

The latter technology is taking over the former one for interoperability reasons, in fact Ethernet is one of the, if not the, most spread technology for wired LANs, hence, if we use Ethernet we can more easily communicate with the Internet. Moreover, fieldbusses have their own stack, hence we need a mediator (i.e., a proxy) to move to the Internet data generated by a system that uses fieldbus.

7.1 Fieldbus

Fieldbus is a wired communication technology that aims at connecting field devices using a common bus to which all devices are connected. There exists many variations (i.e,. dialects) of fieldbus, all of which are based on the same foundations and characteristics

- The **messages** exchanged are **very short**.
- The messages have **tight requirements in terms of delay and determinism**. In other words, communication has to be fast (delay) and with low jitter (determinism).
- The protocol stack is divided in three layers
  - The **physical layer**. The physical layer defines the type of bus used to communicate.
  - The **data link layer**. The data link defines the set of rules to handle the shared use of the shared bus.
  - The **application layer**. The application layer, that defines what and how messages are exchanged, is custom made for this stack. This means that we need some proxy to communicate with the Internet. In other words, since the message format isn't the same used by other protocol stacks, a device can't send the same message using different lower layers.

Notice that the application layer, differently from the classical Internet stack, is directly putted on top of the data link layer. In particular, we miss

- The **network layer**. This layer basically handles naming and routing, however in this case we use a common data bus, hence we need no routing.
- The **transport layer**. This layer handles end-to-end communication, however since we don't have multiple hops we don't need it. Furthermore, some communication layers (e.g., TCP) provide reliability mechanism which we don't need here because all devices are connected to a single bus.

Remember that, since all devices are connect to the same bus, when one sends a message, all the others receive that message.

7.1.1 Data link

In fieldbus, all devices are connected to the same bus, hence all motes might access the bus concurrently. This means that we have to handle possible collisions between different devices that try to access and transmit on the bus. There exists two main solutions to handle this issue

- **Scheduled access**.
- **Random access**.

Schedule access

Schedule access defines a structure such that collisions are avoided a priori. In practice, each device is assigned a different time interval in which it can use the bus. Some examples of schedule access protocols are

- **Polling techniques**. In polling techniques a central entity tells to each device, when its turn comes, that it can start using the bus.
- **Centralised scheduling schemes**. Centralised scheduling schemes use a distributed protocol (i.e. no central coordinator is needed) that allows to define, for each device, at what time it can use the bus.

Advantages and disadvantages The main advantage of schedule access is that it provides **guaranteed performance**, however the main drawbacks are that

- **We need a centralised coordinator or some coordination mechanism**.
- It's **complex to implement** due to the coordination logic. This logic also adds some overhead in the communication, hence it requires more power.

Random access

Random access allows every device to use the bus (i.e. we don't avoid collisions a-priori) and when a collision occurs the devices involved have to recover from it (use the bus another time) in a smart way. Usually recovering from a collision involves some randomisation.

Advantages and disadvantages The main advantages of random access are that

- It's **easy to implement**, in fact we only need a random number generator.
- It leverages the **opportunistic use of the bus**.

However,

- It **works poorly under heavy traffic conditions**, because many collisions occur.
- We have **no (or only static) guarantees on performance**.

7.1.2 CAN bus

CAN bus is an implementation (or a flavour) of fieldbus.

Physical layer

The physical layer is a data bus commonly realised with a twisted copper pair (like the one in a landline). The bit rate obtained using a CAN bus is around 1 Mbps, however this number decreases with the length of the bus (i.e. the longer the bus, the lower the bit rate).

The CAN bus also offers a detecting functionality. Namely, it allows a device to check if, while sending a message, some other device is using the bus.

Also remember that the communication happens in broadcast, in fact when a message is sent, everybody receives it, since everyone is connected to the same bus.

Frames

All devices connected to the network exchange frames. Each frame is delimited by some Start Of Frame (SOF) bits and some End Of Frame (EOF) bits so that each device knows exactly where a frame starts and ends. In between a frame contains the following fields

- **Identifier** or Arbitration Part (11 bits). This field is used to uniquely identify a transmission.
- **Control bits**. These bits are used to better specify the content of the frame.
- **Data** (0 to 8 bytes). This field contains the application message.
- **CRC**. This field contains data for the parity check (i.e. to detect integrity errors in the frame).
- **Ack**. This field is used if the frame has to be acknowledged by the intended receiver.

As we can see, the frame doesn't contain the destination address, hence the frame is delivered and processed by every device on the bus. The intended receiver of a message should be specified by the application layer in the data part (i.e. in the application message).

Data link layer

The data link layer defines how each device should access the bus and what to do in case of collisions. In particular, a CAN bus uses Carrier Sensing Multiple Access (CSMA), i.e. the sender, before using the bus, has to check if some other device is sending messages (i.e. is using the bus). If the bus is busy, the sender refrains from sending the message, otherwise it can freely use the bus.

This technique allows the devices to reduce the number of conflicts, however we can still have concurrent access to the bus. Conflicts might still happen because a message takes time to travel the bus, hence when a device senses the bus, it can only check if a message has arrived there, not if a message has been sent. In particular, the time during which a device can't be sure that a message is actually on the bus is called **vulnerability period**. This means that a collision happens when a device sends a message and another devices sends a message during the vulnerability period of that message. The length of the vulnerability period depends on the length of the bus (and the distance between the two devices).

Conflict handling Conflicts are handled by CAN bus using transmission identifiers. In particular, each identifier in the frame specifies also the priority of the message and the lower the value of the identifier, the higher the priority of the transmission. Since the devices can always sense the bus, if a sender realises a collision has happened, it can also check if the collision happened with a frame with higher or lower priority. If the device senses a collision with an higher priority message then it stops transmitting.

This mechanism works very good and, in particular, it allows to obtain perfectly reliable transmission for the frames with high priority.

Application layer

CAN bus doesn't define a single protocol for the application layer, in fact, depending on the field of application, we can use different application layer protocols each with different needs and constraints.

7.1.3 PROFIBUS

PROFIBUS has all the characteristics of a fieldbus, however its distinctive characteristics are

- It supports **different types of busses**, among which fiber optics. For this reason, PROFIBUS can reach higher bit rates than CAN bus. In particular we can reach up to 12 Mbps.
- The bus is accessed using a **scheduling mechanism**. This means that one device is elected as master and it has to schedule transmission through the bus. In particular the master uses a polling scheme, i.e. it tells every device, in turns, that it can start using the bus.
- The application layer depends on the field of application.

7.2 Ethernet

Ethernet based communication is based on the Ethernet standard widely used for wired Internet access. Ethernet is based on two layers

- The **physical layer**. In the physical layer we can use multiple types of cables to transmit data. Nowadays the most used cable is fiber optics.

- The **data link layer**. This layer is responsible for frame filtering, multiplexing and multiple access.

In this case we don't have any upper layer because Ethernet is compatible with any other layer. This is a big advantage, however, it also has its drawbacks. In particular, since it can be used for whatever application, it also has to handle multiplexing depending on the layers above and the information about the upper layer protocol should be inserted in the Ethernet frame. Remember that Ethernet is also based on a bus structure.

7.2.1 Frame

An Ethernet frame is made of

- A **Synchronisation** field (7 bytes) used to synchronise sender and receiver. This field is filled with 7 bytes `0x10101010`.
- A **Frame Delimiter** field (1 byte) used to specify where the frame starts. This field is filled with `0x10101011`.
- The **Source Address** (6 bytes).
- The **Destination Address** (6 bytes).
- The **Type** flag (2 bytes) used to support multiplexing. In particular each upper layer protocol has a different code (e.g. `0x0800` for IP) that is used for multiplexing. This field is also used to define what is the format of the data passed by the upper layer protocol (i.e. the format of the `data` field).
- The **Data** (46 to 1500 bytes).
- The **Frame Check Sequence** (4 bytes) used for error checking.

Just looking at the frame we can see some differences with CAN bus (and fieldbus in general)

- In Ethernet the frame contains the sender's and receiver's address, hence Ethernet also provides unicast transmission (i.e. from sender to receiver). On the other hand, fieldbus only allows broadcast communication.
- In Ethernet the data size is much bigger than CAN bus.

MAC address

The source and destination addresses are MAC addresses, i.e. 48 bits (6 bytes) addresses usually represented as 6 bytes (from `0x00` to `0xFF`) separated by a column.

```
4E:3A:42:FF:7D:8C
```

The bytes are divided as follows

- The first three bytes identify the manufacturer.
- The last three bytes identify the interface.

Ethernet also allows broadcast communication simply specifying the address `FF:FF:FF:FF:FF:FF`.

7.2.2 Collision detection

Ethernet, just like CAN bus, uses Carrier Sensing Multiple Access to reduce the probability of a collision. Basically, each device senses the bus (i.e. the cable) and sends a message only if no one is using it. As for CAN bus, collisions can still happen, however frames don't have priorities, hence we have to use a different mechanism to handle collisions. In Ethernet, if a collision happens, all the devices involved in the collision stop transmitting and try again after some time.

Hubs and switches

In modern Ethernet, the collision problem is solved using hubs and switches. In particular the common bus is replaced by a star architecture in which all devices are connected to a central switch (i.e. there is no shared bus). The switch can read the frames and relay the frame depending on its content (i.e. the content of the fields). Having a switch in the middle solves the problem of collisions because we have a single link for each device and the switch handles the all processes that want to access the bus concurrently. In practice, the switch stores all the frames and sends them to the sender (or in broadcast) as soon as the link is free. Notice that it's possible to use a central switch because the frame contains the source and destination address.

Thanks to switches we can also avoid using CSMA (i.e. sensing the link before sending) because no collisions can happen.

7.2.3 Industrial Ethernet

Ethernet-based technologies for industrial environments are divided in three classes depending on the adds-on provided with respect to classical Ethernet,

- **Class A**. Class A is basically Ethernet as used in Internet. Some industrial technologies that use Class A Ethernet are Ethernet/IP or Foundation Fieldbus. The problem of plain Ethernet is that it has a very high delay caused by the fact that we can't distinguish different traffic types at the data link layer. This happens because we can't prioritise different frames and the Minimum Cycle Time (MCT) only depends on how the switches handles the frame and the conflicts. Class A Ethernet has a reference MCT of not less than 100 ms.

- **Class B**. Class B adds priorities to Ethernet adding some fields in the frames. This means that a switch can analyse the frame and check if the frame comes from a real-time (i.e. high priority) application or a normal application. Notice that, in real-time applications, the network and transport layer are removed and the application layer is put just above the Ethernet layer. Priorities allow to have a lower delay and MCT (around 10 ms) with respect to Class A. Also remember that Class B technologies are based on the same physical layer of Class A's. An example of Class B technology is Profinet.

- **Class C**. Class C is the best class in terms of delay and MCT (around 1 ms, hence comparable to CAN bus). This speed-up is obtained adding scheduling, hence modifying the structure of the switches. In particular, switches doesn't only relay frames but also schedule the transmission of all devices connected to the switch. Some examples of Class C technologies are EtherCAT, Profinet IRT and Ethernet Powerlink.

Chapter 8

Wireless communication technologies

Wireless communication technologies can be divided in three main categories

- **Mobile radio networks**. Mobile radio networks are basically cellular networks that are used to interconnect motes and smart objects.
- **Capillary multi-hop networks**. Capillary multi-hop networks use multi-hop communication.
- **Cellular IoT operators**. Cellular IoT operators similar to mobile radio networks but handle only IoT traffic.

8.1 Mobile radio networks

Mobile radio networks are basically cellular networks that are used to interconnect motes and smart objects. Using this alternative, each mote is like a smartphone connected to the cellular network. Remember that a cellular network has an architecture composed of two main parts

- A **Radio Access Network** (RAN). A RAN is the wireless segment that interconnects all the smartphones (i.e., the motes) with the first element of the network (e.g., an antenna).
- A **Core Network** (CN). All antennas, also called base stations, are connected through the CN. Usually the CN is wired and is composed of different elements to offer different services (e.g., SGWs, MMEs, PGWs and PCRFs). Some examples of services are the handover (i.e., track a phone from a base station to the other while it's moving), billing and radio resource management.

8.2 Capillary multi-hop networks

Capillary multi-hop networks use multi-hop communication. This means that data isn't transmitted directly to the first point of access to the Internet and we have some intermediate layers, each offering different functionalities. In particular a capillary multi-hop network is made of

- A wireless **field network**. The field network is managed by local gateways, that have the same functionality of base stations in mobile radio networks. The main difference with base stations is that in this case the gateways cover only a few meters while the base stations can cover a lot more space (i.e., kilometers).
- A **backhauling network**. The backhauling network is used to connect the local gateways to the first access point to the Internet (i.e., the back-end server). This network can be implemented with different technologies, like PLCs, Wi-Fi, Ethernet, cellular (e.g., 4G) and many others.
- A **back-end server**. The back-end server is the point of access to the internet.

8.3 Cellular IoT operators

Cellular IoT operators similar to mobile radio networks but handle only IoT traffic. In particular the network is made of

- A **Radio Access Network** that interconnects the devices with a base station.
- A **back-end server**. The server is the point of connection with the Internet and is accessed by the base stations.

The most important difference is that this architecture only handles IoT traffic. Also notice that the RAN of cellular IoT operators has the same range as in mobile radio networks (i.e., kilometers).

Chapter 9

Long range wireless communication technologies

Mobile radio networks and cellular IoT are both long range wireless communication technologies, however the latter is completely dedicated and custom made for IoT, hence we will focus on them. All cellular IoT technologies

- **Are much cheaper than mobile radio networks**. In particular, the hardware needed to realise the network costs way less. The reason for this is that IoT applications don't need an high throughput hence the hardware used to transmit and receive data can be much simpler.
- **Run on unlicensed bands**. This means that the operators don't have to pay for a frequency band. Notice that this helps keep the costs low.
- **Support out-of-the-box connectivity**. This means that, since the network is very simple, one can buy a transceiver and straightly connect to the network.

9.1 LoRaWAN

LowRangeWAN (LoRaWAN) is a cellular IoT technology.

9.1.1 Architecture

The LoRa network is made of

- **Sensor nodes**. Each mote is equipped with a LoRa transceiver.
- **Gateways** (or concentrators). A gateway is a piece of hardware that plays the role of the base station in a mobile radio network. Basically, it connects the motes with the other layers of the network.
- A **network server**. The network server is the place where the intelligence of the network is. In particular, it's a software component on a machine or data centre that has the responsibility to menage the entire network. The network server is connected to the gateways with a backhauling network. The network server also provides interfaces that allows applications to use the data generated by the sensor nodes.

An important thing to remember is that the whole architecture is **association-less**. To understand what association-less means we have to describe what association in mobile radio networks is. In particular in a cellular network (i.e. in a mobile radio network), the network knows, for each device to which base station it's connected. This information is dynamically updated when a device moves to a different base station. This behaviour is not present in LoRaWAN. In practice, this means that the

1. End devices don't have to associate to a gateway and they only have to broadcast the message they want to send.
2. The message is read by one or more gateways and sent to the network server.
3. The network server has to manage the messages, i.e.
    - Handle duplicate messages generated by multiple gateways that relay the same message.
    - Manage acknowledgements.
    - Manage radio link parameters.

Having a single management point (i.e. the network server) has it's own advantages and disadvantages. In particular, all the load is on the network server, this is a disadvantage (because we have a single point of failure and load) but also an advantage because it allows us to keep the end devices and the gateways very simple (i.e. they only send or relay packets). This allows to

- Reduce cost.
- Reduce the energy consumption of end devices.

9.1.2 Protocol stack

The protocol stack of LoRaWAN is made of

- The **physical layer**. In particular LoRaWAN defines multiple physical layer because LoRa has to be able to operate on different unlicensed bands depending on the region of the world where it's used.
- The **LoRa modulation layer**. The protocol used at the LoRa modulation layer is proprietary, hence its details are not disclosed.
- The **LoRa MAC layer**. The LoRa MAC layer defines the rules to access the gateways. In practice, LoRa MAC (Medium Access Control) handles the conflicts between packets that are concurrently received by a gateway. This layer can be implemented in three different ways (Class A, Class B and Class C).
- The **application layer**. The application layer is immediately on top of the MAC layer, hence no transport not network layer is needed.

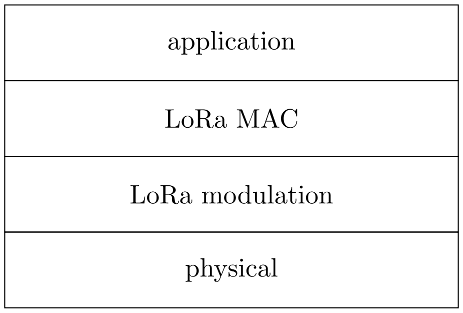
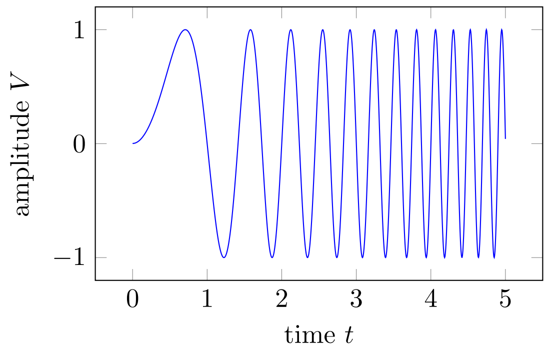


Figure 9.1: LoRaWAN's protocol stack.

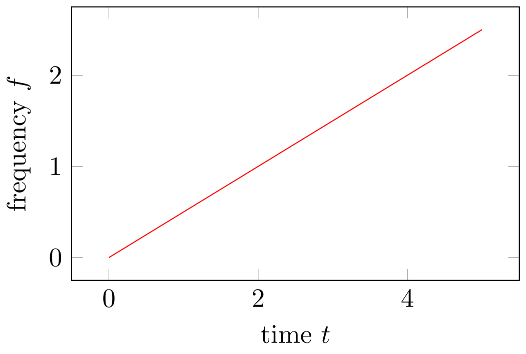
LoRa modulation

LoRa modulation is a proprietary protocol, hence we know only some characteristics of it. In particular we know that it's a **spread spectrum type of modulation**. Modulation means encoding the bits to send in a carrier, i.e. in an electromagnetic wave used to transport a string of bits. We can change modulation by changing the amplitude, the frequency or the phase of the carrier.

The carrier used in LoRa modulation is a **chirp signal** (or chirp symbol), i.e. a signal that increases (up-chirp) or decreases (down-chirp) its frequency in time. An example of chirp signal is showed in Figure 9.2.



(a) A up-chirp signal in the amplitude domain.



(b) A up-chirp signal in the frequency domain.

Figure 9.2: A up-chirp signal.

LoRa defines an alphabet that maps a bit (or a sequence of bit) to a chirp symbol. The sequence of bits is physically delivered as a sequence of chirp signals, each of which represents a bit of the sequence. For instance, say we want to use two bits for each symbol, hence we have to map the sequences 00, 01, 10 and 11 to four different chirp signals. The chirp symbols are obtained adding discontinuities to the chirp in the frequency domain (it's easy to visualise it in the frequency domain, however, we can always go back to the amplitude representation). An example of a symbol is shown in Figure 9.3.

This type of modulation is spread-spectrum because the physically transmitted signal (i.e. the string of bits) has a spectrum (i.e. frequency representation) that is wider than the modulated signal (i.e. the original string of bits).

The advantages of LoRa modulation are

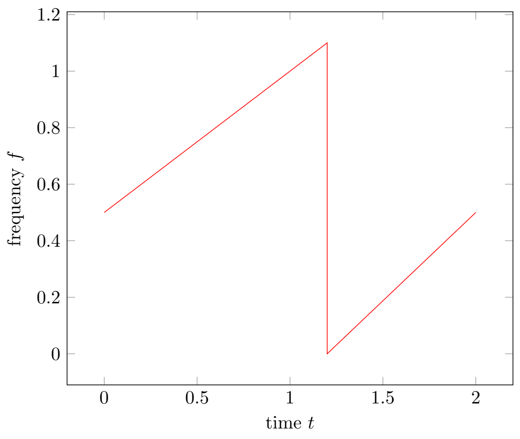


Figure 9.3: A symbol obtained from the chirp carrier.

- It's more robust to noise, since it has a wide spectrum. This type of modulation is even more effective against noise if it is frequency selective.

Chip rate and spreading factor When designing a LoRaWAN communication system we can tune two parameters:

- The **spreading factor** SF. The spreading factor represents the number of bits encoded in a chirp, hence the number of different chirp symbols. Put it in another way, it tells how many chips happen in a single bit time. In other words, the spreading factor is a parameter that tells how spread is the spectrum of the modulated signal in the frequency domain. LoRa defines a finite number of spreading factors (and bandwidths) values to choose from. The spreading factor is decided by the network server.
- The **coding rate** CR. The coding rate defines how many bits are transferred for each frame for protecting the transmission itself. Basically, it tells how many overhead bits are used to check the integrity of the frame.

These parameters can be used to compute the nominal bit rate

$$R_b = SF \frac{\frac{4}{4+CR}}{\frac{2^{SF}}{BW}} = SF \frac{4 \cdot BW}{2^{SF} \cdot (4 + CR)} \quad [bit/s]$$

As we can notice

- If we want an high bit rate, we have to choose a low spreading factor because R_b is a decreasing function with respect to the spreading factor SF (notice that SF appears as the exponent of a power in the denominator of the function).
- If we want high reliability, we have to choose an high spreading factor. In particular, increasing the spreading factor, lowers the sensitivity (Definition 5) hence allows the receiver to demodulate a signal with a lower power.

The chip rate is equivalent to the bandwidth of the channel used for transferring data

$$R_c = BW$$

LoRa MAC

The LoRa MAC layer handles the access to gateways. In particular LoRa defines three different ways to handle access control

- **Class A**. Class A provides more uplink than downlink. This means that most of the communication channel is used for uplink communication. To obtain this behaviour, downlink slots are made available only after an uplink transmission. In practice, this means that a mote can receive a packet only after sending one. In Class A, access control is managed using an ALOHA-like scheme.
- **Class B**. Class B devices provide a more balanced approach, in fact a device has more slots for receiving data after an uplink transmission. In other words, a device can receive more data after sending.
- **Class C**. Class C devices provide a fully balanced channel for uplink and downlink.

These classes can be evaluated in terms of download latency and energy consumption. In particular,

- Class A is the one that consumes less energy, followed by Class B and Class C.
- Class C is the one with less download latency, followed by Class B and Class A (i.e. Class A has the highest download latency).

This difference is explained by the fact that, since Class C devices have the an almost continuous uplink and downlink traffic, they have to be awake for most of the time, hence consuming more energy. On the other hand, Class A devices only have a limited number of downlink slots, hence they spend more time sleeping and consume less energy. Furthermore, Class A devices have to wait until a packet is sent, before receiving one, hence increasing download latency. That being said, Class A is the most used class nowadays.

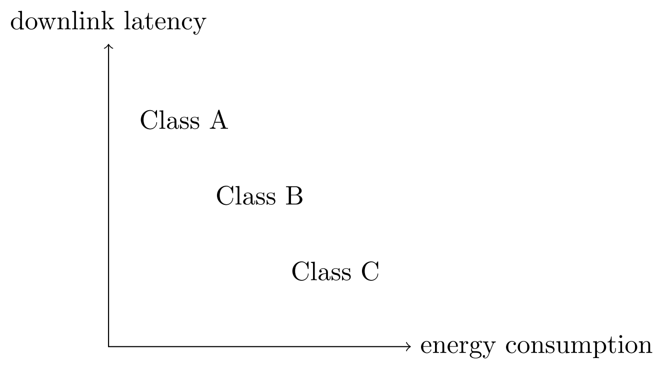


Figure 9.4: Evaluation of the LoRa MAC access control classes.

Class A protocol Class A devices use the following protocol to send and receive packets

1. A device sends a message as soon as it's ready (i.e. asynchronously).

2. After transmission is finished, a device has two time intervals (or slots) in which it can accept packets coming from the network. In other words, in this slots the network can send packets to the device. These time intervals are predefined by the standard and happen at predefined delays from transmission.

Notice that, in the time between transmission and the first slot and between the first and second slot, the mote is sleeping to consume less energy. A diagram that shows this interaction is shown in Figure 9.5.

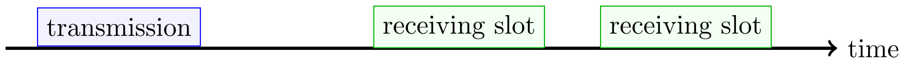


Figure 9.5: Class A transmission protocol.

9.1.3 Frames

Physical layer

At the physical layer, a frame is made of

- A **preamble**. A preamble is a known string of bits used to detect a transmission. This preamble is used by motes in the receiving slot, in fact when a device detects the preamble in the receiving slot, it starts listening and storing the whole message.
- A **physical header** (PHDR). The header contains information about the physical transmission.
- A **physical header CRC** (PHDR_CRC). The header CRC is used for sanity check.
- The **payload**. The payload of the physical frame is the frame of the upper layer (i.e. of the LoRa MAC layer).
- A **CRC**. The CRC is used to check the integrity of the frame.

LoRa MAC layer

At the LoRa MAC layer, a frame is made of

- A **MAC header** (MHDR). The header contains
  - The message type (e.g. acknowledged or not).
  - The message format.
- The **payload**.
- A **MAC Integrity Check** field. The MIC field is used to check the integrity of the frame.

Frame

The payload of the MAC layer is made of

- The **frame header** (FHDR). The header contains
  - The address (4 bytes) of the end device that is sending the message.
  - The Frame Control field to manage adaptive data rate, ACK and MAC commands.
  - The Frame Counter that counts the number of transmitted frames.
  - The Frame Options field containing MAC commands.
- The **Frame Port** (FPort). The Frame Port is used to identify the type of application that is using the frame. This port number is the same used in TCP to specify the application (e.g. 80 for HTTP).
- The **application payload**.

Notice that the frame doesn't contain the destination address because the communication is broadcast.

9.1.4 Conflict handling

Conflicts are handled by LoRaWAN using ALOHA. In particular, a device can send a message whenever it wants and, when a conflict happens, it picks a random delay and waits that time before sending the message again. In practice, when a message is sent, the sender waits a some time (randomly chosen) and if in this interval of time no acknowledgement comes by the receiver (i.e. the device the message is directed to) of the message, then the message is sent again. The acknowledgment interval (i.e. the time waited by the sender before sending the message again) is randomly picked between 1 and 3 seconds.

Notice that, differently from CAN bus, the device doesn't sense the channel before sending a message. ALOHA provides worst performances with respect to CSMA-CA but it also has some advantages, in particular

- It's much simpler (e.g. from the software point of view).
- Sensing requires the radio to be active, hence ALOHA consumes less energy.
- Carrier sensing might fail, since we are considering very long links and the waves take time to propagate. Basically the vulnerability period is very wide.

Performances

To study the performance of the ALOHA protocol in LoRaWAN we can consider the model in Figure 9.6, assuming that the network is made of n devices with no coordinator (since we are using ALOHA) and that two messages coming from different devices collide at the gateway. The signals used in the models are

- The incoming traffic S_{in} of the system.
- The traffic G in one snapshot (i.e. time instant) of the system. G is the sum of the transmissions and the retransmissions.

- The traffic S_{out} that has been handled by the system.
- The collisions $G - S_{out}$ that are, for each snapshot, the traffic that hasn't been handled and has to be summed back with the total traffic of the network.

We will also assume that

- The **traffic is stationary**, that is, all messages transmitted have to exit the system (i.e. $S_{in} = S_{out}$).
- The overall traffic S_{in} that enters the systems is **distributed as a Poisson** of parameter λ (i.e. the probability that a given number of events occur in an interval of time). Notice that, λ is the number of transmissions per second of the process.
- The **duration T of a transmission is fixed**.

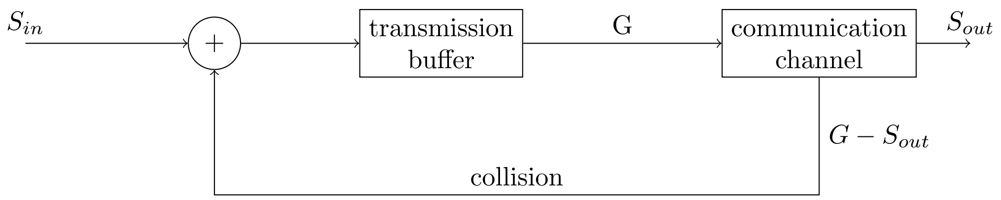


Figure 9.6: ALOHA communication model.

Given this model we have to understand what is the success probability, i.e. what is the probability that one single transmission goes through the channel without causing a collision. To compute this probability, say we have one transmission. According to ALOHA, this transmission is successful if no other transmission happens while it's passing through the channel (i.e. in the same interval T). Since the message transmission are modelled with a Poisson variable, then the probability that k messages arrive in a time period between $t - T$ and $t + T$ is

$$P(k, T) = \frac{\lambda^k \cdot e^{-G}}{k!}$$

where we are using G instead of λ because the latter is a rate. Since we want that 0 messages arrive in the period between $t - T$ and $t + T$ (hence is a period $2T$ long) we obtain

$$P_s = P(N(T - t, t + T) = 0) = \frac{\lambda^0 \cdot e^{-\lambda}}{0!} = e^{-2G}$$

where the number of packets sent G is

$$G = T \cdot \lambda$$

because we consider the traffic during a transmission interval (i.e., the rate at which packets arrive times the transmission interval). Since we want to consider all transmissions we have to multiply this probability for G, obtaining the throughput S of the system

$$S = G \cdot P_s = Ge^{-2G}$$

If we plot this function (Figure 9.7) we notice that the throughput increases with the traffic G, however, at a certain point it starts decreasing because when G is too big, many collisions happen

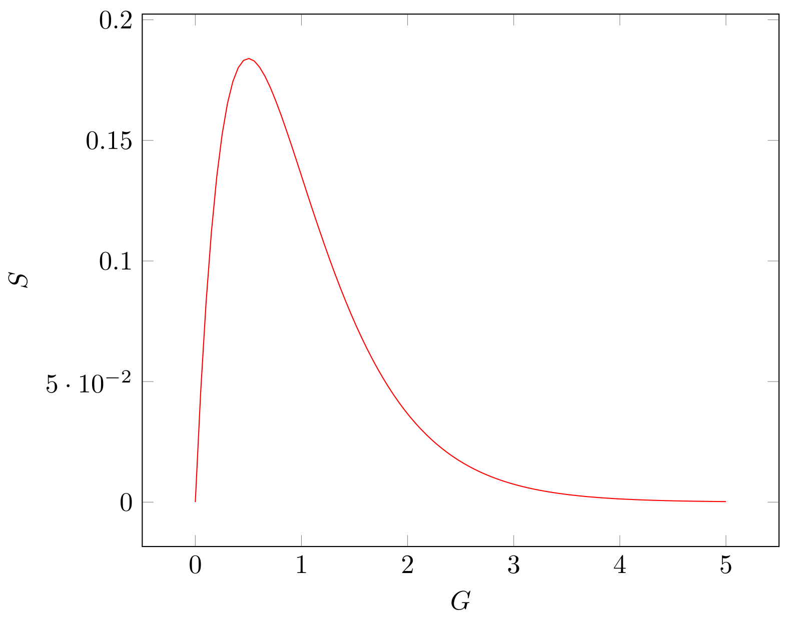


Figure 9.7: Throughput of a LoRaWAN system.

and the channel is saturated with re-transmissions. We also notice that the peak is very low, hence even in perfect conditions, only few transmissions (usually 3 out of 10) are successful. This means that ALOHA isn't too good in terms of throughput, however it's still used for its simplicity and because we can control in what region of the graph we work, namely in the left part, before the peak. This model doesn't consider one important feature of LoRaWAN, in fact we should consider the fact that the duration of the transmission can change. Assume that the system is made of two types of stations (station type 1 and station type 2) with different λ and T (i.e. λ_1, λ_2, T_1, T_2). In this conditions, we can write the probability that a message from station 1 successfully passes through the channel as the product of the probability that no message of type 1 and of type 2 arrives in the same time interval T_1. Since we are considering motes that send messages with a Poisson distribution, we the probability that no message of type i arrives in a time interval T_i is

$$\begin{aligned} P(k=0) &= \frac{(N_i \cdot \lambda_i \cdot T_i)^k e^{-N_i \cdot \lambda_i \cdot 2T_i}}{k!} \\ &= \frac{(N_i \cdot \lambda_i \cdot T_i)^0 e^{-N_i \cdot \lambda_i \cdot 2T_i}}{0!} \\ &= e^{-N_i \cdot \lambda_i \cdot T_i} \end{aligned}$$

If we replace T_i with $2T_1$ for motes of type 1 and $T_i = T_1 + T_2$ (because we have to consider the vulnerability window of both types) for motes of type 2 we obtain

$$P_{succ,1} = e^{-(N_1\lambda_1)\cdot 2T_1} \cdot e^{-N_2\lambda_2(T_1+T_2)} \tag{9.1}$$

Notice that, we have replaced λ in Equation A.3 with the total number of packets sent, which is $N_1 \cdot \lambda_1 \cdot 2T_1$ because λ_1 is a rate. Basically, we are using Equation A.4. Using the same reasoning we can write the probability of a success for a station of type 2 as

$$P_{succ,2} = e^{-(N_2\lambda_2)\cdot 2T_2} \cdot e^{-N_1\lambda_1(T_1+T_2)} \tag{9.2}$$

We can use these probability to compute the probability that a message collides as

$$P_{coll,1} = 1 - P_{succ,1}$$

and

$$P_{coll,2} = 1 - P_{succ,2}$$

Given the collision probability we can now say that the probability P that at least one message, or one of the two station collides is the weighted average of the probability that a message for station i collides. In formulas

$$P = \frac{\lambda_1 \cdot N_1}{\lambda_1 \cdot N_1 + \lambda_2 \cdot N_2} \cdot P_{coll,1} + \frac{\lambda_2 \cdot N_2}{\lambda_1 \cdot N_1 + \lambda_2 \cdot N_2} \cdot P_{coll,2}$$

This probability can be generalised to an arbitrary number L of station types as follows

$$P = \sum_{i=1}^{L} \frac{\lambda_i \cdot N_i}{\sum i = 1^L \lambda_i \cdot N_i} \cdot P_{coll,i}$$

where

$$P_{coll,i} = 1 - \prod_{k=1}^{L} e^{-N_k \cdot \lambda_k (T_i + T_k)}$$

Notice that, stations that require a longer transmission time affect more the performance of the system, worsening it, because the overlapping window is larger.

9.1.5 LoRaWAN ecosystem

The big advantage of LoRaWAN is that we can buy different LoRa components from different manufacturers and still get a working network. Furthermore, one can easily build a network of its own or connect to a network offered by a Public Network Operator.

9.2 Cellular

Cellular networks aren't designed for IoT application. In particular, cellular networks are throughput bound, i.e. their advancements aim at increasing the throughput, however IoT applications don't need an high throughput (but low latency and energy consumption). Cellular networks also assume to be used by high-end devices (like smartphones) hence they don't care about energy consumption, which is super important in the IoT field.

To solve these problems, cellular operators have applied patches to the cellular technology to make it compliant with the requirements and the constraints of IoT systems. In particular, the main patches are

- **LTE-CatNB1** (where NB stands for Narrow Band). This technology is the newest and is the real competitor of LoRaWAN. In particular, it removes the unnecessary complexity of some functionalities offered by LTE and uses narrow band channels. This technology ensures lower throughput with respect to the others, but the hardware used is the cheapest.
- **LTE-CatM1**.
- **LTE-Cat1**. This technology is very similar to LTE, hence it offers an high throughput, however the hardware cost is quite high with respect to the other patches.

9.2.1 5G

The 5G technology, besides increasing the throughput, also includes some features that fit well with IoT. In particular,

- 5G enables millimeter wave access (i.e. high frequency communication) thus creating uplink and downlink pipes with high throughput.
- 5G is more dense, namely there are more antennas with respect to previous technologies. This allows to obtain a more capillary coverage, which is fundamental to reach the field devices in IoT.
- 5G supports mobile edge computing. Basically, cloud resources are brought close to the edge devices to elaborate data coming from motes closer to the source.

These features (together with all the other features of 5G) allow to obtain an high throughput, low latency network that suits with industrial services like autonomous robots and vehicles.

Chapter 10

Short range wireless communication technologies

Short range communication technologies allow devices to communicate when they are close (some meters) one to the other. When talking about short range communication technologies we should differentiate between

- **Active devices**. Active devices use their own energy, usually coming from batteries, to communicate. For instance, motes are active devices.
- **Passive devices**. Passive devices obtain energy from another device. Basically they don't run on batteries and should be close to a device that can pass them the energy require to power them up and communicate. An example of such devices is RFID cards.

Gateway problem Let us consider active devices. Many technologies are available in the field of short range communication, hence we need a different gateway for each technologies (because gateways only speak one technology). Moreover, gateways are usually small, hence we need a lot of them. This means that, since IoT network usually exploits different communication technologies, a network is usually filled with many different gateways. This means that it requires a lot of planning to decide what gateways should be used and where to put them. This problem is called gateway problem and, in recent years, has been solved using gateways that can handle multiple technologies.

Classifying short range communication technologies Communication technologies can be classified in three ways

- Proprietary or open-source.
- Application specific or application agnostic.
- IP compliant or non-IP compliant.

The two main technologies we are going to analyse are **ZigBee** and **6LowPAN**. The former is non-IP compliant, hence it defines its own stack while the latter is IP compliant. However, both technologies share the same lower layers (physical and MAC) and start using different protocols only above such layers. Another difference between the two technologies is that the 6LowPAN doesn't define a specific network layer, beacuse it uses the Internet application layer. On the other hand, ZigBee defines its own application layer because it doesn't use Internet's stack.

Chapter 11

Zigbee

ZigBee is short range communication technology widely used in IoT systems. ZigBee's main features, or requirements, are

- It has to be able to **run on limited hardware** both in terms of cost (i.e. cheap hardware) and capabilities (i.e. hardware with low processing power).
- **It works in a short range**, usually around 10 meters.
- It has to be **energy efficient**. This requirement is very important for IoT applications.
- It has to **provide low latency**.

11.1 Stack

ZigBee's stack is made of 6 levels

- The **physical layer**. The physical layer is the standardised 802.15.4.
- The **Medium Access Control (MAC) layer**. The MAC layer is responsible for formatting transmission characteristics of the signal and controlling the shared medium of communication.
- The **network layer**. The network layer defines the topology (i.e. the shape) of the network. Moreover, this level also provides addressing functionalities (i.e. assigns names to the devices connected to the network). Finally, this level provides also routing for the packets on the network, in fact, being a multi-hop transmission network, the network layer decides which sequence of devices (i.e. the path) a packet has to traverse to reach its destination.
- The **security layer**. The security layer offers security functionalities like authentication and encryption.
- The **application interface layer**. The application interface layer defines the guidelines on how an application should be designed and composed at the application layer. In other words, this layer defines an interface that can be used by developers to build an application.
- The **application layer**. The application layer uses the interface provided at the application interface layer (i.e. uses the guidelines) to build applications for a ZigBee network.

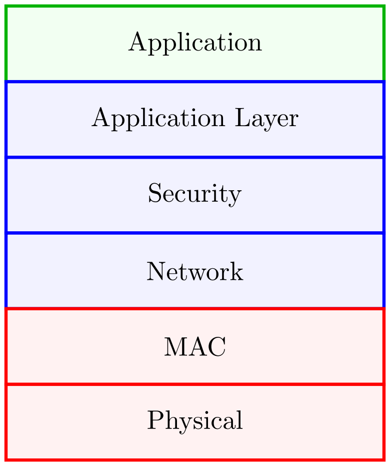


Figure 11.1: ZigBee's protocol stack. Red represents a layer implemented in hardware, blue represents a layer implemented in firmware and green represents a layer implemented in software.

A representation of the stack is shown in Figure 11.1. In particular

- The physical and MAC layers are implemented in **hardware**.
- The network, security and application interface layers are implemented in **firmware**.
- The application layer is implemented in **software**.

11.2 Physical layer - 802.15.4

11.2.1 Device classification

ZigBee uses, at the physical layer, the IEEE 802.15.4 for wireless communication. This standard defines three types of devices,

- **Full Function Device**s (FFDs).
- **Reduced Function Device**s (RFDs).
- **PAN Coordinator**s (PCs). A PAN Coordinator is more a logical role assigned to a devices than a type of device.

Notice that this characterisation of devices isn't present in the classical Internet, in which all devices can have the same role.

Full Function Devices

FFDs are very refined devices that can perform most of the functionalities offered by a ZigBee network. In particular, FFDs can

- Act as normal routers.

- Be the source or destination of a message.
- Relay messages.
- Communicate directly with other FFDs.

Given the number of functionalities offered by these devices (hence the high energy consumption), FFDs usually run on direct power supply.

Reduced Function Devices

RFDs can do only a limited set of functions offered by the ZigBee stack. In particular, RFDs

- Can't route traffic.
- Can only be source or destination of a message.
- Can't communicate directly with other RFDs. This means that they have to go through a FFD to communicate with a RFD.

RFDs are usually motes, hence they can run on batteries since their energy consumption is reduced (they can only perform a limited set of functions).

PAN Coordinators

A PAN Coordinator is responsible for managing all the network and the communication between nodes. For instance it manages device association and traffic (a bit like a LoRaWAN gateway). PAN Coordinator is a logical role assigned to one or more FFDs of the network.

11.2.2 Extensions

The standard 802.15.4 can be extended to modify the functionalities offered by the base version. Each extension adds and modifies some functionalities to fit a specific application vertical (i.e., an application field). Some well known extensions are,

- **802.15.4e**. This extension adds slotted channel access.
- **802.15.4g**. This extension is thought for IoT in smart utility applications (e.g., water consumption monitoring).
- **802.15.4k**. This extension is thought for applications that monitor critical infrastructures.

11.2.3 Network topologies

The network layer defines different types of topologies, however, at the physical layer we can already defines the shape of the network without considering addressing. The most used network topologies in ZigBee are

- The **star topology**. In the star topology, the PAN Coordinator is in the middle and all other devices, both FFDs and RFDs are connected to the PC. Notice that, this configuration doesn't allow cross traffic between the devices (even between two FFDs) because all traffic goes from and to the PAN Coordinator. An example of star topology is shown in Figure 11.2.

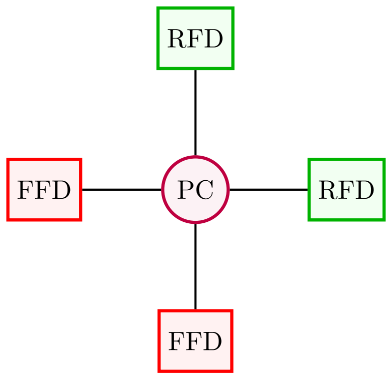


Figure 11.2: A star topology.

- The **mesh topology**. In the mesh topology, all devices are connected one to the other without a specific schema. This means that, in this case cross traffic is allowed. That being said, direct communication between RFDs isn't allowed, hence communication has to pass through a FFD. An example of mesh topology is shown in Figure 11.3.

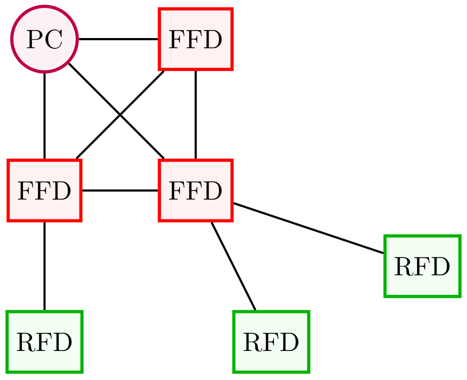


Figure 11.3: A mesh topology.

- The **cluster-tree topology**. The cluster-tree topology is an extension of standard 802.15.4. This topology is multi-hop, hence it needs routing (provided by the network layer). In this topology we have a PAN Coordinator at the centre and some FFDs connected to it. Each FFD connected to the PC is the centre of a new star in which, the connected FFDs (if present) can be a centre of a start themselves. This type of topology is often used when the network has to conver larger areas. An example of cluster-tree topology is shown in Figure 11.4.

11.2.4 Physical layer services

ZigBee's physical layer is implemented in hardware and offers the following services

- **Activation and deactivation of the transceiver**. This functionality is used to implement duty cycles.

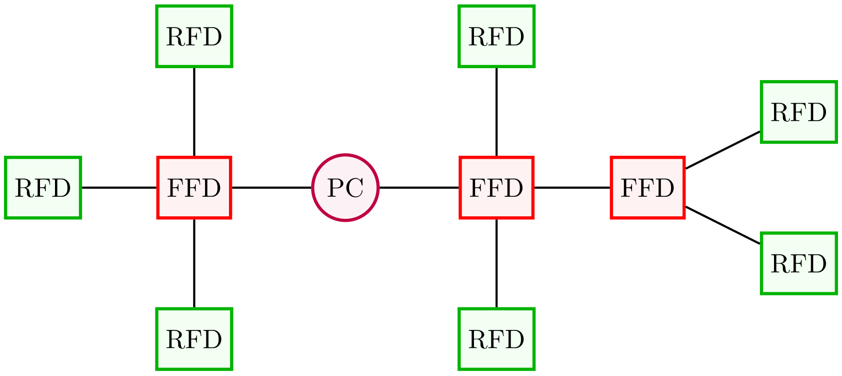


Figure 11.4: A mesh topology.

- **Energy detection**. This functionality is used to select the best channel for transmission. In particular we can check, for each transmission channel, the energy level of that channel to find the one that consumes less.
- **Link Quality Indicator** (LQI). The LQI parameter provides to the receiver a quality measure of the received packets.
- **Clear Channel Assessment** (CCA). This functionality allows us to sense the communication channel and measure if there is any activity on it. For this reason, we can use the CCA for Carrier Sense Multiple-Access with Collision Avoidance (CSMA-CA).
- **Channel frequency selection**. A ZigBee transceiver is able to transmit packets at different frequencies, hence this functionality allows us to select one of these frequencies.
- **Data transmission and reception**.

Frequency bands

The standard 802.15.4 defines 3 frequency bands, all of which are unlicensed, on which data can be transmitted,

- A band with carrier at **868 MHz** on which 3 channels are defined.
- A band with carrier at **915 MHz** on which 30 channels are defined.
- A band with carrier at **2.4 GHz** on which 16 channels are defined. This band is also used by Wi-Fi.

A summary of the characteristic of each band is shown in Table 11.1.

Frame

The frame at the physical level is made of

- A **preamble** used to synchronise the sender and the receiver. In particular, the preamble is used to track the clock of the sender.

Carrier (MHz)	Frequency band (MHz)	Chip rate (kchip/s)	Modulation	Bit Rate (kb/s)	Symbol rate (ksymbol/s)	Symbols
868	868-868.8	300	BPSK	20	20	Binary
915	902-928	600	BPSK	40	40	Binary
868	868-868.8	400	ASK	250	12.5	20-bit PSSS
915	902-928	1600	ASK	250	25	5-bit PSSS
868	868-868.8	400	O-QPSK	100	25	16-ary Orthogonal
915	902-928	1000	O-QPSK	250	62.5	16-ary Orthogonal
2450	2400-2483.5	2000	O-QPSK	250	62.5	16-ary Orthogonal

Table 11.1: Characteristics of each band offered by the standard 802.15.4.

- A **Start of Frame Delimiter** (SFD). The SFD is a known sequence of bits that is used to indicate that the frame is starting.
- A **frame length field** used to indicate the length, in number of octets, of the payload. The length of the payload can be between 9 and 127 octets for MAC data-frames (remember that the payload of a level is the frame of the layer above).
- The **payload**.

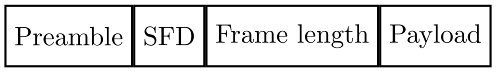


Figure 11.5: The format of a ZigBee frame for the physical layer.

11.3 MAC layer

The MAC layer is responsible for managing access to the communication channel and the communication medium. The functionalities offered by this layer are

- **Channel access management**.
- **Frame validation**.
- **Network association and disassociation**.
- **Frame delivery acknowledgement**.
- **Providing interfaces for the security layer**.
- **Beacon management** (more on this later).
- **GTS management** (more on this later).

11.3.1 Beacon operation modes

A ZigBee network can operate in two modes, that change the behaviour of the network

- **Beacon-enabled mode**. In beacon-enabled mode, the PAN Coordinator periodically sends beacons to the other devices of the networks. A beacon is a packet that contains information for the nodes regarding when to start transmitting or receiving and how the duty cycle should work (i.e. how long the node should stay awake). This mode is usually adopted in star topologies. Since the PAN Coordinator can send out some information to synchronise the devices, the medium access control is handled in hybrid mode, using both CSMA-CA and scheduled transmission.
- **Non-beacon mode**. In non-beacon mode, the PAN coordinator doesn't use beacons to handle medium access control, hence the only way to use the channel is to adopt random access control with CSMA-CA.

Beacon-enabled medium access control

When a ZigBee network operates in beacon-enabled mode, medium access control is handled in hybrid mode, i.e. using

- Carrier Sensing Multiple Access with Collision Avoidance.
- Scheduled transmission.

In other words, either the controller tells the devices, using beacon messages, when they should send or listen for incoming messages, or the devices use CSMA.

In particular, time is slotted and each slot can host incoming or outgoing transmissions. Periodically, the PAN Coordinator sends out beacon messages that delimit the beginning of a duty cycle (and the end of the previous duty cycle). The time interval between two beacon messages is called **beacon interval** or **superframe** and it's divided in two parts

- The **active part**. During the active part all devices on the network are active. The active part is further divided in
  - **Collision Access Part** (CAP). The CAP is divided in slots in which the devices can compete in a random way to access the medium using CSMA-CA.
  - **Collision Free Part** (CFP). During the CFP, the access to the medium is totally scheduled. More precisely, the CFP interval is divided in Guaranteed Time Slots (GTS) in which the devices can exclusively send messages.
- The **inactive part**. During the inactive part all devices of the network are sleeping.

The parameters used to divide each part (i.e. the duration of the different parts and of the time slots and the devices that can send messages in the time slots) are written in the beacon message that marks the beginning of the duty cycle. Also note that a beacon frame is sent in an interval slot.

Rates An important characteristic of a frame is the **one slot per beacon interval rate** r, which is different from the one offered by the communication channel, i.e., the nominal rate R. More precisely

- The **nominal rate** is the quantity of data that each device is capable of sending in a second. Since time is slotted, we can compute this rate as

$$R = \frac{L}{T_{slot}}$$

where L is the length of a packet sent in a slot and T_{slot} is the duration of a slot.

- The **one slot per beacon rate** is the quantity of data that a device can send in a beacon interval if we assign to it only a slot (assuming that the length of a packet in a slot is L bits).

$$r = \frac{L}{BI}$$

Carrier Sensing Multiple-Access with Collision Avoidance

During the CAP part of the active interval, the devices use CSMA-CA, hence it's important to analyse in details how it works for ZigBee. Each device maintains 3 variables

- The **Number of Back-offs** NB. The number of back-offs is the number of consecutive times that the device has to give-up sending a message because the channel is occupied. This variable is initialised to 0 when a new transmission begins.
- The **Contention Window** CW. The contention window is the number of consecutive back-off periods (one back-off period is the duration of 20 symbols) that need to be clear of channel activity before the transmission can commence. Basically the device can send a message if the channel is free for CW consecutive back-off periods. The initial value of this variable is 2.
- The **Back-off Exponent** BE. The back-off exponent says how long a device should wait after finding out that the channel was busy before trying to send the message again.

These variables are used, by a device that wants to send a message, in the following procedure,

1. The device gets a random number D between 0 and $2^{BE} - 1$.
2. The device waits for D back-off periods.
3. After the delay expires, the devices does carrier sensing for CW consecutive time slots.
4. If the channel was free for CW consecutive time slots, the message is sent and the devices waits for the acknowledgement.
5. If the channel wasn't free, BE and NB are incremented and the device goes back to step 1.
6. If $NB > NB_{max}$ then the channel has been found busy too many times and the packet is discarded. The device will try to send the packet in the following duty cycles.

If a collision happens, the procedure is restarts from the beginning. A diagram of this procedure is shown in Figure 11.6.

This procedure must start and end withing the CAP interval. Notice that, this procedure isn't used for beacons and acknowledgement frames (i.e. they have higher priority than normal messages).

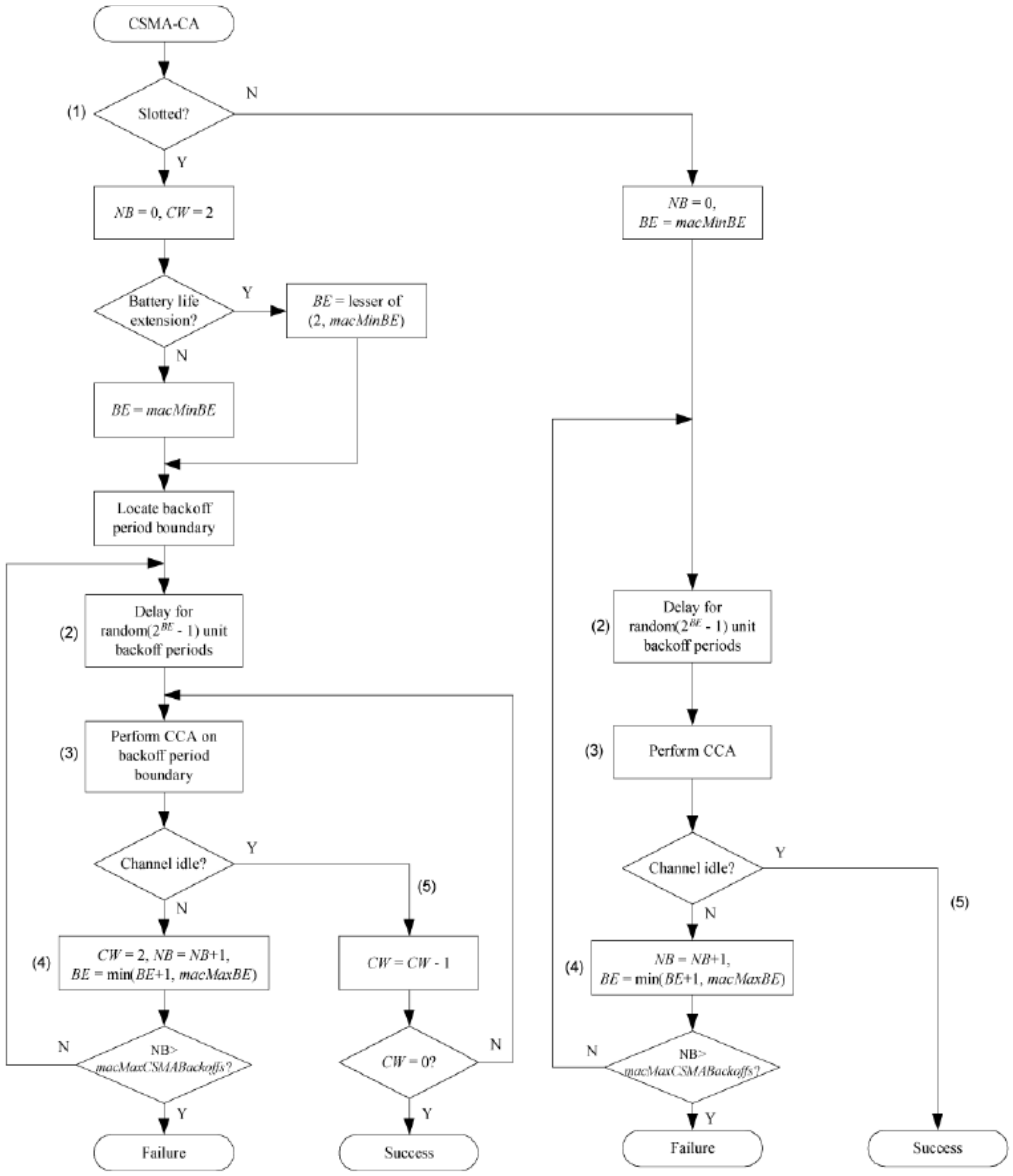

CSMA-CA
(1)
Slotted?
N
Y
NB = 0, CW = 2
NB = 0, BE = macMinBE
Battery life extension?
Y
BE = lesser of (2, macMinBE)
N
BE = macMinBE
Locate backoff period boundary
(2)
Delay for random(2^BE - 1) unit backoff periods
(2)
Delay for random(2^BE - 1) unit backoff periods
(3)
Perform CCA on backoff period boundary
(3)
Perform CCA
Channel idle?
Y
(5)
N
Channel idle?
Y
(5)
N
(4)
CW = 2, NB = NB+1, BE = min(BE+1, macMaxBE)
CW = CW - 1
(4)
NB = NB+1, BE = min(BE+1, macMaxBE)
N
NB> macMaxCSMABackoffs?
Y
CW = 0?
N
Y
N
NB> macMaxCSMABackoffs?
Y
Failure
Success
Failure
Success

Figure 11.6: ZigBee's CSMA procedure.

11.3.2 MAC Frame

The MAC frame is made of

- The **frame control** field (2 octets).
- The **sequence number** of the frame (1 octet).
- The **destination's PAN identifier** (0 or 2 octets).
- The **destination address** (0, 2 or 8 octets).
- The **source's PAN identifier** (0 or 2 octets).
- The **source address** (0, 2 or 8 octets).
- The **auxiliary security header** (0, 5, 6, 10 or 14 octets).
- The **payload**.
- The **Frame Check Sequence** (2 octets).

Beacon frame

The beacon frame is a particular type of MAC frame that contains

- The **frame control** field (2 octets).
- The **sequence number** of the frame (1 octet).
- The **source's PAN identifier** (0 or 2 octets).
- The **source address** (0, 2 or 8 octets).
- The **auxiliary security header** (0, 5, 6, 10 or 14 octets).
- The **superframe specification** (2 octets). The superframe specification defines the length and structure of the superframe (i.e. of the beacon interval). In particular this field contains
  - The Beacon Order (BO). The BO defines the length of the Beacon Interval (BI).
    $$BI = aBaseSuperframeDuration \cdot 2^{BO} \text{ [symbols]}$$
  - Superframe Order (SO). The SO defines the Superframe Duration (SD).
    $$SD = aBaseSuperframeDuration \cdot 2^{SO} \text{ [symbols]}$$

  Where $aBaseSuperframeDuration = 16$ slots $\cdot$ 60 symbols.
- The **GTS fields**. The GTS fields are used to specify which devices can send or receive messages in each slot of the Collision Free Part.
- The **Pending address fields**.
- The **beacon payload**.
- The **Frame Check Sequence** (2 octets).

Notice that, since the beacon message is sent to all devices, we only have to specify the source addresses.

11.4 Network layer

The network layer is used to route packets through the network.

11.4.1 Functionalities

The functionalities offered by this layer are:

- **Addressing**. Addressing gives unique names to the devices.
- **Routing**. Routing offers the procedures to move packets on multiple hop.
- **Reception control**. Reception control is the ability for a device to control when the receiver is activated and for how long, enabling MAC sub-layer synchronization or direct reception.
- **Neighbor discovery**. Neighbor discovery is the ability to discover, record, and report information pertaining to the one-hop neighbors of a device.
- **Configuring a new device**. Configuring a new device is the ability to sufficiently configure the stack for operation as required.
- **Starting a network**. Starting a network is the ability to establish a new network.
- **Joining, rejoining and leaving a network**. This is the ability to join, rejoin or leave a network as well as the ability of a ZigBee coordinator or ZigBee router to request that a device leave the network.

11.4.2 Node hierarchy

The same hierarchical distinction done among Full Function Devices, Reduced Function Devices and PAN coordinators can be done for the devices at the network layer. In particular, we can divide nodes in

- **ZigBee coordinators**. A ZigBee coordinator is the equivalent of a PAN coordinator. A PAN coordinator is also a ZigBee coordinator and therefore a FFD (since PAN coordinators are FFDs).
- **ZigBee routers**. A ZigBee router routes packets through the network and is a FFD.
- **End devices**. End devices generate and receive packets and can be both FFDs and RFDs.

11.4.3 Routing

Routing can be handled with two different approaches:

- Ad-hoc On-demand Distance Vectors (AODVs).
- Cluster tree algorithm.

Cluster tree algorithm

The cluster tree algorithm creates a tree rooted in the ZigBee coordinator where each node represents a node of the network. When a node joins, it is added to the tree, hence the cluster tree is created dynamically. Routers can be both inner nodes or leaves but end-devices can be only leaves. For each level, the number of children for each node is fixed.

The tree is created by a FFD that chooses a channel and sets its own Network Address to 0 to identify itself as the coordinator. After this initiation phase, every other node (either FFD or RFD) can join the network contacting the PAN coordinator. Upon receiving a join request, the PAN coordinator assigns an address to the node. In particular,

- If the node is a **router** then coordinator assigns a range of addresses because a router can be the root of a sub-tree, hence it has to handle all the addresses of the nodes rooted in it. The router keeps the first address of the range for itself and assigns the remaining addresses to its children. A router at depth d is assigned A nodes

$$A(d) = \begin{cases} 1 + D_m + R_m & \text{if } d = L_m - 1 \\ 1 + D_m + R_m A(d+1) & \text{otherwise} \end{cases}$$

  where D_m is the number of end devices connected to the router, R_m is the number of routers connected to the router and L_m is the number of levels. These parameters are decided by the ZigBee coordinator.

- If the node is an **end-device** it's assigned a single device.

Having assigned to each router all the addresses of the nodes in its sub-tree, each node knows where to send a message. In other words, when we build the tree, we are also building the routing tables.

When a packet has to be routed by a router R to an address `addr`,

- If the node with address `addr` is directly connected to R, the packet is sent to that node.
- If the node with address `addr` is in the sub-tree rooted in R, the packet is sent to the next hop according to the routing table.
- Otherwise, the packet is sent to the parent node.

The cluster tree algorithm is very easy to implement and use, however routing is not optimal because in some cases a packet has to go all the way to the PAN coordinator and than travel downwards to a leaf. Moreover, this algorithm is quality agnostic, in fact the routing doesn't consider the quality of the links.

Ad-hoc On-demand Distance Vectors

AODV is applied whenever a router has to manage a packet whose next hop is not known. This explains why this protocol is called on-demand: the protocol is executed only when needed by a router.

Distance vectors AODV uses distance vectors, that is vectors that contain all known destinations and the cost to reach them. In particular, each router has two routing tables

- A **routing table** that contains the next hop for each known destination.

- A **routing discovery table** that contains all ongoing discovery processes. In particular, this table contains
  - The **id of the discovery session** randomly generated by the initiator of the discovery process.
  - The **source address**, i.e., the address of the node that initiated the discovery process.
  - The **sender address**, i.e., the address of the node from which the discovery message was sent.
  - The **forward cost**, i.e., the accumulated cost to go from the issuer to the current node.
  - The **residual cost**, i.e., the accumulated cost to go from the current node to the issuer.

When a router has to send a packet whose next hop is unknown, it initiate the discovery process. The router broadcasts to its neighbours a packet to request if they know the unknown address. The request is flooded through the network until the node is found. At this point the request is sent back, in unicast through the paths used to reach the unknown node, to the node that initiated the discovery process. Every time a node receives a discovery request, it adds an entry in the routing discovery table and when the request is sent back by the unknown node,

1. The routing table is updated adding the unknown node.
2. The residual cost in the discovery table is updated (initially set to ∞ because unknown).
3. The response is sent to the address in the sender address field of the discovery table. If the node is the one that initiated the request, the message isn't relayed.

Routing cost An important part of the distance vector protocol is the computing of the cost $C(P)$ to go from one node to another node through path P made of $L-1$ hops.

$$C(P) = \sum_{i=1}^{L-1} C(i, i+1)$$

The cost can be

- The number of hops to reach the destination.

$$C(i, i+1) = 1$$

- The reliability of the links from source to destination. The reliability can be computed using the packet reception rate of the links. Notice that the reception rate is not part of the ZigBee protocol.

$$C(i, i+1) = \min\left(7, round\left(\frac{1}{p_{i,i+1}^4}\right)\right)$$

  where $p_{i,i+1}$ is the reception rate of the link from node i to node $i+1$.

11.4.4 Packet format

A network packet is composed of

- A **frame control** field that contain the type of the frame and the route discovery information.
- The **destination address**.
- The **source address**.
- The **broadcast radius**. The broadcast status is the maximum number of hops that the packet can do before being dropped. Basically, it represents the Time To Leave and the lifetime of the packet.
- The **broadcast sequence number** to identify the route discovery requests.
- The **payload**.

A graphical representation of a packet is shown in Figure 11.7.

Frame control	Destination address	Source address	Broadcast radius	Broadcast sequence number	Payload

Figure 11.7: The format of a ZigBee packet.

11.5 Application profiles

ZigBee provides some guidelines under the form of interfaces that have to be used to build ZigBee applications. These guidelines, called **profiles**, are similar to libraries and are specific to some type of application. Profiles use an approach similar to Object Oriented Programming, in fact profiles defines attributes and methods that have to be used by the developers.

11.6 Network formation

When a mote switches on, it has to decide to which network it should connect. In particular, the connection to a network can be divided in two phases

- The **network discovery phase**. In the network discovery phase, a mote checks what networks are available around it and decides which network to join.
- The **association phase**. In the association phase, a mote asks the PAN coordinator to join the network chosen in the discovery phase.

11.6.1 Network discovery

In the network discovery phase, a mote scan the networks around it. There exists two types of scanning both based on the fact that a node can be in a network only if it can receive beacon messages from the PAN coordinator. The types of scanning are

- **Active scanning**.
- **Passive scanning**.

Active scanning

In active scanning, a mote uses a **trigger message** to actively request a beacon message to the PAN coordinator. Upon receiving a trigger message, the PAN coordinator sends a beacon message, even outside its periodicity (i.e., even not at the beginning of a duty cycle) to speed up the joining process. This procedure is repeated with all channels and at the end the mote decides to which network it should connect, among those from which the mote has received a beacon message. A mote can use different policies to decide to which network it should connect, in particular the mote can choose

- The least loaded network.
- The network with the strongest power.
- The network with the most appropriate duration of the duty cycle or of the active part. Both information are contained in the beacon message.
- The network with the fewer nodes already connected. This information can be obtained from the slots assigned in the beacon.

In a nutshell, a mote can choose a network depending on the physical measure of the connection or the actual structure of the network.

Passive scanning

Passive scanning works exactly like active scanning, the only difference being that the mote doesn't send a trigger message and only waits for a beacon message from the PAN coordinator (since beacon messages are sent in broadcast).

11.6.2 Network association

After deciding which network to join, the actual join procedure has to be initiated. The network association is carried out using a 4-ways handshake. In particular,

1. The mote sends an `Association request` message to the PAN coordinator.
2. The PAN coordinator acknowledges the request with an `Association acknowledgement` message.
3. The PAN coordinator sends an `Association response` message to the mote.
4. The mote acknowledges the response with an `Association acknowledgement` message.

Chapter 12

6LoWPAN

12.1 Introduction

6LoWPAN is an adaptation of IPv6 designed to work in constrained environments (e.g., with respect to energy consumption) like in the case of IoT systems. In order to fit IPv6 to IoT we have to modify a bit the stack protocol, from the network layer below (i.e., network, data-link and physical layers), to make it compliant with the constraints of the IoT field. The main differences with a normal Internet stack are

- At the **transport layer**, 6LoWPAN uses only UDP (or better said, it doesn't use TCP). This choice is motivated by the fact that, UDP is much more lightweight than TCP in terms of header and computation overhead hance it consumes less power than TCP. In particular, UDP is purely best-effort (i.e., we don't have frame re-transmission), connection-less and offers less services.
- At the **network layer**, 6LowPAN uses only IPv6 with some adaptations. Also the routing protocol has to be changed because the standard IPv6 routing protocol can't be used (more on this later).
- At the **data-link and physical layers**, 6LoWPAN uses the IEEE 802.15.4 standard (like ZigBee).

Since neither UDP nor 802.15.4 (at lower layers) offer a reliability service (it's best effort), an mote should implement it (the reliability functionality) at the application layer. Note that, 6LoWPAN is an open Internet standard.

12.1.1 Topologies

In 6LoWPAN network, we can find

- **Simple LoWPAN**s. A simple LoWPAN is a cluster of field nodes, also called field network, connected to an edge router, i.e., a simple router like the ones used in classical Internet (because 6LoWPAN uses the same network protocol, IPv6, as Internet).
- **Ad-hoc LoWPAN**s. An ad-hoc LoWPAN is a field network without a router. In an ad-hoc LoWPAN, field devices can only exchange data locally, without being connected to other LoWPANs.

- **Extended LoWPANs**. An extended LoWPAN is a set of simple LoWPANs which are partially overlapped. Namely, some motes belong to two different LoWPANs. Each simple LoWPAN still has its own edge router and the edge routers of the simple networks composing the extended LoWPAN are connected, via a backbone link to support cross traffic.

All edge routers can be connected to classical Internet routers to connect the field devices to the Internet. Figure 12.1 show an example of a 6LoWPAN architecture with a simple LoWPAN and an extended LoWPAN.

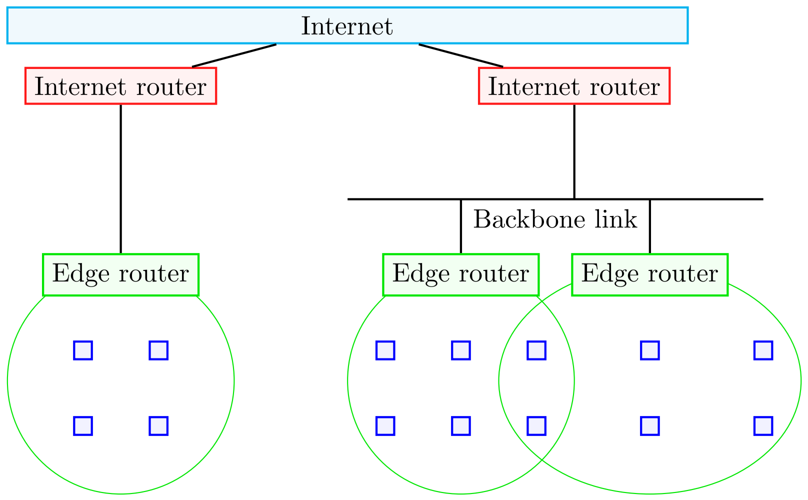


Figure 12.1: A 6LoWPAN topology with a simple LoWPAN (on the left) and an extended LoWPAN (on the right).

12.2 Network layer adaptations

At the network layer, 6LoWPAN defines two adaptations:

- **Header compression**. Header compression allows to reduce the size of the IPv6 header compressing, or sometimes removing, some fields. Compressing the header allows to reduce the IPv6 overhead.
- **Fragmentation**. Fragmentation allows to take large UDP segments and chunk them up to fit in the 802.15.4 MTU (which is 127 bytes). Fragmentation is provided by IPv4 but not by IPv6, hence it's up to 6LoWPAN to implement it.

12.2.1 IPv6

Addresses

IP version 6 is the evolution of IP version 4. The main difference is that IPv6 redesigns the addresses of the devices. In particular, the address length goes from 32 bits to 128 bits (i.e, 8 chunks of 16 bits). An IPv6 address can be written as 8 chucks of 2 bytes each separated by a semicolon.

```
2901:5e1d:44aa:6ddc:0000:0000:0000:0000
```

An IPv6 address is divided in two pats:

- 64 bits of **prefix** used to identify the network in which the device is and where the network is globally. This partition of the address is hierarchical because it defines an hierarchy of devices and networks.
- 64 bits of **interface id** (IID) used to identify the network interface (i.e., the device in the network). Usually the IID is the hash of the MAC address. Differently from IPv4, the same network interface card can be assigned multiple IPv6 addresses depending on the scope of the transmission.

Long story short, the prefix identifies a network and the IID identifies where the device is in the network.

Header

IP version 6 redefines the network header, too. More specifically, the header contains

- A **version** field. The version field specifies the version of IP used.
- **Traffic class** and **flow label** fields. This fields are used to specify some sort of Quality of Service.
- An **hop limit** field. The hop limit field, just like the Time To Live field in IPv4, specifies after how many hops a packet should be dropped.
- A **next header** field. The next header field is equivalent to the protocol field in IPv4 and specifies the protocol used in the payload of the packet.
- A **payload length** field.
- The **source** and **destination addresses**.

As we can see, an IPv6 header is much smaller and less complex than an IPv6 header, in fact we don't have some fields used in IPv4 (IHL, Identification, Flags, Fragment Offset, Header Checksum). In particular we can notice that, the fields used for fragmentation and header integrity verification are missing in IPv6.

12.2.2 Header compression

The network layer in 6LoWPAN tries to compress both the IPv6 and UPD headers to reduce the overhead of a message. The compression is **stateless**, which means that all packets are compressed independently one from the other, hence two packets of the same logical flow might be compressed in different ways.

To compress an IPv6 header for 6LoWPAN we can notice that,

- Common values for some header fields can be written in a more compact form. For instance, a field could take only a limited of values, hence we can enumerate them and use the enumeration instead of the actual value, effectively reducing the size of the field.
- 6LoWPAN uses only IPv6, hence the version field is useless, because it's value would always be 6.

- The traffic generated by 6LoWPAN in an IoT system is mostly uniform (i.e., all packets have the same requirements), hence the traffic-class and flow-label fields can be put to 0.
- The payload length can be derived from data-link layer because 6LowPAN uses only one transport protocol, hence we can remove it.
- The source and destination addresses can be compressed considering the scope of the communication.

Header compression header

To compress an IPv6 header we need to add a new header, called **header compression header**, to the classical IPv6 header. This might seem counter-intuitive, however the space saved by compression is much more than the length of the header compression header. The header compression header has a field for each field of the classical IPv6 header and each field defines how the respective field in the normal header is compressed. For instance, the Next Header field in the header compression header, defines how to compress the Next Header field in the IPv6 header to compress. Each field defines a dictionary of values, each of which represents a mode of compression of the respective field in the IPv6 header. This means that the header compression header has

- A **starting sequence** (i.e., `011`) to recognise the beginning of the header compression header.
- A **TF** field for Traffic Class and Flow Label compression. This field can take four values (2 bits)
  - 0: Carried Inline, namely ECN, DSCP and Flow-Label fields are explcitly written in the IPv6 header. In other words, no compression is applied.
  - 1: Only the ECN and Flow-Label fields are not compressed. The DSCP field is set to 0.
  - 2: Only the ECN and DSCP fields are not compressed. The Flow-Label field is set to 0.
  - 3: All fields are set to 0.
- **NH** for Next Header compression. This field can take two values (1 bit):
  - 0: Carried Inline (i.e., no compression is applied).
  - 1: The Next Header field is compressed.
- **HLIM** for Hop Limit compression. This field can take four values (2 bits)
  - 0: Carried Inline (i.e., no compression is applied).
  - 1: The field in the IPv6 packets contains the value 1.
  - 2: The field in the IPv6 packets contains the value 64.
  - 3: The field in the IPv6 packets contains the value 255.
- **CID** for Context Identifier Extension. This field can take 2 values (1 bit):
  - 0: No 1-byte CID identifier,
  - 1: 1-byte identifier follows
- **SAC and DAC** for Source and Destination Address Compression. This field can take 2 values (1 bits):

  - 0: Stateless
  - 1: Context-based

- **SAM and DAM** for defining the Source and Destination Address Mode. This field can take 4 values (2 bits):
  - 0: 16 bytes inline,
  - 1: 8 bytes inline,
  - 2: 2 bytes inline,
  - 3: elided
- **M** for defining the Multicast Destination. This field can take two values (1 bits):
  - 0: Destination is not multicast.
  - 1: Destination is multicast.

This header allows us to understand how the actual IPv6 header has been compressed so that we can rebuild the initial header.

Address compression

The fields in the header compression header used to compress the source and destination addresses are SAM, SAC, DAM, DAC and CID. In particular

- The SAM and DAM fields define how many bits we decide to compress (i.e., to eliminate), in the source and destination address, respectively. More precisely, if the SAM or DAM is
  - 0, no compression is applied.
  - 1, the prefix is elided (i.e., the address written in the address field of the IPv6 header is made of 64 bits and the 64 bits of the prefix are implicitly 0s).
  - 2, 112 prefix bits are elided (i.e., the address written in the address field of the IPv6 header is made of 16 bits and the 112 bits before are implicitly 0s).
- The SAC and DAC fields define the compression mode for the source and destination addresses, respectively. More precisely, if the SAC or DAC is
  - 0, the compression is done in **stateless mode**. In stateless mode, the device that is handling the packet should only look at the bits in the SAM or DAM field to understand how the addresses are compressed.
  - 1, the compression is done in **context-based mode**. In context-based mode, the device that is handling the packet should look into the CID field of the header compression header to understand how the header is compressed.

  The CID field is used to identify the compression mode in context-based compression. More precisely, if the CID is
  - 0, the device handling the packet doesn't have to include a ContextID in the source and destination addresses.
  - 1, the device handling the packet should include 4-bit a ContextID for the source and destination addresses.

Moreover, 6LoWPAN offers broadcast address compression.

UDP header compression

6LoWPAN doesn't only compress the IPv6 header, but also the UDP one (8 bytes). 6LoWPAN, as for IPv6, adds a header compression header with four fields, one for each field of the UDP header (source port, destination port, length and checksum). In particular,

- The source and destination port assume only a limited number of values in 6LoWPAN, from 61616 to 61632. Since we have 16 ports, we can reduce the size of the port field from 16 bits to 4 bits. Port compression is handled by the **P** (2 bits) field in the header compression header. In particular, if **P** is
  - 0, the full port number is written in the port field of the UDP header.
  - 1, the first 8 bits of the destination port are elided.
  - 2, the first 8 bits of the source port are elided.
  - 3, the first 12 bits of source and destination ports are elided.
- The length field can be completely deleted because it can be derived from the data-link layer.
- The checksum field can be elided if other integrity checks are applied. The checksum is controlled by the **C** field in the header compression header. In particular, if the field **C** is,
  - 0, the checksum field is present in the UDP packet.
  - 1, the checksum field is deleted from the UPD packet.

In a nutshell, an header compression header is made of

- The **starting sequence** `11110` (5 bits).
- The **C** field (1 bit) to control the checksum field.
- The **P** field (2 bits) to control the port fields.

12.2.3 Fragmentation

802.15.4 frames have a length limit of 127 bytes, hence a long IPv6 packet should be divided in multiple chucks. In the basic implementation, IPv6 doesn't support fragmentation, hence we have to reintroduce it adding new fields in the IPv6 header. In particular, we should add

- A **datagram tag** (16 bits). The datagram tag is an identifier that is common to all fragments that belong to the same original IPv6 packet. In other words, all the fragments in which an IPv6 packet has been divided have the same datagram id.
- A **datagram size** field (11 bits). The datagram size field stores the size of the fragment in bytes.
- A **datagram offset** field (8 bits). The datagram offset field is used to identify a segment in an IPv6 packet specifying the distance (in 8 bytes words) from the first byte of the IPv6 packet. The datagram offset is eliminated in the first fragment (it would be 0).

In practice, fragmentation has a big impact on a device's performance because fragmenting creates multiple IPv6 compressed packets, hence for the same amount of information (because the payload is the same, in total, for the original or fragmented packet) a device is generating more headers. Moreover, each fragment is a IPv6 header, which has to be handled by each device on the route.

For these reason it's good practice to avoid fragmenting at the network layer and moving this functionality at the application layer (e.g., block transfer mode for COAP). The same is true for IPv6 in classical Internet.

12.3 Routing

6LoWPAN can't use the classical IP routing protocol, hence we have to design a new one.

12.3.1 Types of routing protocols

Routing protocols can be divided in

- **Distance-vector algorithms**. In distance vector algorithms, a node tells its neighbour nodes all the destinations that it can reach together with the cost to reach them and the route used. Basically a device says to the neighbours that to reach a destination D it uses a path P with cost C.
- **Link-state algorithms**. In link-state algorithm, each device stores locally the state of the local links. These information is broadcasted to all the other devices of the network. This allows all devices to have the local link state information of the whole network.

Moreover, signaling can be

- **Proactive**. In proactive signaling, the routing information is acquired before it's needed.
- **Reactive**. In reactive signaling, the routing information is acquired when it's needed.

Another important thing to consider when talking about routing protocols is the cost metric used to evaluate the cost of a path. Some examples of metrics are

- The number of hops.
- Signal strength metrics.

Defining a routing protocols means deciding which algorithm, signalling type and metric to use.

12.3.2 Ripple

6LoWPAN uses a custom proactive distance-vector routing protocol called **Ripple**.

Chapter 13

RFID

13.1 Introduction

Radio Frequency IDentification (RFID) is an extension of the concept of barcodes brought to the frequency domain. In other words, RFID is used to identify an item using electromagnetic waves.

13.1.1 Building blocks

An RFID system is made of two building blocks

- A **tag**.
- A **reader**.

Tags

A tag is a piece of electronics attached to an item that sends a signal used to identify the item the tag is attached to. A tag is usually composed of

- Circuitry to scavenge energy. This means that most of RFID tags are passive (i.e., they don't need batteries and they get the power to work from the environment).
- A memory to store whatever information it needs to store (e.g., the identifier of the item or the manufacturer of the item).
- A control logic to handle the problem of collision.

A tag can be seen as the field part of the system. Tags can be divided in

- **Passive tags**. Passive tags don't have batteries, hence they have to scavenge energy without being active. Passive tags are the cheapest but their reading range is very limited (lower than a metre).
- **Semi-passive tags**. Passive tags are still stimulated by a reader but the operational power is provided by a battery. Semi-passive tags are slightly more complex and cost a bit more than passive tags, however they can operate in longer distances (around tens of metres).

- **Active tags**. Active tags are motes and field sensors in which the operational power is provided by a battery and a transmitter is built in the device.

The core challenges with tags, especially with passive ones, are related to energy and in particular how to scavenge enough energy to work.

Readers

A reader is responsible for reading RFID tags and the data stored on their memory. Practically, a reader is a computer with radio frequency capabilities to stimulate tags and collect data from the communication with the tag. A reader doesn't have battery issues since it's considered as a normal PC or something similar (e.g., a smartphone). The reader is usually connected, through other interfaces (e.g., WiFi, cable, 4G), to some back-end services that use the data collected from the tag. Readers can be

- **Fixed**. Fixed readers can't be moved and usually are directly connected to power.
- **Mobile**. Mobile readers can be moved and usually run on batteries.

The reader doesn't have many problems with respect to energy consumption, since it's a computer. The problem with readers is that, if it has to interact with passive tags, the radio frequency interface has to provide enough power for the tag to respond back. Moreover, since the tags are passive, they can't send powerful signals, hence the reader has to be able to discriminate low-power signals coming back from the tags.

13.1.2 Performance measures

When measuring the performance of RFID systems, we should consider

- **Reading range**. The reading range measures from how far a reader can read a tag (i.e., how far the tag can be for the reader to read its data).
- **Throughput**. The throughput measures how many tags per second a reader can read.
- **Robustness**. Robustness measures how robust is the system with respect to the external environment (i.e., how the reader can cope with the environment in which it has to operate).

That being said, the real performance of a system depends on

- The **carrier frequency**.
- The **emitted power** (by the reader and the tag).
- The **environment** (e.g., water).
- **Concurrency** (i.e., multiple tags in the reading field of the reader).

13.1.3 Standardisation

RFID is standardised by the ISO that defined a standard for each part of an RFID system. In particular, the reference standard for the tags is called **GS1 – EPCGlobal initiative, UHF C1 Gen 2** and defines

- How tags should communicate.
- How the tags' identifier, called EPC address (96 bits), should be. The EPC address has an hierarchical structure.

13.2 Passive tags

Passive tags can be be further classified depending on their storage capabilities. In particular, we can distinguish between

- **One bit tags**. One bit tags don't have storage capabilities. These tags are very simple and are composed of some magnetic material (e.g., a folded coil) and when the magnetic material crosses the reader range, the magnetic field of the reader changes, hence the reader senses a variation in its current, so it understand that a tag is close to it. The reader can't detect what tag has approached it, but it can only say that a tag got close to it because no identifier is sent to the reader. This also means that, multiple detection is impossible (i.e., readers can't tell if multiple tags are in the reading range).
- **Tags with storage space**. Tags with storage space have some PROM or ePROM memory inside them, which can be read-only or read-write. These tags are more complex and are based on the exchange of electromagnetic waves (not only on magnetic fields like for one bit tags). Moreover, they also have a circuitry responsible for sending radio-frequency waves to the reader with the information it requires. A classification of tags with storage space, depending on their capabilities is shown in Table 13.1.

Class 0	UHFI Read only, programmed passive tag
Class 1	UHF or HF; write once read many
Class 2	Passive read-write tags that can be written at any point
Class 3	Read-write tags with onboard sensors capable of recording various parameters; semi-passive or active
Class 4	Read-write tags with integrated transmitters to communicate with other tags and readers
Class 5	Class 4 with the capability of passing power to other tags and communicate with devices different from readers.

Table 13.1: Classes RFID tags with memory can be divided into.

13.3 Communication

13.3.1 Carriers

RFID systems can use different radio frequency fields:

- The Low Frequency (LF) field. This field is used for tags attached to animals.
- The High Frequency (HF) field. This field is used for Near Field Communication (NFC) systems.
- The Very High Frequency (VHF) field.
- The Ultra High Frequency (UHF) field, which goes from 400 MHz to 2 GHz. This field is the most used.
- The Super High Frequency (SHF) field.

Depending on the frequency we use, we have different ways to provide power to the tags. In particular:

- From LF to HF, we use **magnetic coupling**.
- From VHF to SHF, we use **electromagnetic coupling**.

Magnetic coupling

In magnetic coupling, also called inductive coupling, the physical principle used in called Near Field Model NFM. The carrier frequencies in which magnetic coupling has the best performance is from few hundreds of kHz (125 kHz) to tens of MHz (13.56 MHz), i.e., in the LF and HF fields.

A reader, which can be modelled as a coil, generates a magnetic field. Any other object (still modelled as a coil) with magnetic capabilities in its range experiences the magnetic field and a current generates, thanks to induction, in the coil. The current generated is used to power up the tag, its logic and the circuit used to send data to the reader. Tag-wise, there exist two different operation modes:

- **Sequential**, in which the tag first accumulates energy and then transmits back data.
- **Duplex** (most used), in which the tag sends data while receiving power from the reader.

When the tag wants to send some data to the reader, it has to change its resistive load, so that the reader feels a change in it's voltage and the reader can get the data from the voltage variation.

The tag needs enough power to power up. The power delivered depends on

- The size of the coils (i.e., the antennas) of the tag and the reader.
- The orientation of the tag and the reader one with respect to the other.
- The hardware of the tag and the reader.

For this reasons, the reading range in magnetic coupled systems is usually limited and the performance depends a lot on orientation. However, the coupling is almost insensitive to organic interference (e.g., water is not an issue) and the cost is very low.

Electromagnetic coupling

In electromagnetic coupling, the physical principle used in called Far Field Model FFM. In electromagnetic coupling, the energy is not only scavenged using magnetic field but also the electrical component generated by the field. The principle of transmission is back-scattering (as for radars). More precisely, an electromagnetic wave, emitted by the reader hits a tag which back-scatters the wave properly modulated, depending on the data it wants to send. Using the response wave, the reader can read the data sent by the tag. The response of the tag is obtained changing the impedance of the tag's circuitry, in particular

- With an high impedance, the tag generates low back-scattering (i.e., most waves sent by reader are absorbed).
- With a low impedance, the tag generates high back-scattering (i.e., most waves sent by reader are reflected).

The impedance variation can be used to send a 0 (high impedance) or a 1 (low impedance). Schematically,

1. The reader sends an electromagnetic wave to the tag.
2. The tag powers up.
3. The tag changes its impedance so that a different wave is back scattered. The easiest way to modulate the back-scattered wave (i.e. to encode some information in it) is to absorb most of the sent wave to send back a 0 or to back-scatter most of it to send a 1 (other encoding with multi-bit symbols are possible).
4. The reader receives the wave back and demodulates it (i.e., if it has received the signal it send, it understands that the tag sent a 1, otherwise it knows the tag sent a 0).

The performance of electromagnetic coupling depends on

- The power of the reader's antenna (the higher, the furthest a tag can be from a reader).
- The carrier (the lower, the furthest a tag can be from the reader).

With respect to magnetic coupling, electromagnetic coupling,

- Has higher reading range.
- Has higher bit rate (because it depends on the carrier frequency).
- Is influenced more by the environment.

The fact that the bit rate increases with the frequency, but the reading range decreases, means that the optimal carrier frequency is not at very high frequencies but from 100 MHz to 1 GHz (lower frequencies would offer low bit rates while higher frequencies would make the reading range too narrow).

13.3.2 Tags identification

The reader-tag interaction is based on an interrogation model, namely the reader asks a tag some data, which is sent back by the tag. If multiple tags are in the reading range of the reader, then all of them will answer and the reader should be able to disambiguate the requests. This means that we need some protocols that allow either the tags not to reply at the same time or the reader to handle collisions. Notice that, this problem is an access control problem (i.e., multiple resources who share a common communication medium) with some peculiarities

- The tag population is unknown, but we know that it's a fixed number.
- We can't used complex protocols because the tags are off, hence they can't sense the channel before sending the response.
- The communication process is controlled and driven by the reader, hence it's centralised.

Conflict handling protocols can be divided into

- **Dynamic Frame ALOHA** (i.e., a variation of slotted ALOHA).
- **Binary trees**.

The performance measure used to understand if a protocol is doing a good job is called arbitration efficiency η and considers how much time the reader needs to read all tags that need to be read. Formally, the arbitration efficiency is the ratio between the population size N and the arbitration period $L(N)$ (or the number of slots, if the time is slotted)

$$\eta = \frac{N}{L(N)}$$

The ideal case is $\eta = 1$ because it means that the reader could read N tags in T time slots.

Dynamic Frame ALOHA

Frame ALOHA is a version of ALOHA in which time is slotted and framed. Namely, time is divided in frames that are in turns divided in slots. The protocol works as follows:

1. At the beginning of a frame, all tags activated by the reader choose randomly one of the slots of the frame.
2. Each tag sends its id in the slot it has chosen.
3. If a tag collides (i.e. more tags chose the same slot), they have to repeat the procedure in the following frame (choosing a new random slot). Notice that, the tags that succeeded don't have to transmit their id in the following frames.

Performance To evaluate the efficiency η_{ALOHA}, let us consider a single frame. The average throughput $\mathbb{E}[S]$ of that frame is

$$\mathbb{E}[S] = n \cdot \left(1 - \frac{1}{r}\right)^{n-1}$$

where

- n is the number of tags competing.
- r is the number of slots.

The average throughput can be computed from the probability $P_{slot\ succ}$ that a slot is successful, i.e., that only one tags chooses that slot. $P_{slot\ succ}$ is the probability that a tag chooses a specific slot (with probability $\frac{1}{r}$) and that no other tag chooses that same slot (with probability $\left(1 - \frac{1}{r}\right)^{n-1}$)

$$P_{slot\ succ} = \frac{1}{r} \cdot \left(1 - \frac{1}{r}\right)^{n-1}$$

Since any station can choose among r slots, we can multiply $P_{slot\ succ}$ by r and since we have n station, we have to multiply $P_{slot\ succ}$ also by n to obtain

$$\begin{aligned} n \cdot r \cdot P_{slot\ succ} &= n \cdot r \cdot \frac{1}{r} \cdot \left(1 - \frac{1}{r}\right)^{n-1} \\ &= n \cdot \left(1 - \frac{1}{r}\right)^{n-1} \\ &= \mathbb{E}[S] \end{aligned}$$

Since we are considering only a frame, the number of time slots is r and the number of tags that successfully sent their id is the average throughput $\mathbb{E}[S]$, hence

$$\eta_{ALOHA,1\ frame} = \frac{\mathbb{E}[S]}{r} = \frac{n}{r} \cdot \left(1 - \frac{1}{r}\right)^{n-1}$$

If we maximise $\eta_{ALOHA,1\ frame}$ as a function of r, we obtain that it's maximised if the number of slots in a frame is equal to the number of tags.

$$\begin{aligned}\eta_{ALOHA,max} &= \eta_{ALOHA,1\ frame}(r = n)\\ &= \frac{n}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1}\\ &= \left(1 - \frac{1}{n}\right)^{n-1}\end{aligned}$$

Frame ALOHA considers however multiple frames. To consider multiple frames we have to write a recursive equation for the number of slots $L(N)$ needed to get a reply from all N tags. In particular, the number of slots $L(N)$ is

$$L(N) = r + \sum_{i=0}^{n-1} P(S = i) \cdot L(N - i) = \frac{r + \sum_{i=1}^{n-1} P(S = i) \cdot L(N - i)}{1 - P(S = i)}$$

where

- r is the number of slots in a frame.
- $P(S = i)$ is the probability that the throughput is equal to i, namely that i tags could successfully send their id.

This means that, to compute $L(N)$, we only have to recursively resolve the equation and find the probabilities $P(S = i)$.

Schoute's estimate for setting frame size As we have seen, the efficiency is maximum when the frame size is equal to the number of concurrent tags. This means that, if some tags successfully send their id, the number of concurrent tags in the following frames reduces, hence we should also reduce the number of slots in the frame. To solve this problem, each tag has

- A **backlog estimation module** that estimates the number of tags n_{est} that still have to send their id.
- A **collision resolution module** that runs frame ALOHA with $r = n_{est}$.

The easiest way to compute n_{est} is the Schoute's estimate heuristics. The Schoute's estimate says that a tag should set n_{est} only consider the number c of slots that collided in the previous frame (which is known by the tag) and using the following formula

$$n_{est} = \text{round}(H \cdot c)$$

where

- H is the average number of colliding tags in one colliding slot and is computed as

$$H = \frac{1 - e^{-1}}{1 - 2e^{-1}} = 2.39$$

- c is the number of slots that collided in the previous frame.
- round() is a function that rounds up the result of $H \cdot c$ to the closest integer.

Let us take a moment to understand where does the number H come form. Before starting we have to assume that the frame size set by the estimate n_{est} is so that tags competing are distributed as a Poisson of $\lambda = 1\ \frac{\text{tag}}{\text{s}}$. This means that on average there is one tag per slot. Given this assumption, we want to compute the average number of tags colliding H, given that a slot is colliding.

$$H = \mathbb{E}\big[N|N \geq 2\big]$$

where N is the number of tags colliding in a slot. Basically, are computing the number of tags N colliding in a slot, given that the number of tags colliding is more than 1 (i.e., we are actually colliding). If we apply the definition of expected value, we obtain

$$\begin{aligned} H &= \mathbb{E}\big[N|N \geq 2\big] \\ &= \sum_{k=0} p(N = k|N \geq 2) \cdot N \end{aligned}$$

Now we can rewrite the probability $p(N|N \geq 2)$ using the Bayes rules.

$$p(N|N \geq 2) = \frac{p(N = k) \cdot p(N \geq 2|N = k)}{p(N \geq 2)}$$

Let's develop each term, starting from $p(N = k)$. $p(N = k)$ is the probability that a k tags collide in a slot and, since tag arrival is distributed as a Poisson of $\lambda = 1$ (i.e., on average one process chooses a certain tag), we obtain

$$p(N = k) = -\frac{1^k e^{-1}}{k!} \quad \forall k = 0, \ldots, k$$

Let us now focus on the probability $p(N \geq 2)$ that at least two tags collide in a slot. This probability can be seen as 1 minus the probability that one tag collides in a slot (i.e., that only one tag chooses that slot, hence no collision happens) or that no tags collide in the slot.

$$\begin{aligned} p(N \geq 2) &= 1 - P(N = 0) - P(N = 1) \\ &= 1 + \frac{1^0 e^{-1}}{0!} + \frac{1^1 e^{-1}}{1!} \\ &= 1 - e^{-1} - e^{-1} \\ &= 1 - 2e^{-1} \end{aligned}$$

Finally let us consider the probability that at least two tags collided if the number of tags in a slot is k. In this case, if k is greater than 1 then surely two tags collided, hence the probability is 1, otherwise, no tags can have collided, hence the probability is 0.

$$p(N \geq 2|N = k) = \begin{cases} 1 & \text{if } k \geq 2 \\ 0 & \text{otherwise} \end{cases}$$

Putting all this values together we can compute the H as

$$\begin{aligned}
H &= \sum_{k=0}^{1} \left(\frac{\frac{1^k e^{-1} \cdot 0}{k!}}{1-2e^{-1}} + \sum_{k=2}^{\infty} \frac{\frac{1^k e^{-1} \cdot 1}{k!}}{1-2e^{-1}} \right) \cdot k \\
&= \sum_{k=2}^{\infty} \frac{\frac{1^k e^{-1}}{k!} \cdot 1}{1-2e^{-1}} \cdot k \\
&= \frac{1}{1-2e^{-1}} \sum_{k=2}^{\infty} \frac{1^k e^{-1}}{k!} \cdot k
\end{aligned}$$

Now we can notice that what the summation is the expected value of a Poisson process of parameter $\lambda = 1$ without the terms for $k = 0$ and $k = 1$. This means that we can write this term as the mean of a Poisson process of $\lambda = 1$, which is 1, minus the terms for $k = 0$ and $k = 1$.

$$\begin{aligned}
H &= \frac{1}{1-2e^{-1}} \sum_{k=2}^{\infty} \frac{1^k e^{-1}}{k!} \cdot k \\
&= \frac{1}{1-2e^{-1}} \left(1 - \frac{1^k e^{-1}}{k!} \cdot k - \frac{1^k e^{-1}}{k!} \cdot k \right) \\
&= \frac{1}{1-2e^{-1}} \left(1 - \frac{1^0 e^{-1}}{0!} \cdot 0 - \frac{1^1 e^{-1}}{1!} \cdot 1 \right) \\
&= \frac{1}{1-2e^{-1}} \left(1 - e^{-1} \right) \\
&= \frac{1}{1-2e^{-1}} - \frac{e^{-1}}{1-2e^{-1}} \\
&= \frac{1-e^{-1}}{1-2e^{-1}}
\end{aligned}$$

Finally we have obtained the value of H.

Appendix A

Poisson distribution

The Poisson distribution is a discrete probability distribution that expresses the probability of a given number of events k occurring in a fixed interval of time T or space if on average λ events occur in T or these events occur with a known constant mean rate r and independently of the time since the last event.

$$Y \sim Pois(\lambda)$$

Basically, λ is the number of events occurring in a time interval T.

$$\lambda = r \cdot T$$

Mean The mean of a Poisson distribution is λ.

$$\mathbb{E}[Y] = \lambda \tag{A.1}$$

Variance The variance of a Poisson distribution is λ.

$$Var[Y] = \lambda \tag{A.2}$$

Probability mass function The probability mass function $P(k)$ (i.e., the probability of having k arrivals in a time interval t) is

$$P(k) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{A.3}$$

This probability can also be written as

$$P(k) = \frac{(rt)^k e^{-rt}}{k!} \tag{A.4}$$

where

- r is the rate at which event happen.
- t is the time interval of interest.

Definitions